ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Computer Vision

REAL-TIME EXERCISE RECOGNITION AND REPETITION COUNTING FROM MONOCULAR GYM VIDEO

CANDIDATE

Nicolas Ivan Cridlig

SUPERVISOR

Prof. Stefano Mattoccia

CO-SUPERVISOR

Dott. Enrico Mannocci

Academic Year 2025-2026

Session 5

Abstract

This thesis presents the design, development, and evaluation of a real-time system that classifies gym exercises and counts repetitions from a single monocular camera, without wearable sensors or dedicated hardware beyond a compute backend. While pose estimation, activity recognition, and repetition counting have each been studied independently, no existing system integrates all three into a real-time pipeline that operates on commodity security cameras under realistic gym conditions.

The pipeline chains three models. MediaPipe extracts 2D body landmarks from monocular RGB video, a Temporal Convolutional Network (TCN) classifies exercises across 16 categories from joint-angle time series, and RepNet counts repetitions in a class-agnostic manner. We compare five classification approaches, from scikit-learn baselines to a custom TCN. The TCN achieves 95.82% test accuracy and, unlike the baselines, produces temporally stable predictions suitable for real-time display. The complete pipeline processes each frame in 82 ms on a consumer laptop and is deployed as a containerized web application. In end-to-end evaluation on 81 ground truth exercise sets, the system detects 72.8% of exercises (F1 = 83.7%), correctly classifies 88.1% of detected events, and counts repetitions with a mean absolute error of 2.1. A dual-camera experiment reveals that viewpoint is the primary limiting factor, with detection dropping from 70% to 40% when the camera moves from an elevated angle to a frontal position. The system currently processes one user at a time, with identity detection and multi-person tracking left as future work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter motivates the work, states the problem, and enumerates the contributions.

1.1 Context and Motivation

The global health and fitness club market reached a valuation of USD 121 billion in 2024, and analysts project it to grow at a compound annual growth rate of 9.3%, reaching USD 245 billion by 2032 [20]. A key driver of this growth is the adoption of connected fitness technologies, including real-time progress tracking, AI-powered coaching tools, and smart wearable devices. Yet for the vast majority of gym members, the training experience remains one-size-fits-all: the same equipment, the same routines, and no exercise-specific feedback unless the member hires a personal trainer. Personal trainers improve outcomes, since members train more effectively and with fewer injuries, but they are expensive and cannot scale to every user on every visit. A survey of gym members conducted by Khurana et al. [34] corroborates this gap: the leading reasons for gym dropout include lack of knowledge in using equipment and lack of access to a personal trainer, and approximately 60% of respondents expressed reluctance toward wearable-based tracking solutions, suggesting that any monitoring system must integrate into the existing gym

environment rather than require user-worn hardware.

A small number of commercial systems have begun to address this gap. Products such as Tempo [58] and Tonal [59] use dedicated depth cameras and proprietary hardware to track exercises and provide real-time coaching. However, these solutions require specialized equipment that most gyms do not have and cannot afford to retrofit. Meanwhile, nearly every gym already operates a network of cameras for security and safety purposes that staff use only to playback incidents, cameras which capture high-definition footage of the exercise floor.

A key insight motivating this work is that monocular 2D pose estimation, while geometrically ill-posed due to the loss of depth information, can still yield highly discriminative features when paired with a robust temporal modeling technique. As we show in Chapter 5, this combination achieves over 95% classification accuracy across 16 exercise categories, detects 72.8% of exercise sets in continuous gym video (F1 = 83.7%), and counts repetitions within ± 3 of ground truth for 88% of detected sets, all while processing each frame in under 100 ms on a consumer laptop. These results demonstrate that expensive depth sensors or multi-camera rigs are unnecessary for reliable exercise recognition and repetition counting.

The project grew out of the author's membership at Sana Health and Fitness in Bologna, whose owner, Vincenzo Bassi, was already investing in connected gym technology such as 24/7 keycard access and phone-linked treadmills. Bassi envisioned further digitization of the gym experience and initially proposed using his security cameras to identify members entering and exiting the facility. An early project meeting in October 2024, however, revealed a more impactful application of the same infrastructure: recognizing which exercises members perform and counting their repetitions in real time. Together, we defined a two-phase development plan. Phase 1, which constitutes this thesis, focuses on exercise identification and repetition counting to collect data.

Phase 2, left as future work, extends the system with form assessment and posture correction to deliver feedback. Figure 1.1 shows the system in operation at the gym: a security camera captures a member performing lateral raises, the pose estimator overlays body keypoints, the TCN classifier identifies the exercise, and RepNet counts the completed repetitions.

Figure 1.1: The exercise monitoring system running on a live security camera feed at Sana Health and Fitness. The overlaid skeleton shows the body keypoints tracked by the pose estimator, with numerical joint-angle readouts at select joints (Section 2.2.2). The interface displays the classified exercise (lateral raise) and the estimated repetition count. The bounding box defines the crop region used by RepNet for temporal analysis.

1.2 Problem Statement

Given a video stream from a fixed surveillance camera, the system developed in Phase 1 must perform the following tasks:

1. **Exercise detection:** determine whether a person in the frame is performing a gym exercise.

2. **Exercise classification:** identify which exercise the person performs from a set of 16 categories.

3. **Repetition counting:** count the number of repetitions completed within each set.

4. **Result delivery:** log all results to a persistent database and present them to the user through a web interface in real time.

The system must satisfy several constraints. It must operate on the existing security camera infrastructure without requiring additional sensors or dedicated hardware. Processing must be real-time, so that results are available during the exercise rather than after. The classification pipeline must be extensible, allowing new exercise categories to be added by supplying new training videos without redesigning the model architecture. All software dependencies must carry permissive open-source licenses suitable for commercial deployment. Finally, the system must function as a working prototype with a user-facing interface, since a secondary goal of the project is to demonstrate feasibility to potential partners and investors.

In its current form, the system processes one person at a time. This is a deliberate scope constraint for Phase 1; we design the software architecture to support concurrent processing of multiple individuals, contingent on a multi-person pose estimation model and sufficient processing resources.

1.3 Contributions

The main contributions of this work are as follows:

1. **An end-to-end exercise monitoring system** that processes a live camera feed through pose estimation, exercise classification, and repetition counting, evaluated on real gym data collected at Sana Health and Fitness in Bologna with videos recorded by gym staff and members under realistic conditions.

2. **A curated and publicly released exercise dataset** [14] of pose landmark sequences extracted from 330 videos recorded at a real gym, organized into a hierarchical taxonomy of 16 macro-categories and expanded through standard augmentation techniques to improve classifier robustness.

3. **A comparative evaluation of five classification approaches**, namely Decision Trees, Linear Support Vector Classification, k-Nearest Neighbors, Histogram-based Gradient Boosting, and a custom Temporal Convolutional Network, analyzing accuracy, inference speed, and suitability for real-time deployment.

4. **Real-time integration** of three independently developed models (MediaPipe, TCN, RepNet) into a single inference pipeline that operates at interactive frame rates over a WebSocket connection.

5. **A dual-camera viewpoint analysis** demonstrating that camera placement is the primary factor limiting system performance: on the same workout recorded simultaneously by two cameras, detection rates dropped from 70% to 40% when the camera moved from an elevated three-quarter angle to a level frontal position.

6. **A reproducible cloud deployment** using Docker containerization, Terraform infrastructure-as-code, and Google Cloud Platform, demonstrating that the system can be provisioned at a new gym without manual configuration.

A commercial feasibility assessment was also developed in collaboration with a multidisciplinary team, comprising a business plan (Appendix B) with a SWOT analysis, a five-year financial forecast, competitor benchmarking, and a go-to-market roadmap for the boutique fitness segment.

1.4 Thesis Outline

The remainder of this thesis is organized into five chapters and two appendices.

Chapter 2 surveys the technical foundations that the system builds upon: human pose estimation from classical methods through BlazePose, exercise and activity recognition across sensor-based and vision-based paradigms, repetition counting from heuristic thresholds to learned approaches, and temporal models for sequential classification. Chapter 3 describes how these components were assembled into a working product. It covers the hardware infrastructure at the gym, the software architecture of the FastAPI backend and React frontend, the communication protocol, and the Docker and Terraform pipeline used to deploy the system on Google Cloud Platform.

Chapter 4 presents the methodology in detail: the collection and organization of the exercise video dataset, the data processing pipeline that transforms raw video into normalized joint angle features, the five classification approaches evaluated, the integration of RepNet for repetition counting, and the real-time inference pipeline that ties these stages together. Chapter 5 evaluates the system through controlled experiments, comparing classifier performance across all five architectures, analyzing repetition counting accuracy on continuous gym video, quantifying per-stage inference latency, and cataloging failure modes encountered during development and user testing.

Chapter 6 reflects on the contributions, distills the lessons learned over seven months of development, acknowledges the limitations of the current system, and outlines directions for future work. Appendix A collects supplementary technical material, including the complete TCN source code, the evolution of hyperparameter configurations across training rounds, per-class metrics for all classifiers, and a catalog of notable bugs and their resolutions. Appendix B reproduces the business plan developed in collaboration with the gym owner.

Chapter 2

Background and Related Work

This chapter reviews the foundational techniques and prior work that underpin our exercise monitoring system. Section 2.1 surveys exercise and activity recognition approaches across sensor-based, vision-based, and pose-based paradigms, motivating our choice of pose keypoints as the input representation. Section 2.2 covers human pose estimation, tracing the progression from classical methods to modern deep learning architectures and the specific model we adopt. Section 2.3 examines repetition counting methods, from heuristic peak detection to learned approaches. Section 2.4 introduces temporal models for classifying sequential data, with emphasis on the Temporal Convolutional Network architecture central to our pipeline. Section 2.5 positions our work relative to existing commercial and research systems.

2.1 Exercise and Activity Recognition

Human Activity Recognition (HAR) is the task of identifying human actions from sensor observations. The field spans diverse applications, including rehabilitation monitoring, surveillance, and fitness tracking [72, 10]. HAR systems are broadly categorized by their sensing modality: sensor-based systems, where subjects wear inertial or physiological devices, and vision-based systems, where cameras observe the subject from the environment. Our system

adopts a vision-based approach tailored to exercise-specific recognition. Section 2.1.1 contrasts sensor-based and vision-based approaches to motivate our choice of modality, Section 2.1.2 surveys vision-based classification methods, and Section 2.1.3 presents pose keypoints as the intermediate representation we adopt.

2.1.1 Sensor-Based Approaches

Wearable sensors, including inertial measurement units (IMUs), accelerometers, and smartwatches, achieve high classification accuracy for exercise recognition by measuring motion and physiological signals directly on the body [72, 10]. However, these are personal devices; commercial gyms cannot require members to purchase compatible hardware, and providing shared wearables raises both cost and hygiene concerns. Vision-based systems eliminate the per-user hardware requirement entirely. A gym needs only its existing camera infrastructure, and the system scales to every member without additional equipment or compliance burden.

2.1.2 Vision-Based Approaches

End-to-end video classification methods attempt to learn activity representations directly from raw pixel data. Two-stream networks [53] process a spatial stream (a single RGB frame, capturing appearance) and a temporal stream (a stack of optical flow fields, capturing motion) through separate convolutional networks, then fuse their predictions. Three-dimensional CNNs such as C3D [61] and I3D [9] extend this idea by applying 3D convolutions over short video clips to learn spatiotemporal features jointly. While these approaches avoid explicit preprocessing, they require large labeled video datasets and

tend to learn person-specific and environment-specific features (e.g., clothing, background) rather than body structure features. These limitations motivated our adoption of pose-based methods, which reduce the input to a compact skeleton representation that is invariant to appearance and anthropometric differences.

2.1.3 Pose-Based Classification

Modern deep learning achieves its strongest results when models learn features end-to-end from large-scale data. In domains where such datasets exist, for instance ImageNet for object recognition or Kinetics for action recognition, this paradigm excels. Pose-based classification leverages this insight by decoupling the problem: a pose estimator pre-trained on millions of annotated frames extracts body keypoints, and a lightweight downstream classifier operates on those keypoints rather than on raw pixels. The pose estimator absorbs the complexity of appearance variation, while the classifier sees only a compact, person-invariant signal.

This two-stage design offers several practical advantages. The classifier is invariant to lighting conditions, clothing, and anthropometric differences such as height, because these factors are abstracted away by the pose model. Training the classifier is fast, since it processes small feature vectors rather than high-resolution video frames. The stages can also be developed independently: a new pose estimator can replace the current one without retraining the classifier, and vice versa. A classifier trained on the the Muhannad Tuameh Kaggle exercise dataset [62] confirmed that joint angles alone encode sufficient information for reliable exercise classification, validating the choice of this intermediate representation.

A critical challenge for real-world deployment is the open-world recognition problem. Unlike a controlled benchmark, a gym camera observes not only exercises but also people walking, standing, stretching, or talking. A standard

closed-set classifier assigns every input to one of the trained exercise classes, regardless of whether the input actually belongs to any of them. We experimented with confidence thresholding early in development, where predictions below a probability cutoff are rejected as unknown, and found the threshold difficult to tune: setting it too high rejected legitimate exercises performed at unusual angles, while setting it too low admitted false positives from non-exercise motion. We ultimately found a more effective alternative: modeling the non-exercise distribution explicitly by training a dedicated "Other" class on examples of walking, sitting, standing, and similar gym activities that are not exercises. This design decision proved essential for deployment reliability. It converts the open-world problem into a closed-world problem with an explicit rejection category, and the classifier learns the decision boundary from data rather than from a hand-tuned threshold. The "Other" class constitutes the largest category in our dataset (35% of all frames, Table 4.1), reflecting its importance as a robust rejection mechanism. Chapter 4 details how we constructed this class and the effect it had on deployment reliability.

Having established that pose-based features offer the strongest trade-off for our setting, the next section reviews pose estimation methods in detail, from classical approaches to the specific model we adopt.

2.2 Human Pose Estimation

The human pose estimation problem determines the spatial location of K anatomical keypoints of the human body from an image of size $W \times H \times C$. The problem gained commercial visibility in 2010 when Microsoft released the Kinect, which paired a structured infrared light projector with an infrared camera to estimate 3D body poses in real time using randomized decision forests [52]. Since the Kinect, the computer vision community has invested heavily in deep learning approaches that deliver higher precision, denser joint sets, and greater robustness to occlusion and viewpoint variation.

Three-dimensional pose estimation from a single monocular camera remains an ill-posed problem, since the projection from 3D to 2D discards depth information. Section 2.2.2 discusses how we address this constraint by operating entirely in the 2D projected domain.

2.2.1 Deep Learning Approaches

Before deep learning, pose estimation relied on hand-crafted models such as pictorial structures [18] and deformable parts models [71], which required explicit feature engineering and struggled with occlusion and viewpoint variation. DeepPose [60] marked the transition to learned representations by applying cascading convolutional neural networks that captured spatial context across the entire image, enabling the network to reason about occluded joints without hand-designed spatial priors.

In 2017, OpenPose [8] introduced multi-person pose estimation through a bottom-up architecture. Rather than first detecting each person and then localizing their joints (the top-down paradigm), OpenPose detects all keypoints in the image simultaneously and assembles them into individual skeletons using Part Affinity Fields. This bottom-up strategy maintains constant computational cost regardless of the number of people in the frame, whereas top-down approaches scale linearly with each detected person. HRNet [56] advanced the top-down paradigm by maintaining high-resolution feature maps throughout the network, rather than recovering spatial detail from a low-resolution bottleneck. Table 2.1 summarizes the key characteristics of these architectures.

While HRNet achieves the highest accuracy on the COCO benchmark, its top-down design requires a separate person detector, adding pipeline complexity. For our system, we require a pose estimator that runs at interactive frame rates on consumer hardware, carries a permissive open-source license suitable for commercial deployment, and provides a dense set of keypoints. These requirements led us to adopt BlazePose, described in the following subsection.

Table 2.1: Comparison of deep learning pose estimation architectures. GFLOPs depend on input resolution; values use each paper's default configuration. COCO AP and PDJ are different metrics on different benchmarks and are not directly comparable.

Model	Year	Approach	Keypoints	GFLOPs	COCO AP	PDJ (%)
OpenPose	2017	Bottom-up	18	136.0	61.8[a]	
HRNet-W32	2019	Top-down	17	7.1	74.4[b]	
HRNet-W48	2019	Top-down	17	14.6	75.1[b]	
BlazePose	2020	Top-down	33	≈0.007[c]		94.2[d]

We omit DeepPose [60] (2014, 14 keypoints, top-down): it predates the COCO benchmark, did not report GFLOPs, and its PDJ was evaluated on FLIC/LSP with 10 upper-body joints, making its numbers incomparable to later work. [a]COCO test-dev, 368×368 input [8]. [b]COCO val2017, 256×192 input [56]. [c]Reported for the Full variant (6.9 MFLOPs); exact GFLOPs for the Heavy variant are not published. See Table 2.2. [d]Averaged across 14 geographic subregions [5].

2.2.2 MediaPipe Pose

We selected the BlazePose architecture within the MediaPipe framework for four reasons: its Apache 2.0 license (compatible with commercial deployment, unlike the AGPL-3.0 license of YOLO-based alternatives), compatibility with the Kaggle exercise dataset used in our early experiments, its 33-keypoint topology (called *landmarks* in the MediaPipe API) that provides a denser skeleton than the 17 keypoints in the COCO person instance dataset, and its lightweight architecture designed for on-device inference. The network runs at over 30 frames per second on a Pixel 2 smartphone [5] using a dual heat map and regression localization strategy. MediaPipe, developed by Google, provides deployment libraries across multiple platforms and programming languages [24]. For real-time applications, the architecture uses the inverted residual with linear bottleneck modules from MobileNetV2, which operate on low-dimensional compressed representations [49].

BlazePose ships in three variants that trade model size and accuracy for inference speed. Table 2.2 summarizes the key differences. The heavy model achieves 94.2% average PDJ (Percentage of Detected Joints, equivalent to PCK@0.2) across 14 geographic subregions, while the lite model reaches 87.0% at over ten times the frame rate. We use the heavy model because

our application targets 10 frames per second and runs on an M1 MacBook, which is considerably more powerful than the Pixel 3 benchmark device. At this frame rate, the heavy model's inference time is well within budget, and its more precise keypoints reduce noise that would otherwise propagate into classification error.

Table 2.2: BlazePose model variants. FPS measured on a Pixel 3 smartphone. PDJ (Percentage of Detected Joints) averaged across 14 geographic subregions [5].

Variant	**Size (MB)**	**FPS (CPU)**	**FPS (GPU)**	**Avg. PDJ (%)**
Lite	3	44	49	87.0
Full	6	18	40	91.8
Heavy	26	4	19	94.2

Each variant takes a $256 \times 256 \times 3$ RGB crop centered on the detected person. The output is a 33×5 array: each keypoint carries x, y, and z coordinates alongside visibility and presence scores. The x and y coordinates are local to the crop and range from 0 to 255. The z coordinate encodes depth relative to the hip plane, derived from synthetic data via the GHUM articulated body model rather than from human annotation; it is therefore up to scale and not metrically accurate. Visibility indicates, after a sigmoid activation, the probability that the keypoint is both in-frame and unoccluded, while presence indicates only that the keypoint is in-frame regardless of occlusion.

The model card [25] documents several limitations relevant to our deployment. BlazePose tracks only one person per frame; when multiple people appear, the model locks onto one and ignores the rest. The model also requires the subject's head to be visible for reliable tracking, due to their use of a face detector as a fast proxy for a person detector, and performance degrades when the subject is farther than approximately four meters from the camera. Degraded lighting, motion blur, and body-part occlusion increase inter-frame "jittering" in keypoint predictions. These limitations directly influenced our

system design choices, discussed in Chapter 3. Figure 2.1 shows the 33 keypoint locations that the model tracks.

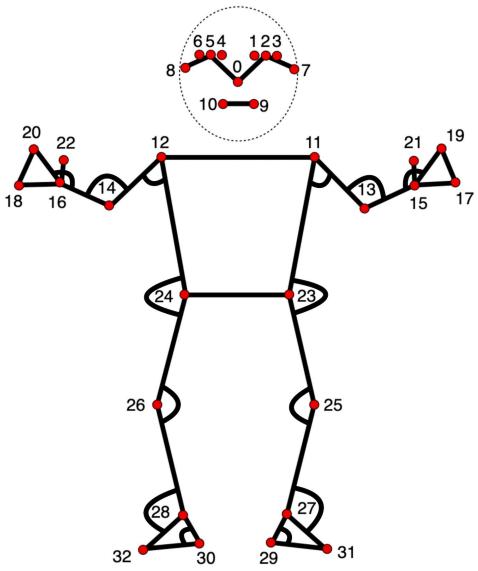


Figure 2.1: The 33 body keypoints tracked by BlazePose. Each keypoint carries x, y, z coordinates alongside visibility and presence scores [24]. In addition, we annotate in arcs the 14 angles used downstream, computed from the BlazePose keypoints.

2.2.3 Keypoint Topologies

A pose estimator's *topology* defines which anatomical points it tracks and how those points connect to form a skeletal representation. Different topologies reflect different design priorities: benchmarking convenience, application-specific detail, or computational budget. Table 2.3 summarizes the major topologies in the literature.

The COCO topology is the de facto standard for benchmarking, defining 17 keypoints that cover the nose, eyes, ears, shoulders, elbows, wrists, hips, knees, and ankles. However, COCO stops at the wrist and ankle, providing no hand or foot detail. BlazePose extends the COCO set to 33 keypoints by incorporating keypoints from two related Google models: BlazeFace contributes facial keypoints (mouth corners), and BlazePalm contributes hand keypoints

Table 2.3: Common keypoint topologies in human pose estimation. Keypoint counts refer to per-person detections.

Topology	Keypoints	Coverage
LSP [32]	14	Major joints, neck, head top
MPII [2]	16	Adds pelvis and thorax
COCO [38]	17	Joints + nose, eyes, ears
Kinect v2 [52]	25	Full body including hands and spine
OpenPose BODY_25 [8]	25	COCO core + 6 foot keypoints
BlazePose [5]	33	Superset of COCO, BlazeFace, and BlazePalm

(index finger knuckles for wrist orientation) [5]. BlazePose further adds six foot keypoints (heels, foot indices, and big toes) that no COCO-based model tracks.

We inherit the BlazePose topology for the following reasons:

- **Foot keypoints enable lower-body exercise discrimination.** Exercises such as squats, lunges, deadlifts, and step-ups depend on foot placement and heel contact. The heel and foot index keypoints allow us to compute ankle and heel angles (Table 2.4) that would not be possible without ad-hoc approximation under COCO's ankle-only tracking.

- **Denser keypoints yield more joint angles.** The 33-point topology lets us define 14 meaningful joint angles across seven bilateral pairs, compared to at most 8 angles derivable from COCO's 17 keypoints, which lack the hand and foot keypoints needed for wrist, ankle, and heel angles.

- **Kaggle dataset compatibility.** The public exercise dataset [62] used in our early experiments was extracted with BlazePose, so adopting the same topology avoids a remapping step.

- **Superset compatibility.** BlazePose is a strict superset of COCO, so any downstream model trained on BlazePose features can be adapted to a COCO-only estimator by dropping the extra keypoints. This preserves forward compatibility if the pose model is replaced in Phase 2.

2.3 Repetition Counting

Repetition counting estimates the number of cycles a periodic action completes within a video or signal segment. In a gym setting, the repetition count is the primary metric that users expect alongside exercise classification. Approaches fall into two categories: class-specific methods, which hard-code motion patterns for each exercise, and class-agnostic methods, which detect periodicity in the signal regardless of the action being performed. Class-specific detectors are brittle because they must be redesigned for every new exercise; class-agnostic methods generalize by operating on the structure of the signal itself.

2.3.1 Heuristic Methods

There are several naive repetition counting methods based on the following heuristics:

1. **Peak detection** increments a counter on one or more joint angle signals.
2. **PCA projection** reduces multi-dimensional joint angle time series to a single quasi-periodic signal via the first principal component, then counts peaks or zero-crossings on the resulting waveform.
3. **Binary threshold crossing** defines an UP and a DOWN state, then tracks the number of crossings completed.

All three approaches reduce a multi-dimensional exercise signal to a scalar time series and apply a deterministic counting rule. This simplicity comes at a cost. Each method requires per-exercise tuning, is sensitive to noise in the pose estimates, and must be redesigned whenever a new exercise category is added. Moreover, deterministic thresholds are, in principle, susceptible to adversarial manipulation once users identify the counting criterion. These limitations motivated our adoption of a learned, class-agnostic approach.

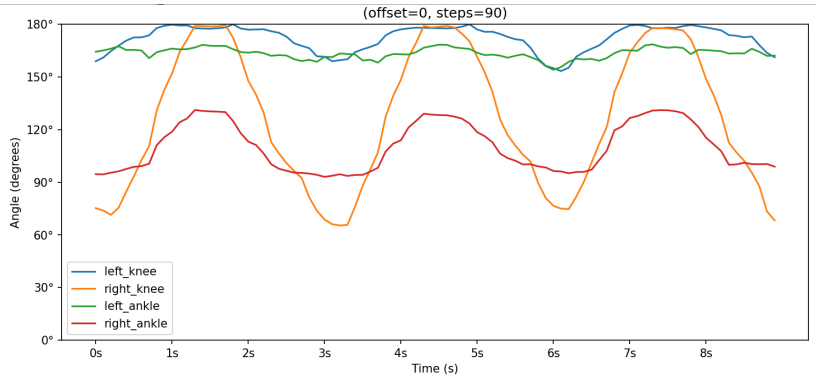


Figure 2.2: Knee angle over time during a squat exercise. By reducing the pose signal to a single dimension, threshold crossings on the periodic waveform can be counted to estimate the number of repetitions completed.

2.3.2 Learned Methods

RepNet [16] introduced a class-agnostic approach to repetition counting based on temporal self-similarity. The model first encodes each frame of an input video into a learned embedding, then computes a pairwise similarity matrix across all frames: if the action is periodic, this matrix exhibits a regular pattern of off-diagonal stripes corresponding to repeated cycles (Figure 2.3). A transformer processes the similarity matrix to predict a per-frame periodicity length, from which the total count is derived. Because RepNet operates on visual similarity rather than exercise-specific rules, it generalizes to unseen action classes without retraining.

An alternative line of work, exemplified by ESCounts [54], matches a user-provided exemplar of one repetition against the video to count occurrences, enabling few-shot counting without class-specific training. We selected RepNet because it requires no exemplar input and a public implementation was available at the time of development. In our pipeline, RepNet receives a cropped video of the user containing only the frames classified as exercise, and its predictions are typically accurate to within ± 1 repetition. RepNet's known failure modes include variable-speed repetitions (where the period changes mid-set), very slow exercises whose period exceeds the model's

temporal window, and compound movements such as a clean-and-press where sub-phases create ambiguous periodicity.

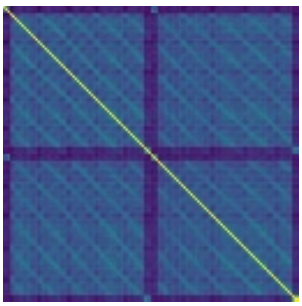


Figure 2.3: Temporal self-similarity matrix for a 24-second video of a person performing squats. The bright main diagonal represents each frame compared with itself. The off-diagonal stripes indicate frames with similar poses, recurring at the period of the exercise. Ten stripes are visible along each row, corresponding to the ten repetitions performed. RepNet processes this matrix with a transformer to predict the repetition count.

2.4 Temporal Models for Sequential Data

The pose estimator produces 33 keypoints per frame, but raw keypoint coordinates are sensitive to the subject's position within the frame, their distance from the camera, and their body proportions. We therefore convert the keypoints into 14 joint angles, each defined by a triplet of keypoints and measured at the center joint. Table 2.4 lists the 14 angles, organized as seven bilateral pairs (left and right shoulder, elbow, wrist, hip, knee, ankle, and heel) chosen with input from a fitness trainer. Each angle is computed via `arctan2` and normalized to $[0, 1]$ by dividing by $180°$, yielding a compact, person-invariant feature vector. Alongside each angle, we retain the corresponding keypoint visibility score from BlazePose, giving a 28-dimensional input vector per frame (14 angles + 14 visibility scores). The visibility scores allow the classifier to discount angles computed from occluded or uncertain keypoints.

The classification task becomes a sequence-to-label problem: given a sliding window of T frames, each described by 28 features, assign an exercise

class. This formulation requires a model capable of capturing temporal dependencies across the window. Our initial classifiers (Decision Tree, Support Vector Classifier, k-Nearest Neighbors) treated each window as a flat feature vector, discarding temporal structure. We swept the window size from 1 to 100 frames (Figure 4.3) and found that 30 frames offered the best balance between classification accuracy and perceptive lag: shorter windows lack sufficient context to distinguish similar exercises, while longer windows delay predictions and discard short videos from the training set.

Even with a tuned window, the k-NN classifier that achieved the highest offline accuracy failed to generalize to live camera feeds, overfitting to training-set viewpoints and body proportions. This generalization gap motivated us to adopt a model that could learn temporal features rather than memorize flattened vectors. Section 2.4.1 briefly reviews recurrent neural networks and explains why we did not pursue them, and Section 2.4.2 introduces the Temporal Convolutional Network architecture we adopted.

Table 2.4: The 14 joint angles used as input features. Each angle is defined by three BlazePose keypoints; the angle is measured at the **vertex** (center) keypoint.

#	Joint	Endpoint	Vertex	Endpoint
1	Left shoulder	left hip	left shoulder	left elbow
2	Right shoulder	right hip	right shoulder	right elbow
3	Left elbow	left shoulder	left elbow	left wrist
4	Right elbow	right shoulder	right elbow	right wrist
5	Left wrist	left elbow	left wrist	left index
6	Right wrist	right elbow	right wrist	right index
7	Left hip	left shoulder	left hip	left knee
8	Right hip	right shoulder	right hip	right knee
9	Left knee	left ankle	left knee	left hip
10	Right knee	right ankle	right knee	right hip
11	Left ankle	left knee	left ankle	left foot index
12	Right ankle	right knee	right ankle	right foot index
13	Left heel	left ankle	left heel	left foot index
14	Right heel	right ankle	right heel	right foot index

2.4.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are the classical approach to sequence modeling, processing inputs one time step at a time through a hidden state. Vanilla RNNs struggle with long-range dependencies due to vanishing or exploding gradients [6]; gated variants such as LSTM [28] and GRU [11] mitigate this through learned gates that control information flow. Despite their effectiveness, RNNs have two drawbacks for our application: their sequential processing prevents parallelization across time steps, making training slower than feed-forward alternatives, and their hidden state size and training dynamics require careful tuning. Drawing on prior experience with temporal convolutional architectures in a University of Bologna Formula Student perception pipeline, we moved directly from scikit-learn baselines to a Temporal Convolutional Network without experimenting with RNNs. We acknowledge that including an LSTM or GRU baseline would have strengthened the comparison by isolating the contribution of temporal convolutions over recurrent processing. This remains a limitation of the experimental scope.

2.4.2 Temporal Convolutional Networks

Temporal Convolutional Networks (TCNs) apply one-dimensional convolutions along the time axis to extract features from sequential data. The architecture originates from WaveNet [63], which introduced dilated causal convolutions for audio generation; Lea et al. [37] adapted the idea for action segmentation in video, and Bai et al. [3] formalized the generic TCN framework and showed that it matches or exceeds LSTM performance across a range of sequence benchmarks.

The key mechanism is *dilated convolution*: by spacing the kernel elements at exponentially increasing intervals (dilation factors 1, 2, 4, …), each successive layer doubles its receptive field without increasing the number of parameters. A network with L layers and dilation factor 2^l at layer l achieves

a receptive field of $O(2^L)$ time steps, meaning that a modest stack of layers can cover the entire input window. Residual connections [27] between layers stabilize gradient flow and allow deeper networks to train reliably.

TCNs offered three advantages over RNNs for our setting:

- **Parallelism.** Convolutions over the time axis are fully parallelizable, exploiting GPU hardware far more efficiently than sequential recurrence.
- **Interpretable receptive field.** The fixed receptive field is interpretable: we can calculate exactly how many frames the network considers and verify that it spans at least one full exercise repetition.
- **Stable gradients.** TCNs produce stable gradients by construction, since the computational graph has bounded depth regardless of sequence length, eliminating the vanishing gradient problem that plagues vanilla RNNs.

Our implementation follows the generic TCN framework of Bai et al. [3] with three practical modifications. First, we use batch normalization [30] rather than the weight normalization prescribed in the original paper. Weight normalization is generally preferred for sequence tasks because it is independent of batch size and unaffected by zero-padded variable-length inputs. However, our setting avoids both pitfalls: all input sequences are fixed-length (no padding required) and we train with a batch size of 32, which is large enough for stable batch statistics. Batch normalization smooths the optimization landscape and provides an implicit regularization effect that complements the explicit dropout in each block. Second, we use non-causal (symmetric) padding instead of causal left-padding, since the entire sliding window is available at classification time and there is no need to prevent the model from attending to future frames. Third, we append a global average pooling layer followed by a fully connected layer to produce a single class prediction per window, converting the architecture from sequence-to-sequence to sequence-to-label.

We also considered transformer-based architectures [64], which have become dominant in natural language processing and are increasingly applied to time series. However, at the time of development we found no lightweight transformer tailored to small input tensors (14 channels $\times$ 30 frames), and the self-attention mechanism's quadratic cost offered no benefit over convolutions at this scale. The TCN's simplicity and the author's prior familiarity with the architecture made it the pragmatic choice. Chapter 4 details the specific architecture, hyperparameters, and training procedure.

A notable omission from our experimental comparison is skeleton-based action recognition using Graph Convolutional Networks (GCNs). ST-GCN [70] models the skeleton as a spatial graph where joints are nodes and bones are edges, applying graph convolutions that exploit the body's structural topology. ST-GCN and its successors (2s-AGCN, MS-G3D) have become the dominant paradigm for skeleton-based action recognition on benchmarks such as NTU-RGB+D. In retrospect, GCNs represent the most prominent alternative to our TCN approach for pose-based classification. We did not consider them during development because our feature engineering pipeline reduces the skeleton to 14 joint angles rather than preserving the full graph structure of 33 keypoints, and because our literature review at the time focused on temporal sequence models rather than graph-based architectures. Adapting ST-GCN to our angular feature representation would require either preserving the raw keypoint coordinates (forgoing the position-invariance of joint angles) or defining a meaningful adjacency matrix over angle features, neither of which was straightforward given the project timeline. Evaluating whether GCN-based approaches would outperform our TCN on the Sana dataset remains an open question for future work.

2.5 Related Systems and Commercial Solutions

Existing solutions for technology-assisted exercise monitoring fall into three categories: hardware-integrated systems that bundle proprietary equipment with AI coaching, software-only platforms that run on personal devices, and open-source tools for manual video analysis. Tables 2.5 and 2.6 summarize the landscape.

2.5.1 Hardware-Integrated Systems

Several companies sell dedicated fitness hardware with built-in exercise tracking. Tempo [58] pairs a screen with 3D time-of-flight sensors to track body position during strength exercises. Tonal [59] is a wall-mounted electromagnetic resistance machine with 17 motion sensors and a camera for form assessment. Peloton [45] equips its higher-end bikes, treadmills, and rowers with computer vision through its "Peloton IQ" system. Technogym [57] produces commercial-grade cardio and strength machines with integrated motion tracking, designed for gym-wide deployment.

Table 2.5: Hardware-integrated fitness systems. All require purchasing proprietary equipment. "CV" indicates whether the product uses computer vision for exercise tracking.

Product	Price (USD)	Subscription	CV	Retrofit	Equipment
Tempo	250–4,000	$39/mo	Yes	Partial	Screen + 3D sensors
Tonal	4,000	$60/mo	Yes	No	Wall-mounted machine
Peloton	1,700–6,700	varies	Yes	No	Bike, tread, rower
Technogym	2,000–10,000	$10/mo[a]	Yes	Yes	Commercial machines

[a]Technogym lists £9/mo; converted at approximate exchange rate.

The common limitation of hardware-integrated systems is their cost and deployment model. Each product requires purchasing proprietary equipment that cannot be retrofitted to a gym's existing infrastructure (with the partial exception of Technogym, which sells modular commercial machines). A gym with 50 Tonal stations would spend approximately $200,000 on hardware

alone, plus recurring subscription fees. Our system requires zero additional cameras or sensors: it repurposes the security cameras that most commercial gyms already operate, requiring only a processing machine to run the inference pipeline.

2.5.2 Software-Only Platforms

Software-only competitors use computer vision through personal devices rather than dedicated hardware. Table 2.6 compares the major platforms. XTRA Vision AI [68] provided a 3D motion tracking API that transformed any camera into a smart tracker for musculoskeletal health screening, targeting healthcare and digital wellness platforms; the company ceased operations in 2025, and its website is no longer available. VAY Sports [65] is a mobile app that uses the smartphone's front-facing camera to provide real-time audio feedback during workouts, comparing user movements against reference repetitions designed with personal trainers. Viso.ai [66] offers a no-code computer vision development platform that can be configured for fitness applications among other domains, rather than a fitness-specific product. Kemtai [33] provides a motion tracking platform for physiotherapy and fitness with a library of over 2,000 exercises, FDA-listed for clinical use.

Table 2.6: Software-only fitness platforms. All use computer vision through personal devices or APIs. "Recog." = exercise recognition, "Reps" = repetition counting, "Form" = form feedback.

Platform	Camera	Recog.	Reps	Form	Market
XTRA Vision AI[†]	Any (API)	Yes	–	Yes	B2B healthcare
VAY Sports	Smartphone	Yes	Yes	Yes	Consumer
Viso.ai	Any	(general-purpose CV platform)			B2B enterprise
Kemtai	Any (API/app)	Yes	Yes	Yes	B2B healthcare
Ours	**Security cam**	**Yes**	**Yes**	**No**	**B2B gyms**

[†]Defunct as of 2025.

Software-only platforms address the cost problem but introduce a different constraint: they require each user to point a personal device at themselves

during their workout. This limits the field of view to a single person, requires user compliance (they must remember to set up the device), and cannot scale to gym-wide coverage. Our system avoids this limitation by using ceiling-mounted or wall-mounted security cameras that observe the gym floor continuously, without any action required from the user.

2.5.3 Open-Source Tools

Kinovea [36] is a notable open-source video analysis tool used in sport science, physiotherapy, and coaching. It supports manual and semi-automated tracking annotation of video recordings, including angle measurement, trajectory tracking, and side-by-side comparison. However, Kinovea is fundamentally an offline tool: an analyst must mark keypoints and draw measurements frame by frame. It provides no automated exercise recognition, no repetition counting, and no real-time feedback. Its value lies in detailed biomechanical analysis of individual recordings, a use case complementary to, rather than competitive with, automated systems like ours.

2.5.4 Positioning

Our system occupies a different point in this landscape. Unlike hardware-integrated solutions, it requires no proprietary equipment and minimal capital expenditure beyond the cameras a gym typically already owns. Unlike software-only platforms, it does not depend on user compliance or personal devices. Unlike Kinovea, it operates in real time with fully automated recognition and counting. Tables 2.5 and 2.6 include our system in the final row for direct comparison.

These advantages come with significant trade-offs. The system currently tracks only one person per camera, a limitation inherited from BlazePose's single-person architecture (Section 2.2.2). It does not provide form feedback, does not identify individual users, and its 16-category exercise vocabulary is

narrow compared to commercial platforms that support hundreds of movements. Section 6.3 provides a full accounting of these constraints.

The closure of XTRA Vision AI in 2025 illustrates the fragility of standalone fitness CV platforms that lack a physical integration point: without embedding into existing gym infrastructure, adoption depends entirely on convincing operators to change their workflow. Our system mitigates this risk by attaching to cameras the gym already owns and operates for security purposes, requiring no behavioral change from staff or members.

Having surveyed the technical foundations and the competitive landscape, we now turn to the architecture of the system we built. Chapter 3 describes how we assembled these components into a working product, from the hardware infrastructure at the gym through the cloud deployment that made the system accessible from any browser.

Chapter 3

System Design and Architecture

The system architecture emerged through iterative design, stakeholder feedback, and practical constraints imposed by the gym environment. A key requirement was browser-based access from any device, enabling a live product demonstration without requiring users to install software. This motivated a cloud-deployed microservice architecture in which each component runs as an independent, containerized service that can be developed, scaled, and updated in isolation. The system is packaged into Docker [15] containers, provisioned with Terraform [26] infrastructure-as-code, and deployed on Google Cloud Platform [23] using Cloud Run. A comprehensive business plan [4] (reproduced in Appendix B) with market analysis (pp. 7–9), competitor benchmarking (p. 10), a SWOT assessment (p. 12), and a five-year financial forecast (pp. 16–17) informed the success criteria in Section 3.1. Section 3.2 presents the pipeline at a high level, and Section 3.4 details the three software modules. Section 3.3 describes the hardware infrastructure, Section 3.5 documents the REST and WebSocket communication protocol, and Section 3.6 covers containerization and cloud deployment. Figure 3.1 illustrates the end-to-end pipeline.

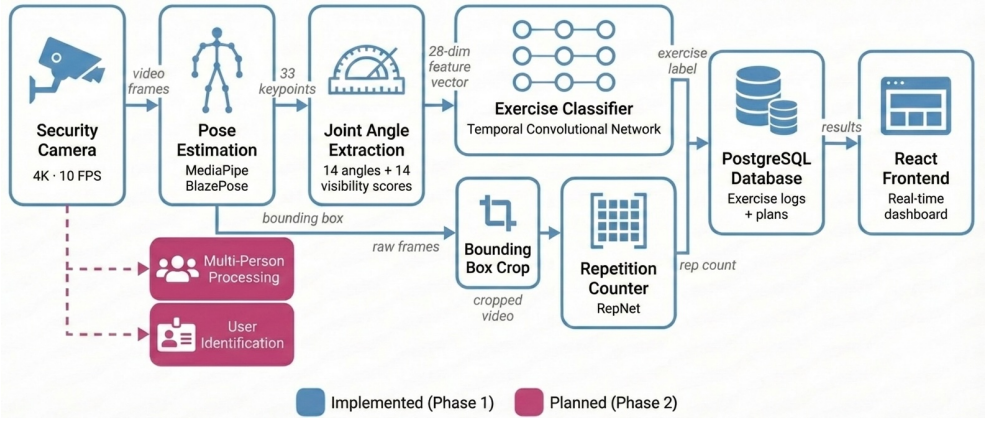


Figure 3.1: End-to-end system architecture. A security camera captures video at 4K resolution. The backend extracts 2D pose keypoints via MediaPipe, computes joint angles, and feeds them to the exercise classifier TCN. In parallel, RepNet analyzes cropped frames to estimate repetition counts. Results flow to the database and the web frontend. Pink elements indicate planned Phase 2 extensions.

3.1 Requirements Analysis

Functional and non-functional requirements were derived from the October 2024 kickoff meeting with Bassi and refined iteratively through demonstrations of the system to clients, gym trainers, and the gym owner.

Functional Requirements

- Identify which exercise a person is performing from a camera feed.
- Count repetitions in real time.
- Display results to the user via a web interface.
- Log exercises to a database for workout history.
- Allow a trainer to define exercise plans (sets, reps).
- Support multiple camera sources (webcam, RTSP IP camera, video file).

Non-functional Requirements

- Real-time: the gym's NVR records at a native frame rate of 10 FPS, so the pipeline must process each frame within 100 ms to keep pace with the incoming feed without accumulating latency.
- Browser-based: accessible from any device without installing software.
- Use existing gym security cameras (no new cameras or sensors; a processing machine is required).
- Cloud-deployable: demo-ready on mobile devices.
- Scalable when idle (cost control).

Chapter 5 evaluates the system against each of these requirements.

3.2 System Overview

As illustrated in Figure 3.1, the security camera records continuously for safety. The backend captures frames from this feed and passes each one to BlazePose, which estimates 33 body keypoints. From these keypoints, the system computes joint angles and visibility scores that serve as input to the exercise classifier. BlazePose also produces a bounding box around the detected person. When the classifier labels a consecutive sequence of frames as a particular exercise, the corresponding frames are cropped to the maximal bounding box and forwarded to RepNet, the repetition counter, which processes them asynchronously. The resulting exercise label and repetition count are then written to the database and broadcast to the frontend. Phase 2 will extend this pipeline with a multi-person pose estimator and user identification.

The pipeline is decomposed into three independent models rather than a single end-to-end network for two reasons. First, we lacked the large-scale annotated video dataset and the compute budget that an end-to-end model

would require. Second, a modular design allows each stage to be upgraded independently as better models become available, without retraining the entire pipeline. Repetition counting runs asynchronously because RepNet processes the full accumulated video sequence at once, requiring substantially more than the 100 ms per-frame budget (Section 5.3); running it synchronously would stall the classification loop. The 10 FPS processing rate matches the native frame rate of the gym's NVR security camera system, which recorded at 10 FPS when the project began.

3.3 Hardware Setup

A key non-functional requirement was to use the gym's existing security cameras rather than install new hardware. This section describes the camera infrastructure we leveraged and the processing machines used for development, training, and deployment.

3.3.1 Camera Infrastructure

Sana Health and Fitness operates a network video recorder (NVR) system with multiple cameras connected via Ethernet to a private VLAN. The cameras record continuously at up to 3840×2160 pixels (4K UHD) and are monitored through VMS-Pro 6.0 management software. For this project, we utilized only the CH2 and CH3 cameras, since these monitor the free weights fitness room where exercises are performed. Other channels were considered but rejected because they lacked unobstructed views of the exercise area.

The cameras stream video using RTSP [50], delivering H.264 or H.265 video over the local network. The stream URL accepts several parameters that we tuned for our use case:

- `channel`: selects the physical camera. We used channels 02 and 03 for the two functional training room angles.

- `subtype`: selects the stream quality. Setting `subtype=1` requests the sub-stream at 1280×720 (HD) rather than the full 4K main stream (`subtype=0`). Since BlazePose internally resizes its input to 256×256 pixels [5], streaming at 4K would waste bandwidth on resolution that the model discards. We used the HD sub-stream for real-time inference and reserved the 4K main stream for dataset collection, where higher resolution preserves detail during manual trimming and augmentation.

- `no-audio`: disables the audio track, eliminating decode overhead for data we do not use.

On the application side, OpenCV's `VideoCapture` handles RTSP connection, authentication, and H.264 decoding. Each `capture.read()` call returns an uncompressed NumPy array ready for inference, while the network transfer itself remains compressed. A buffer size of two frames minimizes latency while preventing frame accumulation during processing spikes.

Cropping for Keypoint Precision. BlazePose downscales its input to 256×256 pixels regardless of the original frame resolution. When the full camera view is passed directly, a person standing several meters from the camera occupies only a small fraction of the model's effective input, degrading keypoint precision. We observed empirically that cropping the frame to the region around the subject before passing it to the pose estimator substantially improved detection reliability: the person then fills a larger portion of the 256×256 input, giving the network more pixels to localize each joint. This insight also explains why the HD sub-stream (1280×720) performs comparably to the 4K main stream for real-time inference, since both are downscaled to the same 256×256 resolution by the model. The administrator can define the crop region interactively from the frontend (Section 3.4.4), and the dual-camera experiment in Section 5.3.3 further demonstrates how camera angle and distance affect keypoint quality.

3.3.2 Camera Placement and Data Collection

Camera placement also influenced data quality. We observed that off-axis camera angles produced more reliable pose estimates than head-on views, consistent with the well-known viewpoint sensitivity of 2D pose estimators: frontal views cause left-right limb ambiguity because overlapping joints are projected onto the same image coordinates [5]. Section 5.3.3 quantifies this effect through a controlled dual-camera experiment. The CH2 camera provides a roughly head-on view of the exercise area, approximately 30 degrees off axis. The CH3 camera captures a complementary angle from behind and to the left. During data collection, recordings from both cameras were trimmed to isolate individual exercises and organized by exercise category. Care was required to ensure the 4K main stream was uploaded rather than the HD sub-stream; on more than one occasion, gym staff accidentally uploaded the sub-stream, which degraded the quality of the resulting training data.

The NVR retains recordings for approximately four days before overwriting. This imposed a strict workflow: new exercise recordings had to be downloaded via VMS-Pro from the gym's server within that window, or the data was lost. On one occasion, a miscommunication with gym staff resulted in recordings being overwritten before they could be retrieved. The data collection workflow, including how recordings were organized and stored, is described in Section 4.1.

3.3.3 Processing Hardware

Development and deployment spanned three computing environments, each serving a distinct role in the pipeline. Table 3.1 summarizes their specifications.

MacBook Air. The primary development machine ran macOS and hosted the FastAPI backend, React frontend, and local PostgreSQL database during development. For inference, PyTorch's Metal Performance Shaders (MPS)

Table 3.1: Processing hardware used for development, training, and deployment.

Machine	CPU	GPU	RAM
MacBook Air	Apple M1 (8-core)	M1 integrated (7-core)	16 GB
Ubuntu Workstation	AMD Ryzen 7 2700 (16 threads)	NVIDIA RTX 4080 SUPER	32 GB DDR4
GCP Cloud Run Gen2	4 vCPU (amd64)	NVIDIA L4	16 GB

backend provided GPU acceleration on the M1's integrated GPU. All dataset trimming, video augmentation via FFmpeg, and data organization were performed on this machine.

Ubuntu Workstation. A dedicated Linux workstation equipped with an NVIDIA RTX 4080 SUPER and CUDA 12.8 served as the training machine. All model training, including the Decision Tree baselines, the TCN classifier, and the LinearSVC, KNN, and HistGradientBoosting experiments, ran on this machine. Datasets were transferred from the MacBook Air via SMB over VPN. The MediaPipe pose extraction pipeline for producing CSV datasets from raw videos also ran on this workstation to leverage GPU acceleration.

Google Cloud Platform. The production deployment target, described in detail in Section 3.6, ran on GCP Cloud Run Gen2 in the `europe-west1` region. The backend container was provisioned with 4 vCPUs, 16 GB of RAM, and one NVIDIA L4 GPU (24 GB VRAM), matching the resource limits defined in the Terraform configuration. The frontend container required no GPU and ran with default Cloud Run resources.

3.4 Software Architecture

The software architecture consists of three modules. The backend is built on FastAPI [47] and hosts all neural network inference. The frontend, built with React, provides the user-facing web interface. The database stores a persistent log of recognized exercises and the trainer-defined workout plan. The following subsections describe each module.

3.4.1 Backend: FastAPI

The need for interactive controls, such as confidence thresholds and visualization toggles, initially pointed toward desktop GUI frameworks like Tkinter or PyQt5. However, desktop applications could not be deployed to the cloud or accessed from a mobile device, both of which were requirements for the stakeholder demonstration and eventual production use. We therefore pivoted to a web framework. The backend needed to stream video from IP cameras and webcams, serve processed results in real time, interact with a database, accept user configuration changes without restarting, and run inside a Docker container on GCP Cloud Run. Among the candidates, Flask lacks native WebSocket and async support, requiring extensions such as Flask-SocketIO and an external event loop. Django provides these through Django Channels, but introduces a heavier ORM and application structure that exceeded our needs. We also briefly tried Streamlit, but abandoned it the same day because it could not stream real-time video. FastAPI [47] satisfied every requirement out of the box: first-class WebSocket endpoints via Starlette, native `async/await` for concurrent frame processing, Pydantic models for automatic request validation, a dependency injection system for sharing stateful objects such as the `FrameProcessor` across endpoints, and a lifespan context manager for background camera capture tasks. Prior experience with FastAPI and its thorough documentation further shortened the learning curve.

3.4.2 Real-Time Architecture

The real-time constraint shaped the backend architecture. An initial pull-based design polled for new frames at 10 Hz. This suffered from temporal drift: when the incoming video feed was inconsistent, the polling instant would not coincide with the frame arrival, causing the inference pipeline to lag progressively behind the live feed. Even with a frame buffer, the buffer would inevitably fill during processing spikes and force frame drops. To resolve this,

we refactored to a push architecture, illustrated in Figure 3.2. FastAPI pushes each incoming frame onto a shared queue, and whenever the queue is non-empty, the inference module `livestreaming.py` consumes and processes it.

Bounded Buffers. The push architecture still requires a strategy for when the inference pipeline is slower than the camera. A naive unbounded queue would accumulate frames and introduce ever-growing latency. Instead, the `FrameStreamer` class uses an `asyncio.Queue` with `maxsize=1` for both its input and output stages. When a new frame arrives and the input queue is already full, the camera source replaces the stale frame using a `get_nowait()` and `put_nowait()` pair, bounding worst-case latency to a single frame interval. The same strategy applies on the output side: if the MJPEG stream consumer has not yet fetched the previous annotated frame, the new one silently replaces it. The overall design philosophy is to sacrifice completeness for freshness [22]: it is acceptable to skip frames, but unacceptable for the displayed results to lag behind the live feed.

Downstream of the `FrameStreamer`, the inference pipeline uses several additional bounded buffers. As shown in Figure 3.2, joint angle vectors flow from the main thread into a `queue.Queue` with a capacity of 60 entries, decoupling the pose estimation rate from the classification rate. The classifier thread pulls from this queue and appends each vector to `angle_sequence`, a `deque` whose maximum length N is read from the model metadata (30 or 60 frames depending on the model checkpoint, representing a three- or six-second window at 10 FPS). When full, the TCN produces a per-frame exercise prediction. Each prediction is also appended to a separate `exercise_window` (`deque`, maxlen = 1,280), which accumulates labels across an entire exercise set. When the classifier detects that the exercise has ended, a `Counter` tallies the labels in this window; if the dominant class exceeds a ratio threshold, the set is forwarded to RepNet.

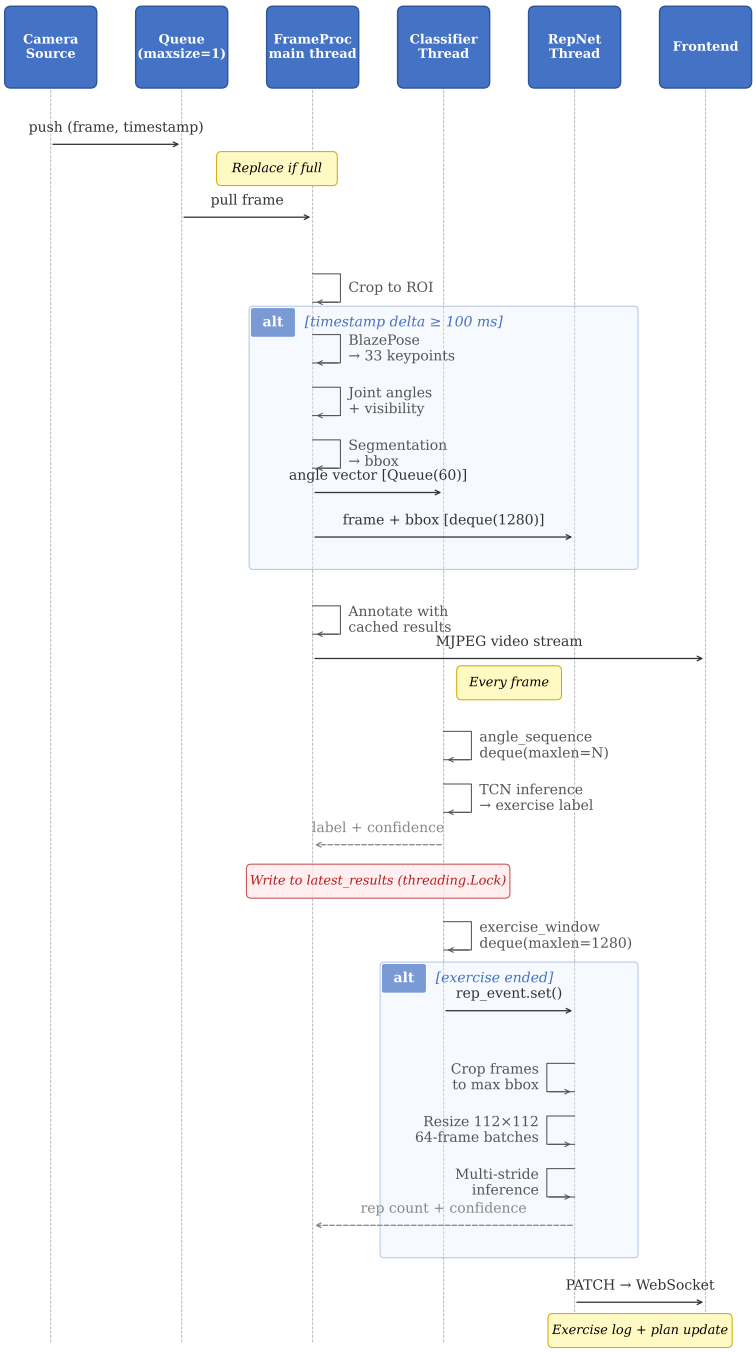


Figure 3.2: Sequence diagram of the real-time inference pipeline. The `FrameProcessor` crops and annotates every frame for the live MJPEG preview, but only runs pose estimation and classification when the timestamp delta exceeds 100 ms (10 FPS gate). The classifier and repetition counter operate in dedicated threads, writing results to a shared dictionary protected by a `threading.Lock`. Completed exercise sets are broadcast to the frontend via a REST callback.

In parallel, raw video frames and their bounding-box coordinates accumulate in a second `deque` with a maximum length of 1,280 entries (approximately two minutes at 10 FPS). Because RepNet requires the original spatial context to crop and resize each frame, these buffers store full-resolution frames with no compression. At Full HD resolution (1920×1080, 3 bytes per pixel), each frame occupies approximately 5.9 MB, so the buffer can grow to roughly 7.4 GB at capacity. On the 16 GB Apple M1 MacBook used for development and testing, this already consumes nearly half of the available RAM, making a larger buffer impractical. This constraint also motivates the use of the cameras' Full HD substream rather than their native 4K output: at 3840×2160, the same 1,280-frame buffer would require approximately 30 GB, far exceeding the available memory. In practice, the 1,280-frame cap limits the maximum exercise set duration to roughly two minutes, which is sufficient for common gym exercises. The classifier triggers the repetition counter thread via a `threading.Event`. The counter crops each buffered frame to the maximal bounding box, resizes the crops to 112×112 pixels, and feeds them to RepNet in batches of 64 frames at multiple temporal strides.

Thread Safety. The system employs a hybrid threading-asyncio architecture. The main FastAPI event loop distributes frames to three concurrent workers: the `FrameProcessor` (runs in the event loop via `asyncio.to_thread`), the classifier thread, and the repetition counter thread. CPython's Global Interpreter Lock guarantees that individual `deque.append()` and iteration operations are atomic at the bytecode level, preventing memory corruption during concurrent access [46]. This guarantee is specific to CPython and would not hold under alternative interpreters such as PyPy. For the shared results dictionary, which both the classifier and repetition counter write to, an explicit `threading.Lock` provides mutual exclusion.

3.4.3 Inference Pipeline

The FastAPI application, defined in `main.py`, manages startup and shutdown lifecycle events. On startup, it instantiates a camera source through a factory pattern that selects among three input modes:

- **Frontend:** webcam frames streamed from the React client over WebSocket, used for cloud demonstrations.
- **IP camera:** RTSP stream from a networked security camera, used for in-gym operation.
- **File:** OpenCV playback of a pre-recorded video at 10 FPS, used for reproducible testing and prototyping.

Each camera source pushes frames to the `FrameProcessor`, an object from `livestreaming.py` that loads all neural network models and orchestrates the inference pipeline. The `FrameProcessor` runs pose estimation, exercise classification, and repetition counting asynchronously, then publishes annotated frames and results to the frontend. Hyperparameters such as confidence thresholds can be modified at runtime from the frontend without restarting the server. Since the models operate at 10 FPS but the input video may arrive at a higher frame rate, the processor uses timestamps to feed the models at the target rate while annotating results on every frame. When pose estimation detects no person in the frame, the remainder of the inference pipeline is skipped entirely for idle efficiency. The administrator can also define a crop region from the frontend to select a specific user in a multi-person scene before the inference pipeline processes the frame.

3.4.4 Frontend: React

The frontend evolved from a minimal HTML page served by FastAPI, through a Flask application with Jinja2 templates, to a React 19 single-page application. The final migration was motivated by the need for independent component re-rendering: server-side templating forced full-page refreshes on every control change, which was incompatible with streaming video at 10 FPS alongside live controls. React's component model solved this friction: components re-render independently, so adjusting a confidence slider does not interrupt the video stream.

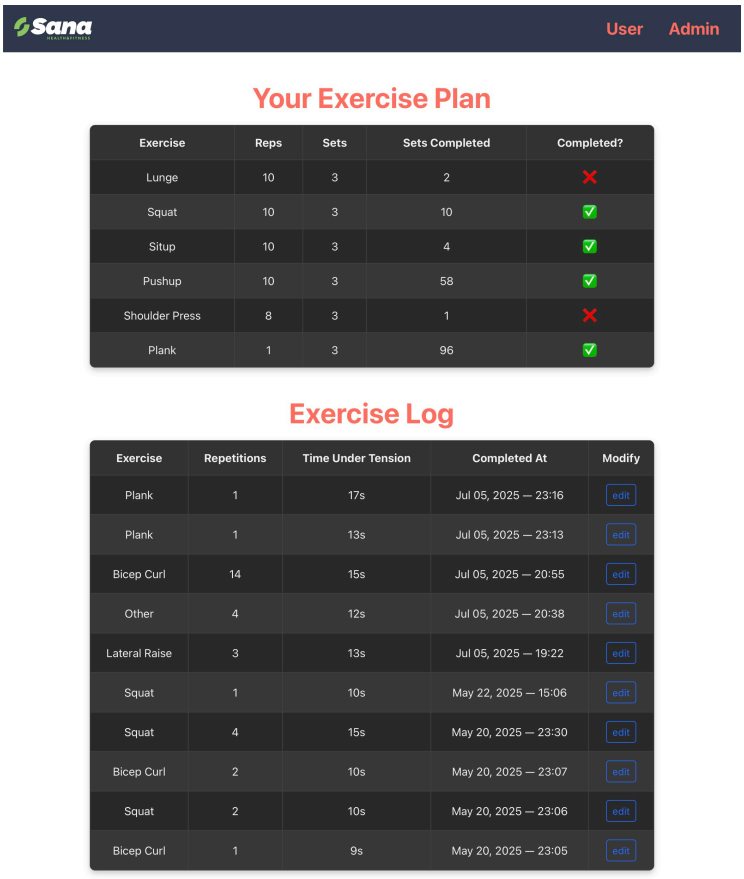


Figure 3.3: User page: the trainer-defined workout plan with completion status and an editable exercise log showing recognized exercises, repetition counts, and timestamps.

The production frontend uses React 19 [40], Bootstrap 5 for responsive

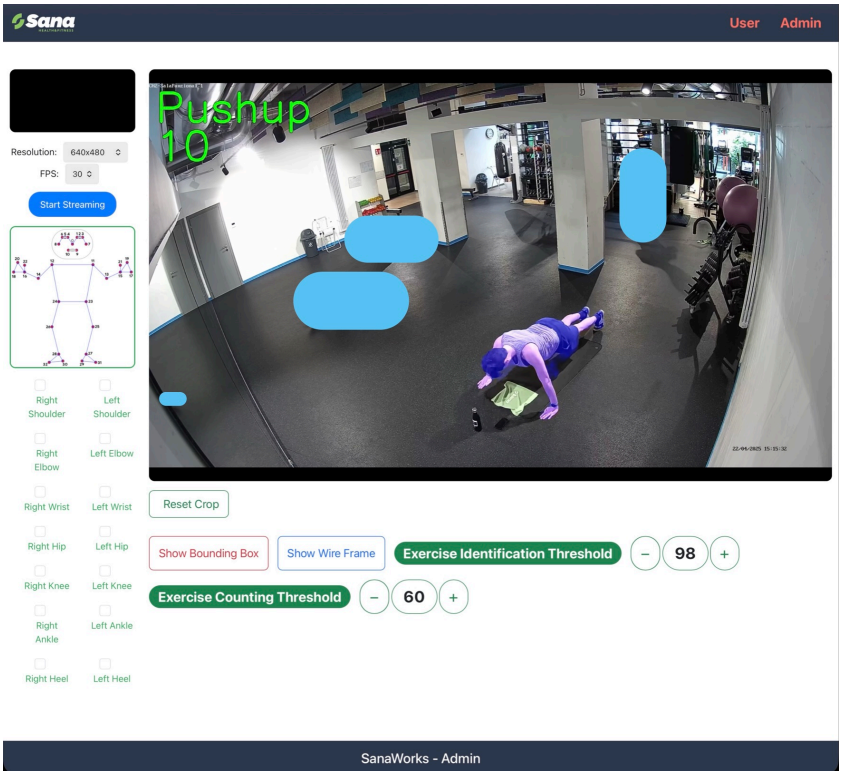


Figure 3.4: Admin page: live video stream with crop overlay, skeleton visualization, joint selector, and hyperparameter controls for confidence thresholds. The administrator can also stream video from a local device to the cloud deployment.

layout, and React Router 7 for client-side navigation. Table 3.2 summarizes the four pages.

Six reusable components compose the pages listed above, handling webcam capture, hyperparameter controls, crop region selection, joint visualization toggles, log editing, and WebSocket communication. The most architecturally significant is `useWebSocket.js`, a custom React hook that abstracts the WebSocket lifecycle: it opens a connection on mount, parses incoming JSON messages, and closes the connection on unmount. All real-time components share this hook rather than managing WebSocket state independently, ensuring consistent connection handling across pages. React's component model was essential for this application because components re-render independently: adjusting a confidence slider triggers a re-render only for the controls panel, not for the video stream, which would cause visible frame drops

Table 3.2: Frontend pages and their roles.

Page	Purpose
`IndexPage` (`/`)	Landing page listing all recognized exercises with reliability badges. Read-only.
`UserPage` (`/user`)	Displays the trainer-defined exercise plan and the exercise log with timestamps. Users can edit or delete log entries via a modal form. Updated in real time over WebSocket.
`AdminPage` (`/admin`)	Live video stream with interactive controls: webcam capture, bounding-box crop overlay, hyperparameter sliders for classifier and RepNet confidence thresholds, wireframe and bounding box toggles, and a joint selector for skeleton visualization.
`DemoPage` (`/demo`)	Simplified view exposing only the hyperparameter controls, used for quick demonstrations.

if it were interrupted.

The frontend communicates with the backend through REST endpoints for infrequent stateless operations and WebSocket connections for all real-time streaming. The motivation for this dual-protocol design and the full list of endpoints are discussed in Section 3.5. All WebSocket messages use a consistent `{type, data}` JSON envelope. During development, the frontend can be accessed from any device on the local network by navigating to the host machine's IP address, enabling testing from smartphones and tablets without cloud deployment. For production, the React application is containerized with Node.js 23 and served by the `serve` package. An entrypoint script injects the backend URL as an environment variable at container startup, allowing a single Docker image to target both local development and cloud environments without rebuilding.

3.4.5 Database: PostgreSQL

We initially planned to use TimescaleDB, a PostgreSQL extension purpose-built for time-series data. However, we found that TimescaleDB's performance advantages over standard PostgreSQL materialize primarily at scale, with millions of rows and complex time-windowed aggregations. For our use case of simple inserts and lookups across two small tables, standard PostgreSQL is equally performant and avoids the additional operational complexity. Since the backend accesses the database through the `psycopg` adapter with plain SQL queries, migrating to TimescaleDB or another PostgreSQL-compatible store remains straightforward if future scaling demands it. The database provides persistent storage across sessions and enables both per-user workout history and aggregated analytics.

Table 3.3: Database schema. The `exercise_plan` table (top) stores the trainer-defined workout. The `log` table (bottom) records completed sets.

`exercise_plan`			
Column	**Type**	**Default**	**Purpose**
`name`	`TEXT` (PK)	—	Exercise name (unique identifier)
`rep`	`INTEGER`	0	Target repetitions per set
`set_assigned`	`INTEGER`	0	Number of sets to complete
`set_completed`	`INTEGER`	0	Number of sets completed so far
`complete`	`BOOLEAN`	FALSE	Whether all sets are done
`completed_at`	`TIMESTAMP`	NULL	Timestamp when exercise was finished
`log`			
`id`	`SERIAL` (PK)	auto-increment	Log entry identifier
`completed_at`	`TIMESTAMP`	NOT NULL	When the set was completed
`exercise`	`VARCHAR(255)`	NOT NULL	Exercise name
`count`	`INT`	NOT NULL	Repetitions performed
`under_tension_seconds`	`INT`	0	Time under tension

The database consists of two tables (Table 3.3). The `exercise_plan` table

stores the trainer-defined workout that the user must complete. The `log` table records every completed exercise set as a time-stamped entry.

The two tables interact during a workout session through a single backend endpoint. When the inference pipeline determines that the user has completed a set, the `PATCH /exercise/{name}/{count}/{tut}/complete` handler updates both tables in sequence: it increments the plan's `set_completed` counter and then inserts a timestamped row into the log. The two operations are separate database calls rather than a single transaction, which is acceptable here because a partial failure (e.g., the log insert succeeding but the plan increment failing) is correctable through the user-facing edit endpoints. The updated plan and log are then broadcast to the frontend over WebSocket so the User page reflects the change immediately. If `set_completed` reaches `set_assigned`, the plan row is marked as complete. The two tables therefore serve complementary roles: the plan tracks progress toward the trainer's targets, while the log provides an append-only audit trail of every recognized set.

All database access is centralized in a single module, `communicate.py`, which exposes eight functions through a context-managed connection pattern. Table 3.4 summarizes these operations. The `exercise_plan` table is populated by the trainer before the session begins and is only modified by the inference pipeline during the workout; there is no user-facing endpoint to add, remove, or edit plan entries at runtime. The `log` table, by contrast, is written automatically by the pipeline but is correctable by the user: the frontend exposes PUT and DELETE endpoints so the user can fix a miscount or remove a false positive.

One notable design decision concerns fuzzy matching in the plan update function. `increment_exercise_plan_entry()` only increments the plan if the detected repetition count falls within $\pm$`fuzzy` of the plan's target. With the default tolerance of 3, a plan targeting 10 repetitions will accept any count

Table 3.4: Database operations defined in `communicate.py`.

Function	Table	Operation
`add_exercise_plan_entry()`	`exercise_plan`	INSERT
`remove_exercise_plan_entry()`	`exercise_plan`	DELETE
`list_exercise_plan_entries()`	`exercise_plan`	SELECT
`increment_exercise_plan_entry()`	`exercise_plan`	UPDATE
`add_exercise_log_entry()`	`log`	INSERT
`list_log_entries()`	`log`	SELECT
`update_exercise_log_entry()`	`log`	UPDATE
`delete_exercise_log_entry()`	`log`	DELETE

between 7 and 13, accommodating small counting errors from RepNet without requiring the user to manually correct every entry. As shown in Section 5.3, RepNet's mean absolute error of 2.1 repetitions is well within this tolerance window. All eight functions share a context-managed connection pattern: `get_db_cursor()` opens a connection, yields a cursor, commits on success, rolls back on failure, and closes the connection in all cases.

3.5 Communication Protocol

3.5.1 WebSocket Streaming

The original design used only REST endpoints, which are sufficient for request-response interactions such as fetching the exercise plan or deleting a log entry. However, REST proved inadequate for two categories of communication that emerged during development. First, the Admin page needs to stream webcam frames to the backend at up to 30 FPS. An HTTP POST per frame would incur connection overhead on every frame and overwhelm the server with short-lived requests. Second, the User page needs to receive live updates (new exercise completions, plan progress) as they happen, without polling. HTTP polling at a fixed interval would either waste bandwidth when idle or miss updates between polls.

WebSockets solve both problems: a single persistent connection supports

continuous bidirectional data flow with minimal overhead. After an initial prototype using HTTP POST for button and slider state, we migrated all real-time communication to WebSockets, which eliminated the reliability issues we had observed with repeated POST requests timing out after sustained use. Table 3.5 lists the six WebSocket endpoints. The inconsistent naming convention between `/ws/input` (slash-separated) and the remaining endpoints (hyphen-separated) is minor technical debt: `/ws/input` was the first WebSocket endpoint implemented, and the naming convention was only standardized for subsequent endpoints.

Table 3.5: WebSocket endpoints used by the frontend.

Endpoint	**Direction**	**Purpose**
`/ws/input`	Client → Server	Streams webcam frames as WebP blobs at the selected FPS.
`/ws-crop`	Client → Server	Sends normalized bounding-box crop coordinates.
`/ws-buttons`	Client → Server	Sends hyperparameter updates (thresholds, toggles).
`/ws-joints`	Client → Server	Sends joint visibility selections.
`/ws-user`	Server → Client	Pushes exercise plan updates to the User page.
`/ws-log`	Server → Client	Pushes exercise log updates to the User page.

3.5.2 REST API

The REST endpoints handle stateless, infrequent operations where a persistent connection is unnecessary, following the Representational State Transfer architectural style defined by Fielding [19]. A `/health` endpoint supports cloud monitoring by reporting server status and processing rate. The `PATCH` endpoint is invoked automatically by the inference pipeline when an exercise set is completed above the confidence threshold; the remaining endpoints are accessible to the user through the frontend.

Table 3.6 lists the REST endpoints exposed by the FastAPI backend.

Table 3.6: REST API endpoints exposed by the backend.

Method	Path	Purpose
GET	`/exercise-plan`	Retrieve the current exercise plan from the database.
GET	`/exercise-log`	Retrieve the exercise log history.
PUT	`/exercise-log/{id}`	Update a log entry's repetition count and time under tension.
DELETE	`/exercise-log/{id}`	Delete a log entry.
PATCH	`/exercise/{name}/{count}/{tut}/complete`	Mark a set as complete: increment the plan, log the entry, and broadcast the update.
POST	`/update_hyperparameters`	Update classifier and RepNet confidence thresholds and visualization toggles.
GET	`/video_stream`	Stream annotated video frames as MJPEG.
GET	`/health`	Return server status and average frame processing rate.

3.6 Cloud Deployment

After the May 6 stakeholder demonstration, the business partner confirmed that cloud deployment was the next priority: the system needed to be accessible from any web browser, including mobile devices, without requiring the audience to install software or connect to the gym's local network. The architecture agreed upon at that meeting called for two Docker containers and a hosted database on GCP or AWS, with the services scaling to control costs.

3.6.1 Docker Containerization

The system is packaged as two Docker [15] containers, orchestrated locally with Docker Compose. Each container isolates its runtime and dependencies so that the same images run identically on the development MacBook, the Ubuntu training workstation, and GCP Cloud Run.

- **backend-python** uses `nvidia/cuda:12.8.0-base-ubuntu24.04` as

its base image rather than a standard Python image, because the inference pipeline requires CUDA for GPU-accelerated pose estimation and classification. Python 3.10 is installed via the `deadsnakes` PPA, and all dependencies run inside a virtual environment. One notable workaround was required for MediaPipe: installing it with `pip install --no-deps mediapipe` to prevent it from pulling a conflicting version of OpenCV that would overwrite the CUDA-compatible build already present in the image.

- **frontend-react** is built from `node:23-slim`. The React application is compiled at build time with `npm run build`, and the resulting static files are served by the `serve` package. The backend URL is injected as an environment variable at container startup, allowing the same image to point at `localhost:8000` during development or at the Cloud Run backend URL in production.

Getting the two containers to communicate locally required debugging a subtle networking issue: the FastAPI server bound to `localhost` by default, which is unreachable from a sibling container. Changing the `uvicorn` host to `0.0.0.0` resolved the problem. The backend container reserves one NVIDIA GPU through the Compose `deploy.resources.reservations.devices` directive. Since the development machine runs an Apple M1 (ARM) but GCP Cloud Run executes on `amd64`, building images for production required cross-compilation with `docker buildx --platform linux/amd64`.

3.6.2 Google Cloud Platform

The cloud infrastructure is defined entirely in Terraform [26], an infrastructure-as-code tool that provisions resources declaratively. The Terraform configuration, stored in the repository's `terraform/` directory, creates the following resources in the `europe-west1` region:

- **Two Cloud Run Gen2 services.** The backend service, `backend-python`, is provisioned with 4 vCPUs, 16 GiB of RAM, and one NVIDIA L4 GPU. The frontend service (`frontend-react`) runs with default Cloud Run resources and no GPU. Both services are configured with public IAM access so that any browser can reach them. Phase 2 will implement authentication. The frontend's `REACT_APP_API_URL` environment variable is set automatically to the backend's Cloud Run URL through a Terraform output reference, eliminating manual URL configuration. A bash script was written to inject this URL into the frontend at deploy time.

- **Cloud SQL.** A managed PostgreSQL instance hosts the exercise plan and log tables. The backend connects to it over a private IP address through a VPC Access Connector, which bridges the serverless Cloud Run environment to the project's VPC network. A dedicated IAM binding grants the Cloud Run service account the `cloudsql.client` role.

- **VPC Access Connector.** A Serverless VPC Access connector on subnet `10.8.1.0/28` routes traffic from both Cloud Run services to the Cloud SQL instance without exposing the database to the public internet. The connector is configured with 2 to 3 instances for redundancy.

The cloud deployment unfolded over one intensive week. On July 1, CUDA was confirmed working inside the Docker container on the Ubuntu workstation. On July 2, the Terraform configuration was written. On July 3, after debugging IAM and network permissions, the frontend and backend Cloud Run services communicated for the first time. This milestone confirmed that the full pipeline could execute end-to-end in the cloud: webcam capture in the browser, pose estimation and classification on the cloud GPU, and results displayed back in the frontend. On July 6, the database connection was established through the VPC connector, completing the deployment.

Security Considerations. The current deployment leaves several security concerns unaddressed. The RTSP stream URL embeds NVR credentials in plaintext (an early leak into version control is documented in Appendix A). The REST and WebSocket endpoints carry no authentication: any client with the URL can read the exercise plan, modify log entries, or stream video. The Cloud Run services are configured with public IAM access. These gaps are acceptable for a Phase 1 prototype demonstrated in controlled settings, but production deployment would require TLS for all connections, token-based API authentication, and role-based access control. These are scoped as Phase 2 work items.

Ethical Considerations. Deploying a vision-based monitoring system in a gym raises ethical questions beyond data privacy. Continuous automated observation of physical activity may create a sense of surveillance that affects member behavior, comfort, or willingness to use the facility. Unlike a personal fitness app that the user chooses to activate, a ceiling-mounted camera system operates passively and may monitor individuals who are unaware of or have not consented to the analysis. Production deployment would require clear signage, an opt-out mechanism, and transparent communication about what data is collected, how it is processed, and who can access it. Our Phase 1 prototype was operated only during controlled demonstrations with informed participants, but scaling to continuous gym-wide monitoring would need to address these concerns through both technical safeguards (such as on-device processing that avoids transmitting video) and institutional review of the consent model.

This chapter has described the system from its physical infrastructure through its cloud deployment. Each design decision, from the push-based frame architecture to the multi-stride repetition counter, was motivated by the constraints of a real gym environment and the goal of delivering results while the user is

still exercising. With the architecture in place, Chapter 4 turns to the methodology: how we collected the training data, engineered the features, trained the classifiers, and composed the inference pipeline that brings the system to life.

Chapter 4

Methodology

This chapter describes the systematic process of decisions, experiments, and iterations that led to the final system. Section 4.1 describes how we collected and organized the exercise video dataset. Section 4.2 details the processing pipeline that transforms raw video into normalized joint angle features suitable for classification. Section 4.3 presents the classification architectures we evaluated, from scikit-learn baselines to the Temporal Convolutional Network. Section 4.4 describes the integration of RepNet for repetition counting. Section 4.5 documents the real-time inference pipeline that connects these components into a live system.

4.1 Data Collection

Training a pose-based exercise classifier requires a dataset of exercise videos recorded under conditions that match the deployment environment. We therefore recorded the majority of our training data at the partner gym using the same camera infrastructure described in Section 3.3, supplemented by a public video dataset [1] for additional variety.

No standardized benchmark dataset exists for pose-based exercise recognition [48], and the datasets that are publicly available each present significant

limitations. Image-only datasets such as the Kaggle "Workout/Exercise Images" collection provide static frames with no temporal information, making them unsuitable for classifiers that rely on motion patterns. Video datasets that do exist are either small (the Muhannad Tuameh dataset [62] covers only 5 exercises and provides pre-extracted landmarks rather than raw video), target a different task entirely (Fitness-AQA [42] targets form quality assessment with only 3 exercise classes), or skew heavily toward machine-based exercises where gym equipment occludes the body. The Hasyim Abdillah "Workout/-Fitness Video" dataset [1], for example, contains 652 videos across 22 exercise categories, but many categories involve machines such as the chest fly machine, leg extension, lat pulldown, and tricep pushdown that occlude the torso and limbs. We selected only 9 of the 22 categories that involve free weights or bodyweight movements compatible with our pose-based approach, yielding 307 usable videos. The remaining majority of our training data required custom collection under real gym conditions.

4.1.1 Recording Setup

We recorded exercise videos over multiple sessions between February and May 2025 using two sources: the gym's network video recorder (NVR) and a handheld smartphone. The NVR captures at 4K resolution from ceiling-mounted cameras. We primarily used two camera channels: CH2, mounted at approximately four meters and providing a wide-angle view of the gym floor, and CH3, mounted closer to the exercise area and providing a tighter framing of individual subjects. Recordings from CH2 often included multiple people in the frame and required cropping, while CH3 provided cleaner single-person footage but the area was often in use.

Five gym staff members contributed to the recording effort. Martina, a gym trainer with a Bachelor's degree in Physical Science (Laurea in Scienze Motorie), performed many of the exercises with proper form and advised on

which joint angles are most informative for distinguishing exercise types. Her input directly shaped the selection of the 14 joint angles described in Section 4.2.2.

We uploaded recorded videos from the NVR and trimmed them to individual exercise sets using QuickTime. Each trimmed video contains a single person performing a single exercise type for 8-12 repetitions, typically lasting 10 to 60 seconds.

4.1.2 Exercise Selection

We organized the dataset into 16 exercise categories selected to cover the most common movements observed at the partner gym. Several categories contain subcategories that share the same gross body motion but differ in equipment or grip. For instance, the "squat" category includes air squats (bodyweight), barbell squats (bar on shoulders), and goblet squats (weight held at chest). The classifier treats these subcategories as a single class because their joint angle trajectories are nearly identical from the camera's perspective. In practice, however, the classifier is sensitive to equipment-driven variants that change the arm configuration while preserving the primary movement pattern. Section 6.3 discusses this limitation in detail.

Table 4.1 lists all 16 categories with their subcategories, video counts, and total frame counts at 10 frames per second. The dataset exhibits a long-tailed distribution (Figure 4.1): the "Other" class dominates with 109 videos (35% of all frames), while tail classes such as lateral raise and leg raise contain fewer than 10 videos each. This imbalance reflects the natural frequency of exercises at the gym and is partially addressed through data augmentation (Section 4.2.3).

As discussed in Section 2.1.3, a classifier deployed in an open-world setting must handle non-exercise activity. We constructed four subcategories to represent common gym behavior that is not exercise: standing (idle between

sets), walking (moving between stations), sitting (resting on a bench), and crop (frames where only a portion of the body is visible). Training with explicit negative examples converts the open-world recognition problem into a closed-world classification problem with a learned rejection boundary.

Table 4.1: Exercise categories in the Sana dataset. Video and frame counts refer to the original (non-augmented) dataset. Frame counts are measured at 10 FPS.

Category	**Subcategories**	**Videos**	**Frames**
Other	crop, sit, stand, walk	109	41,361
Squat	air, barbell, goblet	31	11,755
Row	barbell, dumbbell	32	10,996
Shoulder press	barbell, dumbbell	28	10,716
Lunge		20	9,488
Situp		16	6,262
Deadlift		15	4,960
Pushup	regular, wide	14	4,133
Bicep curl		15	3,521
Donkey kick		11	2,742
Lateral lunge		4	2,563
Plank		5	2,251
Step up		6	2,239
Crisscross		11	1,747
Leg raise		7	1,707
Lateral raise		6	1,217
Total		**330**	**117,658**

4.1.3 Data Labeling and Organization

Each video was manually labeled by placing it into a directory named after its exercise category. The resulting CSV files follow Tidy Data principles [67]: each row represents one frame and each column represents one feature (a joint angle or its visibility score), with one file per exercise video. This structure simplified data loading and made it straightforward to verify label correctness by inspecting file paths.

The labeling process exposed several data quality challenges. First, approximately 10% of videos required relabeling after visual review revealed incorrect exercise assignments, particularly between visually similar exercises

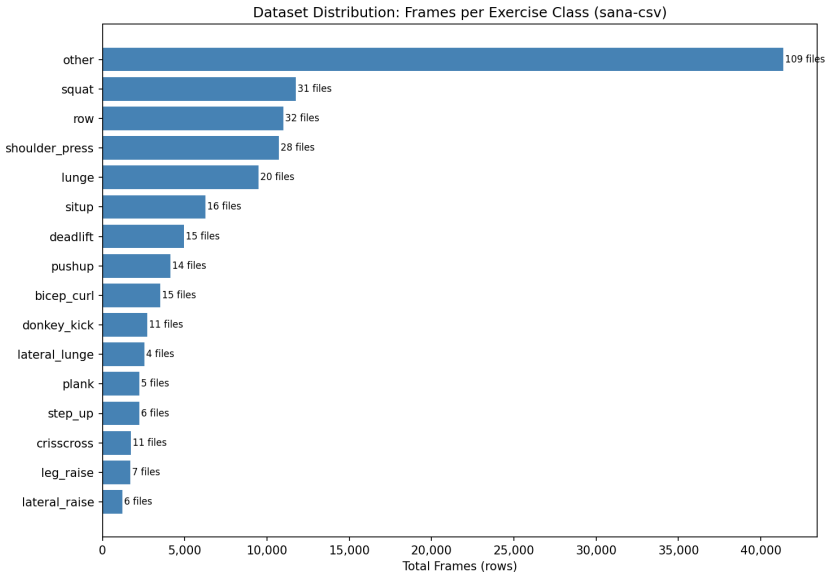


Figure 4.1: Distribution of frames across exercise categories in the Sana dataset (before augmentation). The "Other" class is intentionally large to provide robust rejection of non-exercise activity.

such as barbell rows and deadlifts. Second, videos recorded from CH2 (the far camera) frequently contained other gym members in the background; since MediaPipe's single-person tracker can lock onto the wrong individual, we cropped these videos to isolate the target subject. Third, some NVR recordings were rotated 90 degrees, requiring preprocessing before pose extraction.

The raw video dataset totals approximately 18 GB across 330 videos. After pose extraction and angle computation (Section 4.2), the intermediate representation compresses to 60.5 MB of CSV files, a 300× reduction that makes classifier re-training to add more exercises and variations economically feasible.

4.1.4 Dataset Availability

We release the Sana (original) CSV dataset publicly [14]. The released data contains only normalized joint angles and visibility scores; no raw video, images, or metadata are included. The remaining components of the combined training set are reproducible: the augmented variant can be regenerated using

the pipeline in Section 4.2.3, and the Kaggle component can be obtained from the Abdillah dataset [1] and processed through the same BlazePose extraction pipeline.

Because the `VideoPoseProcessor` was configured with `num_poses=1`, MediaPipe tracked exactly one person per video, so background gym members' keypoints were never extracted or stored. Re-identification from normalized angular sequences is unlikely, though we did not conduct a formal privacy impact assessment. We obtained written GDPR consent from all six demonstrating participants appearing in the recordings.

In addition to our own data, we incorporated 307 videos across 9 free-weight-compatible exercise categories from the Abdillah dataset [1], selected from a pool of 22 categories. Table 4.2 summarizes the three dataset components and the augmented variant described in Section 4.2.3.

Table 4.2: Dataset components used for training. All datasets use the same 28-feature CSV format (14 joint angles + 14 visibility scores) at 10 FPS.

Dataset	**Files**	**Frames**	**Size (MB)**	**Categories**
Sana (original)	330	117,658	60.5	16
Sana (augmented)	959	354,007	182.0	16
Kaggle	307	26,986	14.1	9
Combined	**1,596**	**498,651**	**256.6**	**16**

4.2 Data Processing Pipeline

The data processing pipeline transforms raw exercise videos into compact feature vectors suitable for classification. The pipeline consists of three stages: pose keypoint extraction, joint angle computation, and data augmentation. We implemented the pipeline in the `VideoPoseProcessor` class, which processes each video independently and outputs a CSV file of joint angles.

4.2.1 Video to Pose Keypoints

We process each video with MediaPipe's heavy pose landmarker model [24, 5]. The processor resamples every input video to 10 FPS, regardless of its original frame rate, by computing a frame interval and selecting every n-th frame. We chose 10 FPS because it matches the native recording rate of the gym's IP cameras, and it is sufficient to capture even fast exercises, which rarely complete a repetition in under one second. At this frame rate, even the fastest exercise produces at least 10 frames per repetition, providing sufficient temporal resolution for the TCN's kernel size of 3. This maintains a consistent temporal resolution across all data sources, whether recorded by the NVR system or by a handheld smartphone at a higher native frame rate.

Since the NVR records at 4K resolution (3840×2160), the processor downscales frames to 640×360 pixels before passing them to MediaPipe, reducing per-frame memory by over $35\times$ with no effect on pose estimation accuracy because MediaPipe internally rescales all input to 256×256 for its neural network.

The processor converts each selected frame from BGR to RGB and runs MediaPipe's video detection method with the frame's timestamp in milliseconds. The detection confidence thresholds are set to 0.6 for pose detection (above MediaPipe's default of 0.5, raised to prevent false detections on gym equipment) and 0.5 for both pose presence and tracking confidence. When MediaPipe fails to detect a pose in a frame, that frame is skipped and recorded in a dictionary keyed by frame index. This design preserves the temporal alignment of successfully detected frames while allowing downstream code to identify and handle gaps.

The output for each detected frame is a set of 33 keypoints, each carrying normalized (x, y) coordinates within the frame, a z coordinate encoding relative depth, a visibility score (probability that the keypoint is both in-frame

and unoccluded), and a presence score (probability that the keypoint is in-frame, regardless of occlusion). We retain only the x, y, and visibility values for downstream processing, discarding z (unreliable for monocular video, as discussed in Section 2.2) and presence (redundant with visibility for our application).

4.2.2 Keypoint to Joint Angles

Raw keypoint coordinates are sensitive to the subject's position within the frame, their distance from the camera, and their body proportions. A person performing squats in the left half of the frame produces different (x, y) coordinates than the same person performing the same exercise on the right, even though the movement is identical. We therefore convert the 33 keypoints into 14 joint angles (Table 2.4), each defined by a triplet of keypoints and measured at the center (vertex) joint using the two-argument arctangent:

$$\theta = |\text{atan2}(c_y - b_y,\ c_x - b_x) - \text{atan2}(a_y - b_y,\ a_x - b_x)| \tag{4.1}$$

where (a_x, a_y), (b_x, b_y), and (c_x, c_y) are the coordinates of the three keypoints defining the angle, with b as the vertex. We clamp the result to $[0°, 180°]$ and normalize by dividing by 180, yielding a value in $[0, 1]$. This normalization makes the features invariant to the subject's height, limb length, position in the frame, and distance from the camera. Alongside each angle, we retain the mean visibility score of its three constituent keypoints, producing a 28-dimensional feature vector per frame (14 angles + 14 visibility scores).

The central hypothesis of our classification approach is that each exercise produces a unique temporal fingerprint in the joint angle domain. A squat, for instance, produces a periodic pattern in the knee and hip angles that differs qualitatively from the shoulder and elbow patterns of a bicep curl. Although the 2D projected angles computed from monocular video differ from the true 3D joint angles, the temporal patterns remain discriminative because the same

exercise viewed from a consistent camera angle produces a consistent 2D projection (Figure 4.2).

4.2.3 Data Augmentation

The long-tailed class distribution shown in Figure 4.1 is an intentional design choice: in deployment, the classifier will encounter far more non-exercise activity than actual exercises, so the "Other" class should dominate the training set. To increase the diversity and volume of training data without the expense of collecting additional videos (coordinating gym staff, scheduling recording sessions), we applied data augmentation at the video level before pose extraction [51]. Spatial augmentations additionally simulate alternative camera perspectives, improving the system's flexibility for deployment in different gym layouts.

We generated three augmented variants per original video, targeting 990 augmented files (330 $\times$ 3). In practice, 31 augmented files were lost: 15 because a misspelled directory name caused the pose extraction step to skip one subdirectory, 12 because all three augmented variants of four lunge videos degraded MediaPipe's pose detection below the confidence threshold, and 4 because individual augmented variants of other videos similarly failed detection. The pipeline logs these failures silently and continues, yielding 959 successful augmented files and a total of 1,289 (330 originals + 959 augments).

The augmentation pipeline applies a randomized sequence of transforms, each activated with 50% probability:

1. **Horizontal flip:** mirrors the frame left-to-right, simulating the exercise performed from the opposite side. This is important because our gym data was recorded predominantly from one camera angle and we do not want to bias the model to favor one side of the body more than the other.

2. **Rotation** ($\pm 10°$): simulates minor variations in camera tilt.

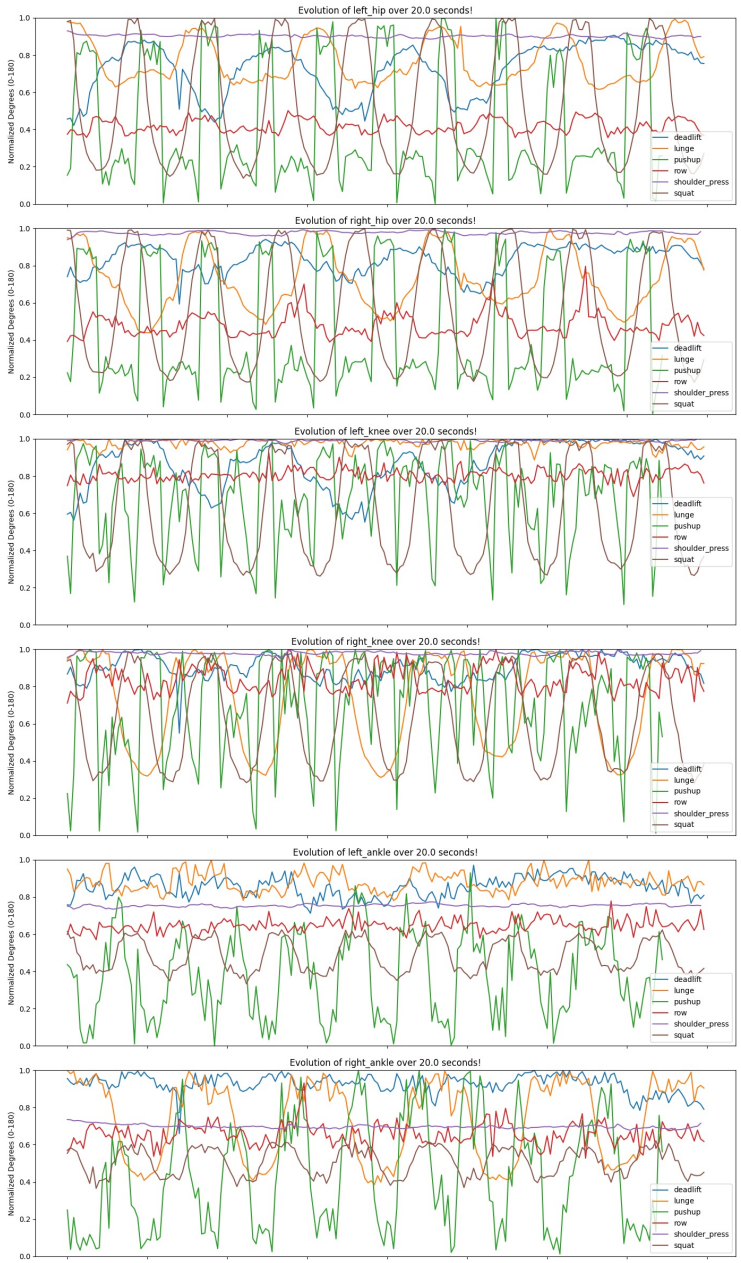


Figure 4.2: Joint angle time series for seven exercise categories over 20 seconds at 10 FPS. Each row shows one joint angle (left/right hip, knee, and ankle); each colored line represents a different exercise. The distinct temporal patterns across exercises validate joint angles as a discriminative intermediate representation.

3. **Shear** ($\pm 15\%$): simulates perspective distortion from different viewing angles.

4. **Brightness and contrast:** multiplies pixel values by a factor in $[0.5, 2.0]$ and adds a shift in $[-50, 50]$, simulating different lighting conditions.

5. **Salt and pepper noise:** randomly sets pixels to black or white, simulating sensor noise and partial occlusion.

6. **Temporal reversal:** plays the video backward, doubling the effective number of motion trajectories.

7. **Temporal resampling:** downsamples (ratio $[0.5, 1.0]$) or upsamples (ratio $[1.0, 2.0]$) the frame sequence, simulating exercises performed at different speeds.

8. **Temporal elastic deformation:** applies smooth, non-linear time warping to simulate natural variation in movement tempo within a single repetition.

Augmentation operates on the raw video frames before pose extraction. The augmented CSV dataset contains 354,007 frames (182 MB), approximately $3\times$ the original dataset.

The eight transforms serve two distinct purposes. Temporal transforms (reversal, resampling, elastic deformation) are critical for generalization: exercises performed with different weights or levels of fatigue are executed at different speeds, and the TCN must be wholly invariant to execution tempo. Spatial and photometric transforms (flip, rotation, shear, brightness, noise) do not directly affect the downstream joint angles when pose estimation succeeds, since we are not training BlazePose itself. However, they stress-test the pose estimator by simulating degraded recording conditions such as poor lighting, camera tilt, and sensor noise. Noisier pose estimation may improve TCN generalization by exposing the classifier to the kind of imprecise joint angles it will encounter in deployment.

We did not perform an ablation study comparing classifier performance with and without augmentation. The only indirect evidence that augmentation helped is the accuracy improvement between the May 5 TCN checkpoint (92.21%, trained on a smaller dataset) and the July 5 checkpoint (95.82%, trained on the full combined dataset including augmented data), but this improvement confounds augmentation with other changes such as the expanded window size and lower learning rate. We acknowledge the absence of a controlled ablation as a limitation.

4.3 Exercise Classification

Given the joint angle time series produced by the processing pipeline, the classification task is to assign an exercise label to a sliding window of T consecutive frames. The window size T controls a fundamental trade-off: longer windows guarantee that the model sees at least one complete repetition, improving classification certainty, but they also increase the perceptive lag between the camera feed and the classification output, which degrades the user experience. Larger windows also reduce the temporal precision of the crop fed to RepNet (degrading count accuracy) and diminish the precision of the Time Under Tension metric used principally for isometric exercises such as the Plank. Shorter windows reduce lag but may not contain enough temporal context to distinguish similar exercises.

4.3.1 Feature Engineering

For the scikit-learn baseline classifiers, we flatten each window of T frames $\times$ 28 features into a single vector of length $28T$. This representation discards temporal structure (the classifier treats the features as an unordered set) but is compatible with standard tabular classifiers. For the TCN, we preserve the temporal structure by feeding the window as a 2D tensor of shape $28 \times T$ (channels $\times$ time steps).

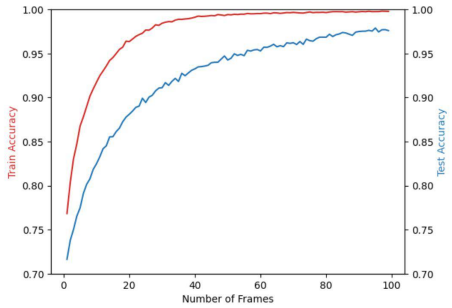


Figure 4.3: Decision Tree classification accuracy as a function of the sliding window size. Accuracy plateaus around 30 frames; the final system uses a 60-frame window ($T = 60$) to ensure at least one complete repetition is captured.

We swept the window size from 1 to 100 frames using a Decision Tree classifier and found that 30 frames (3 seconds at 10 FPS) offered the best balance between classification accuracy and perceptive lag (Figure 4.3). Shorter windows lack sufficient context to distinguish exercises with similar starting positions, such as squats and deadlifts. Longer windows introduce noticeable annotation delay and exclude short videos from the training set, since any video shorter than T frames cannot contribute a training sample.

During development, a feature extraction bug caused the training data processor to use only 7 left-side joint angles instead of the intended 14 bilateral angles, creating a mismatch with the live inference pipeline (Appendix A.3.1). After correcting the pipeline on March 18, 2025, all baseline accuracies were re-evaluated.

4.3.2 Baseline Models

We evaluated four scikit-learn [44] classifiers as lightweight baselines to determine whether the temporal structure of joint angles warranted more complex modeling before investing in the TCN. All baselines operate on the flattened feature vector described above. To enable a fair comparison with the TCN, the final baseline evaluation uses the same train/validation/test split (56% train, 33% validation, 11% test) described in Section 4.3.4, stratified by video to

prevent data leakage between temporally adjacent frames of the same video.

- **Decision Tree** [7]: achieved 97% test accuracy on the original Sana dataset with a window of 30 frames. The Decision Tree served as our primary development baseline due to its fast training time (under one second) and interpretable decision boundaries.
- **Linear Support Vector Classification (LinearSVC)** [12]: a linear SVM trained on the flattened feature vectors. LinearSVC was chosen over kernel SVM for its scalability to the large combined dataset.
- **k-Nearest Neighbors (KNN)** [13]: achieved high offline test accuracy among the baselines. However, the k-NN classifier failed to generalize to live camera feeds, producing noisy frame-by-frame predictions that flickered between classes. This failure motivated the move to a model that learns temporal features.
- **Histogram-based Gradient Boosting (HistGradientBoosting)** [21]: an ensemble of decision trees trained sequentially, where each tree corrects the errors of its predecessors. This was the strongest non-temporal baseline, approaching the TCN's accuracy.

Despite the high offline accuracy, all four baselines share a fundamental limitation: they treat the flattened window as a bag of features, ignoring the temporal ordering of frames. The k-NN classifier's poor real-time performance demonstrated that high test accuracy on randomly held-out sequences does not guarantee robust live inference. Chapter 5 presents the detailed performance comparison.

4.3.3 Temporal Convolutional Network

Our TCN architecture follows the generic framework of Bai et al. [3] with the three modifications described in Section 2.4.2: batch normalization instead of

weight normalization, symmetric (non-causal) padding, and a global average pooling head for sequence-to-label classification. The network consists of three `TCNBlock` modules followed by an adaptive average pooling layer and a fully connected output layer. Table 4.3 details the architecture.

Each `TCNBlock` contains two dilated 1D convolutions, each followed by batch normalization [30], ReLU activation, and dropout [55] at a rate of 0.2. The block includes a residual connection [27]: if the input and output channel dimensions differ, a 1×1 convolution projects the input to match. The dilation factor doubles at each level $(1, 2, 4)$, giving the final layer a receptive field of approximately 29 frames (the global average pooling layer described below aggregates features beyond this receptive field).

Table 4.3: TCN architecture. Each TCNBlock contains two dilated Conv1d layers with BatchNorm, ReLU, and Dropout (0.2), plus a residual skip connection. Padding is symmetric (non-causal).

Layer	**Channels**	**Kernel**	**Dilation**	**Receptive Field**
Input	28	–	–	1 frame
TCNBlock 0	$28 \to 64$	3	1	5 frames
TCNBlock 1	$64 \to 128$	3	2	13 frames
TCNBlock 2	$128 \to 256$	3	4	29 frames
Global AvgPool	256	–	–	full window
Fully Connected	$256 \to 16$	–	–	–

The input to the network is a tensor of shape $(\text{batch}, 28, T)$, where T is the sequence length. The complete model contains 436,304 trainable parameters. Listing A.1 in Appendix A reproduces the full source code.

Although the receptive field of the final convolutional layer spans 29 frames, the global average pooling that follows aggregates features across the entire input window. A 60-frame window therefore provides richer pooled statistics and ensures that at least one complete repetition is visible even for slower exercises. The architecture is inherently length-invariant, producing valid class logits for any input length $T \geq 1$.

4.3.4 Training Procedure

We implemented the training pipeline in PyTorch [43]. The `ExerciseDataset` class loads all CSV files from the combined dataset, extracts non-overlapping subsequences of T frames from each video, and assigns the label of the final frame in each subsequence.

The train, validation, and test sets are split by video identifier (56% train, 33% validation, 11% test) with stratification by exercise class, ensuring that frames from the same video never appear in different splits. All splits use a fixed random seed (42) for reproducibility. Because video lengths vary, the frame-level proportions differ from the nominal video-level ratio: the held-out test set contains 44,346 frames (8.9% of total frames) despite comprising approximately 11% of videos. The validation set is larger than typical (33% vs. the conventional 15%) because the split operates at the video level, not the frame level: with only 330 unique original videos across 16 unbalanced classes, a smaller validation set would leave some tail classes with too few validation videos for reliable per-class accuracy estimates during early stopping.

We did not apply class weights to the cross-entropy loss, relying instead on augmentation to partially mitigate the class imbalance. In retrospect, inverse-frequency weighting would likely have improved performance on tail classes such as Lateral Lunge and Crisscross; we discuss this as a limitation in Section 6.3.

Table 4.4 summarizes the training configuration. We trained the model with the Adam optimizer [35] using a learning rate of 10^{-4} and cross-entropy loss. Early stopping monitors the validation loss with a patience of 3 epochs. For the final model, validation loss plateaued after epoch 8 and training halted at epoch 11.

We trained the model four times between March and July 2025, each time incorporating an expanded or cleaned dataset. The final model (July 5) was

Table 4.4: TCN training configuration.

Parameter	Value
Optimizer	Adam [35]
Learning rate	10^{-4}
Batch size	32
Max epochs	30
Early stopping patience	3 epochs
Loss function	Cross-entropy
Sequence length	60 frames (6 s)
Train/val/test split	56% / 33% / 11%
Split strategy	By video ID, stratified by class

trained on the full combined dataset of 498,651 frames across 1,596 CSV files. Chapter 5 reports the accuracy progression across training rounds.

4.4 Repetition Counting

The exercise classifier identifies *what* exercise is being performed; the repetition counter determines *how many* repetitions have been completed. As discussed in Section 2.3, we adopted RepNet [16] as a class-agnostic, learned alternative to heuristic peak detection.

4.4.1 RepNet Architecture

As described in Section 2.3.2, RepNet is a class-agnostic repetition counter based on temporal self-similarity. It takes a fixed-length input of 64 RGB frames at 112×112 resolution and produces two outputs per frame: a period length (the estimated number of frames per repetition cycle) and a periodicity score (the confidence that the frame is part of a repeating action). The final repetition count is computed by summing the reciprocals of the per-frame period lengths, weighted by the periodicity confidence. We use the publicly available PyTorch implementation with pre-trained weights, without fine-tuning on our exercise data.

4.4.2 Integration with Classifier

RepNet and the exercise classifier operate sequentially. The classifier runs continuously on incoming frames, accumulating predictions into a sliding window. When the classifier's confidence drops below the threshold, indicating that the user has stopped exercising or switched exercises, the system increments a patience counter. Once the exercise window contains at least 90 frames (9 seconds of confident predictions), the system evaluates it: if a single class constitutes at least 60% of the predictions (the *exercise ratio threshold*), the system considers the exercise set complete and triggers RepNet on the buffered video frames. The 60% threshold allows for brief classification noise (for instance, a momentary "Other" prediction while the user adjusts their grip) while still requiring a clear majority before committing to a count. These thresholds (60% ratio, 90-frame minimum) were tuned through informal experimentation during development rather than a systematic sweep; we acknowledge this as a limitation in Section 6.3.

RepNet requires cropped video frames centered on the exercising person, not the full camera view. The system extracts a bounding box from MediaPipe's segmentation mask for each frame and maintains a coherent region of interest across the entire sequence by taking the union of all per-frame bounding boxes. This ensures that the crop does not jump between frames, which would confuse RepNet's temporal self-similarity computation.

To handle exercises of varying duration, the system evaluates RepNet at multiple temporal strides $[1, 2, 3, 4, 5, 6, 8]$. At stride s, the system selects every s-th frame, effectively simulating a lower frame rate. For each stride, RepNet processes the subsampled sequence in chunks of 64 frames and produces a count estimate. The system selects the stride whose periodicity confidence is highest, and the final count is scaled by the stride factor. This multi-stride approach allows RepNet to count both fast exercises (high stride captures the full set in 64 frames) and slow exercises (low stride preserves temporal detail).

When the frame count is not a multiple of 64, the system duplicates the final frame to pad the last chunk. Because this frame is static, it effectively freezes RepNet's counting, ensuring that padding does not inflate the repetition estimate.

An implementation issue arose during integration: an unstable bounding box computation that used per-frame extremes rather than a cumulative union caused the crop region to jump between frames (Appendix A.3.2). This was identified through visual inspection of RepNet's intermediate outputs and corrected before evaluation.

4.4.3 Time Under Tension

Repetition counting is meaningful for cyclic exercises such as squats and bicep curls, but some exercises are measured by duration rather than repetitions. Planks, for example, are held statically for a target number of seconds. To accommodate both paradigms, we implemented a Time Under Tension (TUT) metric that records the total duration (in seconds) for which the classifier detects a given exercise. TUT is computed using wall-clock time: the system records a timestamp when the first confident prediction enters the exercise window and computes the elapsed seconds when the exercise ends. The system reports both the repetition count and the TUT for every detected exercise set, allowing the frontend to display whichever metric is appropriate.

A threading bug in the TUT implementation caused the classification thread to hang after the user stopped exercising, resulting in incorrect duration measurements (Appendix A.3.3).

4.5 Real-Time Inference Pipeline

The previous sections describe the individual components of the pipeline: pose estimation, angle extraction, exercise classification, and repetition counting. This section describes how these components are composed into a real-time system that processes a live camera feed and produces annotated output.

4.5.1 Frame Processing

The `FrameProcessor` class (in `livestreaming.py`) is the central orchestrator of the inference pipeline. It receives frames from the camera source at the source's native frame rate, but performs pose estimation at a fixed rate of 10 FPS by comparing each frame's timestamp against the previous processed timestamp. Frames that arrive within the 100 ms processing interval are annotated with the most recent results and forwarded without running inference, ensuring a smooth viewing experience regardless of the inference frame rate.

At inference time, the backend reads the sequence length T from the model's metadata file and constructs a sliding window of the same size. The window advances one frame at a time, producing a new classification with each incoming frame at 10 FPS.

When MediaPipe fails to detect a person in a frame, the system stops feeding new joint angles to the classifier and new frames to the RepNet buffer. The exercise window and frame buffer retain their contents, so a brief tracking loss (for example, the subject turning away momentarily) does not reset the accumulated exercise data. When tracking resumes, new data flows into the pipeline and accumulation continues. If the subject leaves the frame entirely, the system displays a "No Person Detected" overlay and suppresses classification until a person reappears. One edge case is not handled: if a prolonged tracking loss occurs mid-exercise and the subject reappears performing a different exercise, stale predictions from the previous exercise remain in the exercise window and mix with new ones, potentially causing an incorrect

RepNet trigger.

On initialization, the `FrameProcessor` detects the available compute device using a fallback chain: CUDA (NVIDIA GPU) → MPS (Apple Metal) → CPU. Frame processing uses OpenCV [41] for image manipulation, including color space conversion, cropping, and annotation rendering.

4.5.2 Multithreading Strategy

Three threads are necessary because the pipeline's computational stages operate at fundamentally different rates. Pose estimation and angle extraction run at 10 FPS (100 ms per frame). The TCN classifier processes a full T-frame window per invocation. RepNet processes the entire accumulated frame buffer in a single batch, which can take several seconds. Without threading, each RepNet invocation would freeze the live video feed for the duration of its computation, and each TCN classification would block frame capture. Decoupling these stages allows the camera feed to remain smooth while classification and counting proceed asynchronously.

The inference pipeline uses three threads:

1. **Main thread:** receives frames from the camera source, performs pose estimation (MediaPipe), computes joint angles, and pushes results into a queue. This thread also handles frame annotation and output.

2. **Classifier thread:** consumes angle vectors from the queue, maintains the sliding window, runs the TCN classifier, and updates the exercise prediction and confidence score.

3. **RepNet thread:** waits on an event signal from the classifier thread. When triggered, it processes the buffered video frames through RepNet and updates the repetition count.

Communication between threads uses Python's `Queue` and `Event` primitives from the `threading` module. The angle queue between the main thread

and the classifier thread has a maximum capacity of 60 vectors (6 seconds at 10 FPS), allowing the classifier to absorb transient slowdowns without blocking the main thread. When the queue is full, new angle vectors are dropped to keep the live video preview responsive. Results are shared through a thread-safe dictionary protected by a `Lock`.

Python's Global Interpreter Lock (GIL) means that only one thread executes Python bytecode at a time. In practice, this limitation has minimal impact on our pipeline because the computationally expensive operations (MediaPipe inference, TCN forward pass, RepNet forward pass) execute in C/C++ extensions that release the GIL during computation.

4.5.3 Confidence Thresholding

The TCN outputs raw logits (unnormalized scores) for each exercise class. We convert these to probabilities using a softmax function with a temperature parameter K:

$$p_i = \frac{\exp(z_i/K)}{\sum_{j=1}^{16} \exp(z_j/K)} \tag{4.2}$$

where z_i is the logit for class i. We retain the standard softmax ($K = 1.0$) because the operating classification threshold of 60% is well below the typical confidence of correct predictions, making temperature tuning unnecessary.

The system applies two user-adjustable thresholds, exposed as sliders on the Admin interface (Figure 3.4):

- **Classification confidence threshold** (default 60%): the minimum probability required to accept a frame's prediction. Predictions below this threshold are discarded and contribute to the patience counter described in Section 4.4.2.
- **Repetition confidence threshold** (default 40%): the minimum periodicity score required for RepNet's count to be accepted.

With the pipeline fully assembled, every frame that contains a visible person triggers pose estimation, angle extraction, and classification (the tracking-loss behavior is described in Section 4.5.1). When a person is present and performing a recognized exercise, the system produces a classification. Chapter 5 quantifies the accuracy across all five classification architectures, evaluates the repetition counter, and measures the system's real-time latency under deployment conditions.

Chapter 5

Experiments and Results

This chapter evaluates the complete system across five classification architectures, nine end-to-end test sessions comprising 81 ground truth exercise sets, and five user demonstrations. Section 5.1 describes the hardware, software, and evaluation methodology. Section 5.2 compares exercise classification across model families, from Decision Tree baselines through the production TCN. Section 5.3 evaluates repetition counting accuracy on continuous gym video. Section 5.4 quantifies per-stage inference latency. Section 5.5 reports findings from live user demonstrations, and Section 5.6 catalogs failure modes encountered during development.

5.1 Experimental Setup

All training and offline evaluation were run on an Ubuntu workstation equipped with an NVIDIA GeForce RTX 4080 SUPER (16 GB VRAM) and CUDA 12.4. Real-time inference and latency measurements were performed on an Apple M1 MacBook Air with 16 GB of unified memory, using the Metal Performance Shaders (MPS) backend for PyTorch and the XNNPACK delegate for MediaPipe's TensorFlow Lite runtime.

The software stack consists of Python 3.9, PyTorch 2.6, scikit-learn 1.6, OpenCV 4.11, MediaPipe 0.10 (heavy pose landmarker), FastAPI 0.115, and

uvicorn 0.34. Training was conducted on the Ubuntu workstation and inference on the MacBook. The Ubuntu training environment was reflashed before version numbers could be recorded, so the versions listed here are from the macOS inference environment where the final models were evaluated. Both environments used the same model weights and checkpoint files.[1]

All classification models were trained and evaluated on the combined dataset described in Section 4.1: 1,596 CSV files containing 498,651 frame rows across 16 exercise classes, drawn from the original recordings, augmented videos, and the Kaggle dataset. The dataset was split at the video level (not frame level) with stratification by class, using a 56% train, 33% validation, 11% test ratio. This non-standard ratio is an artifact of the video-level stratification: because videos vary in length and each must be assigned entirely to one split, the exact proportions deviate from a target 60/20/20 split. Video-level splitting prevents data leakage: frames from the same exercise video never appear in both the training and test sets. For the sklearn baseline comparison, models were trained on the same combined dataset with 30-frame windows and a fixed random seed to ensure reproducibility.

5.2 Classification Results

5.2.1 Decision Tree Baseline

The Decision Tree served as the initial proof of concept, establishing that joint angle time series contain sufficient signal for exercise classification. On the Kaggle dataset (5 classes: jumping jack, pull-up, push-up, situp, squat), the Decision Tree achieved 97% test accuracy using 90-frame input windows, confirming the viability of the angle-based feature representation described in Section 4.2.2.

[1] The exact training environment versions may have differed from the inference environment; however, the exported model weights are deterministic once saved, so the evaluation results are reproducible regardless of training-time library versions.

Frame count tuning (Figure 4.3) revealed that accuracy plateaus around 30 frames (3 seconds at 10 FPS), with longer windows providing diminishing returns while increasingly excluding shorter videos from the dataset. At 150 frames, over 33% of videos in the Sana dataset were too short to produce a single sample. Based on this analysis, 30 frames was selected as the default window length for all subsequent sklearn models.

5.2.2 Sklearn Classifiers

Four scikit-learn classifiers were evaluated as baselines against the TCN: Decision Tree, Linear Support Vector Classification (LinearSVC), K-Nearest Neighbors (KNN, k=5)[2], and HistGradientBoosting. Each was trained on 30-frame windows from the combined dataset using the same train/test split as the TCN.

Table 5.1: Sklearn baseline results on the combined dataset (30-frame windows).

Model	**Train Acc (%)**	**Test Acc (%)**
Decision Tree	100.00	78.09
LinearSVC	78.59	66.80
KNN (k=5)	99.97	91.00
HistGradientBoosting	99.98	95.26

Table 5.1 reveals a clear hierarchy. The Decision Tree achieved 78.09% test accuracy despite perfect training accuracy, consistent with the Decision Tree's tendency to memorize the training set when no depth constraint is applied. LinearSVC achieved only 66.80% test accuracy, demonstrating that a linear decision boundary cannot separate the 16 exercise classes in the flattened angle feature space. KNN performed substantially better at 91.00%, but its near-perfect training accuracy (99.97%) similarly indicates overfitting. HistGradientBoosting achieved 95.26% test accuracy, approaching the TCN's performance. Appendix A.4 presents the per-class confusion matrices for all four baselines.

[2]The default scikit-learn value for k was retained without tuning, since KNN serves as an offline baseline rather than a deployment candidate.

However, these offline metrics obscure a critical limitation. When deployed for real-time inference, all four sklearn classifiers exhibited rapid prediction flickering: because each window is classified independently, minor variations in joint angles between consecutive frames cause the predicted label to oscillate even when the user continues performing the same exercise. This flickering prevents the downstream pipeline from accumulating stable frame buffers for RepNet and from measuring Time Under Tension, rendering the system non-functional regardless of aggregate accuracy. This motivated the adoption of a temporal model.

5.2.3 TCN Performance

The Temporal Convolutional Network was trained iteratively across four checkpoints over four months, with each iteration incorporating more data, more exercise classes, and refined hyperparameters.

Table 5.2: TCN training progression across model checkpoints.

Checkpoint	Dataset	Win.	Epochs	LR	Test Acc
March 27	sana-csv	30	10	10^{-3}	91.36%†
April 17	sana + augment	30	20	10^{-3}	97.07%††
May 5	combined	30	10	10^{-3}	92.21%*
July 5	combined	60	11	10^{-4}	**95.82%**

†Nine classes on sana-csv only; not directly comparable to 16-class models.
††Measured on the original 60/20/20 split at training time, not re-evaluated on the July 5 split.
*Originally reported 93.44% on its own 60/20/20 split. Re-evaluated on the July 5 split for fair comparison.

The March 27 model was the first successful deployment, trained on the original Sana recordings (no augmentation) with only 9 exercise classes. It confirmed that the TCN architecture could produce stable, non-flickering predictions in real time.

The April 17 model expanded to all 16 classes by incorporating the augmented dataset. It achieved 97.07% test accuracy on 67,878 test samples using a 60/20/20 split. This was the highest single-checkpoint accuracy, though the

comparison is not direct: the April 17 figure was measured on its own split at training time, while the July 5 and May 5 figures were re-evaluated on a common 56% / 33% / 11% split with a fixed random seed.

The production model (July 5) introduced two key changes over the May 5 checkpoint: 60-frame windows (6 seconds of temporal context) instead of 30, and a 10× smaller learning rate (10^{-4} vs 10^{-3}). Both checkpoints trained on the same combined dataset. Early stopping monitored validation loss with a patience of 3 epochs: training halted after epoch 11 because validation loss had not improved since epoch 8, at which point the best checkpoint was restored. The model achieved 95.82% overall test accuracy on the held-out test set (42,491 out of 44,346 correct predictions). Seven of the sixteen classes achieved perfect precision and recall on the test set: Leg Raise, Lunge, Pushup, Donkey Kick, Lateral Raise, Deadlift, and (for precision) Lateral Lunge. These are exercises with distinctive, repetitive motion patterns that produce unambiguous joint angle signatures.

Two classes require special attention, for different reasons. Lateral Lunge achieved only 37.8% recall, with 62% of its test samples misclassified as Squat, as discussed further in Section 5.6. Plank achieved 70.8% precision: the model correctly identified all planks, but also misclassified other stationary poses as Plank. This is expected, as the plank is an isometric hold with no movement, an unusual pattern for a temporal model trained to recognize repetitive motion. Notably, some classes that achieve perfect test set metrics (e.g., Donkey Kick) perform poorly in production (Section 5.3.3). The training set Donkey Kick samples were recorded by different individuals than the author who performed the end-to-end benchmarks, highlighting a generalization gap across performers.

Figure 5.1 shows the confusion matrix for the production model. The dominant diagonal confirms strong performance across most classes, with the Lateral Lunge regression and Other class leakage as the primary off-diagonal concentrations. Appendix A.5 provides the full precision, recall, and F1 scores.

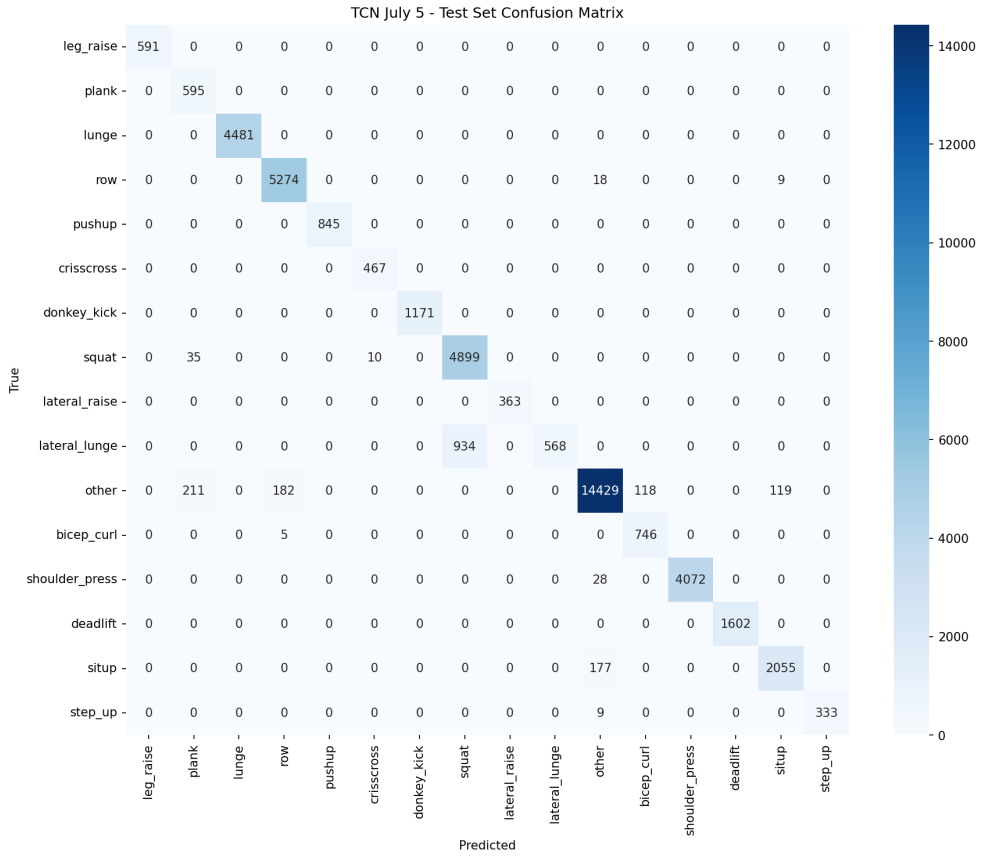


Figure 5.1: Confusion matrix for the July 5 TCN (production model) on the held-out test set. The Lateral Lunge class exhibits severe confusion with Squat, while seven classes achieve perfect classification. The Other class dominates the dataset (15,059 test samples), reflecting the reality that most input frames are non-exercise activity. No class weighting was applied during training.

5.2.4 Model Comparison

Table 5.3 summarizes the classification accuracy across all evaluated models. HistGradientBoosting achieves 95.26%, remarkably close to the TCN's 95.82%, a difference of only 0.56 percentage points. At the same 30-frame window size, the May 5 TCN (92.21%) underperforms HistGradientBoosting on the test set. However, offline accuracy alone does not capture the models' suitability for real-time deployment. The May 5 TCN was used in production for over a month precisely because its temporal architecture produced stable, non-flickering predictions that the downstream pipeline requires, despite its

Table 5.3: Classification accuracy across all evaluated models on the combined dataset.

Model	Window	Test Acc (%)	Temporal?
Decision Tree	30	78.09	No
LinearSVC	30	66.80	No
KNN (k=5)	30	91.00	No
HistGradientBoosting	30	95.26	No
TCN (May 5)	30	92.21	Yes
TCN (July 5)	60	**95.82**	Yes

lower test accuracy. As discussed in Section 5.2.2, the prediction flickering inherent to window-independent classifiers renders the pipeline non-functional regardless of aggregate accuracy. The TCN's convolutional architecture eliminates this problem by design, making it the clear choice for production deployment.

The July 5 TCN's accuracy improvement over May 5 (95.82% vs 92.21%) reflects both the longer 60-frame window and the lower learning rate, though a controlled ablation isolating each factor was not performed. Both changes were motivated by the qualitative user testing described in Section 5.5: after observing the May 5 model's behavior during live demonstrations, the 60-frame window was adopted to give the classifier more temporal context.

For context, the TCN's 95.82% on 16 exercise classes is broadly consistent with results reported in the pose-based exercise classification literature. Sensor-based approaches (accelerometers, IMUs) report accuracies above 99% on smaller class sets [72, 10], but camera-based systems operating on 2D joint angles typically report 90% to 97% depending on the number of classes and the diversity of the evaluation data. For instance, Riccio et al. [48] achieve 99% accuracy on 4 exercise classes using a BiLSTM on pose keypoints, while Khurana et al. [34] report 93.6% exercise recognition accuracy in an unconstrained multi-person gym setting. Direct numerical comparison across studies is difficult because each uses its own dataset, class set, and evaluation protocol [48]. The results presented here should therefore be interpreted in terms of the specific 16-class problem and the evaluation conditions described in Section 5.1.

5.3 Repetition Counting Results

The classification evaluation in Section 5.2 measures accuracy on pre-segmented windows: each test sample is a fixed-length sequence known to contain a single exercise. In deployment, the system must also detect when an exercise begins and ends, classify it correctly, and count the repetitions within the detected segment. This section evaluates all three capabilities simultaneously through an end-to-end benchmark on continuous gym video.

5.3.1 Evaluation Protocol

A continuous gym session is recorded with the system's target camera, producing a single long video containing multiple exercise types performed in sequence with rest periods between sets. The video is played through the live backend in file mode, which reads frames at the source video's native rate but triggers model inference at a fixed 10 FPS, and produces a console log recording every detected exercise event with its classified name, repetition count, periodicity confidence, period length, stride, and duration.

Ground truth is annotated independently by watching the video and recording every exercise set performed (exercise name and repetition count), including sets that the system may have missed entirely. Sets are numbered sequentially: detected sets receive the event number from the console log, while missed sets are marked explicitly. In five of the nine test videos, the ground truth annotator and the exercise performer are the same person, which introduces a potential annotation bias: the annotator may unconsciously round ambiguous counts in favor of the system's output. While the exercise types and repetition counts are unambiguous for the majority of sets, this self-annotation bias should be considered when interpreting the results.

An automated parsing script merges the machine predictions with the ground

truth annotations and computes three metrics. The script processes the backend's console log sequentially: each successful exercise event (one that produces a database entry) is assigned an incrementing event number. A ground truth set is considered *detected* if there exists a non-discarded system event at the corresponding sequential position, as determined by temporal ordering of the log events against the ground truth annotations. Ground truth sets with no corresponding log event are marked as missed.

1. **Detection rate and F1 score:** the detection rate measures the fraction of ground truth exercise sets that produced a successful database entry (recall), while precision measures what fraction of the system's outputs correspond to real exercise sets:

$$\text{Recall} = \frac{N_{\text{detected}}}{N_{\text{detected}} + N_{\text{missed}}}, \qquad \text{Precision} = \frac{N_{\text{detected}}}{N_{\text{detected}} + N_{\text{discarded}}} \tag{5.1}$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.2}$$

   Sets that the system missed entirely (no log event) are counted as missed. Events where RepNet ran but the periodicity confidence fell below the repetition threshold (Section 4.5.3) are counted as *discarded*: these represent false positives that the confidence mechanism successfully filtered. Including discarded events in the precision calculation ensures that the evaluation accounts for the system's tendency to occasionally initiate tracking on non-exercise motion.

2. **Classification accuracy on detected events:** the fraction of successfully detected events where the predicted exercise name matches the ground truth label:

$$\text{Classification accuracy} = \frac{N_{\text{correct}}}{N_{\text{detected}}} \tag{5.3}$$

3. **Repetition counting accuracy:** measured by Mean Absolute Error (MAE),

Off-By-One accuracy (OBO), and within-± 3 accuracy across all N detected events with numeric ground truth:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |p_i - g_i| \tag{5.4}$$

$$\text{OBO} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[|p_i - g_i| \leq 1], \qquad \text{Within } \pm 3 = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[|p_i - g_i| \leq 3] \tag{5.5}$$

where p_i is the predicted count and g_i is the ground truth count for event i. OBO, introduced by Dwibedi et al. [16], captures the fraction of predictions within one repetition of ground truth. The ± 3 threshold provides a more lenient measure of practical accuracy. Plank sets are excluded because they are isometric holds with no meaningful repetition count. RepNet returns a single "repetition" for any sustained pose, so comparing this output to a ground truth duration would conflate two different quantities. Plank is instead evaluated by duration in Section 5.3.4.

Across the full test suite, only 1 event was discarded (in *Workout1*, a Row classified at 32% RepNet confidence), yielding a precision of 98.3%. Because the confidence threshold is highly effective at filtering false positives, the F1 score in this evaluation is dominated by recall rather than precision. The number of discarded events per test is reported alongside detection rates in subsequent sections.

This protocol is deliberately conservative. The F1 score penalizes the system both for exercises it fails to detect (recall) and for false positive detections that the confidence mechanism must filter (precision), while classification and counting metrics are computed only over detected events. Reporting all three metrics together prevents a system that detects few exercises but counts them well from appearing more capable than it is.

5.3.2 RepNet Accuracy

We evaluated the full pipeline, including RepNet [16] for repetition counting, on a 50-minute continuous gym session (*Workout1*, 1920×1080, 12.5 FPS) recorded with the NVR camera described in Section 4.1.1. The video contains 27 exercise sets spanning 9 exercise types, performed by the author. Because all exercises were performed by a single individual, these results reflect the system's accuracy for one body type and movement style. Multi-performer evaluation is reported in Section 5.5 through qualitative demonstrations, but no other performer was evaluated quantitatively. Deadlift was evaluated separately (see Table 5.8). Ground truth was annotated by watching the video independently before examining the system's predictions.

The two confidence thresholds described in Section 4.5.3 were set to Exercise Identification Threshold = 60% and Exercise Counting Threshold = 40% for all evaluations in this chapter. These values were selected based on the author's qualitative experience during the user demonstrations described in Section 5.5, not through a formal ablation study. A systematic sweep of threshold values would strengthen these results but was not performed. Plank bypasses the Exercise Counting Threshold entirely because it is an isometric hold: RepNet returns a single period with low confidence for any sustained pose, so the system records Plank sets based on the classification confidence alone.

Detection and classification. Of the 27 ground truth exercise sets, 25 produced successful database entries, yielding a detection rate of 92.6%. The two undetected sets fall into two categories:

- **Single-limb exercises** (2 sets): one left-side bicep curl and one left-side donkey kick fell below the 60% classification confidence threshold. The training data predominantly features bilateral movements, producing lower confidence for single-limb exercises.

All 25 detected events were classified correctly, yielding 100% classification accuracy on detected events. One additional classifier output (Row) was correctly filtered by RepNet's periodicity confidence, which returned 32%, below the 40% acceptance threshold. This demonstrates the value of two-stage confidence filtering: the classification threshold catches low-confidence predictions, while the repetition threshold catches events where the detected motion does not exhibit a repeating pattern.

Repetition counting. Table 5.4 summarizes per-exercise repetition counting performance. Of the 27 total ground truth sets, 2 are Plank (evaluated separately by duration in Section 5.3.4). Of the remaining 25, 23 were detected, leaving 23 events with numeric ground truth.

Table 5.4: Per-exercise repetition counting results on the *Workout1* benchmark. Plank sets (2/2 detected) are excluded because plank is evaluated by duration (Section 5.3.4). Det./GT shows detected sets over total ground truth sets. MAE, OBO, and within-± 3 are computed over detected events only.

Exercise	Det./GT	GT Reps	MAE	OBO	± 3
Pushup	3 / 3	10	1.00	2 / 3	3 / 3
Situp	3 / 3	10	2.33	1 / 3	3 / 3
Lateral Raise	3 / 3	10	0.67	3 / 3	3 / 3
Donkey Kick	1 / 2	10	2.00	0 / 1	1 / 1
Bicep Curl	4 / 5	10*	2.75	2 / 4	3 / 4
Crisscross	3 / 3	13, 10, 10	2.00	2 / 3	2 / 3
Leg Raise	3 / 3	10	1.33	2 / 3	3 / 3
Shoulder Press	3 / 3	10	1.67	1 / 3	3 / 3
All (excl. Plank)	**23 / 25**		**1.74**	**13 / 23 (57%)**	**21 / 23 (91%)**

*Fourth Bicep Curl set had 4 GT reps (incomplete set).

The system achieves a mean absolute error of 1.74 repetitions, with an OBO accuracy of 57% and 91% of predictions falling within ± 3 of ground truth. Three predictions exactly match the ground truth (13%). The dominant error pattern is undercounting: 13 of the 20 non-exact predictions underestimate the true count. This systematic bias is primarily a pipeline integration issue rather than a RepNet accuracy limitation. The classifier startup delay (Section 4.4.2) is the primary cause: the classifier requires several frames of

sustained confidence before declaring an exercise, so the first 1 to 3 repetitions occur before RepNet begins recording. A secondary factor is that at higher stride values the system samples every s-th frame, reducing temporal resolution. This is the inherent drawback of a sequential pipeline: downstream accuracy can only be as good as upstream recall.

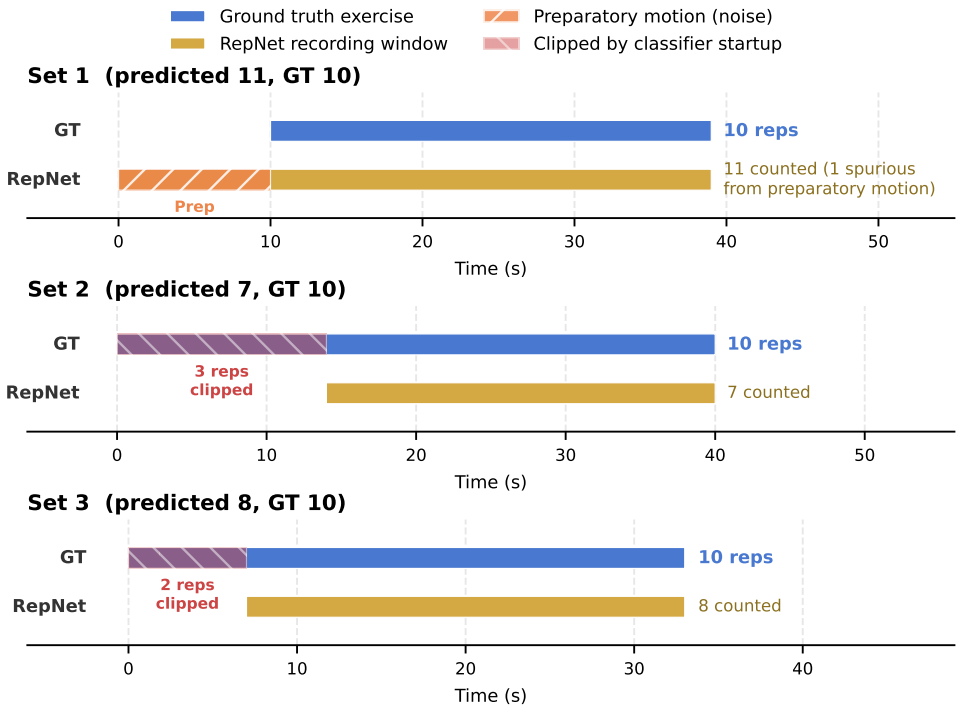


Figure 5.2: Temporal cropping of RepNet input for three Situp sets from the *Workout1* benchmark. Blue bars show the ground truth exercise window; gold bars show the recording window that RepNet received. Hatched regions indicate frames that were included erroneously (orange, Set 1) or excluded by the classifier startup delay (red, Sets 2 and 3).

Figure 5.2 illustrates this temporal cropping effect for the three Situp sets. The spatial crop (bounding box) is correct in all three cases, as shown in Figure 5.3. The errors are caused by when the recording window starts and stops relative to the actual exercise. Set 1 begins recording too early: several seconds of the user lying on the mat preparing to begin are included before the actual situps start. This preparatory motion introduces noise into RepNet's temporal self-similarity matrix, causing one spurious repetition to be counted

(11 predicted vs. 10 ground truth). Sets 2 and 3 exhibit the opposite problem: the classifier required several seconds to reach the confidence threshold, so by the time RepNet begins receiving frames, the first two to three repetitions have already occurred (7 and 8 predicted vs. 10 ground truth, respectively). Within the frames it received, RepNet counted the exact number of visible repetitions. The gap between predicted and ground truth counts is entirely attributable to repetitions that were temporally clipped before the classifier triggered recording.

Figure 5.3: Example of the bounding box crop fed to RepNet. The frame is cropped to the maximal bounding box across all accumulated frames and resized to 112×112 pixels.

The single detected Donkey Kick set (12 predicted vs. 10 ground truth) illustrates a unilateral exercise limitation at the classifier level: only the right-leg portion was detected, while the left-leg kicks fell below the classification confidence threshold. The training data predominantly features bilateral movements, producing lower confidence for single-limb exercises. Separately, Dwibedi et al. [16] describe a related ambiguity at the counting level, where RepNet counts individual leg cycles rather than bilateral pairs in asymmetric motions. This did not arise here because each unilateral set was performed entirely on one leg. Shoulder Press produced the largest single overcount (13 predicted vs. 10 ground truth) on a 25-second event that included warm-up arm movements before the main set, which RepNet interpreted as additional repetitions.

Exercises with simple, symmetric motion patterns perform best. Pushup

(MAE 1.00), Lateral Raise (MAE 0.67), and Leg Raise (MAE 1.33) all achieve at least 67% OBO accuracy. These exercises produce large, visually distinctive body displacements that create strong self-similarity signals in RepNet's temporal embedding space.

Comparison to standalone RepNet and subsequent methods. Dwibedi et al. [16] evaluate RepNet using normalized MAE ($\frac{1}{N}\sum |p_i - g_i|/g_i$) and OBO accuracy. Since 2020, two notable successors have been proposed: TransRAC [29], a transformer-based approach that encodes multi-scale temporal correlations (CVPR 2022), and ESCounts [54], an exemplar-based method that discovers visual correspondences across repetitions (ACCV 2024). Table 5.5 compares published results on the Countix benchmark.

Table 5.5: Repetition counting methods on the Countix benchmark. Normalized MAE is lower-is-better; OBO accuracy is higher-is-better. Our result is computed on 23 gym exercise events from the *Workout1* benchmark and is not directly comparable to the Countix evaluation (2,719 videos of diverse human activities), but is included to contextualize the error magnitude.

Method	**Norm. MAE ↓**	**OBO ↑**
RepNet [16]	0.364	0.70
TransRAC [29]	0.443*	0.29*
ESCounts [54]	0.276	0.70
Ours (end-to-end pipeline)	0.19††	0.57

*TransRAC results on Countix as reported by Sinha et al. [54]; the original paper primarily evaluates on the RepCount dataset.
††Computed using per-sample normalization ($\frac{1}{N}\sum |p_i - g_i|/g_i$), consistent with the formula used by Dwibedi et al. [16].

We include our result for reference, though it is not directly comparable: the Countix dataset contains 2,719 videos of diverse human activities (cooking, exercising, crafting), whereas our evaluation covers 23 gym exercise events from a single session. Our lower OBO accuracy (0.57 vs. 0.70 for both RepNet and ESCounts on Countix) reflects the difference between evaluating RepNet in isolation on pre-segmented video clips, where the repeating action fills the entire clip, and evaluating it within an end-to-end pipeline

where the classifier startup delay clips initial repetitions, bounding box crops are computed automatically rather than manually, and transitions between exercises introduce non-periodic frames that can confuse the self-similarity computation.

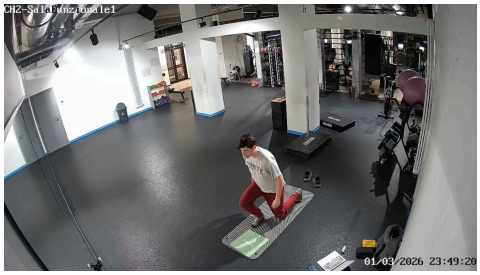
5.3.3 Dual-Camera Experiment

The *Workout1* benchmark was recorded from an elevated three-quarter camera, matching the dominant perspective in the training data. However, a system designed for gym use should also accommodate users who exercise at home with their own devices. These users are likely to place a phone or laptop camera facing them directly, producing a level frontal viewpoint. The training pipeline applies horizontal flip augmentation (Section 4.2.3), but this only simulates left-right mirroring, not a change in viewing angle. A natural question is whether these augmentations produce sufficient viewpoint invariance for the frontal perspectives that end users are likely to choose.

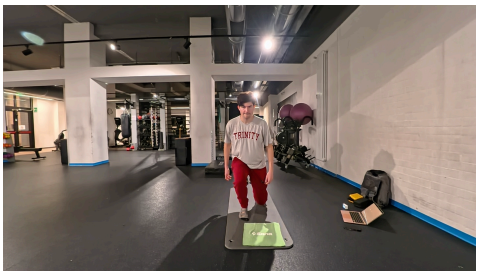
To test this, a second workout was recorded simultaneously by two cameras: a ceiling-mounted security camera (elevated three-quarter, identical to the *Workout1* setup) and an Insta360 Go Ultra (level frontal, lens-distortion corrected). Both cameras recorded at 1080p. The workout contained 20 ground truth exercise sets across 6 exercise types: 4 Lunge, 3 barbell Row, 4 Lateral Lunge, 4 Step Up, 4 Donkey Kick, and 1 Plank (60 seconds). For unilateral exercises (Lunge, Lateral Lunge, Step Up, Donkey Kick), each set was performed entirely on one leg, with sets alternating between left and right sides.

Because the two cameras differ in both placement and sensor characteristics (the Insta360 uses an ultra-wide-angle lens with software dewarping), the observed differences may reflect a combination of viewpoint and camera optics. A controlled comparison would ideally use two identical cameras at different angles, but the gym's security cameras are permanently mounted near

the ceiling and cannot be repositioned. Nevertheless, the exercise-specific pattern of failures described below is better explained by geometric projection effects than by camera-specific image quality artifacts. Table 5.6 compares the system's performance on each recording. Figure 5.4 shows a simultaneous frame from both cameras during a Lunge.



(a) Elevated three-quarter (security camera)



(b) Level frontal (Insta360 Go Ultra)

Figure 5.4: Simultaneous frames from both cameras during a Lunge. The elevated three-quarter view (a) preserves some sagittal plane motion, while the level frontal view (b) collapses depth information, making exercises like Row and Squat harder to distinguish. Note the difference in image quality: the security camera produces a grainy, low-contrast image with overlay text, while the Insta360 produces a sharp, well-lit image.

Table 5.6: Dual-camera comparison on the same workout recorded simultaneously. Both cameras record at 1080p. All metrics are defined in Section 5.3.1. Rep counting metrics exclude Plank (time-based).

Metric	**Elevated 3/4 (IP cam)**	**Level frontal (Insta360)**
Ground truth sets	20	20
Detected	14	8
Missed entirely	6	12
Detection rate	70.0%	40.0%
Classification accuracy	71.4% (10/14)	62.5% (5/8)
Rep count MAE	1.8	2.9
OBO	53.8% (7/13)	28.6% (2/7)
Within ± 3	92.3% (12/13)	71.4% (5/7)

Detection rate drops from 70% to 40% when the camera moves from the elevated three-quarter position to level frontal. The repetition counting MAE is substantially worse for the level frontal camera (2.9 vs. 1.8), and the OBO accuracy drops from 53.8% to 28.6%. The level frontal camera detected only 7

countable events, predominantly from exercise types with large body displacements (Lunge), while the elevated three-quarter camera detected 13 events spanning a wider range of difficulty. The most informative per-exercise comparison is shown in Table 5.7.

Table 5.7: Per-exercise dual-camera breakdown. Det. shows detected sets over ground truth. Class. shows correctly classified sets over detected sets. Exercises marked "n/a" were not detected.

Exercise	**Elev. Det.**	**Elev. Class.**	**Front. Det.**	**Front. Class.**
Lunge	4 / 4	4 / 4	4 / 4	4 / 4
Row	3 / 3	3 / 3	3 / 3	0 / 3
Lateral Lunge	4 / 4	0 / 4	0 / 4	n/a
Step Up	2 / 4	0 / 2	0 / 4	n/a
Donkey Kick	0 / 4	n/a	0 / 4	n/a
Plank	1 / 1	1 / 1	1 / 1	1 / 1

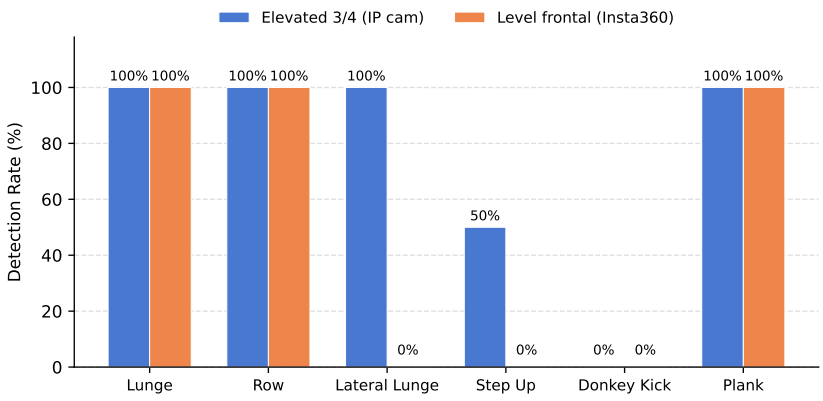


Figure 5.5: Per-exercise detection rate for elevated three-quarter and level frontal cameras. Lunge, Row, and Plank are detected regardless of viewpoint, while Lateral Lunge and Step Up are only detected from the elevated position. Donkey Kick is missed from both angles.

Figure 5.5 visualizes the per-exercise detection rates. Lunge is the only exercise that is fully invariant to camera angle: all 8 sets across both cameras were detected and correctly classified. Its distinctive alternating-leg stepping pattern produces a unique temporal signature in joint angle space regardless of viewpoint. Row, by contrast, is perfectly classified from the elevated three-quarter camera (3/3) but misclassified as Squat from the level frontal camera (0/3). From a frontal view, the bent-over row and the squat produce similar

2D joint angles because the depth dimension that distinguishes them is lost in monocular projection. The root cause and its relationship to the Lateral Lunge vs. Squat confusion are analyzed in Section 5.6.2.

Three exercise types performed poorly regardless of camera angle. Lateral Lunge was detected from the elevated camera (4/4) but misclassified in every case (as Squat, Lunge, or Row), consistent with the 37.8% per-class recall documented in Section 5.6.2. From the level frontal camera, Lateral Lunge was not detected at all. Step Up was partially detected from the elevated camera (2/4) but misclassified as Donkey Kick in both cases, and from the level frontal camera it was completely missed. Donkey Kick was missed from both cameras in this experiment (0/8), including sets performed both facing the camera and from the side. Combined with the 1/2 detection rate in *Workout1*, the system detected only 3 of 10 Donkey Kick sets across all recordings. This is surprising given the perfect test set metrics reported in Section 5.2.3: the test set Donkey Kick samples were recorded by different individuals (Marco and Matteo) than the author who performed the end-to-end benchmarks, suggesting that the apparent accuracy reflects performer-specific patterns rather than true generalization of the Donkey Kick class.

Test suite summary. Table 5.8 summarizes all nine test videos evaluated during this thesis. The five short single-exercise tests contribute Squat (4 sets), Situp (2 sets), and Plank (3 sets) coverage with 100% detection and classification accuracy. Combined with the longer benchmarks, Plank reaches 7 total sets and Situp reaches 5, providing reasonable coverage for these exercise types.

Across the full test suite, the system detects 59 of 81 ground truth exercise sets (72.8% recall, 98.3% precision, F1 = 83.7%), correctly classifies 52 of the 59 detected events (88.1%), and achieves a repetition counting MAE of 2.1 across 52 events with numeric ground truth. The combined OBO accuracy is 48% and the within-± 3 accuracy is 88%.

Table 5.8: Summary of all end-to-end test videos. Detection rate, classification accuracy, MAE, OBO, and within-± 3 are defined in Section 5.3.1. Rep counting metrics exclude Plank sets.

Video	Camera	Angle	GT	Det.	Class.	MAE	OBO	± 3
Workout1	IP cam	Elev. 3/4	27	92.6%	100%	1.7	57%	91%
Deadlift	IP cam	Elev. 3/4	5	60.0%	100%	3.0	0%	100%
Workout2	IP cam	Elev. 3/4	20	70.0%	71.4%	1.8	54%	92%
Workout2_360	Insta360	Level front.	20	40.0%	62.5%	2.9	29%	71%
Situp_test	IP cam	Elev. 3/4	2	100%	100%	1.5	50%	100%
Squat_test	IP cam	Elev. 3/4	2	100%	100%	2.0	50%	100%
Squat	IP cam	Elev. 3/4	2	100%	100%	4.0	50%	50%
Plank	IP cam	Elev. 3/4	2	100%	100%	n/a	n/a	n/a
Plank_1080	IP cam	Elev. 3/4	1	100%	100%	n/a	n/a	n/a
Combined			**81**	**72.8%**	**88.1%**	**2.1**	**48%**	**88%**

Table 5.9 aggregates per-exercise results across the entire test suite. The exercises divide into three tiers. Eight exercises achieve 100% detection and classification accuracy, with MAE ranging from 0.67 (Lateral Raise) to 3.00 (Squat, Deadlift). Bicep Curl (80% detection) and Row (50%) are partially successful: both are correctly classified when detected, but unilateral or viewpoint-dependent sets are missed. The bottom tier consists of exercises where the system fails systematically: Lateral Lunge is detected at 75% but misclassified in every case, Step Up is rarely detected (13%), and Donkey Kick is detected in only 30% of sets.

Evaluation scope. The combined test suite covers 15 of the 16 exercise types (all except Other) across 81 ground truth sets. Five exercises (Lateral Raise, Crisscross, Leg Raise, Bicep Curl, Shoulder Press) appear only in the *Workout1* recording and would benefit from additional test data. Squat, Situp,

Table 5.9: Per-exercise results aggregated across all nine test videos. Det./GT shows detected sets over ground truth. Class. shows correctly classified sets over detected. Plank is evaluated by duration (Section 5.3.4); counting metrics are not applicable.

Exercise	**GT**	**Det./GT**	**Class.**	**MAE**	**OBO**	± 3
Lunge	8	8 / 8	8 / 8	2.50	3 / 8	6 / 8
Plank	7	7 / 7	7 / 7	n/a	n/a	n/a
Situp	5	5 / 5	5 / 5	2.00	2 / 5	5 / 5
Squat	4	4 / 4	4 / 4	3.00	2 / 4	3 / 4
Pushup	3	3 / 3	3 / 3	1.00	2 / 3	3 / 3
Lateral Raise	3	3 / 3	3 / 3	0.67	3 / 3	3 / 3
Crisscross	3	3 / 3	3 / 3	2.00	2 / 3	2 / 3
Leg Raise	3	3 / 3	3 / 3	1.33	2 / 3	3 / 3
Shoulder Press	3	3 / 3	3 / 3	1.67	1 / 3	3 / 3
Bicep Curl	5	4 / 5	4 / 4	2.75	2 / 4	3 / 4
Deadlift	5	3 / 5	3 / 3	3.00	0 / 3	3 / 3
Row	6	3 / 6	3 / 3	1.33	2 / 3	3 / 3
Donkey Kick	10	3 / 10	3 / 3	2.33	0 / 3	3 / 3
Lateral Lunge	8	6 / 8	0 / 6	2.00	4 / 6	5 / 6
Step Up	8	1 / 8	0 / 1	2.00	0 / 1	1 / 1
All (excl. Plank)	**74**	**52 / 74**	**45 / 52**	**2.06**	**25 / 52**	**46 / 52**

Plank, Deadlift, and Lunge have coverage across multiple test videos. Lateral Lunge, Step Up, and Donkey Kick have extensive coverage (8 or more sets each) but consistently poor results, indicating genuine system limitations rather than insufficient evaluation. All ground truth was annotated by the author, and most test performances were by a single individual. Multi-performer evaluation would strengthen the generalization claims made in Section 5.5.

Although the Other class is excluded from ground truth evaluation (there are no "Other sets" to detect), it functioned as intended throughout all nine test videos: the system produced zero false positive exercise detections during non-exercise periods such as walking, resting, or adjusting equipment. This 100% precision for the rejection class validates the explicit Other-class modeling strategy described in Section 2.1.3.

5.3.4 Time Under Tension

Plank sets are evaluated by duration rather than repetition count, since planks are isometric holds with no meaningful repetition. The system reports Time Under Tension (TUT) as described in Section 4.4.3. For Plank, the system relies exclusively on the TCN classifier confidence to detect and time the exercise. RepNet is not thresholded for Plank because it returns a single period with low confidence (39% and 49% for the two *Workout1* Plank sets), correctly reflecting the absence of repeating motion.

Ground truth was measured with a stopwatch. Table 5.10 summarizes all seven Plank sets across five test videos.

Table 5.10: Time Under Tension results for Plank sets. Ground truth was measured with a stopwatch. Difference is predicted minus ground truth.

Test Video	Set	Predicted (s)	GT (s)	Diff (s)
Workout1	#1	34	30	+4
Workout1	#2	20	16	+4
Plank	#1	33	32	+1
Plank	#2	17	15	+2
Plank_1080	#1	67	62	+5
Workout2	#1	63	60	+3
Workout2_360	#1[†]	77	60	+17
Mean				**+5.1**

[†]The level frontal camera caused the classifier to initially predict Crisscross during the pre-Plank positioning phase, inflating the recorded duration. Excluding this outlier, the mean overestimate is +3.2 s.

Across the first six Plank sets (excluding the *Workout2_360* outlier), the system overestimates duration by a mean of 3.2 seconds (range: +1 to +5 seconds). The consistent overestimate reflects the classifier startup delay: the internal timer begins when the first confident Plank prediction enters the sliding window, which lags slightly behind the true exercise start. Since the same delay affects the end of the exercise symmetrically, the net effect is a small positive bias. These results demonstrate that the TCN classifier alone provides satisfactory duration measurement for isometric exercises without requiring RepNet, provided the camera angle supports reliable classification.

5.4 Real-Time Performance

5.4.1 Latency Analysis

To quantify real-time feasibility, we instrumented each pipeline stage with `time.time()` calls and processed the *Workout1* video (1920×1080, 12.5 FPS, approximately 50 minutes) in file mode, yielding 5,259 timed inference frames. File mode reads frames directly from disk, so the measurements exclude the network latency of receiving frames from a live camera. The backend still streams annotated frames to the frontend via HTTP multipart MJPEG during file mode, but the encoding and transmission run asynchronously in a separate task with a non-blocking output queue, so they do not affect the per-frame inference timings reported below. All measurements were taken on an Apple M1 MacBook with 16 GB of RAM. MediaPipe runs on the CPU via TensorFlow Lite's XNNPACK delegate, while the TCN and RepNet run on the Metal Performance Shaders (MPS) backend. RepNet's `AdaptiveMaxPool3d` layer falls back to CPU at runtime because MPS does not yet support this operator.

Table 5.11: Per-stage inference latency on Apple M1 (16 GB RAM). The first TCN measurement (103.4 ms cold start) was excluded. RepNet latency is aggregated across all test videos (3 invocations exceeding 10 s were excluded as outliers caused by unusually long input sequences).

Stage	Mean (ms)	Std (ms)	N	% of Total
Main thread (per frame):				
Frame crop + resize	0.24	0.15	5,259	0.3%
MediaPipe pose estimation	76.28	9.62	5,259	92.6%
Joint angle computation	5.90	0.96	5,259	7.2%
Total per frame	**82.42**	**9.85**	**5,259**	**100%**
Separate threads (overlapping with frame processing):				
TCN classification	9.63	37.23	5,199	n/a
RepNet (per invocation)	4,211	1,554	57	n/a

TCN and RepNet each run in dedicated threads that overlap with the main thread's frame processing. RepNet runs once per detected exercise set, not per frame. Across the 50-minute *Workout1* benchmark, RepNet accounted for 3.3% of total wall-clock runtime (Section 5.4.2).

Table 5.11 shows the per-stage breakdown. MediaPipe pose estimation

dominates the per-frame cost at 76.28 ms, accounting for 93% of the total. Frame cropping and joint angle computation are negligible (0.24 ms and 5.90 ms respectively). The TCN classifier adds 9.63 ms but runs in a separate thread, so it overlaps with the next frame's processing. The effective wall-clock latency per frame is therefore approximately 82 ms (crop + pose + angles), comfortably within the 100 ms budget imposed by the 10 FPS processing rate. During the 50-minute Workout1 benchmark, the system sustained the target 10 FPS without dropping frames under normal operation. Frame drops can occur when the system's memory is under pressure, for example when RepNet accumulates a long buffer from a 4K input, but this did not occur during any of the 1080p test sessions.

RepNet is invoked once per detected exercise set rather than per frame, with a mean latency of 4.21 seconds across 57 invocations from all test videos (3 outliers exceeding 10 seconds were excluded; these corresponded to exercise sets with over 50 seconds of accumulated frames). Its cost depends on the number of accumulated frames and the number of valid stride candidates tested. Because it runs in its own thread and the user continues to receive annotated video while RepNet processes, this latency does not affect the perceived frame rate.

5.4.2 Power Consumption

To provide an approximate characterization of the system's energy footprint, power draw was measured on the Apple M1 MacBook Air (16 GB RAM) while streaming a file saved from a live RTSP feed at 1920×1080 at 10 FPS from the gym's security camera. Measurements were taken using AlDente[3], a macOS battery monitoring utility that reports system-level power draw (not per-process), with the frontend, backend, and PostgreSQL database all running simultaneously. The values in Table 5.12 are approximate readings rounded to the nearest watt and should be interpreted as order-of-magnitude indicators

[3] https://apphousekitchen.com/

rather than precise per-component measurements.

Table 5.12: System power consumption on Apple M1 MacBook Air by pipeline state.

Pipeline State	**Power (W)**
Idle (applications open, no video)	6
Video streaming only	7
Person detected (+ BlazePose + TCN)	12
Full pipeline (+ RepNet)	28

The progression from idle to full pipeline reveals where power is consumed. These per-stage attributions are derived by subtraction across pipeline states and assume additive, non-interacting power consumption. Video capture and WebSocket transmission add 1 W over idle, MediaPipe pose estimation and TCN classification add a further 5 W, and RepNet contributes an additional 16 W when active. The steady-state power draw during a typical workout is therefore 12 W (person detected, no RepNet), a 2× increase over idle. The 28 W peak occurs only during RepNet invocations, which account for approximately 3.3% of total runtime (100.5 seconds of RepNet computation across 26 invocations during the 50-minute Workout1 benchmark). The system's reject-before-processing architecture contributes to this efficiency: each pipeline stage activates only when the preceding stage produces a valid result, so RepNet never runs unless a complete exercise set has been detected and classified. RepNet is also not optimized for the MPS backend and would likely draw less power on CUDA hardware, where `AdaptiveMaxPool3d` runs natively on the GPU.

5.4.3 Scalability

The system was containerized with Docker (Section 3.6.1) and deployed to Google Cloud Run (Section 3.6.2), but production-scale load testing was not performed for Phase 1. The current architecture processes one user per backend instance: each WebSocket connection maintains its own `FrameProcessor`

with dedicated MediaPipe, TCN, and RepNet threads. Serving multiple concurrent users would require either horizontal scaling (one Cloud Run instance per user) or architectural changes to share the pose estimation model across connections. Cloud Run's autoscaling would handle the former approach automatically, though the cold start latency for loading the MediaPipe and TCN models on each new instance has not been measured.

5.5 User Testing

The system was demonstrated to five external users across three formal sessions and several informal tests. These demonstrations served both as usability evaluations and as generalization tests, since one of the five users (Federico) was not represented in the training data. No quantitative data (per-exercise accuracy, repetition counts, or detection rates) was recorded during these demonstrations; the observations reported below are qualitative.

5.5.1 Martina (March 19, 2025)

The first live user test was conducted with Martina, a gym trainer who had also recorded exercise demonstration videos for the training dataset. This session revealed two issues: the repetition count was computed only after the exercise set ended rather than in real time, and a training data mismatch caused spurious predictions (Appendix A.3.4). Both were resolved before subsequent tests. Predictive counting was implemented after this session, providing live feedback during the set and correcting to the final RepNet count afterward.

5.5.2 Vincenzo (May 6, 2025)

The second demonstration was conducted for Vincenzo Bassi, the business partner and gym owner who co-proposed the project. Although Vincenzo had recorded exercise videos for the training dataset, his test was valuable as the

first demonstration to a non-technical stakeholder focused on commercial viability. The full pipeline was demonstrated live, including exercise classification and repetition counting. Classification and counting performed reliably across seven exercises tested. Vincenzo's feedback shifted the project's focus from accuracy refinement to deployment: he recommended designing for mobile access, which led to the decision to containerize the application with Docker and deploy to Google Cloud Platform. He also proposed future directions including multi-person tracking and identity matching, which are discussed further in Section 6.4.

5.5.3 Federico (July 4, 2025)

The final formal demonstration was conducted with Federico, who tested the cloud-deployed version using a webcam. This session was particularly valuable because Federico was the only tester who did not appear in the training data, serving as a stress test of the system's ability to generalize to a truly unseen user. Although webcam input had been tested previously by the author, Federico's session combined an unseen user with a different camera setup (webcam instead of security camera). Exercise classification performed reliably for the exercises tested (Shoulder Press, Lateral Raise, Squat, among others), and the overall system was assessed as functional for a minimum viable product. Several bugs were discovered during this session, including partial-body misclassification, squat hand placement sensitivity, and a selfie camera orientation issue (Appendix A.3.5). A webcam mirroring feature was added after this session so that users see a natural mirror view, and the 60-frame input window was adopted based on these real-world observations. The next day, the July 5 production model was trained, achieving 95.82% test accuracy.

5.5.4 Generalization

Of the five users who tested the system, four (Martina, Vincenzo, Marco, and Matteo) had recorded exercise videos that appeared in the training data. Only Federico was a truly unseen user. The system correctly classified exercises for all five testers, including Federico, whose successful results on a webcam he had never used before are consistent with the hypothesis that normalized joint angles generalize across individuals. However, with only one unseen tester, the evidence for user-independent generalization is limited. The primary failures encountered during testing (single-arm curls, partial body visibility) were exercise-specific limitations rather than user-specific ones, which further suggests that the joint angle representation is person-invariant by design. A formal multi-performer evaluation with a larger pool of unseen subjects remains a priority for Phase 2.

5.6 Error Analysis and Failure Cases

Development spanned seven months and involved continuous real-world testing in the gym environment. This section catalogs the failure modes encountered, organized by the pipeline stage in which they originate, and documents whether each was resolved, mitigated, or persists as a known limitation of the final system.

5.6.1 Pose Estimation Failures

The earliest and most consequential failure modes originate in MediaPipe's pose estimation stage, because all downstream classification and counting depend on accurate joint coordinates.

Multiple people in frame. When more than one person is visible, MediaPipe (configured with `num_poses=1`) may lock onto a bystander instead of

the exerciser. This occurs when a background person appears larger at the start of a video, or when an occlusion such as a gym pillar momentarily hides the exerciser, causing MediaPipe to switch targets without switching back. The training data was cleaned by manually cropping videos to isolate the exerciser, and in production the system assumes a single person occupies the field of view. If the camera covers a wide area, cropping to a smaller region of interest eliminates the problem. A multi-person tracking system would require explicit identity management, which falls outside the scope of Phase 1.

Low contrast (black clothes on black floor). When a subject wears dark clothing against a dark gym floor, MediaPipe's skeleton detection becomes intermittent, appearing and disappearing between frames. In practice, the contrast was rarely low enough to cause sustained failure. Moving from the standard to the heavy BlazePose model resolved the remaining edge cases, as the larger model is more robust to low-contrast conditions.

Far camera distance. The BlazePose model card recommends a maximum distance of approximately four meters. In the gym, degradation was caused not by absolute distance but by MediaPipe's internal downscaling: when the subject occupied too few pixels after resizing, joint localization became unreliable. By cropping the 4K camera feed onto the region containing the exerciser, successful pose estimation was achieved at distances of up to eight meters, effectively doubling the documented range.

Pushup joint errors. During pushups, MediaPipe occasionally returned incorrect joint positions for frames where the body was close to the ground and limbs were partially occluded. The visibility scores for the affected joints were correspondingly low, which the classifier received as input features. This failure was not explicitly corrected but was partially mitigated by the inclusion of visibility scores in the feature vector, allowing the TCN to learn to discount low-confidence joints.

5.6.2 Classifier Failures

Single-arm bicep curls. On April 15, left-handed bicep curls were classified as Other. The horizontal flip augmentation (Section 4.2.3) resolved the asymmetry between left-arm and right-arm curls, producing equal confidence for both sides. However, single-arm curls overall still produce lower confidence (approximately 90%) than two-arm curls (approximately 98%), because the model was predominantly trained on bilateral exercises. This was initially mitigated by lowering the classifier display threshold to 90%, and the production system now uses 60% (Section 5.3.1), which accommodates single-arm exercises comfortably.

Situp generalization. Situps exhibited highly variable postures across different practitioners: legs wide or narrow, hands reaching forward, behind the head, or under the hips. The TCN achieved 98.8% test accuracy on the April 17 dataset, yet live situps performed by the author were misclassified until May 5. The resolution was not a model architecture change but a dataset expansion: once enough situp variants were included in the training data, the TCN learned to generalize to the common signal (hip angle oscillation) rather than overfitting to a specific posture. This fix persisted through the July 5 model, and the same principle applies to all exercise classes: increasing variety in the training data is the most effective way to improve classifier robustness.

Lateral Lunge vs. Squat confusion. This confusion was first observed during live testing on April 15 and represents the most significant per-class failure in the final model. The April 17 confusion matrix showed no test-set confusion between these classes, but the July 5 model exhibits a severe regression: 934 out of 1,502 Lateral Lunge test samples (62.2%) are misclassified as Squat,

reducing per-class accuracy to 37.8%. The two exercises share similar lower-body kinematics (bent knees, lowered hips), making them difficult to distinguish from joint angles alone. Contributing factors include the camera angle (from certain views the poses are nearly identical in angle space) and the barbell resting on the shoulders during squats creating upper-body angles similar to the lateral lunge arm position. This remains the most critical failure in the final system and a clear target for future improvement, potentially through additional features such as foot placement width or torso lean angle.

Row vs. Squat confusion from level frontal camera. The dual-camera experiment (Section 5.3.3) revealed a camera-angle-dependent misclassification: all three barbell Row sets were correctly classified from the elevated three-quarter camera but misclassified as Squat from the level frontal camera. From the front, the bent-over row posture produces 2D joint angles similar to a squat because both exercises involve bent knees and a lowered center of mass; the depth dimension that distinguishes the hip hinge (row) from the knee bend (squat) is lost when projected onto a single image plane. This failure shares a root cause with the Lateral Lunge vs. Squat confusion described above: the 14-angle feature representation discards depth information, making exercises that differ primarily in the sagittal plane indistinguishable from certain viewpoints. This is a fundamental limitation of monocular 2D pose estimation rather than a deficiency specific to this system: any approach that reduces 3D body pose to 2D joint angles will lose the depth axis that separates these exercise classes. This limitation reinforces the importance of the elevated three-quarter camera position used in the gym deployment, which preserves partial sagittal plane information and minimizes depth ambiguity.

Squat sensitivity to hand placement. The Squat class is sensitive to arm position because the model uses all 14 joint angles, including wrist and elbow angles. Different squat variants (air squat, goblet squat, barbell squat) place

the hands in fundamentally different positions, leading to variable confidence scores across variants despite identical lower-body mechanics.

Wrist visibility triggering Other. When wrist visibility drops to zero (for example, when a hand moves behind the body), the classifier switches to Other because the visibility score is included as an input feature. Frames with zero visibility for key joints produce out-of-distribution inputs that the model defaults to the catch-all class.

Partial body visibility. On July 4, the system classified Bicep Curl when a person was only partially visible in the frame. The model has no mechanism to reject partially visible subjects. This was not investigated further because the system assumes the exerciser is fully within the camera's field of view during use.

5.6.3 Repetition Counting Failures

Classifier patience splitting sets. On April 22, a ten-repetition set was counted as "6 + 4" because the classifier briefly switched to a different exercise mid-set, causing the repetition counter to reset and start a new count. This highlights the coupling between classification stability and counting accuracy: any momentary misclassification interrupts the repetition accumulation.

5.6.4 System-Level Failures

Selfie camera orientation. On July 4, videos recorded with a front-facing camera were flipped upside down in the augmentation pipeline due to unhandled metadata rotation flags (Appendix A.3.5).

Vertical video rotation. On May 5, portrait videos caused rotated crops due to a landscape orientation assumption in the bounding box code (Appendix A.3.6).

Browser inactivity freeze. On July 4, the web interface was observed to freeze after five minutes of inactivity across multiple browsers and devices. The WebSocket connection is not maintained during periods without user interaction, requiring a page refresh to resume.

Mobile latency. When accessed from a mobile phone, the system exhibited noticeably higher latency than on desktop browsers, attributed to the device's lower processing power and the overhead of capturing and encoding camera frames on mobile hardware.

5.6.5 Statistical Limitations

The end-to-end evaluation comprises 81 ground truth exercise sets across 9 test videos, with per-exercise sample sizes ranging from 1 set (Plank in *Plank_1080*) to 10 sets (Donkey Kick across all recordings). Many of the per-exercise conclusions in this chapter rest on small samples: Pushup, Lateral Raise, Crisscross, Leg Raise, and Shoulder Press each appear in only one test video with 3 sets each. At these sample sizes, individual outliers can substantially shift aggregate metrics, and formal statistical significance testing is not meaningful. The per-exercise comparisons between cameras in Table 5.7 and the per-model accuracy differences in Table 5.3 should be interpreted as indicative trends rather than statistically validated conclusions.

Furthermore, the quantitative evaluations were conducted by only three individuals: the author performed the majority of tests, while Marco demonstrated *Situp_test* and *Squat*, and Martina the *Squat_test*. Both Marco and Martina appear in the training data, so these tests do not constitute an evaluation on unseen users. The user demonstrations in Section 5.5 provide qualitative evidence that the system works for one unseen user (Federico), but no multi-performer quantitative benchmark with unseen subjects was conducted. These limitations should be considered when interpreting the numerical results reported throughout this chapter.

5.7 Summary

Taken together, these results demonstrate that a monocular camera system using open-source pose estimation can classify 16 exercises at 95.82% accuracy, count repetitions within ± 3 of ground truth for 88% of detected sets, and process each frame in under 100 ms on consumer hardware. The system performs reliably under specific conditions: a single user fully visible in frame, an elevated three-quarter camera angle, and well-represented exercise types (those with distinctive, repetitive motion patterns). Outside these conditions, performance degrades substantially, as demonstrated by the 40% detection rate from a level frontal camera and the consistent misclassification of exercises that share similar 2D joint angle profiles. Viewpoint-dependent confusions and exercise variant sensitivity persist in the final version. These limitations are specific and addressable, and Chapter 6 reflects on the results and outlines the path forward.

Chapter 6

Conclusion

This thesis has presented the design, implementation, and evaluation of an integrated system for real-time exercise recognition and repetition counting from monocular gym video. The system operates on existing security cameras and consumer hardware, requiring no wearable sensors or dedicated infrastructure beyond a compute backend and the cameras already present in most gyms.

In this chapter, we summarize the primary contributions, reflect on practical insights from the development process, discuss the system's limitations, and outline directions for future work.

6.1 Summary of Contributions

This work produced the following contributions:

1. **An end-to-end exercise monitoring system** that processes a live camera feed through pose estimation, exercise classification, and repetition counting. In end-to-end evaluation on 81 ground truth exercise sets, the system detected 72.8% of exercises (F1 = 83.7%), correctly classified 88.1% of detected events, and counted repetitions with a mean absolute error of 2.1. The system functioned during live demonstrations to gym staff, external users, and the business partner (Section 5.5).

2. **A curated and publicly released exercise dataset** [14] of pose landmark sequences extracted from 330 gym videos, organized into a hierarchical taxonomy of 16 macro-categories and expanded through standard augmentation techniques. The dataset and its release conditions are described in Section 4.1.4.

3. **A comparative evaluation of classification architectures** (Chapter 5), including four scikit-learn baselines (Decision Tree, LinearSVC, k-Nearest Neighbors, HistGradientBoosting) and a custom Temporal Convolutional Network. The TCN achieves 95.82% test accuracy on 16 exercise classes; while HistGradientBoosting reaches a comparable 95.26%, the TCN produces temporally stable predictions that eliminate the frame-to-frame flickering that renders scikit-learn classifiers unusable in real time.

4. **Real-time integration** of three independently developed models (MediaPipe, TCN, RepNet) into a single inference pipeline that operates at interactive frame rates over a WebSocket connection, with an end-to-end latency of 82 ms per frame on an Apple M1 MacBook Air (Table 5.11).

5. **A dual-camera viewpoint analysis** (Section 5.3.3) demonstrating that camera placement is the primary factor limiting system performance: on the same workout recorded simultaneously by two cameras, detection rates dropped from 70% to 40% when the camera moved from an elevated three-quarter angle to a level frontal position. This finding quantifies a fundamental limitation of monocular 2D pose-based exercise recognition and has direct implications for deployment guidelines.

6. **A reproducible cloud deployment** (Section 3.6) using Docker containerization, Terraform infrastructure-as-code, and Google Cloud Platform, enabling operators to provision the system at a new location from

a single configuration file.

A commercial feasibility assessment for the boutique fitness segment was also developed in collaboration with a multidisciplinary team (Appendix B).

6.2 Practical Insights

Several observations from the development process may inform similar applied machine learning efforts:

1. **Data quality outweighed model complexity.** The largest performance gains came from expanding the breadth and diversity of the training dataset, not from deeper or wider model architectures. While the TCN outperformed the shallow baselines, variations in TCN configuration (depth, kernel size, dropout) produced marginal differences compared to the effect of adding more training videos.

2. **Offline accuracy is a poor proxy for real-time performance.** A classifier that achieves high accuracy on held-out test splits may still produce noisy, flickering predictions when applied frame-by-frame to a live video stream. Bridging this gap required engineering beyond the model itself, including input smoothing, confidence thresholds, and the patience-based accumulation mechanism.

3. **Patience-based trust improved perceived accuracy.** Rather than committing to a classification on every frame, the system accumulates high-confidence predictions over a sliding window before triggering the repetition counter (Section 4.4.2). This patience-based approach eliminated spurious classifications at the cost of a small annotation delay, a trade-off that proved effective in practice.

4. **Infrastructure consumed a substantial share of development effort.** Integrating trained models into a production web application required

engineering effort comparable to the machine learning work itself, spanning WebSocket communication, database logging, Docker packaging, and cloud deployment.

5. **Iterative testing with real users was essential.** Live demonstrations to the gym owner, trainers, and external observers revealed failure modes (e.g., left-handed users, camera angles, exercise transitions) that offline evaluation alone could not surface. Each round of user feedback led to targeted improvements in the pipeline.

6.3 Limitations

The system in its current form has several limitations that constrain its applicability:

1. **Single-person processing.** The pipeline processes one user at a time. While the software architecture supports concurrent instances, MediaPipe's pose landmarker cannot reliably track multiple people in the same frame (see Section 6.4.2).

2. **Exercise vocabulary.** The classifier covers only 16 macro-categories. Exercises not in the training set are mapped to an "Other" class, and visually similar exercises (e.g., Lateral Lunge and Squat, or Row and Squat from a frontal camera) remain a source of confusion due to overlapping 2D joint angle profiles.

3. **Exercise variant sensitivity.** The classifier is sensitive to within-category variants that alter the arm or stance configuration while preserving the primary movement. Single-arm Bicep Curls, for example, were never represented in the training data; during live testing, the classifier oscillated between "Bicep Curl," "Row," and "Other" without reaching the confidence threshold. Notably, the left arm produced more confident (though still sub-threshold) predictions than the right, suggesting

a laterality bias in the training data. Similarly, Goblet Squats produced lower classification confidence than Barbell Squats, sometimes falling below the threshold, because the model has not learned to discount arm position when classifying a lower-body exercise. These failures reflect insufficient variety in the training data rather than a fundamental architectural limitation (see Section 6.4.4).

4. **No person identification.** The system does not identify individual users, a deliberate scoping decision: exercise recognition and repetition counting were prioritized as the core technical contributions, while person identification was designated as a Phase 2 goal from the project's inception (Appendix B). As a result, exercise logs belong to a session rather than an individual, limiting the ability to build longitudinal training histories.

5. **Full-body visibility required.** Both the pose estimator and the repetition counter require an unoccluded view of the subject. Partial occlusion by equipment or other people degrades performance.

6. **Dataset scope.** We recorded all training data at a single gym with approximately six subjects. While the user demonstrations (Section 5.5) provide qualitative evidence that the system generalizes to unseen individuals of different body types, we have not conducted a systematic evaluation across different gym layouts or camera configurations beyond the dual-camera experiment.

7. **Evaluation scope.** The quantitative evaluations comprise 81 ground truth exercise sets across 9 test videos, conducted by three individuals: the author performed the majority of tests, while Marco demonstrated *Situp_test* and *Squat*, and Martina demonstrated *Squat_test*. Both Marco and Martina appear in the training data, so these tests do not constitute an evaluation on unseen users. Per-exercise sample sizes range from

1 to 10 sets, limiting the statistical reliability of per-class conclusions. A more rigorous evaluation with multiple unseen performers and larger sample sizes would strengthen the generalization claims.

8. **No form assessment.** The system identifies *which* exercise is performed but does not evaluate *how well* it is performed. Posture correction and safety feedback are beyond the scope of this thesis and are discussed as future work in Section 6.4.5.

9. **Threshold sensitivity.** Several pipeline parameters, including classification and repetition confidence thresholds, minimum exercise duration, and the exercise ratio threshold, were tuned through informal experimentation rather than a systematic sweep. The system's sensitivity to these values has not been formally characterized (see Section 6.4.6 for the specific values and a proposed analysis).

6.4 Future Work

Several extensions would bring the system closer to a commercially viable product.

6.4.1 Person Identification

User identification is the most impactful missing feature. By associating exercise data with a specific individual, whether through face recognition, Bluetooth wearables, or integration with the gym's access control system, the platform could build longitudinal training histories and automatically push session summaries to health applications. Face recognition would classify as processing of biometric data under Article 9 of the General Data Protection Regulation [17], requiring explicit consent and a lawful basis. The primary technical challenge is re-identification as users move between camera views, although

the single-zone deployment model adopted in the current system partially mitigates this challenge.

6.4.2 Multi-Person Tracking

Extensive experimentation with MediaPipe's `num_poses` parameter confirmed that its pose landmarker, based on a MobileNetV2 backbone [49], cannot reliably track more than one person simultaneously. Several alternative pose estimators offer native multi-person support: OpenPose [8] pioneered bottom-up multi-person estimation using part affinity fields, YOLO-Pose [39] integrates keypoint detection into the YOLO object detection framework, and ViTPose [69] achieves state-of-the-art results using vision transformers. However, licensing constraints (e.g., YOLO-Pose uses the AGPL-3.0 license) and model size must be weighed against commercial deployment requirements. The software architecture already supports parallel user instances; the bottleneck is the pose estimation model itself. Replacing it with a permissively licensed multi-person estimator is the primary prerequisite for scaling the system.

6.4.3 Browser-Based Video Streaming

The current system streams annotated video frames as a sequence of independent JPEG images. Migrating to WebRTC or a similar browser-native protocol would reduce bandwidth consumption and enable true video streaming with hardware-accelerated codec support.

6.4.4 Exercise Variant Robustness

The variant sensitivity described in Section 6.3 can be addressed through three complementary strategies. First, expanding the training data to include single-arm variants, different grip widths, and diverse stances (e.g., sumo versus conventional Deadlifts) would allow the model to learn which joint angle trajectories define a category and which are incidental to the equipment or stance used. For Squats specifically, the model needs sufficient variety to learn that leg angles are discriminative while arm angles vary freely across barbell, goblet, and bodyweight variants. Because manual data collection is time-consuming (the current 330-video dataset required months of coordinated gym sessions), scaling this approach may benefit from synthetic data generation. Second, self-supervised pretraining on unlabeled pose sequences could improve the model's robustness to exercise variants without requiring additional labeled data, reducing the labeling cost of supporting new exercise variants. Third, a symmetry-aware architecture that processes left-side and right-side joint angles independently could make the classifier inherently robust to unilateral exercises. By training on half-body feature vectors, a single-arm Bicep Curl would present the same angular pattern as a two-arm curl on the active side, eliminating the laterality confusion observed during testing.

6.4.5 Form Counseling

Form assessment would compare the user's joint angle trajectories against reference executions to detect deviations and generate corrective feedback. One possible approach would be to use a large language model to translate the structured output (exercise class, repetition count, detected deviations) into natural-language coaching cues, without requiring the LLM to process raw joint angles. Recent work has begun exploring LLM-based health and fitness coaching [31], though the safety and accuracy requirements of exercise prescription remain an open challenge.

6.4.6 Threshold Sensitivity Analysis

Several pipeline parameters were tuned through informal experimentation: the 60% classification confidence threshold, the 40% repetition confidence threshold, the 60% exercise ratio threshold, and the 90-frame minimum exercise duration. A systematic sweep of these values, measuring detection rate, classification accuracy, and repetition counting MAE across the full test suite, would characterize the system's sensitivity to each parameter and identify configurations that improve performance on the exercises where the system currently struggles.

6.4.7 Outlook

The future work directions outlined above vary in scope and feasibility. Multi-person tracking and exercise variant robustness are the most immediately actionable: the former requires replacing the pose estimation model, for which several candidates already exist (Section 6.4.2), and the latter requires expanding the training dataset and exploring self-supervised pretraining rather than architectural changes. Threshold sensitivity analysis is a relatively contained engineering effort that could improve performance without modifying any model. Person identification and form counseling are longer-term goals that introduce additional privacy, safety, and regulatory considerations. The commercial feasibility assessment developed in Appendix B identifies person identification and multi-person support as prerequisites for the go-to-market roadmap, reinforcing that these two directions carry both technical and commercial priority. Preliminary discussions with Technogym [57] have taken place, and a pitch meeting is scheduled at the time of writing.

This work demonstrated that an existing security camera, combined with open-source pose estimation and a lightweight temporal classifier, can classify 16 exercises at 95.82% offline accuracy, detect 72.8% of exercise sets in continuous gym video, and count repetitions within ± 3 of ground truth for

88% of detected sets, all while processing each frame in 82 ms on consumer hardware. The dual-camera experiment revealed that viewpoint is the dominant factor limiting real-world performance, with detection rates dropping from 70% to 40% when the camera moved from an elevated three-quarter angle to a level frontal position. Open challenges remain: multi-person tracking, exercise variant robustness, viewpoint invariance, and form assessment are all necessary steps toward a commercially viable product.

Nevertheless, the system provides a concrete, end-to-end proof of concept that automated exercise monitoring from monocular video is feasible, and the specific failure modes identified in this thesis offer clear targets for future improvement.

Bibliography

[1] Hasyim Abdillah. Workout/fitness video. https://www.kaggle.com/datasets/hasyimabdillah/workoutfitness-video, 2023. Accessed: 2025.

[2] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2D human pose estimation: new benchmark and state of the art analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3686–3693, 2014.

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271, 2018.

[4] Silviya Baytcheva, Nicolas Cridlig, Ernesto Crimaldi, Leonardo Fossà, and Marco Faccioli. Century club business plan, December 2024. Laboratory of Business Plan M (course code 91252, A.Y. 2024/2025), Alma Mater Studiorum Università di Bologna. Reproduced in Appendix B.

[5] Valentin Bazarevsky, Ivan Grishchenko, Karthik Ravi, Tyler Uber, and Matthias Grundmann. BlazePose: on-device real-time body pose tracking. In *CVPR Workshop on Computer Vision for Augmented and Virtual Reality*, 2020.

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[7] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees*. CRC Press, 1984.

[8] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7291–7299, 2017.

[9] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6299–6308, 2017.

[10] Shibo Chen, Junhui Qi, Shiyu Fan, Zeng Qiao, Jian Cheng Yeo, and Chwee Teck Lim. Deep learning in human activity recognition with wearable sensors: a review on advances. *Sensors*, 22(4):1476, 2022. DOI: 10.3390/s22041476.

[11] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[13] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[14] Nicolas Ivan Cridlig. Sana exercise pose landmark dataset. Version 1.0. Zenodo, 2026. DOI: 10.5281/zenodo.18903598. URL: https://doi.org/10.5281/zenodo.18903598.

[15] Docker, Inc. Docker: accelerated container application development. https://www.docker.com/, 2024. Accessed: 2025.

[16] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. Counting out time: class agnostic video repetition counting in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10387–10396, 2020.

[17] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council (general data protection regulation). Official Journal of the European Union, L 119, 1–88, 2016. Article 9: Processing of special categories of personal data.

[18] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, 2005.

[19] Roy Thomas Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[20] Fortune Business Insights. Health and fitness club market size, share & industry analysis. https://www.fortunebusinessinsights.com/health-and-fitness-club-market-108652, 2026. Report ID: FBI108652. Accessed: February 2026.

[21] Jerome H. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[22] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, 2003. DOI: 10.1145/776985.776986.

[23] Google. Google Cloud Platform. https://cloud.google.com/, 2024. Accessed: 2025.

[24] Google. MediaPipe pose landmarker. https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker, 2023. Accessed: 2025.

[25] Google. Model Card: BlazePose GHUM 3D. Technical report, 2021. URL: https://storage.googleapis.com/mediapipe-assets/Model%5C%20Card%5C%20BlazePose%5C%20GHUM%5C%203D.pdf. Accessed: 2026-02-18.

[26] HashiCorp. Terraform by HashiCorp. https://www.terraform.io/, 2024. Accessed: 2025.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[29] Huazhang Hu, Sixun Dong, Yiqun Zhao, Dongze Lian, Zhengxin Li, and Shenghua Gao. TransRAC: encoding multi-scale temporal correlation with transformers for repetitive action counting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19013–19022, 2022.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[31] Nicholas Jin et al. A personal health large language model for sleep and fitness coaching. *Nature Medicine*, 2025. DOI: 10.1038/s41591-025-03888-0.

[32] Sam Johnson and Mark Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2010.

[33] Kemtai. Kemtai: motion tracking exercise platform. https://kemtai.com/, 2024. Accessed: February 2026.

[34] Rushil Khurana, Karan Ahuja, Zac Yu, Jennifer Mankoff, Chris Harrison, and Mayank Goel. Gymcam: detecting, recognizing and tracking simultaneous exercises in unconstrained scenes. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, volume 2 of number 4, pages 1–17, 2018.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[36] Kinovea. Kinovea: video analysis software for sport. https://www.kinovea.org/, 2024. Accessed: 2025.

[37] Colin Lea, Michael D. Flynn, Rene Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 156–165, 2017.

[38] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[39] Debapriya Maji, Soyeb Nagori, Manu Mathew, and Deepak Poddar. YOLO-Pose: enhancing YOLO for multi person pose estimation using object keypoint similarity loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2637–2646, 2022.

[40] Meta Platforms, Inc. React: a JavaScript library for building user interfaces. https://react.dev/, 2024. Accessed: 2025.

[41] OpenCV Team. OpenCV: open source computer vision library. https://opencv.org/, 2024. Accessed: 2025.

[42] Paritosh Parmar, Amol Gharat, and Helge Rhodin. Domain knowledge-informed self-supervised representations for workout form assessment. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 105–123, 2022.

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[45] Peloton Interactive. Peloton: connected fitness. https://www.onepeloton.com/, 2024. Accessed: February 2026.

[46] Python Software Foundation. Collections — container datatypes. https://docs.python.org/3/library/collections.html, 2024. Python 3.12 Standard Library. Accessed: 2025.

[47] Sebastián Ramírez. FastAPI. https://fastapi.tiangolo.com/, 2021. Accessed: 2025.

[48] Riccardo Riccio, Luca De Martino, and Carlo Sansone. Real-time fitness exercise classification and counting from video frames. *arXiv preprint arXiv:2411.11548*, 2024.

[49] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[50] Henning Schulzrinne, Anup Rao, and Rob Lanphier. Real time streaming protocol (RTSP). RFC 2326, Internet Engineering Task Force, 1998. https://tools.ietf.org/html/rfc2326.

[51] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[52] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1297–1304, 2011.

[53] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

[54] Saptarshi Sinha, Alexandros Stergiou, and Dima Damen. Every shot counts: using exemplars for repetition counting in videos. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2024.

[55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[56] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for visual recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5693–5703, 2019.

[57] Technogym. Technogym: the wellness company. https://www.technogym.com/, 2024. Accessed: February 2026.

[58] Tempo. Tempo: AI-powered home gym. https://www.tempo.fit/, 2024. Accessed: February 2026.

[59] Tonal. Tonal: the world's most intelligent home gym. https://www.tonal.com/, 2024. Accessed: February 2026.

[60] Alexander Toshev and Christian Szegedy. DeepPose: human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1653–1660, 2014.

[61] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3D convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4489–4497, 2015.

[62] Muhannad Tuameh. Exercise recognition time series. https://www.kaggle.com/datasets/muhannadtuameh/exercise-recognition-time-series, 2023. Accessed: 2025.

[63] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: a generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[65] VAY Sports. VAY Sports: real-time AI fitness coaching. https://vay-sports.com/, 2024. Accessed: February 2026.

[66] Viso.ai. Viso Suite: end-to-end computer vision platform. https://viso.ai/, 2024. Accessed: February 2026.

[67] Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. DOI: 10.18637/jss.v059.i10.

[68] XTRA Vision AI. XTRA Vision AI: 3D motion tracking APIs. https://web.archive.org/web/2025/https://www.xtravision.ai/, 2024. Original site defunct as of 2026; archived via Wayback Machine.

[69] Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. ViTPose: simple vision transformer baselines for human pose estimation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[70] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32 of number 1, 2018.

[71] Yi Yang and Deva Ramanan. Articulated human detection with flexible mixtures of parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2878–2890, 2013.

[72] Mi Zhang and Alexander A. Sawchuk. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys & Tutorials*, 14(4):1067–1088, 2012.

Appendix A

Technical Details

A.1 TCN Source Code

Listing A.1 reproduces the complete TCN model definition. The `TCNBlock` class implements a single residual block with two dilated 1D convolutions, batch normalization, ReLU activation, and dropout. A 1×1 convolution projects the residual connection when input and output channel dimensions differ (following the projection shortcut of He et al. [27]). The `TCN` class stacks three blocks with exponentially increasing dilation $(1, 2, 4)$, followed by global average pooling and a fully connected layer that maps to 16 class logits. The architecture is described in Section 4.3.3 and Table 4.3.

Listing A.1: TCN model definition.

```
class TCNBlock(nn.Module):
    def __init__(self, in_channels, out_channels,
                 kernel_size=3, dilation=1, dropout=0.2):
        super().__init__()
        self.conv1 = nn.Conv1d(in_channels, out_channels,
            kernel_size, padding='same', dilation=dilation
                )
        self.bn1 = nn.BatchNorm1d(out_channels)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout)
```

```
10          self.conv2 = nn.Conv1d(out_channels, out_channels,
11              kernel_size, padding='same', dilation=dilation
                    )
12          self.bn2 = nn.BatchNorm1d(out_channels)
13          self.residual = nn.Conv1d(in_channels,
                out_channels, 1) \
14              if in_channels != out_channels else nn.
                    Identity()
15
16      def forward(self, x):
17          out = self.conv1(x)
18          out = self.bn1(out)
19          out = self.relu(out)
20          out = self.dropout(out)
21          out = self.conv2(out)
22          out = self.bn2(out)
23          out = self.relu(out)
24          return out + self.residual(x)
25
26  class TCN(nn.Module):
27      def __init__(self, input_size, num_classes,
28                   num_channels=[64, 128, 256],
29                   kernel_size=3, dropout=0.2):
30          super().__init__()
31          layers = []
32          for i in range(len(num_channels)):
33              in_ch = input_size if i == 0 else num_channels
                        [i-1]
34              layers.append(TCNBlock(in_ch, num_channels[i],
35                  kernel_size, dilation=2**i, dropout=
                            dropout))
36          self.network = nn.Sequential(*layers)
37          self.global_pool = nn.AdaptiveAvgPool1d(1)
38          self.fc = nn.Linear(num_channels[-1], num_classes)
39
40      def forward(self, x):
```

```
        out = self.network(x)
        out = self.global_pool(out).squeeze(-1)
        return self.fc(out)
```

A.2 Hyperparameter Configurations

Table 4.4 in Section 4.3.4 lists the TCN training configuration used for the final July 5 model. Table A.1 below documents how the training setup evolved across the three production training rounds. The April 17 model trained on the Sana original and augmented subsets only (no Kaggle data), while the May 5 and July 5 models trained on the full combined dataset (Sana, augmented, and Kaggle subsets). The shift from 30 to 60 training frames between the May 5 and July 5 checkpoints improved the model's ability to capture full repetition cycles. The July 5 run additionally reduced the learning rate from 10^{-3} to 10^{-4} and introduced early stopping with patience 3.

Table A.1: Training configuration evolution across the three model checkpoints.

Parameter	**Apr 17**	**May 5**	**Jul 5**
Dataset	Sana + augmented	Combined	Combined
Training frames	30	30	60
Classes	16	16	16
Max epochs	20	20	30
Epochs reached	20	10	11
Optimizer	Adam	Adam	Adam
Learning rate	10^{-3}	10^{-3}	10^{-4}
Batch size	32	32	32
Early stopping	No	No	Yes (patience 3)
Test accuracy	97.07%*	92.21%	95.82%

*Measured on the original 60/20/20 split at training time, not re-evaluated on the July 5 split (see Table 5.2).

A.3 Notable Bugs and Fixes

Over seven months of development, numerous bugs were discovered through live testing, user demonstrations, and systematic evaluation. This section catalogs the most instructive examples, organized chronologically. Each entry describes the symptom, root cause, and resolution.

A.3.1 Half Feature Extraction

Discovered: March 18, 2025. An early version of the training data processor extracted only 7 left-side joint angles per frame, producing a feature vector of $7T$ dimensions (210 for $T{=}30$ frames), instead of the intended 14 bilateral angles ($14T{=}420$). The root cause was a broken configuration check that tested `None` instead of `len() == 0`, causing the processor to silently fall back to default features without issuing a warning. Meanwhile, the live inference pipeline correctly used all 14 angles, creating a distribution mismatch between training and inference. After correcting the pipeline, the feature vector was correctly sized at $28T$ dimensions (14 angles + 14 visibility scores) and all baseline accuracies were re-evaluated.

A.3.2 RepNet Bounding Box: Mean vs. Union

Discovered: April 6, 2025. RepNet requires cropped video frames centered on the exercising person. The initial implementation computed the bounding box extremes (min/max of keypoint coordinates) independently per frame, without tracking them across the sequence. Because a person's width is approximately constant during a set, using per-frame extremes caused the crop region to jump when the subject shifted laterally. An earlier iteration also used mean keypoint coordinates rather than extremes, compounding the instability. Both issues confused RepNet's temporal self-similarity computation. The fix was to maintain a cumulative bounding box union across all frames in the sequence, ensuring a stable crop region that only grows over time.

A.3.3 Queue.get() Blocking in TUT Computation

Discovered: April 2025. Python's `Queue.get()` method blocks indefinitely when the queue is empty. In the classification thread, this caused the thread to hang after the user stopped exercising, because no new frames were being enqueued. The Time Under Tension metric reported approximately half the actual elapsed duration as a result. Replacing the blocking call with `Queue.get(timeout=1.0)` allowed the thread to periodically check for termination signals, resolving the issue.

A.3.4 Training Data Placeholder Rows

Discovered: March 19, 2025 (Martina user test). The training CSV files contained placeholder rows for frames where MediaPipe failed to detect a pose. During live inference, the pipeline simply skipped such frames rather than inserting placeholders, creating a distribution mismatch between training and inference. This affected classification accuracy during the first user test. The fix was to remove all placeholder rows from the training data so that both training and inference operated on detected-frames-only sequences.

A.3.5 Selfie Camera Orientation

Discovered: July 4, 2025 (Federico user test). Videos recorded with a front-facing (selfie) camera were flipped upside down when processed through the data augmentation pipeline, because the augmentation code did not handle video metadata rotation flags embedded in the file container. The fix was to re-export affected videos with corrected orientation using `ffmpeg`. A related but distinct camera issue was webcam mirroring: the frontend was updated to horizontally flip the webcam preview so users see themselves as in a mirror, while the backend receives the original unflipped frames for correct pose estimation.

A.3.6 Vertical Video Rotation

Discovered: May 5, 2025. Vertical (portrait) videos caused the person to appear rotated after the bounding box crop, because downstream components in the pipeline assume landscape orientation (frame width greater than height). The fix enforces a minimum frame width of 1,080 pixels: when the cropped frame height exceeds its width, black bars are added equally to the left and right sides using `cv2.copyMakeBorder`. In practice, the black padding did not degrade pose estimation accuracy, as MediaPipe correctly ignores the featureless regions.

A.3.7 Frame-Based TUT Timing

Discovered: March 2, 2026. The original Time Under Tension implementation computed duration as `len(exercise_window) / fps`, counting the number of accumulated frames and dividing by the processing frame rate (10 FPS). This produced inaccurate durations for two reasons: frames dropped during processing reduced the count below the true elapsed time, and the frame buffer has a fixed maximum length that caps the measurable duration. The fix replaced frame counting with wall-clock timestamps: `exercise_start_time = time.time()` is recorded when the first frame of an exercise enters the window, and `time.time() - exercise_start_time` is computed when the exercise ends. Two follow-up fixes were required. First, the start time was not cleared when the patience mechanism rejected a spurious classification sequence, causing the timer to accumulate across unrelated exercise sets. Second, the start time was not reset when a crop region change interrupted classification. Both were resolved by explicitly setting `exercise_start_time = None` at every code path that clears the exercise window.

A.3.8 Global Variables Crashing WebSocket

Discovered: April 8, 2025. Early versions of the FastAPI backend stored inference state in global variables rather than per-connection instances. When multiple WebSocket connections were opened simultaneously, the shared state caused race conditions that crashed the server. The fix was to instantiate a dedicated `FrameProcessor` object per WebSocket connection, ensuring complete isolation between concurrent sessions.

A.3.9 CUDA Passthrough in Docker

Discovered: July 1, 2025. The Docker container for the backend initially could not access the host GPU. Running CUDA inside Docker requires the NVIDIA Container Toolkit (`nvidia-container-toolkit`) on the host and the `deploy.resources.reservations.devices` directive in the Docker Compose configuration. On the Ubuntu workstation, confirming that `nvidia-smi` returned valid output inside the container verified that GPU passthrough was functional.

A.4 Sklearn Baseline Confusion Matrices

Figures A.1–A.4 show the confusion matrices for the four scikit-learn baselines evaluated on the combined dataset (30-frame windows, 16 classes). All models were trained and evaluated using the same video-level stratified split described in Section 5.1.

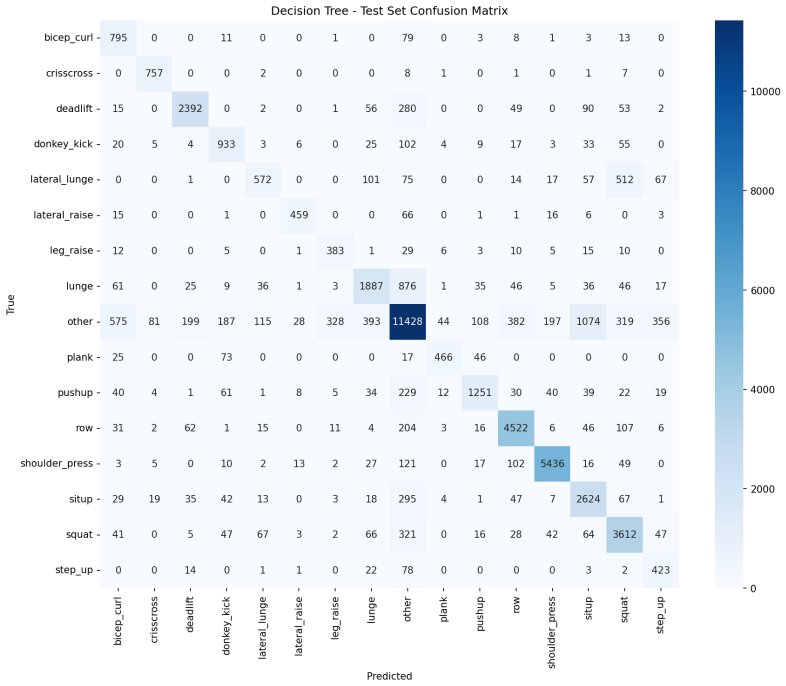


Figure A.1: Decision Tree confusion matrix on the combined test set (78.09% accuracy). The model achieves perfect training accuracy but generalizes poorly, with widespread off-diagonal errors across most class pairs.

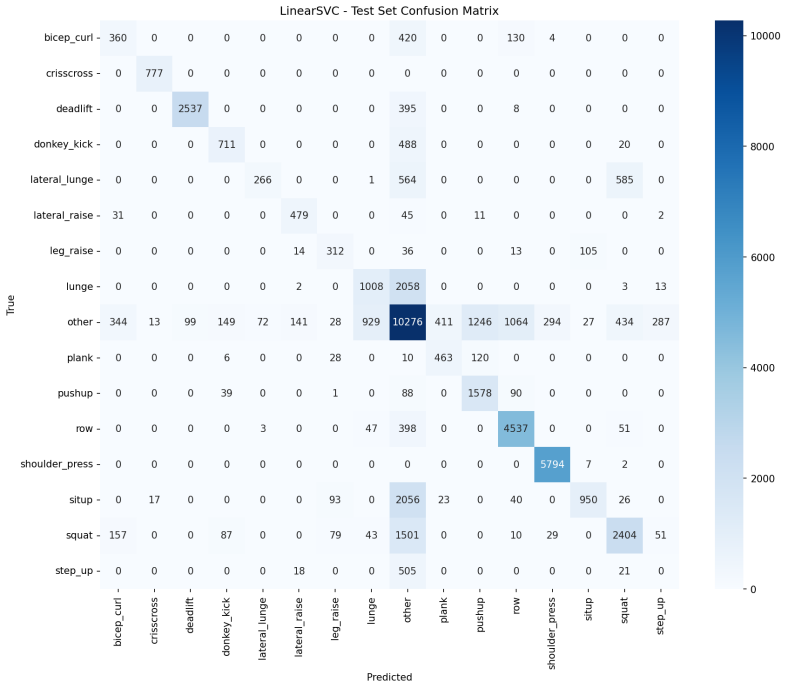


Figure A.2: LinearSVC confusion matrix on the combined test set (66.80% accuracy). The linear decision boundary fails to separate visually similar exercises, with Step Up receiving zero correct predictions and Lateral Lunge achieving only 18.8% recall.

Figure A.3: KNN (k=5) confusion matrix on the combined test set (91.00% accuracy). Despite high aggregate accuracy, Lateral Lunge recall drops to 46.1%, foreshadowing the same confusion pattern observed in the TCN.

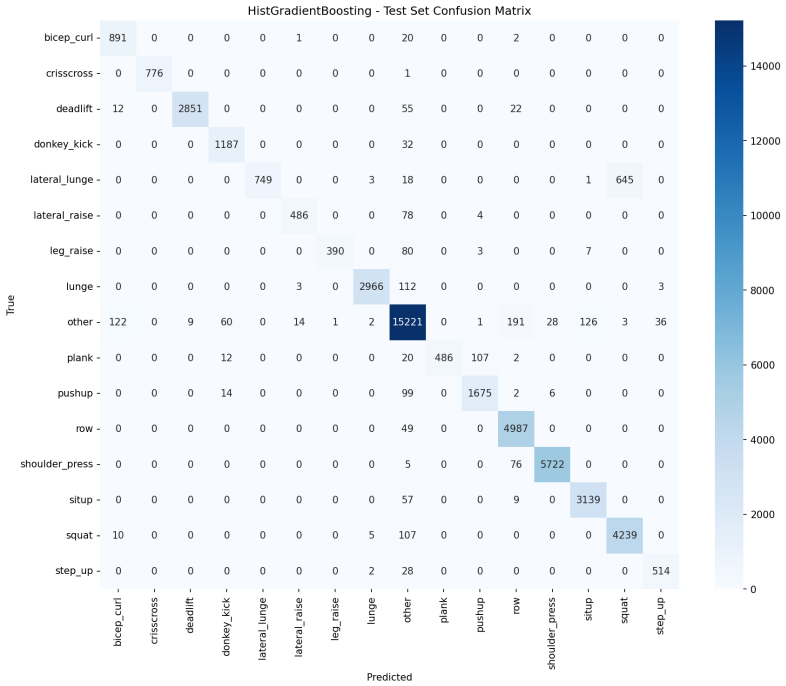


Figure A.4: HistGradientBoosting confusion matrix on the combined test set (95.26% accuracy). The strongest sklearn baseline, with most classes well separated. Lateral Lunge (52.9% recall) and Leg Raise (81.3% recall) remain the weakest classes.

A.5 TCN Per-Class Metrics

Table A.2 reports the per-class precision, recall, and F1 score for the July 5 production TCN on the held-out test set (95.82% overall accuracy). These metrics complement the confusion matrix in Figure 5.1.

Table A.2: Per-class precision, recall, and F1 score for the July 5 TCN on the held-out test set (95.82% overall accuracy). Classes are sorted by F1 score descending.

Class	Precision	Recall	F1	Support
Leg Raise	1.000	1.000	1.000	591
Lunge	1.000	1.000	1.000	4,481
Pushup	1.000	1.000	1.000	845
Donkey Kick	1.000	1.000	1.000	1,171
Lateral Raise	1.000	1.000	1.000	363
Deadlift	1.000	1.000	1.000	1,602
Shoulder Press	1.000	0.993	0.997	4,100
Crisscross	0.979	1.000	0.989	467
Step Up	1.000	0.974	0.987	342
Row	0.966	0.995	0.980	5,301
Other	0.984	0.958	0.971	15,059
Situp	0.941	0.921	0.931	2,232
Bicep Curl	0.863	0.993	0.924	751
Squat	0.840	0.991	0.909	4,944
Plank	0.708	1.000	0.829	595
Lateral Lunge	1.000	0.378	0.549	1,502
Weighted Avg	**0.963**	**0.958**	**0.955**	**44,346**

Appendix B

Business Plan

The following pages reproduce the Century Club Business Plan [4], developed in December 2024 by a five-person team for the Laboratory of Business Plan M (course code 91252, A.Y. 2024/2025) at the University of Bologna. The author led the project as Chief Executive Officer, coordinating task assignments across team members and contributing the technology description, computer vision service offering, and deployment architecture sections.

The plan contains market analysis (pp. 7–9), competitor benchmarking (p. 10), a SWOT assessment (p. 12), a business model canvas (p. 14), a go-to-market roadmap (p. 15), and a five-year financial forecast (pp. 16–17) for the AI-powered gym monitoring concept described in this thesis. The go-to-market roadmap targets a partnership with Technogym [57], whose connected machines are installed in thousands of facilities worldwide, providing a distribution channel that a pure software API cannot replicate. Preliminary discussions have taken place, and a pitch meeting is scheduled at the time of writing. The document retains its original format and promotional tone as submitted for the entrepreneurship course.

B.1 Century Club Business Plan Document

DECEMBER 2024

Century Club Business Plan

Our Future is Moving

Presented To: **Riccardo Fini and Natalia Panchieri**

Presented By: **Baytcheva, Cridlig, Crimaldi, Fossà and Faccioli**

Alma Mater Studiorum Università di Bologna

Table of Contents

Executive Summary

Getting fit sells.

Mission

In the middle of the largest AI boom of our generation, we enable each and every person with personalized training.

Market

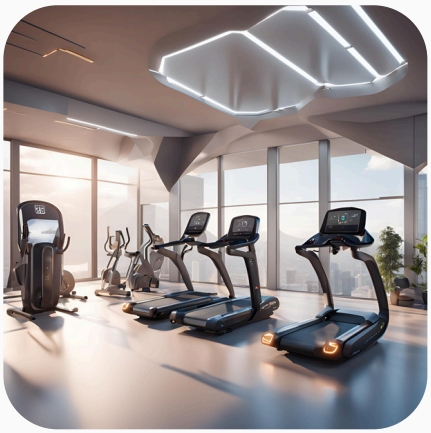
High end health and fitness centers and their trainers.

Duolingo revolutionized the language learning market. We take inspiration from their business model to thrive off subscriptions.
Reaching fitness goals takes years, and people have a limited amount of time each day to dedicate to them.
The health and fitness market is growing rapidly and expanding from North America. As the population ages, first world sedentary lifestyles contribute evermore to early deaths. The gym is a great place to stay in shape, but also intimidating. We offer health and fitness centers advanced tools to help their customers and provide comfort.

The Service	Century Club brings artificial intelligence to the gym workout.
The Leadership	Students with a combined 11 years of expertise in AI and 9 years experience at the gym.
The Industry	The global health and fitness club market size was valued at USD 105 billion in 2022 and is expected to double by 2030. We target the high end segment of this expansive market, in North America and Europe.
The Competitors	From the likes of Technogym and Xtra. Century Club differentiates itself by offering a computer vision neural network to owners which simultaneously localizes users, identifies their exercise, and provides them recommendations in real time, for all the people.
The Finances	Century Club wants to raise $1 million to implement its service. So far, the company has raised $5000 of its target from personal contributions. Financial resources are focused on developing the neural network and deploying to the first client: Sana Health and Fitness.
Future Plans	By 2026, the company would partner with Technogym. They are industry leaders and would help scale the service rapidly.

Introduction

Our Service



Timing

Lately, we've seen a lot of applications and programs implementing their own AI tools. Microsoft Visual Studio Code added Autopilot, an AI-powered tool that assists you while you're coding. Adobe Photoshop introduced some other tools like Generative Expand, that utilizes AI to expand your pictures with AI generated content, and Neural Filters, that can adjust your subject's expression, transfer the color palette of an image to another and much more.
Now, it's clear that the business world understands the potentialities of AI and many companies are interested in investing in that field. The hype for this brand of technology is at an all-time high.

Upscale (Boutique) Experience

At the same time, the gym industry is a very popular market, with a huge sales volume. The global health and fitness club market size was valued at USD 104.5 billion in 2022 and is expected to grow to USD 202.78 billion by 2030. In particular, North America holds the biggest part of the market with a market share of 42.59% in 2022 (Fortune Business Insights, October 2024).
In particular, we're targeting a portion of this market, Boutique Gyms. These gyms are focused on delivering an exclusive fitness experience through cutting edge facilities, personalized services and holistic wellness. There is a steady growth in the market as wealthy customers are not only interested in physical wellness but also mental wellness and in general improving the quality of their workouts, since they spend hours a week at the gym.

Digitalization

The ramping digitalization has induced a lot of gym owners to rely on Mobile apps for training scheduling, management and assisting the customers, and surely the COVID-19 pandemic catalyzed this process. Through these apps gym owners can offer the customers a much more complete service: people can request training plans via app, can schedule their training, book an appointment with a personal trainer and so on. At the same time it's easier for the owner to manage the gym and to check in real time the state of the gym.

Introduction

Our Technology

The Power of Computer Vision

Computer Vision is a branch of Artificial Intelligence that allows machines to “see”: starting from an image, a computer will then be able to acquire information from it. It can tell what’s in the image, detect movement and where. From this data computations can be performed and more statistics can be calculated.
Our service is based on this concept. We’re making devices able to check on the customers in order to prevent injuries and help them with their workout.

Dockerized

Imagine you carry a magic lunchbox. When you open it, it always has everything you need for your lunch—your sandwich, your drink, even your napkin—no matter where you are! You don’t have to go looking for anything; it's all packed inside, ready to use.
A Docker container is like the lunchbox, but for computer programs. It holds everything a program needs to work right, so you can take it anywhere and it will always be ready to go. We build the application in this manner to provide the service to any fitness center!

Our goal

The goal of CENTURY CLUB is to leverage these these aspects and create an innovative experience for gym owners and their customers. To make gyms a place where to feel taken care of and where it’s possible to work out stress-free.

The Team

Introducing ourselves

Nicolas Cridlig

Chief Executive Officer

Born in Paris and graduated from UC San Diego in Electrical Engineering, he is attending the MSc of AI in Bologna, Italy. Intuition and passion for our globalized society make him suited to run the company.

Alessia Crimaldi

Chief Marketing Officer

Graduated in Computer Science for Management, she is pursuing the MSc of AI in Bologna, Italy. Her analytical skills and forward-thinking approach make her the ideal leader to drive our marketing vision forward.

Federico Faccioli

Chief Operations Officer

Graduated in 2023 as Computer Engineer at the University of Bologna, is now attending a Master in Artificial Intelligence. His role is to manage company's daily operations and ensure efficiency.

Christina Baytcheva

Chief Financial Officer

Graduated with a double major in Computer science and Business Administration from AUBG, she is now pursuing a MSc in Artificial Intelligence in Bologna, Italy. Has a keen eye for detail .

Andrea Fossà

Chief Technical Officer

With a background in bioinformatics, he is poised to make technology which respects the body and natural rhythms. Currently attending the Master in Artificial Intelligence.

Global Fitness Market

Global Health and Fitness Market

The global health and fitness club market size was valued at USD 104.05 billion in 2022 and is projected to grow from USD 112.17 billion in 2023 to USD 202.78 billion by 2030, exhibiting a CAGR of 8.83% during the forecast period. North America dominated the health and fitness club market with a market share of 42.59% in 2022.

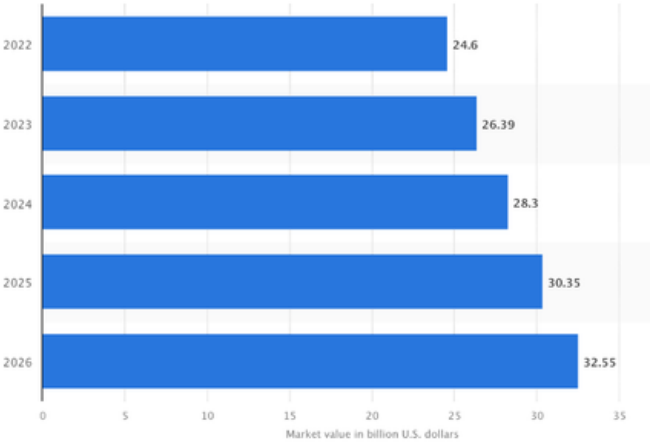
Forecast European Health & Fitness Market

The health and fitness market in Europe was forecast to grow in size between 2022 and 2026. Following the impact of the coronavirus (COVID-19) pandemic, the industry was projected to show signs of recovery and reach 32.55 billion U.S. dollars by 2026.

+7.5% increase in memberships form 2022 to 2023

+1.4% increase in the number of fitness clubs

Overall penetration of fitness club membership as % of total population in 2023 was 8.4% (7.9% in 2022)



Revenue Forecast Europe (2022-2026)
Source: Statista

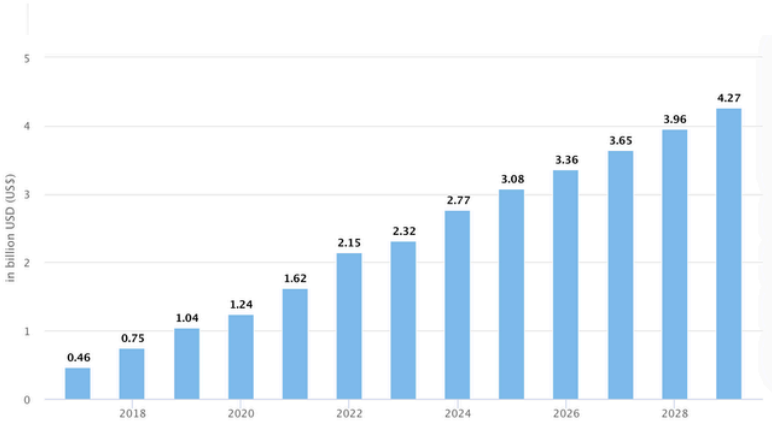
Forecast North American Health & Fitness Market

Total revenue is expected to show an annual growth rate (CAGR 2022-2029) of 9.04%, resulting in a projected market volume of US$4.27 billion by 2029.
The fitness landscape is dynamic and continually evolving, with emerging trends shaping how individuals approach their health and wellness.

The primary negative factors affecting this industry are high competition and low barriers to entry.

Most revenue generated: US(US$1,993.00m in 2022)

The number of US health clubs has grown by an average of 2.98% annually over the last 10 years.



Revenue Forecast North America (2018-2028)
Source: Statista

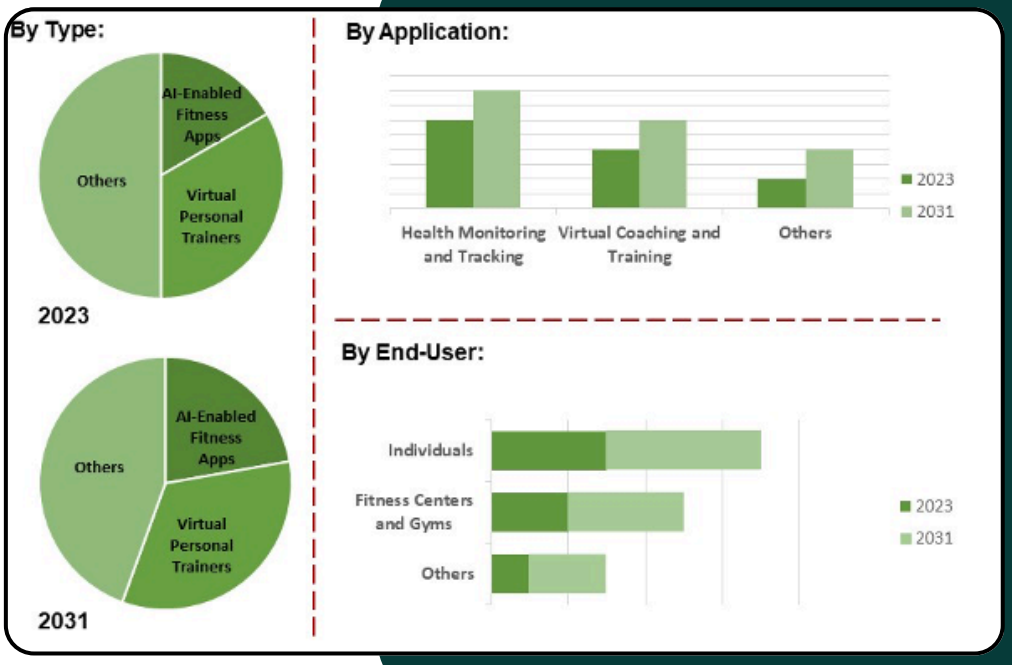
AI in the Fitness Market

Current size and growth rate of AI in Fitness

The global AI in Fitness and Wellness Market size was valued at US 7.8 billion in 2022 and is projected to expand at compound annual growth rate (CAGR) of 20.5% during the forecast period, reaching a value of USD 30.56 billion by 2030.

Dominating Regions & Countries

Developed regions like North America (largest market) and Europe, along with countries in the Asia Pacific region (fastest growing), are expected to dominate the market due to high technology adoption and growing health consciousness.



The AI in the fitness and wellness market is segmented into type, application, and end user. As per the type segment, the market is segmented into AI-enabled fitness apps, AI-integrated wearable devices, virtual personnel trainers, and AI-powered smart gym equipment.

Current Trends

Current trends include AI-powered fitness apps, wearable devices for health tracking, and the integration of AI in smart gym equipment to optimize workouts and enhance user experiences.

Key Driving Factors

The key factors driving the growth of the market include increasing demand for personalized fitness solutions, advancements in AI technology, and the rise of virtual coaching and training as traditional training reaches all time costs.

Space in the High End Boutique Fitness Market

Definiton

As their owners prefer to call them, “wellness destinations” or “social wellness clubs” there has been an explosion in luxury, application-only gyms which was reported by the New York Times.

Market size

The global boutique gym studios market was valued at US$34.35 billion in 2023. The market value is expected to reach US$54.81 billion by 2029.

The market is expected to grow at a CAGR of approx. 8% during the forecasted period of 2024-2029.

The global boutique gym studio market has witnessed significant popularity in recent years with consumers increasingly turning to boutique gym studios for a more personalized and unique workout experience.

This shift away from one-size-fits-all approaches seen in traditional gyms has attracted health-conscious individuals seeking a more immersive and effective fitness experience.

B2B

We operate exclusively as a B2B provider, focusing on partnerships with boutique high-end fitness centers to deliver tailored solutions that meet the needs of premium clientele. Additionally, we are exploring a potential collaboration with Virgin Active, a major player in the fitness industry with substantial resources and a strong market presence. With 40 locations across Italy alone, Virgin Active offers an extensive network and reach, making them an ideal partner to help elevate our offerings in this space.

Competitors

Xtra Vision AI, the healthcare startup that aims to revolutionize access to health protection tools with an AI platform for musculoskeletal services. As it states on Xtra's website, it "transforms any camera into a smart tracker to identify and quantify how the body moves." With this technology, users can train in front of their phones or computers and receive instructions on their movement. Computer vision analyzes the whole range of users' moves, assesses whether they perform the exercise properly, and provides them with real-time feedback.

The First AI Fitness Coach VAY detects mistakes in the execution of fitness exercises with your smartphone's camera.The VAY Fitness Coach is a fitness app analyses human movements in real-time and can provide insights during the training - a fully virtual personal coach, absolutely scalable as no direct human input is required.The technology behind the product is human pose estimation, a certain kind of computer vision that analyses human posture and movements.

Viso.ai offers a platform called Viso Suite. It's designed to be an end-to-end solution for businesses to leverage computer vision technology. It simplifies AI vision development. It allows for customization and integration with existing systems and tools. It offers enterprise-grade security. It has a list of visual AI applications in sports such as: Automated video assessment, Athletic motion tracking, Fitness Exercise Recognition, Rehabilitation support, AI-based injury detection and many more.

Kemtai's advanced computer vision technology analyzes human motion and provides real-time exercise feedback. Kemtai creates safer and more effective home physiotherapy and rehabilitation sessions. Kemtai provides the most accurate home exercise computer guidance and at affordable cost.

Service Offering

How does our service work and how do we protect it?

An AI-driven gym assistant integrated in the workout routine.

It operates through real-time video analysis of gym cameras, making it accessible without additional hardware. When a gym adopts the service, it can be integrated into the existing gym app, allowing members to use the AI features during their standard workout routine. Gym members can access use the camera footage to monitor and analyze their movements. For instance, if a user is performing squats too quickly, the app will prompt them to slow down, while adjusting exercises to match their fitness level. The feedback loop is responsive and adaptive, allowing the AI to suggest slower movements, extra reps, or even a more advanced exercise plan based on the user's progress, creating an interactive, real-time coaching experience.

Killer Features

- Real-time exercise monitoring and correction
- Rep counting and pace adjustments
- Dynamic adjustment of workout difficulty based on individual progress
- Health and safety features, such as posture correction to prevent injuries
- Live updates to the user's personal trainer

Protection Mechanisms

Patents will not be filed due to their high cost and public nature. We protect our technological edge through the "black box" approach, since neural networks cannot be easily reverse engineered.
Due to privacy and compliance considerations, *we* require explicit user consent for the monitoring. Privacy policies and terms of service will address data usage and ensure adherence to local data protection regulations. Forming strategic partnerships with gym app providers, such as Technogym, will facilitate early adoption of the feature, allowing it to be offered exclusively within these ecosystems and building a moat from competitors.

SWOT Analysis

STRENGTHS	WEAKNESSES
REAL-TIME PERSONALIZED ATTRACT NEW MEMBERS LOWER COSTS LONG TERM VALUE IDENTIFY HIDDEN NEEDS HEALTH-FOCUSED INJURY PREVENTION DOCKERIZED	EXPENSIVE INFRASTRUCTURE HALLUCINATIONS TECHNICAL DEVELOPMENT BRICK AND MORTAR PARTNERSHIPS RUNNING COSTS
OPPORTUNITIES	**THREATS**
GROWING FITNESS SPENDING EXPANSION IN ASIA INDUSTRY COLLABORATION WEARABLES SMART LIFESTYLE CHEAPER HARDWARE	PRIVACY AND REGULATORY COMPLIANCE PUBLIC DISTRUST CYBERSECURITY NOVELTY WEAR HUMAN CONNECTION

Possible tools

Live Leaderboard and User Competitions:

The app features a real-time leaderboard for users to compete in exercises, earning weekly badges and participating in monthly challenges. Motivational notifications encourage users to surpass their performance.

Augmented Reality (AR) Mode for Corrections:

This feature uses augmented reality to provide visual feedback on posture and exercise execution in real time, enhancing workout effectiveness and preventing injuries. (YOLO)

“Workout Buddy” Challenges and Co-op Mode

Users can train together with friends in cooperative challenges, sending encouragement and earning virtual rewards for reaching daily goals.

“Mood-Driven Workout Playlist” with Biometric Integration:

The app adapts music and exercises based on the user's mood by monitoring biometric parameters, offering a personalized workout experience.

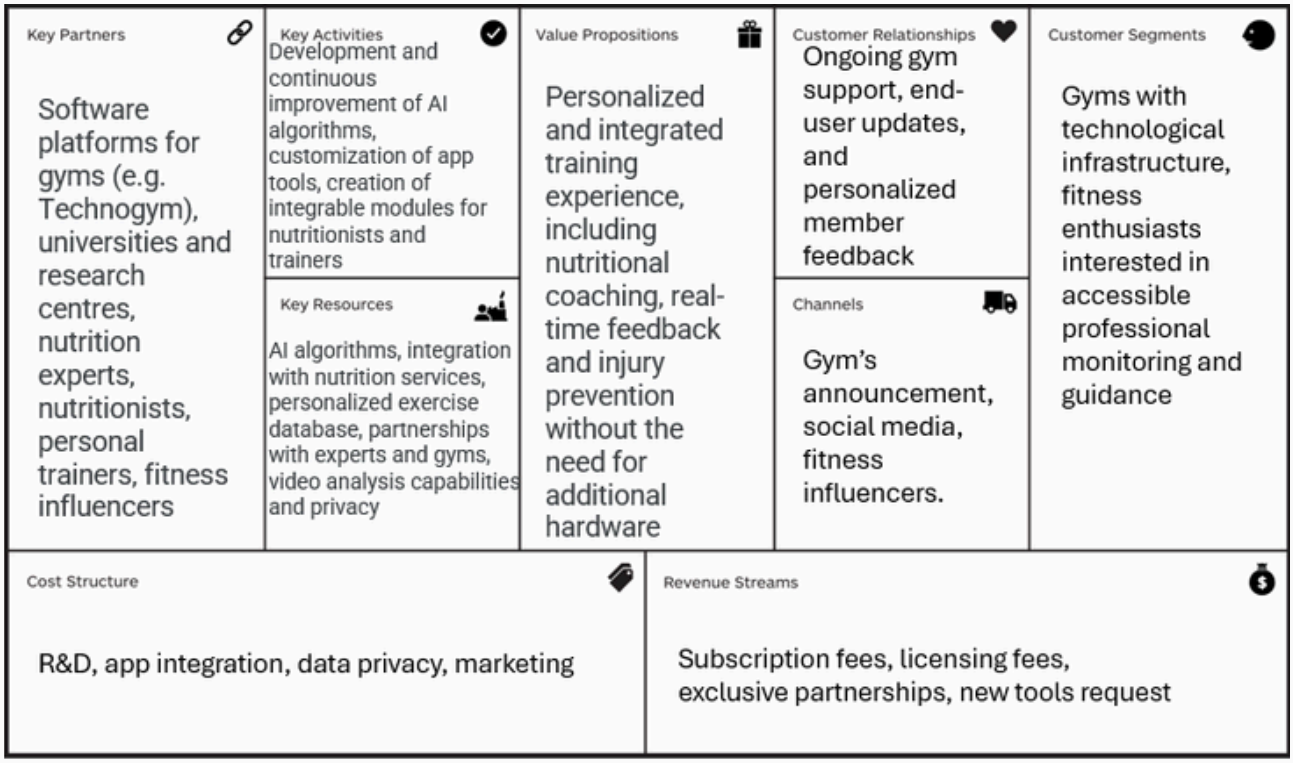
“Social Meal Prep” with Meal Planning Challenges:

Users can share meal plans and participate in healthy cooking challenges, earning badges and discounts, while also receiving pre-planned menus optimized for their workouts.

Business Model

Iterating on our Loss Function

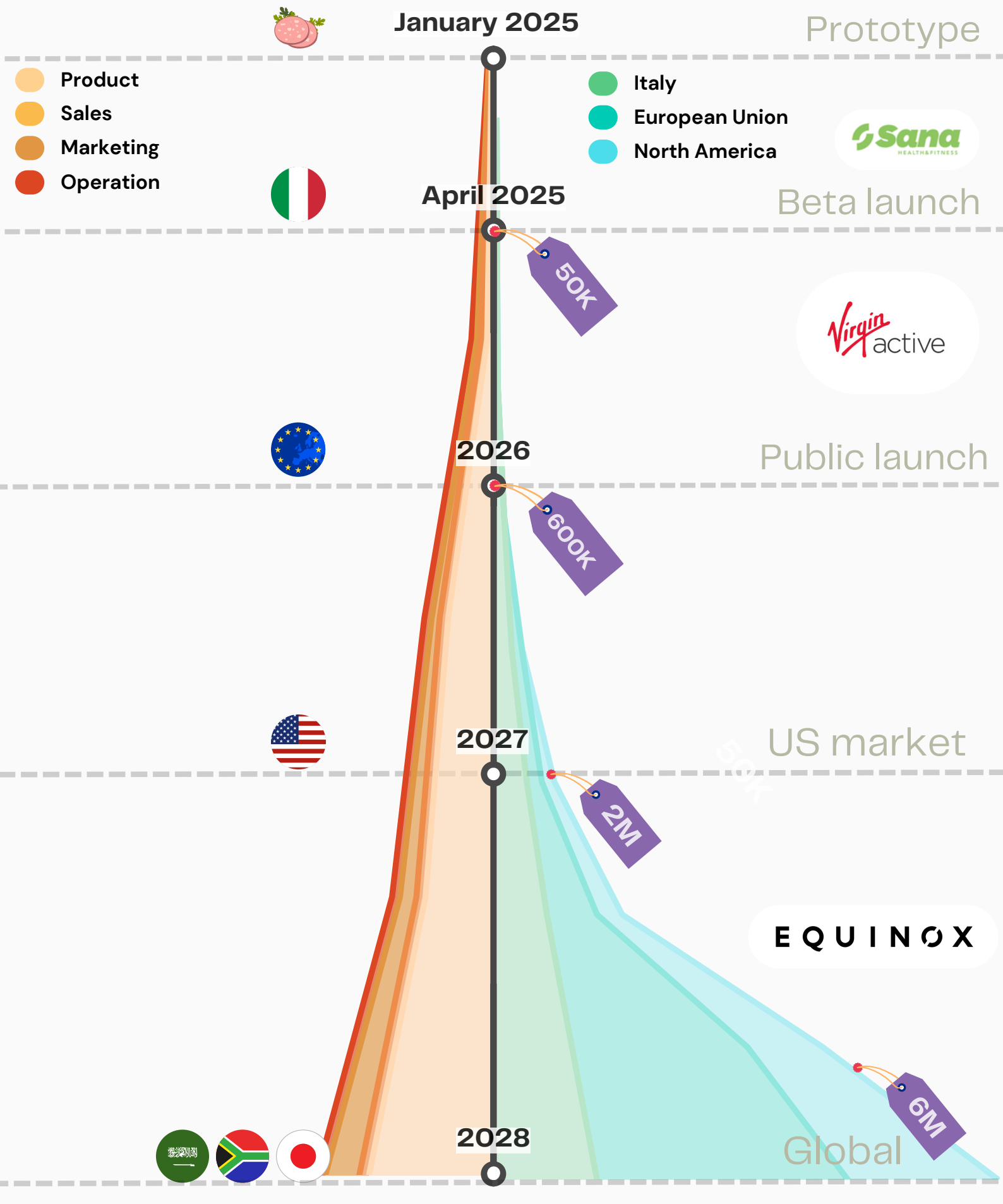
The Business Model Canvas



How to Interpret?

We place our business idea into the business flowchart to investigate our prototypical model. On our first iteration, we identified a privacy issue in customer relationships. Europe has strict data regulation. We can mitigate the problem by moving to North America. However, we will also increase transparency to build customer trust. It will be made clear this is not a replacement but a supplement for humans in the gym. We also identified a dependence on gym infrastructure. Adoption could be slow if brick and mortar owners are required to install high definition cameras. To comply with data regulations, we may be required to process everything locally, which necessitates state of the art computers to run the services.

Roadmap


January 2025
Prototype
Product
Sales
Marketing
Operation
Italy
European Union
North America
Sana
April 2025
Beta launch
50K
Virgin active
2026
Public launch
600K
2027
US market
2M
EQUINOX
6M
2028
Global

Financial Estimate 1

The costs

STARTUP COSTS

	Value
Capital Items (make fixed costs)	€41.500
Revenue Items (make variable costs)	€62.636
Other Costs	€18.500
TOTAL	€113.636

PUBLIC FUNDS

	Value
Smart&Smart Italia Program	€500.000
TOTAL	€500.000

SALES FORECAST (5 years)

	2025	2026	2027	2028	2029
Italy	€46.800		€1.123.200	€1.684.800	€1.684.800
EU		€624.000	€1.123.200	€2.808.000	€3.931.200
North America				€1.684.800	€3.931.200
International					€1.684.800
Total	**€46.800**	**€624.000**	**€2.246.400**	**€5.616.000**	**€11.232.000**
Growth rate		**12,3%**	**2,6%**	**1,5%**	**1%**

Financial Estimate 2

The capital

CASH FLOW FORECAST

	2025	2026	2027	2028	2029
Cash In	**€146.800**	**€1.124.000**	**€2.246.400**	**€5.616.000**	**€11.232.000**
Cash Out	**€135.060**	**€98.360**	**€209.660**	**€398.460**	**€405.460**
Yearly Balance	€11.740	€1.025.640	€2.036.740	€5.217.540	€10.826.540
Brought Forward		€11.740	€1.037.380	€3.074.119	€8.291.659
Cum. Balance	**€11.740**	**€1.037.380**	**€3.074.119**	**€8.291.659**	**€19.118.199**

PROFIT & LOSS (YEAR 1)

	2025
Gross Profit	€46.800
Overheads	€123.236
Net profit	- €76.436

Appendix

Find our sources here

Competitors

Research on our competitors and their specialization

Technogym
- https://www.technogym.com/it-IT/
- Computer Vision

Xtravision
- https://www.xtravision.ai
- Person identification

Technogym
- https://www.technogym.com/it-IT/
- Computer Vision

Xtravision
- https://www.xtravision.ai
- Person identification

Market

Research on the Fitness Industry

Fortune Business Insights
- https://www.fortunebusinessinsights.com/health-and-fitness-club-market-108652
- North America holds the biggest part of the market with a market share of 42.59%

New York Times
- https://www.nytimes.com/2023/03/25/style/exclusive-gym-memberships.html
- The premium fitness center business is growing and lucrative

Frameweb
- https://frameweb.com/article/why-luxury-gym-experiences-will-be-the-next-big-thing-for-fitness-again
- Gyms as experiences

Financial Analysis

Research to create the financial ask and operational roadmap

SWOT
- Template and instructions
- https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis

Roadmap
- How to organize a business roadmap
- https://www.aha.io/roadmapping/guide/roadmap/business-roadmap

Smart&Smart Italia (Italy's Program)
- From 100.000 to 1,5 million
- https://www.invitalia.it/cosa-facciamo/creiamo-nuove-aziende/smartstart-italia?utm_source=chatgpt.com

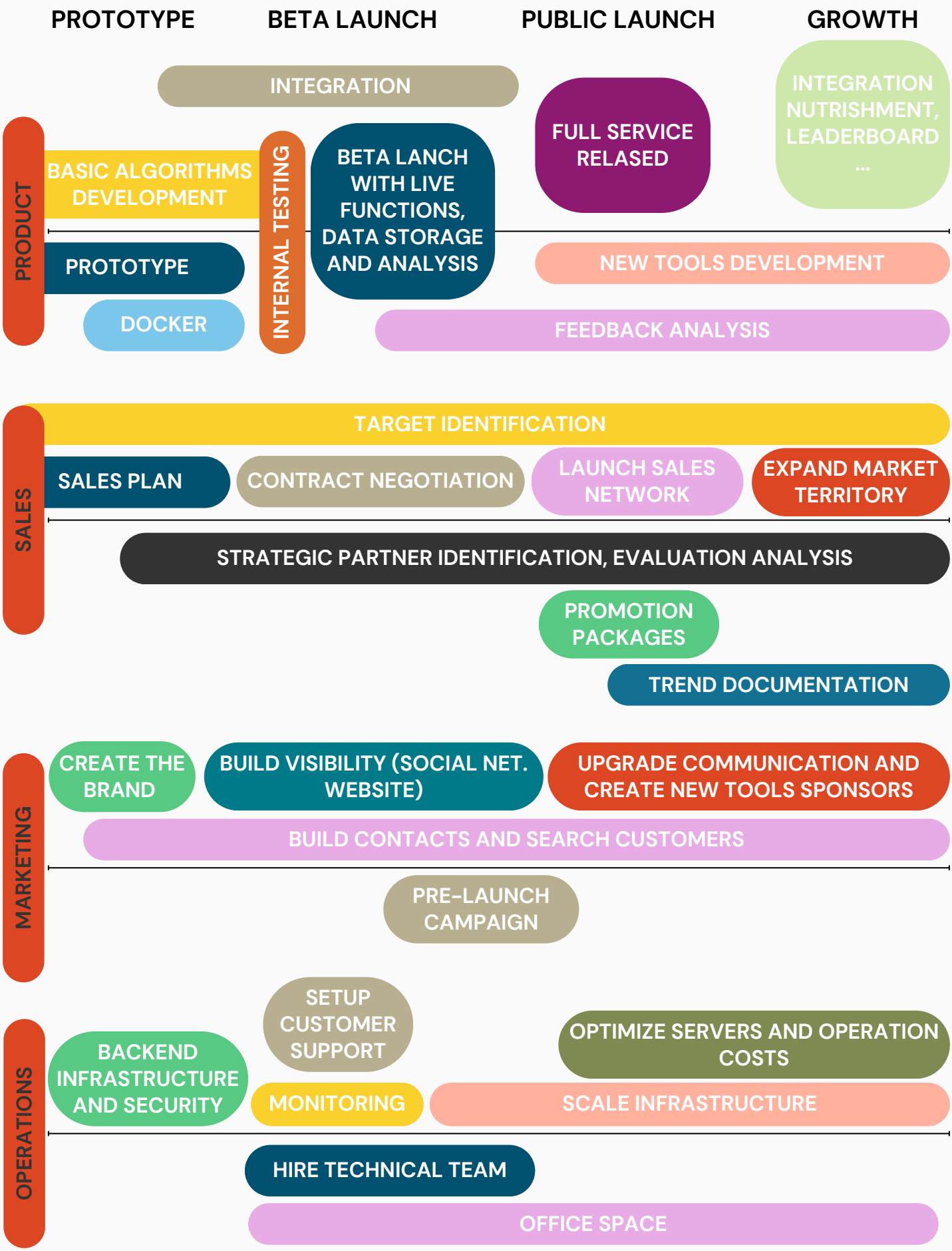
Mentorship

People we thank for their help and expertise

Riccardo Fini
- Full Professor Department of Management UniBo

Natalia Panchieri
- PhD Student Department of Management UniBo

Business Roadmap



Product Roadmap

PROTOTYPE 3 Months	BETA LAUNCH 9 Months	PUBLIC LAUNCH 1 Year	GROWTH 3 Years

INTEGRATION

BASIC ALGORITHMS DEVELOPMENT

PROTOTYPE

DOCKER

INTERNAL TESTING

BETA LAUNCH WITH LIVE FUNCTIONS, DATA STORAGE AND ANALYSIS

REAL SERVICE RELASED

INTEGRATION NUTRISHMENT, LEADERBOARD ...

NEW TOOLS DEVELOPMENT

FEEDBACK ANALYSIS

HARDWARE

PROTOTYPE 3 Months	BETA LAUNCH 9 Months	PUBLIC LAUNCH 1 Year	GROWTH 3 Years
PC/LAPTOP WITH CPU CAMERAS AND SENSORS	GPU CLOUD DATABASE aws	SERVER GPU BASED computation SERVER CLOUD data analysis	SERVER SCALABILITY
3500 EUROS	2666 EUROS	26k EUROS	76k EUROS

SOFTWARE

PROTOTYPE 3 Months	BETA LAUNCH 9 Months	PUBLIC LAUNCH 1 Year	GROWTH 3 Years
FRAMEWORK pytorch PRE TRAINED MODEL openpose APP DEVELOPER electron O.S. docker	capturing BODY MOTION IN 3D arkit TRAIN MODELS electricity API / WEBSITE squarespace backen DATASETS	ML DEVELOPMENT one engineer FULL STACK one developer	NEW SOFTWARE tools and features STABILITY squashing bugs
0 EUROS	25804 EUROS	55k EUROS	140k EUROS

Sales Roadmap

PROTOTYPE 3 Months	BETA LAUNCH 9 Months	PUBLIC LAUNCH 1 Year	GROWTH 3 Years

TARGET IDENTIFICATION

SALES PLAN

CONTRACT NEGOTIATION

LAUNCH SALES NETWORK

EXPAND MARKET TERRITORY

STRATEGIC PARTNER IDENTIFICATION, EVALUATION ANALYSIS

PROMOTION PACKAGES

TREND DOCUMENTATION

MARKET RESEARCH

CUSTOMER PROFILING

SALES FUNNEL DESIGN

PREPARE CONTRACT TEMPLATES

NEGOTIATE TERMS & SIGN CONTRACTS

BETA TESTING AND PERFORMANCE ANALYSIS

PARTNERSHIP DEVELOPMENT

CRM INTEGRATION

8500 EUROS	12140 EUROS	2000 EUROS	1200 EUROS

Marketing Roadmap

PROTOTYPE 3 Months	BETA LAUNCH 9 Months	PUBLIC LAUNCH 1 Year	GROWTH 3 Years

CREATE THE BRAND

BUILD VISIBILITY (SOCIAL NET. WEBSITE)

UPGRADE COMMUNICATION AND CREATE NEW TOOLS SPONSORS

BUILD CONTACTS AND SEARCH CUSTOMERS

PRE-LAUNCH CAMPAIGN

ADVERTISEMENT

INTERNAL

LOGO Designer	COMPANY CULTURE Activities	MERCHANDISE Sweatshirt	MARKETING DIRECTOR
BUSINESS PLAN Century Club	SEOs	DEVELOP MARKETING COLLATERAL	
SALES PLAN Century Club	TRADE SHOWS		
0 EUROS	5720 EUROS	3072 EUROS	30k EUROS

EXTERNAL

	WEBSITE Front End Squarespace	SOCIAL MEDIA Instagram	MERCH Brand Clothes
		INFLUENCER Brand Showcase	CAMPAIGN Long term
?? EUROS	?? EUROS	8k EUROS	30k EUROS

Operations Roadmap

PROTOTYPE 3 Months	BETA LAUNCH 9 Months	PUBLIC LAUNCH 1 Year	GROWTH 3 Years
BACKEND INFRASTRUCTURE AND SECURITY	SETUP CUSTOMER SUPPORT	OPTIMIZE SERVERS AND OPERATION COSTS	
	MONITORING	SCALE INFRASTRUCTURE	
	HIRE TECHNICAL TEAM	HUMAN RESOURCES	
	OFFICE SPACE		
	ACCOUNTING		
	LEGAL COSTS		

	OFFICE		
	Additional FURNITURE		
various SUBSCRIPTIONS	ACCOUNTING services	REGULATORY AND COMPLIANCE fees	
	LEGAL services	INSURANCES	
1000 EUROS	9400 EUROS	4000 EUROS	1000 EUROS

Appendix

CAPITAL ITEMS			
DETAILS	HAVE ALREADY (£)	NEED TO BUY (£)	TOTAL
Hardware (PC)	9000	2000	11,000 €
Camera and Sensors	0	3500	3,500 €
Initial Dataset acquisition	0	25000	25,000 €
Prototyping tools		2000	2,000 €
TOTALS	£9000	£32500	41,500 €

REVENUE EXPENSES			
DETAILS	EXTRA INFO		TOTAL
Electricity	50 euros/month		600 €
Salaries	ML engineer = 30,000, full-stack dev = 25,000		55,000 €
Cloud	AWS		1,332 €
Licences	For software like frameworks, etc.		100 €
Marketing	SEO = 2000, Trade shows = 3000		5,000 €
Subscription services			1,204 €
TOTAL			62,636 €

OTHER COSTS		
DETAILS	EXTRA INFO	TOTAL
Rent for office		7,200 €
Insurance		4,000 €
Legal and Accounting Services	accountanting services = 2600 yearly, lawyer services = 3000 yearly	5,600 €
Regulatory and Compliance Fees		1,500 €
Office equipment		200 €
TOTAL		18,500 €

Appendix

5-year sales forecast						
	2025	2026	2027	2028	2029	5-year totals
Monthly Price per Member	0,50%	1,00%	1,80%	1,80%	1,80%	
Monthly Price per gym	260€	520€	936€	936€	936€	
Quantity of Gyms	15	100	200	500	1.000	1.815,00
Annual revenue	46.800€	624.000€	2.246.400€	5.616.000€	11.232.000€	19.765.200€
Growth rate		12,3	2,6	1,5	1,0	

Appendix

CASH FLOW FORECAST						
	2025,	2026,	2027,	2028,	2029,	TOTAL
CASH IN						
Sales	46.800€	624.000€	2.246.400€	5.616.000€	11.232.000€	19.765.200€
Angel Investor	50.000€					50.000€
Smart&Smart Italia Program		500.000€				500.000€
Loans	50.000€					50.000€
TOTAL	146.800€	1.124.000€	2.246.400€	5.616.000€	11.232.000€	9.133.200€
CASH OUT						
Software	0					0€
Camera and Sensors	3.500	0	0	3.000	0	6.500€
Hardware (PC)	2.000	2.000	6.000	6.000	6.000	22.000€
Datasets	25.000	0	0	0	0	25.000€
Prototyping tools	2.000	0	0	0	0	2.000€
Cloud AWS	1.332	1.332	1.332	1.332	1.332	5.328€
Subscription services	1.204	1.204	1.204	1.204	1.204	3.612€
Marketing	5.000	5.000	7.000	15.000	15.000	32.000€
Advertisement	0	3.000	8.000	20.000	30.000	61.000€
Regulatory and Compliance Fees	13.000	2.000	5.700	2.000	2.000	24.700€
Office Rent	5.400	7.200	24.000	24.000	24.000	60.600€
Office Equipment	200	200	1.000	500	500	2.400€
Insurance	4.000	4.000	8.000	8.000	8.000	24.000€
Legal & Accounting Services	5.600	5.600	25.600	25.600	25.600	62.400€
Salaries	55.000	55.000	110.000	280.000	280.000	780.000€
Repaying Loan (TAN 6.80% TAEG 8,22%)	11.824	11.824	11.824	11.824	11.824	59.121€
TOTAL	135.060€	98.360€	209.660€	398.460€	405.460€	841.541€
YEARLY BALANCE	11.740	1.025.640	2.036.740	5.217.540	10.826.540	
BROUGHT FORWARD	0	11.740	1.037.380	3.074.119	8.291.659	
CUM. BALANCE	11.740€	1.037.380€	3.074.119€	8.291.659€	19.118.199€	

Appendix

PROFIT & LOSS ACCOUNT (YEAR 1 – PROJECTED)		
SALES		15,000€
Cost of Sales		
Gross Profit		15,000€
OVERHEADS		
Software	0	
Camera and Sensors	3,500	
Hardware (PC)	2,000	
Datasets	25,000	
Prototyping tools	2,000	
Cloud AWS	1,332	
Subscription services	1,204	
Marketing	5,000	
Regulatory and Compliance Fees	13,000	
Office Rent	5,400	
Office Equipment	200	
Insurance	4,000	
Legal & Accounting Services	5,600	
Salaries	55,000	
Repaying Loan (TAN 6.80% TAEG 8,22%)	11,824	
TOTAL	123,236€	
NET PROFIT		-108,236€

Contact us for further inquiries

nicolasivan.cridlig@studio.unibo.it | +1 8587222735

Acknowledgements

I would like to thank my supervisor, Prof. Stefano Mattoccia, and my co-supervisor, Dott. Enrico Mannocci, whose feedback and encouragement kept the thesis on track.

I am grateful to Vincenzo for believing in this idea from the very beginning, providing access to his gym, and hiring me. Martina contributed far more than exercise demonstrations: her expertise as a trainer shaped the selection of joint angles and exercises, and her management made the data collection run smoothly.

I owe a special thanks to my parents, Mila and Regis, whose never-ending support made it possible for me to pursue this degree. To my siblings, Anna and Alexandre, for being there. And to Richard, for inspiring me to study in Italy.

Finally, I owe the Boy Scouts of America, for building me physically strong, mentally awake, and morally straight.