
# Measuring Movement, Not Just Motion: A Pose-Based, DTW+LSTM Pipeline for Football Rehab

Jaime Tellie  MSc Data Analytics

August 14, 2025

**Abstract**

This thesis presents a football–centric pipeline for physiotherapy and exercise analysis that measures *movement degradation* and technique quality directly from single–view videos. Using MediaPipe Pose to extract joint–angle time series, we compute a windowed *DTW degradation score*—a timing–robust distance from each 30-frame window to a baseline reference from the same session—capturing execution drift under fatigue or compensation. On top of these per–window labels, we train LSTM-based sequence models to predict degradation from angle windows (basic features) and from angles plus engineered descriptors (ROM, angular velocity, symmetry dispersion; featured). The pipeline includes reproducible preprocessing (30 FPS, windows of 30 with stride 10), robust handling of missing landmarks, a two–stage normalization scheme (dataset-time MinMax, training-time RobustScaler + target scaling), and comprehensive EDA/visualization. Results across squats, lunges, bench press, and pull-ups show consistent window-level prediction with basic features outperforming in three of four exercises, highlighting the sensitivity of derivative features to low-quality 2D landmarks. The approach enables objective, clinician-aligned tracking of mobility, tempo/load proxies, and asymmetry for return-to-play support.

# Acknowledgements

I would like to thank my supervisors, mentors, and peers who provided guidance and support throughout this research.

# Executive Summary

This project develops a practical AI system to evaluate movement quality for **return-to-play** analysis and rehab. Using pose estimation to extract joint-angle time series, the pipeline computes a DTW-based *degradation* distance to session baselines and provides window-level scoring. LSTM models learn temporal patterns to predict movement-quality trends (e.g., fatigue, compensation). We also describe a prospective natural-language reporting layer to aid clinical decision-making. The system is modular, scalable, and intended to augment physiotherapists during recovery and players during technique practice.

# Contents

# Chapter 1

# Introduction and Research Questions

## 1.1 Problem Statement: Medical/Physio Movement Quality

Professional football clubs require reliable, objective methods to monitor movement quality during rehabilitation and technique practice:

- **Physiotherapy departments** work closely with players recovering from injury and need consistent, quantitative indicators of progress.
- **Recovering players** must re-establish good form and avoid compensatory patterns (e.g., asymmetries) that risk reinjury.
- **Technique practice** demands timely, actionable feedback—ideally *window-level*—rather than coarse per-session summaries.
- **Operational needs** include real-time or near-real-time feedback and longitudinal tracking across sessions, cameras, and exercises.

These requirements motivate an automated, interpretable pipeline that extracts joint angles, scores movement quality, highlights asymmetry/compensation, and communicates results in clinician-friendly language.

## 1.2 Research Questions

1. Can joint-angle sequences be reliably used to score movement quality in footballers recovering from injury?

2. Can LSTMs predict fatigue, asymmetry, or compensation trends over a training session?

3. Can pose-derived features assist in classifying roles, tactics, or positions? (*exploratory; out of current scope for experiments*)

## 1.3 Scope and Contributions

**Current scope.** Pose-based analysis for physiotherapy and training. From video, we extract angles per frame, compute engineered metrics (range of motion, angular velocity), measure *degradation* to references with DTW, and train LSTMs to predict movement-quality scores. The pipeline supports *return-to-play* comparisons (pre- vs. post-injury), *real-time rehab feedback (prototype)*, and *joint-load proxies* via angular motion velocity.

**Future work:** A computer-vision application for football actions (e.g., free kicks) that: (1) analyzes technique against professional exemplars and returns a degradation score; (2) suggests targeted improvements; and (3) optionally simulates successful executions via a 3D model or outline of the player. A VLM layer would provide granular, clinician-friendly feedback on form elements (hips, shoulders, ankles).

# Chapter 2

# Objectives

- Develop a pose estimation module for technique comparison and window-level scoring.
- Train sequence models (LSTM) for movement-quality trend prediction and anomaly detection.
- Provide reproducible, modular code and visualizations for research and applied settings.

# Chapter 3

# Methodologies and Techniques

## 3.1 Dataset

We use a custom, exercise-based motion dataset designed to evaluate and score the quality of movements using pose estimation and time-series analysis. The corpus comprises raw videos and derived pose sequences for four football-adjacent exercises: **squats**, **lunges**, **bench press**, and **pull-ups**. Each sample contains (i) the original video, (ii) frame-level pose landmarks, and (iii) engineered joint-angle features.

### 3.1.1 Overview

The dataset targets physiotherapy and return-to-play scenarios. It captures a range of performance variations (controlled vs. inconsistent tempo, shallow vs. deep ROM, compensated vs. symmetric execution) to foster generalization across body types, camera viewpoints, and speeds.

### 3.1.2 Data Acquisition

**Public sources.** We bootstrap with publicly available human activity datasets (e.g., KTH Action) and carefully select clips relevant to our target movements. While such datasets are not exercise-specific, they provide motion diversity to pre-test the pipeline.

### 3.1.3 Specific Exercise Extraction and Labelling

We (1) label videos by exercise type (squat, lunge, bench press, pull-ups), (2) trim clips to the relevant action window, and (3) retain a spread of technique qualities. For supervised settings, optional notes on visible compensations (valgus knee, hip shift) are attached.

### 3.1.4 Dataset Characteristics and Class Balance

Exercise counts are not perfectly balanced (e.g., more squats than lunges). Where class imbalance emerges (either by exercise or quality labels), we mitigate via stratified splits and, when needed, sample weighting on the learning objective.

### 3.1.5 Privacy, Ethics, and Limitations

Pose estimation is sensitive to lighting, occlusion, and resolution. Lower-quality footage leads to unstable landmarks and, consequently, noisier angle series. This practical constraint influences preprocessing choices (see Section 4.2).

### 3.1.6 Pros and Cons

**Pros:** real-world relevance for physio feedback; low-cost capture (camera + pose model); broad applicability (rehab tracking, technique coaching). **Cons:** variable video quality; annotation effort for labels; potential class imbalance.

## 3.2 Modelling

We model time-series of joint angles using LSTMs/GRUs for prediction and scoring.

### 3.2.1 Input Representation

- Window length: `30` frames (1.0 s at 30 FPS); stride `10` frames ($\sim$0.33 s).
- FPS: `30`.
- Feature sets: **Basic** (angles only) vs **Featured** (angles + ROM + angular velocity + symmetry indices + degradation).
- Scaling: `MinMax` to [0,1] across features (fit on train; applied to val/test).

### 3.2.2 Architecture and Training Hyperparameters

- Architecture: Conv1D(64) → GRU(128) → Dense(128) → Dense(1).
- Dropout: [0.2, 0.3, 0.3] with BatchNorm after Conv1D and GRU.
- Optimizer: Adam with learning rate = 0.005.
- Loss: Huber; Metrics: MAE and $R^2$.
- Batch size: 32; Max epochs: 100 with EarlyStopping (patience 10) and ReduceLROnPlateau (factor 0.5); ModelCheckpoint on best val loss.

## Loss and metrics

We optimize the *Huber* loss (Huber, 1964), which is less sensitive to outliers than MSE:

$$\mathcal{L}_\delta(r) = \begin{cases} \frac{1}{2}r^2, & |r| \le \delta, \\ \delta\big(|r| - \frac{1}{2}\delta\big), & |r| > \delta, \end{cases} \qquad r = \hat{y} - y,\ \delta > 0.$$

We report mean absolute error (MAE) and coefficient of determination ($R^2$):

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i|, \qquad R^2 = 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i (y_i - \bar{y})^2}.$$

### 3.2.3 Final Model Configuration

| ID | Features | Win | Architecture | Dropout | LR | Loss |
|---|---|---|---|---|---|---|
| A1 | basic / featured | 30 | Conv1D(64) → GRU(128) → Dense(128) → Dense(1) | 0.2/0.3/0.3 | $5 \cdot 10^{-3}$ | Huber |

Table 3.1: Final configuration used across all experiments.

# Chapter 4

# Data Preprocessing and Feature Engineering

## 4.1 Video Frame Extraction

**Objective.** Convert videos to temporally consistent frame streams for pose inference and time-series modeling. **Method.** Frames are sampled at a fixed rate (actual: 30 FPS). OpenCV handles video I/O, resizing, and optional deinterlacing. Consistent sampling preserves temporal structure for sliding-window inputs.

## 4.2 Pose Estimation

**Model.** MediaPipe Pose Landmarker (VIDEO mode) produces 2D landmarks per frame.

**Keypoints.** We track joints needed for hip, knee, elbow, and shoulder angle calculations.

**Thresholds.** Due to low-quality source footage in parts of the corpus, we set:

- `min_pose_detection_confidence = 0.1`
- `min_pose_presence_confidence = 0.1`

*Rationale:* With stricter thresholds (e.g., $\geq$0.6), MediaPipe failed to return landmarks in more than half of the videos/frames, which would have:

- Drastically reduced usable data
- Biased angle distributions

Lower thresholds retain samples but require downstream robustness measures:

- Interpolation of missing values
- Temporal smoothing
- Statistical sanity checks

**Limitations:** This trade-off means that video quality, lighting conditions, occlusion, and resolution materially affect landmark stability, feature quality, and ultimately model performance.

## 4.3 Feature Extraction

From 2D landmarks, we compute per-frame joint angles and derive temporal features for each body side and joint family:

- **Angles (baseline features):**
  - Lower body: Hip, knee (left/right)
  - Upper body: Elbow, shoulder (left/right)
- **Range of Motion (ROM):** maximum–minimum angle over the sliding window (e.g., `hip_angle_left_rom`)
- **Angular Velocity:** first-order derivative; window mean, max, std (e.g., `knee_angle_right_velocity_max`)
- **Symmetry Indices:** dispersion between left/right trajectories (e.g., `hip_angle_symmetry_std`)
- **ROM Drop-off:** relative ROM change vs. a reference window (fatigue/movement constraint proxy)
- **DTW Degradation:** DTW distance to a reference execution summarized per window

**Implementation Notes:**

- Sliding window size: 30 frames (1 second at 30 FPS)
- Angle calculations use a 2D dot-product method
- Velocity computed via central differences
- MinMax scaling fit on the training split only; applied to val/test

*Clinical relevance.* ROM relates to mobility; angular velocity summarises tempo and provides a proxy for joint load; symmetry indices highlight compensations; ROM drop-off and DTW degradation track form consistency/fatigue over time.

## Exercise-Specific Feature Sets

- **Squat/Lunge:** `[hip_angle_left, knee_angle_left, hip_angle_right, knee_angle_right]` plus ROM, velocity, symmetry, drop-off, and DTW.
- **Bench Press:** `[elbow_angle_left, elbow_angle_right]` plus velocity profiles, bilateral symmetry, drop-off, and DTW.
- **Pull-Ups:** `[elbow_angle_left, elbow_angle_right, shoulder_angle_left, shoulder_angle_right]` plus ROM, velocity, symmetry, drop-off, and DTW.

*Illustrative outputs.* Per-exercise JSON reports include fields like `*_rom`, `*_velocity_mean/max/std`, `*_symmetry_std {value, score, level}`, `*_rom_dropoff`, and `*_dtw_degradation`, with a natural-language interpretation summarizing mobility, tempo, and variability.

## Engineered Feature Map

Table 4.1: Engineered feature map linking biomechanical measurements to physiotherapy-relevant insights. Key: ROM = Range of Motion, DTW = Dynamic Time Warping.

| **Feature** | **Measurement** | **Clinical Insight** |
|---|---|---|
| ROM | Maximum–minimum joint angle over sliding window | • Mobility/flexibility assessment<br>• Movement depth sufficiency<br>• Joint lockout completion |
| Angular velocity (mean) | Average rate of angle change | • Tempo control evaluation<br>• Tendon loading estimation<br>• Movement pacing quality |
| Angular velocity (max) | Peak rate of angle change | • Explosiveness metric<br>• Stress risk indicator<br>• Transition abruptness |
| Angular velocity (std) | Variability in movement speed | • Motor control consistency<br>• Movement instability<br>• Coordination assessment |
| Symmetry (std) | Left-right trajectory dispersion | • Compensation patterns<br>• Unilateral weakness<br>• Guarding behavior |
| ROM drop-off | ROM reduction vs. baseline | • Fatigue progression<br>• Stiffness development<br>• Pain-avoidance patterns |

Continued on next page

| Feature | Measurement | Clinical Insight |
|---|---|---|
| DTW degradation | Distance from reference trajectory | • Technique deviation<br>• Form deterioration<br>• Movement pathology |

## 4.4 Windowing

Windows of 30 frames (1.0 s at 30 FPS) with stride 10 ($\sim$0.33 s) are extracted from angle time-series to support LSTM inputs and window-level scoring. Peaks/troughs are not required; the method is robust across variable tempos and partial motions.

## 4.5 Normalization

We employ a two–stage scheme designed to (i) standardize feature ranges across the corpus and (ii) add robustness to outliers during training, without leakage.

**Stage A — dataset preparation (offline).** When creating the basic/featured datasets, we apply per–feature MinMax scaling to $[0, 1]$ within the training split and reuse those parameters for validation/test.

**Stage B — model training (online load).** At training time we load the already MinMaxed arrays and apply:

- **Robust scaling on** $X$ (per feature, fit on *train only*):

$$x^{\text{rob}} = \frac{x - \text{median}(x)}{\text{IQR}(x)},$$

  mitigating the influence of outliers from jittery landmarks.

- **Range scaling on** $y$ to $[0, 1]$ within train (reused on val/test):

$$y' = \frac{y - y_{\min}}{\max(y_{\max} - y_{\min},\, \varepsilon)}, \quad \varepsilon = 10^{-10}.$$

**Implementation.** The loader below (abridged) reflects Stage B exactly:

```
def load_and_preprocess_data(exercise, augment_with_features=True):
    # loads X(.npy), y(.npy)  -> RobustScaler on X, [0,1] on y
```

**Leakage guard.** All scalers in Stage B are fit on the training partition only and applied to validation/test. This keeps label distributions comparable while preserving a fair evaluation.

## 4.6 Augmentation (Optional)

To improve generalization without altering semantics, we use time-series augmentations on angle/velocity sequences: mild Gaussian jitter, temporal jitter (shift), magnitude scaling, and temporal dropout (cutout). Augmentations are disabled for validation/test. Experiments compare **basic (angles only)** vs. **featured** (angles + ROM + velocity + symmetry + degradation).

## 4.7 Exploratory Data Analysis (EDA)

We compute correlation heatmaps across angles and engineered features to detect redundancy and drive feature selection. Representative windows of angle trajectories are visualized per exercise to confirm window coverage and typical dynamics.

# Chapter 5

# Method: Pose-Based Movement Quality Pipeline

## 5.1 Joint Angle Extraction

We use MediaPipe Pose (VIDEO mode) to obtain 2D landmarks per frame at 30 FPS. Let $\mathbf{p}_A = (x_A, y_A)$, $\mathbf{p}_B$, $\mathbf{p}_C$ be joint coordinates (in pixels) for three landmark points forming the angle at vertex $B$. We define vectors $\mathbf{u} = \mathbf{p}_A - \mathbf{p}_B$ and $\mathbf{v} = \mathbf{p}_C - \mathbf{p}_B$, and compute the joint angle

$$\theta = \arccos\left(\frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \, \|\mathbf{v}\|}\right) \cdot \frac{180}{\pi}, \quad \text{with } \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \, \|\mathbf{v}\|} \in [-1, 1] \text{ (clamped)}.$$

Angles are computed per frame for both sides using the following landmark triplets:

- Hip: $\angle(\text{shoulder}, \text{hip}, \text{knee})$
- Knee: $\angle(\text{hip}, \text{knee}, \text{ankle})$
- Elbow: $\angle(\text{shoulder}, \text{elbow}, \text{wrist})$
- Shoulder: $\angle(\text{hip}, \text{shoulder}, \text{elbow})$

**Smoothing and velocity.** Let $\theta_t$ be the angle at frame $t$ (sampling period $\Delta t = \frac{1}{30}$ s). We apply a short moving average (window $k$=5) to reduce frame noise:
$\tilde{\theta}_t = \frac{1}{k} \sum_{i=-(k-1)/2}^{(k-1)/2} \theta_{t+i}$.

Angular velocity uses a central difference:

$$\dot{\theta}_t \;=\; \frac{\tilde{\theta}_{t+1} - \tilde{\theta}_{t-1}}{2\,\Delta t} \quad [\deg\,\mathrm{s}^{-1}].$$

**Missing data handling.** Due to low video quality and occlusion, landmarks may be missing. We linearly interpolate gaps of up to 5 consecutive frames per joint; longer gaps mark the affected frames invalid. A window is considered valid if at least 70% of its frames are valid for all joints used in that window (otherwise the window is skipped).

**Windowing.** From each angle time series we extract fixed windows of $W$=30 frames (1.0 s) with stride $S$=10 frames ($\sim 0.33$ s). For learning we later apply MinMax scaling (fit on train) to all features (see Data Preprocessing). For degradation only (below), we $z$-normalize each windowed series (per channel) to remove scale bias: $\hat{\theta} = (\tilde{\theta} - \mu)/\sigma$.

## 5.2 Degradation Scoring (DTW)

We quantify per–window *movement degradation* by aligning each windowed multi–joint sequence $X$ to a session baseline $R$ using Dynamic Time Warping (DTW) (Sakoe and Chiba, 1978; Salvador and Chan, 2007). DTW non-linearly aligns sequences to handle variable timing and tempo while preserving shape. The resulting distance is our degradation label and is aggregated across windows for session-level summaries.

$$D(i,j) = d(x_i, r_j) + \min\{D(i-1,j),\, D(i,j-1),\, D(i-1,j-1)\},$$

with $D(0,0) = 0$ and $D(i,0) = D(0,j) = \infty$ for $i, j > 0$. We use squared Euclidean local cost $d(x_i, r_j) = (x_i - r_j)^2$.

**Multi-channel extension.** For a set of $C$ joints (channels), we sum channel-wise DTW distances:

$$\mathrm{DTW}_{\mathrm{multi}}(X,R) \;=\; \sum_{c=1}^{C} \mathrm{DTW}\big(x^{(c)}, r^{(c)}\big),$$

after per-channel $z$-normalization within the window. In this study we use equal weights across channels; exercise-specific weighting is possible but not required for our results.

**Warping constraint.** To prevent pathological alignments and reduce complexity, we apply a Sakoe–Chiba band of width $b = \lfloor 0.1\,W \rfloor$ (i.e., $\pm 3$ frames at $W{=}30$), restricting the admissible path to $|i - j| \leq b$. This reduces runtime from $O(W^2)$ to $O(W\,b)$ per channel and favors physiologically plausible alignments.

**Reference choice and degradation score.** For session-relative analysis we define the reference $R$ as the first valid window of the same session (baseline). The per-window *degradation score* is then

$$s_{\text{deg}}(X) \;=\; \text{DTW}_{\text{multi}}(X, R),$$

and session-level summaries use the mean and 95th percentile across windows. When an external canonical reference is available (e.g., a clean exemplar), the same formulation applies with $R$ fixed to that template.

### Learning target

All supervised experiments use the window–level degradation score $s_{\text{deg}}$ as the regression target. Predictions are made *per window* (30 frames), not per frame, and evaluated with MAE and $R^2$ on held-out splits.

## 5.3 Asymmetry and Window-Level Analysis

To detect compensations, we compare left/right homologous joints within each window.

**Symmetry dispersion.** For a left/right pair (e.g., knees), define the pointwise difference $\delta_t = \hat{\theta}_t^{\text{left}} - \hat{\theta}_t^{\text{right}}$ (normalized signals). The *symmetry index* for the window is

$$\text{SymStd} \;=\; \text{std}_t(\delta_t) \quad \text{[z-units]},$$

where larger values indicate greater bilateral dispersion. We report SymStd per joint family (hips, knees, elbows, shoulders) and aggregate by mean across families present in the exercise.

**ROM and velocity summaries.** Within each window we compute range of motion $\text{ROM} = \max(\tilde{\theta}_t) - \min(\tilde{\theta}_t)$, and angular-velocity summaries (mean / max / std). These

complement DTW by quantifying depth, tempo, and stability.

**Session-level aggregation.** For each session we compute:

- Mean and 95th percentile of $s_{\text{deg}}$ (DTW degradation).
- Mean of SymStd across windows (and per joint family).
- Mean of ROM and velocity statistics per joint.

These window-level metrics support longitudinal tracking (e.g., pre- vs. post-injury) without relying on explicit repetition segmentation.

# Chapter 6

# Implementation and System Development

## 6.1 Overview

The system is implemented as a modular, configuration-driven pipeline:

$$\begin{aligned}\text{video ingestion} &\rightarrow \text{pose landmarks} \rightarrow \text{angle computation} \\ &\rightarrow \text{feature engineering} \rightarrow \text{scaling} \rightarrow \text{windowing} \\ &\rightarrow \text{DTW scoring \& LSTM modeling} \rightarrow \text{visualization.}\end{aligned}$$

Modules communicate via versioned, on-disk artifacts (CSV/JSON for features and metrics; PNG for charts), enabling reproducibility and partial re-runs without reprocessing earlier stages.

## 6.2 Software Stack and Reproducibility

We use Python (NumPy, pandas, scikit-learn, matplotlib), OpenCV for video I/O, and MediaPipe Pose for landmarking. Modeling uses TensorFlow/Keras. Experiments are configured through small YAML/JSON configs (exercise, feature set = basic/featured, window length = 30, stride = 10, learning rate, callbacks). Each run writes:

- a `metrics.json` with split metrics (MAE, $R^2$),
- per-exercise CSVs (`key_metrics_comparison.csv`, `feature_augmentation_impact.csv`),

- plots under `images/` using predictable paths (see Results & Appendices).

Random seeds are fixed where applicable (data shuffles, model init). We avoid rep segmentation; all learning/scoring is window-based (30 frames, stride 10).

## 6.3 Data I/O and File Layout

Raw videos reside in exercise-named folders. Processing creates:

- `exports/angles/{video}.csv` — per-frame angles,
- `exports/features/{video}.json` — windowed features (ROM/velocity/symmetry/DTW degradation),
- `results/` — experiment tables,
- `images/` — EDA and per-exercise figures used in the thesis.

This layout mirrors the thesis structure so each figure/table is generated once and directly referenced.

## 6.4 Pose and Angle Module

Videos are decoded at 30 FPS. MediaPipe Pose (VIDEO mode) returns 2D landmarks; angles are computed per frame using vector dot-product geometry (Section 4.2). To accommodate low-quality footage, thresholds are set to `0.1` for detection/presence; short gaps ($\leq$5 frames) are linearly interpolated; longer gaps mark frames invalid. Angles are smoothed (moving average, window 5) and saved.

## 6.5 Feature Engineering Module

From each joint time series we compute windowed features:

- **ROM**: max–min angle within the window (mobility/depth proxy),
- **Angular velocity**: central difference; mean/max/std per window (tempo & load proxies),

- **Symmetry dispersion**: std of L–R difference (asymmetry/compensation),
- **DTW degradation**: multi-channel DTW distance to the first valid window in the session (baseline).

Two feature sets are produced: *basic* (angles only) and *featured* (angles + engineered metrics). All learning features are MinMax scaled (fit on train only).

## 6.6 Windowing and Scaling

We extract fixed windows of $W$=30 frames (1.0 s), stride $S$=10 (0.33 s). For DTW degradation, signals are $z$-normalized per channel within the window to remove scale bias. For learning, we use a two-stage scheme: (i) persist MinMax-scaled arrays at dataset build time (fit on train, reused on val/test), then (ii) apply a *RobustScaler* to $X$ at load time (fit on train only) and range-scale $y$ to $[0, 1]$, per Section *Normalization*.

## 6.7 Degradation Scoring (DTW)

Per-window multi-channel DTW uses squared Euclidean local costs and a Sakoe–Chiba band (±3 frames) to ensure physiologically plausible alignments and $O(W\,b)$ complexity (Section 6.7). Session-level summaries report mean and 95th-percentile DTW degradation and mean symmetry dispersion.

## 6.8 Sequence Modeling Module

We use a compact configuration (**A1**): Conv1D(64) → GRU(128) → Dense(128) → Dense(1), dropout = [0.2, 0.3, 0.3], Adam (LR = $5 \times 10^{-3}$), Huber loss, batch size 32, max 100 epochs with EarlyStopping (patience 10), ReduceLROnPlateau (factor 0.5), and ModelCheckpoint (best val loss). Splits are exercise-specific. Outputs include predictions, residuals, MAE, and $R^2$, consolidated into `key_metrics_comparison.csv`.

## 6.9 Experiment Orchestration and Artifacts

A small experiment runner loads a config, builds datasets (basic/featured), trains the model, evaluates on validation/test, and writes:

- **Tables**: `key_metrics_comparison.csv`, `feature_augmentation_impact.csv`, `all_experiments_full_results.csv`.
- **Plots**: line comparisons, errors, loss curves, prediction scatters under `images/{exercise}/{basic|featured}/...`.
- **EDA**: per-exercise correlation and label distribution under `images/visualizations/{exercise_base}/...`.

This makes the thesis fully regenerable from the repository.

## 6.10 Performance and Resource Notes

Pose extraction is CPU-bound; modeling is GPU-accelerated (optional). Batched window extraction and vectorized DTW reduce latency. With 30 FPS inputs and stride 10, the system targets near-real-time window scoring; true real-time deployment would need lighter pose models or edge acceleration (future work).

## 6.11 Quality Assurance

We include lightweight checks:

- angle range sanity (0–180 deg) post-smoothing,
- window validity threshold ($\geq 70\%$ valid frames),
- split integrity (no shared windows across sets),
- metric invariants (non-negative ROM; DTW monotonicity vs. noise).

Failures are logged and the affected windows/videos are excluded to prevent biased metrics.

## 6.12 Visual and Analytical Feedback

Figures are produced per exercise and per feature set to mirror the Results chapter:

- **EDA panels:** correlation heatmaps and label distributions (base features).
- **Model fit panels:** line comparisons (pred vs. target), residual/error plots, loss curves, and prediction scatters for *basic* vs. *featured*.
- **Comparative charts:** aggregated MAE/$R^2$ bars.

All figure paths correspond one-to-one with the references in the thesis, avoiding manual asset duplication.

# Chapter 7

# Results and Evaluation

## 7.1 Evaluation Objectives

We assess:

- Stability and interpretability of DTW-based *degradation* scoring.
- LSTM performance for movement-quality prediction (regression).

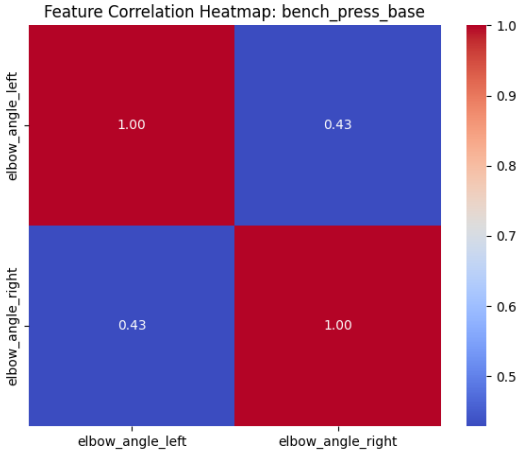
### 7.1.1 Model Architecture

We use the A1 configuration (Section 3.2).

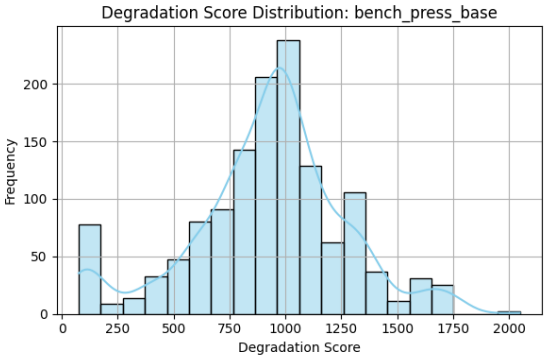
## 7.2 Exploratory Visualizations (EDA)

### 7.2.1 Per-Exercise Feature Structure (Base feature sets)

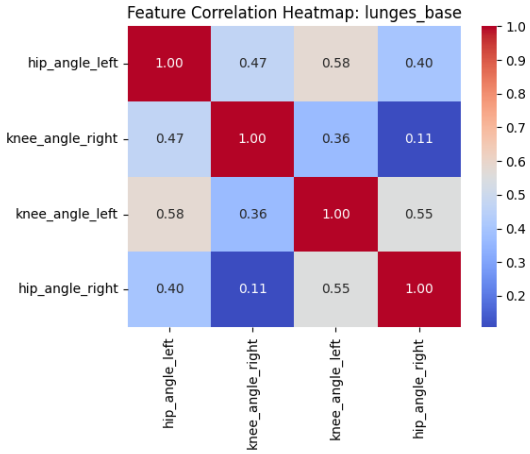
*Each pair shows the feature correlation structure and the label distribution for the base (angles-only) dataset of that exercise.*



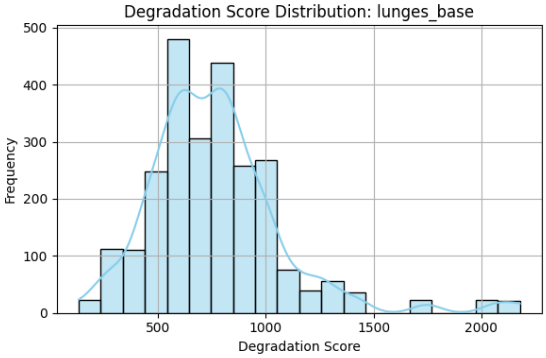

(a) Bench press — correlation




(b) Bench press — label distribution

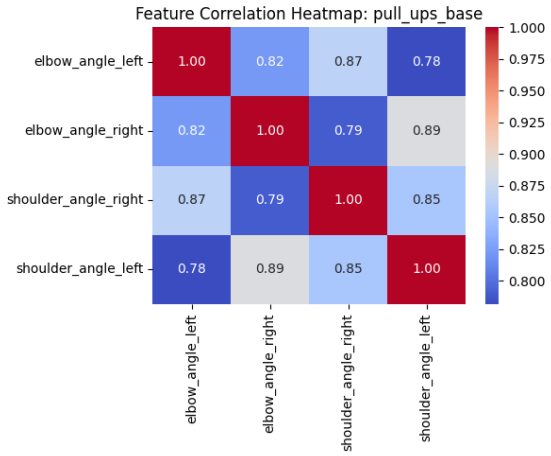



(c) Lunges — correlation

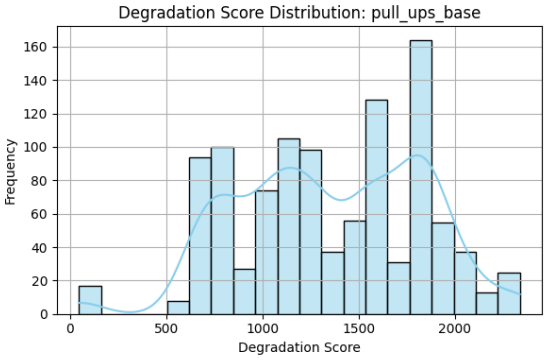



(d) Lunges — label distribution

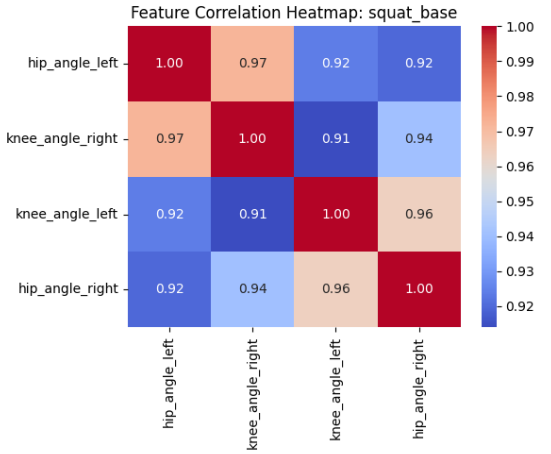
Figure 7.1: EDA (base features): Bench press and Lunges.



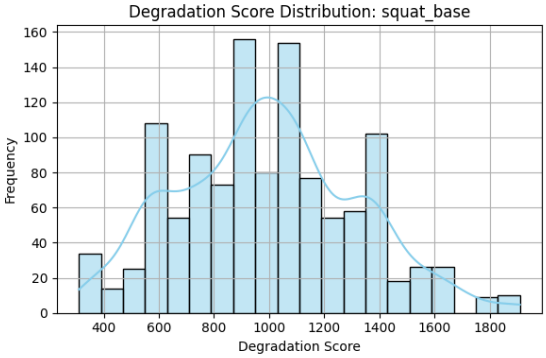

(a) Pull-ups — correlation




(b) Pull-ups — label distribution




(c) Squat — correlation

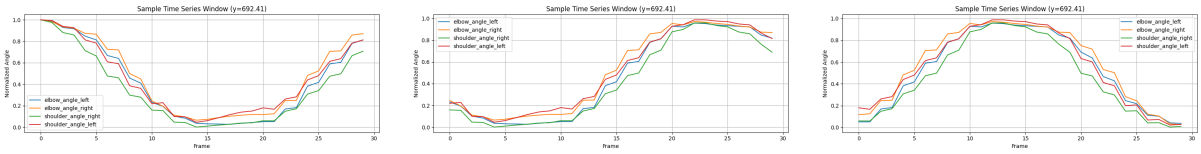



(d) Squat — label distribution

Figure 7.2: EDA (base features): Pull-ups and Squat.

### 7.2.2 Window Sampling (Pull-Ups)

*Thirty-frame (1.0 s at 30 FPS) windows with stride 10 (∼0.33 s) illustrate typical within-window dynamics for elbow/shoulder angles.*




(a) Window A: early phase (b) Window B: mid-ascent (c) Window C: near peak

Figure 7.3: Sample pull-ups windows showing normalized elbow/shoulder angles over 30 frames.

## 7.3 Sequence Model Evaluation (LSTM/GRU)

Sequence models were trained on windowed joint-angle features with two-stage normalization (dataset-time MinMax; train-time RobustScaler on $X$ and range scaling on $y$) and windowing (30 frames, stride 10; 30 FPS).

### 7.3.1 Performance Summary (by exercise and feature set)

*Source:* `key_metrics_comparison.csv`

| Exercise | $R^2$ (basic) | $R^2$ (featured) | MAE (basic) | MAE (featured) |
|---|---|---|---|---|
| bench_press | 0.560 | 0.453 | 0.086 | 0.099 |
| lunges | 0.612 | 0.505 | 0.067 | 0.074 |
| pull_ups | 0.612 | 0.657 | 0.092 | 0.087 |
| squat | 0.703 | 0.624 | 0.085 | 0.092 |

Table 7.1: Prediction performance by exercise and feature set. Basic: angles only. Featured: angles + engineered features (ROM, angular velocity, symmetry, degradation).

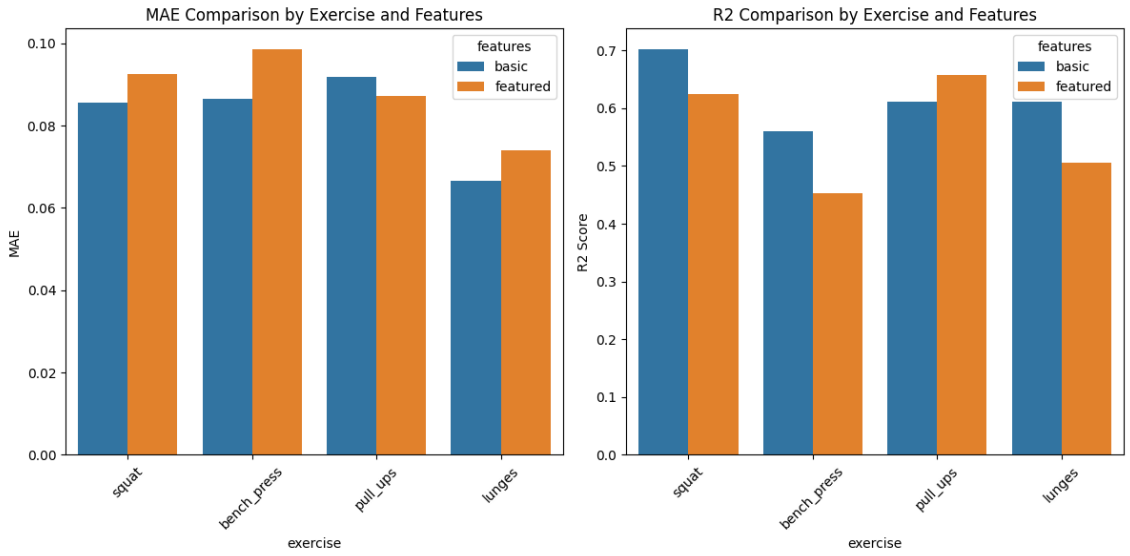



Figure 7.4: MAE and $R^2$ comparison across exercises for basic vs. featured feature sets.

### 7.3.2 Feature Augmentation Impact

*Source:* `feature_augmentation_impact.csv`

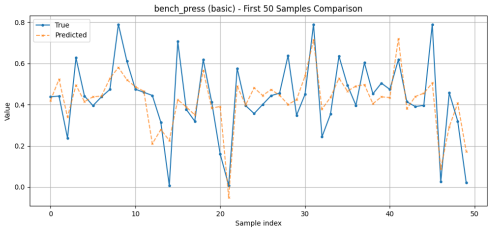
| **Exercise** | $\Delta$**R²** (featured − basic) | $\Delta$**MAE** (featured − basic) |
|---|---|---|
| bench_press | -0.108 | +0.012 |
| lunges | -0.107 | +0.007 |
| pull_ups | +0.045 | -0.005 |
| squat | -0.079 | +0.007 |

Table 7.2: Impact of feature augmentation by exercise. Positive $\Delta R^2$ and negative $\Delta$MAE indicate improvement.

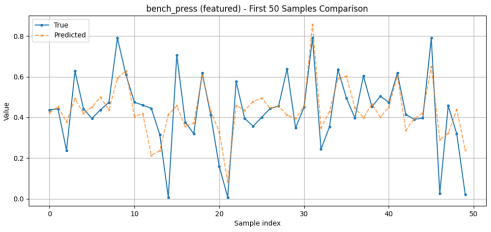
## 7.4 Per-Exercise Results (Predictions vs. Targets)

*Each row shows basic (left) vs. featured (right) line-comparison plots for the same exercise.*

## Bench Press



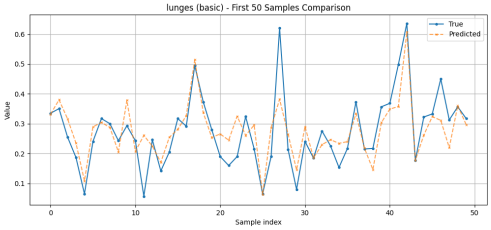

(a) Basic



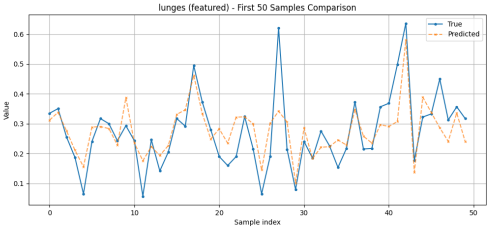

(b) Featured

Figure 7.5: Bench press: prediction vs. target (side-by-side).

## Lunges



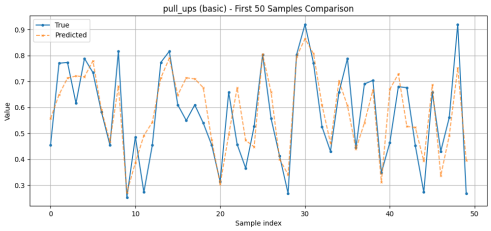

(a) Basic



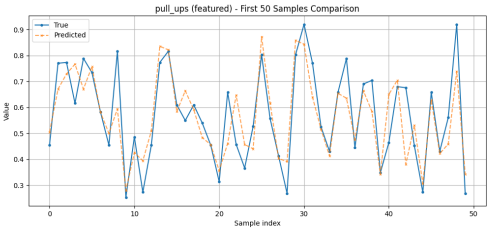

(b) Featured

Figure 7.6: Lunges: prediction vs. target (side-by-side).

## Pull-Ups



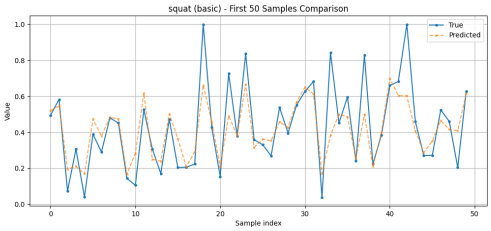

(a) Basic



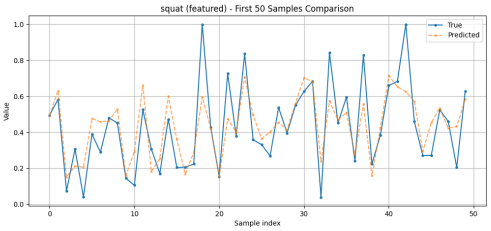

(b) Featured

Figure 7.7: Pull-ups: prediction vs. target (side-by-side).

## Squat




(a) Basic




(b) Featured

Figure 7.8: Squat: prediction vs. target (side-by-side).

# Chapter 8

# Discussion and Limitations

## 8.1 Interpretation of Results

The DTW-based window scoring captured timing-invariant deviations in execution, providing a robust degradation signal even when tempo varied within sessions. Sequence models trained on angle windows achieved consistent performance across exercises, with basic features (angles only) competitive or superior in three of four exercises. Specifically (Table 7.1):

- **Squat:** best $R^2$=0.703 (basic), MAE = 0.085.
- **Lunges:** $R^2$=0.612 (basic), MAE = 0.067.
- **Bench press:** $R^2$=0.560 (basic), MAE = 0.086.
- **Pull-ups:** the sole case where *featured* improved performance ($\Delta R^2$=+0.045; $\Delta$MAE = −0.005).

These patterns suggest that engineered features can help when the primary angles underspecify movement (upper-body pulling with complex scapular/shoulder mechanics), but may inject noise when derived from unstable landmarks (e.g., ROM/velocity computed from low-quality 2D keypoints).

**Why did basic outperform featured in 3/4 exercises?** Two practical factors likely dominated: (i) low detection thresholds (0.1/0.1) retained more data but increased jitter, which propagates and amplifies in derivatives (velocity) and ranges (ROM); (ii)

featured sets introduced correlated signals that the model may overfit when sample sizes are modest. The pull-ups exception indicates potential value for engineered features when the kinematics genuinely benefit from richer descriptors.

## 8.2 Use Case Implications

### 8.2.1 Sports Medicine / Physio

Window-level DTW and symmetry dispersion provide quantitative markers for *return-to-play* progress without rep segmentation. Clinicians can track (i) reduced DTW degradation over sessions (technique stabilization), (ii) lower symmetry dispersion (compensation resolving), and (iii) ROM/velocity trends (mobility and tempo control).

### 8.2.2 Coaching Utility (Technique Practice)

For practice scenarios, line-comparison plots and error panels surface temporal regions where execution deviates from the baseline template. Because metrics are window-based, feedback is robust to variable cadence and partial reps.

## 8.3 Limitations

- **2D Pose Sensing:** Depth ambiguities (e.g., torso pitch, scapular motion) cannot be disambiguated from single-view 2D landmarks; lighting/occlusion effects are pronounced in low-quality footage.
- **Low Thresholds & Noise:** Operating MediaPipe at 0.1/0.1 was necessary to avoid data loss but increases landmark noise; derived features (velocity/ROM) are especially sensitive.
- **Dataset Scale & Balance:** Limited and uneven per-exercise counts constrain generalization and hyperparameter tuning; featured models may overfit exercise-specific idiosyncrasies.
- **Split Integrity:** If windows from the same original clip are not grouped by session in the same split, leakage can inflate metrics. (We designed splits to avoid this; any

deviation would be a threat to validity.)

- **Model Interpretability:** While DTW is interpretable, LSTM/GRU internal states are opaque; attention maps or saliency on time steps/joints would aid clinical trust (future work).
- **Real-Time Constraints:** Current pipeline is batch-oriented; live use would require faster landmarkers and on-device inference with careful latency budgeting.

## 8.4 Recommendations

- Prefer angles-only (basic) features when video quality is low; introduce engineered features selectively (e.g., upper-body pulling) and regularize strongly.
- Use multi-angle, well-lit recordings to stabilize landmarks; adopt subject-consistent camera placement.
- Consider robust derivatives (Savitzky–Golay) and clipped ROM to reduce outlier sensitivity.
- Explore 3D keypoints or multi-view triangulation for depth-critical assessments (future work).
- Add explainability overlays (per-window attention on joint channels) to support clinical adoption.

# Chapter 9

# Conclusion

This work delivers a physiotherapy-focused, pose-based pipeline for movement-quality analysis in football. From single-view videos, we extract joint-angle time series, score windowed execution quality with DTW, and train sequence models to predict movement-quality trends. The approach yields consistent performance across four football-adjacent exercises and surfaces clinically interpretable markers—mobility (ROM), tempo/load (angular velocity), and compensation (symmetry dispersion)—without relying on explicit rep segmentation. In three of four exercises, basic (angles-only) features performed best, highlighting the sensitivity of derivative features to landmark noise under low-quality footage; pull-ups were the notable exception where richer features improved accuracy. Together, these results demonstrate the feasibility of objective, window-level feedback for return-to-play monitoring and technique practice.

## Future Work

- **3D pose & multi-view capture:** Incorporate 3D keypoints (e.g., BlazePose 3D, VIBE) or calibrated multi-camera setups to resolve depth ambiguities (torso pitch, scapular motion).
- **Data scale and diversity:** Expand to larger, balanced cohorts with standardized, well-lit recordings and subject-consistent camera placement; include in-the-wild sessions.
- **Real-time delivery:** Optimize the inference stack (lighter landmarkers, quantized

models, batching) for on-device or edge deployment with clinician UI.

- **Technique analysis for football skills:** Extend from rehab exercises to football-specific actions (e.g., free kicks), using exemplar templates and per-phase scoring.
- **Prospective reporting layer:** Add a natural-language interface that turns window-level metrics into concise practitioner notes (VLMs considered as an optional component, subject to domain adaptation and governance).
- **Explainability:** Add time-step/joint attention maps or saliency to increase trust and aid clinical interpretation of LSTM predictions.

# Bibliography

Huber, P. J. (1964). Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35(1):73–101.

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. In *IEEE Trans. Acoustics, Speech, and Signal Processing*, volume 26, pages 43–49.

Salvador, S. and Chan, P. (2007). Toward accurate dynamic time warping in linear time and space. In *Int'l Conf. on KDD Workshop on Mining Temporal and Sequential Data*.

# Appendix A

# Appendix A: Code Snapshots

## Overview

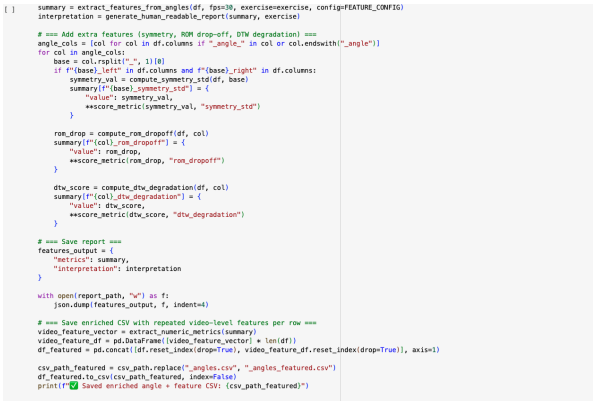
This appendix presents curated screenshots of the most important code from the notebooks, arranged in the same order as the pipeline (pose estimation → preprocessing/feature engineering → modeling/training → reporting). All code uses low detection thresholds (`0.1/0.1`) to cope with challenging video quality.

# Pose Estimation (MediaPipe)


(a) Pose landmarker options (VIDEO mode, thresholds, annotators).


(b) Video capture, frame loop, time–based detection.


(c) Landmark indexing and joint triplets for angle computation.




(d) CSV/JSON write-out and (optional) annotated frame writer.

Figure A.1: Pose estimation pipeline.

# Data Loading & Feature Engineering

```python
[ ] # === Load & Normalize CSV ===
    def load_and_window_csv(csv_path: str, window_size: int, stride: int) -> np.ndarray:
        df = pd.read_csv(csv_path)
        df = df.drop(columns=["timestamp_ms"], errors="ignore")

        scaler = MinMaxScaler()
        normalized = scaler.fit_transform(df)

        windows = []
        for i in range(0, len(normalized) - window_size + 1, stride):
            windows.append(normalized[i:i + window_size])

        return np.array(windows), df.columns.tolist()
```

```python
[ ] # === Extract Degradation Label from JSON ===
    def extract_degradation_score(json_path: str, joints: List[str]) -> float:
        with open(json_path, "r") as f:
            data = json.load(f)
        try:
            metrics = data["metrics"]
            scores = [
                metrics[f"{joint}_dtw_degradation"]["value"]
                for joint in joints if f"{joint}_dtw_degradation" in metrics
            ]
            return float(np.mean(scores))
        except KeyError:
            return np.nan
```

(a) `load_and_window_csv` + `extract_degradation_score` (DTW).

```python
def extract_features_from_angles(df: pd.DataFrame, fps: float, exercise: str, config: dict) -> dict:
    """
    Extract ROM and angular velocity features from angle CSV.

    Args:
        df (pd.DataFrame): DataFrame with angle columns.
        fps (float): Video frame rate.
        exercise (str): Exercise name (e.g. "squat").
        config (dict): FEATURE_CONFIG dict.

    Returns:
        dict: Summary of extracted features.
    """
    summary = {}
    features = config[exercise]["features"]

    for feature_name, meta in features.items():
        base_name = feature_name.replace("_velocity", "")
        sides = meta.get("sides", ["left", "right"])
        feat_type = meta["type"]

        for side in sides:
            col = f"{base_name}_{side}"
            if col not in df.columns:
                continue

            if feat_type == "rom":
                summary[f"{col}_rom"] = compute_rom(df[col])

            elif feat_type == "velocity":
                vel_series = compute_velocity(df[col], fps)
                summary[f"{col}_velocity_mean"] = float(vel_series.mean())
                summary[f"{col}_velocity_max"] = float(vel_series.max())
                summary[f"{col}_velocity_std"] = float(vel_series.std())

    return summary
```

(b) ROM & angular velocity features.

```python
def compute_dtw_degradation(df: pd.DataFrame, col: str, window_size=30, stride=15) -> float:
    """
    Compares all windows to the first using DTW and returns average DTW distance.
    Fixes input shape to (T, 1) for univariate DTW.
    """
    def ensure_scalar(x):
        if isinstance(x, (np.ndarray, list)):
            return float(np.array(x).flatten()[0])
        return float(x)

    series = df[col].dropna().apply(ensure_scalar).to_numpy()

    windows = [
        series[i:i + window_size]
        for i in range(0, len(series) - window_size + 1, stride)
    ]

    if len(windows) < 2:
        return np.nan

    baseline = windows[0]
    distances = []
    for idx, w in enumerate(windows[1:], start=1):
        if len(w) != len(baseline):
            continue
        try:
            # Reshape to (T, 1) so each timestep is a 1D vector
            d, _ = fastdtw(w.reshape(-1, 1), baseline.reshape(-1, 1), dist=euclidean)
            distances.append(d)
        except Exception as e:
            print(f"❌ Error in DTW comparison (window {idx}): {e}")

    return float(np.mean(distances)) if distances else np.nan
```

(c) DTW degradation (baseline vs. subsequent windows).

```python
def process_video_folder(video_folder: str, exercise: str, use_featured_csv: bool = False) -> Tuple[np.ndarray, np.ndarray, List[str]]:
    suffix = "_angles_featured.csv" if use_featured_csv else "_angles.csv"
    angle_files = glob.glob(os.path.join(video_folder, f"*{suffix}"))
    json_files = glob.glob(os.path.join(video_folder, "*_features_metadata.json"))

    if not angle_files or not json_files:
        print(f"⚠️ Skipping {video_folder}: missing {suffix} or JSON.")
        return np.empty((0,)), np.empty((0,)), []

    csv_path = angle_files[0]
    json_path = json_files[0]
    joints = EXERCISE_JOINTS.get(exercise)

    if not joints:
        print(f"⚠️ No joint config for exercise: {exercise}")
        return np.empty((0,)), np.empty((0,)), []

    try:
        X, used_features = load_and_window_csv(csv_path, WINDOW_SIZE, STRIDE)
        y_val = extract_degradation_score(json_path, joints)
        if np.isnan(y_val):
            print(f"⚠️ Skipping (csv_path): invalid score.")
            return np.empty((0,)), np.empty((0,)), []

        y = np.full((X.shape[0],), y_val)
        return X, y, used_features
    except Exception as e:
        print(f"❌ Error in {video_folder}: {e}")
        return np.empty((0,)), np.empty((0,)), []
```

(d) Per-video extractor: read CSV/JSON, build $X$, derive $y$.

Figure A.2: Data preparation and feature engineering.

# Model Architecture & Training

```python
[ ] def build_enhanced_cnn(input_shape):
        """Enhanced model with more regularization"""
        inputs = Input(shape=input_shape)
        conv = Conv1D(64, kernel_size=3, activation='relu', padding='same')(inputs)
        conv = BatchNormalization()(conv)
        conv = Dropout(0.2)(conv)

        gru = GRU(128, return_sequences=False)(conv)
        gru = BatchNormalization()(gru)
        gru = Dropout(0.3)(gru)

        dense = Dense(128, activation='relu')(gru)
        dense = Dropout(0.2)(dense)
        output = Dense(1)(dense)

        return Model(inputs=inputs, outputs=output)
```

(a) Model: Conv1D → GRU → Dense(1).

```python
def train_enhanced_model(X, y, exercise_name, augment_flag):
    """Training with proper validation set handling"""
    # Create exercise-specific results directory
    feature_type = "featured" if augment_flag else "basic"
    exercise_dir = os.path.join(RESULTS_DIR, exercise_name, feature_type)
    os.makedirs(exercise_dir, exist_ok=True)

    # Stratified split
    y_bins = np.digitize(y, bins=np.quantile(y, [0.25, 0.5, 0.75]))
    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, stratify=y_bins, random_state=42
    )

    model = build_enhanced_cnn(X.shape[1:])

    # Experiment parameters
    params = {
        'exercise': exercise_name,
        'features': feature_type,
        'learning_rate': 0.001,
        'loss': 'huber',
        'batch_size': 32,
        'epochs': 100,
        'dropout_rates': [0.2, 0.3, 0.3],
        'architecture': 'Conv1D(64)→GRU(128)→Dense(128)',
        'early_stopping_patience': 20,
        'reduce_lr_patience': 10,
        'input_shape': X.shape[1:]
    }

    model.compile(
        optimizer=Adam(learning_rate=params['learning_rate']),
        loss=params['loss'],
        metrics=['mae']
    )

    callbacks = [
        EarlyStopping(patience=params['early_stopping_patience'], restore_best_weights=True),
        ReduceLROnPlateau(factor=0.5, patience=params['reduce_lr_patience'], min_lr=1e-6)
    ]

    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=params['epochs'],
        batch_size=params['batch_size'],
        callbacks=callbacks,
        verbose=1
    )

    return model, history, X_val, y_val, params
```

(b) Training routine: params, compile, callbacks, fit.

```python
def run_experiment(exercise, augment_flag):
    """Run full experiment for one exercise with specified feature type"""
    try:
        feature_type = "featured" if augment_flag else "basic"
        print(f"\n{'='*60}")
        print(f"Starting experiment for: {exercise} ({feature_type} features)")
        print(f"{'='*60}")

        # Load data
        X, y = load_and_preprocess_data(exercise, augment_with_features=augment_flag)

        # Train model
        model, history, X_val, y_val, params = train_enhanced_model(
            X, y, exercise, augment_flag
        )

        # Predict and evaluate
        y_pred = model.predict(X_val)

        # Debug shapes
        print(f"Shapes - y_val: {y_val.shape}, y_pred: {y_pred.shape}")

        # Flatten arrays
        y_val_flat = y_val.flatten()
        y_pred_flat = y_pred.flatten()

        # Save results
        results = save_enhanced_results(
            history, y_val_flat, y_pred_flat, params, exercise_dir
        )

        # Print summary
        print(f"\n✅ Completed experiment for: {exercise} ({feature_type} features)")
        print(f"Final Validation MAE: {results['validation_mae']:.4f}")
        print(f"Final Validation R2: {results['validation_r2']:.4f}")
        print(f"Results saved to: {exercise_dir}")

        return results

    except Exception as e:
        print(f"\n❌ Error occurred for {exercise} ({'featured' if augment_flag else 'basic'}): {str(e)}")
        if 'y_pred' in locals():
            print(f"y_pred shape: {y_pred.shape}")
        if 'y_val' in locals():
            print(f"y_val shape: {y_val.shape}")
        return None
```

(c) Runner: load → train → predict → save.

Figure A.3: Model, training loop, and orchestration.

# Results Generation

```python
def generate_comparison_tables(all_results):
    """Generate comparison tables and visualizations"""
    if not all_results:
        return

    df = pd.DataFrame(all_results)

    # Save full results
    full_results_path = os.path.join(RESULTS_DIR, "all_experiments_full_results.csv")
    df.to_csv(full_results_path, index=False)

    # Create comparison tables
    comparison_dir = os.path.join(RESULTS_DIR, "comparisons")
    os.makedirs(comparison_dir, exist_ok=True)

    # 1. Basic metrics comparison
    metric_cols = ['exercise', 'features', 'prediction_mae', 'prediction_r2',
                   'num_epochs_run', 'final_val_loss']
    metrics_df = df[metric_cols].sort_values(['exercise', 'features'])
    metrics_path = os.path.join(comparison_dir, "key_metrics_comparison.csv")
    metrics_df.to_csv(metrics_path, index=False)

    # 2. Feature augmentation impact
    feat_impact = df.pivot_table(
        index='exercise',
        columns='features',
        values=['prediction_mae', 'prediction_r2'],
        aggfunc='mean'
    )
    feat_impact_path = os.path.join(comparison_dir, "feature_augmentation_impact.csv")
    feat_impact.to_csv(feat_impact_path)

    # 3. Create visual comparison plots
    plt.figure(figsize=(12, 6))

    # MAE comparison
    plt.subplot(1, 2, 1)
    sns.barplot(data=df, x='exercise', y='prediction_mae', hue='features')
    plt.title('MAE Comparison by Exercise and Features')
    plt.ylabel('MAE')
    plt.xticks(rotation=45)

    # R2 comparison
    plt.subplot(1, 2, 2)
    sns.barplot(data=df, x='exercise', y='prediction_r2', hue='features')
    plt.title('R2 Comparison by Exercise and Features')
    plt.ylabel('R2 Score')
    plt.xticks(rotation=45)

    plt.tight_layout()
    plot_path = os.path.join(comparison_dir, "metrics_comparison.png")
    plt.savefig(plot_path)
    plt.close()

    print(f"\nComparison tables and plots saved to: {comparison_dir}")
```

Figure A.4: Exports: metrics tables and comparison plots.

# Appendix B

# Appendix B: Dataset Samples

## Overview

This appendix shows representative raw and pose-annotated frames for each exercise used in the study. All experiments used 30 FPS input, 30-frame windows (1.0 s), and stride 10. In practice, video quality (low resolution, backlighting, motion blur, and occlusion) substantially affected landmark stability and downstream features. To avoid discarding a large fraction of frames/videos, we set `min_pose_detection_confidence = 0.1` and `min_pose_presence_confidence = 0.1`. We expect performance to improve with higher-quality, well-lit, fronto-lateral recordings.

## Squat

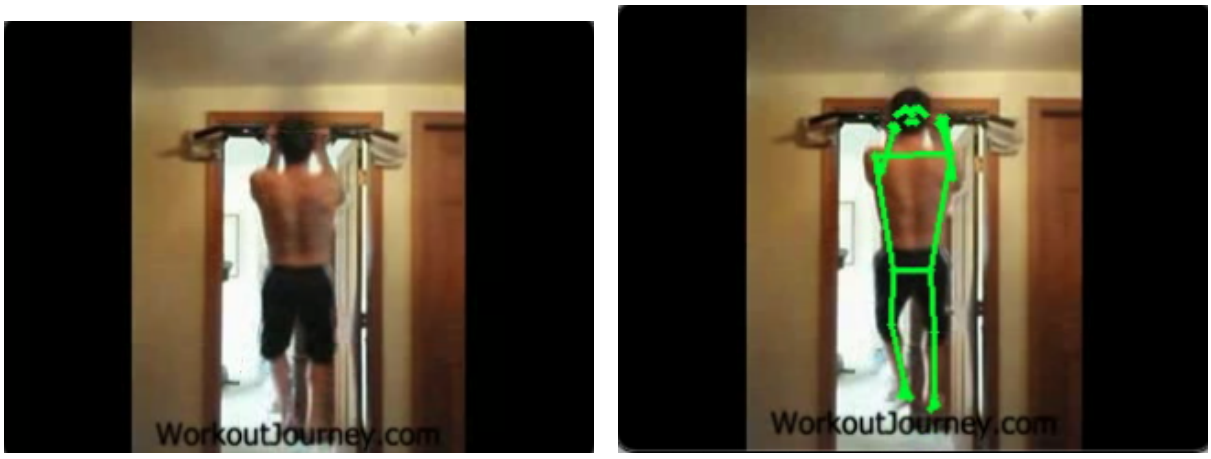
(a) Raw frame (b) Pose-annotated frame

Figure B.1: Raw (left) vs. overlay (right). Deep knee flexion; backlighting and shadowing cause partial occlusion.

**Joints scored:** `hip_angle_left`, `knee_angle_left`, `hip_angle_right`, `knee_angle_right`.

**Video-quality note:** Low-resolution outdoor footage with harsh lighting created landmark dropouts around the knees and ankles; thresholds were lowered to retain data, at the cost of noisier angles.

## Pull-Ups




(a) Raw frame (b) Pose-annotated frame

Figure B.2: Raw (left) vs. overlay (right). Back view emphasizes elbow/shoulder motion; doorway creates contrast issues.

**Joints scored:** `elbow_angle_left`, `elbow_angle_right`, `shoulder_angle_left`, `shoulder_angle_right`.

**Video-quality note:** Camera distance and back view reduced wrist visibility; intermittent forearm

occlusion increased shoulder-angle jitter.

## Lunge

(a) Raw frame (b) Pose-annotated frame

Figure B.3: Raw (left) vs. overlay (right). Forward knee and hip depth captured for ROM assessment.

**Joints scored:** `hip_angle_left`, `knee_angle_left`, `hip_angle_right`, `knee_angle_right`.

**Video-quality note:** Indoor gym lighting, fast motion, and clothing folds occasionally obscured knee/hip landmarks, increasing ROM variance across windows.

## Bench Press

(a) Raw frame (b) Pose-annotated frame

Figure B.4: Raw (left) vs. overlay (right). Perspective makes elbow angle estimation sensitive to barbell occlusion.

**Joints scored:** `elbow_angle_left`, `elbow_angle_right`.

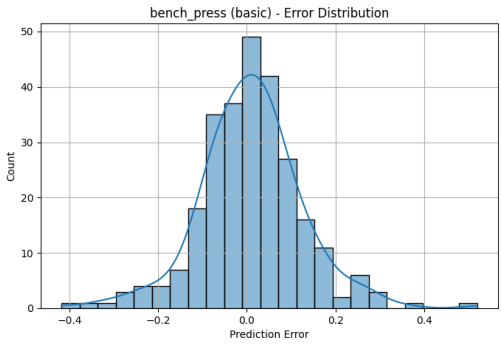
**Video-quality note:** Lying posture, spotter occlusion, and bench angle caused intermittent wrist/elbow keypoint loss; this added noise to velocity/ROM features.

# Appendix C

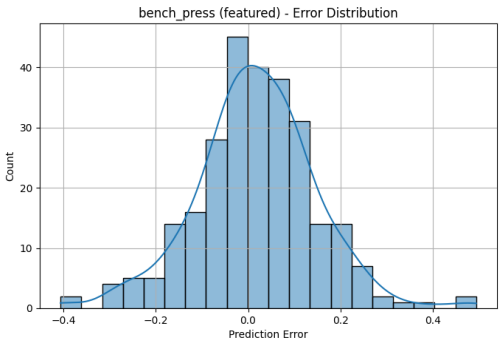
# Appendix C: Visualizations and Full Results

## Per-Exercise Result Galleries

### Bench Press

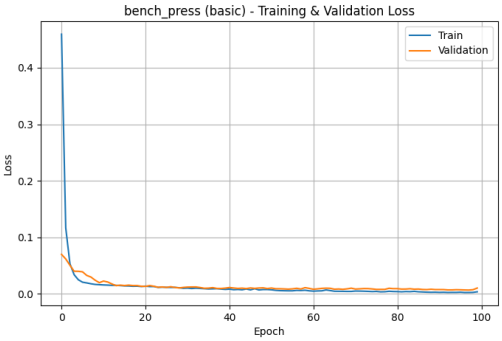



(a) Errors (basic)

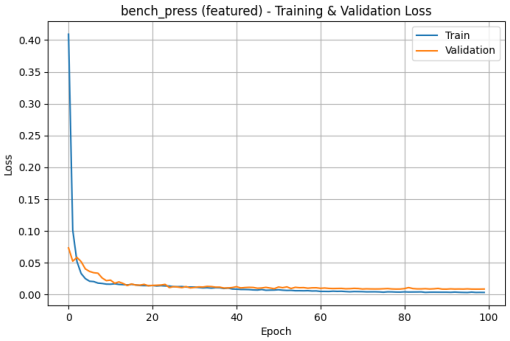



(b) Errors (featured)

Figure C.1: Bench press: error plots (side-by-side).

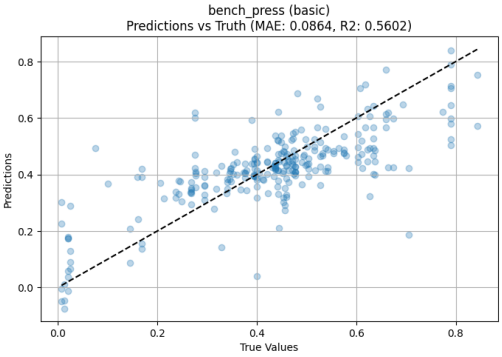



(a) Loss curves (basic)

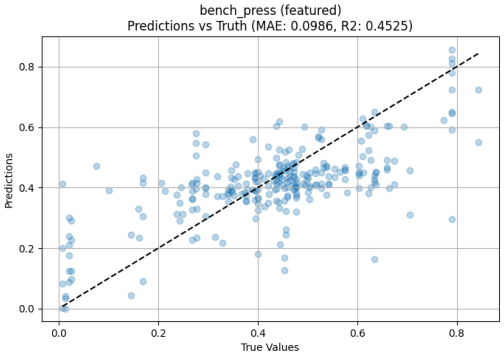



(b) Loss curves (featured)

Figure C.2: Bench press: training/validation loss curves.

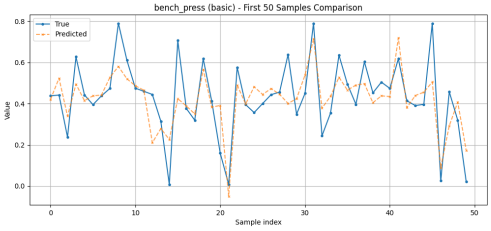



(a) Scatter (basic)

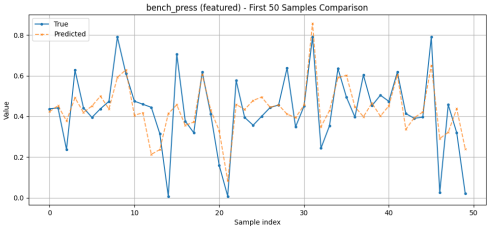



(b) Scatter (featured)

Figure C.3: Bench press: prediction scatter plots.



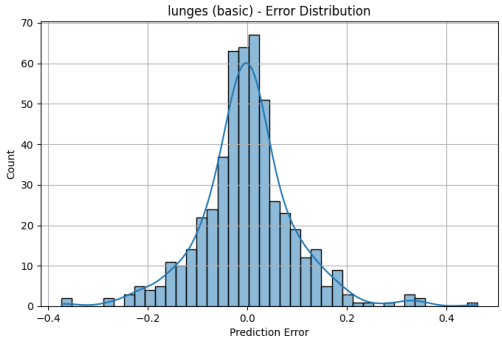

(a) Line comparison (basic)



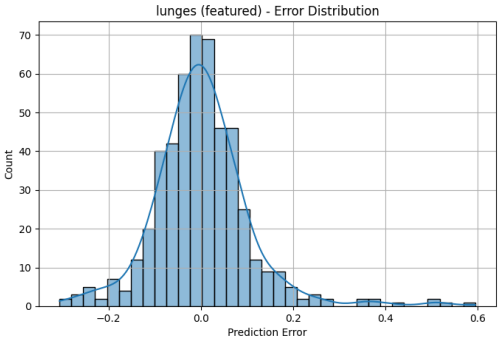

(b) Line comparison (featured)

Figure C.4: Bench press: prediction vs. target (side-by-side).

## Lunges

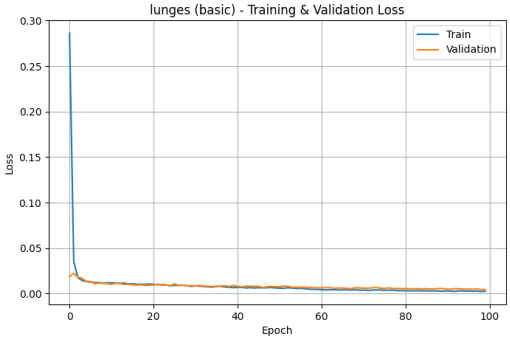



(a) Errors (basic)

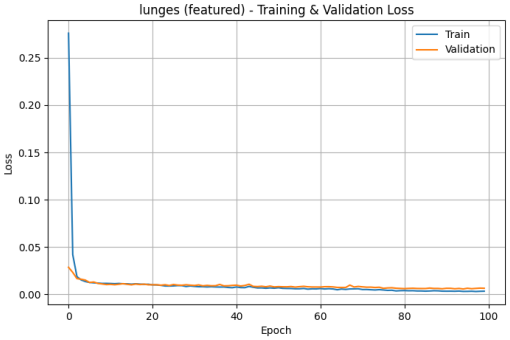



(b) Errors (featured)

Figure C.5: Lunges: error plots (side-by-side).

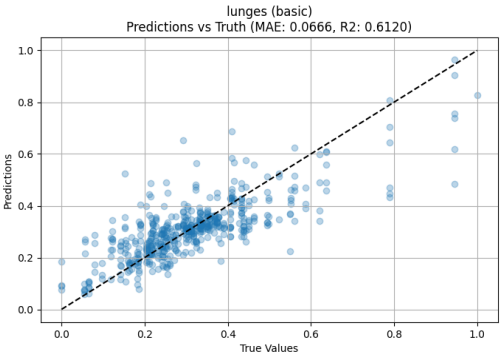



(a) Loss curves (basic)

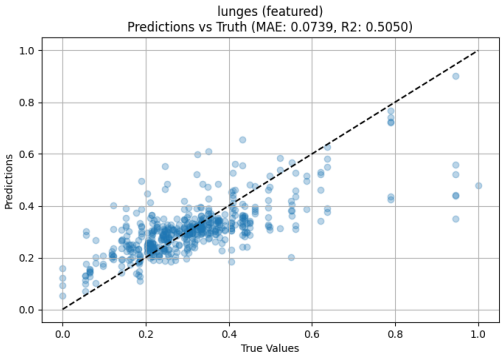



(b) Loss curves (featured)

Figure C.6: Lunges: training/validation loss curves.

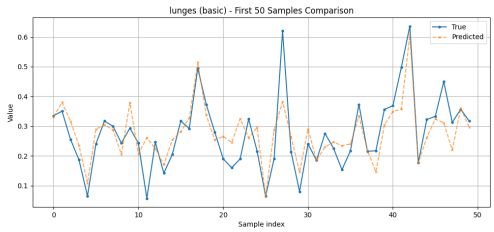



(a) Scatter (basic)

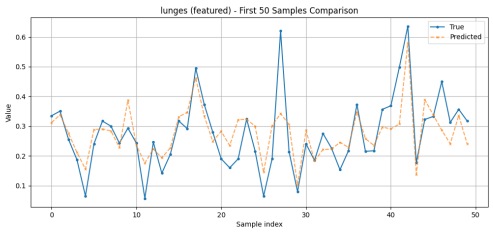



(b) Scatter (featured)

Figure C.7: Lunges: prediction scatter plots.



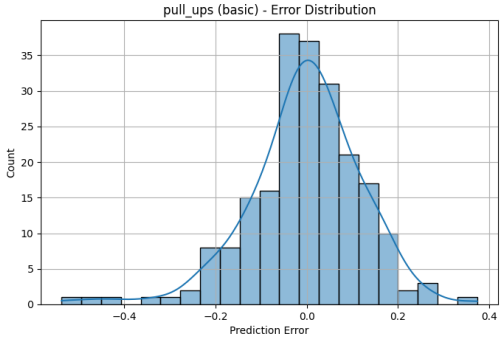

(a) Line comparison (basic)



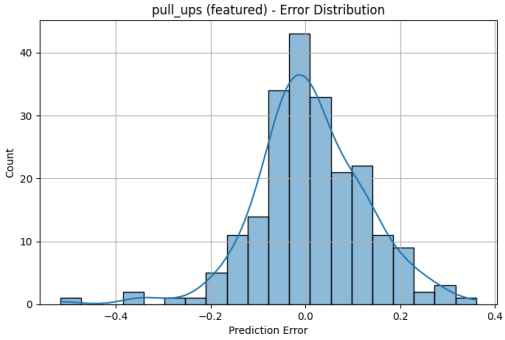

(b) Line comparison (featured)

Figure C.8: Lunges: prediction vs. target (side-by-side).

## Pull-Ups

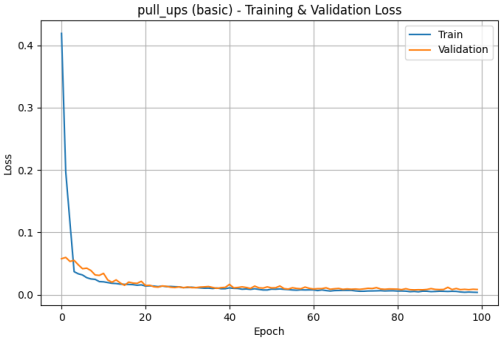



(a) Errors (basic)

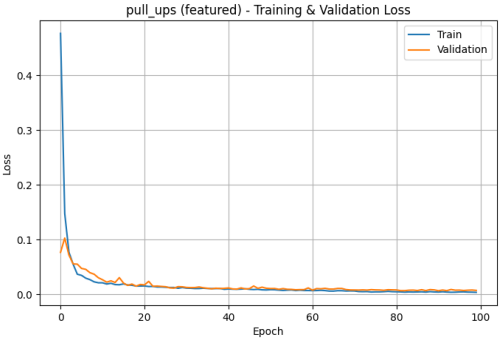



(b) Errors (featured)

Figure C.9: Pull-ups: error plots (side-by-side).

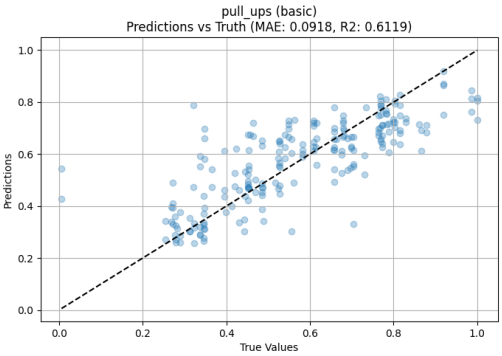



(a) Loss curves (basic)

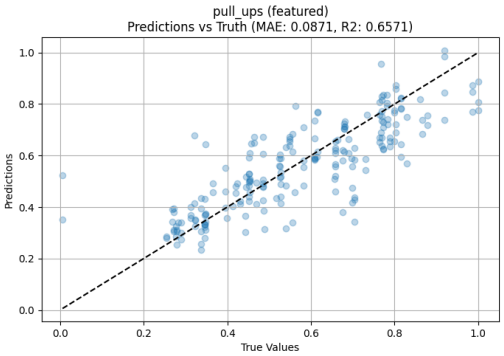



(b) Loss curves (featured)

Figure C.10: Pull-ups: training/validation loss curves.

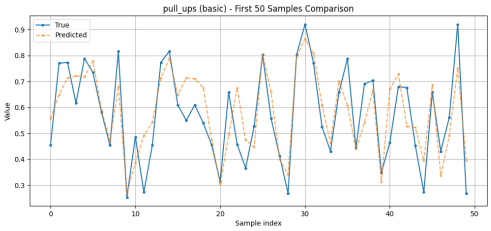



(a) Scatter (basic)

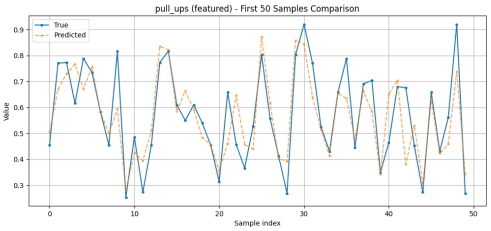



(b) Scatter (featured)

Figure C.11: Pull-ups: prediction scatter plots.



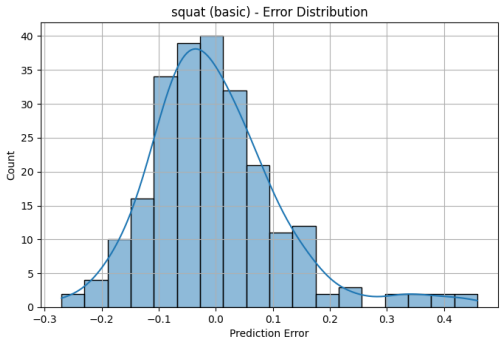

(a) Line comparison (basic)



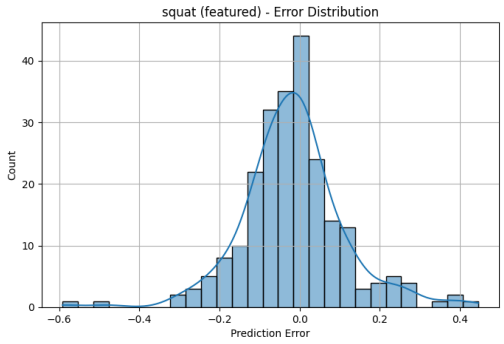

(b) Line comparison (featured)

Figure C.12: Pull-ups: prediction vs. target (side-by-side).

## Squat

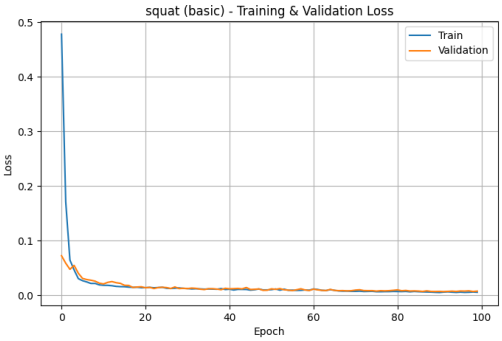



(a) Errors (basic)

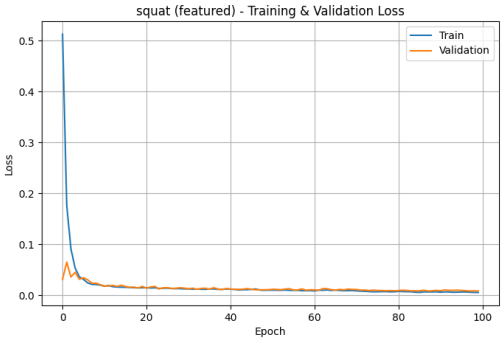



(b) Errors (featured)

Figure C.13: Squat: error plots (side-by-side).

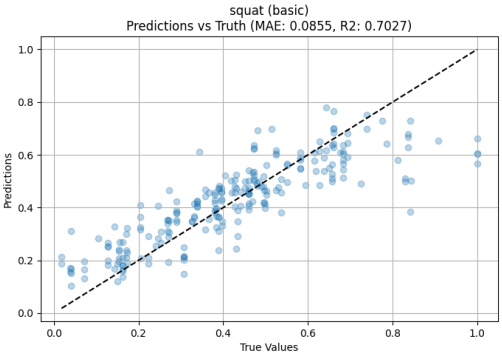



(a) Loss curves (basic)

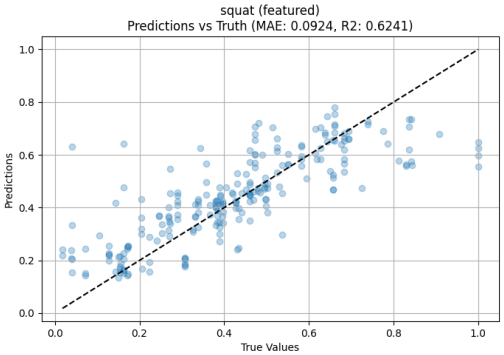



(b) Loss curves (featured)

Figure C.14: Squat: training/validation loss curves.

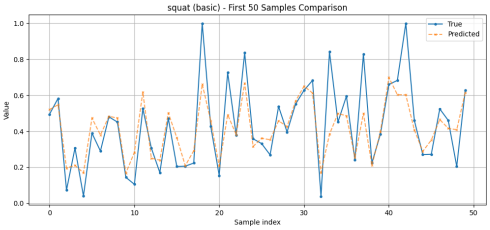



(a) Scatter (basic)

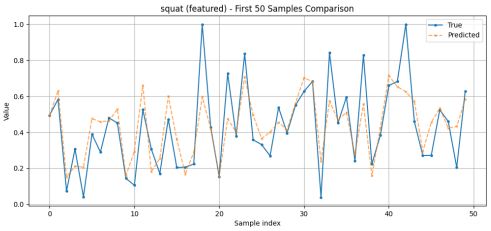



(b) Scatter (featured)

Figure C.15: Squat: prediction scatter plots.




(a) Line comparison (basic)




(b) Line comparison (featured)

Figure C.16: Squat: prediction vs. target (side-by-side).

# All Experiments: Compact Summary

| Exercise | Features | Arch | MAE | $R^2$ | Best epoch |
|---|---|---|---|---|---|
| bench_press | basic | A1 | 0.086 | 0.560 | 98.000 |
| bench_press | featured | A1 | 0.099 | 0.453 | 91.000 |
| lunges | basic | A1 | 0.067 | 0.612 | 99.000 |
| lunges | featured | A1 | 0.074 | 0.505 | 95.000 |
| pull_ups | basic | A1 | 0.092 | 0.612 | 85.000 |
| pull_ups | featured | A1 | 0.087 | 0.657 | 91.000 |
| squat | basic | A1 | 0.085 | 0.703 | 92.000 |
| squat | featured | A1 | 0.092 | 0.624 | 88.000 |

Table C.1: Compact outcomes with architecture code A1. The first column is stretchable (X), eliminating right-side whitespace.

**A1:** Conv1D(64) → GRU(128) → Dense(128); Adam, learning rate = 0.005; Huber loss; batch size = 32; epochs (max) = 100; dropout = [0.2, 0.3, 0.3].