Quantifying Defensive Impact on Pass Outcomes Using NFL Tracking Data

Table of contents

```r
knitr::opts_chunk$set(echo = TRUE)
library(dplyr)
library(data.table)
library(xgboost)
library(tidyr)
library(tidyverse)
library(knitr)
library(pROC)
set.seed(1121)
oi_colors <-
  palette.colors(palette = "Okabe-Ito")
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
```

0.1 Motivation

Player-tracking data has transformed modern football analytics by allowing teams, analysts, and researchers to quantify aspects of the game that were previously impossible to measure. Despite this advancement, one question has remained difficult to answer: how much impact does a defender actually have on a pass while the ball is in the air? Traditional defensive metrics, such as forced incompletions, interceptions, or completion percentage allowed, only capture the final outcome of the play. They ignore the dynamic interaction between the receiver and the defender throughout the ball's flight.

In this project, we use the NFL Big Data Bowl tracking data to model catch probability frame-by-frame, allowing us to measure how defensive coverage changes the likelihood of a completion in real time. By computing the change in catch probability from release to arrival (CP_DROP), we can quantify defender impact in a way that accounts for separation, leverage, route type, and ball trajectory. This approach provides a richer, more continuous understanding of coverage performance, offering a new analytic tool for evaluation individual defenders and defensive schemes.

0.2 Data Preparation

0.2.1 Load and combine all input tracking files

Our first step is to load all of the input tracking data files for the 2023 season and stack them into a single table. This gives us every pre-throw frame for all tracked players, which is important for joining to the ball-flight data and the supplementary play information later on.

```r
input_files <- list.files(
  path = "C:/Users/blain/Downloads/train",
  pattern = "^input_2023_w[0-9]{2}.csv$",
  full.names = TRUE
)
full_input <- input_files |>
  lapply(fread) |>
  bind_rows()
```

0.2.2 Load and combine all output tracking files

Next, we load the output tracking files, which contain player locations during the ball's flight. Just like with the input files, we combine all weeks into one large dataset so that we can work with every tracked play at once.

```r
output_files <- list.files(
  path = "C:/Users/blain/Downloads/train",
  pattern = "^output_2023_w[0-9]{2}.csv$",
  full.names = TRUE
)

full_output <- output_files |>
  lapply(fread) |>
  bind_rows()
```

0.2.3 Load the supplementary play-level data

We also load the supplementary CSV, which includes information about pass results, routes, coverage types, and whether plays were nullified by penalties. This metadata will let us define outcomes (caught vs. not caught) and stratify results by route and coverage later on.

```r
meta_data <- readr::read_csv("supplementary_data.csv")
```

0.2.4 Filter to players to predict on the first output frame

From the full input tracking data, we only keep the players flagged as "player_to_predict" on the first output frame of each play. This gives us one row per relevant player-play combination, including the targeted receiver and coverage defenders, along with ball landing location and context.

```r
filtered_input <- full_input |>
  filter(frame_id == 1, player_to_predict == TRUE) |>
  select(game_id, play_id, nfl_id, player_height, player_position, player_side,
         player_role, num_frames_output, ball_land_x, ball_land_y, player_name)
glimpse(filtered_input)
```

```
Rows: 46,045
Columns: 11
$ game_id           <int> 2023090700, 2023090700, 2023090700, 2023090700, 2023~
$ play_id           <int> 101, 101, 101, 194, 194, 194, 194, 194, 219, 219, 21~
$ nfl_id            <int> 46137, 52546, 44930, 44888, 55910, 53953, 54653, 413~
$ player_height     <chr> "6-1", "6-1", "6-3", "6-3", "6-0", "5-11", "5-11", "~
$ player_position   <chr> "SS", "CB", "WR", "OLB", "SS", "CB", "ILB", "RB", "O~
$ player_side       <chr> "Defense", "Defense", "Offense", "Defense", "Defense~
$ player_role       <chr> "Defensive Coverage", "Defensive Coverage", "Targete~
$ num_frames_output <int> 21, 21, 21, 9, 9, 9, 9, 9, 8, 8, 8, 16, 16, 16, 7, 7~
$ ball_land_x       <dbl> 63.26, 63.26, 63.26, 84.94, 84.94, 84.94, 84.94, 84.~
$ ball_land_y       <dbl> -0.22, -0.22, -0.22, 21.75, 21.75, 21.75, 21.75, 21.~
$ player_name       <chr> "Justin Reid", "L'Jarius Sneed", "Josh Reynolds", "A~
```

0.2.5 Join input and output tracking data

We now join the output tracking frames to the filtered input using game, play, and player IDs. This attaches the ball landing coordinates and player roles to every frame during the ball's flight, which we will later use to identify targeted receivers, defenders and the catch point.

```r
joined_input_output <- full_output |> inner_join(filtered_input,
                                                 by = c("game_id", "play_id",
                                                        "nfl_id"))
glimpse(joined_input_output |> select(frame_id, x, y))                    (1)
```

(1) showing changed columns

```
Rows: 562,936
Columns: 3
$ frame_id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18~
$ x        <dbl> 56.22, 56.63, 57.06, 57.48, 57.91, 58.34, 58.75, 59.14, 59.51~
$ y        <dbl> 17.28, 16.88, 16.46, 16.02, 15.56, 15.10, 14.57, 14.01, 13.41~
```

0.2.6 Add pass result, route and coverage information

Next, we bring in the supplementary metadata, which adds pass results, routes, coverage labels, pass length, and penalty flags to each player-frame row. This allows us to define a binary "caught" outcome and later slice results by route and coverage scheme.

```
fully_joined <- joined_input_output |>
  left_join(meta_data |>
              select(game_id, play_id, pass_result, route_of_targeted_receiver,
                     team_coverage_man_zone, team_coverage_type, pass_length,
                     play_nullified_by_penalty), by = c("game_id", "play_id"))
```

0.2.7 Keep valid pass plays and define catch indicator

We restrict out data to plays where the pass result is recorded as a catch or an incomplete pass type and where the play was not nullified by a penalty. We then define a binary caught variable indicating whether the pass was completed.

```
fully_joined <- fully_joined |>
  filter(
    pass_result %in% c("C", "I", "IN"),
    play_nullified_by_penalty == "N"
  ) |>
  mutate(
    caught = ifelse(pass_result == "C", 1, 0)
  )
glimpse(fully_joined |> select(pass_result, route_of_targeted_receiver,
                               team_coverage_man_zone, team_coverage_type,
                               pass_length, play_nullified_by_penalty,
                               caught))                                      (1)
```

(1) showing changed columns

```
Rows: 562,922
Columns: 7
$ pass_result                <chr> "I", "I", "I", "I", "I", "I", "I", "I", "I"~
$ route_of_targeted_receiver <chr> "CORNER", "CORNER", "CORNER", "CORNER", "CO~
$ team_coverage_man_zone     <chr> "ZONE_COVERAGE", "ZONE_COVERAGE", "ZONE_COV~
$ team_coverage_type         <chr> "COVER_2_ZONE", "COVER_2_ZONE", "COVER_2_ZO~
$ pass_length                <dbl> 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22,~
$ play_nullified_by_penalty  <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N"~
$ caught                     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

0.3 Feature Engineering

0.3.1 Seperate targeted recievers and coverage defenders

From the full joined dataset, we split off two key groups of players, the targeted receivers and the defensive coverage players. We also rename their x, y coordinates and IDs to clearly distinguish between receiver and defender positions in later joins.

```r
wr_frames <- fully_joined |>
  filter(player_role == "Targeted Receiver")

wr_frames <- wr_frames |> rename(
  nfl_id_wr = nfl_id,
  x_wr = x,
  y_wr = y
)

def_frames <- fully_joined |>
  filter(player_role == "Defensive Coverage")

def_frames <- def_frames |>
  rename(
    nfl_id_def = nfl_id,
    x_def = x,
    y_def = y
  )
```

0.3.2 Compute movement and ball distance features for WRs and defenders

We now engineer frame-level features that describe how the targeted receiver and defenders move relative to the catch point. For each group, we compute distance to the ball, movement between frames, approximate speed and changes in distance to the ball. These features are key inputs for predicting catch probability.

```r
wr_frames <- wr_frames |>
  mutate(
    dist_wr_ball = sqrt((x_wr - ball_land_x)^2 + (y_wr - ball_land_y)^2),
    flight_fraction = frame_id / num_frames_output
  ) |>
  group_by(game_id, play_id, nfl_id_wr) |>
  arrange(frame_id) |>
  mutate(
    dist_to_prev_frame = sqrt((x_wr - lag(x_wr, default = first(x_wr)))^2 +
                              (y_wr - lag(y_wr, default = first(y_wr)))^2),
    v_wr = dist_to_prev_frame / 0.1,
    delta_dist_wr_ball = dist_wr_ball - lag(dist_wr_ball,
                                            default = first(dist_wr_ball))
  ) |> ungroup() |> filter(frame_id != 1)


def_frames <- def_frames |>
  mutate(
    dist_def_ball = sqrt((x_def - ball_land_x)^2 + (y_def - ball_land_y)^2)
  ) |> group_by(game_id, play_id, nfl_id_def) |>
  arrange(frame_id) |>
  mutate(
    dist_to_prev_frame = sqrt((x_def - lag(x_def, default = first(x_def)))^2 +
                                (y_def - lag(y_def, default = first(y_def)))^2),
    v_def = dist_to_prev_frame / 0.1,
    delta_dist_def_ball = dist_def_ball - lag(dist_def_ball, default =
                                                  first(dist_def_ball))
    ) |> ungroup() |> filter(frame_id != 1)
def_frames |> arrange(play_id)
```

```
# A tibble: 370,639 x 25
   game_id play_id nfl_id_def frame_id x_def y_def player_height player_position
     <dbl>   <dbl>      <int>    <int> <dbl> <dbl> <chr>         <chr>
 1  2.02e9      54      43306        2  46.8  5.56 6-1           FS
 2  2.02e9      54      46698        2  44.4 14.2  6-0           CB
 3  2.02e9      54      43306        3  47.4  5.12 6-1           FS
 4  2.02e9      54      46698        3  44.3 13.3  6-0           CB
 5  2.02e9      54      43306        4  48.1  4.7  6-1           FS
 6  2.02e9      54      46698        4  44.2 12.3  6-0           CB
 7  2.02e9      54      43306        5  48.8  4.3  6-1           FS
 8  2.02e9      54      46698        5  44.0 11.4  6-0           CB
 9  2.02e9      54      43306        6  49.5  3.9  6-1           FS
10  2.02e9      54      46698        6  43.9 10.4  6-0           CB
# i 370,629 more rows
# i 17 more variables: player_side <chr>, player_role <chr>,
#   num_frames_output <int>, ball_land_x <dbl>, ball_land_y <dbl>,
#   player_name <chr>, pass_result <chr>, route_of_targeted_receiver <chr>,
#   team_coverage_man_zone <chr>, team_coverage_type <chr>, pass_length <dbl>,
#   play_nullified_by_penalty <chr>, caught <dbl>, dist_def_ball <dbl>,
#   dist_to_prev_frame <dbl>, v_def <dbl>, delta_dist_def_ball <dbl>
```

0.3.3 Join WR and defender frames and compute seperation

We next join each targeted receiver frame with all defender frames from the same play and frame. This gives us every receiver-defender paring over time. From this, we compute the straight-line seperation between the receiver and each defender at each frame.

```
wr_def_all <- wr_frames |>
  inner_join(def_frames,
             by = c("game_id", "play_id", "frame_id"))

wr_def_all <- wr_def_all |>
  mutate(dist_wr_def = sqrt((x_wr - x_def)^2 + (y_wr - y_def)^2))

wr_def_all
```

```
# A tibble: 370,639 x 49
      game_id play_id nfl_id_wr frame_id  x_wr  y_wr player_height.x
        <dbl>   <dbl>     <int>    <int> <dbl> <dbl> <chr>
 1 2023090700     101     44930        2  54.0  13.8 6-3
 2 2023090700     101     44930        2  54.0  13.8 6-3
 3 2023090700     194     41325        2  87.9  21.8 5-9
 4 2023090700     194     41325        2  87.9  21.8 5-9
 5 2023090700     194     41325        2  87.9  21.8 5-9
 6 2023090700     194     41325        2  87.9  21.8 5-9
 7 2023090700     219     53591        2  75.3  10.2 6-4
 8 2023090700     219     53591        2  75.3  10.2 6-4
 9 2023090700     361     38696        2  34.2  48.6 6-2
10 2023090700     361     38696        2  34.2  48.6 6-2
# i 370,629 more rows
# i 42 more variables: player_position.x <chr>, player_side.x <chr>,
#   player_role.x <chr>, num_frames_output.x <int>, ball_land_x.x <dbl>,
#   ball_land_y.x <dbl>, player_name.x <chr>, pass_result.x <chr>,
#   route_of_targeted_receiver.x <chr>, team_coverage_man_zone.x <chr>,
#   team_coverage_type.x <chr>, pass_length.x <dbl>,
#   play_nullified_by_penalty.x <chr>, caught.x <dbl>, dist_wr_ball <dbl>, ...
```

0.3.4 Compute nearest defender distance at each frame

For each targeted receiver frame, we summarize over defenders to find the smallest receiver-defender separation. This gives a frame-level measure of how tightly covered the receiver is, regardless of which specific defender is closest.

```r
wr_def_agg <- wr_def_all |>
  group_by(game_id, play_id, nfl_id_wr, frame_id) |>
  summarise(
    nearest_def_dist = min(dist_wr_def, na.rm = TRUE),

    .groups = "drop"

  )
wr_def_agg
```

```
# A tibble: 138,732 x 5
      game_id play_id nfl_id_wr frame_id nearest_def_dist
        <dbl>   <dbl>     <int>    <int>            <dbl>
 1 2023090700     101     44930        2             4.09
 2 2023090700     101     44930        3             3.75
 3 2023090700     101     44930        4             3.44
 4 2023090700     101     44930        5             3.18
 5 2023090700     101     44930        6             2.99
 6 2023090700     101     44930        7             2.80
 7 2023090700     101     44930        8             2.63
 8 2023090700     101     44930        9             2.48
 9 2023090700     101     44930       10             2.36
10 2023090700     101     44930       11             2.27
# i 138,722 more rows
```

0.3.5 Identify the primary defender at the catch point

We define the "arrival" frame for each target as the last output frame for that receiver. Among all the defenders at that frame, we choose the one with the smallest seperation as the primary defender. This allows us to attribute the change in catch probability to the defender who is closest when the ball arrives.

```
arrival <- wr_frames |>
  group_by(game_id, play_id, nfl_id_wr) |>
  summarise(arrival_frame = max(frame_id), .groups = "drop")

wr_def_arrival <- wr_def_all |>
  inner_join(arrival,
             by = c("game_id", "play_id", "nfl_id_wr")) |>
  filter(frame_id == arrival_frame)



closest_defender <- wr_def_arrival |>
  group_by(game_id, play_id, nfl_id_wr) |>
  slice_min(dist_wr_def, n=1, with_ties=FALSE) |>
  select(game_id, play_id, nfl_id_wr, nfl_id_def)


primary_defender_frames <- wr_def_all |>
  inner_join(closest_defender,
             by = c("game_id", "play_id", "nfl_id_wr", "nfl_id_def")) |>
  rename(primary_nfl_id_def = nfl_id_def) |>
  select(
    game_id, play_id, frame_id, nfl_id_wr, primary_nfl_id_def,
    dist_wr_def, x_def, y_def, v_def, delta_dist_def_ball
    )
```

0.3.6 Build the final frame-level feature set

We construct our modeling dataset by merging the receiver features with the primary defender frames. We also add a feature capturing the change in receiver-defender distance between consecutive frames. Finally, we remove unused columns to keep the modeling data compact and focused.

```
features <- wr_frames |>
  left_join(primary_defender_frames,
```

```r
            by = c("game_id", "play_id", "nfl_id_wr", "frame_id"))

features <- features |>
  group_by(game_id, play_id, nfl_id_wr) |>
  arrange(frame_id) |>
  mutate(
    delta_dist_wr_def = dist_wr_def - lag(dist_wr_def),
    delta_dist_wr_def = ifelse(is.na(delta_dist_wr_def), 0, delta_dist_wr_def)
  ) |> ungroup()

features_final <- features |> select(-player_height, -player_position,
                                     -player_side, -player_role, -ball_land_x,
                                     -ball_land_y, -pass_result,
                                     -num_frames_output,
                                     -play_nullified_by_penalty, -player_name,
                                     -delta_dist_wr_def)
colnames(features_final)
```

```
 [1] "game_id"                     "play_id"
 [3] "nfl_id_wr"                   "frame_id"
 [5] "x_wr"                        "y_wr"
 [7] "route_of_targeted_receiver" "team_coverage_man_zone"
 [9] "team_coverage_type"          "pass_length"
[11] "caught"                      "dist_wr_ball"
[13] "flight_fraction"             "dist_to_prev_frame"
[15] "v_wr"                        "delta_dist_wr_ball"
[17] "primary_nfl_id_def"          "dist_wr_def"
[19] "x_def"                       "y_def"
[21] "v_def"                       "delta_dist_def_ball"
```

```r
features_final |> arrange(play_id)
```

```
# A tibble: 146,239 x 22
      game_id play_id nfl_id_wr frame_id  x_wr  y_wr route_of_targeted_receiver
        <dbl>   <dbl>     <int>    <int> <dbl> <dbl> <chr>
 1 2023100806      54     53456        2  43.3 12.0  CROSS
 2 2023100806      54     53456        3  43.2 11.2  CROSS
 3 2023100806      54     53456        4  43.0 10.3  CROSS
 4 2023100806      54     53456        5  42.8  9.41 CROSS
 5 2023100806      54     53456        6  42.7  8.54 CROSS
 6 2023100806      54     53456        7  42.6  7.69 CROSS
 7 2023091006      55     54517        2  40.1 33.8  HITCH
 8 2023091011      55     53579        2  30.4 37.3  ANGLE
 9 2023091012      55     42412        2  39.3  9.26 HITCH
10 2023091702      55     54604        2  80.9 35.4  HITCH
# i 146,229 more rows
# i 15 more variables: team_coverage_man_zone <chr>, team_coverage_type <chr>,
#   pass_length <dbl>, caught <dbl>, dist_wr_ball <dbl>, flight_fraction <dbl>,
#   dist_to_prev_frame <dbl>, v_wr <dbl>, delta_dist_wr_ball <dbl>,
#   primary_nfl_id_def <int>, dist_wr_def <dbl>, x_def <dbl>, y_def <dbl>,
#   v_def <dbl>, delta_dist_def_ball <dbl>
```

0.3.7 Clean categorical variables and filter plays

We fill in missing route and coverage labels with an "UNKNOWN" category and convert them to factors. We also ensure the outcome is numeric for modeling, keep only rows where a primary defender is identified, and filter to passes of at least 10 yards downfield.

```r
features_final <- features_final %>%
  mutate(
    route_of_targeted_receiver = replace_na(route_of_targeted_receiver, "UNKNOWN"),
    team_coverage_man_zone     = replace_na(team_coverage_man_zone, "UNKNOWN"),
    team_coverage_type         = replace_na(team_coverage_type, "UNKNOWN")
  )

features_final <- features_final %>%
    mutate(
        route_of_targeted_receiver = as.factor(route_of_targeted_receiver),
        team_coverage_man_zone = as.factor(team_coverage_man_zone),
        team_coverage_type = as.factor(team_coverage_type),
        caught = as.numeric(caught)
    )
features_final <- features_final |>
  drop_na(primary_nfl_id_def)|> filter(pass_length >= 10)
features_final |> arrange(game_id, play_id)
```

```
# A tibble: 74,181 x 22
      game_id play_id nfl_id_wr frame_id  x_wr  y_wr route_of_targeted_receiver
        <dbl>   <dbl>     <int>    <int> <dbl> <dbl> <fct>
 1 2023090700     101     44930        2  54.0  13.8 CORNER
 2 2023090700     101     44930        3  54.7  13.5 CORNER
 3 2023090700     101     44930        4  55.4  13.3 CORNER
 4 2023090700     101     44930        5  56.1  13.0 CORNER
 5 2023090700     101     44930        6  56.7  12.6 CORNER
 6 2023090700     101     44930        7  57.4  12.1 CORNER
 7 2023090700     101     44930        8  57.9  11.7 CORNER
 8 2023090700     101     44930        9  58.4  11.2 CORNER
 9 2023090700     101     44930       10  59.0  10.6 CORNER
10 2023090700     101     44930       11  59.4  10.0 CORNER
# i 74,171 more rows
# i 15 more variables: team_coverage_man_zone <fct>, team_coverage_type <fct>,
#   pass_length <dbl>, caught <dbl>, dist_wr_ball <dbl>, flight_fraction <dbl>,
#   dist_to_prev_frame <dbl>, v_wr <dbl>, delta_dist_wr_ball <dbl>,
#   primary_nfl_id_def <int>, dist_wr_def <dbl>, x_def <dbl>, y_def <dbl>,
#   v_def <dbl>, delta_dist_def_ball <dbl>
```

0.4 Modeling

0.4.1 Convert features to XGBoost matrix format

We encode the modeling dataset into a numeric matrix suitable for XGBoost. This includes one-hot encoding of categorical variables and dropping identifier columns such as game IDs, play IDs, frame IDs and player IDs that we do not want the model to use directly.

```r
play_ids <- features_final %>%

  distinct(game_id, play_id)

train_play_idx <- sample(seq_len(nrow(play_ids)), size = 0.8 * nrow(play_ids))

train_plays <- play_ids[train_play_idx, ]
test_plays <- play_ids[-train_play_idx, ]


train_df <- features_final %>%
  inner_join(train_plays, by = c("game_id", "play_id"))

test_df <- features_final %>%
  inner_join(test_plays, by = c("game_id", "play_id"))

Train_x <- model.matrix(
  caught ~ . - game_id - play_id - frame_id - nfl_id_wr - primary_nfl_id_def - 1,
  data = train_df
)
Train_y <- train_df$caught

dtrain <- xgb.DMatrix(data = Train_x, label = Train_y)

Test_x <- model.matrix(
  caught ~ . - game_id - play_id - frame_id - nfl_id_wr - primary_nfl_id_def - 1,
  data = test_df
)
Test_y <- test_df$caught

dtest <- xgb.DMatrix(data = Test_x, label = Test_y)
```

0.4.2 Train an XGBoost model to predict catch probability

We train a gradient-boosted decision tree model with binary logistic objective to estimate catch probability at each frame. The model uses our engineered features describing receiver and defender movement, separation, route and coverage to learn patterns associated with catches versus incompletions.

```r
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 8,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  seed = 1121
)

watchlist <- list(
  train = dtrain,
  eval  = dtest
)

xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 500,
  early_stopping_rounds = 25,
  watchlist = watchlist,
  verbose = 0
)
```

```
Warning in xgb.train(params = params, data = dtrain, nrounds = 500,
early_stopping_rounds = 25, : xgb.train: `seed` is ignored in R package.  Use
`set.seed()` instead.
```

0.4.3 Inspect training logloss

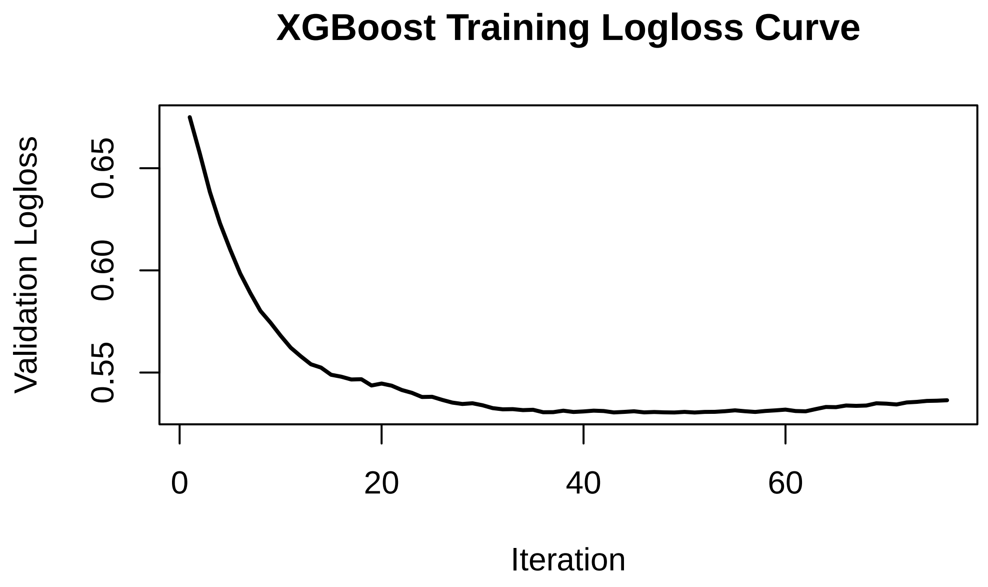
We then inspect the first and last few iterations of logloss and visualize how it changes over training.

```r
eval_log <- xgb_model$evaluation_log

eval_log_preview <- rbind(
  head(eval_log, 3),
  tail(eval_log, 3)
)

eval_log_preview
```

```
    iter train_logloss eval_logloss
   <num>         <num>        <num>
```

```
1:     1     0.6707987    0.6749947
2:     2     0.6486319    0.6571477
3:     3     0.6266996    0.6383130
4:    74     0.3173219    0.5361151
5:    75     0.3157997    0.5362232
6:    76     0.3150348    0.5364469
```

```r
plot(
eval_log$iter,
eval_log$eval_logloss,
type = "l",
lwd = 2,
xlab = "Iteration",
ylab = "Validation Logloss",
main = "XGBoost Training Logloss Curve"
)
```



Our log loss is .53, this is because in air catch probability can be difficult to determine due to things such as

- The data does not track more granular things like hand position, body position, or timing.
- The interactions between the receiver and defender happen on a faster frame rate than what is given.
- Many outcomes are still highly variable for the receiver, such as just them dropping a ball even if it was perfectly catchable.

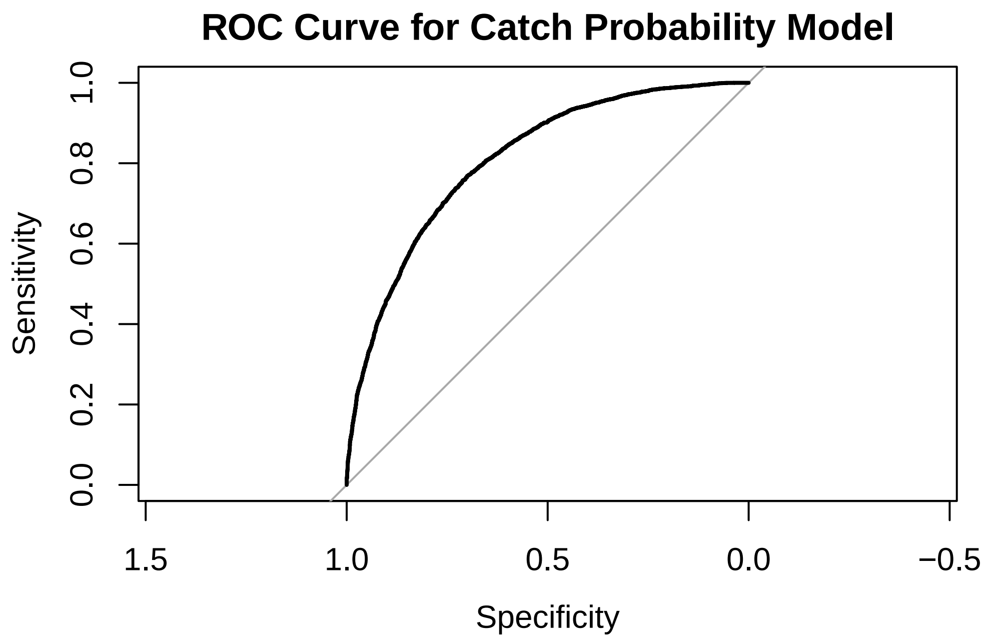
  Therefore, even strong models will not achieve extremely low error. Do not need to perfectly predict catches, but to generate consistent probabilities that allow us to measure defensive impact through CP_DROP.

0.4.4 ROC/AUC Evaluation

```r
test_pred <- predict(xgb_model, dtest)

roc_obj <- roc(Test_y, test_pred)
```

```
Setting levels: control = 0, case = 1

Setting direction: controls < cases
```

```r
plot(roc_obj, main = "ROC Curve for Catch Probability Model")
```



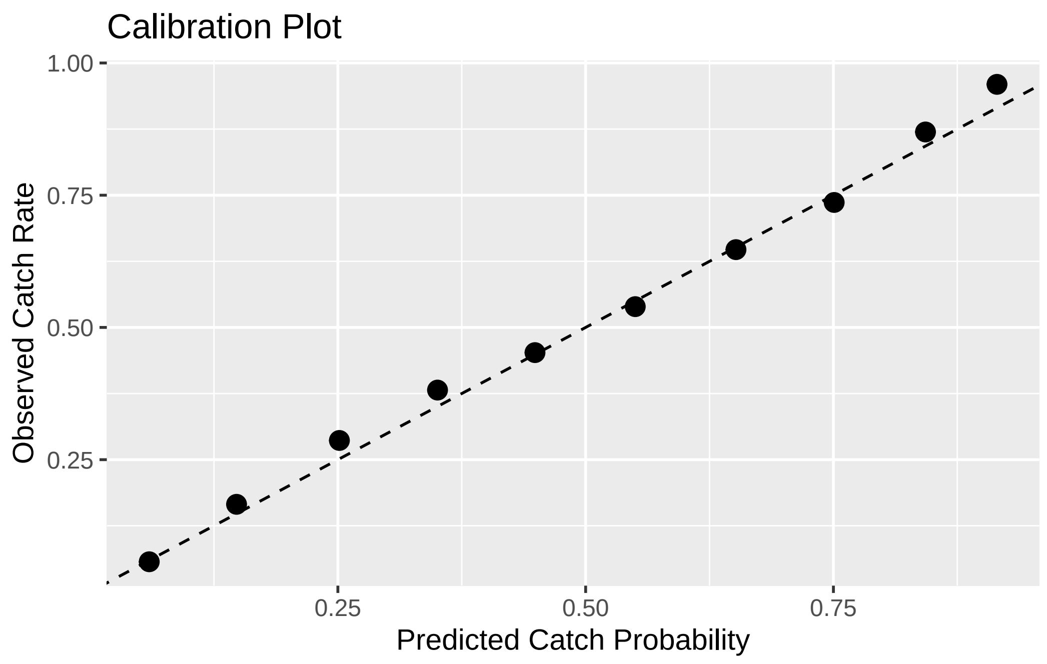
```r
auc_val <- auc(roc_obj)
print(auc_val)
```

```
Area under the curve: 0.81
```

The ROC curve shows how the model can reliably distinguish between catches and incompletions/interceptions, higher predicted probabilities correspond to higher likelihood of a catch. An AUC of .815 shows that the model correctly ranks catches above incompletions over 81% of the time.

0.4.5 Calibration Plot

```r
cal_df <- data.frame(
  prob = test_pred,
  obs  = Test_y
) %>%
  mutate(bin = cut(prob, breaks = seq(0,1,.1), include.lowest = TRUE)) %>%
  group_by(bin) %>%
  summarize(
    avg_prob = mean(prob),
    avg_obs  = mean(obs),
    n = n()
  )

ggplot(cal_df, aes(x = avg_prob, y = avg_obs)) +
  geom_point(size = 3) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  labs(
    title = "Calibration Plot",
    x = "Predicted Catch Probability",
    y = "Observed Catch Rate"
  )
```



The calibration plot shows that the predicted catch probability does increase with the actual observed catch probability, showing that the model is still able to capture the underlying likelihood probability, even with high variability data.

0.4.6 Compute frame-level catch probabilities

Once the model is trained, we use it to predict a catch probability for each frame in our feature set. These probabilities will be used to measure how the catch chance changes over the course of the ball's flight for each target-defender pairing.

```r
Features_x <- model.matrix(
  caught ~ . - game_id - play_id - frame_id - nfl_id_wr - primary_nfl_id_def - 1,
  data = features_final
)

dfeatures <- xgb.DMatrix(data = Features_x)

features_final$pred_prob <- predict(xgb_model, dfeatures)
```

0.5 Results

0.5.1 Define the catch probability drop (CP_DROP) per play

For each target and primary defender, we summarize the model's predictions into a simple metric. WE take the predicted catch probability at the earliest frame and at the arrival frame and compute "cp_drop = p0 - pT." This measures how much the defender(and the play dynamics) reduced the catch chance from the start of the ball's flight to the end.

```r
cp_drop <- features_final %>%
    group_by(game_id, play_id, nfl_id_wr, primary_nfl_id_def) %>%
    summarise(
        p0 = pred_prob[frame_id == min(frame_id)],
        pT = pred_prob[frame_id == max(frame_id)],
        cp_drop = p0 - pT,
        .groups = "drop"
    )

cp_drop <- cp_drop %>%
    mutate(
        p0 = round(p0, 3),
        pT = round(pT, 3),
        cp_drop = round(cp_drop, 3)
    )
```

0.5.2 Rank defenders by average CP_DROP and add player information

Finally, we average CP_DROP over all plays for each primary defender to get a defender-level impact score. We then merge in player names and positions, and filter to defenders with a reasonable sample size. This gives us a ranking of coverage players based on how much they consistently lower catch probability when they are the nearest defender.

```
defender_rankings <- cp_drop %>%
    group_by(primary_nfl_id_def) %>%
    summarise(
        avg_drop = round(mean(cp_drop, na.rm = TRUE),3),
        n_targets = n()
    ) %>%
    arrange(desc(avg_drop))
defender_rankings
```

```
# A tibble: 486 x 3
   primary_nfl_id_def avg_drop n_targets
                <int>    <dbl>     <int>
 1              54705    0.51          1
 2              56010    0.427         1
 3              46968    0.336         1
 4              56374    0.33          2
 5              47872    0.29          4
 6              56405    0.281         6
 7              53688    0.274         1
 8              42488    0.251         2
 9              48588    0.247         1
10              55919    0.24          1
# i 476 more rows
```

```
player_lookup <- fully_joined %>%
    select(nfl_id, player_name, player_position) %>%
    distinct()

defender_rankings <- defender_rankings %>%
    left_join(player_lookup, by = c("primary_nfl_id_def" = "nfl_id"))

defender_rankings <- defender_rankings |> filter(n_targets >= 20)

defender_rankings
```

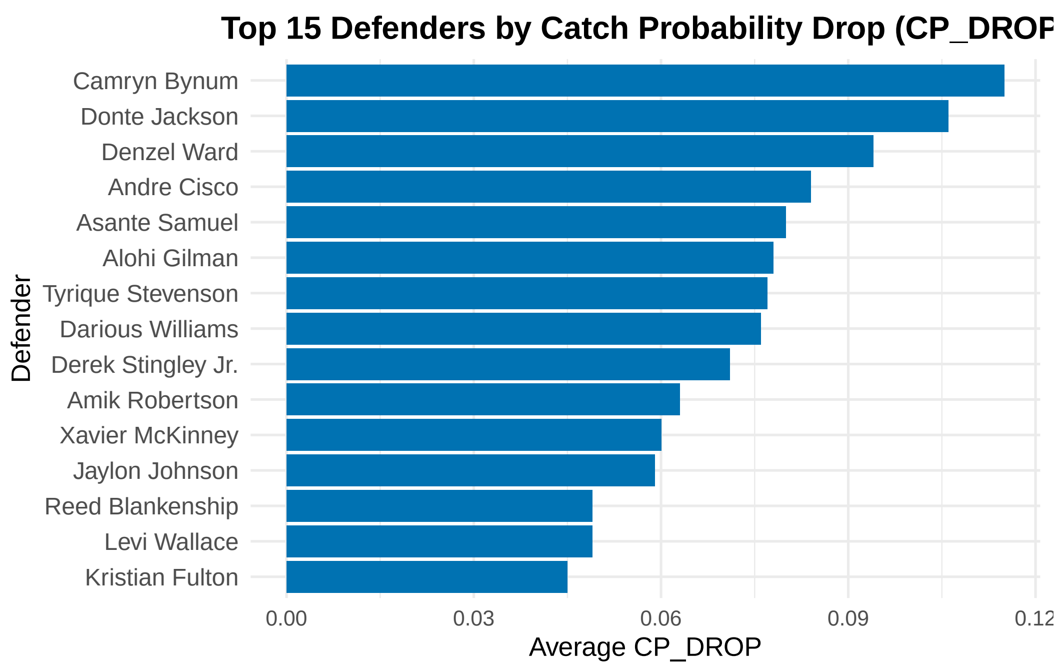
```
# A tibble: 91 x 5
   primary_nfl_id_def avg_drop n_targets player_name        player_position
                <int>    <dbl>     <int> <chr>              <chr>
 1              53554    0.115        27 Camryn Bynum       FS
 2              46124    0.106        22 Donte Jackson      CB
 3              46073    0.094        26 Denzel Ward        CB
 4              53494    0.084        20 Andre Cisco        FS
 5              53476    0.08         29 Asante Samuel      CB
 6              52594    0.078        20 Alohi Gilman       FS
 7              55921    0.077        41 Tyrique Stevenson  CB
 8              46456    0.076        37 Darious Williams   CB
 9              54468    0.071        24 Derek Stingley Jr. CB
10              52547    0.063        23 Amik Robertson     CB
# i 81 more rows
```

As we can see above, the top players are able to on average drop the probability of a catch by 6-11%, from the time it leaves the quarterback to the time it arrives at the target. This reduction reflects the players abilities to close space, disrupt timing, and make receivers less comfortable at the point that they should be catching the ball.

0.6 Visualization

0.6.1 Primary Defender Impact (Top 15)

```r
top_def <- defender_rankings |>
  dplyr::arrange(desc(avg_drop)) |>
  dplyr::slice_head(n = 15)

ggplot(
  top_def,
  aes(x = reorder(player_name, avg_drop), y = avg_drop)
) +
  geom_col(fill = oi_colors[6]) +
  coord_flip() +
  labs(
    title = "Top 15 Defenders by Catch Probability Drop (CP_DROP)",
    x = "Defender",
    y = "Average CP_DROP"
  ) +
  theme_minimal(base_size = 10) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    axis.text.y = element_text(size = 9)
  )
```



This chart highlights defenders who consistently create the largest drop in catch probability from release to arrival. Players like Camryn Bynum, Donte Jackson, and Denzel Ward excel at closing separation late in the play and disrupting throws at the catch point. Their high CP_DROP values indicate strong coverage impact that isn't fully captured by standard stats like pass breakups or completion percentage allowed.

0.6.2 WR–Defender Start vs End Separation

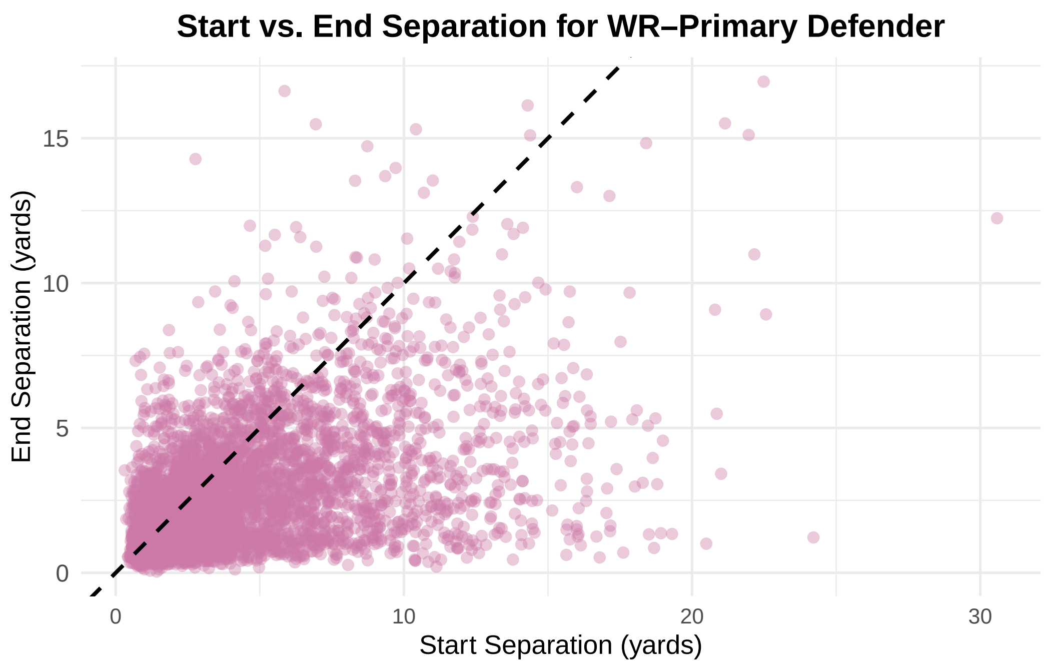
```r
sep_summary <- features_final |>
  dplyr::group_by(game_id, play_id, nfl_id_wr, primary_nfl_id_def) |>
  dplyr::summarise(
    start_sep = dist_wr_def[frame_id == min(frame_id)],
    end_sep   = dist_wr_def[frame_id == max(frame_id)],
    .groups = "drop"
  )
```

```r
df_check <- sep_summary %>%
  inner_join(cp_drop, by = c("game_id","play_id","nfl_id_wr"))
cor(df_check$start_sep - df_check$end_sep, df_check$cp_drop)
```

```
[1] 0.2074705
```

This positive correlation of .21 shows that when defenders close more distance, the cp_drop tends to be higher.

```r
ggplot(sep_summary, aes(x = start_sep, y = end_sep)) +
  geom_point(alpha = 0.4, color = oi_colors[8]) +
  geom_abline(intercept = 0, slope = 1,
              linetype = "dashed",
              color = oi_colors[1],
              linewidth = 0.7) +
  labs(
    title = "Start vs. End Separation for WR-Primary Defender",
    x = "Start Separation (yards)",
    y = "End Separation (yards)"
  ) +
  theme_minimal(base_size = 10) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    axis.text.y = element_text(size = 9)
  )
```



This scatter plot compares the receiver-defender separation at the start of the ball's flight to the separation at arrival. Most plays fall within typical NFL spacing (0-10 yards), and the dashed diagonal line highlights how often defenders shrink the separation by the catch point. Because most points fall below the diagonal, defenders generally close space as the ball travels. This supports the idea that tighter late coverage often reduces catch probability. This reinforces CP_DROP as a meaningful metric of catch-point disruption.

0.6.3 Analyzing Camryn Bynum's Highest Impact Play

To better understand how Camryn Bynum affects a play, we identify the snap where he produced his largest decrease in catch probability (CP_DROP). We filter all plays where Bynum was the primary defender, compute the change in predicted catch probability from the start to the end of the route, and select the play with the biggest drop. This gives us the single rep where Bynum most dramatically reduced the receiver's chances of making the catch, his “best coverage play by the model.

```r
bynum_plays <- cp_drop |>
  dplyr::filter(primary_nfl_id_def == player_lookup$nfl_id
                [player_lookup$player_name == "Camryn Bynum"])

best_bynum_play <- bynum_plays |>
  dplyr::arrange(desc(cp_drop)) |>
  dplyr::slice(1)

best_bynum_play
```

```
# A tibble: 1 x 7
     game_id play_id nfl_id_wr primary_nfl_id_def    p0    pT cp_drop
       <dbl>   <dbl>     <int>              <int> <dbl> <dbl>   <dbl>
1 2023102300    3969     46256              53554 0.622 0.084   0.538
```

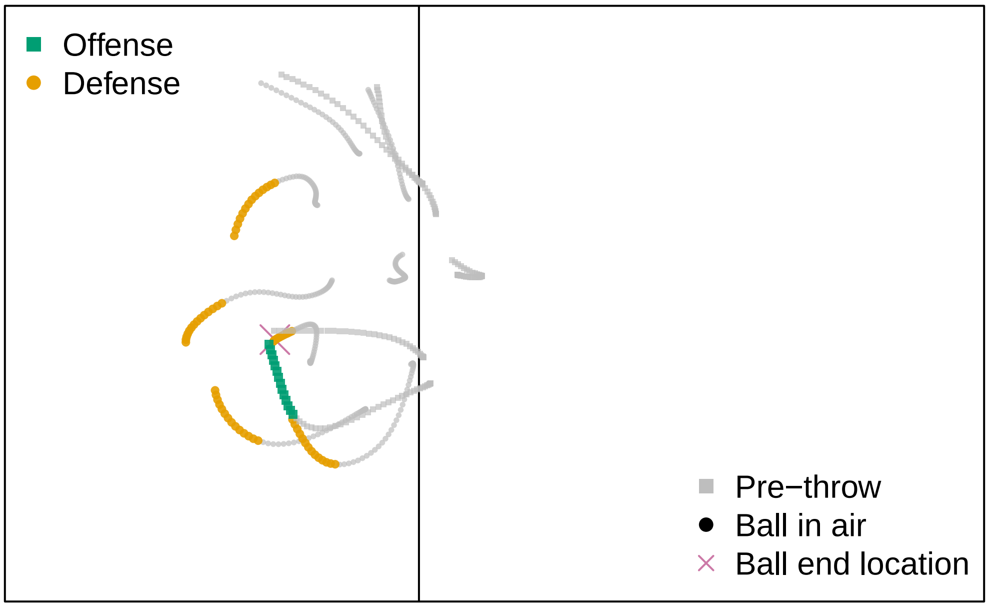
Here we are finding the game and play ID for Bynum's most impactful coverage rep according to our model. Next, we use those IDs to pull full tracking data for that snap and visualize all player movement before and after the ball was thrown.

```r
gid <- best_bynum_play$game_id
pid <- best_bynum_play$play_id

tmp_in <- full_input |>
  dplyr::filter(game_id == gid, play_id == pid)

tmp_out <- full_output |>
  dplyr::filter(game_id == gid, play_id == pid)

ball_land_x <- tmp_in$ball_land_x[1]
ball_land_y <- tmp_in$ball_land_y[1]
all_players <- unique(c(tmp_in$nfl_id, tmp_out$nfl_id))

oi_colors <- palette.colors(palette = "Okabe-Ito")

par(mar = c(1,1,1,1), mgp = c(1.8, 0.5, 0))
plot(
  1, type = "n",
  xlim = c(0, 120), ylim = c(0, 54),
  xaxt = "n", yaxt = "n",
  xlab = "", ylab = ""
)
                                                                                   ①
```

```r
points(ball_land_x, ball_land_y, pch = 4, cex = 2, col = oi_colors[8])
lines(
  x = c(tmp_in$absolute_yardline_number[1], tmp_in$absolute_yardline_number[1]),
  y = c(par("usr")[3], par("usr")[4])
)

for (player in all_players) {
  player_in <- tmp_in |>
    dplyr::filter(nfl_id == player)

  if (nrow(player_in) == 0) next

  role <- player_in$player_role[1]
  side <- player_in$player_side[1]

  side_col <- ifelse(side == "Offense", oi_colors[4], oi_colors[2])
                                                                                  ②
  points(
    player_in$x, player_in$y,
    pch = ifelse(side == "Offense", 15, 16),
    cex = 0.4,
    col = adjustcolor("grey", alpha.f = 0.7)
  )
                                                                                  ③
  if (isTRUE(player_in$player_to_predict[1])) {
    player_out <- tmp_out |>
      dplyr::filter(nfl_id == player)

    if (nrow(player_out) > 0) {
      points(
        player_out$x, player_out$y,
        pch = ifelse(side == "Offense", 15, 16),
        cex = 0.6,
        col = adjustcolor(side_col, alpha.f = 0.9)
      )
    }
  }
}

legend(
  "topleft",
  legend = c("Offense", "Defense"),
  pch = c(15, 16),
  col = oi_colors[c(4, 2)],
  bty = "n"
)
```

```r
legend(
  "bottomright",
  legend = c("Pre-throw", "Ball in air", "Ball end location"),
  pch = c(15, 16, 4),
  col = c("grey", "black", oi_colors[8]),
  bty = "n"
)
```

① Ball landing spot and line of scrimmage
② Pre-throw path (grey)
③ Ball-in–air path (only for players in the prediction set)



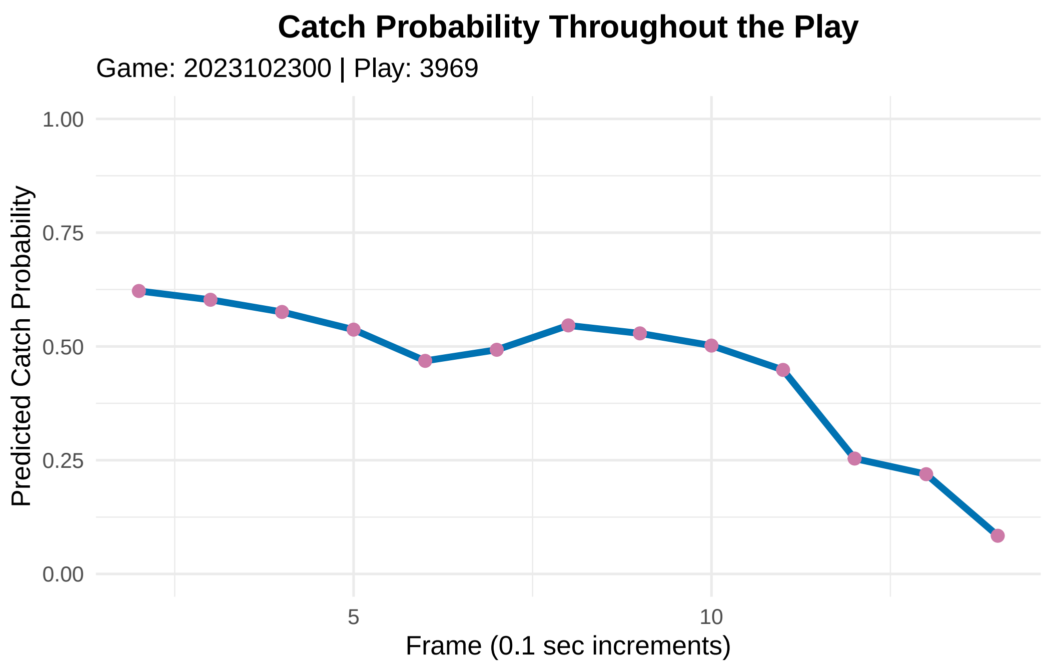
To put this play into context, we also pull the game metadata (teams, quarter, gameclock, and description) from the supplementary file for the same game and play.

```r
play_info <- meta_data |>
  dplyr::filter(game_id == gid,
                play_id == pid) |>
  dplyr::select(
    game_date,
    home_team_abbr,
    visitor_team_abbr,
    quarter,
    game_clock,
    play_description
  )

play_info
```

```
# A tibble: 1 x 6
  game_date home_team_abbr visitor_team_abbr quarter game_clock play_description
  <chr>     <chr>          <chr>               <dbl> <time>     <chr>
1 10/23/20~ MIN            SF                      4 34'00"     (:34) (Shotgun)~
```

Finally, we look at how the model's predicted catch probability evolves over the course of this play. Using the frame-level predictions already stored in feature_final, we can trace how Bynum's coverage influences the chance of a completion from the early route stem to the catch point.

```r
play_frames <- features_final |>
  dplyr::filter(game_id == gid, play_id == pid) |>
  dplyr::arrange(frame_id)

ggplot(play_frames, aes(x = frame_id, y = pred_prob)) +
  geom_line(color = oi_colors[6], linewidth = 1.2) +
  geom_point(color = oi_colors[8], size = 1.8) +
  labs(
    title = "Catch Probability Throughout the Play",
    subtitle = paste("Game:", gid, "| Play:", pid),
    x = "Frame (0.1 sec increments)",
    y = "Predicted Catch Probability"
  ) +
  scale_y_continuous(limits = c(0, 1)) +
  theme_minimal(base_size = 10) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5)
  )
```



0.6.4 Video of the play

In conjunction with these tables, play take a look at this clip of the play itself on youtube (https://youtu.be/goAvP2I-twQ?si=HbudrvMUrutsa0X2&t=803)

0.7 Defender Drops Over Expected

Want to find how often a defender prevents a catch the model think should be caught. This can give defenders credit for things like breaking up throws that were likely completions, disrupt timing of play, or forcing the receiver to under perform. For defender a good play is when the probability of catch is high, but the ball is not caught.

```r
def_target_summary <- features_final %>%
  group_by(game_id, play_id, nfl_id_wr, primary_nfl_id_def) %>%
  summarise(
    caught = caught[1],
    pT = pred_prob[frame_id == max(frame_id)],
    .groups = "drop"
  ) %>%
  mutate(
    defender_doe = pmax(pT - caught, 0)
  )
```

```r
defender_doe <- def_target_summary %>%
  group_by(primary_nfl_id_def) %>%
  summarise(
    total_doe = sum(defender_doe, na.rm = TRUE),
    avg_doe = mean(defender_doe, na.rm = TRUE),
    n_targets = n(),
    .groups = "drop"
  ) %>%
  left_join(player_lookup, by = c("primary_nfl_id_def" = "nfl_id")) %>%
  arrange(desc(total_doe))
defender_doe
```

```
# A tibble: 486 x 6
   primary_nfl_id_def total_doe avg_doe n_targets player_name    player_position
                <int>     <dbl>   <dbl>     <int> <chr>          <chr>
 1              53505      6.90   0.141        49 Paulson Adebo  CB
 2              44878      6.53   0.145        45 Ahkello Withe~ CB
 3              43700      6.43   0.214        30 Jonathan Jones CB
 4              46757      6.30   0.162        39 Charvarius Wa~ CB
 5              43351      6.26   0.149        42 James Bradber~ CB
 6              52546      6.24   0.208        30 L'Jarius Sneed CB
 7              46456      6.19   0.167        37 Darious Willi~ CB
 8              54632      6.11   0.139        44 DaRon Bland    CB
 9                54622      6.07   0.178        34 Zyon McCollum   CB
10                54969      5.90   0.295        20 Ja'Quan McMil~ CB
# i 476 more rows
```

The top Defender DOE players are those who consistently turn likely catches into incompletions, showing strong catch-point disruption. Players like Paulson Adebo, L'Jarius Sneed, Jonathan Jones, and DaRon Bland excel at breaking up throws the model expects to be completed. This complements CP_DROP by highlighting defenders who may allow some separation early but finish plays exceptionally well at the catch point.

0.8 Putting them Together

```r
defender_combined <- defender_rankings %>%
  inner_join(defender_doe, by = "primary_nfl_id_def", suffix = c("_drop", "_doe"))
```

0.8.1 Elite players

```r
drop_cut  <- quantile(defender_combined$avg_drop, 0.75)
doe_cut   <- quantile(defender_combined$total_doe, 0.75)

elite_both <- defender_combined %>%
  filter(avg_drop >= drop_cut, total_doe >= doe_cut) %>%
  arrange(desc(avg_drop))
elite_both
```

```
# A tibble: 4 x 10
  primary_nfl_id_def avg_drop n_targets_drop player_name_drop
               <int>    <dbl>          <int> <chr>
1              55921    0.077             41 Tyrique Stevenson
2              46456    0.076             37 Darious Williams
3              46698    0.049             37 Levi Wallace
4              54622    0.041             34 Zyon McCollum
# i 6 more variables: player_position_drop <chr>, total_doe <dbl>,
#   avg_doe <dbl>, n_targets_doe <int>, player_name_doe <chr>,
#   player_position_doe <chr>
```

0.8.2 Weaker players

```r
drop_low_cut <- quantile(defender_combined$avg_drop, 0.25)
doe_low_cut  <- quantile(defender_combined$total_doe, 0.25)

weak_both <- defender_combined %>%
  filter(avg_drop <= drop_low_cut, total_doe <= doe_low_cut) %>%
  arrange(avg_drop)
weak_both
```

```
# A tibble: 8 x 10
  primary_nfl_id_def avg_drop n_targets_drop player_name_drop
               <int>    <dbl>          <int> <chr>
1              44828   -0.081             21 Marlon Humphrey
2              54580   -0.064             20 Damarri Mathis
3              54687   -0.056             21 Montaric Brown
4              44872   -0.054             26 Chidobe Awuzie
5              56086   -0.052             20 Jaylon Jones
6              43327   -0.048             22 Xavien Howard
7              53565   -0.034             27 Marco Wilson
8              54486   -0.027             22 Trent McDuffie
# i 6 more variables: player_position_drop <chr>, total_doe <dbl>,
#   avg_doe <dbl>, n_targets_doe <int>, player_name_doe <chr>,
#   player_position_doe <chr>
```

These results show that combining CP__DROP and Defender DOE provides a fuller picture of coverage performance by capturing both separation tightening during the route and disruption at the catch point. Defenders who score highly on both metrics consistently limit receiver opportunities and finish plays at a level above expectation. Conversely, those low in both measures exert little influence on route progression or catch-point outcomes, highlighting meaningful differences in defensive impact across the league.

0.9

Next Steps

While the model provides meaningful insights into defender impact, several extensions could significantly improve its accuracy and usefulness.

1. *Incorporate More Contextual Features:*

```
Variables such as QB pressure, throw velocity, route depth, and leverage angles
could provide a fuller picture of how coverage evolves.
```

2. *Separate Models for Man vs. Zone Coverage*

```
Defender responsibilities differ dramatically across coverage types. Training
distinct models for man and zone would likely produce more interpretable results.
```

3. Integrate All Defenders Instead of Only Primary One

```
Safety help, bracket coverage, and overlapping zones all affect catch
probability. Expanding beyond nearest defender would better capture real
defensive structure.
```

Overall, these next steps would refine catch probability estimation, enhance interpretability, and make defender-impact metrics more actionable for analysts, coaches, and couting departments.