



BEACONHOUSE NATIONAL UNIVERSITY

ShotSense

PRJ-F23/337
Design Documentation

EXTERNAL SUPERVISOR
Mirza Danial Masood

INTERNAL SUPERVISORS
Huda Sarfraz

GROUP MEMBERS

Umair Ali Khan	F2020-532
Muhammad Fahad	F2020-157

SCHOOL OF COMPUTER & IT
January 2024

Table of Contents

Table of Contents.....	2
Introduction.....	3
Technologies and Development Tools.....	3
Front-End.....	3
Back-End.....	3
Storage.....	3
Machine Learning Technologies.....	3
Services and Cloud Functions.....	4
Video Processing for dataset.....	4
High Level Application Diagram.....	5
Mobile Application Flow Diagram.....	6
System Architecture (C4 Diagram).....	7
Level 1: Software System.....	7
Level 2: Containers.....	8
The Mobile Application Container.....	8
The AI Backend Container.....	8
AI Models Container.....	8
Level 3: Components.....	10
Components of Mobile Application.....	10
Components of AI Backend.....	10
Components of AI Models.....	11
Backend/API Design.....	12
Batch Predictions to Flutter, Backend Design.....	12
ML Design for Shot Classification.....	15
Model Development.....	15
VGG-16.....	15
Google AutoML.....	16
Conclusion.....	16

Introduction

This design document entails the comprehensive overview of the design and system architecture for ShotSense mobile application. The application is a sophisticated platform designed to enhance cricket players' performance by leveraging cutting-edge technologies, including machine learning and computer vision. ShotSense provides users with a holistic experience, from recording and analyzing individual shots to offering personalized insights and recommendations for skill improvement.

In this document, we will delve into the architectural components and the integration of artificial intelligence models that power key features such as shot classification, pose estimation, and ball tracking. The robustness of the design ensures scalability, security, and usability, aligning with the application's core objectives.

Technologies and Development Tools

Front-End

- **Flutter**
A versatile UI toolkit for building natively compiled applications across mobile, web, and desktop from a single codebase.
- **Dart**
The programming language optimized for building mobile, desktop, server, and web applications with Flutter.

Back-End

- **Firebase Firestore**
A NoSQL document database for efficient and scalable data storage, part of the Firebase platform.
- **NodeJS**
A server-side JavaScript runtime for building scalable network applications.

Storage

- **Google Cloud Storage**
A scalable and secure object storage solution for storing and retrieving data on Google Cloud Platform.

Machine Learning Technologies

- **TensorFlow**
An open-source machine learning framework for building and deploying machine learning models.

- **Keras**
A high-level neural networks API, running on top of TensorFlow, facilitating easy and fast prototyping.
- **YOLOv8**
You Only Look Once, a real-time object detection system known for its speed and accuracy.
- **Google Cloud Vertex AI**
A managed machine learning (ML) platform for building, training, and deploying models on Google Cloud.
- **Google AutoML**
A user-friendly tool that simplifies the process of training machine learning models without extensive expertise.

Services and Cloud Functions

- **Google Firebase**
A comprehensive mobile and web application development platform offering various services like authentication, real-time databases, and more.
- **Google Cloud Functions**
Serverless functions that run in response to events, automatically scaling with the number of executions.

Video Processing for dataset

- **OpenCV**
An open-source computer vision and machine learning software library for image and video processing.
- **Ffmpeg**
A multimedia processing tool that handles multimedia data such as audio and video
- **Microsoft ClipChamp**
A video processing tool for editing and compressing videos.

High Level Application Diagram

A High level Diagram showing complete working of the entire application and its related backend through which a user would get classification for their shot played. As a user upload videos of them playing cricket the video gets uploaded to the cloud. Using node server and google's vertex AI we use batch prediction for the videos of complete over(6 balls) recorded by the user. The videos are processed by the shot classification AI and the predictions for each ball is stored in a firebase firestore collection from where the prediction for each shot is displayed to the user in a user-friendly format

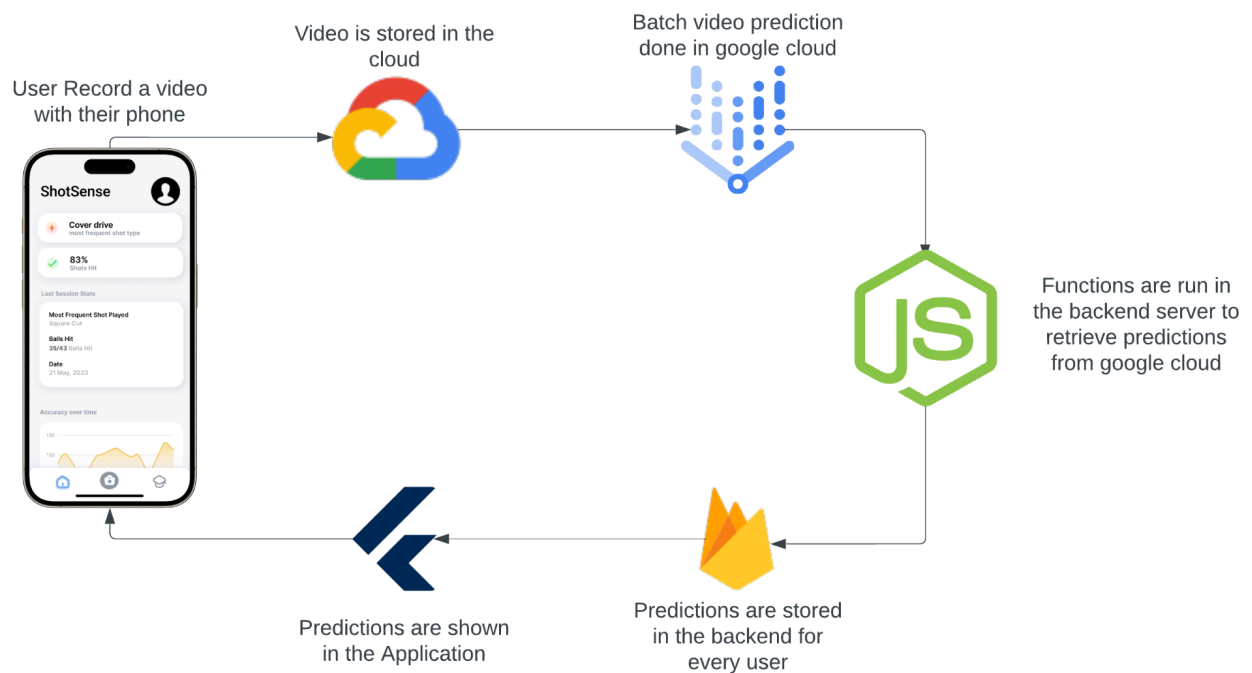


Figure 1 High Level Diagram showing all the processes when a video is uploaded

Mobile Application Flow Diagram

The application operates seamlessly, offering users a comprehensive platform for analyzing their cricket shots. Upon video upload to the cloud, a robust backend, powered by a Node server and Google's Vertex AI, orchestrates the processing through batch prediction of vertex ai. This AI system discerns shot classifications from videos capturing an entire over (6 balls) played by the user. Using shot classification AI, predictions for each ball are meticulously generated and stored in a Firebase Firestore collection. The user is then presented with a user-friendly display, showcasing detailed shot predictions derived from the processed videos

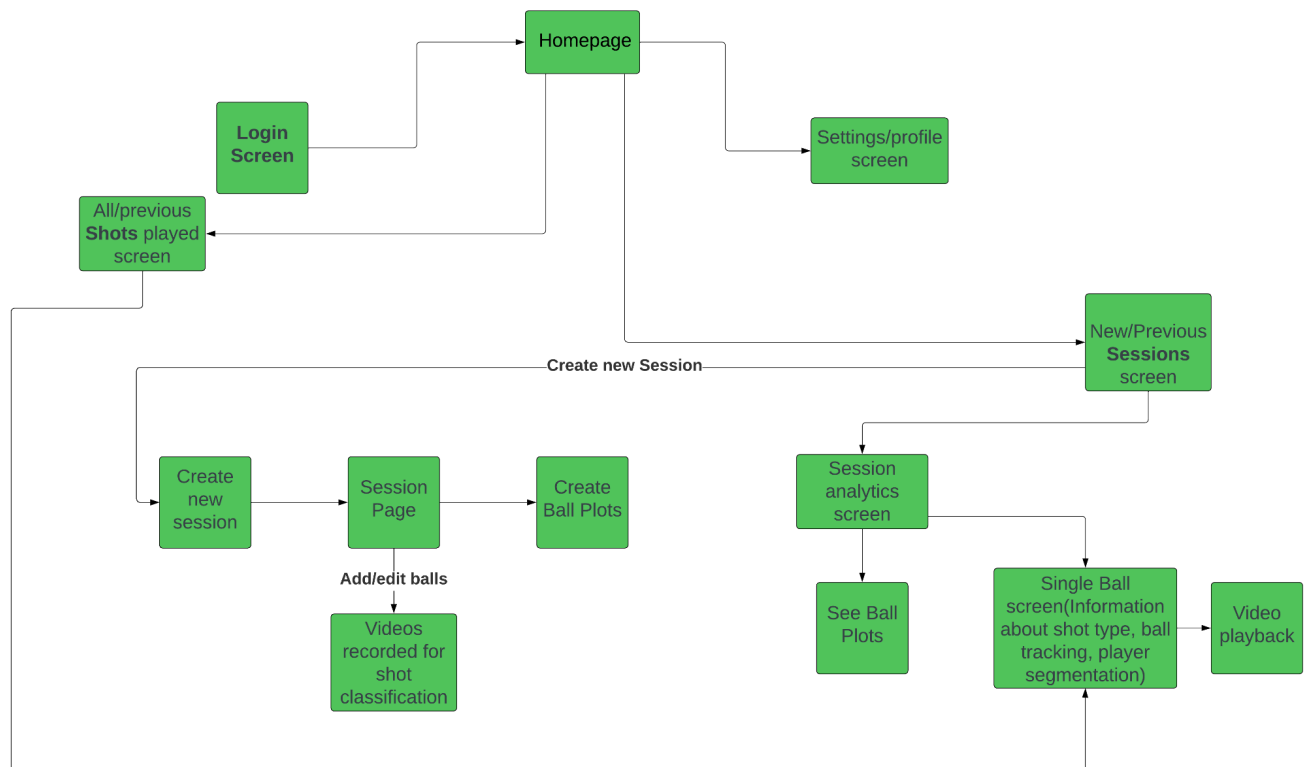


Figure 2 Application Flow Diagram

System Architecture (C4 Diagram)

The C4 architecture diagram breaks down our project into three main levels: software system, container, and components. Looking at the software system level, we have a snapshot of the entire project, mainly divided into two important parts: the core application and the Backend for AI.

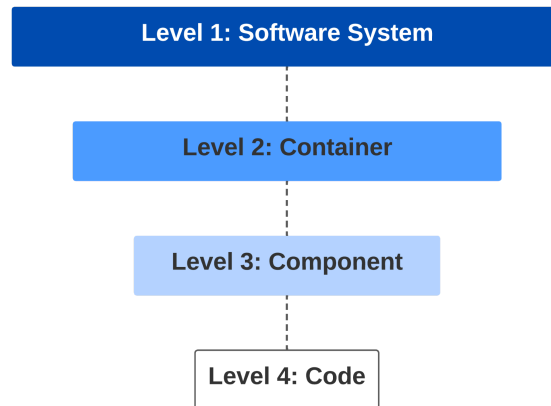


Figure 3

Level 1: Software System

Software system shows an overview of the project separated by application. The Backend for AI acts like a bridge, making sure the AI model and the main application can work together smoothly. This diagram helps us see how everything fits together in shotsense.

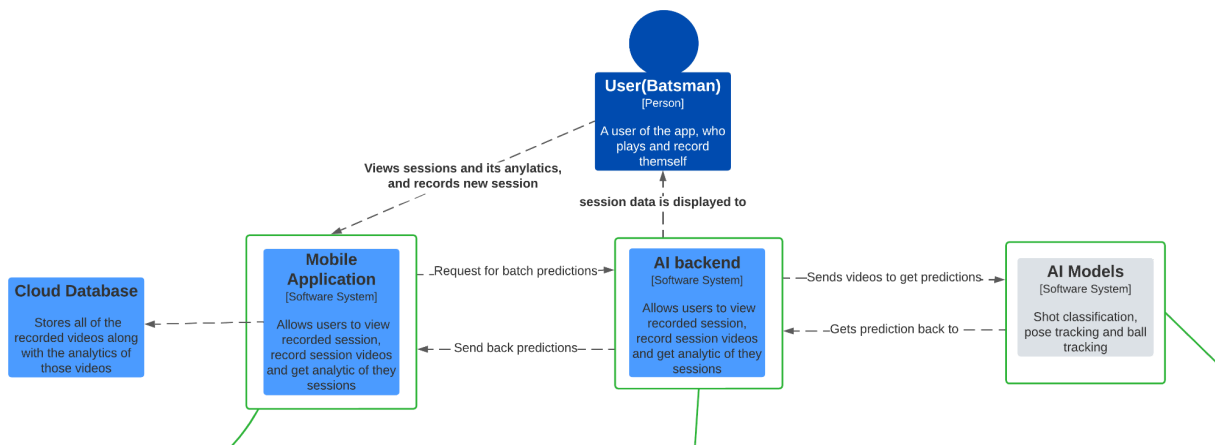


Figure 4 The software system

Level 2: Containers

The container level of our system architecture provides insight into the internal mechanisms of the mobile application, AI backend, and AI models.

The Mobile Application Container

The Mobile Application (refer to Figure 5) functions as the user interface, allowing users to record and upload videos. It captures video data from the user's device, forwards it to the AI backend for processing, and stores the predictions in the Firebase database.

The AI Backend Container

The AI Backend (see Figure 6) is composed of a node server that handles requests from the mobile application. It creates a JSON Lines file, containing URIs of all the videos uploaded to cloud storage. This file is then utilized to generate predictions for the requested videos.

AI Models Container

AI Models (depicted in Figure 7) showcase various models integral to the application. Pose tracking enhances video playback by illustrating body movements of the player, complemented by player segmentation (refer to the implementation document for details). Ball tracking is employed to generate ball plots and heatmaps for session analysis.

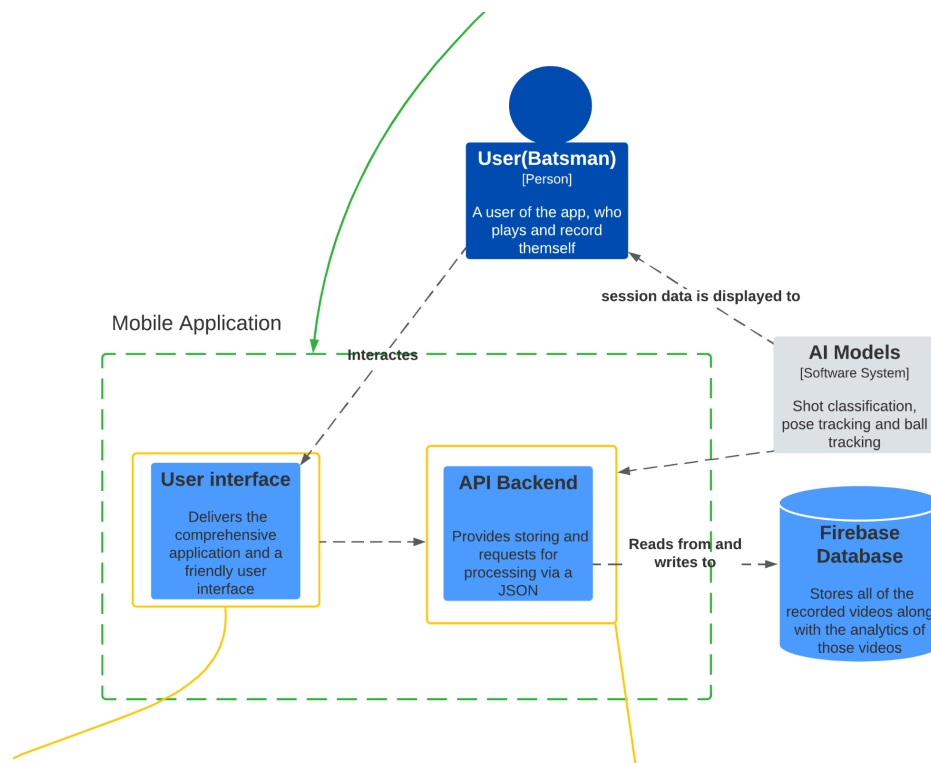


Figure 5 Mobile Application Container

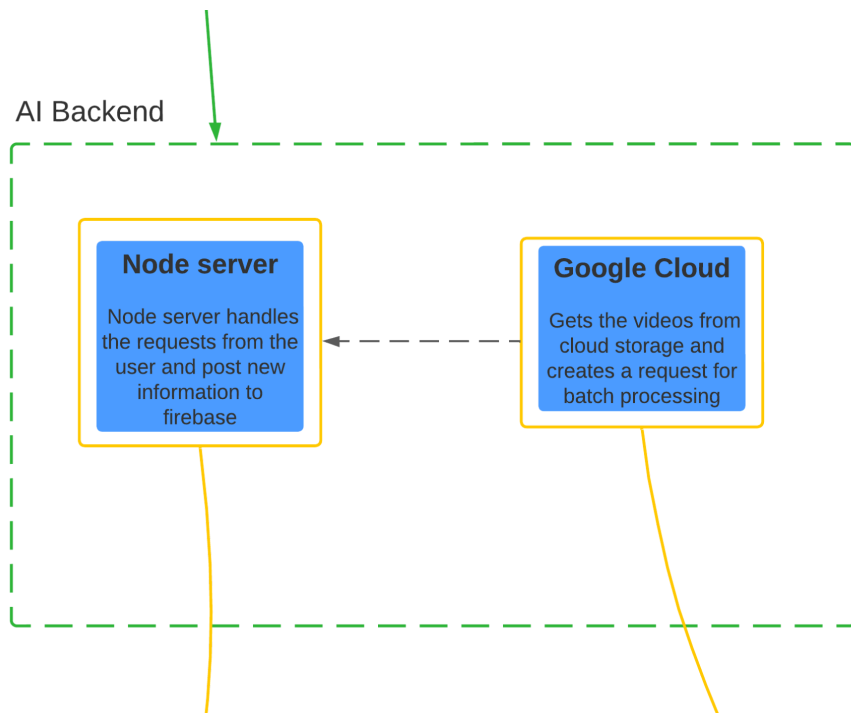


Figure 6 AI Backend Container

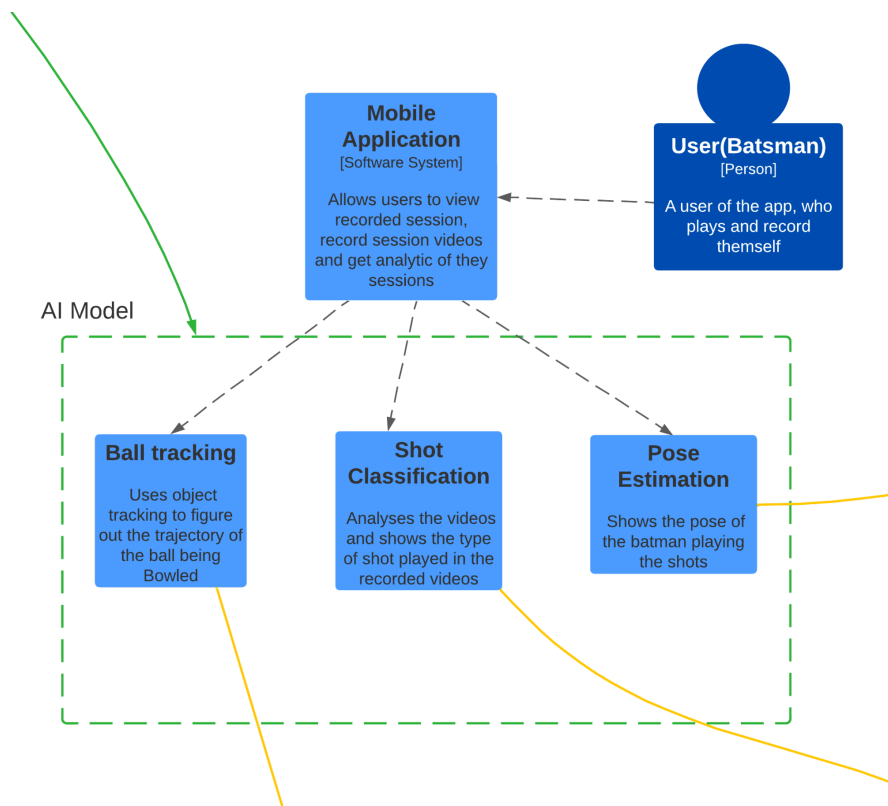


Figure 7 AI Model Container

Level 3: Components

Components delve into the intricacies of each container, shedding light on the fundamental functionalities that define their core operations. This detailed exploration aims to provide a comprehensive understanding of the specific roles and interactions within the Mobile Application, AI Backend, and AI Models containers.

Components of Mobile Application

Figure 7 shows how videos recorded are processed through the AI models to then show the results in each session. Mobile application also houses the process of storing user stats in the database

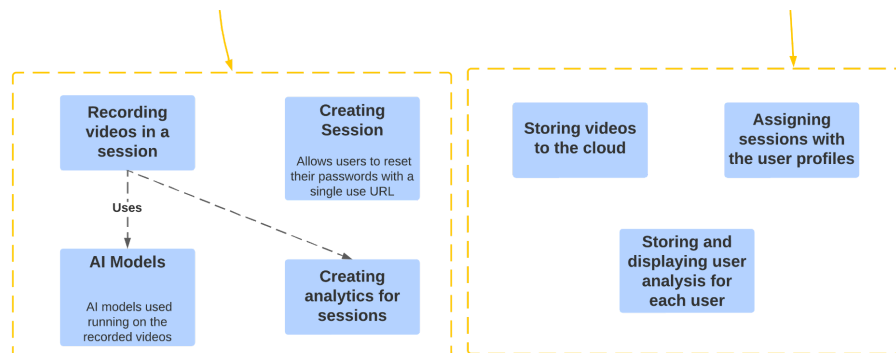


Figure 8 Components of Mobile Application Container

Components of AI Backend

The figure below shows the process of getting URIs and creating JSON Lines file for the requested videos and eventually getting the batch predictions from vertex AI and sending the predictions back to the firebase database

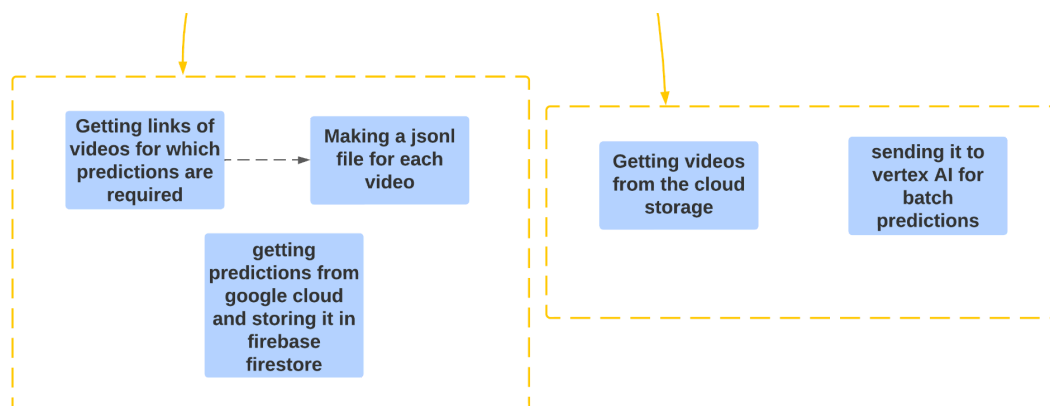


Figure 9 Components of AI Backend Container

Components of AI Models

Show all the AI models in shotsense and use the videos uploaded creating pose tracking and player segmentation. Another AI model showing the shot predictions and finally a model that uses ball tracking to create ball plots for a session

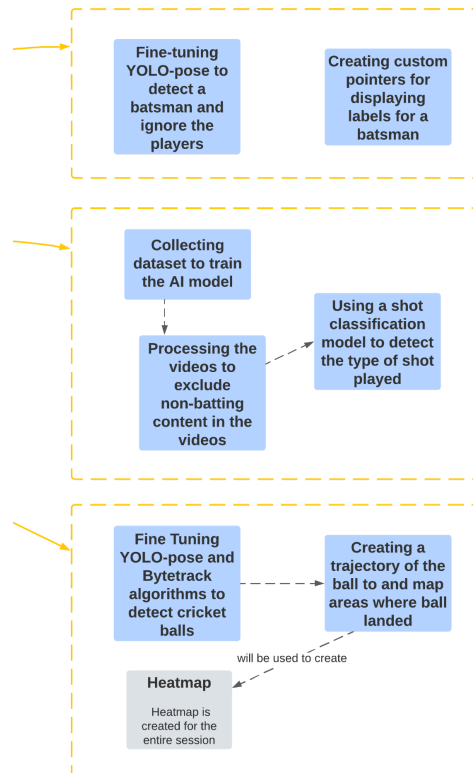


Figure 10 Components of AI Models Container

Backend/API Design

Batch Predictions to Flutter, Backend Design

The backend design for the shot classification model involves a sophisticated sequence of interactions across various project components, seamlessly integrating storage, databases, AI models, and the mobile application interface. At the heart of this architecture, a main bucket serves as the project's core repository, while individual folders in that bucket are dynamically generated for each user and their sessions within Google Cloud Storage during the signup process. Upon video uploads, a dedicated cloud function is triggered, producing JSON Lines files for each video, subsequently transmitted to Google's Vertex AI for processing. The AI model generates predictions, organized into a predictions folder in JSON Lines format. Another cloud function is then invoked to persist these predictions into a Firebase Firestore collection specific to each user. Finally, this data is presented in the Flutter app, delivering a seamless and informative experience to users keen on exploring and understanding their cricket shot predictions.

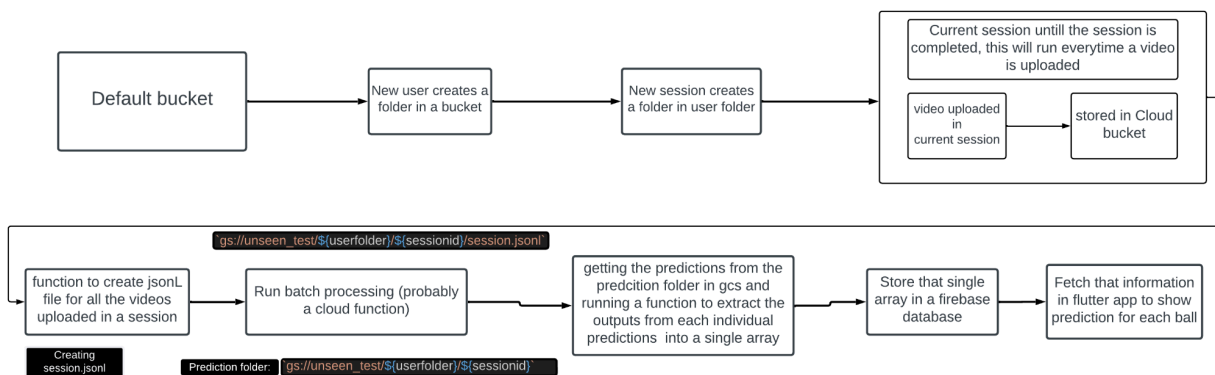


Figure 11 Complete backend flow diagram from a video being uploaded to getting predictions

```
const batchPredictionDisplayName = "backend_test";
const modelId = "7361324905960308736";
const gcsSourceUri = `gs://unseen_test/${userfolder}/${sessionid}/session.jsonl`;
const gcsDestinationOutputUriPrefix = `gs://unseen_test/${userfolder}/${sessionid}`;
const project = "shotsense-f8806";
const location = "us-central1";
const aiplatform = require("@google-cloud/aiplatform");
const { params } = aiplatform.protos.google.cloud.aiplatform.v1.schema.predict;

const { JobServiceClient } = require("@google-cloud/aiplatform").v1;

const clientOptions = {
  apiEndpoint: "us-central1-aiplatform.googleapis.com",
};
```

Figure 12 Defining parameters for batch prediction

```

const jobServiceClient = new JobServiceClient(clientOptions);

async function createBatchPredictionJobVideoClassification() {
  const parent = `projects/${project}/locations/${location}`;
  const modelName = `projects/${project}/locations/${location}/models/${modelId}`;

  const modelParamsObj = new params.VideoClassificationPredictionParams({
    confidenceThreshold: 0.87,
    maxPredictions: 1000,
    segmentClassification: true,
    shotClassification: true,
    oneSecIntervalClassification: true,
  });

  const modelParameters = modelParamsObj.toValue();

  const inputConfig = {
    instancesFormat: "jsonl",
    gcsSource: { uris: [gcsSourceUri] },
  };
  const outputConfig = {
    predictionsFormat: "jsonl",
    gcsDestination: { outputUriPrefix: gcsDestinationOutputUriPrefix },
  };
  const batchPredictionJob = {
    displayName: batchPredictionDisplayName,
    model: modelName,
    modelParameters,
    inputConfig,
    outputConfig,
  };
  const request = {
    parent,
    batchPredictionJob,
  };

  const [response] = await jobServiceClient.createBatchPredictionJob(request);

  console.log("Create batch prediction job video classification response");
  console.log(`Name : ${response.name}`);
  console.log("Raw response:");
  console.log(JSON.stringify(response, null, 2));
}

createBatchPredictionJobVideoClassification();

```

Figure 13 Batch Prediction Node.js Client

Utilizing the Node.js client library, the script configures and submits the job, specifying a pre-trained model identified by `modelId`. Parameters for the video classification model, including confidence threshold and maximum predictions, are set in `modelParamsObj`. Input data, formatted as a JSON Lines file (JSONL) in a Google Cloud Storage bucket, undergoes classification, and the results are stored in a designated location.

```

{
  "instance": {
    "content": "gs://unseen_test/cover_01.mp4",
    "mimeType": "video/mp4",
    "timeSegmentStart": "0.0s",
    "timeSegmentEnd": "Infinity"
  },
  "prediction": [
    {
      "id": "7136266145748746240",
      "displayName": "cover",
      "type": "segment-classification",
      "timeSegmentStart": "0s",
      "timeSegmentEnd": "1.100s",
      "confidence": 0.9900411
    },
    {
      "id": "7136266145748746240",
      "displayName": "cover",
      "type": "shot-classification",
      "timeSegmentStart": "0s",
      "timeSegmentEnd": "1.120s",
      "confidence": 0.9900411
    },
    {
      "id": "7136266145748746240",
      "displayName": "cover",
      "type": "one-sec-interval-classification",
      "timeSegmentStart": "0.640s",
      "timeSegmentEnd": "0.640s",
      "confidence": 0.9900411
    }
  ]
}

```

Figure 14 Prediction in form of JSON output for single video

The JSON output from the `createBatchPredictionJobVideoClassification` function provides predictions for a video instance named "cover_01.mp4." The predictions include multiple classifications, such as segment classification, shot classification, and one-second interval classification, all indicating the presence of a "cover" shot type. The predictions include specific time segments within the video, their corresponding confidence scores, and the unique identifier for each classification. The high confidence scores, particularly around 99%, signify the model's strong confidence in identifying the "cover" shot type during various intervals throughout the video. These detailed predictions offer valuable insights into the model's performance and contribute to the comprehensive shot analysis within the ShotSense application.

ML Design for Shot Classification

In the development of our application, the initial focus was on implementing the shot classification model, a pivotal component of our machine learning architecture. This model is designed to analyze videos, determining and classifying the specific type of shot depicted. The emphasis during the implementation phase has been on ensuring optimal performance through careful attention to the preparation of both training and testing datasets.

Model Development

In the model development phase, our primary goal is to create a reliable and effective system for categorizing cricket shots based on user-uploaded videos. This fundamental aspect of the application allows users to gain valuable insights into their gameplay by understanding the types of shots they play. The following sections provide an in-depth exploration of the specific machine learning techniques employed, including the utilization of pre-trained models such as VGG16 and Google's AutoML. The focus remains on delivering a practical and accurate shot classification model that enhances the overall user experience.

VGG-16

Leveraging pre-trained models like VGG16 is a strategic choice in our pursuit of shot classification accuracy. VGG16, having excelled in image recognition on ImageNet, provides a robust foundation for understanding intricate features in diverse datasets. By incorporating this proven architecture into our model, we tap into its ability to recognize patterns, streamlining the learning process for shot-specific nuances.

To tailor VGG16 to cricket-shot dynamics, we introduced a personalized touch. Our model enhancement involves integrating a custom fully connected layer and opting for a Gated Recurrent Unit (GRU) layer over the conventional LSTM. This modification caters to the unique demands of cricket shots, ensuring that the shot classification model not only benefits from VGG16's pre-learned features but also aligns optimally with the complexities of cricket gameplay, delivering precise insights efficiently.

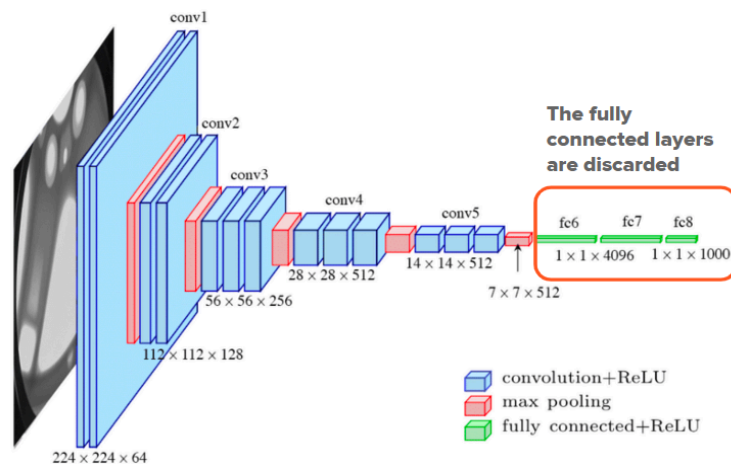


Figure 15 VGG-16 with its hidden layers. FC was discarded for our customized layers

Google AutoML

Google AutoML serves as an invaluable tool, offering an accessible and efficient platform for training machine learning models without the need for extensive expertise. By leveraging AutoML's user-friendly interface, we seamlessly integrated our dataset, allowing for straightforward customization of model parameters and training processes. This not only accelerates the development lifecycle but also democratizes machine learning capabilities, enabling us to harness the power of sophisticated models with ease. With Google AutoML, our shot classification model benefits from advanced training techniques and optimizations, fostering a user-centric approach to model development and achieving reliable outcomes in the dynamic context of cricket shot analysis.

Training

TRAIN NEW MODEL

REFRESH

LEARN

TRAINING PIPELINES

CUSTOM JOBS

HYPERPARAMETER TUNING JOBS

NAS JOBS

Training pipelines are the primary model training workflow in Vertex AI. You can use training pipelines to create an AutoML-trained model or a custom-trained model. For custom-trained models, training pipelines orchestrate custom training jobs and hyperparameter tuning with additional steps like adding a dataset or uploading the model to Vertex AI for prediction serving. [Learn more](#)

Region

us-central1 (Iowa)

Filter

Enter a property name

Name	ID	Status	Job type	Model type	Duration	Last updated	Created	Ended	Labels
Shot_Classification_v2	2570604515828957184	Finished	Training pipeline	Video classification	1 hr 43 min	Dec 20, 2023, 9:38:15 PM	Dec 20, 2023, 7:54:29 PM	Dec 20, 2023, 9:38:15 PM	—
Shot_Classification_v1-20231214-200814	3566713672082391040	Finished	Training pipeline	Video classification	2 hr 3 min	Dec 14, 2023, 10:12:07 PM	Dec 14, 2023, 8:08:15 PM	Dec 14, 2023, 10:12:07 PM	—
Shot_Classification_v1	6899799608801624064	Finished	Training pipeline	Video classification	1 hr 48 min	Dec 14, 2023, 7:54:08 PM	Dec 14, 2023, 6:05:08 PM	Dec 14, 2023, 7:54:08 PM	—

Figure 16 Training pipeline for AutoML model

Conclusion

In summary, this design document has provided a comprehensive overview of the ShotSense mobile application's system architecture, emphasizing its use of advanced technologies like machine learning and computer vision to enhance cricket players' performance. The architectural components, including shot classification, pose estimation, and ball tracking, have been intricately woven together to ensure scalability, security, and usability. Technologies such as Flutter, Dart, Firebase, Node Js, and machine learning frameworks like TensorFlow, Keras, YOLOv8, Google Cloud Vertex AI, and Google AutoML form the backbone of the application, facilitating a seamless user experience from shot recording to personalized skill improvement insights.

The integration of high-level application and mobile application flow diagrams, along with the C4 architecture diagram, elucidates the cohesive functioning of the mobile application, AI backend, and AI models. The backend/API design underscores a sophisticated flow of data, seamlessly connecting storage, databases, AI models, and the mobile application interface. In the ML design for shot classification, the strategic use of pre-trained models like VGG16 and Google AutoML demonstrates a commitment to accuracy, with VGG16 customized for cricket-specific nuances. Overall, this design document serves as a clear blueprint, outlining the architectural decisions and progress made in shaping ShotSense into an advanced, user-centric platform for cricket enthusiasts.