Phase 4 v2 – Machine Learning Training
(Merged and Expanded)

---

1. Purpose of Phase 4

Phase 4 is where the system transitions from deterministic, rule-driven logic (Phase 3 baseline) to statistical learning.

The machine learning layer:

1. Learns complex, nonlinear relationships between features and outcomes.
2. Trains separate models for each predicted statistic:
    - targets, receptions, rec yards, rec TDs
    - carries, rush yards, rush TDs
    - pass attempts, completions, pass yards, pass TDs, interceptions
3. Incorporates defender-level impact and archetype interactions as features, not rules.
4. Produces predictions that are fed into the ensemble meta-model (Phase 6), which blends them with the baseline.

Phase 4 v1 introduced the idea of "ML models per stat."

Phase 4 v2 expands this into a full ML training system, with:

- multiple models
- multiple stages
- leak-safe training
- versioned artifacts
- cross-validation
- feature importance tracking
- error analysis
- retraining workflows

---

2. Inputs to the ML Training System

The ML training system consumes:

1. player_game_features table (Phase 2 output)

Contains all engineered features:

- usage
- efficiency
- deltas
- team context
- defense metrics
- defender-level metrics
- coverage probabilities
- archetype features
- labels (actual stats)

This is the "X" matrix.

2. Labels (y-values)

From Phase 2:

- targets
- receptions
- receiving yards
- receiving TDs
- carries

- rush yards
- rush TDs
- QB stats
- etc.

Each stat corresponds to one ML model.

3. Baseline Predictions (Phase 3)

These are optional ML inputs but extremely powerful:

- ML learns “residuals”:
  actual − baseline_pred
- Models learn where baseline under/over-predicts
- Greatly stabilizes training

4. Archetype IDs

Used either as:

- one-hot inputs
- embedding vectors
- or combined interaction features

5. Defender features

If the distribution is sparse, we use:

- effective defender metrics
- archetype-based defender summary features
- team-defense aggregates

---

3. Leak-Free Training Splits

The most important rule:

Never train on future weeks.

Never train on future seasons.

Never mix weeks past the prediction horizon.

Correct split:

Option A (recommended): Split by season

- Train on Season1 + Season2 + Season3
- Validate on Season4
- Test on Season5

Option B (advanced): Rolling forward-time split

Examples:

- Train on Weeks 1–8 → Validate Weeks 9–12 → Test Weeks 13–17
- Move the window forward when retraining

Either way:

- No random shuffle splitting
- Always preserve chronological order

This prevents leakage.

---

4. Models to Train (Phase 4 v2)

Phase 4 v1 described:

- linear models
- gradient boosting models

Phase 4 v2 expands this into a modular multi-model system:

4.1 Linear Models (Ridge / ElasticNet)

Used because:

- simple
- interpretable

- robust to outliers
- good for sparse data

Recommended hyperparameters:

- alpha range: [0.01, 0.1, 1.0, 10]
- l1_ratio (for ElasticNet): [0.0, 0.25, 0.5, 0.75]

4.2 Gradient Boosting Models (XGBoost, LightGBM, CatBoost)

Used because:

- nonlinear
- capture interactions between features
- handle missingness
- very strong out-of-the-box performance

Recommended configs:

- shallow to medium depth: 4–8
- trees: 200–600
- learning rate: 0.02–0.1
- subsample: 0.7–1.0
- colsample_bytree: 0.6–1.0

4.3 Optional Models (Phase 4 v2)

Advanced but useful:

Similarity-Based (k-NN)

- Compute predictions based on closest historical “neighbors”
- Works well for players with similar archetypes

Neural Networks (Optional)

- Not required
- Harder to interpret
- Not used in ensemble unless controlled

Gam Models (Generalized Additive Models)

- Very interpretable
- Good for low-dimensional predictions

These are optional, not required.

4.4 Defender-Aware Submodel (NEW in v2)

A dedicated model that predicts:

“Given effective defender metrics + archetype features, what is the likely impact on this stat?”

This model consumes:

- effective defender ypt
- effective defender yac
- delta defender ypt
- coverage probabilities
- defender archetypes

and produces:

- defender_adjusted_targets
- defender_adjusted_yards
- defender_adjusted_tds

This submodel itself is an ML model (often ridge or XGBoost).

Its predictions feed into the ensemble.

4.5 Archetype Response Submodel (NEW in v2)
Learns the relationship:
offensive_archetype × defensive_archetype → expected adjustment
This is learned from data instead of rule-based matrices.
Useful for:
- WR archetypes vs coverage archetypes
- RB archetypes vs run-fit archetypes
- TE archetypes vs linebacker/safety mixes

This model generates:
- arch_response_yards_adj
- arch_response_targets_adj
- etc.

Also part of ensemble inputs.

4.6 Per-Stat Training
You must build one model per target statistic:
WR/TE stats:
- targets
- receptions
- receiving yards
- receiving TDs

RB stats:
- carries
- rush yards
- rush TDs
- receiving components

QB stats:
- attempts
- completions
- pass yards
- pass TDs
- interceptions

Each stat is its own ML model with its own:
- hyperparameters
- feature selection
- training metrics

There is no multi-target (multi-output) training because:
- targets, yards, TDs have different distributions
- different features matter
- different transformations are helpful

---

5. Label & Feature Preprocessing
5.1 Feature Scaling
Tree models → no scaling needed
Linear models → standard scaling (mean 0, std 1)
5.2 Log Transform for Skewed Stats
Receiving yards, rush yards, TDs:
$y_{transformed} = \log(1 + y)$
Then inverse after prediction:
$y_{pred} = \exp(y_{pred\_transformed}) - 1$

5.3 Handling Missing Values

- Linear model: impute with mean or median
- Boosting model: let them handle NaN internally

5.4 Categorical Encoding

Options:

1. One-Hot: position, team, opponent team
2. Embedding: optional for archetypes
3. Ordinal: not recommended

---

6. Hyperparameter Tuning

Use:

- manual tuning for small models
- grid search or random search for boosting models
- time-aware cross-validation
  - train on earlier weeks
  - validate on later weeks

Never use random K-fold CV.

---

7. Model Metrics

Evaluate on:

- MAE (Mean Absolute Error)
- RMSE
- R^2 (for interpretability)
- “Hit rate”:
  - Does model predict up/down movement correctly?
- Ranking metrics (Spearman correlation)
- Calibration:
  - Do predictions systematically under/over predict certain archetypes?
  - Certain defenses?
  - Certain defenders?

Critical insight:

A model can have good raw error but still be miscalibrated vs archetypes or defenders.

This must be detected in Phase 4.

---

8. Output Artifacts

All models must be:

- pickled
- versioned
- hash-labeled
- stored with metadata

Example:

```
models/
   targets/
       2025W06_ridge.pkl
       2025W06_xgb.pkl
   rec_yards/
       2025W06_ridge.pkl
       2025W06_xgb.pkl
```

...
metadata/
   2025W06_training_report.json
Metadata includes:
- feature lists
- hyperparameters
- training/validation metrics
- data_version
- model_version

---

9. Inference Pipeline (Using ML Models)
At inference time:
1. Load player_game_features for the target week.
2. Load all trained models.
3. Generate predictions per model and per stat.
4. Store outputs in:

ml_predictions_<data_version>
With columns:
- player_id
- game_id
- stat_name
- ridge_pred
- xgb_pred
- defender_submodel_pred
- archetype_submodel_pred

These will be consumed by the ensemble meta-model (Phase 6).

---

10. Weekly Retraining Workflow (Season Mode)
Each week:
1. New raw week data is ingested (Phase 1).
2. Features updated (Phase 2).
3. Baseline predictions updated (Phase 3).
4. Retrain ML models:
    - using full historical data up to week N-1
    - validate on prior-season holdout or rolling windows
5. Save model artifacts under new version.
6. Run inference for week N+1.

Full retraining ensures:
- consistency with new defenders
- updated archetype context
- current tendencies captured

---

11. Summary of Phase 4 v2
Phase 4 v2 transforms the original v1 idea (“train ML models for stats”) into a complete, industrial-grade ML pipeline.
New requirements integrated:
- defender-aware submodels
- archetype-response submodels
- baseline residual learning
- multi-model training for each stat

- leak-safe splits
- per-stat hyperparameter tuning
- versioned model artifacts
- multi-model output feeding into ensemble training

At the end of Phase 4 v2, the system has:

- multiple well-trained ML predictors per stat
- defender-aware models
- archetype-aware models
- all predictions standardized and ready for Phase 6 ensemble training