ELEC 490/498 Final Report

Deep Learning Calorie Counter

WORKOUTPAL

Submitted By: Group 59
Group Members: Andy Craig, Nathan Goodman, Brock Tureski, John Turnbull
Faculty Supervisor: Il-Min Kim

Contents

Table of Figures

Table of Tables

Executive Summary

The workout companion app, Workoutpal, sets out to transform the workout fitness industry. Workoutpal provides users with an unparalleled exercise prediction, workout analysis and caloric expenditure predictions. This executive summary lays out a comprehensive overview of the project covering: system integration, UI design, dataset generation, exercise prediction, workout analysis, testing and stakeholder considerations. The focus for system integration was placed on establishing seemless communication channels between the UI and backend API which hosts the machine learning model responsible for exercise inference as well as the exercise prediction and workout analysis algorithms. The API is hosted on a virtual machine, which serves as the backbone of the system. Mechanisms for model storage and retrieval have been implemented, ensuring efficient updates to prediction models stored in Google Cloud Storage. Moreover, user authentication and data security have been prioritized by leveraging Firebase Authentication, thereby safeguarding user privacy. From a user interface perspective, the mobile application has been carefully crafted to prioritize simplicity and functionality. Users can upload workout videos, input work out details such as intensity, body weight and load weight, after submitting this information the user will receive a comprehensive workout summary giving information such as caloric expenditure, reps completed, average time per repetition, as well as a chart showing repetition depth. Dataset generation has been a crucial aspect of the project, ensuring the availability of high-quality data for training the exercise inference model. Data collection sources included a combination of personal workout videos which were recorded with the specific goal of training this model allowing greater focus on angle variance, as well as videos of fitness influencers with their permission. This combination gave the data set an effective range of variance between exercise form, and angles allowing for a diverse model that is effective in a multitude of scenarios. For exercise prediction, a robust model has been implemented, combining YOLOv8 pose estimation model with a Multi-Layer Perceptron Classifier. Rigorous testing and evaluation has proved the model has a high accuracy and precision across diverse user profiles, affirming its efficacy in accurately predicting exercises from user-uploaded videos. Workout analysis encompasses calorie counting and repetition analysis, providing users with convenient and valuable insights into their daily workout routine. Through the use of metabolic equivalent task values and user input, caloric output is predicted. Repetition analysis utilizes pose estimation data to count repetitions performed. Training, evaluation, and validation have been integral to the success of this project, ensuring accuracy and reliability of predictions and analysis. Stakeholder considerations have had one focus through the entire design and development process of the project; delivering an unrivaled experience to the user. With this came a need for a simple, secure, and privacy-focused app. By prioritizing user experience and data security, trust is aimed to be built with users, ensuring a seamless and successful rollout of WorkoutPal.

Section 1: Motivation and Background

Over the last decade, the global fitness industry has experienced significant growth, with that growth expected to continue over the course of the next decade. According to a research study conducted by Custom Market Insights, the Global Health and Fitness Club Market—a market encompassing: gym memberships, gym admission fees, and personal training services; was estimated at 78.0 Billion USD in 2021, and is expected to hit approximately 125.23 Billion USD by 2030. [1] This combined with the rapid growth of in-home workouts and training programs since the pandemic, led to an explosion in fitness related products and software that allow consumers to collect and analyze personal workout data.

For our capstone, motivated by common interests in the field of health and fitness as well as artificial intelligence, Group 59 took on the project of creating a deep learning calorie counter for use during personal workouts. The original project proposal specifies creating a neural network to detect the type of workout being performed within a video sequence, then calculating/estimating the calories burned using information derived from said sequence.

While Group 59, would create a neural network capable of detecting the type of workout being performed within a video sequence and then calculating/estimating the calories burned, Group 59 decided to expand the scope of the project to a “mobile exercise companion application”. With the goal of creating a mobile app that can provide additional workout statistics in addition to estimating the calories burned from a video sequence.

The choice to expand the scope of this project was made based upon the ability of existing products such as a FitBit or Apple Watch, to fulfill the role of a workout calorie estimator/counter. As well as the revelation that calorie expenditure is closely related to heartrate—a statistic that varies based on a subject’s physical health and workout history (variables that are hard to discern from video sequence), that also happens to be precisely measured by the previously mentioned products.

This expansion of scope allowed Group 59 to create a prototype that is not only more useful to consumers but is also more applicable/complementary to products and service offerings a fitness consumer may already use such as: virtual fitness classes and workout programs. Group 59's Deep Learning Calorie Counter application includes: an exercise detection neural network, a mobile UI for analyzing your personal fitness data, and additional scripts for tracking the number and depth of repetitions within a video sequence.

Group 59's Deep Learning Calorie Counter application, fills a niche in the area of fitness analytics as a fitness companion app with video data is theoretically able to do things such as correct a consumer's exercise form and provide real-time workout instruction—feats not typically achievable without visual input or a live instructor. Such an application could be applied alongside existing virtual fitness classes or workout programs to improve the user experience and provide a sort of one-stop-shop to consumers for fitness needs.

Section 2: Design

2.1 Functional Requirements

The functional requirements of the Calorie Counter project encompass the core functionalities that the system must deliver to meet the needs of its users. These requirements are derived from the specifications outlined in the project blueprint, encompassing features such as video submission, model inference, caloric estimation, repetition counting, and user interface. Additionally, various constraints, including performance targets and tolerances, must be considered to ensure the system operates effectively and meets user expectations.

Model Inference:

- The system must accurately predict the exercise class from uploaded videos using pre-trained models.
- The model inference process should achieve a minimum accuracy rate of 90% for exercise classification.

- Inference speed should be optimized to complete within 30 seconds for a typical exercise video, with a tolerance of ±1 second to minimize user wait time.

Caloric Estimation:

- Caloric expenditure estimation should be based on predicted exercise classes and user-provided information.
- The system must provide caloric expenditure estimates within a margin of error of 5% compared to actual expenditure, with a tolerance of ±2%.

Video Submission:

- Users should be able to upload exercise videos through the mobile application.
- Contextual information such as age, weight, height, exercise load, and repetitions must accompany each video submission.
- The system should process video submissions within 10 seconds, with a tolerance of ±2 seconds, to maintain user satisfaction and responsiveness.

User Interface:

- The mobile application must feature a user-friendly interface for seamless navigation.
- Users should be able to authenticate login credentials and access past workout history effortlessly.
- App responsiveness, including smooth transitions between screens, is critical for providing an optimal user experience.

Repetition Counting (Bonus):

- Accurate counting of repetitions performed in exercise videos is essential for tracking progress.
- The system should achieve a minimum accuracy rate of 95% in counting repetitions, with a tolerance of ±1 counts to accommodate variations in exercise performance.

2.2 Constraints

The constraints of this project are primarily related to data privacy considerations as well as some software constraints. It is vital that the data we use for training, testing, validating and evaluating is open

source. If it is not open source, the necessary steps must be taken to ensure compliance with data privacy laws.

Data Collection:

- Obtain explicit permission for using exercise videos from content creators.
- Ensure all images sourced externally are legally acquired and securely stored.
- Implement stringent measures to safeguard user-generated data and adhere to relevant data protection regulations.

Performance Targets:

- The system must meet specified performance targets for inference speed, video submission processing time, and caloric expenditure estimation accuracy to ensure timely and accurate results.
- Any deviations from performance targets should be within acceptable tolerances to maintain overall system efficiency and user satisfaction.

Data Security:

- User data and video uploads must be securely stored and protected from unauthorized access to maintain user privacy and confidentiality.
- Implementation of encryption protocols, such as TLS, is necessary to ensure the security of data transmission and storage.

2.3 Design Methodology

The design of this project underwent many iterative improvements as the group progressed. The original design of the project was centered around the use of the YOLOv5 object detection framework to train and productionize our proprietary exercise prediction system. The team planned to create a custom dataset for the purpose of classifying 3 exercise types: Squats, Deadlifts and Pushups. These exercises were chosen due to the following reasons:

1. Exercise equipment is accessible for all our group members.
2. Videos are easy to obtain from online sources.

3. Videos are easy to record from a single camera angle.

The team understood that most of the time would be spent in the data collection phase of the project, and all subsequent phases of the project would greatly depend on the quality of data that was produced.

The original development of the calorie expenditure prediction system followed a structured approach comprising five distinct phases: Data Acquisition, Exercise Prediction, Workout Analysis, API and Systems Integration, and User Interface development. Each phase was characterized by iterative design, development, and testing.

2.4 Data Acquisition

The quality of our dataset is single-handedly responsible for the success of our exercise prediction model, and thus the most important phase of the project. Every member of the team contributed to this phase, and thus, in order to ensure that all images collected for our dataset were of good quality, the following guidelines were introduced.

- Video must contain one or more of the following exercises: Squat, Deadlift or Pushup.
- Video must contain a clear view of the entire exercise in question.
- Video should contain little to no text on top of the video itself, especially not containing text that that says "squat " or "deadlift".
- Video should consist of above 50% time under tension to video length.
- Video should be taken from a single, fixed angle.
- Video should be taken from a smartphone
- Video should have an aspect ratio of 4:3 or 16:9

These set of guidelines allowed our team to individually collect both self-recorded videos, as well as videos from online sources.

The initial plan was to collect data by recording our own exercises, however, with a team of four people the variance of the dataset would be extremely low if this was the only form of data collected and could

cause issues in model accuracy when encountering variance in lifting forms. To mediate this problem, the team planned to send messages to fitness influencers and friends to obtain permission to use their exercise videos to train the exercise inference model. This mix of video sources was ideal from a data variance perspective, because while a more diverse group of people performing exercises, we allowed for greater variance in lifting form, anecdotally, a lot of fitness influencers film their workouts from similar angles, which causes for little variance in camera position. Recording videos internally remedied this problem by allowing for filming from a wide range of angles, creating even greater variance in the dataset.

2.5 Exercise Classification

As mentioned in 2.3 Design Methodology, the original design opted to use the Ultralytics YOLOv5 image classification framework to assist in training our exercise prediction model. YOLOv5 is an open-source deep learning framework known for its efficiency in real-time object detection tasks, but it can also be adapted for image classification [2]. The team originally decided to use this framework because it offers the capability to train custom image classification models using user-provided datasets through a straightforward workflow, enabling us to create models tailored to our specific classification needs with ease.

When performing more research into YOLOv5, the team discovered the new and improved YOLOv8 framework, which had recently added a new object detection task to its library: Pose Estimation. Pose estimation refers to the process of estimating the pose or orientation of objects or individuals in an image or video. This typically involves determining the position, orientation, and sometimes even the movement of specific keypoints or joints on the object or person being analyzed.

2.5.1 Pose Estimation

The YOLOv8 pose estimation model was trained using the COCO-Pose dataset, which is a specialized version of the COCO (Common Objects in Context) dataset, designed for pose estimation tasks [3]. COCO-Pose builds upon the COCO Keypoints 2017 dataset which contains 200K images labeled with

keypoints for pose estimation tasks. The dataset supports 17 keypoints for human figures, facilitating detailed pose estimation [4]. Below is a figure illustrating the keypoints and their corresponding body part.

Index	Key point
0	Nose
1	Left-eye
2	Right-eye
3	Left-ear
4	Right-ear
5	Left-shoulder
6	Right-shoulder
7	Left-elbow
8	Right-elbow
9	Left-wrist
10	Right-wrist
11	Left-hip
12	Right-hip
13	Left-knee
14	Right-knee
15	Left-ankle
16	Right-ankle

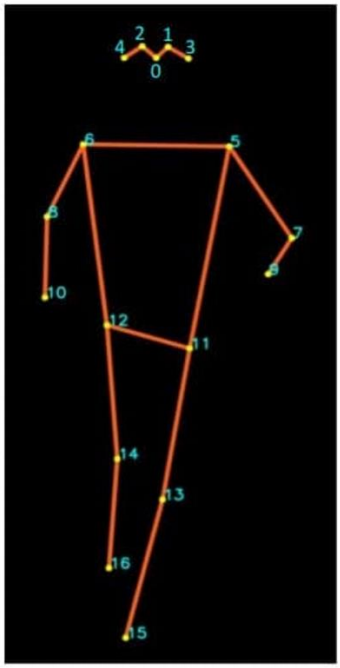

Figure 1: YOLOv8 Pose Estimation, displaying the input image on the left, and the output on the right.

Using a pretrained model on a dataset of over 200,000 images would be considerably more accurate than our original idea of using a custom dataset. It also would do much of the groundwork for us, since predicting an exercise based on keypoints is much easier than predicting from an image alone. There was one problem; YOLOv8 pose estimation does not have exercise prediction capabilities. It can output a pose estimation, but still cannot predict what type of activity the person is doing.

The team decided to revise to original design of using a single neural network, and instead take an ensemble learning approach to exercise prediction. Ensemble learning involves combining multiple models, often of different types or trained on different subsets of data, to improve overall performance. This can lead to better generalization, robustness, and accuracy compared to using a single model.

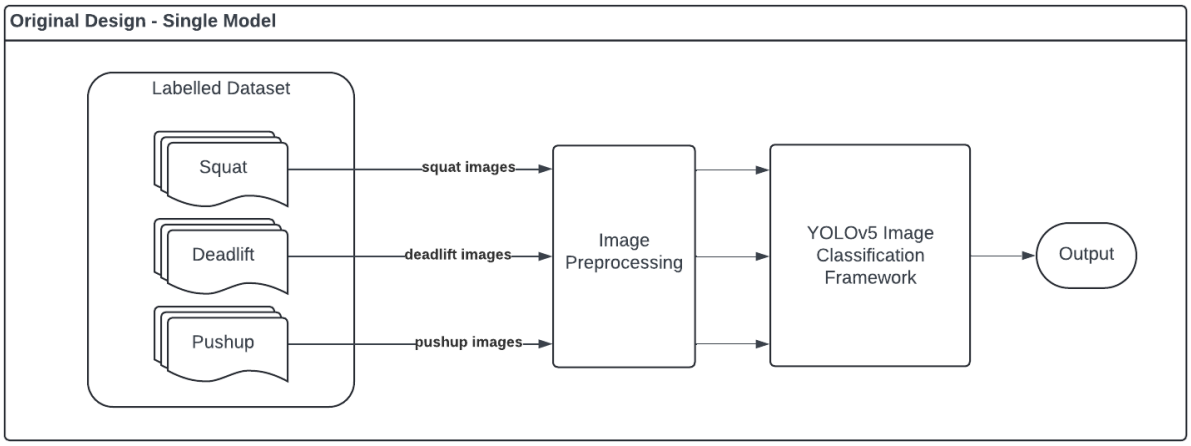

Figure 2: Original Design for a single image classification model.

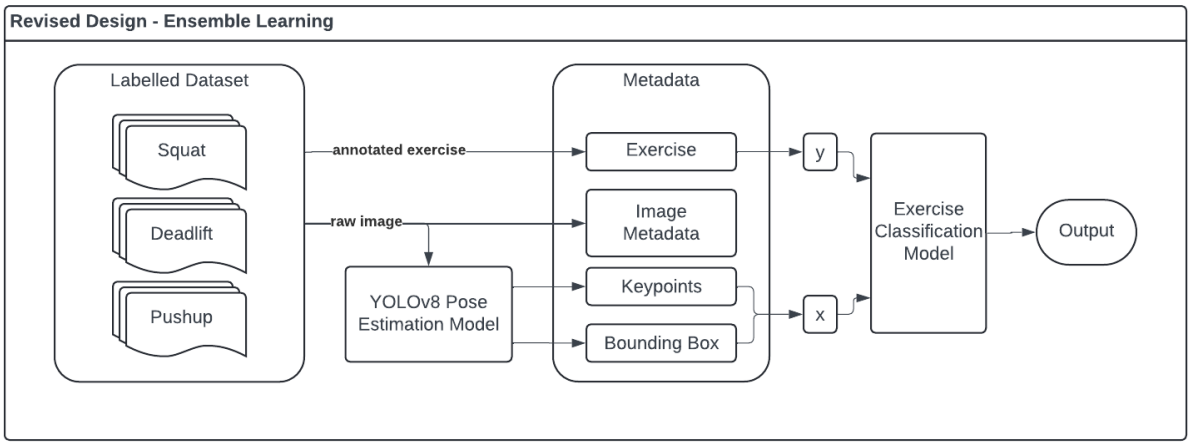

Figure 3: Revised design ensembling a pose estimation model with an exercise classification model.

As shown in the diagram above, the revised design implements the YOLOv8 Pose Estimation model, while still being able to output an exercise classification. Using the outputted key point map and bounding box predictions, a numerical feature set was derived. This feature set would be the input to the exercise classification model. The revised design implements a numerical model on top of the YOLOv8 pose estimation model, that will output an exercise classification.

With this revised design, the complexity of the project was reduced greatly, while still being able to achieve an equal if not greater classification accuracy. If the team decided to continue with the original design, then that would entail addressing problems that are inherent to image classification networks. Since we opted to use the YOLOv8 pose estimation model, we eliminate the need for training an image classification neural network. Instead, we can use simpler classifiers trained on purely numerical training

data (instead of images), such as a logistic regression, support vector classifier (SVC), random forest classifier, or even a multi-layer perceptron (MLP) classifier.

2.5.2 Model Selection

In selecting the appropriate models for exercise prediction, the team considered several factors including the nature of the problem, computational efficiency, interpretability, and classification performance. After evaluating various options, the team decided to compare Support Vector Classifier (SVC), Random Forest Classifier, and Multi-Layer Perceptron (MLP) Classifier due to their suitability for the task at hand.

Option 1: Support Vector Classifier (SVC):

- **Description:** SVM/SVC is a supervised learning algorithm that can be used for classification tasks. It works by finding the hyperplane that best separates different classes in the feature space.
- **Strengths:**
    - Effective in high-dimensional spaces.
    - Memory efficient as it uses a subset of training points in the decision function (support vectors).
    - Versatile as it can handle both linear and non-linear classification tasks through the use of different kernels.
- **Weaknesses:**
    - SVM/SVC can be sensitive to the choice of kernel and its parameters.
    - Computationally intensive, especially for large datasets.
    - Limited interpretability compared to some other models.

Option 2: Random Forest Classifier:

- **Description:** Random Forest is an ensemble learning method that builds multiple decision trees during training and outputs the class that is the mode of the classes of the individual trees.
- **Strengths:**

    - Robust to overfitting due to the ensemble nature of the model.
    - Can handle large datasets with high dimensionality.
    - Provides feature importance scores, allowing for interpretability.
- **Weaknesses:**
    - Can be computationally expensive during training, especially with a large number of trees and features.
    - May not perform as well as SVM/SVC on very high-dimensional or sparse datasets.

Option 3: Multi-Layer Perceptron (MLP) Classifier:

- **Description:** MLP is a type of feedforward artificial neural network consisting of multiple layers of nodes (neurons), where each node is connected to all nodes in the next layer.
- **Strengths:**
    - Suitable for non-linear classification tasks and can learn complex patterns.
    - Flexible architecture allows for customization of the number of layers and neurons.
    - Can capture intricate relationships between features.
- **Weaknesses:**
    - Prone to overfitting, especially with a large number of parameters.
    - Requires careful tuning of hyperparameters such as learning rate, number of hidden layers, and neurons per layer.
    - Computationally intensive, especially for large networks and datasets.

In comparing these models, the team aimed to strike a balance between classification performance, computational efficiency, and interpretability. Each model offers unique advantages and trade-offs, and by evaluating their performance on our task, we can determine the most suitable model for exercise prediction. The following table provides a comparative analysis of the accuracies achieved by each model, which enabled us to make an informed decision regarding model selection.

2.5.3 Model Development Cycle

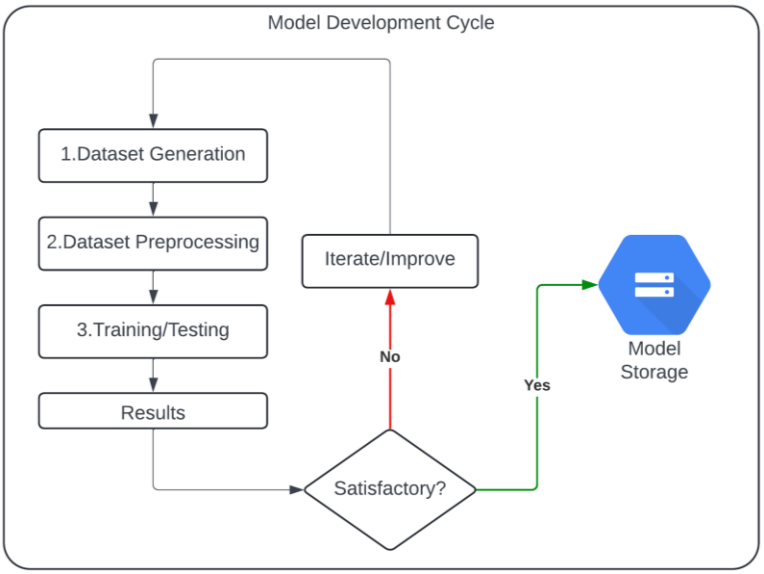

Figure 4: Simplified model development cycle, indicating termination condition.

The model development process was crucial for ensuring that the team achieved the model inference software specification listed in Section 6: Specifications

To achieve our project goal of attaining a 90% accuracy for exercise predictions, our team followed a systematic model development cycle. Step one was to generate a comprehensive dataset encompassing various exercise scenarios. Subsequently, we would meticulously preprocess the dataset to ensure its quality and consistency. The third step involved training and testing our models using different algorithms and techniques. Upon evaluating the results, if the accuracy met our criteria, we proceeded to upload the model to Google Cloud Storage (GCS). However, if the results were unsatisfactory, we iterated through the cycle by refining the dataset through enhancements and adjusting model parameters for better performance. This iterative process enabled us to continually improve our models until achieving the desired accuracy threshold.

2.6 Workout Analysis

The phase of the project involved estimating the caloric output of the exercise. To estimate the caloric output metabolic equivalent of task (MET) values are used [5].

In order to derive caloric output from an exercise's associated MET value, it is multiplied by the amount of time in hours the time is performed and the bodyweight of the person performing the exercise, these. Each exercise has a range of MET values based on the intensity at which it is performed. When uploading a video to the app, the user is asked the intensity of the exercise they performed on a scale of one through ten. The MET value is then calculated based on the user's perceived intensity, which places them in the range of the minimum and maximum MET values for that exercise. The users body weight and the load being used are also collected. The time spent performing the exercise is derived by taking the time between the first and last repetition in the video.

Due to the nature of caloric output for a given exercise being highly dependent on a multitude of factors, such as: genetics, experience performing the exercise, body composition, and heart rate. A lot of these factors are non-quantifiable and thus it is difficult to accurately model caloric output with any real accuracy. The most accurate way this can be tracked right now is using a device like an Apple watch or Fitbit which track heart rate for long periods of time, allowing for them to use the deviation from the user's resting heart rate while performing the exercise to get a better approximation for calorie expenditure. In future implementations of the app direct integration with the Apple watch to provide a more accurate value of calories burned would be possible.

2.7 API and System Integration

The system integration process for our workout companion app involved designing seamless communication between the user interface (UI), the backend API, and the machine learning models responsible for exercise prediction and workout analysis. Here's an overview of the proposed system design.

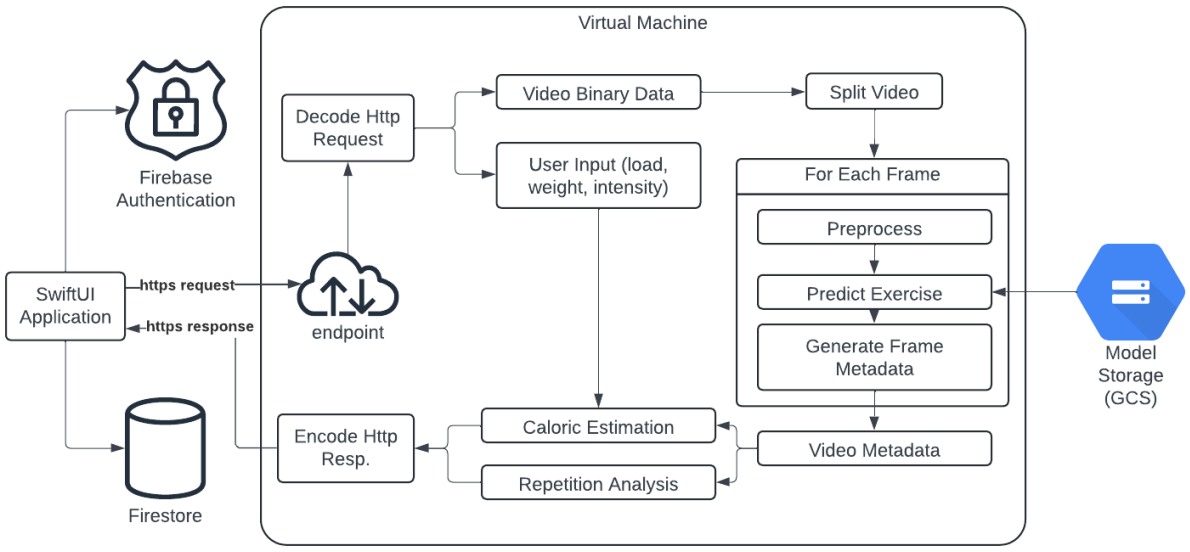

Figure 5: Architecture diagram for the API and Systems Integration infrastructure.

2.7.1 API:

To provide workout analysis and exercise prediction functionalities, we planned to create an API hosted on a virtual machine. This API encapsulates all our business logic, including the machine learning models for exercise prediction, as well as the workout analysis components. The virtual machine hosting the API runs continuously, ensuring availability for processing user requests at any time.

2.7.3 Model Storage and Retrieval

The exercise prediction model developed by our team would be periodically updated with new versions. When a new model version is ready, it is uploaded to Google Cloud Storage (GCS) for storage and easy access. The virtual machine hosting the backend API will periodically check Google Cloud Storage for updates to the exercise prediction model. Upon detecting a new model version, the virtual machine automatically pulls the updated model from GCS and integrates it into the prediction process.

2.7.2 Authentication and Storage:

When a user captures a workout video, they should be able to upload it through the mobile application. The mobile application generates a POST http request containing the video file as binary data, along with any additional user input provided through the UI. In order for this to occur, we needed a way to

authenticate our users so that the API wasn't accessible to the public. For this, we planned to use Firebase Authentication. Firebase Authentication is a service provided by Google Firebase that allows developers to easily authenticate users using email/password, phone numbers, social media logins, and more, while also managing user identities and securely integrating authentication into their apps [6]. The reason we chose to use Firebase, is because of it's compatibility with iOS applications, as well as the fact that it uses the Google Cloud ecosystem, meaning we could also use Firebase for our app storage (workout history, etc.).

By implementing this system integration design, we ensured that users can seamlessly upload workout videos via the mobile application, receive accurate exercise predictions and workout analysis from our backend API, and benefit from the continuous improvement of our machine learning models through dynamic model retrieval from Google Cloud Storage. This cohesive integration enhances the overall user experience and effectiveness of our workout companion app.

2.8 User Interface

The team considered a variety of user interfaces for our project, and ultimately decided that building a mobile application would be the preferred choice. This decision was made for several reasons including:

1. **Ease of use:** Users should be able to upload their own workout videos to our services with ease. To ensure a seamless user experience, the UI should be accessible on the same platform as the device used to record their workout videos. Since most people own a smartphone, mobile devices have become the preferred method for recording personal exercise videos.
2. **Compatibility:** Allowing users to upload just any format of video to our service could produce unexpected results. Since our dataset consists of videos taken from a smartphone, we should ensure that videos uploaded to our service for inference are in the same format.

Since most of our team members owned iPhones, the obvious decision was to opt for a iOS app. Additionally, one of our team members was familiar with iOS development, which reduced the learning curve slightly. The language to be used was SwiftUI [7], which is a declarative framework for building

user interfaces across Apple platforms using the Swift programming language.

The team designed wireframes for the app, although many changes and improvements were made as the UI was being developed.

Section 3: Implementation

3.1 Dataset

The team followed the dataset guidelines listed in 2.4 Data , and began collecting images from two sources:

1. **Self-recorded** – Each team member recorded many videos of themselves performing deadlifts, squats and pushups, making sure to adhere to the guidelines.
2. **Instagram** – Each team member searched Instagram for accounts that post videos of themselves performing deadlifts, squats and pushups, making sure to adhere to the guidelines. We created a list of Instagram accounts that conformed to our needs, and reached out to them asking for permission to use their videos. If they granted us permission, we used a IGLoader, a python package to download their videos and add them to our dataset.

After collecting exercise videos and creating our preliminary dataset, the team utilized Roboflow to create our training image dataset. Roboflow was used to: annotate images, apply preprocessing steps, and split our dataset.

Roboflow was chosen as it is a popular dataset creation platform with a lot of 1st and 3rd party informative media (tutorials available via Youtube). The platform contained a variety of useful features for our project including allowing multiple people to work on the same dataset, tools for distributing annotation tasks, and built-in applicable preprocessing steps. Additionally, hosting a dataset on Roboflow is free as long as your dataset has less than 10,000 images, so there was no harm in initially adopting the platform.

Each video from our collected dataset was uploaded to Roboflow, where it was then split into a series of images using Roboflow's built-in sample tool. A variety of different framerates were selected over the

course of the project. Initially, a framerate of ~10-15fps was chosen, but as the group approached approximately 1000 images for each of our exercise classes—deadlift, squat, and pushup; the frame rate was lowered to between 1-5fps to both increase the diversity of images within our dataset and to avoid hitting the 10,000 image free dataset limit.

After uploading the videos and sampling them into a series of images, the images were assigned to the various group members for annotation. Roboflow projects come with the ability to add team members to a project. Adding a member to a project allows you to assign annotation tasks to these team members. Utilizing this feature, approximately 7500 images were efficiently annotated using Roboflow's included annotation tool—where images were labelled as being one of our exercise classes: deadlift, squat, and pushup; or other.

Following the annotation of our images, approximately 100 of the images were selected and set aside to be used as our validation set. This validation set was chosen to include images of each exercise in the various states one could find themselves in while performing the exercise. Special attention was given to ensure this set had an equal distribution of all exercise classes and that it would include an image of each exercise at the top, middle, and bottom, of a repetition.

The remaining images were then fed to Roboflow's dataset generation tool. This tool allows you to apply a series of preprocessing steps to your dataset hopefully allowing you to make more (a larger dataset) with less (smaller number of images). During the early stages of creating our neural network, the Horizontal Flip preprocessing step was used to turn our initial annotated dataset of ~2500 images into a set containing ~5000 images. However, as we continued to annotate images and our dataset grew to ~7500 images, the use of these Roboflow preprocessing steps ceased again to avoid surpassing the 10,000 image free limit.

Finally, the image training dataset was now complete. Members working on the model would now simply download the most recent version of our dataset from Roboflow before working on our exercise detection neural network.

3.2 Exercise Classification Model

The team implemented the revised design that ensembles the YOLOv8 pose estimation model with our own exercise prediction model. To accomplish this, the team used Google Colaboratory, a python notebook tool used for machine learning and statistical analysis. The most recent version of the dataset would be downloaded from Roboflow, then uploaded to Google Cloud Storage to be used in the python notebooks. The complete architecture of the model development process, from dataset generation to model storage is shown below:

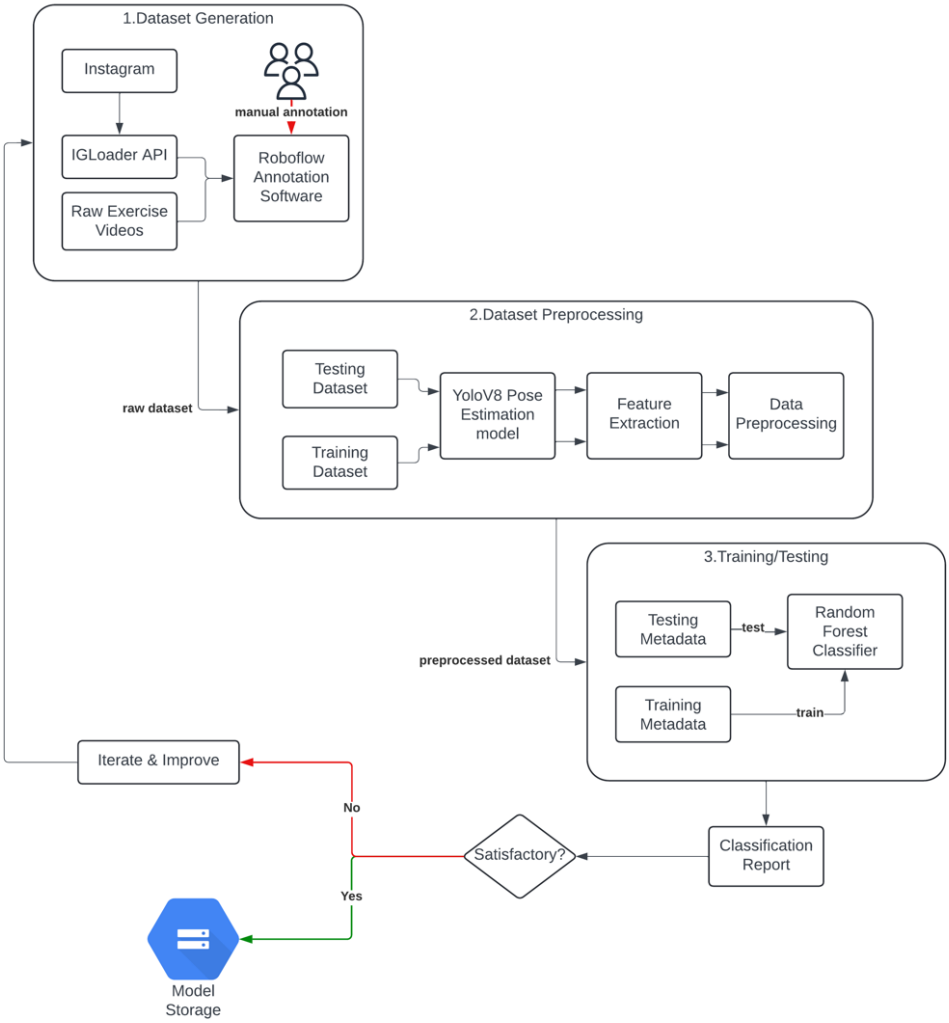

Figure 6: In-depth model development cycle, displaying steps from Dataset Generation to Model Storage.

3.2.1 Pose Estimation and Preprocessing

The following code demonstrates how the YOLOv8 Pose Estimation model was implemented to generate the metadata required for training the exercise prediction model.

```
# Loop through each class folder (deadlift, squat, pushup, other) containing the images
for class_folder in classes:
    class_path = f'{dataset_path}/{class_folder}'
    files = os.listdir(class_path)
    count = 0
    # Loop through each image
    for file_name in files:
        count += 1
        print(f'Processing file {count}/{len(files)} in {class_folder}')
        try:
            # Extract video name, frame number and ID from filename
            video, frame, id = separate_string(file_name)
            # Open the image and predict using the YOLOv8 pose estimation model
            img = Image.open(class_path + '/' + file_name)
            results = model.predict(img, verbose=False)
            # Derive features from results, normalize key points,
            # map them to points relative to bounding box
            row = calculate_rebased_keypoints(key_points=results[0].keypoints.xyn.numpy()[0],
                                              bounding_box=results[0].boxes.xyxyn.numpy()[0])
            # Append row to metadata
            metadata.append([video, frame, id, class_folder] + row)
        except Exception as e:
            # handle other exceptions
            print(f'An error occurred when processing file: {file_name}')
```

Figure 7: Main code block that executes the pose estimation and preprocessing.

The metadata dataframe was then uploaded to Google Cloud Storage (GCS) to be used in training the exercise classification model. As you can see in the code above, there is a function called `calculate_rebased_keypoints(key_points, bounding_box)`. This function is responsible for deriving the features required for the input to our exercise classifier. Originally, the team planned to use the raw key point x and y coordinates as input to our model, however, that approach would be flawed. The problem is that the x and y coordinates outputted from the pose estimation model are relative to the image, not the bounding box of the person object. When we made our dataset, we never set a guideline that the person doing the exercise must be in the "center" of the image. This can cause the exercise model to predict based on the coordinates of the key points, instead of the relationship between

the key points. In order to isolate for this, the key points needed to be mapped relative to the bounding box coordinates, instead of relative to the image.

Another concern with this approach, is that images will be vary in resolution, which means that the bounding boxes and key points will correlate to the resolution of the image. Consider the following scenario

- Camera A, with resolution R_a is used to record exercises mainly in the Squat class.
- Camera B, with resolution R_b is used to record exercises mainly in the Deadlift class.
- If R_a is much greater than R_b , then the exercise classification model could be biased into classifying higher resolution images into the Squat class, and lower resolution images into the Deadlift class.

The figure below shows an example output of the YOLOv8 pose estimation model. The example image has an aspect ratio of 4:3, and a resolution of 400x300 pixels. In the image, a person is performing a pushup. The key points and bounding box of the person are labelled on the frame.

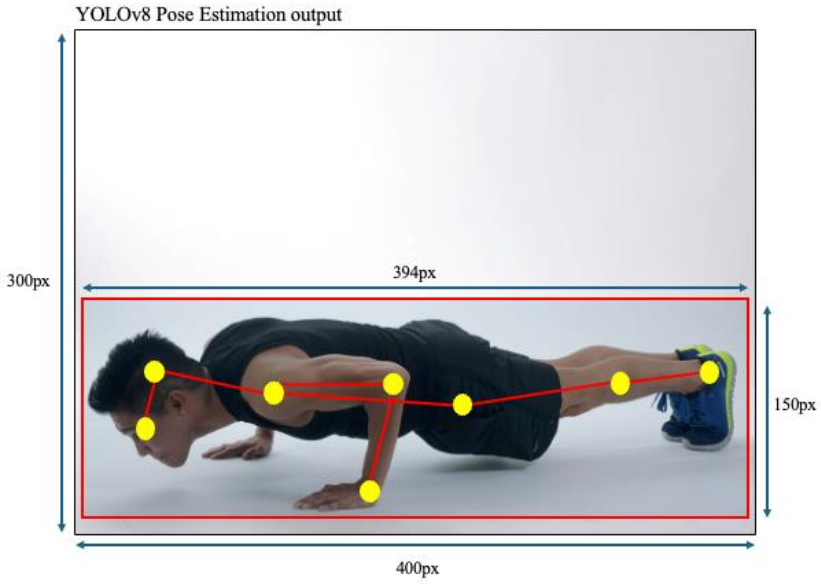

Figure 8: Pose estimation output example, displaying width and height for image and bounding box.

Without any preprocessing, the exercise model would receive the length and width of the bounding box, as well as the xy coordinates of each of the keypoints. The following figure shows what this would look like.

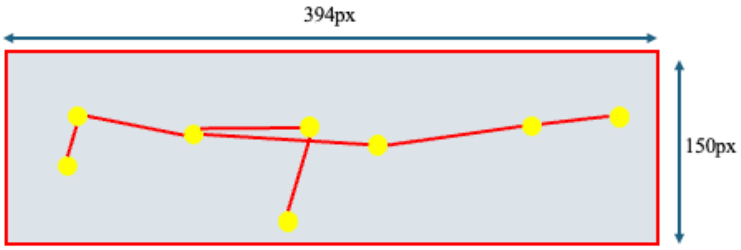

Figure 9: What model would receive without preprocessing.

The team implemented a preprocessing function on the output of the pose estimation model, in order to reduce bias in the exercise classification model. The preprocessing step essentially normalized the bounding box width and height, and then mapped the key points to be relative to the bounding box, instead of the image itself. After this preprocessing step, this is what the exercise model would receive.

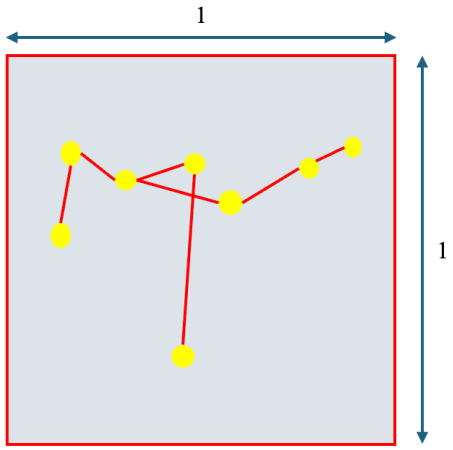

Figure 10: What model would receive with preprocessing.

By eliminating the dependency on image resolution, as well as location within the image, the exercise model is more resistant to being biased. This step increases the robustness of our end-to-end system, through eliminating factors that are not important in classifying the exercise. The preprocessing step helps to prevent accidental biasing of the exercise model by normalizing features that are dependent on, or influenced by, the following factors:

- Dimensions of the user (height, and width)
- Location of the user relative to the frame.
- Distance from the recording device.
- Orientation of the user (horizontal, vertical)

The most important factor is the relationship between the key points, and their positions relative to the center of the body. This is what the preprocessing step aims to isolate.

3.3.2 Model Selection and Parameter Tuning

Upon selecting Support Vector Classifier (SVC), Random Forest Classifier, and Multi-Layer Perceptron (MLP) Classifier as potential models for exercise prediction, we proceeded to implement the design process outlined. We began by training each model on our comprehensive dataset, carefully considering factors such as computational efficiency, interpretability, and classification performance.

The tables below provide a comparative analysis of the performance metrics achieved by each model:

	Duration (s)	Inference Time (s)	Size (KB)	**Accuracy**
SVM	1.046	0.342	422	**0.980**
RF	5.000	0.038	1800	**0.995**
MLP	12.650	0.009	222	**0.996**

Table 1: Table comparing SVM, RF and MLP models using various metrics.

After thorough implementation and evaluation of the design process, our team has decided to utilize the Multi-Layer Perceptron (MLP) Classifier for our exercise prediction task. This decision was based on several key factors:

1. **Inference Speed:** The MLP Classifier demonstrated the fastest inference time among the models evaluated, with an inference time of 0.009 seconds. This attribute is crucial for real-time applications or scenarios where timely predictions are required.
2. **Model Size:** The MLP Classifier exhibited the smallest model size, occupying only 222 KB of disk space when serialized. This compact size is advantageous for deployment and storage considerations, especially in resource-constrained environments.
3. **Accuracy:** While both the Random Forest (RF) Classifier and MLP Classifier achieved high accuracies, the MLP model slightly outperformed the RF model with an accuracy of 0.996. This high accuracy ensures the reliability of predictions, meeting our project goal of attaining a 90% accuracy for exercise predictions.

Comparatively, although the Support Vector Machine (SVM) model showcased faster inference times than the MLP, its accuracy fell slightly short. Additionally, while the Random Forest Classifier exhibited faster inference times than the MLP, its larger model size and marginally lower accuracy made it less favorable for our project requirements.

In summary, the Multi-Layer Perceptron (MLP) Classifier emerged as the optimal choice for our exercise prediction task, offering a balance between inference speed, model size, and accuracy. By selecting this model, we aimed to effectively meet our project objectives while ensuring efficient and accurate predictions for exercise activities.

In selecting the optimal configuration for the Multi-Layer Perceptron (MLP) Classifier in our project, we followed a systematic approach to ensure that the chosen parameters would lead to effective model performance. Firstly, we considered the complexity of the exercise prediction task and the nature of our dataset. Given the need to capture intricate relationships between features, we opted for a multi-layer architecture with hidden layers. We chose the specific configuration of 2 hidden layers, the first containing 100 neurons, and the seconds containing 50 neurons. We chose this configuration based on empirical evidence and experimentation, aiming to strike a balance between model complexity and generalization ability. Additionally, we set the maximum number of iterations to 500 to allow sufficient training time for the model to converge. The choice of these parameters was further informed by considerations of computational resources and time constraints. Finally, we utilized a random seed of 42 to ensure reproducibility and consistency in our experiments. Through this methodical process of parameter selection, we aimed to optimize the MLP Classifier for our exercise prediction task, ultimately contributing to the achievement of our project goal of attaining a 90% accuracy for exercise predictions.

3.3 Workout Analysis

3.3.1 Calorie Counting

The calorie counting component of the project did not vary much from the proposed design. Due to calories being extremely dependent on heart rate, it is hard to provide anything more than an estimation

based on MET values. Using the MET table, we were able to extract the average caloric expenditure equation for each exercise. The function that contains this logic is shown below.

```
def caloricOutput(exercise, total_duration, load_weight_lb, intensity, body_weight_lb):
    out = 0

    if exercise == "Squat":
        out = ((body_weight_lb + load_weight_lb) / 2.2) * ((total_duration / 60) / 60) * (
                5.5 + (intensity / 10) * 2.5)

    elif exercise == "Deadlift":
        out = ((body_weight_lb + load_weight_lb) / 2.2) * ((total_duration / 60) / 60) * (
                3.5 + (intensity / 10) * 2.5)

    elif exercise == "Pushup":
        out = ((body_weight_lb + load_weight_lb) / 2.2) * ((total_duration / 60) / 60) * (
                3.8 + (intensity / 10) * 4.2)

    return out
```

Figure 11: Function that calculates caloric output depending of exercise, using MET values.

3.3.2 Repetition Analysis

In our project, the implementation of repetition analysis involved leveraging data obtained from a pose estimation model to count the repetitions performed during exercises. The focus was primarily on exercises such as deadlifts, squats, or pushups, with the understanding that the same methodology could be applied across these exercises. The data used for repetition counting was the height of the bounding box of the person performing the exercise, as this height oscillates during repetitions, providing key insights into the exercise cycle. The following figure displays a graph of bounding box height vs time, corresponding to a video of a person performing a set of squats.

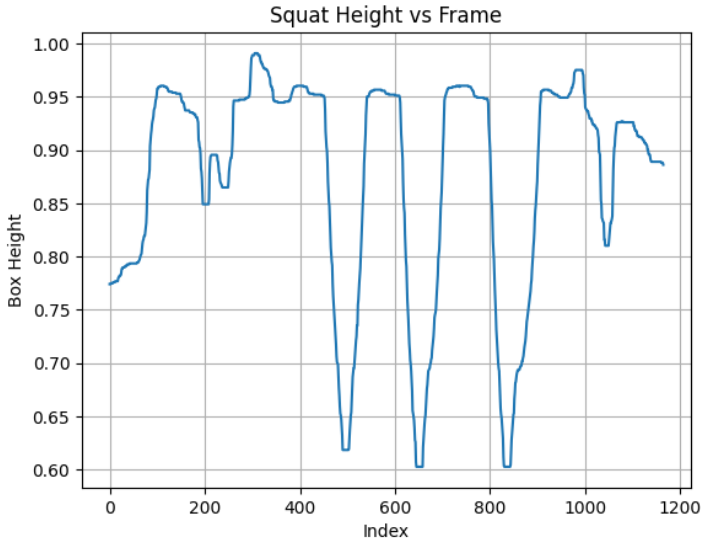

Figure 12: Bounding box height vs. Frame index (time) of a person doing 3 repetitions of a squat.

As you can see in the graph, there are three distinct repetitions. To extract the number of repetitions from this data, we utilized a python script which involved several key steps:

1. **Finding the Most Common Value:** We employed the mode() function from the scipy.stats module to determine the most common height value in the data. This value served as a reference point for establishing thresholds and identifying repetition cycles.
2. **Setting Thresholds**: A threshold was defined as 70% of the most common height value. This threshold served as a reference for distinguishing between the upward and downward phases of repetitions.
3. **Iterating Through Data:** We iterated through the squat height data, analyzing each data point to detect transitions between below-threshold and above-threshold states. This transition indicated the completion of a repetition cycle.
4. **Counting Repetitions:** Repetitions were counted based on the transitions between below-threshold and above-threshold states. Each transition from below-threshold to above-threshold indicated the completion of a repetition. Additionally, we tracked the start and end frames of each repetition to calculate the repetition time.

5. **Extracting Repetition Time:** The repetition time was calculated by determining the difference in frames between the start and end frames of each repetition cycle. Since the data was captured at 30 frames per second, dividing the frame difference by 30 provided the time duration of each repetition.
6. **Calculating Average Repetition Time and Exercise Time**: From the repetition time data, we calculated the average repetition time by averaging the time durations of all repetitions. Furthermore, we calculated the total exercise time by summing the time durations of all repetitions.

By implementing this methodology, we were able to effectively analyze repetitions during exercises, extract important metrics such as repetition time and exercise time, and gain insights into the performance and efficiency of workout routines. This approach provided valuable feedback for optimizing exercise techniques and training regimes.

3.4 API and System Integration

The following architecture diagram describes the overarching connectivity of our different subsystems.

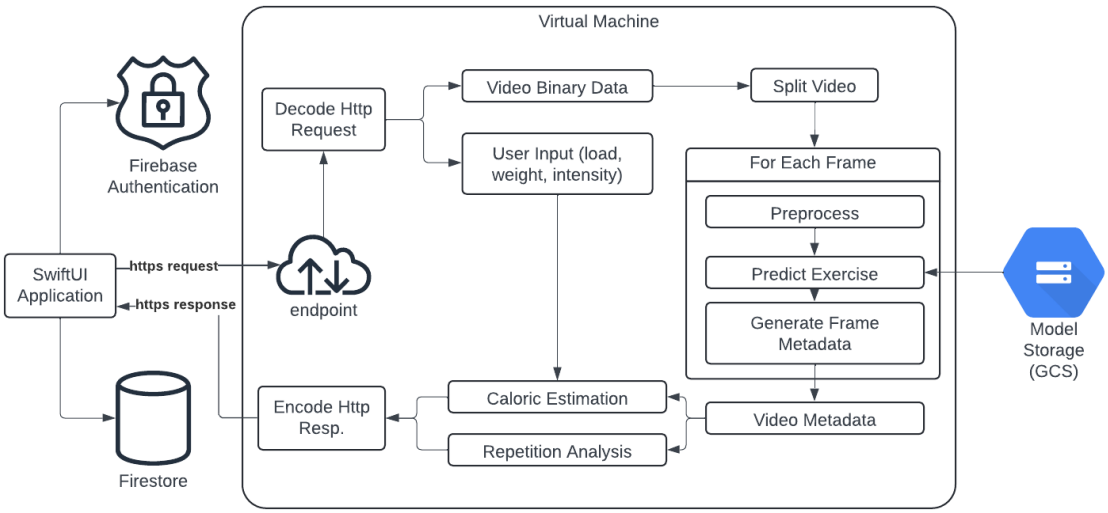

Figure 13: Implemented Architecture for the Systems Integration and API.

Once the dataset generation and exercise prediction stages were complete, the team began to implement the necessary infrastructure to ensure our systems were able to integrate seamlessly. In order for our mobile app to access our exercise prediction and workout analysis components, the team had to containerize our python repository so that it could be executed on a virtual machine. The exercise prediction model and workout analysis components were first combined into a single package. The main code is shown below.

```
metadata = generate_prediction_metadata_from_video(video_path=video_path,
                                                   skip_frames=int(frame_rate_original / frame_rate_processed),
                                                   columns=columns)
# get the most common exercise
exercise = get_exercise(metadata)
# get repetition data
(squat_height_data, active_duration, avg_rep_time, reps) = get_repetitions(exercise=exercise,
                                                                           metadata=metadata,
                                                                           frame_rate=frame_rate_processed)

# calculate total duration of video
total_duration = len(metadata) / frame_rate_processed
# calculate calories burned using MET equations
calories_burned = caloricOutput(exercise=exercise,
                                load_weight_lb=load,
                                total_duration=total_duration,
                                excercise_intensity=intensity,
                                body_weight_lb=body_weight)
# create response dict
response = {
        'repetitions': reps,
        'total_duration': total_duration,
        'active_duration': active_duration,
        'calories_burned': calories_burned,
        'exercise': exercise,
        'intensity': intensity,
        'body_weight': body_weight,
        'load': load,
        'average_rep_time': avg_rep_time,
        'squat_height_data': list(squat_height_data)
    }
```

Figure 14: Main method of python application on virtual machine.

Once the exercise classification and workout analysis components had been combined into a single python package, the team was able to containerize the application for the virtual machine.

The environment was set up to install necessary packages such as ultralytics, as well as sci-kit learn. Once the environment had been set up, the exercise prediction and workout analysis components were installed to the VM. We used a python framework called Flask, to handle endpoint management and request handling. When the VM receives a POST request from our mobile app, the raw video data is decoded and sent to our exercise prediction model, frame by frame. The exercise prediction model then returns the

predicted exercise, and the workout analysis begins. Using the user input, as well as the exercise prediction, the calorie expenditure is calculated. Then, using the metadata generated from our exercise model, the repetition analysis is performed. Once this process has completed, a response is compiled to send back to the mobile app. This is an example response:

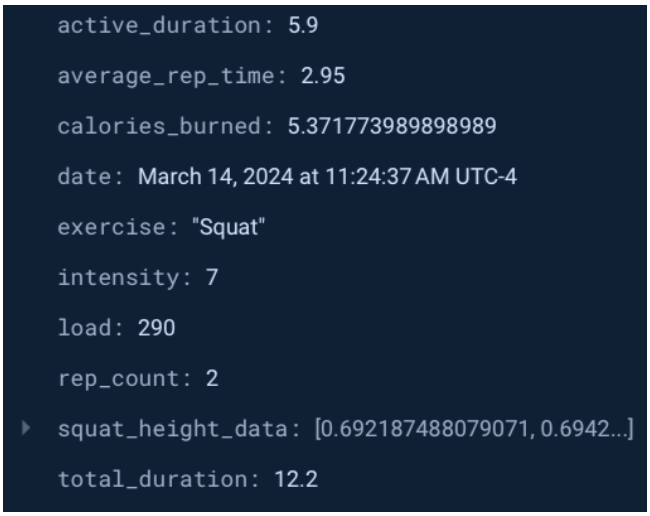

Figure 15: Example API response showing workout analysis data.

Once the response is returned to the mobile app, it is to be stored in a database so that users can view their workout history. We used Firebase for this functionality as it also allows us to authenticate users before accessing our app.

3.5 Mobile App

The mobile app implementation was guided by the following functional requirements:

1. Users must be able to securely log in/sign up to their WorkoutPal app.
2. Users must be able to select videos from their local device, input their workout intensity, body weight, and load weight, and upload their workout to our API for analysis.
3. Users must be able to view their workout summary when the API response is returned.
4. Users must be able to view past workouts and retrieve the workout summary for each of those workouts.

The mobile functionality was divided into 3 tabs, the Home tab, the Upload tab, and the workout History tab.

The Home tab welcomed users to the app, as well as displayed a weekly workout progress indicator for the purpose of motivating users to complete more workouts. The UI of the Home tab is shown below.

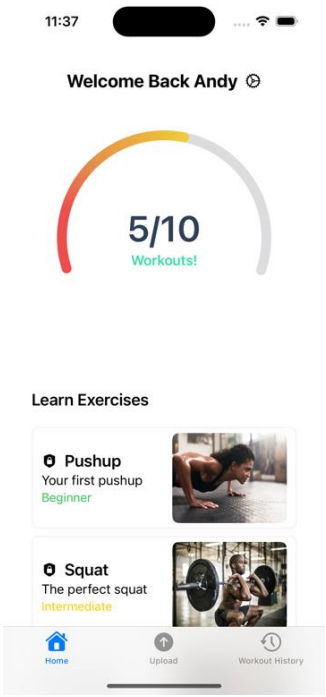

Figure 16: Home Tab UI

The Upload tab was where a user could select a video from their camera roll, then input their intensity, load weight and body weight, then submit their workout for analysis. The UI consists of a "Choose Video" button, that when tapped, prompted the user to select a video from their camera roll. It also consists of 3 sliders corresponding to intensity, body weight, and load weight. When the "Analyse Workout" button is tapped, the inputted information and video are encoded into an http POST request, destined for our API endpoint. The response time of our API varies depending on the length of the video. For this reason, we made this process asynchronous. This means that when the "Analyse Workout" button is tapped, a pop-up is displayed to tell the user that their workout is currently being analysed. When the response is received from the API, another pop up is displayed with the exercise summary screen. This was implemented so that users could still use the app when their video is being processed by our API. The Upload tab and exercise summary UI is shown below.

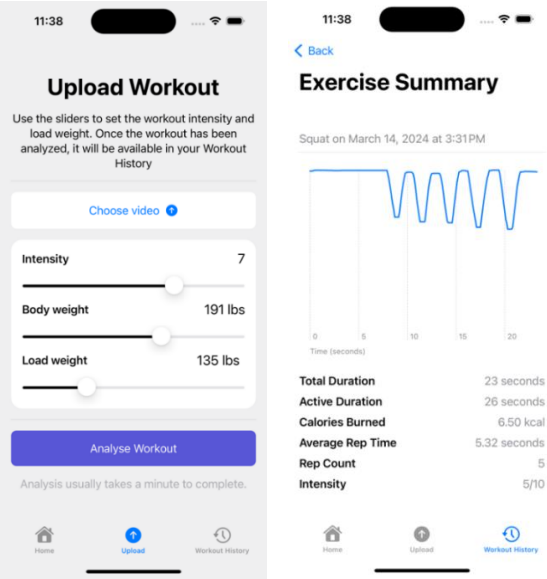

Figure 17: Upload Tab and Exercise Summary UI

The Workout History tab allows users to view their past workouts seamlessly, as well as displaying the exercise summary when tapped. The workout history was pulled from our Firebase backend. The workout history UI is shown below.

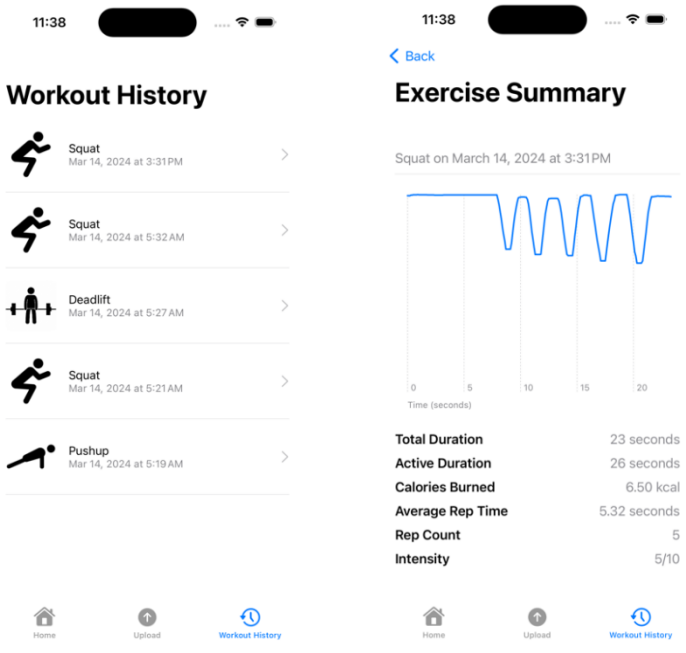

Figure 18: Workout History and Exercise Summary UI

3.6 Budget

Bill Of Materials					
Item no.	Item	Unit	Quantity	Unit cost	Total cost
1	Fake weights	No.	1	$143.63	$143.63
2	PythonAnywhere Subscription	Month	1	$17.25	$17.25

Table 2: Budget table containing list of expenses with associated costs

Section 4: Testing, Evaluation and Validation

4.1 Exercise Predictions

We assessed the accuracy of our exercise prediction model using a small group of four test subjects with diverse body types and instructed them to record themselves performing a variety of exercises including push up, squat, deadlift, and a unique one of their choosing. These videos range from 15s to 25s depending on the exercise and how many repetitions the person attempted. Then the person would record the model's accuracy after it predicts each frame in the video either correctly or incorrectly (this is a percentage). This allows us to thoroughly evaluate the capabilities and limitations of our system. Our volunteers encompass a diverse range of body types, including a small woman with a slim build (Alice), an average height woman with an average build (Hailey), an average height man with an average build (Jacob), and a tall man with a bulkier build (Auston). We structured the test this way to account for all the unique types of users and make sure that our system effectively performs no matter the persons sex, height, or build. Additionally, this test is meant to see how well the model holds up when it's being tested on individuals whose data does not belong to the test set.

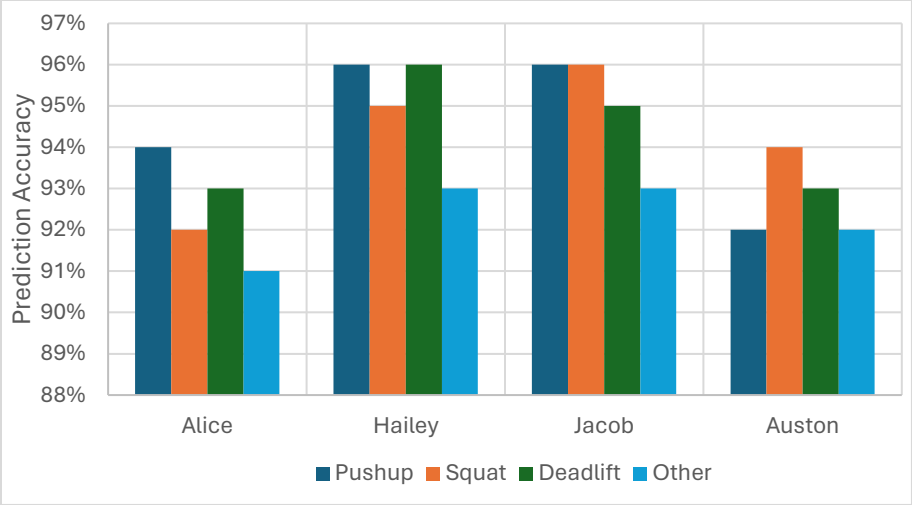

Figure 19: Exercise prediction accuracies for each class in the dataset.

As we can see from the figure above, the people furthest from the average body type are slightly less accurate when it comes to model prediction. This makes sense because the model was trained on exercise videos from people with more average builds. Furthermore, it's important to note that the category of exercises that was predicted the least accurately was the Other class, in other words the random exercise chosen by the testers. This would make sense because the Other class does not have random exercises in the dataset, this is the default class when the model cannot put the exercise into the other three categories. So, the model is less identifying whether the exercise belongs to the Other class, it's more classifying that the exercise does not belong to the other three classes which will inevitably be less accurate. However, these numbers are very high, and this indicates that the model has correctly set the weights and learned the features of the dataset properly.

4.2 Algorithm Testing

Let us now explore some of the model specific statistics. We tested three different learning algorithms to see which one performed the best, these include Random Forest (RF), Support Vector Machine (SVM), and Machine Learning Perceptron (MLP). We then calculated accuracy, precision, recall, and f1-score of each of these algorithms to see which performed the best on our model.

Table 3: Random forest statistics

Exercise Classes	Accuracy	Precision	Recall	F1-Score

Deadlift	99.5%	100%	99.8%	99.9%
Squat	99.5%	100%	100%	100%
Pushup	99.5%	98.5%	100%	99.2%
Other	99.5%	100%	96.0%	97.9%

Table 4: Support Vector Machine statistics

Exercise Classes	Accuracy	Precision	Recall	F1-Score
Deadlift	98.0%	98.7%	99.8%	99.2%
Squat	98.0%	99.5%	100%	99.8%
Pushup	98.0%	95.6%	98.6%	97.1%
Other	98.0%	100%	86.1%	92.5%

Table 5:Machine Learning Perceptron statistics

Exercise Classes	Accuracy	Precision	Recall	F1-Score
Deadlift	99.6%	100%	100%	100%
Squat	99.6%	100%	100%	100%
Pushup	99.6%	98.8%	99.8%	99.3%
Other	99.6%	99.4%	96.5%	97.9%

From the provided data, it's evident that the model exhibited high accuracy across most of the unique algorithms. However, the notable outlier is SVM, displaying the lowest accuracy, precision, recall, and F1-score. RF and MLP demonstrated strong performance overall, with MLP emerging as the clear winner. MLP attained the highest accuracy percentage and achieved near-perfect values for both the Pushup and Other classes, while also demonstrating perfection for both deadlift and squat classifications.

Utilizing Multilayer Perceptron (MLP) for our model proves to be a strategic choice, evident from its exceptional performance across various metrics. MLP excels in handling complex, nonlinear relationships within data, making it well-suited for tasks like exercise classification. Its ability to learn from intricate patterns in the data allows for robust classification of exercise types, as demonstrated by its high accuracy, precision, recall, and F1-score values in our analysis. Moreover, MLP's adaptability and flexibility enable it to effectively handle different exercise classes, accommodating the diverse range of movements and intensities encountered in real-world workout scenarios. Overall, incorporating MLP into

our model enhances its accuracy and reliability, empowering it to provide valuable contributions into exercise classification for users.

4.3 Repetition Counting

One of the key features of our calorie counter app is its precise ability to count the number of repetitions performed during exercises. However, it's imperative to thoroughly test the limitations of this system to identify and rectify any potential errors. Our objective is to assess the accuracy of repetition counting across all three exercises and measure its consistency across different ranges of reps. Moreover, we aim to examine if there's any variability in this data across individuals of different sexes and body types. To conduct this evaluation, we'll enlist two test subjects, Hailey and Auston, who will each perform exercises in sets of four within each range of repetitions, spanning from 1 to 3 reps, 4 to 6 reps, 7 to 9 reps, and 10 to 12 reps. This rigorous testing protocol will enable us to gain insights into the performance and reliability of our repetition counting feature under various conditions.

Table 6: Hailey's repetition counting test results.

Exercise Classes	1 to 3 reps	4 to 6 reps	7 to 9 reps	10 to 12 reps
Deadlift	4 out of 4	4 out of 4	4 out of 4	4 out of 4
Squat	4 out of 4	4 out of 4	4 out of 4	3 out of 4
Pushup	4 out of 4	4 out of 4	4 out of 4	4 out of 4

Table 7: Auston's repetition counting test results.

Exercise Classes	1 to 3 reps	4 to 6 reps	7 to 9 reps	10 to 12 reps
Deadlift	4 out of 4	4 out of 4	4 out of 4	3 out of 4
Squat	4 out of 4	4 out of 4	4 out of 4	4 out of 4
Pushup	4 out of 4	4 out of 4	4 out of 4	3 out of 4

From the tables above, we observe consistency in performance across the exercises for both individuals, with slight variations. Hailey consistently achieved accurate repetition counts across all rep ranges for Deadlift and Pushup, while for Squat, she achieved a slightly lower accuracy in the 10 to 12 rep range. On the other hand, Auston achieved perfect accuracy in Deadlift and Squat across all rep ranges, while showing a minor decrease in accuracy for Pushup in the 10 to 12 rep range.

Our findings reveal impressive and precise repetition counting across a diverse array of reps and exercises. However, as the repetitions extend into the 10 to 12 range, minor discrepancies begin to emerge. This discrepancy can be attributed to the increased duration of the reps, resulting in longer exercise videos with more frames for the model to evaluate and predict accurately. Moreover, there is a potential for human error during these longer repetition exercises, whether due to incorrect form or fatigue. It's essential not to discount any possibilities. Nonetheless, despite these challenges, our results remain highly accurate. This testing phase has underscored that repetition counting maintains precision irrespective of factors such as body type, sex, or height. The only notable limitation arises when the repetition count exceeds 7 to 9 reps.

4.4 Calorie Estimation

The central objective of our project is to attain precise predictions of calorie burn using a deep learning model. Recognized as the gold standard for accurately measuring calories expended during exercise, the utilization of a heart rate monitor is paramount. Leveraging the renowned accuracy of the Apple Watch, equipped with this capability, aligns seamlessly with our testing approach. To comprehensively evaluate the accuracy of our model's calorie burn predictions against this standard, we will enlist an additional four test subjects spanning a spectrum of body weights, heights, and genders. This diverse pool of participants will enable us to gather a substantial volume of calorie data, facilitating thorough comparison and analysis of our model's accuracy relative to the Apple Watch standard of calorie prediction.

Table 8: Calorie Estimation test results

Test Data	Sex	Height (in)	Weight (lbs)	Exercise	Time (sec)	Load Weights (lbs)	Intensity	Apple Watch Calorie Prediction	Model Calorie Prediction
Alice	F	63	109	Squat	26.6	135	Moderate	6.21	6.53
				Deadlift	8.6	250	Vigorous	4.09	4.76
				Deadlift	6.9	135	Moderate	3.56	3.62
				Pushup	8.4	109	Vigorous	2.57	2.63
				Pushup	15.2	109	Light	5.23	6.21
Hailey	F	67	141	Squat	5.7	500	Moderate	5.11	5.71
				Squat	9.9	135	Vigorous	10.56	11.56

				Deadlift	13.3	250	Light	5.89	6.63
				Deadlift	6.5	500	Moderate	5.52	5.75
				Pushup	10.2	141	Moderate	4.81	5.68
Jacob	M	70	183	Squat	11.2	135	Light	2.95	3.43
				Squat	26.6	315	Vigorous	11.23	11.66
				Deadlift	13.2	135	Moderate	6.02	6.41
				Deadlift	8.9	500	Light	6.55	7.34
				Pushup	16.8	183	Vigorous	3.72	3.88
Auston	M	79	220	Squat	5.6	135	Light	1.66	2.19
				Squat	13.3	315	Moderate	5.94	6.04
				Deadlift	4.3	500	Vigorous	4.19	4.45
				Pushup	25	220	Light	7.44	7.92
				Pushup	5.1	220	Moderate	2.22	2.29

The provided data offers valuable insights into the comparison between Apple Watch calorie predictions and model-calculated calorie predictions across a diverse set of exercises performed by various individuals. Within the dataset, we encounter a wide range of exercise types, durations, intensities, and loads, reflecting the complexity of real-world workout scenarios. Upon closer examination, notable variations emerge between the calorie predictions generated by the Apple Watch and our model. Specifically, our model consistently tends to predict slightly higher or significantly higher calorie burn compared to the estimations provided by the Apple Watch, particularly for certain individuals and exercises. This disparity highlights the importance of refining and fine-tuning our model to achieve better alignment with the measurements obtained from the Apple Watch. Additionally, it's important to note that our model utilizes an MRE (Metabolic Rate Estimation) calorie estimation system, which incorporates subjective intensity values, further emphasizing the need for calibration and optimization to enhance accuracy.

4.5 API Latency Testing

In order to test API latency a multitude of videos of different length were sent to the API to be processed and the latency time was recoded. The focus of this test was to gain an understanding of the backend API’s processing time with different file sizes. The results of this test are shown in the following graph below:

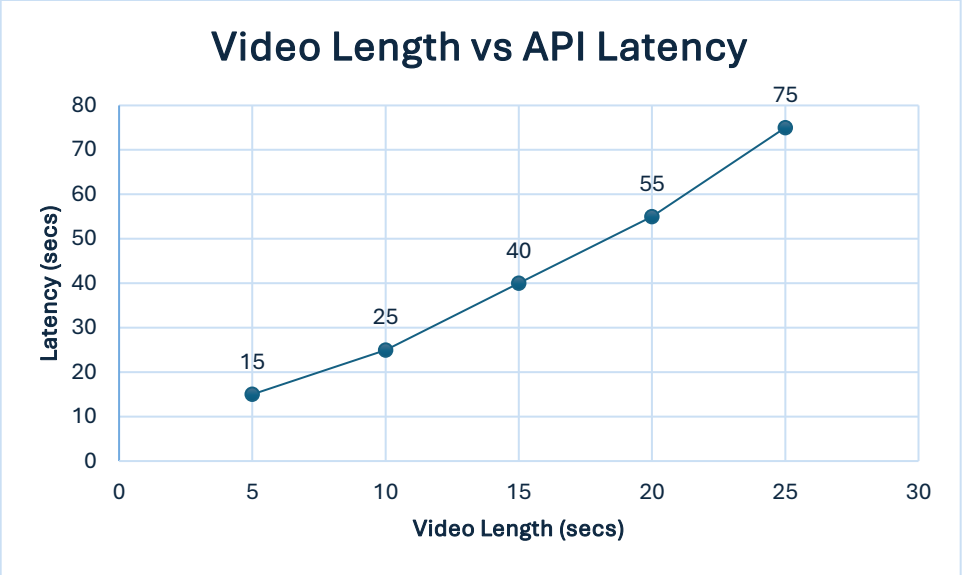

Figure 20: API Latency testing line plot graph.

The results of the API latency test are a slightly exponential, almost linear trend. While the longer the video uploaded is the longer the video will take to upload and process, the caloric output algorithm, the workout analysis algorithm and the API response should take a relatively consistent amount of time regardless of video length. This coincides with the results shown in the chart as most of the latency time takes place in the video upload and processing, however there is a slight decrease in latency per second of video the longer is.

Section 5: Important Factors

Due to this project requiring a low capital investment, the primary stakeholder in this project that was considered when designing this project was the users of the application. The stakeholder's need's that were considered in this project is a simple and easy to use IOS app that allowed for the user to simply upload a video of their workout and input the load used in the video as well as their body weight and receive an analytical report breaking down their workout. With these requirements in mind the app was purposely designed with simplicity in mind, to make the users app interactions as seamless as possible.

The app was also created emphasizing user security and privacy. The app collects data such as user weight and videos of the user exercising. Due to the exercise inference model and the caloric output

algorithm being stored and accessed via a cloud hosted API, it is important to transmit and store data securely. To meet these needs, HTTPS and encryption is used for transmission and storage of data sent to the API. Currently, the app is a prototype and is not live on the app store, and therefore the locally stored portion of the locally stored portion of the app is currently the least secure part of the overarching architecture. If this were to be brought to production, optional security features such as multi-factor authentication, or the use of IOS biometrics to grant access to the account could be implemented to lock unauthorized users out of the app.

Another stakeholder which garnered less attention in the design portion of the app due to the current implementation being a prototype is the development team. The main need of the development team, assuming this is being made available for free and revenue is not a priority, is the minimization of legal implications. A good next step to minimize the legal implications associated with the team would be to incorporate, separating personal liability from the developers, and establish a corporate veil. After that, hiring a lawyer to draft terms and service and a privacy policy is a good next step. This is of benefit to both the user and the developer. On the user side, they can garner a good understanding of exactly how their data is being used. On the developer side, it can allow for liability to be limited if a user were to sue for misuse of data.

Section 6: Specifications

Table 9: Functional, Interface and Performance Specifications, as well as if the group was able to achieve the target.

Specification	**Description**	**Target**	**Tolerance**	**Target Achieved**
Functional Specifications				
1.1 Video Submission	Users should be able to upload exercise videos along with relevant context (age, weight, height, exercise load, repetitions).	Allow users to upload videos within 5 seconds.	±2 seconds	Yes

1.2 Model Inference	The system should accurately predict the exercise class from the uploaded video using pre-trained models.	Achieve an accuracy rate of at least 90% for exercise classification.	±5%	Yes
1.3 Caloric Estimation	The system should estimate caloric expenditure based on the predicted exercise class and user's information.	Provide caloric expenditure estimation within 5% margin of error compared to actual expenditure.	±2%	Yes
1.4 Repetition Counting	The system should accurately count the number of repetitions performed in an exercise video.	Accurately estimate repetition count accurately with 80% accuracy.	±2 counts	Yes
Interface Specifications				
2.1 User Interface	Users should be able to navigate the app easily, authenticate login credentials, and view past workout history.	Ensure app responsiveness with smooth transitions between screens.	N/A	Yes
2.2 Mobile App Interface	The mobile app should have a user-friendly interface with intuitive controls for uploading videos and viewing workout history.	Maintain consistency in design and layout across different mobile devices.	N/A	Yes
2.3 API Integration	The app should integrate seamlessly with backend APIs for model inference and caloric estimation.	Ensure user can still use app while inference in progress. (Asynchronous inference)	N/A	Yes
Performance Specifications				
3.1 Inference Speed	The system should provide quick inference results for uploaded videos to minimize user wait time.	Complete model inference within 30 seconds for a typical exercise video.	±20 seconds	No, virtual machine processing power less than anticipated
3.2 Data Security	User data and video uploads should be securely stored and protected from unauthorized access.	Implement encryption for data transmission and storage using industry-standard protocols	N/A	Yes
3.3 Reliability	The system should be highly reliable, with minimal downtime or errors during operation.	Achieve 99.99% uptime over a 30-day period.	±2%	Yes

The system specifications outline the functional, interface, and performance requirements for the exercise video analysis application. Functionally, users should be able to effortlessly upload videos with accompanying exercise context, predict exercise classes accurately, estimate caloric expenditure within a small margin of error, and count repetitions reliably. Interface-wise, the app should offer smooth navigation, secure authentication, and a user-friendly mobile interface with consistent design elements. Additionally, seamless integration with backend APIs is crucial for uninterrupted user experience. Performance-wise, the system aims for quick inference speeds and robust data security measures. The system demonstrates high reliability with minimal downtime and errors, meeting the specified uptime requirements.

The successfully met all specified targets except for the inference latency. This was due to the limitations of the virtual machine used. The bottleneck arose from the machine's inability to run multiple threads concurrently, restricting the system to sequential frame processing. Consequently, this sequential approach resulted in longer inference times than anticipated. In hindsight, if the project were to be undertaken again, the group would explore parallel processing techniques. By implementing parallel processing, each frame could be processed simultaneously, significantly reducing inference latency and enhancing overall system performance. This approach would leverage the computational resources more efficiently, aligning with the project's objective of providing quick inference results for uploaded videos to minimize user wait time.

Section 7: Conclusions & Recommendations

7.1 Reflection and Recommendations

Through the completion of this project, members of Group 59 learned a variety of technical skills. Members learned the entirety of the development process for creating an application that utilizes a neural network. Some technical skills acquired include: knowing how to create a dataset for a visual input based model, how to create a neural network for a niche application with a relatively limited dataset (<10,000

images) by utilizing a pretrained model, how to virtually host a model and interact with it through a web API, and how to create a UI for a mobile application utilizing a neural network.

By applying these learned skills, Group 59 was able to successfully complete our Deep Learning Calorie Counter application. Achieving our initial goals by producing a mobile app that can: take a video sequence, use a neural network to detect the exercise being performed within the sequence, and then return the prediction as well as some additional workout statistics where it is then stored within the application.

While all our initial goals were met, improvements could still be made to our application. Some ways the group could improve include: fine tune some of the parameters of our exercise detection algorithm to further improve performance, add additional features to our network to reduce the number of user inputted parameters required, and add additional tracked statistics to our mobile app to provide more comprehensive analytics.

Stemming from our accomplishments, if another team was looking to create a similar model for exercise detection, Group 59 would recommend choosing a development path similar to the one mentioned the Design section of this report—mainly the utilization of a pretrained model. As the use of a pretrained model, allowed our group to overcome the limitations of having too small of a dataset and requiring a large amount of computational power. This approach allowed our team to build an effective exercise detection model with low financial investment in a relatively short period of time; all while still being able to produce an accurate and effective model. Therefore, Group 59 would discourage other groups from pursuing alternative development routes such as training a specific exercise detection model from scratch.

7.2 Future Development

Our project has some potential long-lasting impacts beyond this course. Beyond the skills team members acquired during the creation of this project, as mentioned in the Motivation and Background section, our exercise detection application is very complementary to some other existing virtual fitness services such as virtual fitness classes and workouts. As such, our application has some future potential for commercialization, as it is not difficult to imagine a similar application being integrated into these virtual services to provide exercise form correction or provide real-time workout instruction without the need for a live instructor—on top of providing analytics for one's workout.

If our product was to be scaled to a commercial version, it would likely be to complement an existing service. Thus, the previously mentioned features of exercise form correction or real-time workout instruction would need to be researched and explored. This could potentially be done by creating an additional script to monitor the confidence of our exercise detection model, triggering an exercise correction model to provide advice for how to fix your form when the exercise detection algorithm senses a user's form off. Additionally, further investment into the backend of our application would need to be taken to accommodate both this additional model and the larger number of users.

If pursued as theorized, our application would be used to distinguish an existing service offering within the 15.82 billion Online/Virtual Fitness Market (as of 2022) [8]. Providing its users with a more comprehensive user experience by allowing service users to forgo using third party hardware such as an Apple Watch or Fitbit to be able to analyze their personal fitness data. This could potentially give the parent service offering an advantage, as their platform could be seen as being both more comprehensive and having a lower barrier to entry when compared to its competitors.

Section 8: Team Contributions

Table 10: Team member contributions

Name	Overall Effort Expended (%)
Brock Tureski	100
John Turnbull	100
Andy Craig	100
Nathan Goodman	100

References

[1] “Global Health And Fitness Club Market Size, Share 2030 - CMI,” Custom Market Insights, Feb. 2023. https://www.custommarketinsights.com/report/health-and-fitness-club-market/

[2] “Ultralytics YOLOv5 - Ultralytics YOLOv8 Docs,” docs.ultralytics.com. https://docs.ultralytics.com/yolov5/

[3] “COCO - Common Objects in Context,” cocodataset.org. https://cocodataset.org/#home

[4] Ultralytics, “Pose,” docs.ultralytics.com. https://docs.ultralytics.com/tasks/pose/

[5] M. Jetté, K. Sidney, and G. Blümchen, “Metabolic equivalents (METS) in exercise testing, exercise prescription, and evaluation of functional capacity,” Clinical cardiology, vol. 13, no. 8, pp. 555–65, 1990, doi: https://doi.org/10.1002/clc.4960130809.

[6] Google, “Firebase,” Firebase, 2019. https://firebase.google.com

[7] Apple, “Xcode - SwiftUI- Apple Developer,” developer.apple.com. https://developer.apple.com/xcode/swiftui/

[8] “Virtual/Online Fitness Market Growth, Global Forecast Report [2032] | The Brainy Insights,” www.thebrainyinsights.com. https://www.thebrainyinsights.com/report/virtual-online-fitness-market-13940