**🏗️ FULL ARCHITECTURE PLAN: NBA Score Prediction System**

**📋 OVERVIEW**

You're building a **modular ensemble system** that combines:

1. **Team-level models** (ELO) - captures overall team strength
2. **Player-level models** (Neural Nets) - captures roster composition & matchups
3. **Situational adjustments** (Rest, home court, travel) - captures context
4. **Meta-model** (Linear/XGBoost) - intelligently combines everything
5. **Uncertainty quantification** (Gaussian) - provides probabilities

**🎯 COMPONENT 1: DUAL ELO SYSTEM**

**What it does:**

Tracks each team's **offensive ability** and **defensive ability** separately over time.

**Why two ELOs?**

* Some teams score a lot but allow a lot (high pace, bad defense)
* Some teams score little but defend well (slow pace, great defense)
* One ELO can't capture both dimensions

**Input data:**

From your **team data**:

* home\_pts, away\_pts (actual scores)
* home\_team\_id, away\_team\_id
* game\_date (for chronological updates)
* season\_year (separate ELO per season)

**How it works:**

**Initialize:** Every team starts each season at:

* Offensive ELO = 1500
* Defensive ELO = 1500

**After each game:**

1. **Offensive ELO update:**
   * Did team score more than expected? → ELO goes up
   * Did team score less than expected? → ELO goes down
2. **Defensive ELO update:**
   * Did team allow fewer points than expected? → ELO goes up
   * Did team allow more points than expected? → ELO goes down

**Key parameters:**

* **K-factor** = 20 (how fast ratings change)
* **Margin of victory** multiplier (blowouts count more)
* **Regression to mean** each season (30% back to 1500)

**Output:**

For any game, you get:

* home\_offensive\_elo (e.g., 1650 = good offense)
* home\_defensive\_elo (e.g., 1450 = bad defense)
* away\_offensive\_elo (e.g., 1520)
* away\_defensive\_elo (e.g., 1580)

**Expected score calculation:**

home\_expected = f(home\_offensive\_elo, away\_defensive\_elo)

away\_expected = f(away\_offensive\_elo, home\_defensive\_elo)

This gives you a **baseline prediction** before considering players, rest, etc.

**🎯 COMPONENT 2: HOME TEAM NEURAL NETWORK**

**What it does:**

Predicts **how many points the home team will score** based on their roster, recent performance, and opponent.

**Architecture:**

INPUT: 12 players × 40 recent games × 19 features = shape (12, 40, 19)

↓

PLAYER RATING LAYER (20→16→12→4)

Converts each player's recent games into 4 numbers

↓

WEIGHTED AGGREGATION

Weight each player by playing time (starters > bench)

↓

TEAM RATING LAYER (8→16→1)

Combines all player ratings into one team rating

↓

OUTPUT: Expected home team points (e.g., 112.3)

**Input features (19 features per player per game):**

**From your player data:**

1. **Context features:**
   * home\_offensive\_elo / 100 (from Component 1)
   * away\_defensive\_elo / 100 (from Component 1)
   * is\_home (always 1 for home net)
2. **Core box score stats:**
   * min (minutes played)
   * fgm, fga (field goals)
   * fg3m, fg3a (three pointers)
   * ftm, fta (free throws)
   * oreb, dreb, reb (rebounds)
   * ast (assists)
   * stl, blk (defense)
   * tov (turnovers)
   * pf (fouls)
   * pts (points scored)
3. **Advanced features you already have:**
   * ts\_pct (true shooting %)
   * ast\_to\_ratio (assist to turnover)

**How to build the roster (12 players):**

**For each upcoming game:**

1. Look at home team's **last 5 games**
2. Sum up minutes played by each player across those 5 games
3. Take the **top 12 players by minutes** (these are your current rotation)
4. For each of those 12 players, get their **last 40 games of stats**

**Playing time weights:**

If player A played 150 mins in last 5 games → weight = 150/240 = 0.625

If player B played 60 mins in last 5 games → weight = 60/240 = 0.250

**Time decay:**

Games from 30 days ago matter less than games from yesterday.

For each game in a player's history:

weight = 0.9965 ^ (days\_since\_game)

Example:

* Game from yesterday: weight = 0.9965^1 = 0.996
* Game from 30 days ago: weight = 0.9965^30 = 0.901
* Game from 100 days ago: weight = 0.9965^100 = 0.704

**What if a player doesn't have 40 games?**

Fill with zeros (the network learns to ignore these).

**What if a player is injured/new?**

Use available\_flag from your data - if unavailable, set their weight to 0.

**Output:**

One number: **Expected home team points** (e.g., 113.7)

**🎯 COMPONENT 3: AWAY TEAM NEURAL NETWORK**

**What it does:**

Identical to Component 2, but for the away team.

**Key difference:**

* is\_home = 0 (always, for away net)
* Uses home\_defensive\_elo (home team's defense) instead of away
* Uses away\_offensive\_elo (away team's offense)

**Architecture:**

Exact same structure as home net: (12, 40, 19) → 4 dims → 1 output

**Output:**

One number: **Expected away team points** (e.g., 107.2)

**🎯 COMPONENT 4: HOME COURT ADVANTAGE**

**What it does:**

Adds extra points to home team based on **which team and which arena**.

**Why not just let the neural net learn this?**

Home court varies by team in ways that are hard for a neural net to capture (altitude, crowd noise, travel patterns).

**Simple approach (recommended to start):**

Calculate **empirical home court advantage** for each team from your historical data:

For each team:

Average home points - Average away points = Home advantage

Example results:

* Denver (DEN): +5.2 points (altitude)
* Utah (UTA): +4.8 points (altitude)
* Portland (POR): +3.9 points (loud crowd)
* New York (NYK): +3.5 points
* League average: +2.8 points

**Input data:**

From your **team data**:

* home\_team\_abbreviation (e.g., "LAL", "BOS", "DEN")

**Output:**

One number: **Home court points boost** (e.g., +3.2)

**Advanced approach (Phase 2):**

Build an XGBoost model with features:

* Team ID
* Day of week (weekend games = more fans)
* Rivalry indicator (Lakers vs Celtics = louder)
* Playoff vs regular season
* Arena capacity utilization
* Time since last home game

Target: home\_pts - away\_pts controlling for team strength

**🎯 COMPONENT 5: REST & FATIGUE ADJUSTMENTS**

**What it does:**

Adjusts predicted scores based on how rested or tired each team is.

**Input calculation:**

From your **team data** sorted by game\_date:

For each team in each game:

days\_rest = current\_game\_date - last\_game\_date

**Rest penalty lookup:**

**For home team:**

If days\_rest == 0: penalty = -4.0 points (back-to-back, brutal)

If days\_rest == 1: penalty = -1.5 points (tired)

If days\_rest == 2: penalty = 0.0 points (normal)

If days\_rest >= 3: penalty = +0.5 points (well rested)

**For away team:** Same logic, but also consider:

* Did they travel cross-country? (extra fatigue)
* Did they cross time zones? (jet lag)

**How to calculate these penalties:**

Use your historical data:

Group games by days\_rest (0, 1, 2, 3+)

Calculate average points scored for each group

Compare to baseline (2 days rest)

The difference is your penalty

**Output:**

Two numbers:

* home\_rest\_adjustment (e.g., -2.0 for back-to-back)
* away\_rest\_adjustment (e.g., +0.5 for 3 days rest)

**🎯 COMPONENT 6: META-MODEL (THE COMBINER)**

**What it does:**

Takes all the predictions from Components 1-5 and intelligently combines them into final predictions.

**Why not just average them?**

Different components are good at different things:

* ELO is great for team strength, bad for injuries
* Neural net is great for current roster, bad for small samples
* Home court is consistent but doesn't change
* Rest effects vary by team (young teams handle it better)

**Input features:**

For each game, you have these 10 numbers:

From ELO (Component 1):

1. home\_offensive\_elo

2. home\_defensive\_elo

3. away\_offensive\_elo

4. away\_defensive\_elo

From Neural Nets (Components 2-3):

5. home\_nn\_prediction

6. away\_nn\_prediction

From Adjustments (Components 4-5):

7. home\_court\_advantage

8. home\_rest\_adjustment

9. away\_rest\_adjustment

Additional useful features:

10. season\_year (to capture era effects)

11. home\_pace (from recent games avg)

12. away\_pace (from recent games avg)

You can calculate pace from your data:

pace = (home\_fga + home\_fta\*0.44 + home\_tov) + (away\_fga + away\_fta\*0.44 + away\_tov)

**Model type: Ridge Regression (start here)**

Why Ridge?

* Simple, interpretable
* Automatically learns optimal weights
* Regularization prevents overfitting
* Fast to train

**How it works:**

Train the Ridge model on historical games where you know:

- All 10+ input features

- Actual home\_pts and away\_pts

The model learns:

home\_pts = w1\*home\_off\_elo + w2\*away\_def\_elo + w3\*home\_nn\_pred

+ w4\*home\_court + w5\*home\_rest + ... + bias

away\_pts = (similar formula for away)

**Training approach:**

Use **walk-forward validation**:

1. Train on games 1-1000
2. Predict game 1001
3. Add game 1001 to training data
4. Retrain every 500 games
5. Predict game 1002
6. Repeat...

**Output:**

Two numbers:

* final\_home\_prediction (e.g., 114.2)
* final\_away\_prediction (e.g., 108.7)

**Advanced alternative: XGBoost (Phase 2)**

If Ridge isn't accurate enough, try XGBoost which can learn:

* Non-linear combinations (maybe rest matters more in back-to-backs for certain teams)
* Interactions (maybe ELO + neural net together mean something special)
* Team-specific patterns

**🎯 COMPONENT 7: UNCERTAINTY QUANTIFICATION**

**What it does:**

Instead of just predicting "Home: 114, Away: 109", you get "Home: 114 ± 9, Away: 109 ± 8".

This lets you calculate **probabilities** for betting.

**Method: Bootstrap Resampling**

**How it works:**

1. **Take your meta-model from Component 6**
2. **Create 100 slightly different versions:**
3. For i = 1 to 100:
4. - Randomly sample your training data (with replacement)
5. - Train a meta-model on this sample
6. - Store the model
7. **Make predictions with all 100 models:**
8. For the upcoming game:
9. Model 1 predicts: Home 113, Away 107
10. Model 2 predicts: Home 115, Away 108
11. Model 3 predicts: Home 112, Away 109
12. ...
13. Model 100 predicts: Home 114, Away 108
14. **Calculate statistics:**
15. home\_mean = average of all 100 home predictions = 114.2
16. home\_std = standard deviation = 8.3
17. away\_mean = average of all 100 away predictions = 108.1
18. away\_std = standard deviation = 7.9

**What you get:**

Home team: 114.2 ± 8.3 points (90% CI: 100-128)

Away team: 108.1 ± 7.9 points (90% CI: 95-121)

**Output:**

Four numbers:

* home\_mean, home\_std
* away\_mean, away\_std

**🎯 COMPONENT 8: PROBABILITY CALCULATOR**

**What it does:**

Converts your predictions into **betting probabilities**.

**Using the Gaussian assumption:**

You have:

* Home: μ=114.2, σ=8.3
* Away: μ=108.1, σ=7.9

**Calculate: P(Home wins)**

Score difference = Home - Away

Difference mean = 114.2 - 108.1 = 6.1

Difference std = sqrt(8.3² + 7.9²) = 11.5

P(Home wins) = P(difference > 0)

= Using Gaussian CDF

= 70.2%

**Calculate: P(Total > 220)**

Total mean = 114.2 + 108.1 = 222.3

Total std = sqrt(8.3² + 7.9²) = 11.5

P(Total > 220) = P(total > 220 given N(222.3, 11.5))

= Using Gaussian CDF

= 58.4%

**Calculate: P(Home covers -3.5 spread)**

P(Home - Away > -3.5) = P(difference > -3.5)

= P(N(6.1, 11.5) > -3.5)

= 79.8%

**Monte Carlo alternative (more accurate for complex bets):**

Run 10,000 simulations:

- Sample home\_score from N(114.2, 8.3)

- Sample away\_score from N(108.1, 7.9)

- Check if condition is met

P(condition) = (number of times condition met) / 10,000

**Output:**

For any bet type:

* p\_home\_wins (e.g., 0.702 = 70.2%)
* p\_over\_line (e.g., 0.584 = 58.4% for O/U 220)
* p\_cover\_spread (e.g., 0.798 = 79.8% for -3.5)

**📊 FULL WORKFLOW: GAME DAY PREDICTION**

**Scenario: Lakers @ Celtics, March 15, 2025**

**Step 1: Load historical data**

* Last 5 games for each team (roster identification)
* Last 40 games for each player (performance history)
* Season-long data (ELO ratings)

**Step 2: Component 1 (ELO)**

Lakers offensive ELO: 1580

Lakers defensive ELO: 1490

Celtics offensive ELO: 1620

Celtics defensive ELO: 1550

Baseline prediction:

Celtics (home): 113.2

Lakers (away): 108.4

**Step 3: Component 2 (Home Neural Net)**

Celtics roster (last 5 games top 12):

Tatum, Brown, Holiday, Porzingis, White, Horford, ...

Each player's last 40 games fed into network

Time-weighted by recency

Weighted by playing time

Neural net output: 115.7

**Step 4: Component 3 (Away Neural Net)**

Lakers roster:

LeBron, AD, DLo, Reaves, Reddish, ...

Neural net output: 110.3

**Step 5: Component 4 (Home Court)**

Celtics home court advantage: +3.8 points

**Step 6: Component 5 (Rest)**

Celtics: 2 days rest → 0.0 adjustment

Lakers: 1 day rest (back-to-back) → -1.5 adjustment

**Step 7: Component 6 (Meta-Model)**

Input features:

elo\_home\_off: 1620

elo\_home\_def: 1550

elo\_away\_off: 1580

elo\_away\_def: 1490

nn\_home: 115.7

nn\_away: 110.3

home\_adv: 3.8

rest\_home: 0.0

rest\_away: -1.5

Ridge model combines:

Celtics: 0.25\*113.2 + 0.55\*115.7 + 0.20\*3.8 = 114.9

Lakers: 0.25\*108.4 + 0.55\*110.3 + 1.0\*(-1.5) = 108.2

**Step 8: Component 7 (Uncertainty)**

Run 100 bootstrap models:

Celtics: 114.9 ± 8.1

Lakers: 108.2 ± 7.8

**Step 9: Component 8 (Probabilities)**

P(Celtics win) = 72.1%

P(Total > 223) = 48.3%

P(Celtics -6.5) = 54.2%

**Final output:**

PREDICTION: Celtics 114.9, Lakers 108.2

TOTAL: 223.1 points

SPREAD: Celtics -6.7

CONFIDENCE: Celtics 114.9 (±8.1), Lakers 108.2 (±7.8)

PROBABILITIES:

- Celtics win: 72.1%

- Over 223: 48.3%

- Celtics cover -6.5: 54.2%

**🚀 IMPLEMENTATION PHASES**

**Phase 1: Foundation (Weeks 1-2)**

✅ Build ELO system (Component 1)  
✅ Calculate home court advantage table (Component 4)  
✅ Calculate rest adjustments table (Component 5)  
✅ Test these on historical data

**Goal:** Get baseline predictions working

**Phase 2: Neural Networks (Weeks 3-4)**

✅ Build home neural net (Component 2)  
✅ Build away neural net (Component 3)  
✅ Train on 2019-2025 data  
✅ Implement walk-forward validation

**Goal:** Beat ELO-only baseline by 1-2 MAE points

**Phase 3: Integration (Week 5)**

✅ Build Ridge meta-model (Component 6)  
✅ Combine all components  
✅ Validate on 2024-25 season

**Goal:** Final MAE under 15 points

**Phase 4: Uncertainty (Week 6)**

✅ Implement bootstrap (Component 7)  
✅ Build probability calculator (Component 8)  
✅ Backtest betting strategies

**Goal:** Identify +EV bets (expected value > 0)

**Phase 5: Production (Week 7+)**

✅ Automated daily predictions  
✅ Model monitoring (track accuracy over time)  
✅ Iterative improvements

**✅ SUCCESS METRICS**

**Model Performance:**

* MAE < 15 points (vs current 16.58)
* Win accuracy > 60% (vs current 55.5%)
* Within ±10 points: 45%+ (vs current 36.8%)

**Betting Performance:**

* ROI > 3% over 100 bets
* Hit rate on +EV bets: 55%+
* Sharpe ratio > 1.0

**System Health:**

* Predictions generated daily without manual intervention
* Model retrains automatically every 500 games
* Alerts when prediction confidence is low

This architecture gives you a **transparent, debuggable, and improvable system** where you can track which component contributes what, and optimize each piece independently. Start simple, validate each component, then add complexity as needed! 🚀

**📁 PROJECT STRUCTURE - YES, Separate Files!**

Absolutely! Each component should be its own file. Here's the **recommended structure**:

**🗂️ RECOMMENDED FILE ORGANIZATION**

nba\_prediction/

│

├── data/

│ ├── raw/ # Original CSV files

│ │ ├── player\_stats.csv

│ │ └── team\_stats.csv

│ │

│ ├── processed/ # Cleaned/preprocessed data

│ │ ├── player\_features.parquet

│ │ └── team\_features.parquet

│ │

│ └── models/ # Saved model weights

│ ├── home\_nn\_weights.pth

│ ├── away\_nn\_weights.pth

│ ├── elo\_state.pkl

│ └── meta\_model.pkl

│

├── src/

│ ├── \_\_init\_\_.py

│ │

│ ├── data/

│ │ ├── \_\_init\_\_.py

│ │ ├── loader.py # Load and validate raw data

│ │ └── preprocessor.py # Clean and prepare features

│ │

│ ├── components/

│ │ ├── \_\_init\_\_.py

│ │ ├── elo\_system.py # Component 1: ELO ratings

│ │ ├── home\_neural\_net.py # Component 2: Home team NN

│ │ ├── away\_neural\_net.py # Component 3: Away team NN

│ │ ├── home\_advantage.py # Component 4: Home court

│ │ ├── rest\_fatigue.py # Component 5: Rest adjustments

│ │ ├── meta\_model.py # Component 6: Ensemble combiner

│ │ ├── uncertainty.py # Component 7: Bootstrap/uncertainty

│ │ └── probability.py # Component 8: Probability calculator

│ │

│ ├── features/

│ │ ├── \_\_init\_\_.py

│ │ ├── player\_features.py # Extract player-level features

│ │ ├── team\_features.py # Extract team-level features

│ │ └── game\_features.py # Extract game context features

│ │

│ ├── models/

│ │ ├── \_\_init\_\_.py

│ │ └── neural\_net\_base.py # Shared NN architecture classes

│ │

│ ├── training/

│ │ ├── \_\_init\_\_.py

│ │ ├── train\_elo.py # Train ELO system

│ │ ├── train\_neural\_nets.py # Train home/away NNs

│ │ └── train\_meta\_model.py # Train meta-model

│ │

│ ├── prediction/

│ │ ├── \_\_init\_\_.py

│ │ └── predictor.py # Main prediction pipeline

│ │

│ └── utils/

│ ├── \_\_init\_\_.py

│ ├── validation.py # Walk-forward validation

│ ├── metrics.py # MAE, RMSE, accuracy calculations

│ └── plotting.py # Visualization utilities

│

├── notebooks/

│ ├── 01\_data\_exploration.ipynb

│ ├── 02\_elo\_analysis.ipynb

│ ├── 03\_neural\_net\_training.ipynb

│ ├── 04\_component\_evaluation.ipynb

│ └── 05\_betting\_strategy.ipynb

│

├── scripts/

│ ├── train\_full\_pipeline.py # Train all components

│ ├── predict\_today.py # Generate daily predictions

│ ├── backtest.py # Historical betting simulation

│ └── update\_models.py # Retrain with new data

│

├── tests/

│ ├── test\_elo.py

│ ├── test\_neural\_nets.py

│ ├── test\_meta\_model.py

│ └── test\_predictor.py

│

├── config/

│ ├── model\_config.yaml # Model hyperparameters

│ ├── data\_config.yaml # Data paths and settings

│ └── training\_config.yaml # Training parameters

│

├── requirements.txt

├── README.md

└── main.py # Entry point

**🎯 WHY THIS STRUCTURE?**

**1. Modularity**

Each component is independent - you can:

* Test ELO without touching neural nets
* Swap Ridge for XGBoost in meta-model without breaking anything
* Update home advantage table without retraining NNs

**2. Reusability**

# In any script, you can do:

from src.components.elo\_system import ELOSystem

from src.components.home\_neural\_net import HomeNeuralNet

elo = ELOSystem()

home\_nn = HomeNeuralNet()

**3. Testing**

Each component has its own test file:

# tests/test\_elo.py

def test\_elo\_initialization():

elo = ELOSystem()

assert elo.get\_rating('LAL', 2024) == 1500

def test\_elo\_update():

elo = ELOSystem()

elo.update\_game('LAL', 'BOS', 110, 105)

assert elo.get\_rating('LAL', 2024) > 1500

**4. Version Control**

Git works better with small files:

* Change home advantage? Only home\_advantage.py shows in diff
* Update neural net? Only home\_neural\_net.py changes
* Easy to see what broke when

**5. Collaboration**

If someone helps you:

* Person A works on elo\_system.py
* Person B works on neural\_net\_base.py
* No merge conflicts!

**📝 KEY FILE DESCRIPTIONS**

**components/elo\_system.py**

class ELOSystem:

def \_\_init\_\_(self, k\_factor=20):

"""Initialize ELO ratings"""

def get\_offensive\_rating(self, team\_id, season):

"""Get team's offensive ELO"""

def get\_defensive\_rating(self, team\_id, season):

"""Get team's defensive ELO"""

def update\_game(self, home\_id, away\_id, home\_pts, away\_pts):

"""Update ratings after game"""

def predict(self, home\_id, away\_id, season):

"""Predict scores based on ELO"""

def save(self, filepath):

"""Save ELO state"""

def load(self, filepath):

"""Load ELO state"""

**components/home\_neural\_net.py**

class HomeNeuralNet:

def \_\_init\_\_(self):

"""Initialize network architecture"""

def prepare\_input(self, roster, player\_stats, game\_date):

"""Convert raw data to network input"""

def predict(self, home\_team\_id, away\_team\_id, game\_date):

"""Predict home team score"""

def train(self, training\_data, epochs=40):

"""Train the network"""

def save\_weights(self, filepath):

"""Save model weights"""

def load\_weights(self, filepath):

"""Load model weights"""

**components/meta\_model.py**

class MetaModel:

def \_\_init\_\_(self, model\_type='ridge'):

"""Initialize meta-model (Ridge or XGBoost)"""

def prepare\_features(self, elo\_pred, nn\_pred, home\_adv, rest):

"""Combine all component predictions"""

def train(self, training\_data):

"""Learn optimal weights"""

def predict(self, features):

"""Final score prediction"""

def get\_feature\_importance(self):

"""Which components matter most?"""

**prediction/predictor.py**

class GamePredictor:

def \_\_init\_\_(self):

"""Load all components"""

self.elo = ELOSystem()

self.home\_nn = HomeNeuralNet()

self.away\_nn = AwayNeuralNet()

self.meta = MetaModel()

def predict\_game(self, home\_id, away\_id, game\_date):

"""

Full pipeline:

ELO → NNs → Adjustments → Meta → Uncertainty → Probability

Returns:

{

'home\_mean': 114.2,

'home\_std': 8.3,

'away\_mean': 108.1,

'away\_std': 7.9,

'p\_home\_win': 0.721,

'p\_over\_220': 0.583

}

"""

**🔧 HOW COMPONENTS INTERACT**

**Example: Making a prediction**

# main.py or predict\_today.py

from src.prediction.predictor import GamePredictor

predictor = GamePredictor()

result = predictor.predict\_game(

home\_id='BOS',

away\_id='LAL',

game\_date='2025-03-15'

)

print(f"Celtics: {result['home\_mean']:.1f} ± {result['home\_std']:.1f}")

print(f"Lakers: {result['away\_mean']:.1f} ± {result['away\_std']:.1f}")

print(f"P(Celtics win): {result['p\_home\_win']:.1%}")

**Behind the scenes, predictor.predict\_game() does:**

# Inside predictor.py

def predict\_game(self, home\_id, away\_id, game\_date):

# Component 1: ELO

elo\_pred = self.elo.predict(home\_id, away\_id, season)

# Components 2-3: Neural Nets

home\_nn\_pred = self.home\_nn.predict(home\_id, away\_id, game\_date)

away\_nn\_pred = self.away\_nn.predict(home\_id, away\_id, game\_date)

# Component 4: Home advantage

home\_adv = self.home\_advantage.get\_advantage(home\_id)

# Component 5: Rest

rest\_home, rest\_away = self.rest\_fatigue.get\_adjustments(

home\_id, away\_id, game\_date

)

# Component 6: Meta-model combines everything

features = self.\_prepare\_meta\_features(

elo\_pred, home\_nn\_pred, away\_nn\_pred,

home\_adv, rest\_home, rest\_away

)

home\_pred, away\_pred = self.meta.predict(features)

# Component 7: Uncertainty

home\_std, away\_std = self.uncertainty.estimate(features)

# Component 8: Probabilities

probs = self.probability.calculate(

home\_pred, away\_pred, home\_std, away\_std

)

return {

'home\_mean': home\_pred,

'home\_std': home\_std,

'away\_mean': away\_pred,

'away\_std': away\_std,

\*\*probs

}

**🚀 DEVELOPMENT WORKFLOW**

**Week 1: Setup**

# Create structure

mkdir -p nba\_prediction/src/components

cd nba\_prediction

# Start with ELO

# Edit: src/components/elo\_system.py

# Test: tests/test\_elo.py

# Validate: notebooks/02\_elo\_analysis.ipynb

**Week 2: Neural Nets**

# Edit: src/components/home\_neural\_net.py

# Edit: src/components/away\_neural\_net.py

# Train: scripts/train\_neural\_nets.py

# Monitor: notebooks/03\_neural\_net\_training.ipynb

**Week 3: Integration**

# Edit: src/components/meta\_model.py

# Edit: src/prediction/predictor.py

# Test: scripts/predict\_today.py

# Validate: notebooks/04\_component\_evaluation.ipynb

**Week 4: Production**

# Automate: scripts/update\_models.py (cron job)

# Monitor: Track predictions in database

# Backtest: scripts/backtest.py

**📦 CONFIGURATION FILES**

**config/model\_config.yaml**

elo:

k\_factor: 20

initial\_rating: 1500

regression\_factor: 0.3

neural\_net:

player\_layers: [20, 16, 12, 4]

team\_layers: [8, 16, 1]

learning\_rate: 0.003

weight\_decay: 0.00002

batch\_size: 64

warmup\_games: 1000

retrain\_frequency: 500

meta\_model:

model\_type: 'ridge' # or 'xgboost'

alpha: 1.0

uncertainty:

n\_bootstrap: 100

confidence\_level: 0.90

**config/data\_config.yaml**

data\_paths:

player\_stats: 'data/raw/player\_stats.csv'

team\_stats: 'data/raw/team\_stats.csv'

data\_range:

start\_date: '2019-10-01'

end\_date: '2025-04-30'

features:

player\_history\_length: 40

roster\_size: 12

lookback\_games: 5

**✅ BENEFITS OF THIS STRUCTURE**

**Easy to Debug**

# Something wrong with predictions?

# Test each component individually:

elo\_pred = elo.predict('BOS', 'LAL') # Works? ✅

nn\_pred = home\_nn.predict('BOS', 'LAL') # Works? ❌ <- Found it!

# Debug just the neural net without touching other components

**Easy to Improve**

# Want to try XGBoost instead of Ridge?

# Just change meta\_model.py

# Everything else stays the same!

class MetaModel:

def \_\_init\_\_(self, model\_type='xgboost'): # Change here

if model\_type == 'xgboost':

self.model = XGBRegressor()

elif model\_type == 'ridge':

self.model = Ridge()

**Easy to Scale**

# Want to add Component 9: Injury Tracker?

# Create: src/components/injury\_tracker.py

# Import in: src/prediction/predictor.py

# Add to meta-model features

# Nothing else breaks!

**Easy to Deploy**

# Production script

# scripts/predict\_today.py

from src.prediction.predictor import GamePredictor

import schedule

predictor = GamePredictor()

def daily\_predictions():

todays\_games = get\_todays\_games()

for game in todays\_games:

pred = predictor.predict\_game(game['home'], game['away'], game['date'])

save\_to\_database(pred)

schedule.every().day.at("09:00").do(daily\_predictions)

**🎯 BOTTOM LINE**

**YES, separate files are essential!**

This structure gives you:

* ✅ Clean separation of concerns
* ✅ Easy testing and debugging
* ✅ Simple to add/remove/swap components
* ✅ Professional, maintainable codebase
* ✅ Ready to deploy to production
* ✅ Easy for others (or future you) to understand

Start with the basic structure, then fill in each component one at a time. You'll thank yourself later! 🚀