Predicting Targeted Receiver and Coverage Defender Movement During Ball Flight

Maximus Alvir and Ethan Taubman

Big Data Bowl 2026 – Prediction

“Ensemble of XGBoost + LightGBM on WR and DB centric feature sets to predict (x,y) while the ball is in the air.”

Introduction

The downfield pass creates the most volatile and high-leverage moments in football. Once the quarterback releases the ball, both the targeted receiver and coverage defenders react in ways that depend on leverage, spacing, and the anticipated landing point. The 2026 Big Data Bowl Prediction competition focuses on modeling this ball-in-air phase by providing player tracking data up to the release frame, the targeted receiver identity, and the pass landing location. The task is to predict player locations for each frame while the ball is in the air.

This project builds a supervised learning pipeline that predicts movement for two key roles: the targeted wide receiver (WR) and the primary coverage defender (DB). The model converts pre-throw tracking into engineered features that capture route momentum, defender proximity and leverage, and context like field position and down-distance. Predictions are generated as future displacements relative to the player's release position, then converted back into x,y coordinates for scoring.

The final approach uses an ensemble of gradient boosted decision trees (XGBoost + LightGBM). The ensemble was chosen because it performs well on structured tabular feature sets, generalizes reliably across games, and allows targeted iteration on feature design without requiring deep sequence training.

Objective: For each pass play, predict the (x,y) location of the targeted receiver and coverage defender for every frame after the ball is released until the ball arrives or the play ends (incomplete/catch). Tracking is recorded at 10 frames per second, meaning a typical ball flight of 2.5 seconds requires roughly 25 predicted frames.

Evaluation: Submissions are scored using Root Mean Squared Error (RMSE) between predicted and observed positions. In this project, predictions are produced per frame, and error is evaluated across both axes. The code computes an average RMSE across x and y by measuring squared error in both directions and scaling appropriately.

Modeling: Instead of predicting absolute future positions directly, the model predicts frame-level displacement:

$dx = x\text{target} - x\text{release}$

$\text{d}y = y\text{target} - y\text{release}$

This framing makes the learning problem easier because the model focuses on motion relative to the player's own starting point, rather than learning absolute field location patterns.

Data Pipeline and Cross-Validation Design

The Prediction competition provides two key files per week:

- Input file (input_2023_wXX.csv): tracking and context available up to the moment the quarterback releases the ball. This includes pre-throw movement, player roles (targeted receiver, defensive coverage, passer), and play-level details such as pass landing location.
- Output file (output_2023_wXX.csv): the "ball-in-air" frames after release that must be predicted. Each player has multiple rows per play, one per frame.

Our pipeline loads and stacks multiple weeks of training data using load_week() and load_train(). This matters because each play produces many training rows (frames), and scaling to more weeks increases the variety of route types, coverage reactions, and ball flight durations the model sees.

The core modeling challenge is that plays have variable ball-flight lengths. Some throws are quick, some hang longer, so the number of frames to predict changes. To solve this cleanly, the code defines the release frame as the last frame present in the input file for each play:

- release_frame := max(frame_id) per (game_id, play_id)
- frame_since_release := frame_id - release_frame

This does two things:

- It standardizes what "time zero" means across every play.
- It makes it easy to build time-aware features later, like t_rel (raw frame index) and t_norm (normalized by total frames in that play).

The model does not predict a single final location. It predicts every frame while the ball is in the air. That means each (game_id, play_id, nfl_id) becomes many training rows.

For each role (WR and DB), the pipeline:

1. Builds a single feature row per play describing the situation at release (plus short history).
2. Joins those features to the output frames, creating a full training table where each row corresponds to a future frame.
3. Defines targets as displacements from release:

- WR: dx = x_tgt - wr_rel_x, dy = y_tgt - wr_rel_y
- DB: dx = x_tgt - cb_rel_x, dy = y_tgt - cb_rel_y

This choice is important because it turns the model into predicting motion from a known start, which is generally easier than learning absolute positions.

A major risk in this competition is leakage caused by splitting randomly by rows. Since each play generates many frames, random splits can accidentally put the same play's frames in both train and validation. To prevent that, the code uses split_by_game(): This ensures the validation set represents new games the model has never seen, which better matches future weeks of the season.

The training rows represent different difficulty levels. Predicting the first few frames after release is usually easier. Predicting late frames is harder and often matters more because separation and contests happen later. To combat this, we create a time-based weighting function:

- Early frames get moderate weight
- Mid-flight frames get the highest weight
- Very late frames get reduced weight again

This weighting is telling the model: "optimize hardest where it matters." It's also consistent with how errors compound over time.

Coordinate Normalization and Geometry Layer

Raw tracking coordinates mix plays going left and right, which forces the model to learn two versions of every pattern (one for each direction). That increases complexity for no gain. This is why coordinate normalization matters. We built a function normalize_direction(), which standardizes the field so:

- plays going left are flipped to behave like plays going right
- x_std becomes a consistent "downfield" axis

This is one of the most important preprocessing steps because it ensures route breaks, leverage, and pursuit angles behave consistently across the full dataset.

After standardizing direction, the pipeline goes one step further and rotates coordinates into a player-centric frame.

WR model: rotate the defender positions relative to the WR using WR heading (wr_rel_dir)

DB model: rotate WR/QB/ball positions relative to the CB using CB heading (cb_rel_dir)

This helps the model because a "break inside" is always a break inside in a player's local frame and a defender "over the top" is always over the top in a CB-centric frame. The model learns movement responses to leverage more directly. This is why features like wr_cb_angle, wr_rx/wr_ry, and ball_rx/ball_ry are so valuable. They embed football geometry directly into the learning problem.

Another geometry feature is the WR-to-ball line distance feature, point_to_segment_dist(). It is used to compute distances from defenders to the segment connecting: WR release location → ball landing location. This is a strong proxy for whether the defender is "in phase" with the throw or in the passing lane. It's also a clean way to rank defenders without needing explicit coverage assignments.

Lastly, rather than including all defenders we select the K nearest defenders based on distance to the WR-to-ball line. Then you store them as wide features with dcast(). This helps keeps features stable across plays, forces the model to focus on the defenders most likely to involved, captures multi-defender pressure without using set-encoding or complex variation models.

Feature Engineering (WR and DB)

WR Model: The features are built around leverage, timing, and pursuit logic. Putting together the wide receiver features, we started with the release snapshot (who the player is at the moment of throw).

From wr_rel, we pull:

- position: (wr_rel_x, wr_rel_y)
- movement: speed, accel, direction, orientation (wr_rel_s, wr_rel_a, wr_rel_dir, wr_rel_o)
- ball landing location: (ball_land_x, ball_land_y)
- contextual field position: (absolute_yardline_number)
- player body metrics: height/weight z-scores (ht_in_z, wt_lb_z)

The core idea is that the WR's future movement depends heavily on their release momentum plus where the ball is going. Next, we included features that act as a lightweight proxy for route families and concept spacing without needing route labels.

- WR pre-snap location (wr_pre_x, wr_pre_y)
- split features (wr_split_left, wr_split_right)
- slot vs boundary indicator (wr_is_slot, wr_is_boundary)
- formation width based on offense spread at the earliest frame (formation_width)

Moving on to short history micro-features (last 3 frames), This helps over simple averages because it captures "what the WR was doing right before release," like:

- burst vs throttle-down (wr_speed_recent3, wr_accel_recent3)
- early break initiation (wr_ang_vel_recent3 (angular velocity proxy))
- direction change rate (wr_dir_change_recent3)

These features often help because real routes have sharp transitions right around the throw. We also added features for ball geometry and timing:

- distance to landing point: (land_dist)
- vector to ball: (wr_to_land_dx, wr_to_land_dy)
- angle to ball relative to heading: (wr_to_ball_angle)

- time to ball: (time_to_ball)

These features anchor the prediction: the WR's movement is constrained by where the ball is expected to arrive. Lastly, coverage leverage and pressure features:

- separation and angle to nearest defender: (wr_cb_dist, wr_cb_angle)
- defender closing speed (per defender): (def_close_1..3)
- WR-CB relative speed and closing: (wr_cb_rel_speed, wr_cb_closing_speed)
- defender density: (def_density)
- cushion and leverage proxy: (cushion, cb_leverage)

This is the heart of logical features. Through many trial and errors this is what we settled on in the end. WE believe it gives the model information about whether the WR is open early, being squeezed, or being carried as best as possible. We also included contextual features to help the model distinguish like quick game vs longer developing routes.

DB Model: The DB model is not just "same features, different player." It's a makeup of WR threat, QB position**,** and ball landing location

To start the features, we computed the DB release and micro-history of last-3-frame micro stats:

- cb_rel_s, cb_rel_a, cb_rel_dir, cb_rel_o
- cb_speed_recent3, cb_accel_recent3
- cb_dir_change_recent3, cb_ang_vel_recent3

This captures whether the DB is already opening hips, driving, or flat-footed at release. Next, the rotated threat positions. We rotate WR, QB, and ball landing location into DB's local frame:

- wr_rx, wr_ry
- qb_rx, qb_ry
- ball_rx, ball_ry

These features represent where the threats are in DB space. More features include distances, angles, lane positioning, velocity and time advantage features:

- CB-to-WR distance/angle (cb_to_wr_dist, cb_to_wr_angle)
- CB-to-ball distance/angle (cb_to_ball_dist, cb_to_ball_angle)
- distance to WR-ball line (dist_to_wr_ball_line)
- CB closing to WR (cb_closing_to_wr)
- relative speed (cb_wr_rel_speed)
- time-to-ball and time-to-WR (cb_time_to_ball, cb_time_to_wr)
- CB vs WR time advantage (cb_wr_time_advantage)

This is essentially a physics-based description of whether the CB is in position to contest, as well as predicting the CB's pursuit path during ball flight. Lastly, the assumption of the DB reading the play and leveraging themselves:

- facing features (cb_facing_wr, cb_facing_ball)
- inside leverage indicator (cb_has_inside_leverage)
- QB-facing-CB (qb_facing_cb)

This approximates whether the CB is "seeing it," in phase, and positioned correctly. This is a big differentiator in the project, as we did not re use WR features, but built another model for DB's.

Model Strategy and Ensemble Design

As a reference point, a simple constant-velocity baseline was implemented using the player's release speed and direction. This baseline assumes the player continues moving in a straight line at the same velocity throughout ball flight. While overly simplistic, the baseline provides an important sanity check and scale reference for RMSE. On validation data, the baseline produced RMSE values of 6.13 yards for WRs and 5.00 yards for DBs, confirming that motion assumptions are insufficient for modeling real route breaks, leverage adjustments, and pursuit behavior during the ball-in-air phase.

We chose to use gradient boosted models for Δx and Δy. The core predictive models use gradient boosted decision trees, trained separately for horizontal (dx) and vertical (dy) displacement. This design choice allows the model to learn different motion dynamics along each axis, which is important given that:

- Vertical movement (downfield) is more constrained by ball depth and timing.
- Horizontal movement is more sensitive to route breaks, leverage, and sideline constraints.

Two boosting frameworks were used:

- XGBoost, which tends to learn strong hierarchical splits and performs well on interaction-heavy features.
- LightGBM, which uses leaf-wise growth and often captures subtle non-linear patterns efficiently.

For each role (WR and DB), four models were trained:

- XGBoost Δx
- XGBoost Δy
- LightGBM Δx
- LightGBM Δy

All models were trained with early stopping on a game-held-out validation set. Hyperparameters were fixed to a proven configuration after early experimentation, prioritizing stability and generalization over marginal gains from aggressive tuning.

Each output frame represents a different prediction difficulty. Early frames are easier, while later frames are harder and more football relevant. To reflect this, training rows were weighted based on normalized time since release. Early frames received moderate weight. Mid-flight frames received the highest weight. Extremely late frames received slightly reduced weight to limit noise amplification. This weighting encourages the model to prioritize accuracy where separation, contests, and interceptions are most likely to occur.

Although XGBoost and LightGBM produced similar aggregate RMSE values, their error patterns were not identical. In some plays, one model tracked breaks more accurately, while the other handled smooth pursuit better.

Predictions were merged using a weighted ensemble in order to take advantage of this similarity:

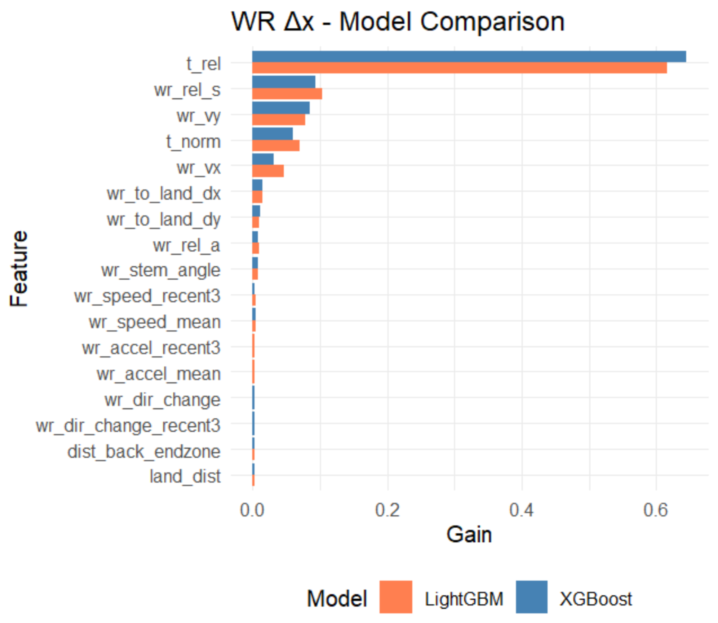
- Final displacement = w * XGBoost + (1 − w) * LightGBM
- The ensemble weight w was tuned on the validation set only via a grid search.

This tuning was performed separately for WR and CB models. Importantly, the tuned weights were not re-fit on the test set, preventing leakage and preserving the integrity of the holdout evaluation.

Model Behavior and Feature Attribution

This section explains what the models are using to make predictions, using feature gain comparisons between XGBoost and LightGBM for each target.

1. WR Δx (Horizontal Movement)

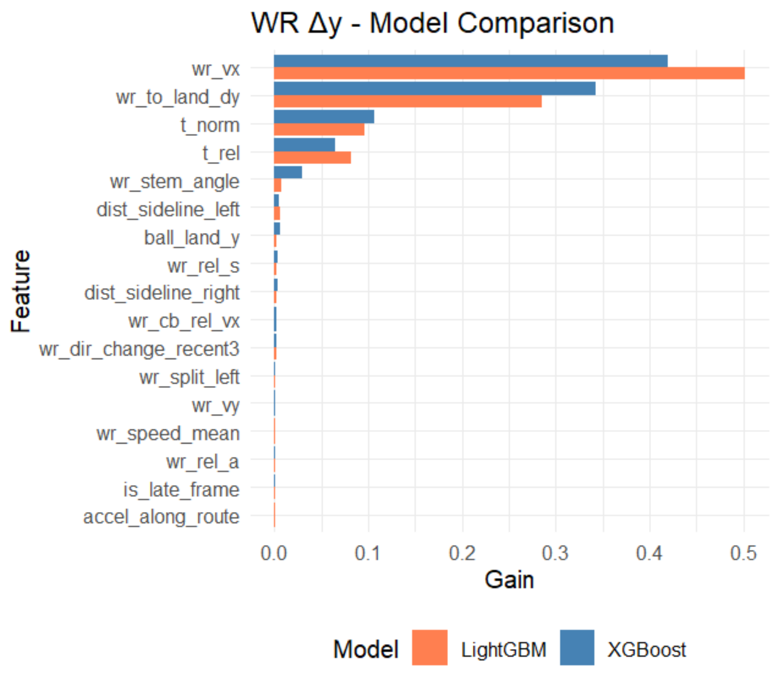


The WR Δx models are dominated by temporal features, particularly t_rel and t_norm. This indicates that horizontal displacement is strongly governed by how long the ball has been in the air, which aligns with football intuition. Most horizontal separation occurs progressively as the play unfolds rather than instantaneously at release.

Receiver release speed (wr_rel_s) and horizontal velocity (wr_vx) are the next most important features, reinforcing that lateral movement is largely driven by early momentum. Directional micro-features such as recent angular velocity and stem angle contribute to smaller but consistent signals, helping the model identify subtle break behavior.

Notably, both XGBoost and LightGBM rank the same features highly, but XGBoost places slightly more emphasis on raw time since release, while LightGBM distributes gain more evenly across velocity-derived terms. This difference in emphasis helps explain why the ensemble outperforms either model individually.

2. WR Δy (Vertical Movement)

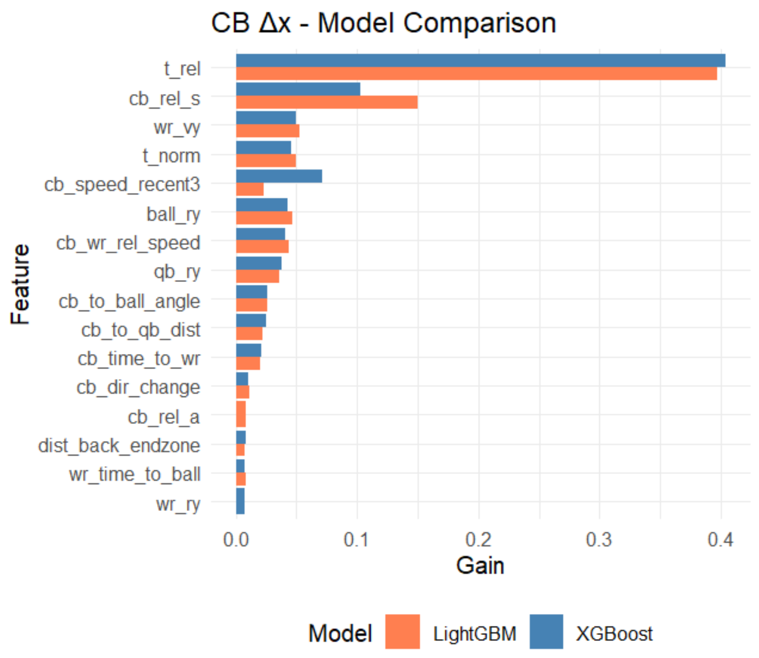


Vertical WR movement shows a different structure. Here, ball-relative geometry dominates. The most important features are wr_vx and wr_to_land_dy, indicating that downfield motion is heavily constrained by the ball's landing location and the receiver's forward velocity at release.

Time features (t_rel, t_norm) remain important but play a secondary role compared to spatial constraints. This illustrates how WRs are essentially "locked in" by throw depth and cannot independently change their vertical movement late in the play.

Sideline distance and alignment features appear with modest gain, suggesting that boundary constraints subtly shape vertical movement without being primary drivers.

3. DB Δx (Horizontal Movement)

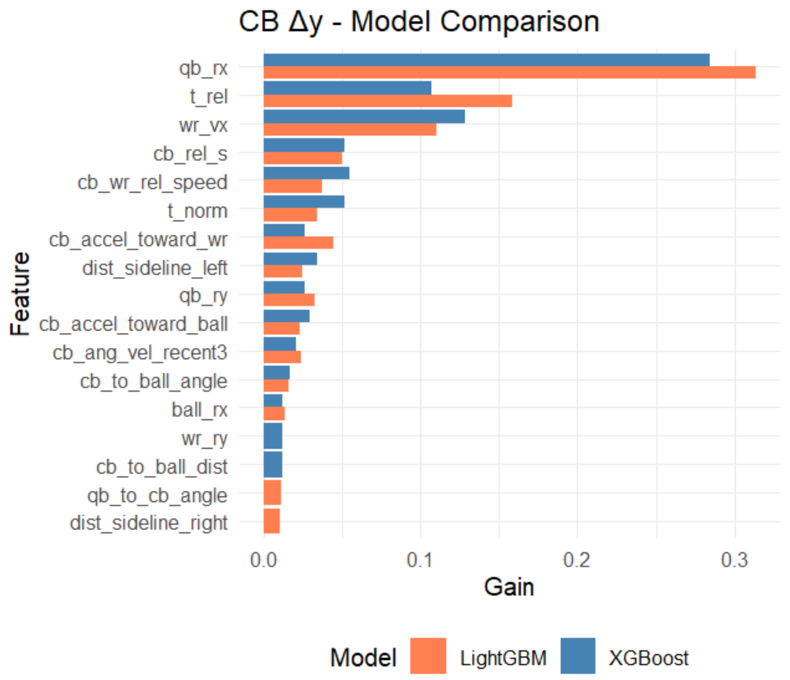


For DB Δx, the dominant feature remains t_rel, but unlike WRs, contextual and opponent-driven features play a larger role. DB release speed, recent acceleration, and relative speed to the WR (cb_wr_rel_speed) all appear prominently.

Ball location (ball_ry) and QB-relative positioning also enter the model, indicating that DBs adjust horizontal pursuit not only based on the receiver, but also based on where the ball is expected to arrive.

This reinforces an important behavioral difference: DBs are reacting rather than executing a pre-planned path. Their horizontal movement reflects pursuit geometry more than route intent.

4. DB Δy (Vertical Movement)

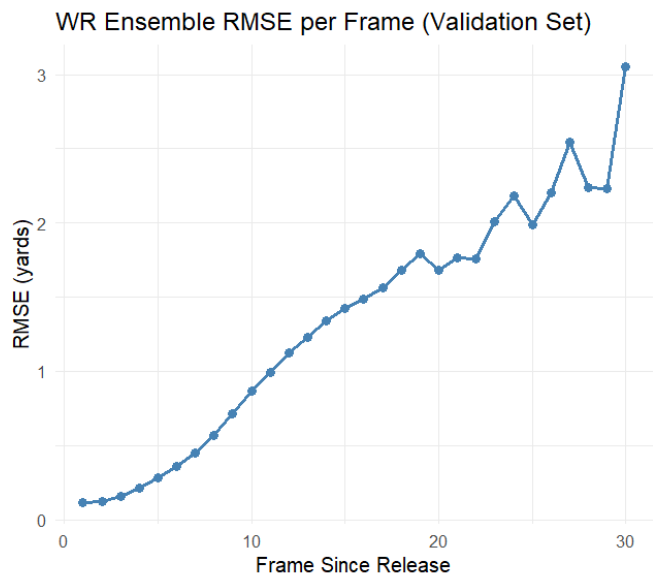


DB Δy is the most complex target. The top feature, qb_rx, indicates that quarterback-relative positioning strongly influences how defenders adjust vertically after the throw. This suggests the model is implicitly learning cues related to throw anticipation and depth reading.

Time since release remains important, but features such as CB–WR relative speed, CB acceleration toward the WR or ball, and angular velocity all contribute meaningfully. These features collectively describe decision transitions, such as opening hips, driving downhill, or bailing vertically.

The broader feature mix compared to WR Δy highlights why DB prediction is harder: defenders must choose between multiple plausible responses.

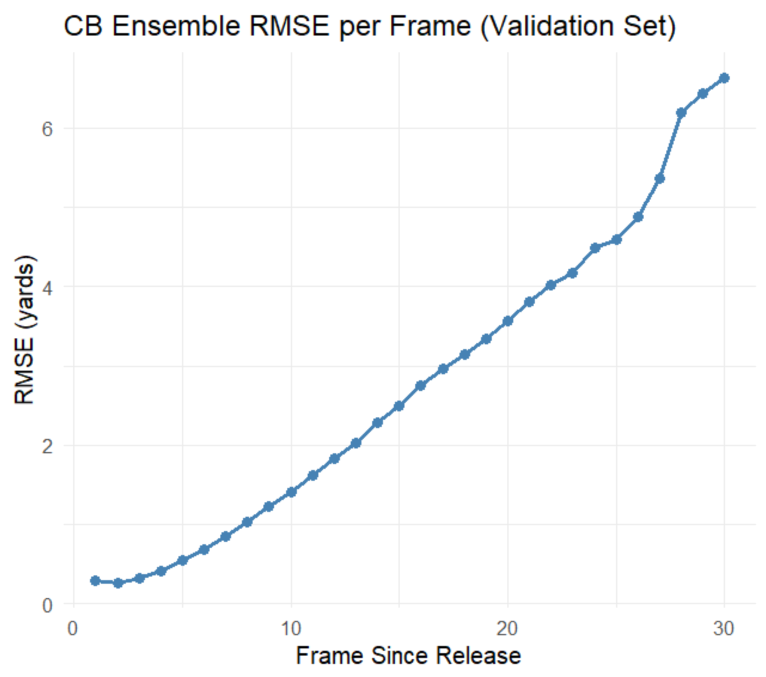
Ensemble RMSE by Frame



WR RMSE increases steadily as frame index grows. Early frames immediately after release show extremely low error (near zero), reflecting that short-term motion is highly predictable given release momentum.

As the ball remains in the air, error increases roughly linearly, reaching just over 3 yards by frame ~30. This pattern is expected. Small early errors compound over time, and late frames often include abrupt movements such as throttling down or adjusting to ball trajectory.

Crucially, there is no sharp instability or explosion in error. The smooth growth indicates that the model produces realistic, continuous trajectories rather than erratic jumps.



DB error grows faster and reaches higher levels, exceeding 6 yards by late frames. This reflects the fundamentally different task faced by defenders.

Unlike receivers, DBs may:

- hesitate before committing
- switch from man pursuit to ball attack
- redirect sharply late in the play.

The smooth but steeper error curve suggests the model is capturing average pursuit behavior while struggling most with late-play decision forks. Importantly, the curve remains monotonic and stable, which indicates the model is not diverging or overfitting to specific frame ranges.

Results, Stability, Interpretation

Using the ensemble of XGBoost and LightGBM:

WR RMSE:

- Validation: 0.526
- Test: 0.571
- Val–Test gap: +0.045

DB RMSE:

- Validation: 1.07

- Test: 1.05
- Val–Test gap: −0.023

The small gaps indicate strong generalization and confirm that game-based splitting and prior handling prevented leakage.

The WR prediction is more accurate than the DB prediction because DBs operate under uncertainty. They must infer throw depth, track the ball and receiver, while the WR has route intent and all the momentum. The higher DB RMSE is expected and does not indicate a weaker model.

The model is learning time conditioned movement, geometry driven constraints, and role-specific response patterns. By establishing a balance between LightGBM's responsiveness to velocity and positional complexity and XGBoost's greater reliance on temporal structure, the ensemble enhances performance.

These predictions enable downstream football analysis such as:

- Estimating separation and contest windows at arrival
- Comparing expected vs actual defender reactions
- Identifying routes that force late DB indecision
- Evaluating coverage discipline independent of catch outcome

This project demonstrates that player movement during the ball-in-air phase can be modeled accurately using geometry-aware features, role-specific representations, and a carefully validated ensemble approach. By anchoring predictions to the moment of release, normalizing spatial context, and separating receiver intent from defender response, the model captures realistic movement patterns rather than relying on static assumptions or overfit trajectories. The small validation–test gaps and stable frame-level error growth indicate strong generalization to unseen games, aligning well with the live leaderboard structure of the Big Data Bowl. More broadly, this work shows that interpretable, logical machine learning can meaningfully describe how receivers and defenders react once the ball is thrown, providing a foundation for future extensions into trajectory simulation, contest probability modeling, and applied football decision-making.