

Department of Computer Science and Engineering

The Chinese University of Hong Kong

CJC2401 Final Year Project Report

Real-Time Posture Correction for Yoga Practice

Hui Wang Chi 1155159410 (Submitted by)

Yuen Ho 1155176855

Contents

| | |
|---|-----------|
| 1. Abstract..... | 6 |
| 2. Introduction | 6 |
| 2.1. Project Goal..... | 6 |
| 2.2. Significance of the Project | 8 |
| 2.3. Problem Statement | 8 |
| 2.4. Proposed Solutions | 9 |
| 2.5. Project Timeline | 10 |
| 3. Related Works..... | 10 |
| 3.1. Existing Market Products | 11 |
| 3.2. Existing Comparison of Human Pose Estimation Models..... | 11 |
| 3.2.1. <i>Brief Introduction to Existing Papers</i> | 12 |
| 3.2.2. <i>Shortcomings of the Current Papers</i> | 12 |
| 3.3. Existing Research on Posture Correction | 13 |
| 3.3.1. <i>Overview of Existing Research</i> | 13 |
| 3.3.2. <i>Heuristics in Existing Papers</i> | 14 |
| 3.3.3. <i>Shortcomings in Existing Papers</i> | 14 |
| 4. Benchmark on Existing Pose Detection Models..... | 16 |
| 4.1. Detailed Approach | 16 |
| 4.1.1. <i>Dataset</i> | 16 |
| 4.1.2. <i>Evaluation of Model Latency</i> | 17 |
| 4.1.3. <i>Evaluation of Accuracy</i> | 17 |
| 4.1.4. <i>Evaluation of Memory Usage</i> | 19 |
| 4.2. Code Implementation | 20 |
| 4.3. Benchmark Results | 21 |
| 4.3.1. <i>Result Table</i> | 21 |
| 4.3.2. <i>Result Samples</i> | 22 |
| 4.4. Model Results Analysis..... | 22 |
| 4.4.1. <i>OpenPose</i> | 22 |
| 4.4.2. <i>PoseNet</i> | 23 |
| 4.4.3. <i>MoveNet</i> | 24 |
| 4.4.4. <i>BlazePose</i> | 24 |
| 4.5. The Chosen Model | 25 |
| 4.5.1. <i>Accuracy</i> | 25 |
| 4.5.2. <i>Latency and Memory Usage</i> | 25 |
| 4.5.3. <i>Additional Benefits</i> | 26 |
| 5. Dataset Preparation | 26 |
| 5.1. First Phase Dataset Preparation | 26 |
| 5.1.1. <i>Observations</i> | 26 |

| | | |
|-----------|---|-----------|
| 5.1.2. | <i>Dataset Optimization</i> | 28 |
| 5.1.2.1. | Enhancement of Dataset Size | 28 |
| 5.1.2.2. | Enhancement of Dataset Correctness..... | 28 |
| 5.2. | Second Phase Dataset Preparation | 29 |
| 5.2.1. | <i>Observations</i> | 29 |
| 5.2.2. | <i>Dataset Construction</i> | 30 |
| 5.2.3. | <i>Characteristics of the Chosen Poses</i> | 30 |
| 5.2.4. | <i>Classification Process</i> | 36 |
| 5.2.5. | <i>Exceptional Cases</i> | 36 |
| 5.2.6. | <i>Ambiguity of the Dataset</i> | 37 |
| 5.3. | Dataset Size | 39 |
| 6. | Model Development | 39 |
| 6.1. | Training Data | 40 |
| 6.1.1. | <i>Raw Dataset</i> | 40 |
| 6.1.2. | <i>Data Splitting</i> | 40 |
| 6.1.3. | <i>Data Balancing</i> | 40 |
| 6.1.4. | <i>Data Augmentation</i> | 40 |
| 6.2. | Input Structure..... | 41 |
| 6.2.1. | <i>Output of the First-Stage Model</i> | 41 |
| 6.2.2. | <i>Normalization of x and y</i> | 41 |
| 6.2.3. | <i>Normalization of z</i> | 42 |
| 6.2.4. | <i>Other Preprocessing</i> | 43 |
| 6.3. | Output Structure | 44 |
| 6.4. | Model Architecture Selection | 44 |
| 6.4.1. | <i>Model Type Selection</i> | 44 |
| 6.4.2. | <i>Shared MLP</i> | 45 |
| 6.4.2.1. | Rationale for Shared MLP | 45 |
| 6.4.2.2. | Technical Explanation..... | 45 |
| 6.5. | Training Configurations..... | 46 |
| 6.5.1. | <i>Batch Size</i> | 46 |
| 6.5.2. | <i>Loss Function</i> | 47 |
| 6.5.3. | <i>Optimizer</i> | 47 |
| 6.5.4. | <i>Scheduler</i> | 48 |
| 6.6. | Development Comparisons | 48 |
| 6.6.1. | <i>MLP vs. Shared MLP</i> | 48 |
| 6.6.2. | <i>Input Masking vs. No Masking</i> | 49 |
| 6.7. | Final Results | 50 |
| 6.7.1. | <i>Model Architecture and Hyperparameters</i> | 50 |
| 6.7.2. | <i>Training Log</i> | 51 |
| 6.7.3. | <i>Evaluation</i> | 51 |
| 6.7.3.1. | Loss & Accuracy | 51 |
| 6.7.3.2. | Confusion Matrices | 52 |
| 6.8. | Flexibility for Future Extension | 54 |
| 7. | Model Integration | 54 |
| 7.1. | Metrics | 55 |

| | | |
|------------|--|-----------|
| 7.2. | Pre-Evaluation | 55 |
| 7.3. | Version 1 – Naïve Sequential Execution..... | 56 |
| 7.4. | Version 2 – Multithreading and Frame Dropping..... | 57 |
| 7.4.1. | <i>Multithreading in Python</i> | 59 |
| 7.4.2. | <i>Why Only Two Threads?</i> | 59 |
| 7.4.3. | <i>Version 2 Performance Results</i> | 60 |
| 7.5. | About Downsampling Technique..... | 60 |
| 7.6. | Post-Integration System Performance | 60 |
| 8. | Result Stabilization | 61 |
| 8.1. | Current Accuracy | 61 |
| 8.2. | Current Issues | 61 |
| 8.2.1. | <i>Accuracy Is Not High Enough</i> | 61 |
| 8.2.2. | <i>Unstable Predictions</i> | 61 |
| 8.3. | Solution: Majority Voting..... | 62 |
| 8.4. | Performance Improvement | 62 |
| 9. | User Interface | 63 |
| 9.1. | Frames Per Second (FPS)..... | 64 |
| 9.2. | Bottom Output Panel..... | 64 |
| 9.3. | Example Screenshots..... | 67 |
| 10. | Practical Evaluation | 68 |
| 10.1. | Evaluation Methodology | 68 |
| 10.2. | Evaluation Results | 69 |
| 10.3. | Discussion | 69 |
| 11. | Limitations and Challenges | 70 |
| 11.1. | Limited Dataset Size and Coverage | 70 |
| 11.2. | Absence of Experienced Yoga Guidance | 70 |
| 12. | Conclusion | 70 |
| 13. | Side Research: Language Model-Based Approach..... | 72 |
| 13.1. | Related Research..... | 73 |
| 13.1.1. | <i>Vision-Language Models</i> | 73 |
| 13.1.2. | <i>Prompting Strategies in VLMs</i> | 73 |
| 13.2. | Deployment..... | 74 |
| 13.2.1. | <i>Hardware Environment</i> | 74 |
| 13.2.2. | <i>Software Environment</i> | 74 |
| 13.3. | Qwen2.5-VL-3B..... | 75 |
| 13.3.1. | <i>Resource Requirements & Feasibility</i> | 75 |
| 13.4. | DeepSeek-VL-1.3B | 75 |
| 13.4.1. | <i>Deployment & Performance</i> | 75 |

| | |
|--|-----------|
| 13.4.2. <i>Evaluating DeepSeek-VL-1.3B</i> | 76 |
| 13.4.3. <i>Conclusion on DeepSeek-VL-1.3B</i> | 78 |
| 13.5. Llama-3.2-11B-Vision..... | 78 |
| 13.5.1. <i>Deployment & Performance</i> | 79 |
| 13.5.2. <i>Evaluating Llama-3.2-11B-Vision</i> | 79 |
| 13.5.3. <i>Time Performance</i> | 80 |
| 13.5.4. <i>Result Analysis</i> | 82 |
| 13.6. Limitations | 85 |
| 13.6.1. <i>Prompting Methodology</i> | 85 |
| 13.6.2. <i>Prompt Format and Content</i> | 85 |
| 13.7. Conclusion | 86 |
| 13.7.1. <i>Accuracy Considerations</i> | 86 |
| 13.7.2. <i>Processing Time Constraints</i> | 86 |
| 13.7.3. <i>Potential Advantages</i> | 86 |
| 13.7.4. <i>Future Prospects</i> | 86 |
| 14. Individual Contributions | 88 |
| 14.1. My Contributions | 88 |
| 14.2. Work Allocation Table..... | 89 |
| 15. Appendices..... | 90 |
| 15.1. Responses to Questions from the Supervisor and Grader (Semester 1 Presentation) | 90 |
| 15.2. LLM Prompts | 90 |
| 16. References | 92 |

1. Abstract

This project aims to develop a real-time yoga posture correction system, addressing the lack of existing applications that provide instant feedback for solo practitioners, thus reducing injury risk. The system employs a two-stage approach: first, it uses BlazePose, a pre-trained pose estimation model, to extract body keypoints from live video streams; second, a custom-built Shared MLP deep learning model classifies four yoga postures (Down Dog, Plank, Side Plank, Warrior II) and evaluates their correctness.

To support training, we compiled a dataset of over 3,800 labeled yoga images from online sources, including both correct and incorrect posture examples. The resulting model achieved approximately 93% accuracy on test data. Integrated into a complete system with a simple graphical interface, it provides smooth real-time feedback at around 30 FPS. Practical evaluations further demonstrated the system's near-perfect stability, enhanced using majority voting.

Additionally, we explored a cutting-edge Vision-Language Model (VLM)-based approach, concluding that current VLM solutions, while promising in eliminating extensive dataset preparation, remain impractical due to significant computational overhead and limited accuracy in fine-grained classification tasks.

2. Introduction

2.1. Project Goal

Objective

The goal of this project is to develop a real-time posture correction system for yoga practice. It is designed for individual users, and all processing is performed locally on a modern laptop, without external cloud computing resources or APIs.

The project is research-oriented, focusing on the development of a working prototype that demonstrates the core functionality. A simple user interface is included, but the goal is not to create a fully standalone application.

Development Breakdown

The development is organized into five core objectives:

1. Model Benchmarking: Evaluate existing pre-trained human pose detection models to find the best-performing one for our system (see [Section 4](#)).

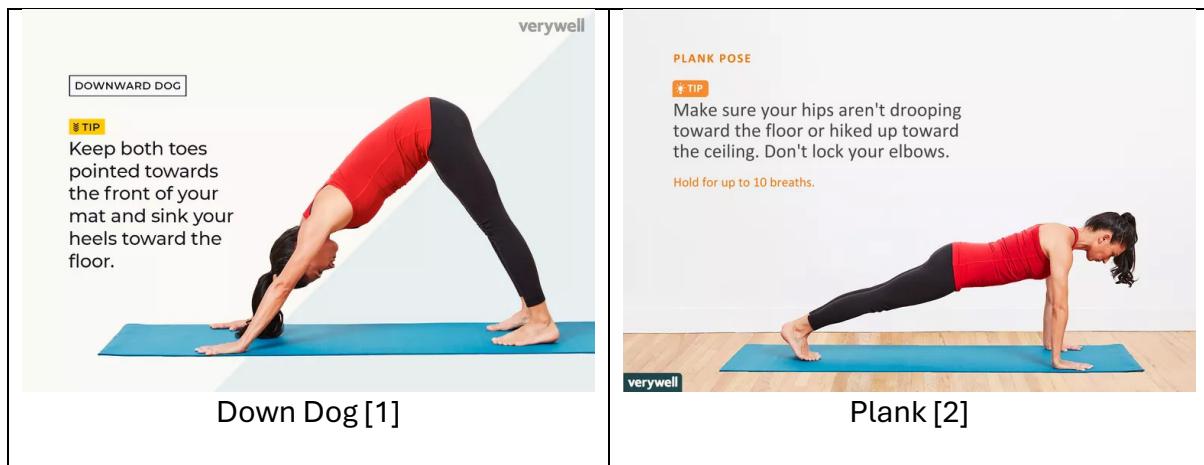
2. Dataset Construction: Compile our own dataset of yoga posture images specifically tailored for this project (see [Section 5](#)).
3. Model Development: Develop a model to recognize the posture type and estimate the correctness of the pose in a given image (see [Section 6](#)).
4. Real-Time Processing: Enable the system to process a continuous image stream, with optimizations for real-time performance (see [Section 7](#)).
5. Feedback Playback: Implement a simple user interface that visualizes the image stream and provides live feedback to the user (see [Section 9](#)).

Scope and Outcome

The final system classifies live camera input into one of four supported yoga postures: Down Dog, Plank, Side Plank, and Warrior II (see [Table 1](#) for examples), along with a binary correctness prediction (correct/incorrect). The results are rendered as a visual playback on the screen in real time.

In the first semester, we initially planned to expand the system to support more postures or finer-grained correctness classification. However, we later shift our focus to an alternative research path that pursues the same goal using a different implementation (see [Section 13](#)). Despite this shift, the system is still designed to be extensible, allowing for easy further enhancements (see [Section 6.8](#)).

As a by-product of the system development, we also created a custom yoga posture dataset containing samples of both correctly and incorrectly performed versions of each of the four supported poses.



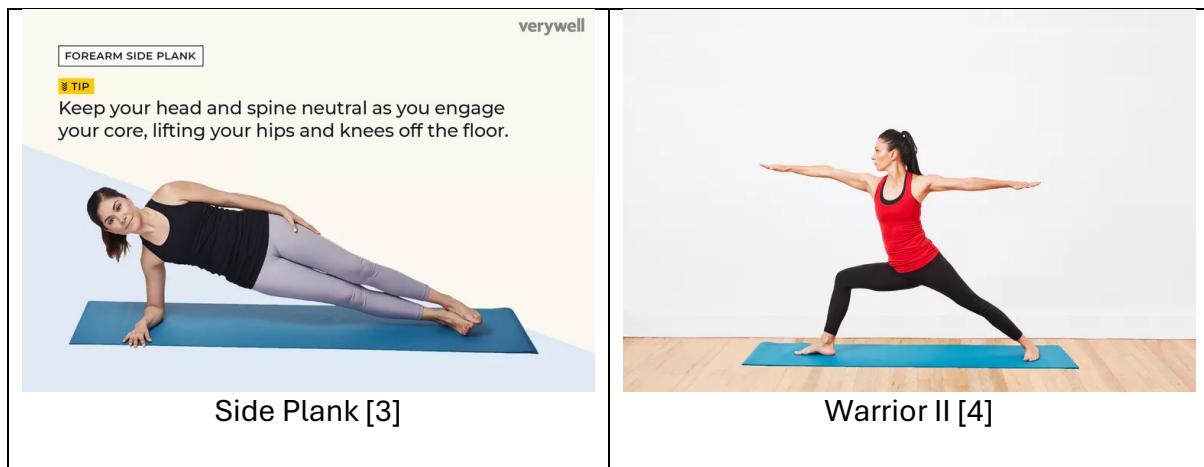


Table 1. Standard representations of the Down Dog, Plank, Side Plank, and Warrior II yoga postures, sourced from the authoritative health and wellness website Verywell Fit.

2.2. Significance of the Project

Proper posture is critical in yoga workouts. Many people may assume that yoga is less likely to cause injuries because its poses are often static and soothing compared to more dynamic sports like football or basketball. However, research [5] shows that improper yoga postures can still lead to pain and injury. In fact, the risk of injury from yoga is comparable to other forms of exercise.

For individuals practicing yoga without a coach, identifying and correcting improper postures can be challenging. Our project aims to develop a posture correction model to assist solo practitioners, helping them train safely and effectively.

2.3. Problem Statement

First, there is a lack of mature yoga posture correction applications on the market that offer real-time correction (see [Section 3.1](#)), which is the primary focus of our project.

Second, while we are using an existing pre-trained human pose detection model, the available research comparing these models does not provide enough information to guide us in selecting the best one for our needs (see [Section 3.2](#)).

Third, no existing dataset fully meets the requirements of our project, with key challenges including the lack of incorrect posture samples and insufficient data for certain postures (see [Section 5.2.1](#)).

Fourth, although some research papers have developed systems similar to ours, their solutions have notable deficiencies that we aim to address (see [Section 3.3](#)).

2.4. Proposed Solutions

To address the aforementioned problems, we started by creating our own dataset of yoga posture images. This dataset is built by combining existing datasets and sourcing additional videos from YouTube to increase the quantity and variety of training data (see [Section 5](#)).

Standard Two-Stage Approach

For the real-time posture correction system, we adopted a two-stage approach, as shown in [Figure 1](#):

1. First Stage: We use a pre-trained human pose detection model to extract the user's body keypoints. To select the best model, we conducted a comprehensive benchmark of mainstream models (see [Section 4](#)), addressing the shortcomings of existing research that provided insufficient information for model selection.
2. Second Stage: We developed and trained deep learning classification model based on "Shared MLP" (see [Section 6](#)). This model takes the keypoints extracted in the first stage as input and outputs the estimated posture type and posture correctness (or can alternatively be extended to posture mistake categories).



Figure 1. The overall two-stage approach pipeline of this project.

Alternative Approach Using VLM

In the second semester, we also explored a side research direction using a Vision Language Model (VLM) as an alternative to our standard two-stage approach (see [Section 13](#)). This investigation served as a direct comparison, motivated by the recent advancements in VLMs and their potential for generalization across image-based tasks, ease of use through prompt engineering, and the elimination of the need for dataset preparation.

In this alternative approach, the VLM replaces both stages of our system by directly estimating the posture type and correctness from a given input image. Although it does not provide body keypoints as our first stage model does, this is not a limitation in our

case, since the final goal is to assess the posture type and correctness rather than to extract keypoints themselves.

2.5. Project Timeline

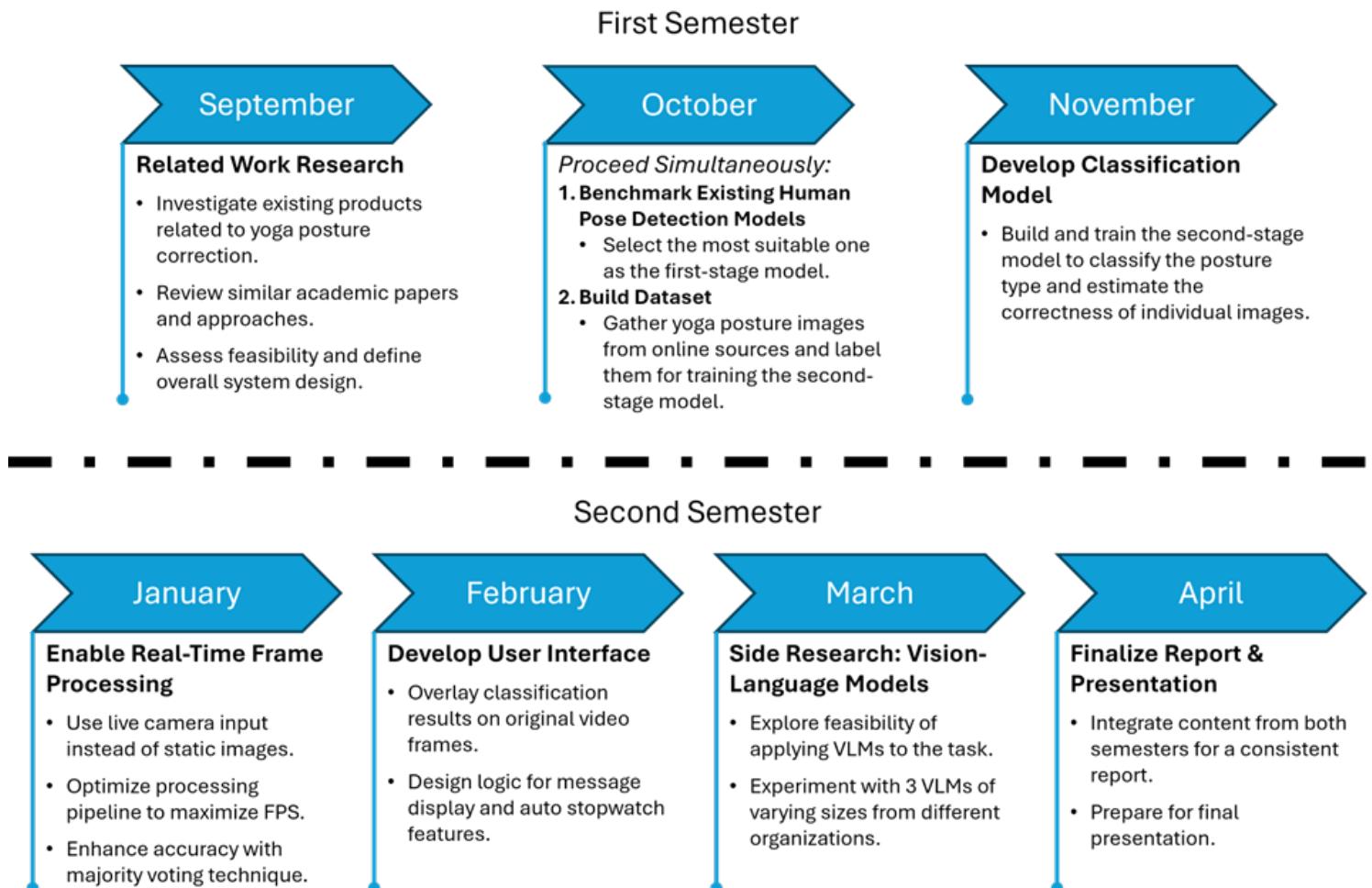


Figure 2. Project roadmap outlining key milestones from September 2024 to April 2025.

In the following sections, Sections 3 to 6 were completed during the first semester and have undergone only minor modifications compared to the first semester's report. Sections 7 and beyond were completed in the second semester.

3. Related Works

(Completed in the first semester)

At the beginning of our project, we conducted research on works related to our topic. This includes exploring existing market products for pose correction in yoga, reviewing

papers comparing pre-trained human pose estimation models, and analyzing prior research with similar objectives to our project. A detailed discussion of these findings is provided below.

3.1. Existing Market Products

Regarding pose correction in yoga, it is essential to ascertain whether any existing market products offer similar functionalities to what we aim to achieve, so that we can draw insights from the approaches used by current developers.

Considering this, we have identified several leading market products that deliver yoga lessons to individual users, including Peloton [6], Yoga Studio App [7], Alo Moves [8], DoYouYoga [9], Gaia [10], and Glo [11]. However, none of these products provide the service or feature for assessing and correcting users' poses.

For example, Peloton and Alo Moves provide pre-recorded lessons to users. The video lessons feature one instructor introducing the poses and demonstrating the correct postures. Although they offer guidance to help users avoid potential injuries, it is difficult for users to assess their posture correctness during exercises without real-time adjustments.

Meanwhile, some products offer live-streaming lessons instead of pre-recorded yoga lessons, such as Glo. These products and platforms provide real-time correction through coaching. However, this feature does not align with our project goal, because our focus is to give pose correction for the solo trainers who practice yoga without the guidance of a coach.

In conclusion, there are no existing products that align with the objectives of our product in the realm of yoga practices. Further research will be undertaken in the upcoming semester to explore any products, if they exist, that are focused on correcting gym postures.

3.2. Existing Comparison of Human Pose Estimation Models

We have reviewed two existing research papers that focus on comparing mainstream human pose estimation models: "Comparative Analysis of OpenPose, PoseNet, and MoveNet Models for Pose Estimation in Mobile Devices" by Smith et al. [12] and "Body Posture Detection and Comparison between OpenPose, MoveNet, and PoseNet" by Johnson and Lee [13]. Our goal in reviewing these papers was to determine which model would be best suited for use in the first stage of our solution system (see [Section](#)

[2.4](#)). However, this effort was unsuccessful due to certain limitations in these papers, which are also discussed in the following sections.

3.2.1. Brief Introduction to Existing Papers

In the first paper, the study compares OpenPose, PoseNet, and MoveNet in terms of the time taken by each model to estimate 1,000 images and their respective accuracy. Images from the Common Objects in Context (COCO) dataset and the Max Planck Institute for Informatics (MPII) Human Pose dataset are used for the evaluation. The study concludes that while MoveNet is the fastest model, PoseNet achieves the highest average accuracy of 97.6%, while also being the second fastest. Although OpenPose ranks second in accuracy, its processing time is approximately 12 times slower than MoveNet.

In the second paper, the study also compares OpenPose, PoseNet, and MoveNet. Over 1,000 images from the COCO dataset were used to evaluate the accuracy and processing time for these three models per testing. The results are consistent with the previous paper: PoseNet achieves the highest average accuracy at 98.04%, MoveNet is the fastest model, and OpenPose remains the slowest, taking approximately 13.5 times longer than MoveNet.

3.2.2. Shortcomings of the Current Papers

Considering the shortcomings of these two papers, they share three limitations.

Omission of BlazePose in Benchmarking

First, both papers benchmark only OpenPose, MoveNet, and PoseNet, but omit BlazePose. Despite being published after 2022, and although BlazePose was released in 2020, it was not included in their comparisons. This is a significant omission, as BlazePose is also a well-known and advanced pose estimation model worth evaluating.

Insufficient Information for Judgment

Second, neither paper provides sufficient information for us to determine whether these pose estimation models can be effectively applied to our task. Since we aim to develop a real-time posture correction application, latency is a critical factor. Although the papers report processing times, they do not specify the hardware or environment used for benchmarking. This makes it difficult to assess whether the models are fast

enough for our use case, as they might have been tested on highly advanced hardware setups that do not reflect typical deployment environments.

Potential Dataset Bias in Benchmarking Results

Third, the datasets used for benchmarking in these two papers may introduce bias into the results. Both papers use COCO or MPII datasets for benchmarking. While these are mainstream open-source datasets for human pose estimation tasks, it is difficult to rule out the possibility that the pre-trained weights of the models were fine-tuned using these datasets. Although we cannot find official documentation on the datasets used to train the pre-trained weights of the three models, it is widely inferred that PoseNet and MoveNet were trained with the COCO dataset, as both output 17 keypoints that match the labeled keypoints in COCO. Therefore, the results of these two papers may be biased, as they use these large public datasets for testing.

3.3. Existing Research on Posture Correction

3.3.1. Overview of Existing Research

We found two research papers with objectives similar to our project's: Real-time Exercise Posture Correction Using Human Pose Detection Technique [14] and Real-Time Workout Posture Correction using OpenCV and MediaPipe [15].

The first paper, published in 2021 by SUNY Polytechnic Institute, aims to develop a real-time exercise posture correction system using human pose detection. The goal was to help people exercise with proper guidance at home, especially during the COVID-19 pandemic when access to gyms and personal trainers was limited. The research used squat exercises as a test case, employing a pre-trained OpenPose model to extract body keypoints and determine whether the pose was correct. This determination was made using a hardcoded threshold based on the angle between the hip, knee, and ankle, calculated from their 2D (x, y) coordinates. However, the system could only function properly with image inputs, as real-time pose detection suffered from lag due to limited computational power.

The second paper, published in 2022 by the National Research Foundation of Korea, has a similar goal to both our project and the first paper: to help users maintain correct posture while exercising at home without the need for a personal trainer. This paper focused on two exercises: squats and push-ups. The pipeline is similar to the first paper in that it first extracts keypoint coordinates of the user's body and then uses this information to assess posture correctness. However, the main difference is that BlazePose from MediaPipe was used to extract keypoints, rather than OpenPose.

Notably, BlazePose predicts an additional dimension, depth, compared to OpenPose. The extracted keypoints are used to calculate body angles, which are then compared to predefined correct ranges to evaluate posture accuracy. The paper reports an accuracy of 100% for squats and 80% for push-ups.

3.3.2. Heuristics in Existing Papers

The two existing papers we reviewed share a similar approach to solving posture correction. Although they use different models in practice, both papers follow the same general process: first, they extract the coordinates of key body parts, and then they assess posture correctness based on these coordinates. This approach is both intuitive and practical because, compared to directly evaluating posture correctness from raw image inputs (e.g., training a Convolutional Neural Network (CNN) from scratch), using pre-trained human pose detection models for feature extraction significantly reduces the workload and resource requirements. As a result, we adopted a similar design, using a two-stage architecture: first, we extract keypoints, and second, we evaluate posture correctness based on those keypoints.

3.3.3. Shortcomings in Existing Papers

While the existing papers successfully achieve their goals to some extent, we identified three shortcomings in their approaches that we aim to address.

High Preparatory Workload

Both papers assess posture correctness by defining a range of acceptable angles for specific body parts. However, determining these predefined ranges is not a straightforward task for developers, and there is no official list of correct angle values for various exercises available on the internet. Given a set of training data with samples of correct and incorrect postures, developers would need to manually analyze this data to determine the appropriate angle ranges. This process becomes increasingly labor-intensive as the complexity of the posture increases beyond simple exercises like squats and push-ups. Additionally, developers may also need prior knowledge of the correct posture to define these ranges accurately.

To address this issue, our solution uses a deep learning model in the second stage, rather than relying on simple conditional logic based on body part angles. This would allow developers to implement the system simply using training data with positive and negative posture samples, without the need for manual analysis of correct angle ranges.

Lack of Scalability

In addition to the high preparatory workload discussed in the previous paragraph, using predefined angle ranges as the criterion for correctness also limits the system's scalability. Since different exercises have different angle criteria, developers would need to implement separate code for each one. While it is possible to integrate all criteria and run the appropriate one based on conditional branching (e.g., first classify the action/posture and then apply the corresponding check), this introduces additional complexity. Developing an action/posture classification system is another task that would need to be completed beforehand, further complicating the process and making it difficult to scale the system to support more postures over time.

To overcome this, our solution uses a deep learning model in the second stage that can simultaneously classify the posture type and evaluate its correctness. This approach simplifies development, reduces manual processes, and improves scalability, allowing the system to support a wider range of postures with minimal additional effort.

Limited Camera Angles

This shortcoming is specific to the first research paper, which uses OpenPose as the human pose detection model to extract body keypoint coordinates. OpenPose, however, can only estimate 2D coordinates (x and y dimensions). As a result, the calculated angles of body parts may vary depending on the camera's point of view. For example, a squatting leg should form a 90-degree angle from a side view, but if the camera is positioned in front, the 2D projection of the thigh and calf will appear nearly aligned. In this case, the calculated angle will not be 90 degrees, leading to incorrect posture evaluation, even though the user may be performing the posture correctly. This issue is especially problematic when angle thresholds are hardcoded, as is the case in the implementation of the research papers. One way to address this limitation is by using a human pose detection model that supports 3D coordinate estimation, such as BlazePose, which was used in the second research paper.

In our solution, we do not pre-select a specific pre-trained human pose detection model for the first stage of keypoint extraction. Instead, we are benchmarking multiple mainstream models, as detailed in [Section 4](#), to identify the best fit for our project. Acknowledging the limitations of models that only provide 2D coordinates, we will take this factor into account, alongside benchmark results, when selecting the model to help address this shortcoming.

4. Benchmark on Existing Pose Detection Models

(Completed in the first semester)

As outlined in our project solution (see [Section 2.4](#)), we aim to use an existing pose detection model to process input images (video frames) and extract the user's body keypoints as the first stage of our system. Among several studies on posture detection and comparison of pose estimation models [12] [13] [14] [15], we identified four mainstream, commonly used models: OpenPose, PoseNet, MoveNet, and BlazePose. Although there is existing research comparing some of these models [12] [13], it does not provide sufficient heuristics, as discussed in [Section 3.2.2](#), to determine which model is best suited for our project. Therefore, we conducted a thorough and fair comparison of their performance in terms of latency, accuracy, and resource usage to select the most suitable model.

4.1. Detailed Approach

Our benchmarking process is straightforward. We are running the four different models on the same dataset and measuring their performance. Specifically, we evaluate:

- Latency: The time taken by the models to process the images.
- Accuracy: How well the models identify and infer keypoints.
- Memory Usage: A rough reference value of the total memory consumption after the inference process on the entire benchmark dataset.

For both latency and memory usage, we are running each model three times and taking the average to ensure more accurate results. The benchmarking is performed under the same conditions (e.g., platform, GPU, software version) to ensure fairness.

4.1.1. Dataset

We used a public dataset (hereinafter referred to as the base dataset) from Kaggle [16] that originally consisted of 1,171 images across six different yoga postures: Down Dog, Goddess, Plank, Side Plank, Tree, and Warrior. After revising the dataset, we removed duplicate images (based on SHA-256 hash values, using the `hashlib` library in Python) and excluded irrelevant images, such as cartoon-style illustrations, to ensure the dataset focuses on real-life scenarios. The resultant dataset for this benchmark now contains 926 images. [Figure 3](#) shows some samples from the dataset after revision.

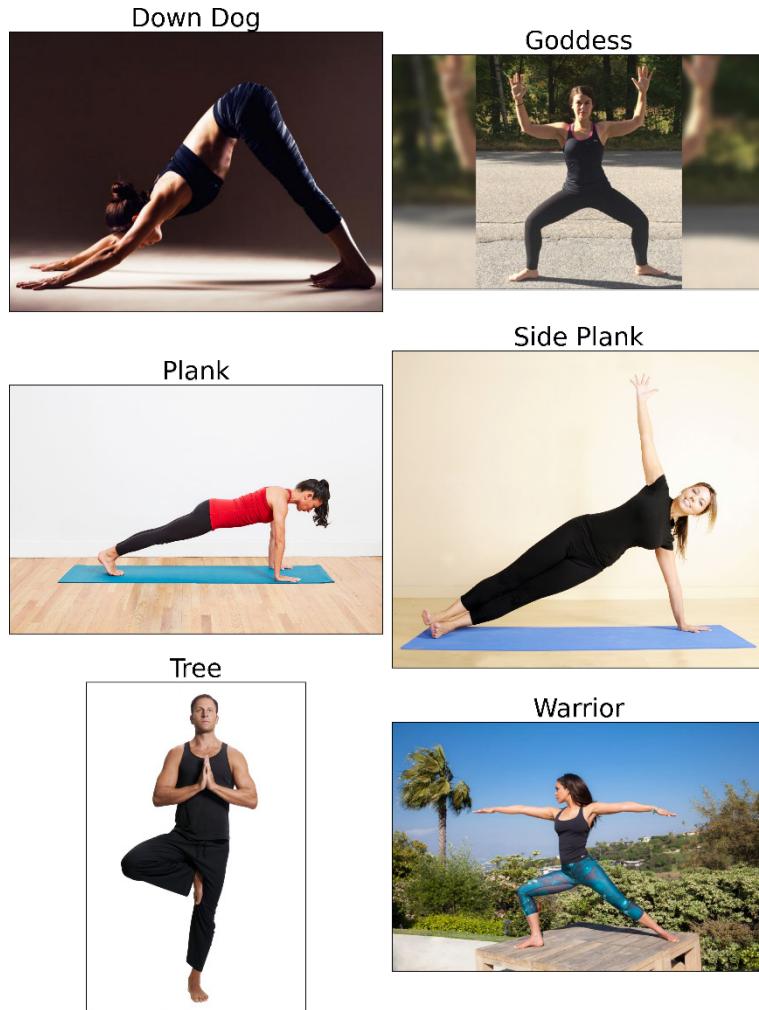


Figure 3. A sample from the dataset, showing one image for each of the six postures.

4.1.2. Evaluation of Model Latency

Latency is measured per image by dividing the total time taken by the number of input images. We only measured the time required for model inference, excluding other processes such as initialization. All four models processed the images one by one (not in batches) to ensure identical conditions for a fair comparison.

4.1.3. Evaluation of Accuracy

Since the dataset consists of raw images of yoga poses without pre-labeled keypoints, the most intuitive way to evaluate the models' accuracy is through human evaluation. We randomly sampled images of four different poses from the dataset and visually inspected the keypoints generated by the four models to assess their accuracy.

The number of keypoints output by different models may vary. Therefore, we evaluated the models using the widely adopted set of 17 COCO keypoints [17]. COCO, which stands for "Common Objects in Context," is a large-scale dataset designed for various computer vision tasks, including keypoint detection [18]. The official COCO keypoints represent the following body parts:

| | | | | | |
|----------------|------------|-------------|------------|-------------|---------------|
| Nose | Left Eye | Right Eye | Left Ear | Right Ear | Left Shoulder |
| Right Shoulder | Left Elbow | Right Elbow | Left Wrist | Right Wrist | Left Hip |
| Right Hip | Left Knee | Right Knee | Left Ankle | Right Ankle | |

For our accuracy evaluations, we focused exclusively on these 17 keypoints, as they are shared among the models we are comparing, ensuring a consistent benchmark.

When visualizing the keypoints on an image, we connect specific keypoint pairs to form a skeleton, which helps facilitate the assessment of keypoint accuracy. The connection rules remain consistent across all models in our evaluation and are based on references from the official TensorFlow site [19].



Figure 4. A sample image showing the 17 COCO keypoints and the paired connections used in this benchmark.

4.1.4. Evaluation of Memory Usage

Different models may be deployed using different frameworks. For instance, PoseNet uses TensorFlow LiteRT, while BlazeNet relies on the MediaPipe module. This variation complicates the definition and measurement of memory usage across models. To simplify and maintain consistency in our benchmarks, we measured peak memory usage immediately after processing the entire dataset with each model, using the code

`resource.getusage(resource.RUSAGE_SELF).ru_maxrss` , as referenced from StackOverflow [20]. This approach measures the usage of unified memory, where the memory is shared between the CPU and GPU [21]. It's important to note that this provides only a rough estimate, as memory usage can vary significantly based on factors such as changes in data processing.

4.2. Code Implementation

For the benchmarking code, we primarily referenced the sample codes provided for each model. It's important to note that the raw output formats of the four candidate models differ significantly, as shown below:

| | OpenPose | PoseNet | MoveNet | BlazePose |
|---------------|-----------------|--|--|--|
| Output Format | N/A | A heatmap along with offset vectors [22] | Relative coordinates of the input image padded into a square | Relative coordinates directly from the input image |

As a result, we put effort into standardizing these outputs into a unified format: an array of shape "number of keypoints" × "keypoint dimensions".

Additionally, we implemented a function called `save_image_with_prediction()` to plot the predicted keypoints and draw the skeleton for visualization, as shown in [Figure 4](#). Although the documentation also provides a plotting function, we chose not to use it because its implementation, which relies on the Matplotlib backend, distorts the original image by adding borders or reducing resolution. Instead, we implemented our solution using OpenCV as the backend, which avoids these issues and preserves the image quality even after plotting the skeleton.

4.3. Benchmark Results

4.3.1. Result Table

| | OpenPose | PoseNet | MoveNet | BlazePose |
|---|-----------------|----------------|----------------|------------------|
| Year of First Release | 2017 | 2017 | 2021 | 2020 |
| Year of Latest Version | 2020 | 2021 | 2023 | 2023 |
| Number of Keypoints | 135 | 17 | 17 | 33 |
| Keypoint Coordinate Dimensions | x, y | x, y | x, y | x, y, z |
| Total Inference Time | N/A | 33.8224 s | 31.6186 s | 24.5882 s |
| FPS (Frames per Second) | N/A | 27.3783 | 29.2866 | 37.6603 |
| Approximate Memory Usage (Unified Memory) | N/A | 4.36 GiB | 5.01 GiB | 4.59 GiB |

This table shows the benchmark results of the four candidate models, highlighting their latency and memory usage, along with some background information about each model. A detailed analysis of the results can be found in [Section 4.4](#).

4.3.2. Result Samples

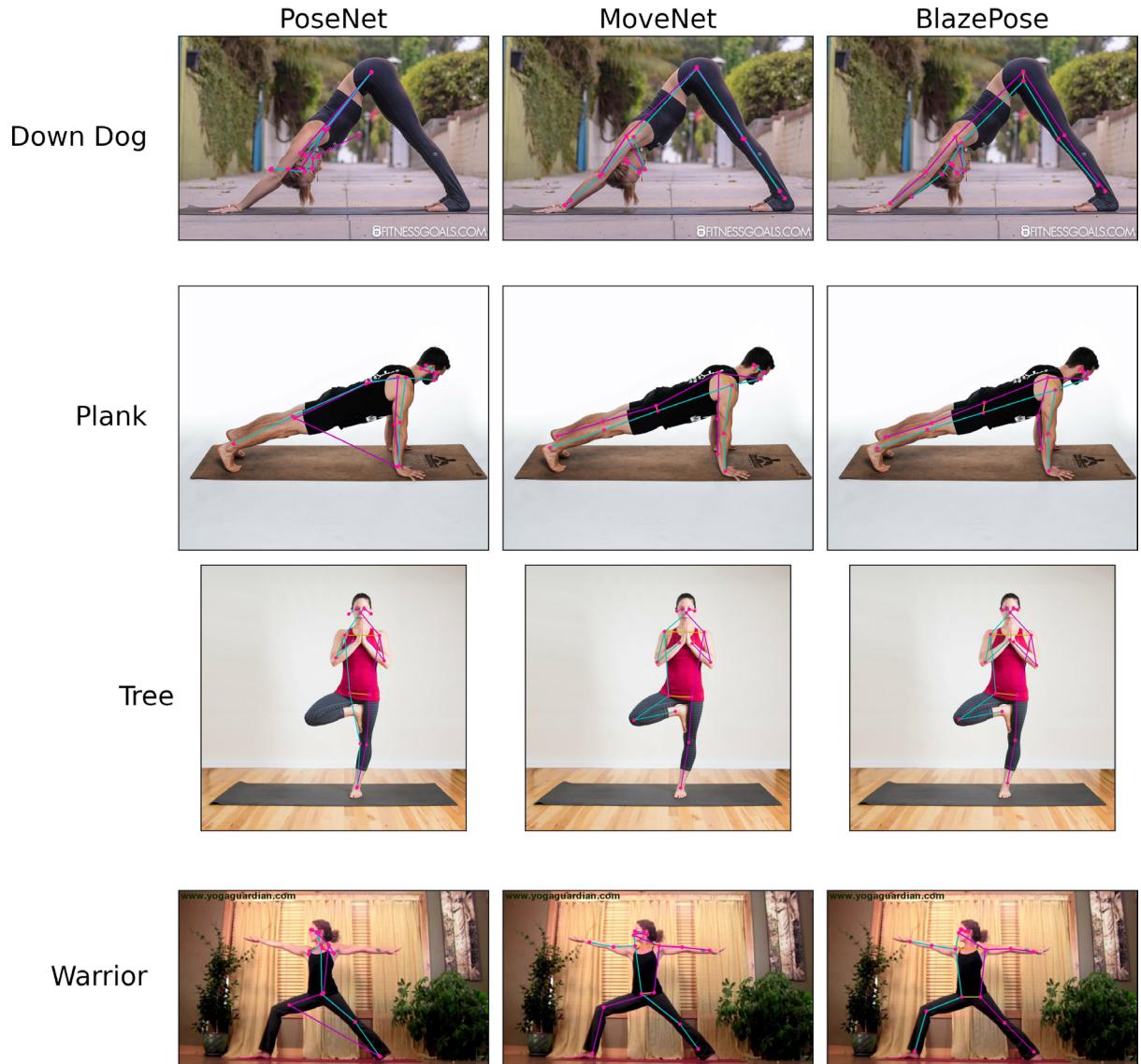


Figure 5. Benchmark results of the three models showing pose detection accuracy. Each column represents a model: PoseNet performs the worst with the skeleton misaligned, while MoveNet and BlazePose perform similarly.

4.4. Model Results Analysis

4.4.1. OpenPose

OpenPose [23], developed by Carnegie Mellon University, is claimed to be the first real-time, multi-person system capable of detecting human body, hand, facial, and foot keypoints (a total of 135 keypoints) in single images. It was first released in 2017, with the latest update in 2020.

In this benchmark, OpenPose did not run successfully. The issue was caused by the failure to download the pre-trained weights from the server using the setup command provided in the documentation. This may be due to the model being out of maintenance. Additionally, as a model last updated in 2020, OpenPose is not as up-to-date as the other three benchmarked models and lacks official support for Python versions newer than 3.7. Moreover, OpenPose is written in C++, rather than commonly used Python deep learning libraries like PyTorch or TensorFlow, making it more difficult to inspect or modify its structure. It also cannot be easily installed via pip, further complicating the setup.

Regarding the issue with downloading the weights, we found a related discussion on the OpenPose GitHub repository [24]. However, no update had been made at the time of our benchmark. As a workaround, we referred to two existing research studies [12] [13] (introduced in [Section 3.2.1](#)) that compared OpenPose with PoseNet and MoveNet. Both studies concluded that OpenPose offers no advantage in terms of accuracy or speed compared to the other two. Based on these findings, we reasonably concluded that OpenPose is not suitable for the purposes of our project.

4.4.2. PoseNet

PoseNet is a human pose estimation model developed by Google, first released in 2017, capable of detecting 17 COCO keypoints.

During the planning phase of our project, we were already aware that PoseNet is no longer actively maintained, as Google has developed a successor called MoveNet. According to Google's official statements [25], MoveNet outperforms PoseNet in both latency and accuracy across various datasets. Despite this, we still decided to simply benchmark PoseNet to gain hands-on experience with the environment setup and assess the actual performance difference between these two generations of pose estimation models from Google.

For this benchmark, PoseNet was run in CPU mode. We had already confirmed that our TensorFlow environment had GPU support enabled by using the official check command, as shown below:

```
>>> print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))  
Num GPUs Available: 1
```

Additionally, we successfully deployed other models in the same environment using the GPU. Therefore, the likely cause of PoseNet's failure to run on the GPU could be unsupported GPU operations in the model. This issue is further explained by the fact

that TensorFlow LiteRT, which PoseNet's main version is based on, supports only a limited set of machine learning operations [26].

Although there may be a workaround to deploy PoseNet on the GPU, we decided not to invest too much time in pursuing this, as PoseNet is deprecated. Our primary objective was simply to familiarize ourselves with the environment setup. Thus, we proceeded with CPU mode, but the time performance was no longer suitable for direct comparison.

In terms of benchmark results, PoseNet showed significant accuracy degradation when the person's pose was not facing the camera, such as in the downward dog and plank poses. Despite running on the CPU, PoseNet achieved around 27.4 fps (926 images / 33.8 seconds), which is acceptable. The memory consumption after the inference process was 4.36 GiB.

4.4.3. MoveNet

As mentioned earlier, MoveNet is the successor to PoseNet, first released in 2021 [19]. It detects the same 17 COCO keypoints as PoseNet but remains one of Google's most advanced human pose estimation models, known for its ultra-fast performance and high accuracy.

Google offers several versions of MoveNet, varying in precision and model size to suit different platforms and user needs. For this benchmark, we selected the default version, MoveNet Lightning (FP32), which balances speed and accuracy.

MoveNet performed well in skeleton estimation across all random samples. The skeleton was accurately drawn regardless of posture, camera angle, or the direction the user was facing. In terms of latency, MoveNet achieved 29.3 fps (926 images / 31.6 seconds) with GPU utilization, and the total memory consumption after the inference process was 5 GiB.

4.4.4. BlazePose

BlazePose, also known as MediaPipe BlazePose, is another human pose estimation model developed by Google, first released in 2020. BlazePose is part of the MediaPipe Solutions framework and is used for pose landmark detection [27]. Unlike other models, BlazePose can detect not only the 17 COCO keypoints but also an additional 16, for a total of 33 keypoints, making it another state-of-the-art pose estimation model from Google.

BlazePose also estimates the z-coordinate (relative depth) of each keypoint, in addition to the 2D coordinates (x , y), allowing for more accurate joint angle estimation regardless of camera angle. However, for this benchmark, we only considered the x and y coordinates when comparing accuracy.

In the MediaPipe interface, model complexity is a customizable parameter that allows developers to choose between better accuracy or lower latency. For this benchmark, we selected the default setting, `model_complexity=1`.

BlazePose performed well in skeleton estimation, comparable to MoveNet. There were no significant issues, and the skeleton was consistently well-fitted to the user's body, regardless of posture. In terms of latency, BlazePose achieved 37.6 fps (926 images / 24.6 seconds) with GPU utilization, and the memory consumption after the inference process was 4.6 GiB.

4.5. The Chosen Model

Based on the individual result analysis in [Section 4.4](#), we concluded that BlazePose is the most suitable model among the four for our project. Below, we provide a more detailed explanation of accuracy, latency, memory usage, and other benefits to support this conclusion.

4.5.1. Accuracy

In terms of accuracy, as demonstrated by the sample estimations in [Section 4.3.2](#), both MoveNet and BlazePose perform almost identically. Both models significantly outperform PoseNet, regardless of whether the background is simple or complex. Even for a human observer, it is difficult to easily judge which model, MoveNet or BlazePose, provides better estimations. However, upon closer inspection, BlazePose produces a more reasonable representation of the hip area in the Warrior pose output sample. Thus, we conclude that BlazePose performs the best in this regard.

4.5.2. Latency and Memory Usage

When it comes to latency and memory usage, BlazePose offers lower latency and consumes less memory. We can confidently say that BlazePose outperforms MoveNet in terms of computational efficiency and resource usage, at least when both models are used with their default configurations.

4.5.3. Additional Benefits

In addition to its advantages in accuracy, latency, and memory usage, BlazePose offers several additional benefits. It provides more keypoints and includes an extra dimension (relative depth), which MoveNet lacks. Furthermore, BlazePose is an up-to-date model that continues to receive maintenance and updates, ensuring long-term support and improvements. For these reasons, BlazePose is the model we have chosen for the next stages of our project.

5. Dataset Preparation

(Completed in the first semester)

In this section, we will explain and introduce the dataset preparation process. We have constructed a dataset tailored to our project's goals by incorporating images collected from Kaggle and YouTube. To clearly present our process of building the dataset, we will separate what we have done in two main phases:

The first phase of our dataset preparation, which took place before mid-October, focuses on searching for and analyzing existing datasets on Kaggle. We analyzed the existing public datasets to assess their suitability for our project and how they could contribute to further development. Additionally, we identified several potential issues within these datasets and implemented optimizations to address them.

The second phase, which took place after mid-October, involved augmenting our dataset with frames from YouTube videos. We work on identifying and classifying the four target poses with evidence supported by yoga articles and videos, distinguishing between positive and negative samples, and resolving any ambiguities in the images. By the end of this phase, we had finalized a comprehensive dataset suitable for our model training.

5.1. First Phase Dataset Preparation

5.1.1. Observations

As mentioned in [Section 4.1.1](#), we utilized a dataset sourced from Kaggle as our base dataset, which was used in the benchmarking process. Before October, out of 6 poses datasets within the base dataset, we selected Down Dog, Plank, Tree, and Warrior as our basic cases. These 4 poses had the highest number of images available for pose correction, making them the basic cases of our project. However, upon observing the data, we identified 4 issues within the current dataset as follows:

Duplicate Images

Although the base dataset has a diverse variety of images regarding each pose, we observed that some images are duplicated within the dataset. This leads to redundancy, consuming extra computational resources and storage. Moreover, it leads to a biased analysis of our model toward new data in future development.

Irrelevant Images

Our project aims to secure pose correction for adolescents and adults in real-time. Considering this, some unrelated images were observed in the dataset. These images include cartoon-style illustrations, images of children doing specific yoga poses, and incomplete poses. This issue might lead to reduced performance of the model if it learns from these irrelevant patterns and unrelated images.

Misclassified Images

First, we identified that the base dataset is not accurately categorized. For example, the Warrior dataset included 4 distinct poses instead of 1 specific pose, which are Warrior I, II, III and Peaceful Warrior.

Second, we observed data interchange between datasets, such as images of side planks present in the plank dataset, and vice versa.

Potentially Suboptimal Image Selections for the Dataset

First, upon observing the previous issue of "[Misclassified Images](#)", we concluded that some of our initially chosen poses might not be optimal for future development. This is due to the potential reduction in the number of images in these datasets after correct categorization.

Second, a comparison of the benchmark images reveals that it is difficult to distinguish some poses' correctness, as the model might not be able to detect minor deviations through skeleton structures and joint angles. For example, yoga tutorials introduce common mistakes in tree poses, such as trainers not keeping their weight evenly distributed in their supporting foot and failing to engage their core muscles to hold the torso upright [28]. These are subtle pose inaccuracies that can be difficult to recognize.

5.1.2. Dataset Optimization

5.1.2.1. Enhancement of Dataset Size

To train the model with a larger image dataset, we explored four additional datasets from Kaggle [29] [30] [31] [32]. After comparing the dataset sizes, we selected Down Dog, Plank, Side Plank, and Warrior II as the poses for our latest dataset (whereas the original dataset included Down Dog, Plank, Tree, and Warrior).

This decision is based on the abundance of images available across the Kaggle datasets, and this solves the "[Potentially Suboptimal Image Selections for the Dataset](#)" issue raised in [Section 5.1.1](#). Below is a table showing the number of images for each pose after combining the five datasets into one single dataset:

| Datasets | Number of Images in the Original Dataset | Number of Images after combining 5 Datasets |
|------------|--|---|
| Down Dog | 253 | 745 |
| Plank | 153 | 328 |
| Side Plank | 106 | 168 |
| Warrior II | 249 | 351 |

5.1.2.2. Enhancement of Dataset Correctness

In order to enhance the accuracy and correctness of our dataset, we conduct the following 2 procedures:

Duplicate Removal

Similar to [Section 4.1.1](#), we removed duplicate images using Python scripts by comparing the SHA256 values of the images in the dataset.

Additionally, we utilized the Duplicate File Finder by Nektony [33] to ensure all the duplicates are removed, which guarantees the reliability and completeness of our dataset to avoid suboptimal model performance. These approaches solve the "Duplicate Images" issue mentioned in [Section 5.1.1](#).

Dataset Categorization

Problems, including "[Irrelevant Images](#)" and "[Misclassified Images](#)" introduced in [Section 5.1.1](#) are solved upon dataset categorization. However, unlike other data preparation procedures, we categorize the pose data by manually distinguishing its correctness. Therefore, further confirmation on classification is required to ensure the accuracy of the dataset in later phases.

5.2. Second Phase Dataset Preparation

5.2.1. Observations

After building up the dataset by merging existing datasets from Kaggle and categorizing the pictures preliminarily, we have observed 4 issues within the dataset as stated in the following sections.

Limited samples for certain postures

After dataset optimization in the first phase, some postures had adequate images for development, while certain were limited. For example, the Down Dog dataset contained 745 images, whereas the Side Plank dataset had only 168 images. Similarly, both the Plank and Warrior II datasets only contained half the number of images compared to the Down Dog dataset. Therefore, it is necessary to augment more data for all the postures in the datasets to enable the model to generalize effectively to the real-life scenarios targeted in our project.

Insufficient negative samples

In addition to the issue highlighted in the previous section, addressing the shortage of negative samples (images showing incorrect poses) remains a priority. All the datasets from Kaggle consist mainly of correct samples for each posture. Considering that one of our project features involves real-time correction of users' poses, negative samples should be included in the dataset to assess posture accuracy and facilitate model training.

Unclassified dataset

As mentioned in the previous "[Insufficient Negative Samples](#)" section, there are inadequate negative samples available for further development. Meanwhile, the classification and categorization of the temporary dataset remain incomplete, as stated in "[Dataset Categorization](#)" in [Section 5.1.2.2](#). Therefore, apart from augmenting the dataset with negative samples, it is also important to classify both the positive and negative samples within the temporary dataset.

Lack of variant samples

While the current dataset contains appropriate images for our model training, it may not adequately detect and correct every user pose in the final product. This limitation arises from the majority of images in the temporary dataset being captured within a specific range of camera angles. Although certain poses, such as the plank, can only be assessed from horizontal angles, it is crucial to include images of other poses from diverse angles and backgrounds to enhance the accuracy of the model.

5.2.2. Dataset Construction

To solve the issues outlined in [Section 5.2.1](#), we decided to extract images from YouTube videos. By reviewing videos on YouTube that demonstrate common mistakes and proper poses, as well as yoga-related articles, we further understood the characteristics of the four poses we selected. This enabled us to classify the dataset correctly. Additionally, we can add negative samples and real-life images to our dataset, enhancing the model's accuracy and our project's credibility.

In the following sections, we will first introduce the key characteristics and common mistakes of our targeted postures.

5.2.3. Characteristics of the Chosen Poses

In this section, we will introduce the key characteristics of the chosen postures to present the method we used to determine whether an image is considered as positive or negative manually. Also, we will discuss the ambiguity of certain images and how we handle with these special cases.

Characteristics of Down Dog

The correct Down Dog posture requires the body of the practitioner to form an inverted V-shape, with hips lifted towards the ceiling, hands and feet pressing into the mat or ground. Then, legs should be extended, and arms should be in a straight line with the spine [34].



Figure 6. Positive Samples of the Down Dog pose. (Inverted V-shape body with straight arms and legs)

Common Mistakes of Down Dog

Meanwhile, we classify images with the following mistakes as negative samples. First, the practitioner rounds their back instead of maintaining a straight spine [35]. Second, the arms and the back are not in a straight line but form an obtuse angle. Third, there is a bend in the knees.



(a) Down Dog Negative Sample 1



(b) Down Dog Negative Sample 1

Figure 7. Negative Samples of the Down Dog pose. ((a) shows a rounded back, with the arms and back not aligned in a straight line; (b) shows a bent knee.)

After recognizing the characteristics of Down Dog pose, we selected 5 videos from YouTube to conduct frame extraction [36] [37] [38] [39] [40].

Characteristics of Plank

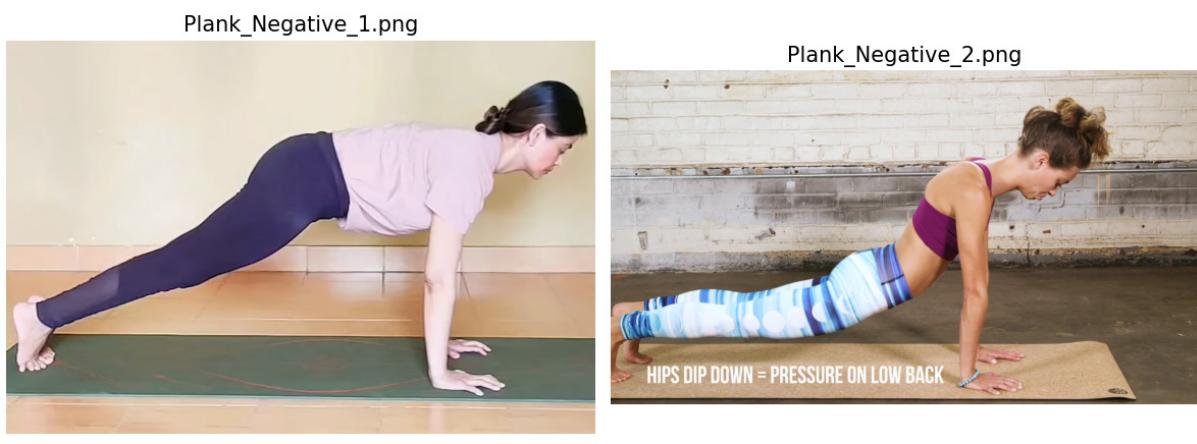
The correct Plank posture requires the body to form a straight line from the head to the heels, while the practitioner is facing downwards [41]. Similarly, the forearms and the upper arms should be in a straight line and perpendicular to the ground.



Figure 8. Positive Samples of the Plank pose. (A straight line from the head to the feet, with the entire arm perpendicular to the ground.)

Common Mistakes of Plank

Common mistakes in the Plank pose include that the hips of the practitioner being raised too high or dropping too low, preventing the formation of a straight line in the body [42]. Second, the arms are bended or not perpendicular with the surface are also a frequent mistake. Besides, there is a special case of negative samples. For the Purvottanasana pose, it has all the characteristics of the correct Plank posture, but the practitioner has to face upwards.



(c) Plank Negative Sample 1

(d) Plank Negative Sample 2

Figure 9. Negative Samples of the Plank pose. ((c) and (d) show a high and low positioned hip respectively.)



Figure 10. Another Negative Samples of the Plank pose. ((e) shows a bent-arm plank; (f) shows the Purvottanasana pose with the face looking upwards.)

After recognizing the characteristics of Plank pose, we selected 4 videos from YouTube to conduct frame extraction [43] [44] [45] [46].

Characteristics of Side Plank

To perform a correct Side Plank, the practitioner starts in a plank position and then shifts weight onto one hand, stacking feet on top of each other. The body is balanced on one hand and the outer edge of one foot, creating a straight line from head to heels [47]. Besides, the support hand should have a right angle between the forearm and the upper arm, with the forearm touching to the ground and the upper arm perpendicular to the surface.

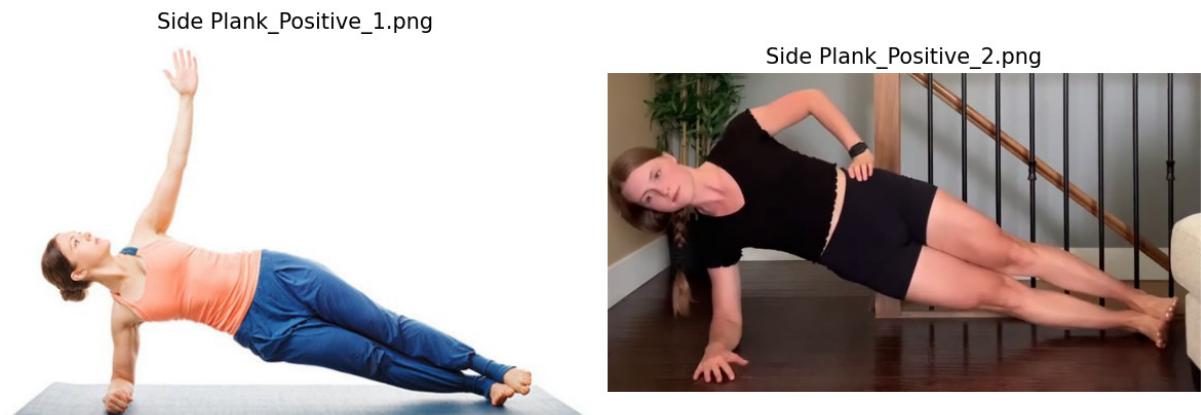


Figure 11. Positive Samples of the Side Plank pose. (A straight line along the body; one bent support hand, with the upper arm perpendicular to the surface.)

Common Mistakes of Side Plank

Meantime, the following Side Plank poses are considered as negative. First, placing the feet in incorrect positions, including placing the top foot in front of the bottom foot, separating the legs too wide apart and not extending straight. Second, similar to the Plank pose, the hips of the practitioner being raised too high or dropping too low and the torso is not in a straight line with the legs [48]. Third, the support hand is straight instead of bent.

The image contains two photographs of women performing a side plank exercise. In the left photograph, a woman with blonde hair tied back, wearing a black tank top and black leggings, is in a side plank position on a blue mat. Her right arm is extended upwards, and her left leg is bent at the knee. In the right photograph, another woman with blonde hair, wearing a purple tank top and pink leggings, is also in a side plank position. Her left arm is extended upwards, and her right leg is bent at the knee. Both images are labeled with their respective file names at the top.

After recognizing the characteristics of Side Plank pose, we selected 5 videos from YouTube to conduct frame extraction [49] [50] [51] [52] [53].

Characteristics of Warrior II

The key characteristics of Warrior II require the practitioner to stand in a wide stance. The front foot points forward and the back foot is angled slightly inward, with the leg spread wide apart [54]. The knee of the front foot is bent at a 90-degree angle, directly over the ankle, while the back leg remains straight [55]. Lastly, the torso should be upright, and the arms extended out to the sides, parallel to the ground [56] [57].



Figure 13. Positive Samples of the Warrior II pose. (An upright torso with arms parallel to the ground, a 90-degree bend in the front foot, and a straight back leg.)

Common Mistakes of Warrior II

Common mistakes in the Warrior II pose include shifting the torso too far forward or backward instead of keeping it upright [58]. Second, allowing the arms to sag or be raised too high [59]. Third, the knee of the front leg not being bent at 90 degrees, or the back leg not being straight.



(i) Warrior II Negative Sample 1

(j) Warrior II Negative Sample 2

Figure 14. Negative Samples of the Warrior II pose. ((i) shows a shifted torso and an incorrectly bent front foot; (j) shows misplaced arms and an incorrect angle of the front-foot knee.)

After recognizing the characteristics of Warrior II pose, we selected 5 videos from YouTube to conduct frame extraction [60] [61] [62] [63] [64].

5.2.4. Classification Process

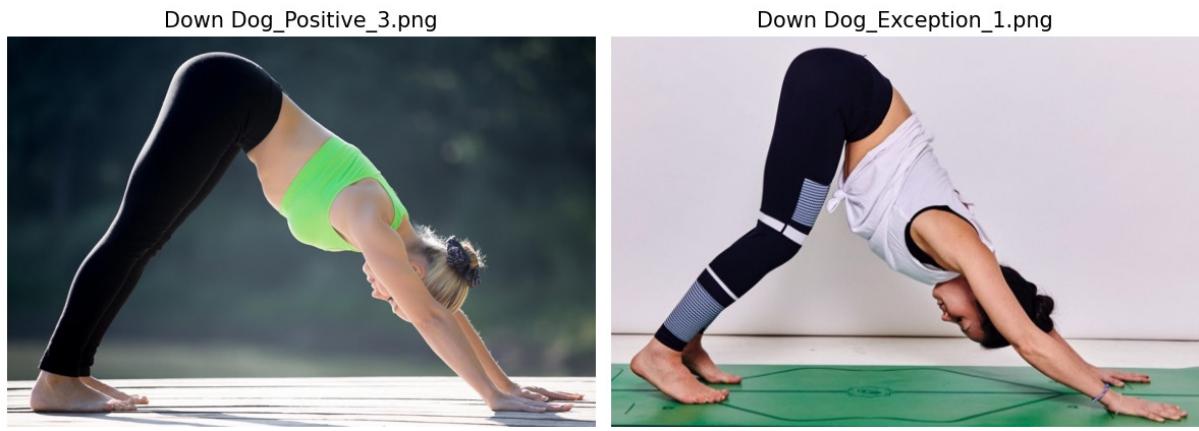
After we identified the characteristics of the correct and incorrect poses, we began searching for related videos on YouTube. During this phase, we found four to five videos for each pose and converted them into MP4 format for subsequent frame extraction.

Next, we recorded the frame rate, or frames per second (fps), for each video. To prevent an imbalance in the number of images between the Kaggle dataset and YouTube frames, and to avoid frames with high similarity, which might lead to overfitting or bias in the model, we extracted the frames at 0.25 of their original fps. For example, if the frame rate of the video is 60 fps, only 15 fps were extracted. We used a Python script and the built-in function of VLC media player, Scene Video Filter, for frame extraction.

With all the frames processed and collected into folders, we proceeded to classify them alongside the images from the Kaggle datasets by manually comparing the characteristics and mistakes of the postures. Following the classification, we organized all the data into the appropriate folders, finalizing the dataset.

5.2.5. Exceptional Cases

Although some images presenting poses are indeed considered correct in reality, they show different postures from the major characteristics mentioned above. In light of this, we classify them as negative in our dataset due to the scarce number of these images. For example, for the correct Down Dog pose, the practitioner can either extend the legs straight or bend the knees as long as the legs are stretched. However, since samples with bent knees are significantly fewer than those with straight legs, we decided to deem these fewer samples as negative. Similarly, it is common to do the Plank Pose with either the arms straight or bent, but we concluded that classifying images with straight arms as positive samples, and those with bent arms as negative samples, will be more suitable for our model due to the same reason mentioned above.



(k) Down Dog Positive Sample 3

(l) Down Dog Exception Sample 1

Figure 15. Comparison of Down Dog pose samples. (Although both images demonstrate correct posture, we classify (l) as negative due the limited number of samples showing a bent knee in the dataset.)



(m) Plank Positive Sample 3

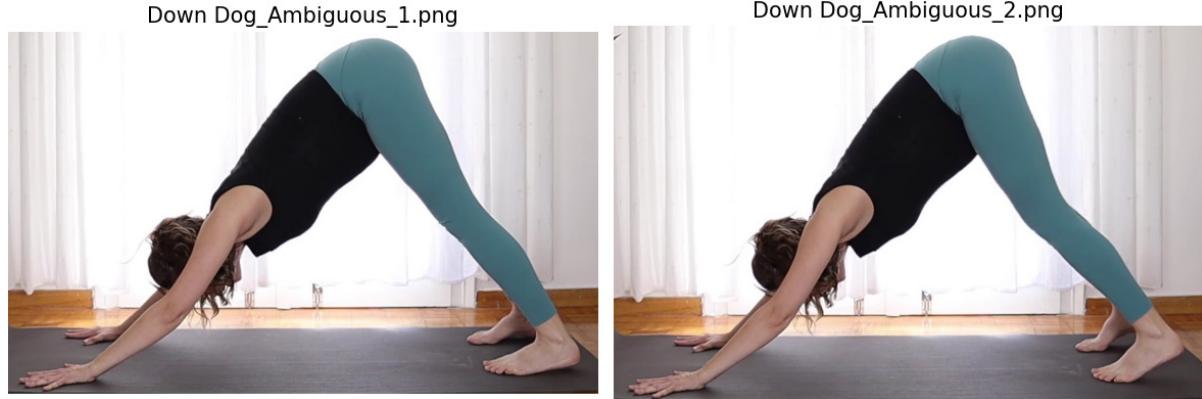
(n) Plank Exception Sample 1

Figure 16. Comparison of Plank pose samples. (Although both images demonstrate correct posture, we classify (n) as negative due the limited number of samples showing a bent arm in the dataset.)

5.2.6. Ambiguity of the Dataset

On the other hand, there are ambiguous images that show postures that are difficult to classify correctly. For example, as outlined in the "[Characteristics of Down Dog](#)" in [Section 5.2.3](#), one of the key characteristics of Down Dog is extending the legs. However, there are images where the practitioner bends the knee at a relatively minor angle, which makes the classification uncertain. Similarly, as stated in the "[Characteristics of Warrior II](#)" in [Section 5.2.3](#), a formal Warrior II pose requires the practitioner to bend the knee at a 90-degree angle, and both arms should be parallel to the ground. Obviously, there are scenarios where the knee is nearly 90 degrees, like 85

or 95 degrees, and the arms are not exactly parallel to the ground. These elements make the boundary line of classification unclear.



(o) Down Dog Ambiguous Sample 1

(p) Down Dog Ambiguous Sample 2

Figure 17. Ambiguous Samples of Down Dog pose. ((p) shows a knee that is slightly bent.)



(q) Warrior II Ambiguous Sample 1

(r) Warrior II Ambiguous Sample 2

Figure 18. Ambiguous Samples of Warrior II pose. ((r) shows that the front-foot knee is not exactly bent at 90 degrees.)

By investigating the number of these special cases, we found these samples only constitute a minor percentage of the whole dataset. Since the focus of our project is to distinguish between correct and incorrect images, ambiguous cases are not our primary concern. Therefore, we address these images by loosely classifying them based on human intuition, as no strict threshold can be applied to such ambiguities by a human labeler.

5.3. Dataset Size

After combining and classifying all the Kaggle datasets and the frames extracted from YouTube videos, we have constructed the finalized dataset containing adequate images for all the poses targeted this semester. Below is a table presenting the dataset size, including the number of positive and negative samples collected from Kaggle and YouTube, and the total number of images for different yoga postures collected throughout this semester.

| | Positive(+ve) Samples | | | Negative(-ve) Samples | | | Total (Ratio of +ve : -ve) |
|------------|-----------------------|---------|-------|-----------------------|---------|-------|-------------------------------|
| | Kaggle | YouTube | Total | Kaggle | YouTube | Total | |
| Down Dog | 591 | 59 | 650 | 145 | 697 | 842 | 1492 (650 : 842) |
| Plank | 217 | 304 | 521 | 112 | 547 | 659 | 1180 (521 : 659) |
| Side Plank | 11 | 146 | 157 | 157 | 185 | 342 | 499 (157 : 342) |
| Warrior II | 299 | 80 | 379 | 53 | 251 | 304 | 683 (379 : 304) |
| Total | 1118 | 589 | 1707 | 467 | 1680 | 2147 | 3854 (1707 : 2147) |

6. Model Development

(Completed in the first semester)

As outlined in [Section 2.4](#), after using a pre-trained human pose detection model (the first-stage model) to extract body keypoint coordinates from the raw input image, these coordinates are passed to the second-stage model, which is the focus of this section.

The second-stage model has two primary objectives: first, to classify the type of posture (e.g., plank, side plank, or down dog); and second, to estimate the correctness of the posture, with outcomes being either correct or incorrect.

6.1. Training Data

6.1.1. Raw Dataset

The raw dataset used to train the model is the yoga posture image dataset we constructed in [Section 5](#). It includes four types of yoga postures: Down Dog, Plank, Side Plank, and Warrior II. For each posture, we have both positive (correct) and negative (incorrect) samples, with a total of 3,854 images.

6.1.2. Data Splitting

The dataset is randomly split into training, validation, and test sets in a ratio of 70:15:15.

6.1.3. Data Balancing

Initially, the dataset was not perfectly balanced, with the least common posture type having approximately 66% fewer images compared to the most common one. While this imbalance was notable, we applied a weighted random sampler to ensure that each training batch had a roughly equal proportion of each posture type, both for positive and negative samples.

6.1.4. Data Augmentation

To increase diversity in the training set and reduce overfitting, we applied horizontal flipping as a simple data augmentation technique, effectively doubling the amount of data.

Other common augmentation techniques, such as padding, cropping, or resizing, were not applied because the raw images are processed by the first-stage model, and the extracted keypoint coordinates (after normalization) remain the same regardless of these augmentations.

6.2. Input Structure

6.2.1. Output of the First-Stage Model

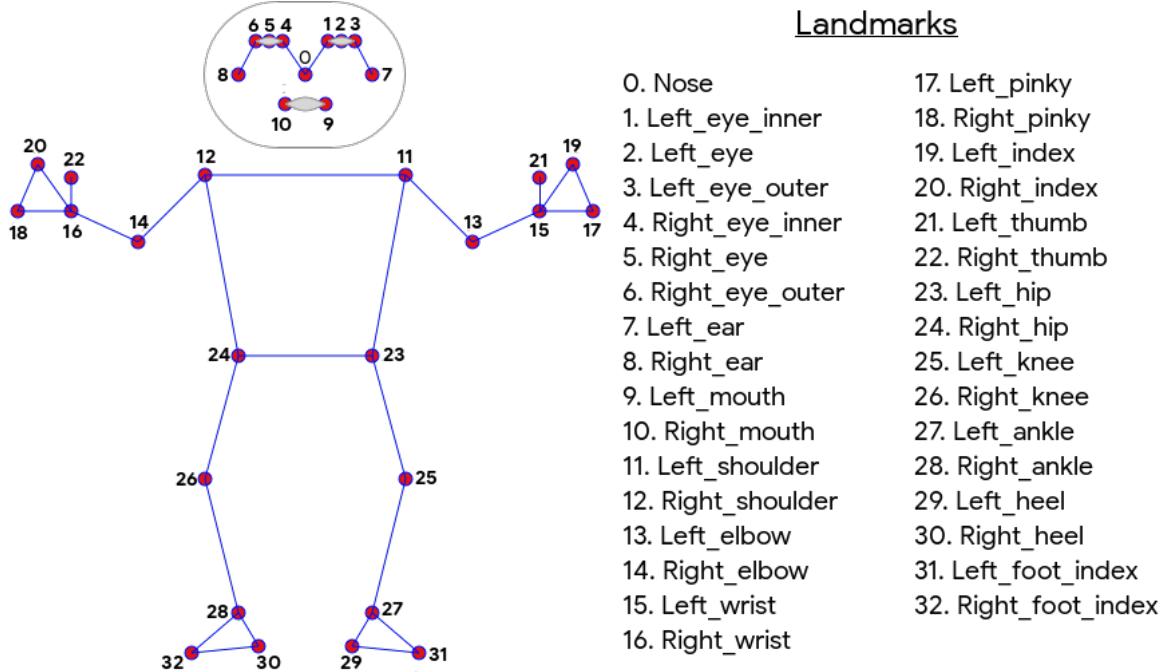


Figure 19. The 33 keypoints output by BlazePose. [65]

The first-stage model outputs a 2D array with a shape of (33, 4). Each of the 33 rows represents a body keypoint, and each keypoint consists of four values: relative x , relative y , relative z , and the visibility score.

x and y are float values, where (0, 0) corresponds to the top-left corner of the image, and (1, 1) to the bottom-right corner. Values outside the [0, 1] range indicate that the point is estimated to be outside the image.

z is another float that represents the relative depth from the plane of the subject's hips, which are considered the origin of the z -axis [66]. Negative values indicate that the point is between the hips and the camera, while positive values indicate the point is behind the hips.

Visibility is a float in the range [0, 1], representing the likelihood that the keypoint is visible and not occluded [67]. This can also be interpreted as the confidence or reliability of the keypoint.

6.2.2. Normalization of x and y

Since x and y are relative to the entire input image, discrepancies can occur if two images show the same posture but the subject is positioned differently or at varying

distances from the camera. In such cases, the extracted keypoint coordinates would be completely different, which is not ideal because we want identical postures to have identical coordinates. To address this, we normalize the x and y coordinates using a MinMaxScaler, applied to each axis independently. Geometrically, this is equivalent to cropping the smallest bounding rectangle that includes all keypoints, and then resizing it to a square with coordinates ranging from [0, 1] on both axes, as shown in [Figure 20](#).

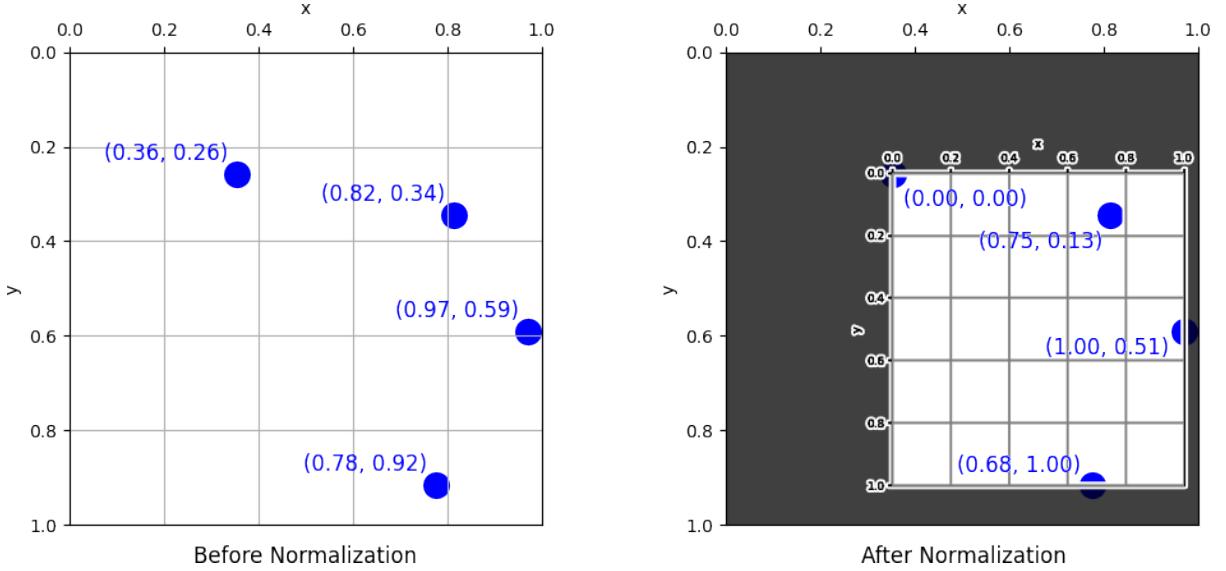


Figure 20. Illustration of the geometric interpretation of normalization for relative x and y coordinates, demonstrated with four sample points.

6.2.3. Normalization of z

Normalizing z is more complex than normalizing x and y , as relative depth is abstract and cannot be visualized directly. The official documentation [67] does not clarify whether the z -values correlate with the subject's distance from the camera or if they are based on the subject's height as a reference length. Specifically, if z is normalized relative to the distance between the subject's hips (the origin of the z -axis) and the camera, the z -value of a point will change as the subject moves closer or farther away, even if the pose remains the same. However, if z is based on the subject's height as a reference length, the z -value of a point will remain constant regardless of the subject's distance from the camera, as long as the pose is unchanged.

To investigate this, we tested two images with identical poses but different scales, positions, and aspect ratios (as shown in [Figure 21](#)) to observe how these factors affect the raw z -values. [Figure 22](#) shows the results, where the plot on the left (before normalization) shows that the z -values of the two images are scaled differently, suggesting that z correlates with the subject's scale (i.e., the user's distance from the camera). To address this, we applied L2 norm normalization to the z -coordinates,

keeping the hips as the origin of the z-axis. The plot on the right (after normalization) shows that the z-values are consistent across both images, confirming that normalization resolves the scaling issue.

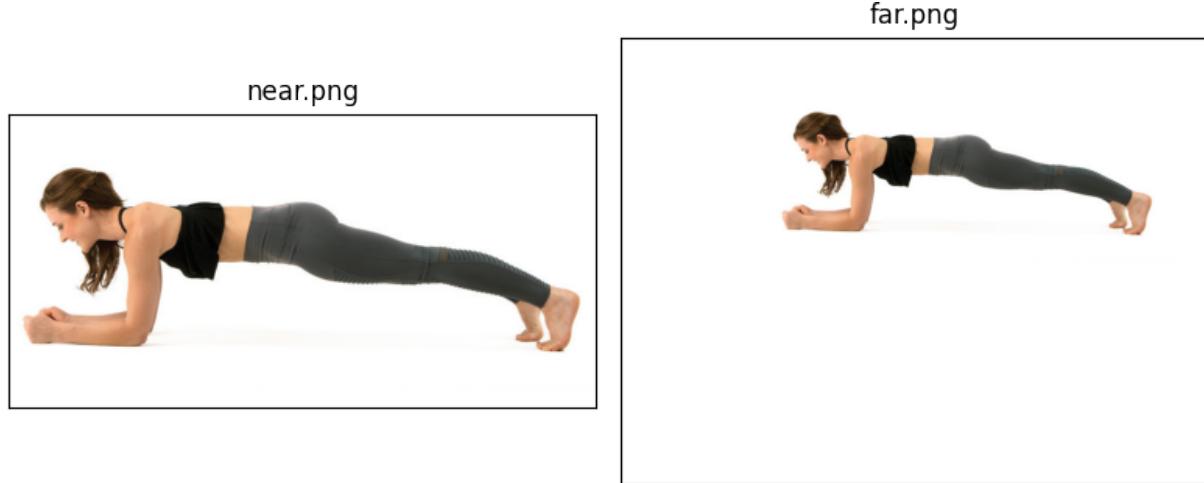


Figure 21. Two image samples used to verify the correlation between predicted z-coordinates and the subject's scale and position. The left shows the original image, while the right simulates the subject being farther away and shifted to the right (the border indicates the image size).

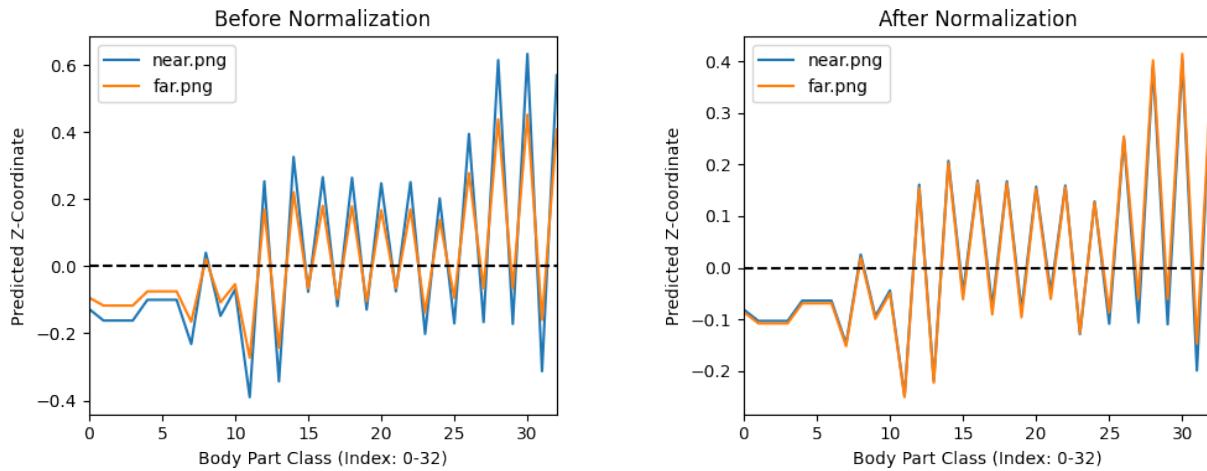


Figure 22. Plot of z-coordinate values for 33 keypoints, showing alignment after normalization.

6.2.4. Other Preprocessing

During development, we didn't limit ourselves to using the raw (33, 4) array as input. Instead, we experimented with various input structures, such as masking potentially irrelevant rows or columns, or adding new columns with precomputed data (e.g.,

angles between keypoints). These modifications were tested to observe their impact on model accuracy.

6.3. Output Structure

For the classification of four yoga postures, the model output is structured as follows:

$$\begin{aligned} & [[a_0, b_0], \\ & [a_1, b_1], \\ & [a_2, b_2], \\ & [a_3, b_3]] \end{aligned}$$

The first column (a) contains logits for each posture type. After applying the softmax function, this yields a probability distribution indicating the likelihood of each posture. The maximum value in this distribution (a_i) corresponds to the predicted posture. The second column (b) contains logits for the correctness of the corresponding posture. Applying a sigmoid function to the corresponding b_i returns the probability that the predicted posture is correct. A posture is considered correct if $\text{sigmoid}(b_i) > 0.5$.

6.4. Model Architecture Selection

6.4.1. Model Type Selection

As mentioned in [Section 2.4](#), we are using a deep learning model as the second-stage model. The primary reason for choosing deep learning over traditional machine learning algorithms, such as SVM, decision trees, or logistic regression, is the flexibility it offers. Deep learning models allow us to define the input/output structure, internal connections, and model complexity in a customizable way, making it a good fit for simultaneously classifying posture types and estimating posture correctness within a single model.

However, with the rapid development of deep learning, there are numerous model types to choose from, including MLP, CNN, GNN, RNN, and others. Below, we present a brief introduction to each type and explain why we chose or ruled out certain models:

- Multi-Layer Perceptron (MLP): A basic neural network architecture commonly used for structured or tabular data. Given that keypoint coordinates are structured and high-level, MLP is a strong candidate for this task.
- Convolutional Neural Network (CNN): Typically used for images, focusing on local patterns and spatial correlations. However, since keypoints lack a meaningful spatial order, CNNs are not ideal for this task.

- Graph Neural Network (GNN): Designed for graph-structured data, which does not align well with our coordinate-based input.
- Recurrent Neural Network (RNN): Useful for time-series data, such as sequential video frames, but not applicable to individual image frames (the current stage of our project).
- Transformer: Originally developed for NLP tasks, transformers have been extended to domains like vision. However, they may be overly complex and unnecessary for this task. Additionally, models like Vision Transformers are reported to be less data-efficient than CNNs [68].

In conclusion, we selected MLP as the most suitable model, as it is theoretically the best fit given the nature and volume of the input data. MLP is simple but sufficient for handling structured, high-level data, such as point coordinates.

6.4.2. Shared MLP

During the development of the model architecture, we came up with a variation of the standard MLP design. After researching online, we discovered that this design already exists and is known as Shared MLP, although it is not widely discussed. This concept was first introduced in PointNet [69], a deep learning model for 3D point cloud data.

6.4.2.1. Rationale for Shared MLP

In a traditional MLP, the input must be flattened into a 1D vector, which can cause a loss of semantic information. Our input is a 2D array of keypoints, where each point has (x, y, z) coordinates, similar to how each pixel in an image has R, G, B channels.

Flattening this data might reduce efficiency by disrupting the inherent structure of the input. While CNNs provide parameter sharing, they also assume spatial locality and local receptive fields, which are not relevant for keypoints, as the order of keypoints does not carry specific meaning. Therefore, we adopted Shared MLP, which retains parameter sharing without assuming spatial locality. This approach preserves the semantic meaning of the input while also reducing the number of parameters.

6.4.2.2. Technical Explanation

In [Figure 23](#), we compare the design of a traditional MLP (left) with that of a Shared MLP (right). In a traditional MLP, each neuron in one layer has a unique connection to all neurons in the next layer, forming a fully connected structure. In contrast, Shared MLP groups input neurons, with each group sharing the same set of parameters. Despite this grouping, there is still a fully connected structure within each group.

Upon further investigation, we realized that Shared MLP is still a special case of a CNN, which is also how PointNet implements it. By transposing our input array from (33, 4) to (4, 33), we treat it as a sequence of 33 points with 4 feature channels. Applying a 1D convolution with a kernel size and stride of 1 essentially replicates the behavior of Shared MLP.

Similar to CNNs, after applying Shared MLP in the shallow layers, we need to concatenate the outputs, either by flattening or using global average pooling, before passing them to a fully connected classification network.

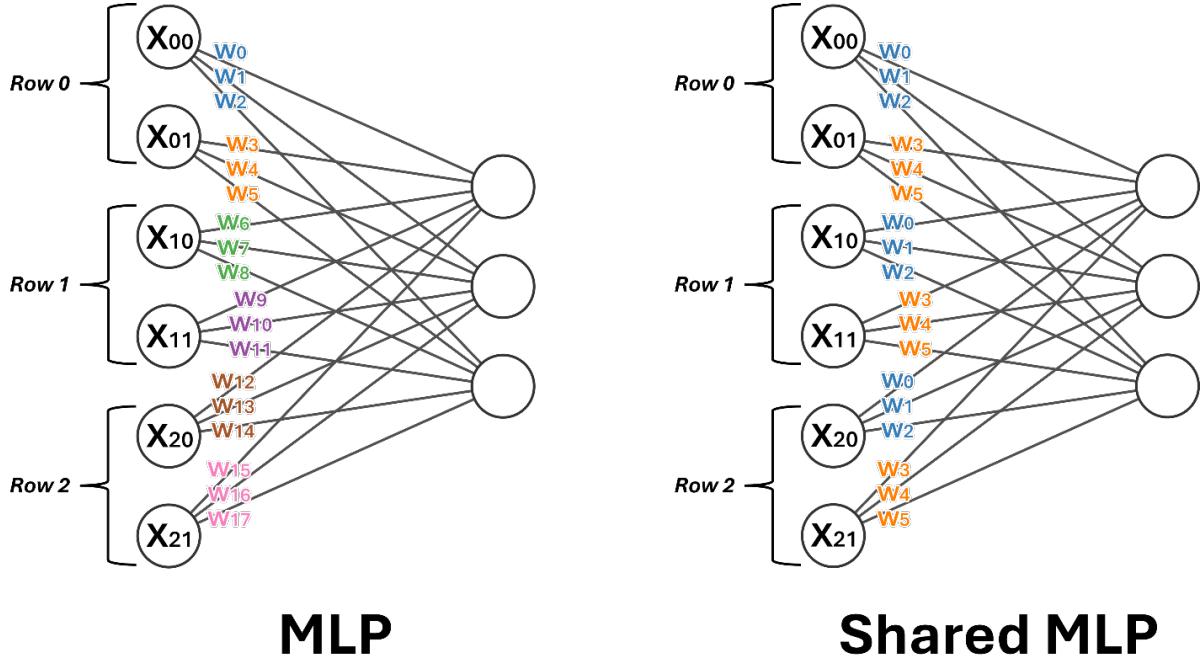


Figure 23. Structural differences between standard MLP and Shared MLP for a (3, 2) input array.

6.5. Training Configurations

Training was conducted using PyTorch, with the following configurations and hyperparameters.

6.5.1. Batch Size

We used a batch size of 32, a typical setting for MLP or CNN training [70].

6.5.2. Loss Function

Our model's loss is a combination of two parts: posture type loss and posture correctness loss.

Posture Type Loss

This is calculated using the first column of the model's output (see [Section 6.3](#)) and the one-hot encoded label of the true posture type, using Cross Entropy Loss, as shown in [Figure 24](#).

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = - \sum_{c=1}^C \log \frac{\exp(x_{n,c})}{\sum_{i=1}^C \exp(x_{n,i})} y_{n,c}$$

Figure 24. Formula for Cross Entropy Loss. [71]

Posture Correctness Loss

This is calculated using b_i , where i is the predicted posture type index (the row index of the output array) (see [Section 6.3](#)), along with the true correctness label (1 for correct postures, 0 for incorrect ones). The loss is computed using Binary Cross Entropy (with logits), as shown in [Figure 25](#).

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))]$$

Figure 25. Formula for Binary Cross Entropy Loss (with logits). [72]

6.5.3. Optimizer

We used Stochastic Gradient Descent (SGD) with momentum set to 0.9 for training [73]. This choice was based on prior experience from other courses, as it is a simple yet effective setup for most basic training tasks.

Weight decay was also enabled to help alleviate overfitting. The value of weight decay was not fixed but manually tuned throughout the training process, based on model performance.

6.5.4. Scheduler

We used ReduceLROnPlateau [74] as the learning rate scheduler. This is a common and intuitive choice, as it reduces the learning rate when the selected metric (in this case, validation loss) stops improving for a specified number of epochs.

The training stopping criterion was set to stop when the learning rate dropped below a certain threshold. At that point, further changes in the model's performance would likely be negligible due to the learning rate being too small.

6.6. Development Comparisons

6.6.1. MLP vs. Shared MLP

During development, we compared the standard MLP and Shared MLP, as shown in [Table 2](#). To ensure a straightforward and clear comparison, we included only the necessary layers in both models: Shared MLP used a convolutional layer for parameter sharing followed by fully connected layers, while the standard MLP consisted only of fully connected layers. Neither model included additional layers, such as dropout, to keep them simple.

| Metric | MLP | Shared MLP |
|--------------------------------------|--------|------------|
| No. of Parameters (including biases) | 4664 | 4560 |
| No. of Epochs | 526 | 407 |
| Training Loss (final epoch) | 0.1785 | 0.1753 |
| Training Accuracy (final epoch) | 0.9348 | 0.9387 |
| Validation Loss (final epoch) | 0.2927 | 0.2930 |
| Validation Accuracy (final epoch) | 0.9029 | 0.9185 |
| Test Loss | 0.2775 | 0.2374 |
| Test Accuracy | 0.9022 | 0.9239 |

Table 2. Comparison of metrics between MLP and Shared MLP models.

To make the comparison as fair as possible, we kept the total number of parameters between the two models as close as possible and maintained small model sizes. This was done to avoid overfitting, where larger models might "memorize" the training data, potentially leading to similar results for both models.

Regarding training and validation metrics, the differences between the two models were minimal. This is likely due to our stopping criteria (see [Section 6.5.4](#)). Since we did not impose an upper limit on the number of epochs, both models continued improving their validation loss and fitting to the validation set as long as there was progress, which led to similar performance on the training and validation metrics.

However, when looking at the number of epochs, the standard MLP required almost 30% more epochs to converge compared to Shared MLP, indicating a faster convergence rate for Shared MLP. More importantly, Shared MLP showed a 15% improvement in test loss and a 2% improvement in test accuracy over the standard MLP, demonstrating better generalization ability.

In conclusion, these observations suggest that the Shared MLP design, which we applied to handle point coordinate data, indeed offers an advantage over the standard MLP by preserving the input's semantic meaning, as we anticipated.

6.6.2. Input Masking vs. No Masking

As discussed in [Section 6.2.4](#), we considered masking potentially irrelevant rows or columns from the raw (33, 4) array in the first-stage model. For example, this could involve removing confidence values for each keypoint coordinate or keypoints from body parts that may not be relevant to yoga postures (e.g., "left_eye_outer" or "right_eye_inner"). In this comparison, we began by masking the column containing confidence values.

[Table 3](#) shows the performance metrics of two models: one trained with the raw input array and the other with the masked input array. All hyperparameters were kept identical between the two models. The results indicate that both models performed almost identically in terms of test loss and accuracy. This suggests that the confidence column does not provide significant information for the model's decision-making process. It also indicates that masking out certain irrelevant data does not significantly improve performance.

| | Without Masking | With Masking |
|-----------------------------------|-----------------|--------------|
| No. of Epochs | 582 | 596 |
| Training Loss (final epoch) | 0.1651 | 0.159 |
| Training Accuracy (final epoch) | 0.9341 | 0.9426 |
| Validation Loss (final epoch) | 0.2273 | 0.2266 |
| Validation Accuracy (final epoch) | 0.9211 | 0.9237 |
| Test Loss | 0.1959 | 0.1967 |
| Test Accuracy | 0.9221 | 0.9299 |

Table 3. Comparison of performance with and without masking the confidence column.

Our interpretation of this result is that the model likely adapts to irrelevant data by converging the associated weights toward values like zero, making them effectively neutral in influencing the logits. Based on this conclusion, we decided not to proceed

with masking keypoints from potentially irrelevant body parts, as we believe this would not result in significant improvements.

6.7. Final Results

6.7.1. Model Architecture and Hyperparameters

This model has approximately 170K parameters (including biases) and consists of four hidden layers, as shown in [Figure 26](#). The first two layers use the Shared MLP design, while the remaining layers are fully connected MLPs. It's important to note that our goal for this final version of the model was not to optimize parameter efficiency but to prioritize accuracy, as long as the model size remained reasonable. Specifically, we ensured the inference time was comparable to the first-stage model so that it would not become a bottleneck.

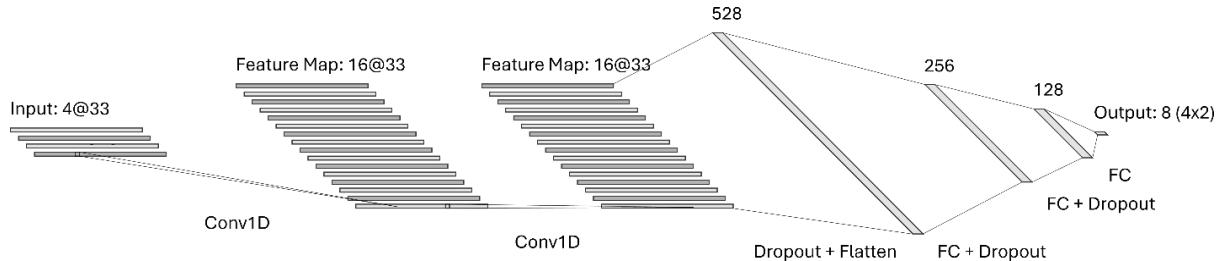


Figure 26. Schematic of the final model architecture.

The activation function used in this model is Leaky ReLU, which is a common choice due to its advantages over ReLU, such as avoiding the "dead ReLU" problem while retaining the benefits of ReLU.

Dropout layers were added after the last convolutional (Shared MLP) layer and after each hidden fully connected layer, with a dropout probability of 0.1 to help alleviate overfitting.

Below is a table of the final hyperparameters (some of which were previously mentioned in [Section 6.5](#), but here we provide the specific values used in the final setup):

| | |
|-----------------------|--|
| Batch Size | 32 |
| Optimizer | SGD [73], momentum 0.9 |
| Initial Learning Rate | 0.001 |
| Weight Decay | 0.00001 |
| Scheduler | ReduceLROnPlateau, factor=0.5, patience=30 |
| Stopping Criteria | learning rate $\leq 1e-5$ |

6.7.2. Training Log

The following two plots show the loss and accuracy on the training and validation sets during the training of the final model. The convergence curves indicate that the model fits the training set reasonably well without overfitting, completing training in a total of 695 epochs.

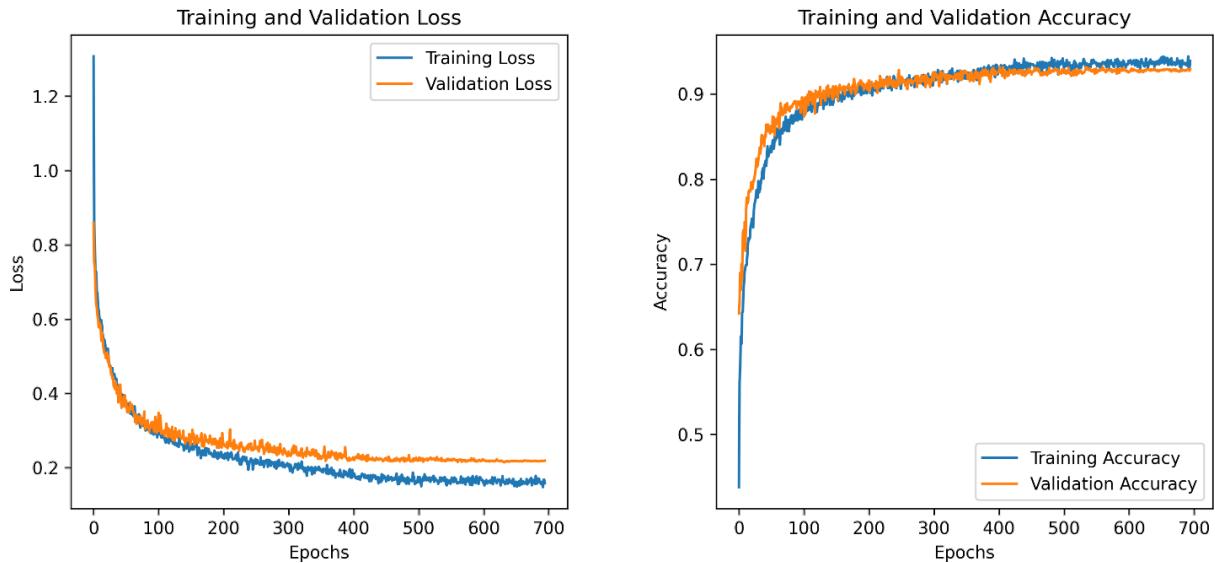


Figure 27. Loss and accuracy curves for the training and validation sets throughout the training process of the final model.

The final metrics for the training and validation sets are as follows:

| | Loss | Accuracy |
|----------------|--------|----------|
| Training Set | 0.1577 | 0.9389 |
| Validation Set | 0.2186 | 0.9289 |

6.7.3. Evaluation

The model was evaluated on a test set containing 1,156 samples (see [Section 6.1.2](#)), which it had never seen before.

6.7.3.1. Loss & Accuracy

| | Loss | Accuracy |
|----------|--------|----------|
| Test Set | 0.1906 | 0.9325 |

The model achieved 93% accuracy on the test set, meaning that it correctly classified the posture and its correctness for 93% of the frames. At 30 frames per second, this

results in approximately two incorrect predictions per second, which is a satisfactory performance.

6.7.3.2. Confusion Matrices

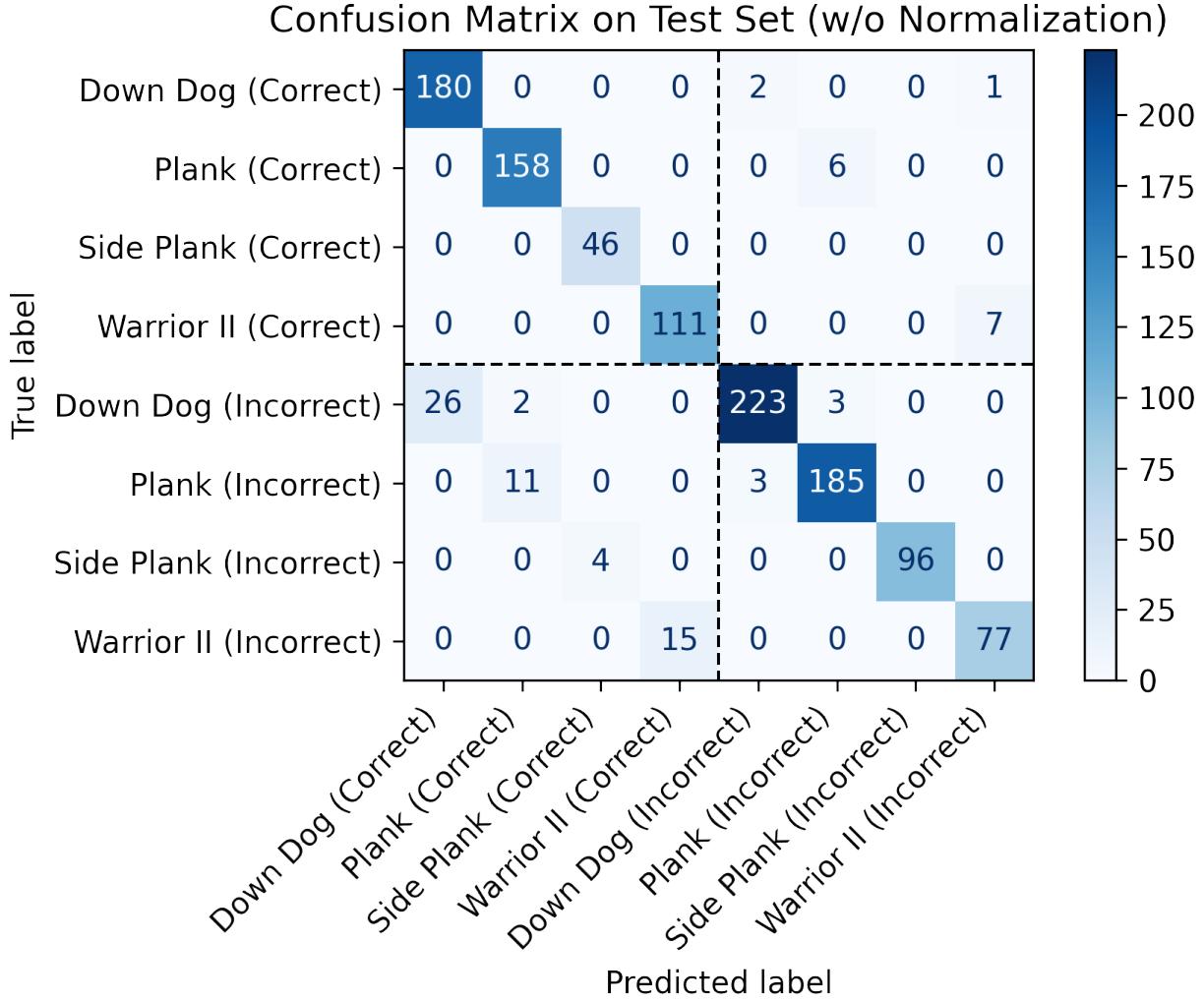


Figure 28. Non-normalized confusion matrix showing posture type and correctness classification results of the final model on the test dataset containing 1,156 samples.

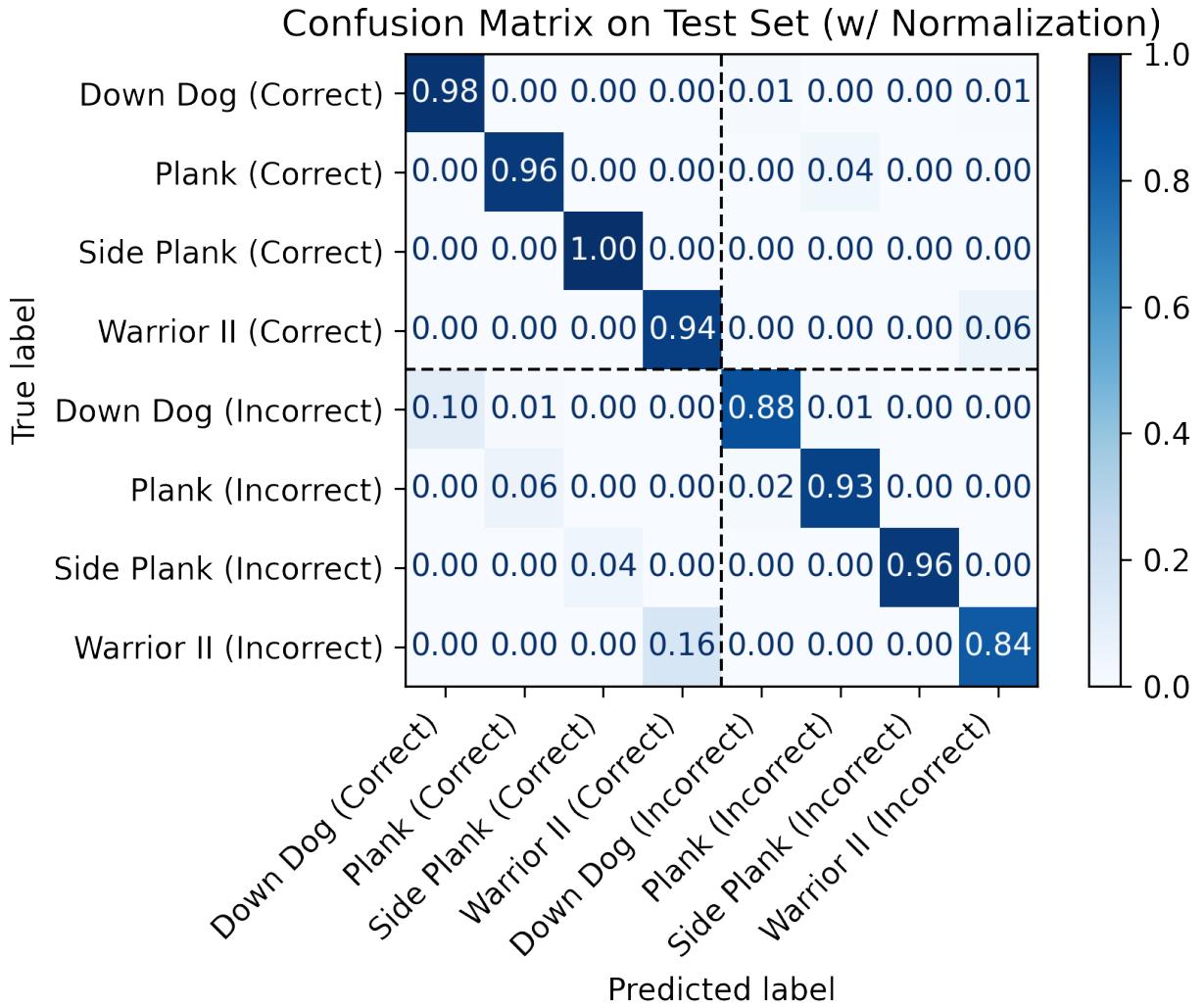


Figure 29. Row-wise normalized confusion matrix presenting the same classification results as in [Figure 28](#), allowing for clearer comparison of relative performance across classes, despite the imbalance in the test dataset.

The confusion matrix shows that the model performs well in distinguishing between different posture types. Only 9 out of thousands of samples were misclassified in terms of posture type (those that don't lie on any diagonal of the sub 4x4 matrices). Indeed, the difference between different posture types is much greater than the difference between correct and incorrect postures within the same type.

The majority of incorrect predictions occurred when classifying incorrect Down Dog and Warrior II postures as correct, with 26 and 15 misclassifications, respectively. This is likely due to the ambiguity of certain samples, where the posture is nearly correct but not entirely so. Some of these samples were difficult to classify even for humans during dataset construction (see [Section 5.2.6](#)). Despite this, the model's overall accuracy remains satisfactory.

6.8. Flexibility for Future Extension

As mentioned in [Section 2.1](#), we initially planned to extend the posture correctness classification from a binary (correct/incorrect) format to a multi-class format that identifies specific types of mistakes (e.g., errors in hand or leg positioning). However, due to changes in the project roadmap, this feature was not implemented.

Nonetheless, our second-stage model was designed with future extensibility in mind. Specifically, the model's output structure can be expanded from the current shape of $(4, 2)$ (see [Section 6.3](#)) to a generalized (N, M) format. This would support classification across N posture types and $M - 2$ distinct mistake categories (while preserving one row element for the posture type probability and another for the "no mistake" class).

To retrieve the predicted mistake category, apply the softmax function to each row of the output array (excluding the first row, which corresponds to posture type) to produce probability distributions over potential mistake classes.

7. Model Integration

(Completed in the second semester)

After successfully training the second-stage model (see [Section 6](#)), we have gathered the core components of our system (see [Section 2.4](#)). By unifying them into a pipeline, our system can now take a yoga pose image as input and output the posture type, its correctness (correct or incorrect), as well as additional details such as inference time and prediction confidence.

In this section, we focus on improving the integration of the two models to build a fully functional posture correction system. While the early-stage implementation in the first semester primarily processed independent images, we now extend the system to support real-time video processing. This enhancement aligns with our core objective: providing instant feedback to users so they can correct their posture during practice rather than reviewing recordings afterward.

To achieve real-time processing on video streams, we are applying various optimizations and specialized techniques beyond our independent-image-based implementation. The following sub-sections detail these improvements, version by version, and the methods used to achieve efficient real-time posture correction from continuous video input.

7.1. Metrics

For real-time processing, delay is a critical factor. Whether the system qualifies as real-time depends entirely on the time taken to process each frame. Beyond delay, ensuring a smooth and lag-free video stream is essential for a good user experience.

Although posture correction is not highly time-sensitive, excessive delay can lead to poor user experience, as outdated frames and predictions feel disconnected.

Minimizing delay ensures instant feedback, making the system more responsive and natural to use. Additionally, optimizing for low delay demonstrates computational efficiency and improves adaptability, allowing the system to run smoothly even on lower-specification devices.

A widely accepted standard in the IT industry is 30 FPS (Frames Per Second), which is commonly used in video capture devices such as webcams. To achieve a fluent and natural experience, we set this as our target. Below, we define the key terms used in this section:

1. **Delay:** The time measured from the moment a frame is captured by the webcam until it is fully processed and displayed to the user. This can also be interpreted as the "inference time" or "processing time" per frame.
2. **FPS (Frames Per Second):** FPS measures the actual number of frames displayed to the user within a one-second window. While influenced by the inference time per frame, it is not always inversely proportional to the frame delay; for example, even if each frame is processed instantly, FPS cannot exceed the camera's capture rate, as the webcam itself introduces an internal delay between frames. Thus, FPS serves as a more comprehensive and direct metric for evaluating real-time performance and user experience.

7.2. Pre-Evaluation

Before implementing real-time video stream processing, we conducted an initial evaluation using our independent-image-based implementation. This test was performed with randomly selected images from our training dataset. To isolate inference time from other potential overheads, images were preloaded into memory, and only the processing time was measured.

For a total of 20 images, the system processed them in around 0.6 seconds, resulting in an average latency of around 0.03 seconds per image. This suggests that, under ideal conditions with minimal system overhead, our system could potentially process approximately 33 frames per second, making it feasible to reach our 30 FPS target for real-time performance.

```

1 LOOP FOREVER:
2     image_ndarray ← get from memory
3
4     // Stage 1: Extract keypoints using BlazePose model
5     keypoints ← BlazePose(image_ndarray)
6
7     // Stage 2: Predict posture and correctness using ShareMLP model
8     (posture, correctness) ← ShareMLP(keypoints)
9
10    // Overlay predicted information on the image
11    image_ndarray.draw_skeletons(keypoints)
12
13    // Display the result to the user
14    show(image_ndarray)
15    PRINT posture, correctness

```

Figure 30. Pseudocode representation of the system pipeline for pre-evaluation of system performance on offline, independent images.

7.3. Version 1 – Naïve Sequential Execution

As a starting point, we implemented the simplest and most straightforward approach: a sequential execution pipeline, where each component runs in order without parallelization. This serves as a baseline implementation. The structure remains the same as in Pre-Evaluation, except that the input now consists of real-time frames from the webcam instead of preloaded images stored in memory.

Under this naïve implementation, the results were disappointing. The system achieved only around 13 FPS, while the frame delay remained at approximately 0.03 seconds per frame, the same as in Pre-Evaluation.

Upon analysis, we identified a key issue. While the frame delay matched our Pre-Evaluation results, the FPS was nearly half of the expected rate. We believe the main cause is I/O waiting time during frame capture.

In this sequential design, the system requests a new frame only after the previous one has been fully processed and displayed. This blocking behavior explains why the frame delay remains unchanged, but the FPS drops significantly compared to the Pre-Evaluation case.

```

1 LOOP FOREVER:
2     frame_ndarray ← capture from webcam
3
4     // Stage 1: Extract keypoints using BlazePose model
5     keypoints ← BlazePose(frame_ndarray)
6
7     // Stage 2: Predict posture and correctness using ShareMLP model
8     (posture, correctness) ← ShareMLP(keypoints)
9
10    // Overlay predicted information on the frame
11    frame_ndarray.draw_skeletons(keypoints)
12
13    // Display the result to the user
14    show(frame_ndarray)
15    PRINT posture, correctness

```

Figure 31. Pseudocode representation of the system pipeline for our Version 1 solution.

It is almost identical to [Figure 30](#), except for a change in the input source.

7.4. Version 2 – Multithreading and Frame Dropping

To improve performance over Version 1, we introduced multithreading along with frame dropping. We modularized the system into three components: frame capturing, posture classification, and frame processing (i.e., drawing on the frame and displaying output). The camera capturing and posture classification modules were moved to separate threads, while the main thread handled only frame processing and display.

In this design, one thread continuously captures new frames, while another processes frames to classify postures. These threads communicate with the main thread through queues. However, this producer-consumer model introduces a potential issue: if the posture classification thread cannot keep up with the frame capture rate, the queue will keep growing, causing infinite delay as the program runs.

To prevent this, we implemented frame dropping by setting a maximum queue size (i.e., limiting the number of pending frames). When the queue is full, newly captured frames are discarded to ensure that the displayed output remains real-time, even if frame capture speed exceeds processing speed.

```

1 FUNC capture_frames(frame_queue):
2     LOOP FOREVER:
3         frame ← capture_from_webcam()
4         IF NOT frame_queue.is_full():
5             frame_queue.put(frame)
6
7 FUNC process_posture(frame_queue, result_queue):
8     LOOP FOREVER:
9         frame ← frame_queue.get()
10
11        // Stage 1: Extract keypoints using BlazePose model
12        keypoints ← extract_keypoints(frame)
13
14        // Stage 2: Predict posture and correctness using ShareMLP model
15        (posture, correctness) ← evaluate_posture(keypoints)
16
17        result_queue.put([frame, keypoints, posture, correctness])
18
19 frame_queue ← Queue(size=max_size)
20 result_queue ← Queue()
21
22 thread_1 ← Thread(capture_frames(frame_queue))
23 thread_2 ← Thread(process_posture(frame_queue, result_queue))
24
25 LOOP FOREVER: // Main thread
26     frame, keypoints, posture, correctness ← result_queue.get()
27
28     // Overlay predicted information on the frame
29     frame.draw_skeletons(keypoints)
30
31     // Display the result to the user
32     show(frame)
33     PRINT posture, correctness

```

Figure 32. Pseudocode representation of the system pipeline for our Version 2 solution, which introduces multithreading with two additional threads alongside the main thread.

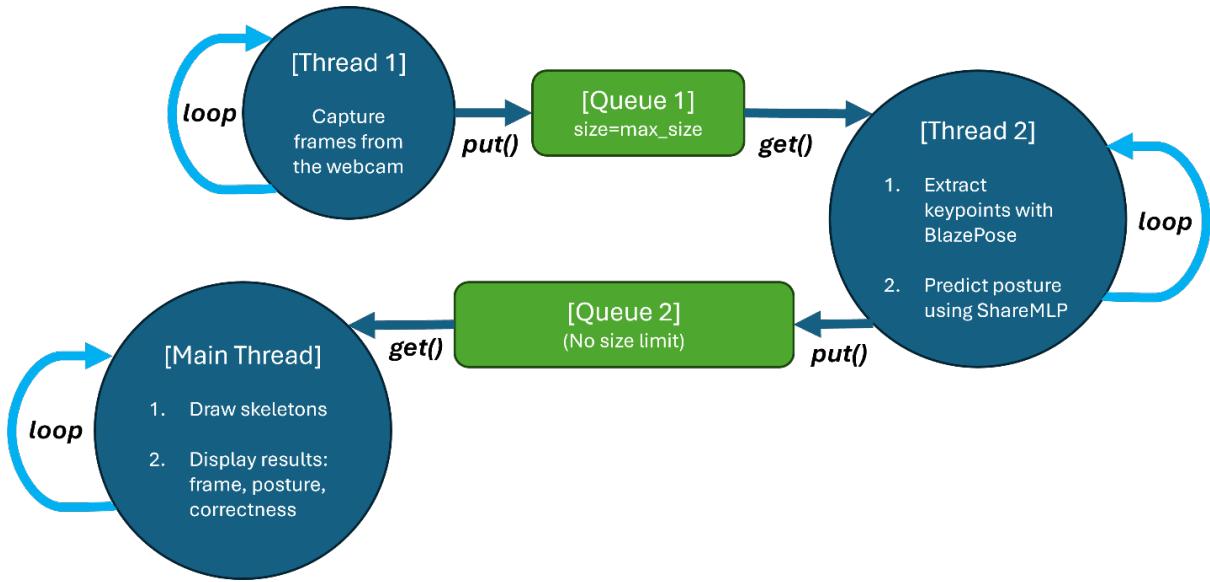


Figure 33. Flowchart illustrating the pseudocode in [Figure 32](#) of our Version 2 solution.

7.4.1. Multithreading in Python

CPython's Global Interpreter Lock (GIL) generally prevents true multi-threading for pure Python code, as it allows only one thread to execute Python bytecode at a time [75]. However, in our case, multithreading still provides significant benefits:

1. Reducing I/O blocking: In Version 1, frame capture was a blocking operation. With multithreading, we offload this to a separate thread, ensuring that the main thread is not stalled by I/O waits.
2. Utilizing more computational resources: Although the GIL prevents pure Python code from running in parallel, it does not apply to underlying C/C++-based libraries like NumPy and PyTorch. Since our posture classification relies on PyTorch, it can still leverage multiple CPU cores effectively.

Multiprocessing was not chosen because it has significantly higher overhead due to resource allocation and inter-process communication. Our program is delay-sensitive and frequently passes data between modules, making multithreading a more efficient choice.

7.4.2. Why Only Two Threads?

We used two additional threads but did not create a third thread for frame post-processing. This is because once a frame is processed, it is immediately displayed. The display function itself is extremely fast, so keeping frame post-processing and display

in the main thread eliminates unnecessary thread communication overhead, improving overall efficiency.

7.4.3. Version 2 Performance Results

With this multithreading design, we achieved around 27 FPS, a significant improvement over the naïve Version 1. The frame delay increased slightly to about 0.04 seconds (compared to ~0.03 seconds in Pre-Evaluation and Version 1), likely due to synchronization overhead from inter-thread communication. This shows that FPS is indeed not always inversely proportional to frame delay, as discussed in [Section 7.1](#). However, the slight delay increase has minimal impact, as overall FPS is the main factor affecting user experience and visual smoothness.

7.5. About Downsampling Technique

During a project meeting with our supervisor, downsampling was suggested as a potential technique to reduce per-frame processing time for real-time applications. With modern webcams supporting resolutions as high as 4K, processing frames at full resolution can significantly increase computational load, leading to higher delay.

However, upon deeper analysis of the MediaPipe graph configuration for BlazePose (our first-stage model) and referencing official documentation [66], we found that the model already scales down input frames internally to $256 \times 256 \times 3$ before processing.

Additionally, for our Shared MLP-based second-stage model, the input array size remains constant regardless of frame resolution. Since downsampling would not further improve performance, no additional downsampling is required for our system.

7.6. Post-Integration System Performance

With multithreading, frame dropping, and redundancy optimization, the two models have been integrated to enable real-time posture correction from live webcam frames. The system achieves approximately 27 FPS with a per-frame processing delay of around 0.04 seconds, a significant improvement over the initial sequential implementation.

8. Result Stabilization

(Completed in the second semester)

In [Section 7](#), we integrated the two models to build a standalone real-time yoga posture correction system, optimizing for performance metrics such as FPS. However, accuracy is equally critical and has been a key focus throughout our model experiments and development. In this section, we address accuracy-related issues and propose methods to improve result stability.

8.1. Current Accuracy

The first-stage model, BlazePose, is a powerful state-of-the-art pre-trained model that can detect the user's posture in nearly all cases, as long as the input image is clear to a human viewer. In the dataset we constructed in [Section 5](#), only 4 out of 3,854 images failed to be detected, resulting in an accuracy of approximately 98%.

The second-stage model, which we trained in [Section 6](#), was evaluated on an unseen test set and achieved around 93% accuracy (see [Section 6.7.3.1](#)).

By combining the probabilities of both models ($0.98 \times 0.93 \approx 0.91$), we infer that the entire system achieves an overall accuracy of around 91%. This was further validated through demonstrations during the first semester's presentation using sample images.

8.2. Current Issues

8.2.1. Accuracy Is Not High Enough

An overall 91% accuracy means that, on average, 9 out of every 100 frames may have incorrect posture classification or correctness assessment. As Dr. Chui pointed out during the first semester's presentation, there is still room for improvement, and steps should be taken to further enhance accuracy.

8.2.2. Unstable Predictions

Although 91% accuracy may not seem low, it introduces instability that affects future development. Given our target FPS of 30 (see [Section 7.1](#)), this implies that in each second, approximately 3 frames could be misclassified. As a result, the displayed posture type and correctness may frequently fluctuate, creating a flashing effect where predictions change erratically within seconds.

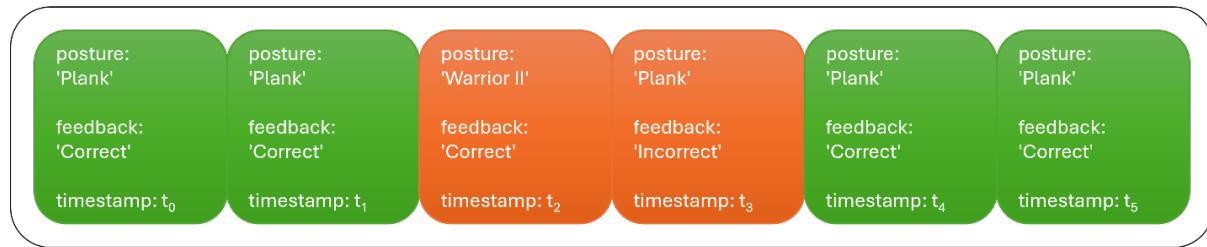
This instability not only negatively impacts user experience but also complicates the implementation of features such as a timer to measure posture duration. Even a single incorrect prediction could cause the timer to reset mistakenly, leading to incorrect assessments of how long a user holds a posture.

8.3. Solution: Majority Voting

To stabilize predictions and improve accuracy, we implemented a majority voting mechanism. Instead of determining posture correctness based solely on the latest frame, the system analyzes a recent time window and selects the most frequent prediction as the final output.

In our implementation, we use a 1-second time window, meaning that 30 frames (for 30 FPS) contribute to the decision. The majority vote is determined by selecting the posture type and correctness that appears most frequently within this window.

Majority: ('Plank', 'Correct') – 4 out of 6



List of predictions for the most recent frames (size = 6)

Figure 34. Sample case of majority voting for final prediction result, with a window size of 6 frames.

8.4. Performance Improvement

With a frame rate of 30 FPS, majority voting is applied over 30 frames to determine a final result. For simplicity, we define a valid majority as at least 16 frames agreeing on the same classification for both posture type and correctness.

As discussed in [Section 8.1](#), the probability that a single frame is successfully processed by the first-stage model and correctly classified by the second-stage model (in terms of both posture type and correctness) is 0.91.

Based on this, we can calculate the probability that the final system output, after applying majority voting, is correct as follows:

$$P(\text{majority voting result is correct}) = P(k \geq 16)$$

$$= \sum_{k=16}^{30} \binom{30}{k} (0.91)^k (1 - 0.91)^{30-k}$$
$$\approx 1.000$$

where k is the number of frames with completely correct results within the 30-frame voting window.

Although our theoretical calculation defines a majority as at least 16 fully correct frames for simplicity, in practice, the system can still produce the correct result even when fewer frames meet that standard. For example, in one case within the 30-frame voting window:

- 12 frames were entirely correct (both posture and correctness),
- 6 had the correct posture but incorrect correctness,
- 6 had the correct correctness but incorrect posture,
- and 6 were completely incorrect.

Despite only 12 frames being fully accurate, the system correctly output the final result because the largest group of consistent predictions still pointed to the correct classification.

9. User Interface

(Completed in the second semester)

As mentioned in the introduction (see [Section 2.1](#)), the primary focus of our project is to implement core functionality through a working prototype, rather than to develop a fully-featured application with a polished user interface. However, to better visualize the system's results and facilitate understanding during demonstrations, we implemented a basic on-screen interface over the video output. This improves both user experience and system accessibility beyond simple terminal output.



Figure 35. A screenshot of the user interface. The upper-right corner displays the current FPS; the central green area shows the live camera input; and the lower black bar presents system feedback, including predicted posture type, correctness, and other relevant information.

9.1. Frames Per Second (FPS)

FPS is a simple yet useful metric that indicates the real-time performance of the system. It helps users confirm that the system is functioning smoothly, and provides developers with valuable debugging information. FPS is displayed in the upper-right corner in gray text with a black outline (see [Figure 35](#)), ensuring readability even against similarly colored backgrounds.

9.2. Bottom Output Panel

To avoid overlaying prediction text directly on the original video feed, we added an extended black margin at the bottom of the output frame. This area displays up to four types of information:

1. Predicted Posture Type

Displayed on the left side, this shows the yoga posture detected by the system, along with a confidence score (range: [0, 1]) in brackets.

2. Predicted Feedback

Displayed on the right side, this indicates whether the posture is correct or incorrect, again with a confidence score (the probability that the posture is correct). To improve usability, the feedback text is green if the posture is correct and red if incorrect, allowing users to quickly understand their performance.

3. Stopwatch

Positioned at the center, this timer tracks how long the user maintains a correct posture. It counts up continuously while the posture is correct, pauses when it becomes incorrect, and resets when the detected posture type changes.

4. Out-of-Frame Warning

If one or more keypoints fall outside the camera's visible area, a warning is shown. Since incomplete keypoint information may affect prediction accuracy, this warning replaces all other bottom-panel content to prevent displaying unreliable feedback. It reminds users to keep their entire body within the camera frame for proper analysis.

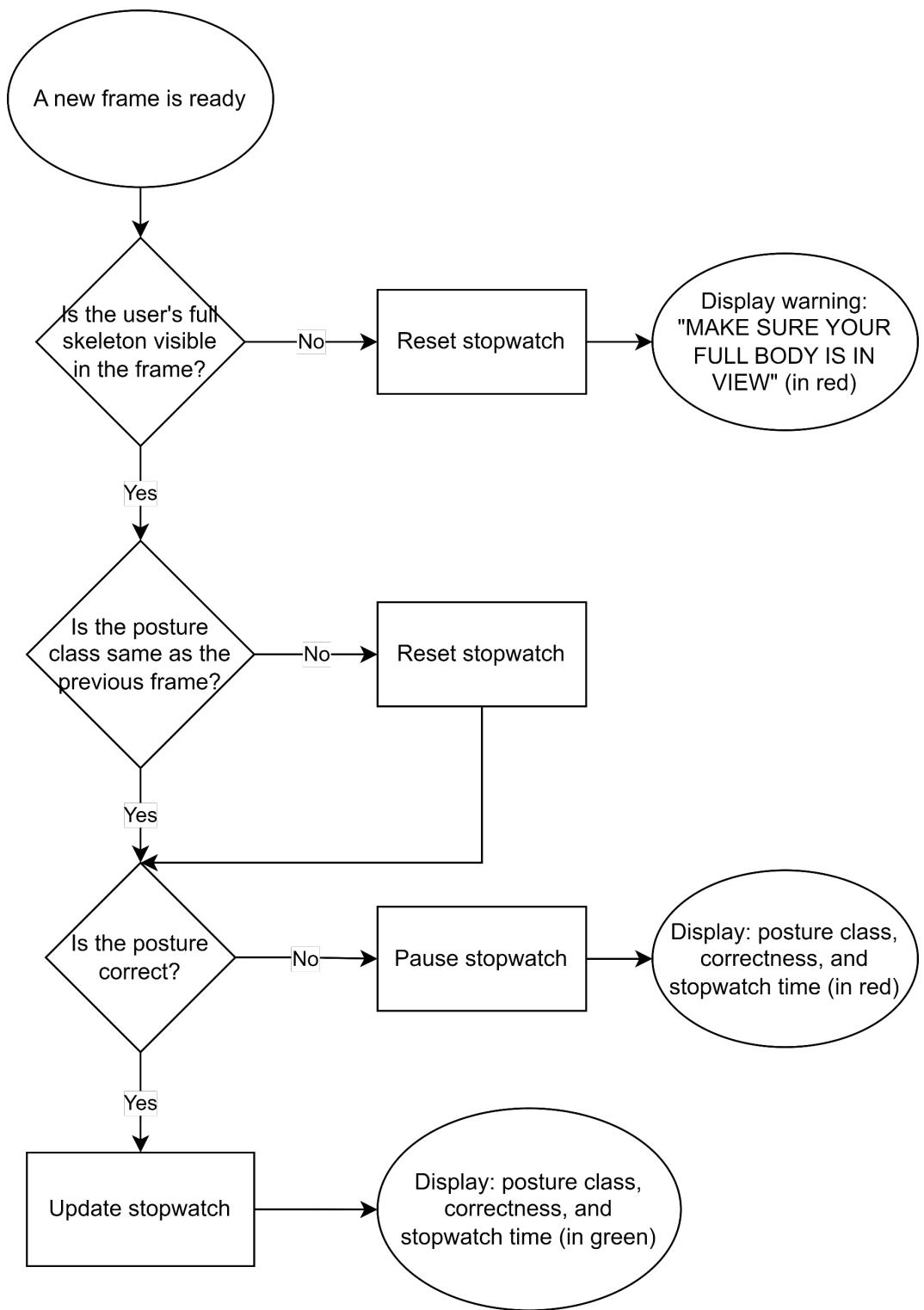
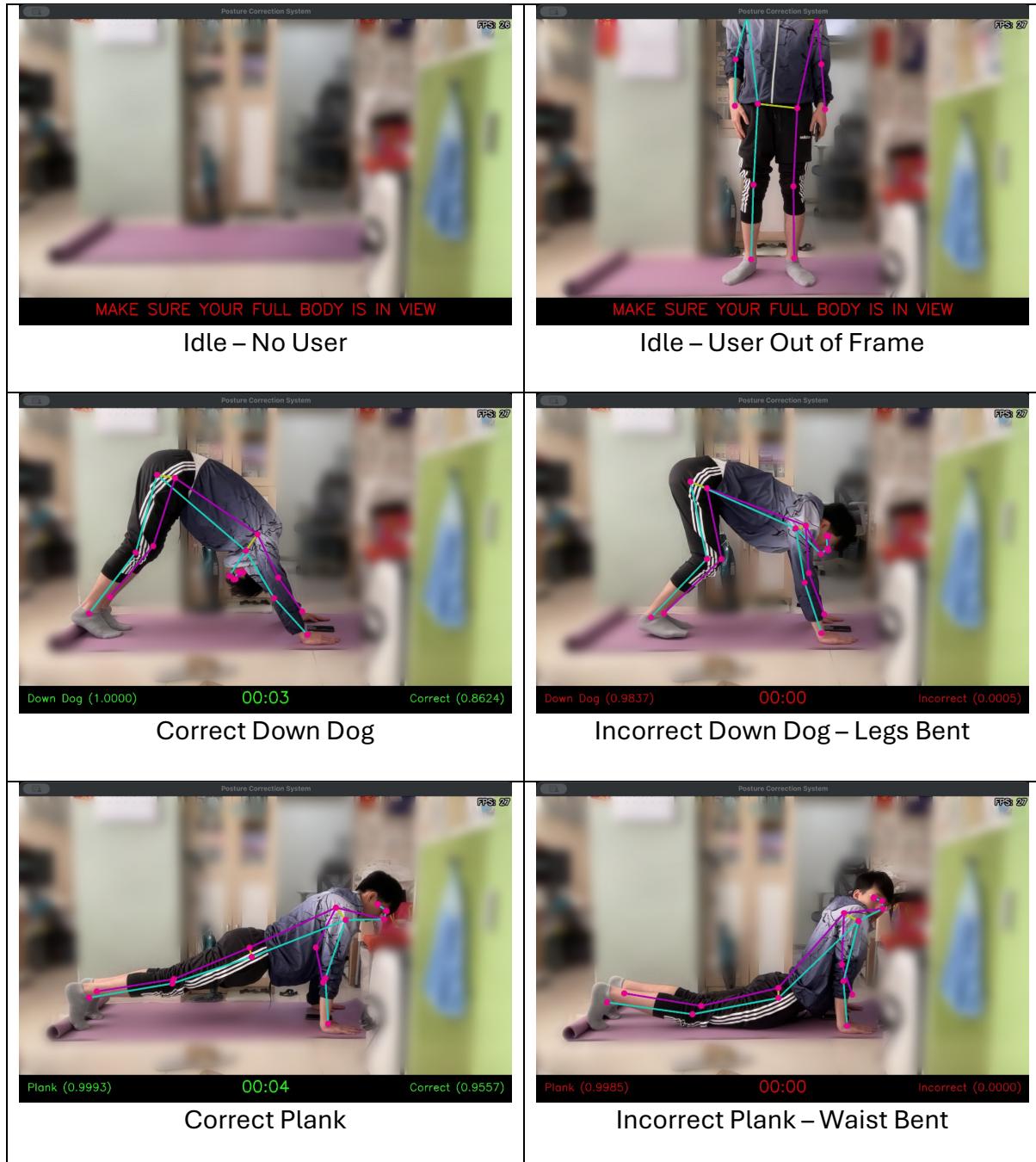


Figure 36. A flowchart showing the decision-making process for what to display in the extended bottom panel.

9.3. Example Screenshots

Below are ten example screenshots of the system interface, including idle states and both correct and incorrect executions for each of the four supported postures: Down Dog, Plank, Side Plank, and Warrior II. Backgrounds have been blurred in post-processing to preserve privacy and highlight the subject.





10. Practical Evaluation

(Completed in the second semester)

We have completed the development of the yoga posture correction system using the two-stage approach. This section presents the performance evaluation of the system under practical usage conditions, using ourselves as the subjects, rather than the purely theoretical evaluation based on the static test dataset conducted earlier in [Section 6.7.3](#).

10.1. Evaluation Methodology

Since our system processes real-time image streams and includes accuracy optimization via majority voting over time, traditional discrete metrics like per-frame accuracy are not ideal for assessing real-world performance.

Instead, we adopt a qualitative evaluation approach, inspired by a method presented in a paper we studied previously [15], with slight modifications to suit our system. For

each of the four supported yoga postures (Down Dog, Plank, Side Plank, Warrior II), we performed:

- 5 correct executions
- 5 incorrect executions

This results in a total of 40 trials. Between each attempt, we fully stood up and moved out of frame to reset the system's prediction state.

Each posture was held for approximately 5 seconds, and a prediction was considered correct only if the system consistently produced the correct result for at least 4 out of those 5 seconds. This ensures that the majority voting mechanism is effectively contributing to the prediction.

We define the system performance for each pose as:

$$\text{Performance} = \frac{\text{\# of correct system outputs}}{5}$$

10.2. Evaluation Results

| Posture | Correct Detection Rate | Incorrect Detection Rate |
|------------|------------------------|--------------------------|
| Down Dog | 0.8 | 1.0 |
| Plank | 1.0 | 1.0 |
| Side Plank | 1.0 | 1.0 |
| Warrior II | 1.0 | 1.0 |

Out of 40 total trials, only 1 incorrect classification was observed. In that case, the user was performing a correct Down Dog pose (intended to resemble an inverted "V" with straight arms and legs), but the system incorrectly classified it as incorrect.

Upon review, we suspect this misclassification was due to the looser-fitting sports jacket worn during the trial. Unlike tight-fitting yoga attire, the jacket may have obscured body contours, causing the first-stage model to inaccurately detect keypoints, leading to misclassification in the second stage.

10.3. Discussion

Overall, the system demonstrates strong practical performance in identifying both the type and correctness of yoga postures. The majority voting mechanism helps maintain consistent predictions while the user holds a pose. This consistency also allows the stopwatch function to operate accurately, as stable predictions are necessary for meaningful time tracking.

11. Limitations and Challenges

11.1. Limited Dataset Size and Coverage

Currently, the system supports only four yoga postures: Down Dog, Plank, Side Plank, and Warrior II. These represent a small subset of the full range of yoga poses practiced in real-life scenarios. Our dataset was constructed primarily from publicly available sources, including existing datasets and YouTube videos. Due to our limited experience in yoga, we were unable to produce original, first-hand training data through professional demonstrations.

Moreover, the lack of high-quality, publicly available, dedicated yoga pose datasets posed a significant challenge. Data collection and manual labeling were particularly time-consuming. Constructing a usable dataset even for just four postures consumed a considerable portion of the first semester. While we had considered expanding the dataset to include more poses or a wider variety of incorrect examples and classes, time and resource constraints made it difficult to pursue further.

11.2. Absence of Experienced Yoga Guidance

The lack of involvement from an experienced yoga practitioner affected both the amount of training data we could collect and the evaluation process. In our practical testing (see [Section 10](#)), we personally performed each pose multiple times to assess the system. However, poses such as Down Dog and Side Plank were physically demanding, and despite our best efforts, our execution may not fully reflect standard posture. As a result, the system was tested and refined primarily from a technical perspective, without direct feedback from actual yoga practitioners.

12. Conclusion

In the main section of this project, we developed a real-time posture correction system for yoga practice using a two-stage approach. The first stage employs BlazePose, a pretrained pose estimation model, to extract body keypoints from the user. The second stage uses a Shared MLP classification model, trained on our self-constructed dataset, to identify one of four supported yoga postures: Down Dog, Plank, Side Plank, and Warrior II, and to determine whether the posture is performed correctly.

The system runs seamlessly on any standard laptop equipped with a webcam. With a single click, users can start the system, which captures live video, performs inference at approximately 30 FPS, and overlays real-time feedback on the visual output. This allows users to receive instant guidance while practicing yoga.

In addition to the system itself, we compiled a dataset of over 3,800 yoga posture images collected from online sources. The dataset includes both correct and incorrect examples of the supported poses and was essential in training the second-stage model. This resource also helps address the scarcity of publicly available yoga-specific datasets.

Overall, the system demonstrates strong practical performance, with high accuracy and real-time capability. Although there are still limitations, such as the dataset's scope and the lack of domain expert involvement, the project serves as a promising prototype for real-time yoga posture assessment.

13. Side Research: Language Model-Based Approach

(Completed in the second semester)

Motivation for an Alternative Approach

Throughout this paper, we have adhered to a two-stage framework combining keypoint extraction and classification for posture correction. This approach is effective and relatively simple, but it involves significant implementation effort, particularly in data collection for training. Additionally, the models and methods we use, while reliable, are not the latest state-of-the-art advancements in AI.

Vision-Language Models

In this section, we explore an alternative approach using Vision-Language Models (VLMs) [76]. With the rise of Large Language Models (LLMs) dominating the AI field, these models are expanding beyond text-based applications to process various data types, including images. Given this rapid advancement, we investigate whether VLMs are powerful enough to generalize across image-based tasks, including yoga posture classification.

While our current method is computationally more efficient, VLMs offer potential advantages, particularly by eliminating the need for large, manually labeled datasets. If VLMs can effectively classify images using only prompt engineering, they could significantly lower the barrier for developers. Therefore, we believe it is worth investigating whether a VLM-based approach is feasible for our task.

Focus on Local Deployment

Although cloud-based VLM APIs provide access to high-performance models, our research focuses on locally deployed models. This aligns with our yoga posture correction system, which is designed to be fully offline, with all computations performed locally. Keeping the models on-device ensures a fair comparison with our previous approach and avoids external dependencies. Additionally, local deployment enhances reproducibility, as cloud APIs may become unavailable when providers update or discontinue models. It also bypasses regional restrictions, which can affect model access in Hong Kong or China.

Models Evaluated

This section presents our research on three state-of-the-art VLMs:

- Qwen2.5-VL
- DeepSeek-VL
- Llama 3.2 Vision

The models are introduced in the order we experimented with them.

13.1. Related Research

13.1.1. Vision-Language Models

The foundational paper on Vision-Language Models (VLMs), "An Introduction to Vision-Language Modeling", was published in 2024 [77]. This highlights that VLMs are still an emerging technique as of early 2025, remaining in the early stages of development. Given its novelty, we found no existing research directly related to our specific experimental focus. Most available papers primarily discuss new model architectures rather than practical applications like image classification. Therefore, we are conducting our own hands-on evaluation to explore the feasibility of using VLMs for our task.

13.1.2. Prompting Strategies in VLMs

Another crucial aspect of working with VLMs is prompt design. Due to the variation in tasks and model architectures, no single prompt format has been universally recognized as the best for research applications. According to "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications" [78], general prompt engineering techniques can be categorized into approaches such as "New Tasks Without Extensive Training" and "Reasoning and Logic," among other more advanced methods. Within "New Tasks Without Extensive Training," prompts can be further classified into "Zero-Shot Prompting" and "Few-Shot Prompting."

These two strategies were popularized in the paper "Language Models are Few-Shot Learners" [79]. In Zero-Shot Prompting, the model is given a task description in natural language without any prior fine-tuning or examples at inference time. In Few-Shot Prompting, a few examples of the task are included in the prompt to help the model infer the pattern before solving the actual target task.

Below is a simple example of a text-based sentiment classification task to illustrate the difference:

| Zero-Shot Prompting | Few-Shot Prompting |
|--|--|
| Classify the sentiment of the following sentence as "positive" or "negative": "I absolutely love this product. It works great!" | Classify the sentiment of the following sentences as "positive" or "negative". Example 1: Sentence: "The movie was boring and too long." Sentiment: negative Example 2: Sentence: "I had a wonderful experience at the restaurant." Sentiment: positive Now you try: Sentence: "I absolutely love this product. It works great!" Sentiment: |

For image-based tasks using VLMs, few-shot prompting would require multiple image inputs as demonstrations. In our case, classifying yoga posture into four types would mean providing at least four labeled posture images along with the target image, making each prompt contain five images. This is highly inefficient and token-intensive, especially when processing time is one of the concerns.

Therefore, in our experiments, we adopted the Zero-Shot Prompting strategy. It is a simpler, more resource-efficient, and commonly used way to submit a task to the language model.

13.2. Deployment

13.2.1. Hardware Environment

We tested VLM models on two local environments:

- CUDA environment: NVIDIA GPU with 12 GiB VRAM.
- Unified memory environment: Apple Silicon CPU with 32 GiB RAM.

Each model was tested on the most suitable hardware setup, as detailed in the following sections.

13.2.2. Software Environment

- Qwen2.5-VL and DeepSeek-VL were deployed using their raw PyTorch implementations.

- Llama 3.2 Vision was deployed using Ollama GGUF (GPT-Generated Unified Format), an optimized binary format designed for efficient model loading and inference [80].

13.3. Qwen2.5-VL-3B

The first model we tested was Qwen2.5-VL-3B [81] [82]. Released in 2024, Qwen2.5-VL is a state-of-the-art model featuring support for both image and video input. It enhances processing speed by implementing window attention for Vision Transformers (ViT) [83]. The model uses OpenCLIP ViT to convert images or videos into embeddings, which are then processed by the LLM component.

13.3.1. Resource Requirements & Feasibility

Qwen2.5-VL-3B contains 3 billion parameters. The estimated VRAM requirement for FP16 precision is:

$$\frac{3 \times 10^9 \times 2}{1024^3} \approx 5.6 \text{ GiB}$$

However, in practice, even a 12 GiB VRAM GPU failed to run the model due to additional memory overhead. We then tested it on an Apple Silicon machine with 32 GiB unified memory, but the model still ran out of memory. Thanks to swap memory, we managed to execute it, but with extremely poor efficiency. A 2024×1365 image took nearly 10 minutes to process, making it impractical for real-world use.

While downscaling input images might reduce memory consumption, we concluded that this model is too resource-intensive for consumer-level hardware and overkill for our specific task.

13.4. DeepSeek-VL-1.3B

Next, we tested DeepSeek-VL-1.3B [84] [85]. Like Qwen2.5-VL, it follows a ViT + LLM architecture, but instead of OpenCLIP, it uses SigLIP-L [86] as the vision encoder.

13.4.1. Deployment & Performance

Unlike Qwen2.5-VL, DeepSeek-VL-1.3B successfully ran on an NVIDIA GPU using only 4 GiB of VRAM. We then proceeded with evaluations of its performance.

13.4.2. Evaluating DeepSeek-VL-1.3B

Note that all prompts were executed in a single-turn manner, without any model memory of previous interactions.

Simple Prompting Approach

Our first approach was to directly ask the model to classify a yoga posture from an input image using a simple one-line prompt, as shown in [Appendix 1](#). However, the model did not perform as expected. Even when provided with a fixed set of posture choices, it consistently repeated the same answer across different images, failing to differentiate postures.

To address this, we expanded the prompt by providing detailed descriptions of each posture, as shown in [Appendix 2](#). Despite this, the model continued selecting the same response for every image. This suggested that it struggled to interpret the descriptions and compare them to the input image.



Figure 37. Down Dog



Figure 38. Plank



Figure 39. Side Plank



Figure 40. Warrior II

Table 4. The four images, each representing one of the four yoga posture types, used in the "[Simple Prompting Approach](#)" experiment.

Prompting for Image Descriptions

We then shifted our approach, asking the model to describe the action performed in an image using the prompt:

"Describe the action performed by the person in the image."

Surprisingly, the model provided detailed and reasonable descriptions of the images.

Below are some example responses:

| Figure | Ground Truth | Raw Model Response |
|--------|--------------|---|
| 37 | Down Dog | <p>The person in the image is performing a downward-facing dog (also known as a "plank") pose. This action involves lying on one's stomach with legs extended straight out from their body, and hands placed directly under the shoulders to support the weight of the upper body. The posture is commonly used for stretching and strengthening the core muscles.</p> |
| 38 | Plank | <p>The person in the image is performing a plank exercise on their hands and feet. They are lying flat on their stomach with their forearms supporting themselves, legs extended straight out in front of them. The individual's body forms an 'X' shape as they support their weight evenly between their arms and legs.</p> |
| 39 | Side Plank | <p>The person in the image is performing a side plank exercise. This can be inferred from several visual clues:</p> <ol style="list-style-type: none">1. The individual's body position: They are lying on their stomach with one hand supporting them above their head and the other hand touching the ground, forming an 'X'. Their legs are extended straight outwards, parallel to each other, which aligns with the typical form of a side plank.2. The positioning of the feet: Both feet are firmly placed on the mat, indicating that they are in a stable base for support during the movement.3. The alignment of the arms: One arm is raised while the other remains close to the body, maintaining balance throughout the exercise.4. The posture of the torso: The upper part of the body (including the neck) is slightly bent forward, suggesting engagement in the movement.5. The surface of the mat: The person is using a yoga mat, which is typically used for such exercises as it provides cushioning and stability. <p>These elements collectively suggest that the person is engaged in a side plank exercise, focusing on core strength and balance.</p> |

| | | |
|----|------------|---|
| 40 | Warrior II | The person in the image is performing a yoga pose known as "Urdhva Dhanurasana." This posture involves spreading out one's legs and arms, with both hands touching behind their head. The individual appears to be in a balanced stance, with their feet parallel to each other on a mat placed against a wall that has white curtains or blinds partially drawn. |
|----|------------|---|

From a human perspective, the responses were somewhat reasonable and captured many key image features, though not all. However, the model failed to recognize posture names correctly. For example, it misidentified Warrior II as Urdhva Dhanurasana (Wheel Pose), which is an entirely different posture.

13.4.3. Conclusion on DeepSeek-VL-1.3B

Our experiments revealed that:

- The model extracts visual features from an image reasonably well.
- However, it lacks a strong understanding of proper nouns (e.g., posture names).
- It struggles to match images with the correct posture type, even when explicit textual descriptions of each posture are provided.

Based on additional tests with a wider variety of prompts, we observed that the model could not achieve satisfactory yoga posture classification.

Given that DeepSeek-VL-1.3B is the smallest version of its model series, we strongly suspect that model size is a bottleneck. Larger models are likely to have better comprehension of instructions and descriptions, reducing errors in task-specific applications.

Nevertheless, DeepSeek-VL-1.3B demonstrates decent image-to-text capabilities, suggesting that Vision Transformers (ViTs) perform well beyond OCR tasks and can generalize to broader visual understanding. Therefore, we believe that larger and more advanced VLMs may be capable of handling general image-based classification tasks.

13.5. Llama-3.2-11B-Vision

To further validate our observations, we tested a larger VLM with more parameters. After research, we selected Llama-3.2-11B-Vision, a vision-enabled variant of Llama-3.2, an LLM developed by Meta in 2024 [87].

Like the previous two models, Llama-3.2-Vision incorporates a pre-trained image encoder to enable vision capabilities. However, the exact details of its encoder are not specified in its documentation [88].

13.5.1. Deployment & Performance

Despite its large parameter count of 11 billion, which is nine times more than DeepSeek-VL-1.3B and even exceeds Qwen2.5-VL-7B (the first model we tested), Llama-3.2-11B-Vision runs smoothly on our Apple Silicon platform. This is made possible by GGUF optimization and Ollama. GGUF quantizes the model, reducing its size while improving inference speed [89].

During inference, the peak VRAM usage was measured at about 9 GiB.

13.5.2. Evaluating Llama-3.2-11B-Vision

Note that all prompts were executed in a single-turn manner, with no model memory of previous interactions.

Simple Prompt on 4 Sample Images

For direct comparison with DeepSeek-VL-1.3B, we used the same four sample images shown in [Table 4](#). However, this time, we directly used the prompt in [Appendix 2](#), which includes the detailed description of each posture. Based on prior results, we suspected that models might lack detailed knowledge about yoga posture classification, so we aimed to supply more context in the prompt from the beginning.

The results were surprising. Likely due to the massive increase in parameters, despite some loss from quantization, the model demonstrated substantially improved comprehension. It correctly classified all four images, suggesting a significant improvement in visual understanding and response accuracy. The raw responses are as follows:

| Figure | Ground Truth | Raw Model Response |
|--------|--------------|--------------------|
| 37 | Down Dog | downdog. |
| 38 | Plank | plank. |
| 39 | Side Plank | side_plank. |
| 40 | Warrior II | warrior_ii |

Advanced Prompt on the Large Test Dataset

Given the model's strong performance on the four-image test, we conducted a more extensive evaluation using the same dataset from [Section 6.7.3](#), allowing for a direct comparison with our previous method.

Until now, our VLM evaluations only considered four correctly performed postures, without any incorrect samples. In this test, we expanded the task to binary correctness classification, assessing whether a posture was performed correctly in addition to identifying its type.

We maintained a 2-class correctness approach for simplicity, as this research stage focuses on exploring VLM potential rather than full integration. This also allows direct result comparison with [Section 6.7.3.2](#).

To classify both posture type and correctness, we extended the base prompt in [Appendix 2](#) by appending explicit criteria for each posture type to evaluate correctness, as shown in [Appendix 3](#). These criteria align with the standards established in [Section 5](#) for dataset construction.

The detailed results from this dataset are discussed in the upcoming Sections 13.5.3 and 13.5.4.

13.5.3. Time Performance

On our unified memory device, as described in [Section 13.2.1](#), processing all 1,156 samples in the test dataset took approximately 14 hours, averaging around 44 seconds per sample.

The histogram below illustrates the distribution of processing times for each sample. The results show that despite variations in image resolution across the dataset, processing time remains consistent at around 44 seconds per image, indicating little to no correlation between resolution and inference speed.

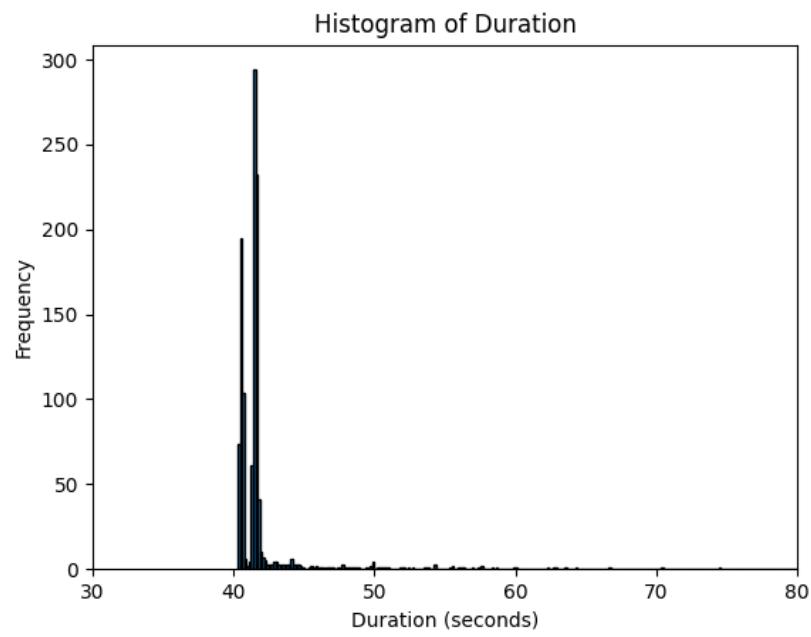


Figure 41. Histogram of processing times for 1,156 sample images.

13.5.4. Result Analysis

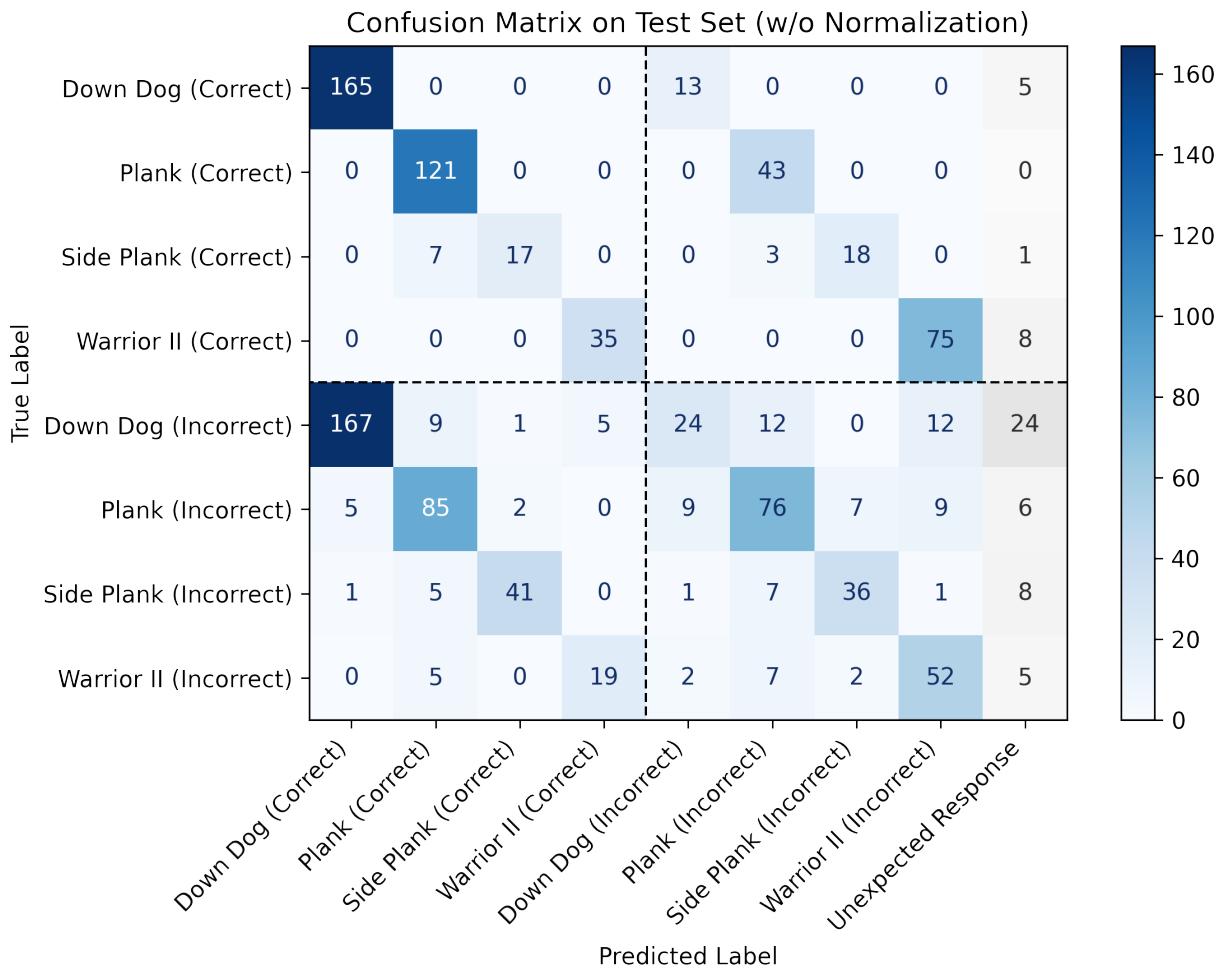


Figure 42. Confusion matrix showing the classification results of Llama-3.2-11B-Vision on the test dataset containing 1,156 samples. The format is consistent with that in [Section 6.7.3.2](#), facilitating direct comparison between the experimental VLM-based approach and our original two-stage method, and allowing for a clear evaluation of their accuracy on the same dataset.

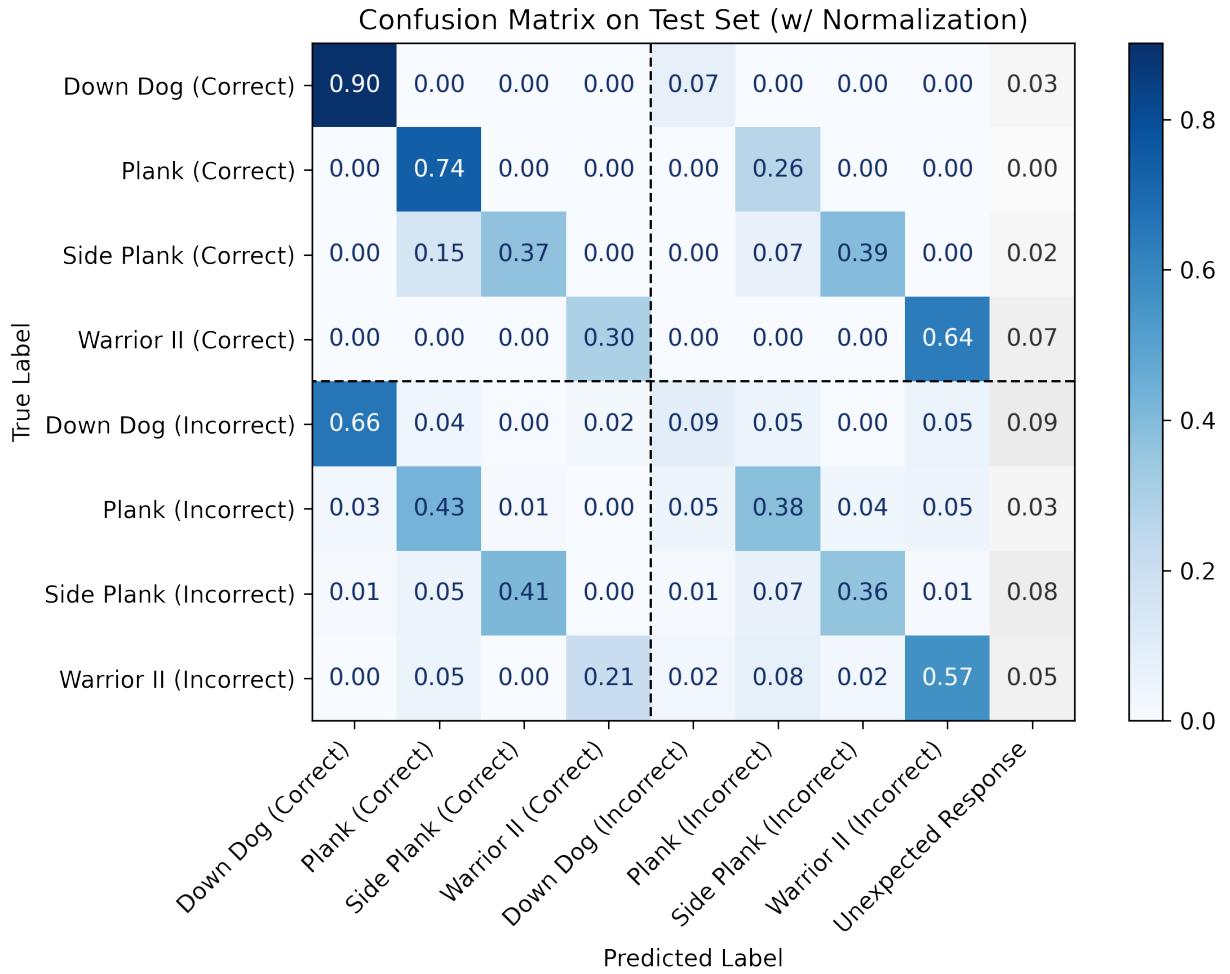


Figure 43. Row-wise normalized confusion matrix presenting the same classification results as in [Figure 42](#), allowing for clearer comparison of relative performance across classes, despite the imbalance in the test dataset.

Insight 1: LLM Instruction Misalignment Issue

The only difference from the confusion matrix in [Section 6.7.3.2](#) is that this matrix includes an additional column: "Unexpected Response." This accounts for instances where the LLM fails to follow the structured response format specified in the prompt.

Although we instructed the model to return predictions in a strict format (e.g., "warrior_ii; True"), it sometimes deviated. Instead of the expected output, it might return a sentence like:

"The posture type is warrior_ii and it is correct (True)."

Deviation from the format was more common when the model was uncertain about the answer. In such cases, it often provided explanations even though it was explicitly prompted not to, and sometimes it failed to classify the posture altogether.

Since these irregular responses are highly unpredictable, it is impractical to design an algorithm that accounts for all possible variations. Thus, we classify any response that cannot be parsed by our extraction algorithm as "Unexpected Response." Fortunately, this issue occurred in only 57 out of 1,156 samples (~5%), making it a minor concern overall.

Insight 2: Strong at General Classification but Weak in Fine-Grained Details

The confusion matrix reveals three clear diagonals:

1. Main diagonal (perfect classification): The model correctly predicted both the posture type and its correctness.
2. Top-right diagonal (false negatives): The posture type was classified correctly, but the model incorrectly marked a correct posture as incorrect.
3. Bottom-left diagonal (false positives): The posture type was classified correctly, but the model mistakenly identified an incorrect posture as correct.

While the model demonstrates strong performance in recognizing general posture types, its ability to assess posture correctness is weak. Taking Down Dog as an example:

- 189 samples were classified perfectly, accurately identifying posture type and correctness (165 correct-posture samples + 24 incorrect-posture samples).
- 180 samples accurately identified posture type but misclassified posture correctness (167 false positives + 13 false negatives).

This means that, even when the posture type is correctly identified, the correctness classification is only about 50% accurate, which is essentially random guessing for a binary decision.

This limitation is understandable. The four tested yoga postures differ significantly, making them easier to distinguish with simple textual descriptions (e.g., down dog forms an inverted "V," while plank is a straight body posture). However, posture correctness depends on subtle details such as leg bending or waist alignment, which are harder to describe in words and likely abstract to the LLM model.

Per-Class Performance Analysis

The following tables present the model's performance in posture type classification and posture correctness classification across the four yoga postures.

| Posture Class | Total Samples | Correct Classifications of Posture Types | Accuracy |
|---------------|---------------|--|----------|
| Down Dog | 437 | 369 | 0.844394 |
| Plank | 363 | 325 | 0.895317 |
| Side Plank | 146 | 112 | 0.767123 |
| Warrior II | 210 | 181 | 0.861905 |

Table 5. Accuracy of posture class classification, ignoring correctness evaluation.

| Posture Class | Correct Classification of Posture Types | Perfect Classifications (Correct in terms of both posture type and correctness) | Accuracy |
|---------------|---|---|----------|
| Down Dog | 369 | 189 | 0.512195 |
| Plank | 325 | 197 | 0.606154 |
| Side Plank | 112 | 53 | 0.473214 |
| Warrior II | 181 | 87 | 0.480663 |

Table 6. Accuracy of posture correctness classification, given that the posture class was correctly identified.

13.6. Limitations

13.6.1. Prompting Methodology

There are various ways to prompt an LLM, as discussed in the paper referenced in [Section 13.1.2](#). However, due to time constraints, our study focused solely on zero-shot prompting.

13.6.2. Prompt Format and Content

Our experiments evaluated only a limited range of prompt formats; in our case, plain text. In practice, LLM prompts can be structured in various ways (e.g., CSV, HTML), which may better suit models trained on more formatted data.

Moreover, prompt content can vary widely; more detailed or more concise descriptions of posture might influence performance. Because we did not have sufficient time to explore these variations, the prompt used in our experiments may not have been optimal.

13.7. Conclusion

Our findings indicate that, in terms of both accuracy and efficiency, the VLM approach does not yet offer advantages over our classical two-stage method for yoga posture classification.

13.7.1. Accuracy Considerations

The VLM models tested appear to lack the fine-grained capability needed to distinguish subtle posture details. Although we observed a notable improvement when moving from DeepSeek-VL-1.3B to Llama-3.2-11B-Vision, the overall accuracy remains limited. Even the best result we obtained with Llama-3.2-11B-Vision for posture correctness classification was nearly equivalent to random guessing. It is likely that larger models could perform better, but our experiments suggest that current VLMs are not yet sufficiently precise for our task.

13.7.2. Processing Time Constraints

Inference speed is a significant limitation for VLM-based approaches in real-time applications. While real-time yoga posture correction does not require millisecond-level responses, the best VLM we tested, Llama-3.2-11B-Vision, requires approximately 44 seconds per frame. Given that yoga poses typically last only a few breaths [90], an evaluation that takes longer than the pose itself makes the system impractical. This inefficiency stems from the large model size and parameter count, which are designed for general-purpose tasks rather than domain-specific classification.

13.7.3. Potential Advantages

Despite these shortcomings, VLMs offer the advantage of eliminating the need for extensive, manually labeled training data. In our study, we used only prompt engineering with Llama-3.2-11B-Vision and achieved at least 76% accuracy in classifying posture types, with accuracy reaching up to 89% for the best performing posture, all without any dataset preparation or model fine-tuning. This demonstrates the potential of VLMs as low-cost, rapid-deployment models.

13.7.4. Future Prospects

Ongoing advancements in hardware and LLM development suggest that future iterations may eventually overcome the aforementioned shortcomings and limitations.

On the hardware side, AI-focused computing advancements, such as Apple's M4 Max chip, claimed to be twice as fast as the M1 Max (a chip from the same series launched three years earlier) [91], could enable more powerful models to run locally with reduced latency.

On the model development side, techniques such as reinforcement fine-tuning, model compression (e.g., quantization), and domain-specific optimization can help reduce model size and improve accuracy for specialized tasks like yoga posture classification. Although these techniques are already in use, future advancements may simplify their application, making them more accessible for any specific task or model, and enabling them to be performed on local devices with limited resources.

In conclusion, although VLMs do not currently outperform our classical approach in terms of accuracy or processing speed, their potential to lower data labeling costs and serve as a foundation for future specialized models is promising. As LLM and VLM research continues to advance, we believe these models will eventually become viable for a broad range of AI applications, including image processing tasks like ours.

14. Individual Contributions

14.1. My Contributions

In this project, I was primarily responsible for the technical parts. At the very beginning, my teammate Leo and I agreed that, since he was less confident with technical tasks such as coding, I would take the lead on those aspects. During the first semester, Leo contributed to the preparation of the dataset for model training, while I handled the rest of the technical work. This division of labor aligned well with our initial plan, and progress in the first semester proceeded smoothly.

In the second semester, Leo went on exchange beginning February 1. Our original plan was to maintain a similar division of responsibilities: we intended to retrain the Shared MLP model (introduced in [Section 6](#)) using a more finely labeled dataset to enable a more detailed posture correctness classification. Leo was responsible for the data labeling task, which was critical for this phase. The plan and task requirements were communicated clearly to him in early March.

However, Leo reported that he lost his laptop during travel, which hindered his ability to complete the task. To prevent delays in project progress, I continued working on my assigned parts that did not depend on the new dataset and also initiated a side research task as a backup plan. We set March 29 as the deadline for the dataset to be completed, allowing sufficient time for model training, evaluation, and corresponding system adjustments.

Unfortunately, by the deadline, the labeled dataset had not been completed or uploaded to our project GitHub repository. As a result, we decided to drop the plan for retraining the Shared MLP model to support more posture correctness categories, as mentioned in [Section 2.1](#). Additionally, Leo participated in very few meetings with me and our supervisor after the exchange began, making it difficult to coordinate and maintain progress. Consequently, all work completed during the second semester was done by me alone.

14.2. Work Allocation Table

The following table outlines the contribution of each group member to the different sections of the report. In our group, the member responsible for a section handled both the implementation and the writing. If a section was completed by one member only, it means that person was solely responsible for the work and reporting of that part.

| Section | Hui Wang Chi, Gordon | Yuen Ho, Leo |
|--|-------------------------|--------------|
| 1. Abstract | ✓ | |
| 2. Introduction | ✓ | ✓ |
| 3. Related Works | ✓ | ✓ |
| 4. Benchmark on Existing Pose Detection Models | ✓ | |
| 5. Dataset Preparation | | ✓ |
| 6. Model Development | ✓ | |
| 7. Model Integration | ✓ | |
| 8. Result Stabilization | ✓ | |
| 9. User Interface | ✓ | |
| 10. Practical Evaluation | ✓ | |
| 11. Limitations and Challenges | ✓ | |
| 12. Conclusion | ✓ | |
| 13. Side Research: Language Model-Based Approach | ✓ | |

15. Appendices

15.1. Responses to Questions from the Supervisor and Grader (Semester 1 Presentation)

Q: Can the overall accuracy of 91% per frame, as mentioned in the presentation, be further improved?

A: Yes, the accuracy has been improved using the frame majority voting technique, as discussed in [Section 8.3](#). This method enhances prediction stability and overall accuracy by aggregating results across multiple frames.

Q: Have you considered using 3D simulations to generate synthetic human skeleton data for training?

A: While we recognized the potential benefits of 3D simulations, we did not study this approach for two main reasons:

- Project Scope and Planning: According to our initial plan, we did not intend to expand the dataset in Semester 2. Instead, we prioritized improving our system and conducting side research using the existing dataset.
- Technical Expertise: Neither of us had prior experience with 3D modeling or simulation. Learning and implementing it from scratch would have required an uncertain amount of time and effort, which could have shifted our focus away from the core objectives.

After consulting with our supervisor, we prioritized our original roadmap. Unfortunately, due to time constraints, we were unable to explore this area further.

15.2. LLM Prompts

1. Identify the yoga pose in the given image and respond with ONE word of the following: [downdog, plank, side_plank, warrior_ii, none].
2. Identify the yoga pose in the given image and respond with ONE word of the following: [downdog, plank, side_plank, warrior_ii, none].
downdog: The body forms an inverted "V" shape with hands and feet on the ground, hips raised high. Arms and legs are straight, and the head aligns with the arms. Heels may be touching or slightly lifted off the ground.
plank: The body is in a straight line from head to heels, supported by hands (or forearms) and toes. The back remains flat, core engaged, and hips should not sag or lift excessively.
side_plank: The body is supported on one hand (or forearm) and the outer edge of one foot. The other foot

may be stacked or staggered. The body remains in a straight line from head to heels, with the free arm often extended upward. warrior_ii: One leg is bent at the knee at approximately 90 degrees, while the other remains straight and extended behind. The arms are extended parallel to the ground, in line with the legs, and the torso remains upright. The head looks over the front hand. none: The pose does not match any of the above categories.

3. Identify the yoga pose in the given image and respond with ONE word of the following: [downdog, plank, side_plank, warrior_ii, none]. downdog: The body forms an inverted "V" shape with hands and feet on the ground, hips raised high. Arms and legs are straight, and the head aligns with the arms. Heels may be touching or slightly lifted off the ground. plank: The body is in a straight line from head to heels, supported by hands (or forearms) and toes. The back remains flat, core engaged, and hips should not sag or lift excessively. side_plank: The body is supported on one hand (or forearm) and the outer edge of one foot. The other foot may be stacked or staggered. The body remains in a straight line from head to heels, with the free arm often extended upward. warrior_ii: One leg is bent at the knee at approximately 90 degrees, while the other remains straight and extended behind. The arms are extended parallel to the ground, in line with the legs, and the torso remains upright. The head looks over the front hand. none: The pose does not match any of the above categories.

Besides reporting the posture type, also determine if the posture is correct, as a Boolean value True or False. For downdog, if an image clearly shows the practitioner forming an inverted V shape with hips lifted, legs extended, and arms in a straight line with the spine, then it is True; however, if the back is rounded, the arms and back do not form a straight line, or there is a noticeable bend in the knees, then it is False. For plank, if the posture displays a continuous straight line from the head to the heels with the forearms and upper arms perpendicular to the ground and no deviation in the hip alignment, then it is True; conversely, if the hips are raised too high or dropped too low, the arms are bent or not perpendicular, or the image resembles a variant like Purvottanasana, then it is False. For side_plank, if the image demonstrates a proper transition from a standard plank with the weight shifted onto one hand, the feet stacked, and the body forming a straight line with a support arm that is correctly bent (forming a right angle), then it is True; otherwise, if the feet are misaligned, the hips are off the straight line, or the support arm is straight instead of bent, then it is False. For warrior_ii, if the stance is wide with the front foot pointing forward, the front knee bent exactly at a 90-degree angle with the knee directly over the ankle while the back leg remains straight, and the torso stays upright with the arms extended parallel to the ground, then it is True; but if the torso is shifted, the arms sag or are raised improperly, or the knee angle deviates from 90 degrees, then it is False. Respond strictly in the format: 'POSTURE_TYPE; TRUE/FALSE'. No additional text or explanations.

16. References

- [1] A. Pizer, "How to Do Downward Facing Dog (Adho Mukha Svanasana) in Yoga," Verywell Fit, 16 7 2021. [Online]. Available: <https://www.verywellfit.com/downward-facing-dog-adho-mukha-svanasana-3567072>.
- [2] E. Quinn, "How to Do a Plank: Proper Form, Variations, and Common Mistakes," Verywell Fit, 12 5 2024. [Online]. Available: <https://www.verywellfit.com/the-plank-exercise-3120068>.
- [3] A. Asher, "Side Plank: Proper Form, Variations, and Common Mistakes," Verywell Fit, 7 6 2024. [Online]. Available: <https://www.verywellfit.com/how-to-safely-progress-your-side-plank-exercise-4016853>.
- [4] A. Pizer, "Fierce Warrior Pose Yoga Sequence," Verywell Fit, 8 12 2021. [Online]. Available: <https://www.verywellfit.com/get-fierce-with-this-sequence-of-warrior-poses-3567198>.
- [5] M. Campo, M. P. Shiyo, M. B. Kean, L. Roberts, and E. Pappas, "Musculoskeletal pain associated with recreational yoga participation: A prospective cohort study with 1-year follow-up," *J. Bodywork Movement Ther.*, vol. 22, no. 2, pp. 418–423, 4 2018. [Online]. Available: <https://doi.org/10.1016/j.jbmt.2017.05.022>.
- [6] Peloton, "Guided yoga classes online | Peloton," [Online]. Available: <https://www.onepeloton.com/classes/yoga>.
- [7] Yoga Studio App, "#1 Yoga Studio App: Mind & Body," [Online]. Available: <https://yogastudioapp.com/>.
- [8] Alo Moves, "Alo Moves | Your At-Home Studio," [Online]. Available: <https://www.alomoves.com/>.
- [9] DOYOU, "DOYOU | Online Yoga, Fitness, and You," [Online]. Available: <https://www.doyou.com/>.
- [10] Gaia, "Streaming Online Yoga Videos | Gaia," [Online]. Available: <https://www.gaia.com/yoga>.
- [11] Glo, "Glo | Online yoga, meditation, & Pilates app for all levels," [Online]. Available: <https://www.glo.com/>.

- [12] B. Jo and S. Kim, "Comparative analysis of OpenPose, PoseNet, and MoveNet models for pose estimation in mobile devices," *Traitement du Signal*, vol. 39, no. 1, pp. 119, 2022. [Online]. Available: <https://www.researchgate.net/publication/359476644>.
- [13] P. Kaushik, B. P. Lohani, A. Thakur, A. Gupta, A. K. Khan, and A. Kumar, "Body posture detection and comparison between OpenPose, MoveNet and PoseNet," in 2023 6th International Conference on Contemporary Computing and Informatics (IC3I), vol. 6, pp. 234-238, IEEE, 2023. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10397937/>.
- [14] N. Kumar Reddy Boyalla, "Real-time Exercise Posture Correction Using Human Pose Detection Technique," *soar.suny.edu*, 12 2021. [Online]. Available: <https://soar.suny.edu/handle/20.500.12648/8622>.
- [15] Y. Kwon and D. Kim, "Real-Time workout posture correction using OpenCV and MediaPipe," *한국정보기술학회논문지*, vol. 20, no. 1, pp. 199-208, 2022. [Online]. Available: <https://ki-it.com/xml/32020/32020.pdf>.
- [16] L. Rajak, "Yoga Pose Dataset," Kaggle, 2023. [Online]. Available: <https://www.kaggle.com/datasets/lakshmanarajak/yoga-dataset>.
- [17] TensorFlow Team, "@tensorflow-models/pose-detection," npm, Version 0.0.3, [Online]. Available: <https://www.npmjs.com/package/@tensorflow-models/pose-detection/v/0.0.3>.
- [18] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," European Conference on Computer Vision (ECCV), Zurich, Switzerland, 9 2014. [Online]. Available: <https://cocodataset.org/#home>.
- [19] TensorFlow Hub, "MoveNet: Ultra fast and accurate pose detection model," TensorFlow, 2023. [Online]. Available: <https://www.tensorflow.org/hub/tutorials/movenet>.
- [20] Basj, "Total memory used by Python process?," Stack Overflow, 7 2 2014. [Online]. Available: <https://stackoverflow.com/a/7669482/15416614>.
- [21] "Choosing a Resource Storage Mode for Apple GPUs," Apple Developer Documentation, [Online]. Available: https://developer.apple.com/documentation/metal/resource_fundamentals/choosing_a_resource_storage_mode_for_apple_gpus.

- [22] D. Oved, "Real-time human pose estimation in the browser with TensorFlow.js," The TensorFlow Blog, 7 5 2018. [Online]. Available: <https://blog.tensorflow.org/2018/05/real-time-human-pose-estimation-in.html>.
- [23] CMU Perceptual Computing Lab, "OpenPose: Real-time multi-person keypoint detection library for body, face, hands, and foot estimation," GitHub repository, 2024. [Online]. Available: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>.
- [24] melodocoder, "Installing models through getBaseModels.bat (Windows 11, OpenPose Release v1.7.0)," GitHub, 30 9 2024. [Online]. Available: <https://github.com/CMU-Perceptual-Computing-Lab/openpose/issues/2316>.
- [25] TensorFlow, "姿态预测," TensorFlow Lite 示例, 11 1 2024. [Online]. Available: https://www.tensorflow.org/lite/examples/pose_estimation/overview?hl=zh-cn.
- [26] Google AI Edge, "GPU delegates for LiteRT," Google AI for Developers, 30 8 2024. [Online]. Available: <https://ai.google.dev/edge/literr/performance/gpu>.
- [27] Google AI, "Pose Landmarker," Google MediaPipe Solutions, [Online]. Available: https://google.dev/edge/mediapipe/solutions/vision/pose_landmarker.
- [28] E. Millard, "A Step-By-Step Guide On How to Do Tree Pose (or One of Its Many Variations)," onepeloton, 28 6 2024. [Online]. Available: <https://www.onepeloton.com/blog/tree-pose/>.
- [29] M. Tyagi, "Yoga Posture Dataset," Kaggle, 2024. [Online]. Available: <https://www.kaggle.com/datasets/tr1gg3rtrash/yoga-posture-dataset>.
- [30] U. Chowdhury, "Yoga Pose Classification," Kaggle, 2024. [Online]. Available: <https://www.kaggle.com/datasets/ujjwalchowdhury/yoga-pose-classification>.
- [31] S. Saxena, "Yoga Pose Image Classification Dataset," Kaggle, 2024. [Online]. Available: <https://www.kaggle.com/datasets/shrutisaxena/yoga-pose-image-classification-dataset>.
- [32] S. Khaorapapong and B. Latif, "Yoga Posture Dataset," Kaggle, 2024. [Online]. Available: <https://www.kaggle.com/datasets/suradechk/yoga-posture-cleaned>.
- [33] Nektony, "Duplicate File Finder for Mac - Free Download," [Online]. Available: <https://nektony.com/duplicate-finder-free>.

- [34] L. Bové, "10 Benefits Of Downward Dog: This Is What Happens To Your Body When You Practice Daily," *Yogajala*, 24 2023. [Online]. Available: <https://yogajala.com/benefits-of-downward-dog/>.
- [35] David O Yoga, "DOWNWARD DOG POSE - 3 Common Mistakes to Avoid 🙌 #shorts," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=d7i3sI8G8g0>.
- [36] Alexia K Yoga, "STOP Doing These 3 Yoga Mistakes!," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=FFe2JjoMdKs>.
- [37] Yogaguru Shailendra, "Common Mistakes in Yoga Poses," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=8npLig1wqiw>.
- [38] Kaya Yoga Therapy, "Downward Facing Dog WRONG/RIGHT," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=zBVpkKKOjZM>.
- [39] YogaEasy, "Train your Downdog - adapt yoga poses for your body," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=dbKYu2NliDQ>.
- [40] Landon Slaughter, "Downward Facing Dog Blunders," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=ustK72GJHio>.
- [41] C. Camille, "Plank Pose In Yoga: Benefits, Technique, and Variations," Everything Yoga Retreat, 20 9 2023. [Online]. Available: <https://www.everythingyogaretreat.com/plank-pose-yoga/>.
- [42] Get Exercise Confident, "How to do the perfect PLANK: technique and common mistakes," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=6LqqeBtFn9M>.
- [43] YouAligned, "HOW TO DO PLANK POSE - Common Mistakes and How to Correct Them," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=ME8LyljylHg>.
- [44] Archie's Yoga, "Plank common mistakes!," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=FScjW6KMVHw>.
- [45] Go Yoga! Express, "How to avoid injury from common mistakes made in Plank Pose (phalakasana)," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=NUWcCoPI3E8>.

- [46] Siddhi Yoga International, "Phalakasana (Plank Pose) Benefits, How to Do & Contraindications by Yogi Sandeep - Siddhi Yoga," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=LrqECYmDexQ>.
- [47] Yoga With Adriene, "Side Plank Pose | Vasisthasana," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=dFCRqn0RzJA>.
- [48] L. McGlashan, "10 Common Mistakes You May Be Making in Plank Pose (And How to Correct Them)," Yoga Journal, 28 10 2024. [Online]. Available: <https://www.yogajournal.com/practice/plank-pose-yoga/>.
- [49] YYOGA at Home — yoga classes + tutorials, "AVOID the Most Common Mistakes in Warrior II!," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=6xuo7SX256Y>.
- [50] Man Flow Yoga, "Most Common Errors in Warrior 2 (and how to fix)," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=tk9pXHH1dn4>.
- [51] Man Flow Yoga, "Biggest Error in Warrior 2 (and how to fix it)," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=i7t3piOb5D4>.
- [52] Manduka, "3 Common Mistakes to Avoid in Warrior 2," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=wSve4dX4Cjs>.
- [53] BrettLarkinYoga, "The Right Way to Do Warrior 2 (in just 60 Seconds) | Greatist Yoga Alignment Fix | Virabhadrasana 2," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=ANvqnuUwcvk>.
- [54] YJ Editors, "Warrior 2 Pose: How to Practice Virabhadrasana II," Yoga Journal, 26 4 2023. [Online]. Available: <https://www.yogajournal.com/poses/warrior-ii-pose/>.
- [55] ISSA, "Warrior 2: Benefits, Instructions, Variations, and More," ISSA Yoga Blog, 27 10 2023. [Online]. Available: <https://yoga.issaonline.com/blog/post/warrior-2-benefits-instructions-variations-and-more>.
- [56] Ashish, "Warrior II Pose (Virabhadrasana II): How to Do, Benefits, and Modifications," Fitsri, 3 8 2024. [Online]. Available: <https://www.fitsri.com/poses/warrior-2>.
- [57] J. Lizzie, "Cultivating Power in Your Practice With Warrior Two Pose," Yogapedia, 13 12 2017. [Online]. Available: <https://www.yogapedia.com/2/7135/asana/asana-tips/cultivating-power-in-your-practice-with-warrior-two-pose>.

- [58] R. Scott, "The Five Most Common Alignment Mistakes in Warrior II," Yoga International, 2024. [Online]. Available: <https://yogainternational.com/article/view/the-five-most-common-alignment-mistakes-in-warrior-ii/>.
- [59] Body By Yoga, "Warrior 2: Do's and Don'ts And Practical Tips When You're Just Starting Out," Body By Yoga Training, 28 10 2021. [Online]. Available: <https://bodybyyoga.training/yoga-for-beginners/warrior-2-pose/>.
- [60] Dakota Durant, "NEVER DO THE SIDE PLANK LIKE THIS! | Most Common Mistakes," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=vYuGESCh7yQ>.
- [61] Claire DeFitt, "Beginner Side Plank Mistake," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=BInNyubgQ88>.
- [62] Drmboy Sunny, "❌ Common Side Plank Mistake | (Don't Do This)," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=4ezvhw0hZYM>.
- [63] Get Exercise Confident, "How to do the perfect Side Plank and most common mistakes," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=rCx2nG9vQ0>.
- [64] Fitness library_1, "SIDE PLANK Mistakes ❌ 🚫 #shorts," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=RSeNVIRPRsU>.
- [65] Google Developers, "Pose detection," ML Kit Documentation, 10 7 2024. [Online]. Available: <https://developers.google.com/ml-kit/vision/pose-detection>.
- [66] Google Developers, "Pose detection model card," ML Kit Documentation, 22 6 2021. [Online]. Available: https://developers.google.com/ml-kit/images/vision/pose-detection/pose_model_card.pdf.
- [67] Google AI, "MediaPipe Pose: Real-time body pose tracking solution," GitHub repository, 2024. [Online]. Available: <https://github.com/google-ai-edge/mediapipe/blob/master/docs/solutions/pose.md>.
- [68] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," arXiv preprint arXiv:2010.11929, 10 2020. [Online]. Available: <https://arxiv.org/abs/2010.11929>.

- [69] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," arXiv preprint arXiv:1612.00593, 12 2016. [Online]. Available: <https://arxiv.org/abs/1612.00593>.
- [70] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," arXiv preprint arXiv:1206.5533, 6 2012. [Online]. Available: <https://arxiv.org/abs/1206.5533>.
- [71] "CrossEntropyLoss — PyTorch 2.5 documentation," PyTorch Documentation, [Online]. Available: <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>.
- [72] "BCEWithLogitsLoss — PyTorch 2.5 documentation," PyTorch Documentation, [Online]. Available: <https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html>.
- [73] "SGD — PyTorch 2.5 documentation," PyTorch Documentation, [Online]. Available: <https://pytorch.org/docs/stable/generated/torch.optim.SGD.html>.
- [74] "ReduceLROnPlateau — PyTorch 2.5 documentation," PyTorch Documentation, [Online]. Available: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html.
- [75] "GlobalInterpreterLock - Python Wiki," wiki.python.org, [Online]. Available: <https://wiki.python.org/moin/GlobalInterpreterLock>.
- [76] M. Noyan and E. Beeching, "Vision Language Models Explained," huggingface.co, 11 4 2024. [Online]. Available: <https://huggingface.co/blog/vlms>.
- [77] F. Bordes et al., "An Introduction to Vision-Language Modeling," arXiv.org, 27 5 2024. [Online]. Available: <https://arxiv.org/abs/2405.17247>.
- [78] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications," arXiv (Cornell University), 5 2 2024. [Online]. Available: <https://arxiv.org/abs/2402.07927>.
- [79] T. B. Brown et al., "Language Models Are Few-Shot Learners," arxiv.org, 22 7 2020. [Online]. Available: <https://arxiv.org/abs/2005.14165>.
- [80] "GGUF," Huggingface.co, 2024. [Online]. Available: <https://huggingface.co/docs/hub/gguf>.

- [81] Qwen et al., "Qwen2.5 Technical Report," arXiv.org, 3 1 2025. [Online]. Available: <https://arxiv.org/abs/2412.15115>.
- [82] "Qwen/Qwen2.5-VL-3B-Instruct-AWQ · Hugging Face," Huggingface.co, 3 2025. [Online]. Available: <https://huggingface.co/Qwen/Qwen2.5-VL-3B-Instruct-AWQ>.
- [83] QwenLM, "GitHub - QwenLM/Qwen2.5-VL: Qwen2.5-VL is the multimodal large language model series developed by Qwen team, Alibaba Cloud., GitHub, 2024. [Online]. Available: <https://github.com/QwenLM/Qwen2.5-VL>.
- [84] H. Lu et al., "DeepSeek-VL: Towards Real-World Vision-Language Understanding," arXiv.org, 2024. [Online]. Available: <https://arxiv.org/abs/2403.05525>.
- [85] "deepseek-ai/deepseek-coder-1.3b-instruct · Hugging Face," Huggingface.co, 16 8 2024. [Online]. Available: <https://huggingface.co/deepseek-ai/deepseek-coder-1.3b-instruct>.
- [86] "timm/ViT-L-16-SigLIP-384 · Hugging Face," Huggingface.co, 2023. [Online]. Available: <https://huggingface.co/timm/ViT-L-16-SigLIP-384>.
- [87] A. Dubey et al., "The Llama 3 Herd of Models," arXiv.org, 23 11 2024. [Online]. Available: <https://arxiv.org/abs/2407.21783>.
- [88] Meta, "Llama 3.2: Revolutionizing edge AI and vision with open, customizable models," Meta.com, 25 9 2024. [Online]. Available: <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>.
- [89] T. Mucci, "GGUF versus GGML," Ibm.com, 3 7 2024. [Online]. Available: <https://www.ibm.com/think/topics/gguf-versus-ggml>.
- [90] T. Burgin, "How Long to Hold a Yoga Pose: Factors, Benefits and Guidelines • Yoga B," Yoga Basics, 29 8 2023. [Online]. Available: <https://www.yogabasics.com/practice/yoga-for-beginners/how-long-to-hold-a-yoga-pose/>.
- [91] Apple, "Apple introduces M4 Pro and M4 Max," Apple Newsroom, 30 10 2024. [Online]. Available: <https://www.apple.com/newsroom/2024/10/apple-introduces-m4-pro-and-m4-max/>.