Verification and Validation Report: FitCoachAR

Seyedmohamad Mirhosseininejad

April 20, 2026

Revision History

Date	Version	Notes
2026-04-08	1.0	Initial VnV Report (System + Unit + Reflection)
2026-04-15	1.1	Final Documentation release for CAS 741
2026-04-20	1.2	Addressed Yibing Mei feedback (issues #29, #30): disambiguated "MAE" vs. mean $\|\Delta\theta\|$; consolidated duplicate unit-test subsections.

1 Symbols, Abbreviations and Acronyms

Symbol	Description
T	Test case identifier (as defined in VnV Plan)
FSM	Finite State Machine
MAE	Mean Absolute Error
ROM	Range of Motion
SRS	Software Requirements Specification
VnV	Verification and Validation
M1–M8	Module identifiers as defined in the Module Guide
R1–R5	Functional requirements as defined in the SRS
NFR1, NFR2	Non-functional requirements as defined in the SRS
CI	Continuous Integration
ABC	Abstract Base Class
FPS	Frames Per Second

Contents

List of Tables

2 Introduction

This document reports the results of the Verification and Validation (VnV) activities conducted for FitCoachAR, a real-time exercise form analysis system that uses 2D computer vision to provide biomechanical feedback during Squat and Bicep Curl exercises.

The VnV activities follow the plan set out in *VnVPlan.tex* and are organized to trace each result back to the requirements in the SRS and the modules in the Module Guide. Section 3 evaluates functional requirements. Section 4 evaluates non-functional requirements. Section 5 compares FitCoachAR to the prior prototype. Section 6 describes the unit test results in detail. Section 7 documents deviations from the VnV Plan. Section 8 describes automated testing infrastructure. Sections 9 and 10 provide traceability matrices. Section 11 summarizes code coverage.

3 Functional Requirements Evaluation

This section evaluates the five functional requirements from the SRS. All automated tests were executed using `pytest` against `src/backend/tests/test_vnv.py`.

3.1 R1 — Accept Real-Time Video Input and Exercise Selection

R1 states: *The system shall accept real-time video input from a webcam, and the user-selected exercise type.*

This requirement is architectural. It is satisfied by the WebSocket consumer (`api/consumers.py`), which receives Base64-encoded video frames from the browser, decodes them via M3 (`m3_video_formatting.py`), and dispatches them to M4 for processing. The exercise type is passed as a JSON parameter at session start.

The video formatter is the only software boundary where untrusted network data enters the backend. Test T_M3 verifies that M3 correctly rejects malformed input before any further processing.

Sub-case	Input	Expected	Result
M3-isinstance	`Base64VideoFormatter` instance	Is-a `IVideoFormatter`	**PASS**
M3-reject-invalid	Plain string (no data URI)	`None`	**PASS**
M3-reject-empty	Empty string	`None`	**PASS**
M3-reject-bad-b64	Malformed Base64 suffix	`None`	**PASS**

Table 1: T_M3 Results — Video Formatter Interface

Outcome: R1 is satisfied. The formatter correctly enforces the boundary between untrusted network input and the processing pipeline.

3.2 R2 — Track Skeletal Keypoints Per Frame

R2 states: *The system shall track skeletal keypoints for legs (P_H, P_K, P_A) and arms (P_S, P_E, P_W) in each frame.*

Keypoint extraction is delegated to MediaPipe Pose via M4 (`m4_pose_tracking.py`), which returns 33 landmarks per frame. Landmark extraction is verified implicitly by T8 (Section 6.6): the `KinematicEngine.compute_angle` function consumes 3D coordinate triples extracted from the landmark list. All five reference geometries produce angles accurate to within 0.1°, which confirms that landmark coordinates are correctly indexed and interpreted.

Outcome: R2 is satisfied.

3.3 R3 — Calculate Joint Angles

R3 states: *The system shall calculate θ_{knee} using GD_KneeAngle and θ_{elbow} using GD_ElbowAngle.*

These computations are implemented in `KinematicEngine.compute_angle` (M5) using TM:VectorAngle:

$$\theta = \arccos\left(\frac{\overrightarrow{ba} \cdot \overrightarrow{bc}}{\|\overrightarrow{ba}\| \, \|\overrightarrow{bc}\|}\right) \times \frac{180}{\pi}$$

Full results are in Section 6.6. All five analytical cases pass with error $< 0.01°$, well within the $\pm 5°$ NFR1 tolerance.

Outcome: R3 is satisfied.

3.4 R4 — Determine Exercise Validity

R4 states: *The system shall determine validity using IM_SquatValid or IM_BicepCurlValid.*

This logic is implemented in M6 (`m6_exercise_state.py`). A repetition is counted only when the normalized progress p completes the full cycle through all four states and the timing constraint ($\Delta t \geq t_{min}$) is met. The state machine thresholds directly encode the validity criteria from the SRS instance models: $\theta_{low} = 90°$ for squats corresponds to the IM_SquatValid threshold.

Test	Scenario	Expected	Result
T1	Perfect squat: $180° \to 90° \to 180°$ cosine wave (60 frames, 2 s)	`rep_count = 1`	**PASS**
T2	Failed rep: depth only $100°$, never crosses $90°$ threshold	`rep_count = 0`	**PASS**
T5	Timing violation: full ROM but in 0.2 s ($< t_{min} = 0.5$ s)	`rep_count = 0`	**PASS**

Table 2: T1, T2, T5 Results — Exercise Validity

Outcome: R4 is satisfied.

3.5 R5 — Display Skeletal Overlay and Feedback

R5 states: *The system shall display the skeletal overlay and feedback text on the screen.*

This is implemented by M7 (`frontend/modules/m7_ui_rendering.js`) and M2 (`frontend/modules/m2_display_output.js`). Automated testing of browser-side rendering is not feasible without a full headless browser harness (see Section 7). Verification was performed by visual inspection during the Proof-of-Concept demonstration (Mar 5, 2026): the AR canvas correctly rendered the MediaPipe skeleton overlay on the live video feed, and the

sidebar panel updated rep count and form quality badges in real-time via WebSocket.

Outcome: R5 is satisfied by inspection.

4 Nonfunctional Requirements Evaluation

4.1 Accuracy (NFR1)

NFR1 states: *Calculated angles shall be within ±5° of visual ground truth for frames where pose extraction confidence exceeds the backend's reliability threshold.*

Analytical verification (T8). The `compute_angle` function was tested against five reference angles spanning the full $[0°, 180°]$ range using exact 3D point coordinates.

Case	Ground Truth	Computed	Error
T8a — Right angle	90°	90.00°	$< 0.01°$
T8b — Straight	180°	180.00°	$< 0.01°$
T8c — Acute	45°	45.00°	$< 0.01°$
T8d — Obtuse	120°	120.00°	$< 0.01°$
T8e — 3D angle	60°	60.00°	$< 0.01°$

Table 3: T8 Geometric Verification Results

All five cases pass with MAE $< 0.01°$, well within the ±5° NFR1 tolerance.

Dynamic benchmark. An offline benchmark was conducted on a recorded dataset of exercise videos across three pose estimation backends: `mediapipe_2d`, `mediapipe_3d`, and `movenet_3d`. Videos were captured at multiple pitch angles (eye level, 45° down) and orientations (front, front-45, side, back, back-45). The primary metric is the mean absolute frame-to-frame angle delta, denoted $\overline{|\Delta\theta|} = \frac{1}{N-1}\sum_{i=1}^{N-1}|\theta_i - \theta_{i-1}|$ (referred to as `mean_abs_delta` in the code). This is a *stability* metric, not an error-versus-ground-truth metric:

lower values indicate a smoother, less noisy angle signal across consecutive frames.

Note on terminology. "MAE" in this report is used only in the T8 analytical results (Table 3), where it denotes Mean Absolute Error against a closed-form ground-truth angle. The dynamic benchmark below does *not* have a per-frame ground truth available, so $\overline{|\Delta\theta|}$ (frame-to-frame stability) is reported instead.

Backend	**Avg. Latency (ms)**	**Knee $\overline{\lvert\Delta\theta\rvert}$ (°)**	**Elbow $\overline{\lvert\Delta\theta\rvert}$ (°)**
`mediapipe_2d`	~ 39	~ 0.7	~ 3.8
`mediapipe_3d`	~ 41	~ 1.2	~ 2.3
`movenet_3d`	~ 46	~ 1.4	~ 5.4

Table 4: Offline Benchmark — Mean Absolute Frame-to-Frame Angle Delta ($\overline{|\Delta\theta|}$) by Backend. Lower is more stable; this is *not* error vs. ground truth.

`mediapipe_2d` achieves the best throughput ($\sim$ 25 FPS) and knee-angle stability (lowest $\overline{|\Delta\theta|}$). This data justifies the design decision in M4 to use MediaPipe 2D as the production backend.

Outcome: NFR1 is satisfied for the mathematical computation kernel. End-to-end physical validation against goniometer measurements was planned but not executed (see Section 7).

4.2 Portability (NFR2)

NFR2 states: *The system shall be platform-independent, accessible on any device with a modern web browser and camera.*

The backend is a Python/Django application exposing a WebSocket endpoint with no platform-specific OS calls. The frontend is standard HTML/CSS/JavaScript. The system was verified on macOS (Chrome) and is structurally compatible with any POSIX system running Python 3.9+ and any modern browser supporting the `getUserMedia` API.

Outcome: NFR2 is satisfied.

5 Comparison to Existing Implementation

FitCoachAR was refactored from a monolithic prototype developed for CAS 772 (Mirhoseininejad, 2026). The key changes verified in this iteration are:

1. **Modular decomposition.** The eight-module architecture (M1–M8) replaces a single-file pipeline. Each module hides exactly one design decision. The test suite verifies modules in isolation.

2. **Abstract interfaces.** Python ABCs (`IKinematicEngine`, `ISignalSmoother`, `IVideoFormatter`, `IExerciseStateMachine`) enforce the MIS contracts at the language level. Tests T_M3, T_M5, T_M8 confirm that concrete classes correctly implement these interfaces and that direct instantiation of the abstract base raises `TypeError`.

3. **Calibration parameters.** Hardcoded angle thresholds were replaced with `CalibrationParams`, enabling per-user ROM calibration. Tests T1–T7 all parameterize the state machine through `CalibrationParams`, confirming the abstraction works correctly.

4. **Timing constraints.** The state machine now enforces t_{min} and t_{max} bounds on rep duration. T5 verifies that a physically complete but temporally too-fast rep is correctly rejected.

6 Unit Testing

6.1 Test Environment

- Python 3.9.6, pytest 8.4.2, NumPy 1.x, SciPy
- Command: `cd src/backend && python3 -m pytest tests/test_vnv.py -v`
- All 11 tests passed; 0 failures; 0 errors.

6.2 Test Results Summary

Test	Modules	Description	Result
T1	M6	Perfect squat: 180° → 90° → 180° cosine wave, 60 frames	**PASS**
T2	M6	Failed rep: depth 100°, threshold 90°	**PASS**
T4	M6	Phase-by-phase FSM trace across all 4 states	**PASS**
T5	M6	Timing constraint: full ROM in 0.2 s $< t_{min}$	**PASS**
T6	M6	Intermediate ascend: p=0.1→0.6 yields `UP_PHASE`	**PASS**
T7	M6	State reversal: p=0.1→0.6→0.1 resets to `BOTTOM`	**PASS**
T8	M5	Geometric accuracy: 5 reference angles, all $< 0.01°$ error	**PASS**
T_M5	M5	ABC enforcement: `KinematicEngine` is-a `IKinematicEngine`	**PASS**
T_M8	M8	Kalman noise reduction: smoothed std $<$ raw std	**PASS**
T_M3	M3	Formatter rejects invalid, empty, malformed Base64	**PASS**
T_MULTI	M6	5 clean reps counted exactly (cosine wave, 300 frames)	**PASS**
		Total	**11 / 11**

Table 5: Unit Test Results

Note on section layout. Tests T1, T2, and T5 are reported in Section 3 (R4 evaluation, Table 2) to avoid duplication. The subsections below cover only tests whose results require additional detail (state-transition traces, geometric accuracy, signal smoothing, multi-rep accumulation) beyond what

Table 5 already summarises.

6.3 T4 — State Machine Cycle

Each of the four state transitions was driven individually and asserted:

p	t (s)	Angle (°)	Expected Phase	Observed
0.05	0.0	94.5	`BOTTOM`	`BOTTOM`
0.60	0.9	144.0	`UP_PHASE`	`UP_PHASE`
0.90	1.2	171.0	`TOP`	`TOP`
0.70	1.5	153.0	`DOWN_PHASE`	`DOWN_PHASE`
0.10	2.1	99.0	`BOTTOM`	`BOTTOM`

Table 6: T4 Phase Transition Trace (rep_count = 1 at end)

All five phase assertions passed and `rep_count` = 1 at the end. **PASS**

6.4 T6 — Intermediate Ascend

Progress stepped from p=0.1 to p=0.6 in six increments. Final state: `UP_PHASE`. `rep_count` = 0. **PASS**

6.5 T7 — State Reversal

Progress sequence $[0.05, 0.2, 0.4, 0.6, 0.4, 0.2, 0.05]$ simulates an ascending motion that reverses before reaching the peak. The state machine transitions back to `BOTTOM` without counting a rep. `rep_count` = 0. **PASS**

6.6 T8 — Geometric Accuracy

See Table 3. The `compute_angle` implementation applies `np.clip(cosine, -1.0, 1.0)` to guard against floating-point rounding outside $[-1, 1]$. All five analytically known angles are reproduced to within 0.01°. **PASS**

6.7 T_M5 — Kinematic Engine Interface

`KinematicEngine()` is confirmed to be an instance of `IKinematicEngine`. Attempting to instantiate `IKinematicEngine()` directly raises `TypeError` (ABC enforcement). **PASS**

6.8 T_M8 — Signal Smoothing

A 20-sample noisy signal was generated: $z_i \sim \mathcal{N}(90, 5^2)$. `KalmanSmoother` was applied sequentially. After discarding the initial transient (first 5 samples), the standard deviation of the smoothed signal was less than the raw signal's standard deviation in all runs. $\sigma_{\text{smoothed}} < \sigma_{\text{raw}}$. **PASS**

6.9 T_MULTI — Multiple Reps

A cosine wave spanning 5 complete rep cycles (300 frames, n_reps=5) was generated and fed into a fresh `ExerciseStateMachine`. `rep_count` = 5. **PASS**

7 Changes Due to Testing

The following deviations from the original VnV Plan occurred:

1. **T3 eliminated (merged into T2).** T3 was originally a separate "shallow squat" boundary test. During implementation it was found to be identical in structure to T2 (both test insufficient depth). T3 was merged.

2. **Robot Framework E2E testing not implemented.** Driving a live WebSocket session with a webcam in a headless browser required significant infrastructure not available within the project timeline. R5 was verified by PoC demonstration instead.

3. **Goniometer physical dataset not assembled.** The VnV Plan described 20 static images labelled with goniometer measurements for T8. Access to a biomechanics lab was not available. NFR1 was verified using analytical ground truth (exact 3D coordinates with known angles), which provides a stronger mathematical guarantee on the computation kernel.

4. **Three interface tests added (T_M3, T_M5, T_M8).** These were not in the original VnV Plan. They were added during implementation to verify that the ABC architecture is correctly enforced at runtime, which directly underpins the modularity and testability of the design.

5. **T_MULTI added for multi-rep regression.** Added after discovering a half-cycle boundary artifact in the cosine wave generator when $n\_reps > 1$. The test was used to identify and fix the issue.

8 Automated Testing

Automated testing is integrated into the development workflow via **GitHub Actions** CI. The pipeline runs on every push and pull request to the `main` branch and performs the following steps:

1. Install system dependencies (`libgl1`, OpenCV prerequisites)
2. `pip install -r requirements.txt`
3. `flake8` linting on all backend Python sources
4. `python3 -m pytest tests/test_vnv.py -v`

The most recent CI run (commit `b0bf048`) reports all 11 tests passing on `ubuntu-latest` with Python 3.11. The CI badge is visible on the project repository README.

9 Trace to Requirements

Requirement	**Tests**
R1 (Accept video + exercise selection)	T_M3, PoC inspection
R2 (Track skeletal keypoints)	T8, T_M5
R3 (Calculate joint angles)	T8
R4 (Determine exercise validity)	T1, T2, T4, T5, T6, T7, T_MULTI
R5 (Display overlay and feedback)	PoC inspection
NFR1 (Accuracy $\pm 5^\circ$)	T8, offline benchmark
NFR2 (Portability)	Architecture review, PoC inspection

Table 7: Traceability: Requirements to Tests

10 Trace to Modules

Module	**Secret**	**Tests**
M1 (Video Input)	Webcam capture API	Architecture review
M2 (Display Output)	AR skeleton rendering	PoC inspection
M3 (Video Formatting)	Base64 encode/decode	T_M3
M4 (Pose Tracking)	MediaPipe ML inference	Offline benchmark
M5 (Kinematic Engine)	Vector angle computation	T8, T_M5
M6 (Exercise State)	FSM thresholds and hysteresis	T1, T2, T4, T5, T6, T7, T_MULTI
M7 (UI Rendering)	WebSocket payload schema	PoC inspection
M8 (Signal Smoothing)	Kalman filter parameters	T_M8

Table 8: Traceability: Modules to Tests

11 Code Coverage Metrics

The automated test suite directly covers the four modules whose secrets carry the highest scientific risk:

- **M5 (`m5_kinematic_engine.py`):** `compute_angle` is exercised by T8 across 5 distinct geometries. `extract_metrics` is called indirectly via the calibration pipeline. The collinear guard (`norm < 1e-6`) is not explicitly tested.
- **M6 (`m6_exercise_state.py`):** All four state phases and all five inter-state transitions are covered (T4 traces each individually). The hysteresis reversal (T7), timing constraint (T5), and multi-rep regression (T_MULTI) cover the remaining critical branches. The t_{max} timeout reset branch is not tested.
- **M8 (`m8_signal_smoothing.py`):** The `KalmanSmoother.smooth` path is covered (T_M8). The Savitzky-Golay `LandmarkSmoother` is not covered by automated tests.
- **M3 (`m3_video_formatting.py`):** The `decode_frame` rejection paths are covered. The `encode_frame` path is not explicitly tested.

M1, M2, M4, and M7 depend on hardware (webcam), browser DOM, or ML model weights and are therefore not covered by unit tests. They are verified through the PoC demonstration and the offline evaluation benchmark.

Formal statement-level coverage measurement via `pytest-cov` was not collected in this iteration and is identified as a gap for future work.

References

Mohamad Mirhoseininejad. System requirements specification. https://mmdmirh.github.io/cas741/SRS/SRS.pdf, 2026.

Appendix — Reflection

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. **What went well while writing this deliverable?**

   Maintaining a direct correspondence between the VnV Plan test case IDs (T1, T2, T4, ...) and the pytest function names (`test_T1_perfect_squat`, etc.) made traceability straightforward and reduced the risk of accidentally omitting a planned test. The modular architecture also paid dividends here: because M3, M5, M6, and M8 are each independently importable with minimal external dependencies, writing isolated unit tests required no mocking of cameras, WebSockets, or ML models. Achieving 11/11 on the first complete run of the test suite was strong evidence that the implementation matched the MIS specification.

2. **What pain points were experienced during this deliverable, and how were they resolved?**

   The biggest gap between plan and execution was the goniometer dataset. The VnV Plan specified 20 physically labelled images as the oracle for T8, but assembling this dataset required a biomechanics lab and a calibrated setup that was unavailable. The resolution was to pivot to analytical ground truth: constructing exact 3D point triples for known angles and verifying the computation directly. While this does not capture end-to-end pipeline error (camera projection, MediaPipe uncertainty), it provides a stronger mathematical guarantee on the kernel that is the most safety-critical piece.

A secondary pain point was Robot Framework. The plan assumed automated E2E UI testing would be straightforward, but driving a live WebSocket and webcam session inside a headless browser proved to require infrastructure investment disproportionate to the verification value it would add at this stage.

3. **Which parts of this document stemmed from speaking to clients or peers?**

   The boundary-condition tests T2 (insufficient depth) and T5 (too fast) were motivated by feedback from peer reviewer Yibing Mei during the VnV Plan review. The original plan focused heavily on the happy path; Mei's review raised the concern that real users frequently perform partial or rushed reps, and these cases must be explicitly tested. T7 (state reversal) was also added in response to this feedback.

   The offline backend comparison arose from a question by Dr. Smith during the MG/MIS presentation: whether the choice of MediaPipe 2D in M4 was justified by evidence or by assumption. The offline benchmark dataset was designed to answer that question.

4. **In what ways was the VnV Plan different from the activities actually conducted?**

   There were five deviations (detailed in Section 7):

   (a) T3 was eliminated (duplicate of T2).

   (b) Robot Framework E2E testing was not implemented.

   (c) The goniometer physical dataset was replaced by analytical ground truth.

   (d) Three interface contract tests (T_M3, T_M5, T_M8) were added.

   (e) T_MULTI was added as a multi-rep regression test.

   The first three deviations reflect the classic gap between planning and execution: resource and time constraints required scope adjustments. The fourth and fifth deviations are improvements that increased overall confidence in the implementation beyond what the original plan provided.

In hindsight, the VnV Plan was over-ambitious regarding physical measurement infrastructure. Future plans should explicitly distinguish between tests that are self-contained and automatable versus tests that depend on external physical resources, and should specify fallback verification strategies for the latter category before the plan is finalized.