# FIFA 23 Player Analytics with Agentic AI

Predicting Player Value, Overall Rating and Optimal Position with ML Models Wrapped in a Llama 3 LLM Agent

Krishhiv Mehra
Ashoka University

**Abstract**

This project builds a complete end-to-end system on top of the FIFA 23 male player dataset. First, supervised machine learning models are trained to predict market value, overall rating, and optimal on-field position from FIFA style attributes. Second, these models are wrapped inside an *agentic AI* layer using Meta Llama 3 8B Instruct hosted via Hugging Face Inference. The agent accepts natural–language questions like "How much is Haaland worth and what is his best position?" and answers by calling the underlying prediction tools. This report describes the dataset, data preparation, model design and evaluation, as well as the architecture and behaviour of the agentic layer.

## 1 Introduction

Football video games like *FIFA* assign each player a rich set of numerical attributes (pace, shooting, passing, defending, etc.), an overall rating and a notional market value.. These ratings encode the game designers' view of player skill, potential and role on the pitch, and they correlate strongly with real-world perception and transfer values.

The core questions this project tackles are:

- Given a player's attributes and age, can we predict their *market value*?
- Can we predict the player's *overall rating* from the underlying attributes?
- Can we infer the player's *most suitable position* (LW, ST, CAM, CDM, CM, LB, CB) purely from attributes, without directly using the positional labels?
- Finally, can we wrap these models in an **agentic AI** that behaves like a football analytics assistant, answering free–form questions while remaining grounded in data?

The system therefore combines classical machine learning (Random Forests and XGBoost) for structured prediction with an LLM–based agent that performs orchestration, tool use and explanation.

## 2 Dataset and Targets

### 2.1 Source

The project uses the *FIFA 23 male players dataset* from Kaggle. The raw file contains roughly 160k rows with 110 columns:

- Player metadata: IDs, names, age, height, weight, nationality, club, league, contract info.

- Aggregate ratings: overall, potential, pace, shooting, passing, dribbling, defending, physical.
- Detailed attributes: attacking_*, skill_*, movement_*, power_*, mentality_*, defending_*, goalkeeping_*.
- Positional ratings: numerical scores for LS, ST, RS, LW, ..., RCB, RB, GK.
- Monetary fields: `value_eur`, `wage_eur`, `release_clause_eur`.

### 2.2 Prediction Targets

Three supervised prediction tasks are defined:

1. **Market Value Regression**
   Target: `value_eur`.
   To stabilise the heavy-tailed distribution of player values, models are trained on $\log(1 + \texttt{value\_eur})$ (for normalisation purposes) and then exponentiated back.
2. **Overall Rating Regression**
   Target: `overall`, an integer between roughly 40 and 94.
3. **Primary Position Classification**
   From the detailed positional ratings, a "best position" column `position_10` is derived by selecting the highest non-goalkeeper outfield rating, restricted to the set: {LW, LB, RB, RW, CB, ST, CM, C
   Players who do not have a clear single best position in this set or are primarily goalkeepers are filtered out.

All three tasks use the same core set of numerical attributes as features, plus age and some high-level aggregate ratings.

## 3 Data Cleaning and Feature Engineering

All preprocessing lives in `data_prep.py`.

### 3.1 Row Filtering

- Keep only rows from FIFA 23, male players.
- Drop rows with missing values on the key targets (`value_eur`, `overall`) or core attributes.
- For position modelling, restrict to players with a single clear primary outfield position among CB, RB, LB, GK, CM, CDM, CAM, ST, RW, LW.

After cleaning, the modelling dataset contains $\approx$ 16,396 players, a manageable but still rich sample.

### 3.2 Feature Selection

From the many columns, the project focuses on attributes that are intuitively predictive and relatively complete:

- Core attributes: pace, shooting, passing, dribbling, defending, physic.
- Key attacking stats: attacking_crossing, attacking_finishing, attacking_heading_accuracy, attacking_short_passing.

- Skill attributes: skill_dribbling, skill_curve, skill_fk_accuracy, skill_long_passing, skill_ball_control.
- Movement: movement_acceleration, movement_sprint_speed, movement_agility, movement_reactions, movement_balance.
- Power: power_shot_power, power_jumping, power_stamina, power_strength, power_long_shots.
- Mentality: mentality_aggression, mentality_interceptions, mentality_positioning, mentality_vision, mentality_penalties, mentality_composure.
- Defending details: defending_marking_awareness, defending_standing_tackle, defending_sliding_tackle.
- Age and the aggregate positional ratings (pace, defending, etc.).

These features are already on compatible scales (0–99) so no additional normalisation is required for tree–based models.

### 3.3 Position Label Engineering

To derive `position_10`:

1. Collect the positional rating columns corresponding to LW, LB, CB, ST, CM, CAM and CDM.
2. For each player, take the argmax over these seven values.
3. Discard players with ties or all zeros (indicating no clear rating for these roles).

This yields a reasonably balanced multi–class label with the following class counts:

| Position | LW | CB | ST | LB | CAM | CDM | CM |
|---|---|---|---|---|---|---|---|
| Count | 4260 | 3673 | 2748 | 2435 | 1221 | 1129 | 930 |

## 4 Predictive Modelling

All models are implemented using `scikit-learn` and `xgboost`, with a common train-test split and re-usable helpers from `data_prep.py`.

### 4.1 Predicting Market Value

#### Models

Two models are trained on $\log(1 + \texttt{value\_eur})$:

- **Random Forest Regressor**
  Baseline ensemble of decision trees with default hyperparameters (n_estimators = 400).
- **XGBoost Regressor**
  Gradient-boosted trees tuned to balance accuracy and overfitting (learning rate, depth and number of trees adjusted after checking train vs test error).

**Random Forest Results**

On the held–out test set:

- RMSE (log value): 0.259
- MAE (log value): 0.197
- $R^2$ (log value): 0.954
- RMSE (value): €2.68 M
- MAE (value): €0.71 M
- $R^2$ (value): 0.893
- MAE as % of mean value: 23.1%

The baseline already explains almost 90% of the variance in player values.

**XGBoost Results**

With XGBoost and log–transformed values:

- **Train vs Test (log value):**
  Train RMSE = 0.146, Train $R^2 = 0.985$
  Test RMSE = 0.202, Test $R^2 = 0.972$
- **Test metrics (back-transformed):**
  RMSE (value): €2.06 M
  MAE (value): €0.54 M
  $R^2$ (value): 0.937
  MAE as % of mean value: 17.5%
  Approx. MAPE: 16.0%

The gap between train and test is small enough to suggest only mild overfitting; overall, XGBoost clearly outperforms the Random Forest.

**Feature Importance for Value**

XGBoost feature importances align nicely with football intuition:

- **movement_reactions** (0.29) and **skill_ball_control** (0.20) dominate: players who react quickly and control the ball well are highly valued. Moreover, it is one of the most fundamental thing that differentiates an amateur footballer from a professional footballer.
- **Dribbling, defending, short passing, age** all have substantial weight.
- Physical attributes and finishing/heading also matter, but less than overall technical quality.

A limitation is that very top players (Mbappé, Haaland) tend to be *undervalued* in absolute euros because the model is trained on log–values and sees relatively few extreme transfer fees. However it still correctly ranks them among the most valuable players.

## 4.2 Predicting Overall Rating

Here the target is the FIFA overall rating.

**Random Forest**

- RMSE: 1.13 rating points
- MAE: 0.86 rating points
- $R^2$: 0.972
- MAE as % of mean overall (65.9): $\approx 1.3\%$

**XGBoost**

- **Train vs Test:**
  Train RMSE = 0.685, Train $R^2 = 0.990$
- **Test metrics:**
  RMSE: 0.94 rating points
  MAE: 0.71 rating points
  $R^2$: 0.981
  MAE as % of mean overall: $\approx 1.1\%$

In practice this means the model's predicted overall is usually within one point of the true rating — essentially at human scouting noise level.

### 4.3 Classifying Optimal Position

For position prediction, the same feature set is used, and the label is `position_10` with the seven classes LW, LB, CB, ST, CM, CAM, CDM.

**Random Forest**

- Overall accuracy: 0.850
- Macro F1: 0.81
- Performance is strongest for CB, ST, LB and LW; CAM/CM/CDM are more frequently confused, which matches how similar the roles' requirements are in reality. The thing that differentiates those 3 positions is on-field IQ and game sense - which cannot be measured in numbers.

**XGBoost with Class Weights**

To handle class imbalance, inverse–frequency class weights are used.

- Train accuracy: 1.00 (as expected for a powerful boosted model)
- Test accuracy: 0.891
- Macro F1: 0.86

Per–class behaviour:

- CB and ST exceed 0.93 F1: strong, distinct feature patterns.
- LW and LB around 0.89 F1.
- CAM, CM, CDM around 0.76–0.82 F1, reflecting their overlapping attribute profiles.

Overall, the model is very effective at inferring a player's best position from their stats alone.

# 5 From Models to an Agentic AI

The second half of the project wraps these models inside an agentic AI framework powered by Meta Llama 3 8B Instruct.

## 5.1 System Architecture

The codebase is organised as follows:

- `data_prep.py`: loading, cleaning and feature engineering utilities.
- `train_value_model.py`, `train_overall_model.py`, `train_position_model.py`: training scripts that persist the Random Forest and XGBoost models as `.joblib` files and print metrics.
- `fifa_models_service.py`: a pure "tool layer" that:
  - loads the cleaned dataset and the trained models;
  - implements fuzzy player search (handling typos and variations like "Leonel Messi" vs "L. Messi");
  - exposes functions to predict value, overall and position for a given player;
  - returns structured JSON with both actual and predicted values for comparison.
- `llm_client.py`: a light wrapper over the Hugging Face Inference API (`/v1/chat/completions`) using the environment variable `HF_TOKEN`.
- `agent.py`: the main agent loop that turns user messages into tool calls and answers.

## 5.2 Agent Behaviour

The agent follows a "tool–first" pattern:

1. The user types a natural–language query, e.g.
   *"How much is Lionel Messi worth and what is his best position?"*
2. The system message defines available tools (search player, predict player, compare players) and instructs the LLM to *never make up numbers*, but instead call tools whenever factual FIFA data is needed.
3. Llama 3 parses the query, decides which tool to invoke and emits a tool call in structured JSON (OpenAI function calling style).
4. Python executes the tool via `fifa_models_service`, gets back predictions and actual values, and returns this JSON as a tool response.
5. Llama 3 then produces a final, user-friendly explanation: summarising predicted value, overall, best position and the confidence (top-3 position probabilities).

Example behaviour:

- For "How much is Lionel Messi worth?" the agent returns an estimated value of roughly €46-47M, predicted overall around 89, and best position CAM with LW/ST as secondary options, while also mentioning that actual dataset value and overall are slightly different.
- For "Compare Haaland and Lewandowski on value and overall" it calls the comparison tool and summarises that Haaland is younger, predicted to be more valuable and slightly higher overall, while both are optimal as STs.

### 5.3 Guardrails and Evaluation

Inspired by recent work on agentic guardrail pipelines, the project implements several safeguards:

- **Grounding in tools**: The system prompt explicitly bans guessing precise numbers; the LLM must call tools for any FIFA statistic.
- **Name resolution layer**: Fuzzy matching is done on the Python side, so the model never fabricates player IDs.
- **Low-temperature sampling**: The LLM uses a relatively low temperature to keep explanations stable and avoid imaginative stories.
- **JSON sanity**: Tools always return well-typed numeric fields; the agent only has to rephrase them.

Empirically, this greatly reduces hallucinations: wrong answers mostly stem from limitations of the underlying models (e.g. undervaluing superstars) rather than from the LLM inventing facts.

## 6 Discussion and Limitations

### 6.1 Predictive Models

- Value and overall models achieve strong performance (value $R^2 \approx 0.94$, overall $R^2 \approx 0.98$), with interpretable feature importances that match football intuition.
- Position classification is robust (almost 89% accuracy) but struggles to perfectly separate CAM/CM/CDM, which is expected given their overlapping roles.
- Because values are log-transformed, absolute euro errors remain high for a few extremely expensive players; however percentage error is much more reasonable and the ranking is preserved.

### 6.2 Agentic Layer

- The LLM adds a "natural language interface" on top of the models, turning them into something closer to an interactive scouting assistant.
- Tool–based grounding works well, but some edge cases remain: ambiguous names, multiple players with similar strings, and questions that mix FIFA data with real–world facts outside the dataset.
- The system currently uses a single Llama 3 instance for both planning and explanation; a richer pipeline could separate these roles and add explicit self–checks.

## 7 Conclusion

This project demonstrates how classical supervised learning and modern LLM agents can be combined in a practical sports-analytics scenario. Tree-based models on structured FIFA 23 attributes achieve strong performance on player value, overall rating and positional role prediction. Wrapping these models inside a Llama 3 based agent transforms them into an interactive system that can answer rich, football-specific questions while staying grounded in data.

Beyond the specific metrics, the main contribution is the *pattern* itself: use traditional predictive models where they are strongest, and let the LLM orchestrate, translate and explain - not hallucinate numbers. This pattern generalises well to many other domains where tabular prediction and human-facing natural language interfaces need to meet.