# Phase 1 to 3 Progress Report – Robust + ML + TCN + Rule-Based Hybrid (XAUUSD M5)

This report presents the progress achieved from Phase 1 through Phase 3 of the Robust + ML + TCN + Rule-Based Hybrid (XAUUSD M5) project. The objective across these phases is to develop a robust, data-driven predictive system capable of identifying high-quality trading signals using multi-stage methodologies. Phase 1 focused on building a clean, stable dataset foundation. Phase 2 introduced machine learning baselines with LightGBM and logistic regression, followed by calibrated probability optimization. Phase 3 marks the beginning of deep sequential modeling using Temporal Convolutional Networks (TCN) for richer temporal feature representation and higher predictive power.

## Phase 1 – Data & Robust Foundation

Goal: Prepare clean and stable M5 data for machine learning.

- Audited 5 years of IC Markets XAUUSD M5 data (2021–2025)  
- Confirmed missing bars only during weekends and rollovers  
- Applied Winsorization, MAD scaling, Hampel filter, and Huberized EMA smoothing  
- Generated canonical dataset and feature specification

Outputs:  
- data/canonical/xauusd\_m5\_cleaned.parquet  
- configs/features\_m5.yaml  
- reports/DATA\_AUDIT.md  
- reports/CLEAN\_SUMMARY.md

## Phase 2 – Machine Learning Models

Goal: Develop predictive ML models on the robust data and establish a baseline before integrating rule-based filters.

Step 1 – Triple-Barrier Labeling

Objective: Create volatility-adjusted directional labels using the triple-barrier method.  
Parameters:  
- Horizon: 20 bars (100 minutes)  
- ATR Window: 48 bars  
- TP Barrier: +0.5 × ATR  
- SL Barrier: -0.35 × ATR

Results:  
- Rows labeled: 340,258  
- Positive: 125,480  
- Neutral: 41,817  
- Negative: 172,961  
- Suppression: 12.29%

Outputs:  
- data/canonical/xauusd\_m5\_labeled.parquet  
- configs/labels.yaml  
- reports/LABEL\_SUMMARY.md

Step 2 – LightGBM Baseline

Objective: Build a gradient-boosted tree model (LightGBM) for directional prediction.  
Setup:  
- Binary classification (exclude neutrals)  
- 80/20 time-based split  
- Balanced class weights, 400 estimators  
Results:  
- Accuracy: 51.49%  
- Hit rate 0.55 threshold: 53.03%  
- Hit rate 0.60 threshold: 54.21%  
Outputs:  
- artifacts/lgbm\_model.pkl  
- reports/EVAL\_REPORT.md

Step 3 – Logistic Regression Baseline

Objective: Provide a linear baseline model using robust-scaled features.  
Setup:  
- StandardScaler + Logistic Regression pipeline  
- Saga solver with balanced weights  
- 80/20 time-based split  
Results:  
- Accuracy: 54.19%  
- Probabilities clustered around 0.50 (few trades at higher thresholds)  
Outputs:  
- artifacts/logit\_model.pkl  
- reports/LOGIT\_EVAL\_REPORT.md

Step 4 – Calibrated Logistic Regression

Objective: Improve probability calibration using isotonic regression so thresholds above 0.50 remain useful.  
Setup:  
- Split into train, calibration, and test sets  
- Used CalibratedClassifierCV (isotonic method)  
Results:  
- Accuracy: 57.99% at 0.50 threshold  
- Hit rate rises to 60.00% at 0.60 threshold  
- Recommended operating threshold: 0.58  
Outputs:  
- artifacts/logit\_model\_calibrated.pkl  
- reports/LOGIT\_CALIB\_EVAL.md  
- configs/model\_ops.yaml

Phase 2 Completion Status:  
✅ Step 1: Triple-Barrier Labeling  
✅ Step 2: LightGBM Baseline  
✅ Step 3: Logistic Regression Baseline  
✅ Step 4: Calibrated Logistic Regression

## Phase 3 – Temporal Convolutional Network (TCN)

Goal: Implement a deep sequential learning model to capture temporal dependencies beyond tabular ML.

Step 1 – Project Scaffolding

Objective: Create a dedicated TCN module and configuration for Phase 3.  
Actions Completed:  
- Created new branch: phase3-tcn  
- Added folders: src/models/tcn/, configs/, reports/figures/  
- Initialized package files: \_\_init\_\_.py  
- Added config file: tcn\_m5.yaml  
- Implemented core TCN components:  
 • dataset.py – handles rolling sequence windows  
 • model.py – self-contained TCN (no external library)  
 • train.py – training loop with class weighting and early stopping  
 • infer.py – probability inference for latest window  
Rationale:  
Self-contained TCN was chosen over the external pytorch-tcn library to retain flexibility and control for hyperparameter experiments and to keep dependencies minimal.  
Outputs:  
- configs/tcn\_m5.yaml  
- src/models/tcn/dataset.py  
- src/models/tcn/model.py  
- src/models/tcn/train.py  
- src/models/tcn/infer.py  
- reports/figures/.gitkeep  
Next Step:  
Phase 3 Step 2 – Data Windowing Validation to verify sequence alignment and memory usage before first training run.