1. Objective

The purpose of this project is to build an internal catcher framing metric that isolates the catcher's influence on called pitches. The workflow consists of:

- Training a pitch-level model to estimate the probability that a taken pitch is called a strike, independent of the catcher.
- Compare actual called strikes to expected called strikes to quantify framing.
- Aggregating framing performance at the catcher-year level.
- Applying Bayesian hierarchical modeling to obtain stable, partially pooled framing estimates.
- Delivering a production-ready script (production.py) capable of applying the model to new data.

Final deliverables:

1. Trained LightGBM called-strike model.
2. `pitch_level_predictions.csv` (pitch-level strike probabilities).
3. `new_output.csv` (catcher-year framing table).
4. Readable, reproducible Python pipeline.

2. Data & Preprocessing

2.1 Dataset Description

The provided dataset contains:

Pitch physics

PLATELOCHEIGHT, PLATELOCSIDE, TOP_ZONE, BOT_ZONE, RELSPEED, RELHEIGHT, RELSIDE, EXTENSION, INDUCEDVERTBREAK, HORZBREAK, HORZAPPRANGLE, VERTAPPRANGLE.

Pitch context

BALLS, STRIKES, OUTS, RUNNERS_ON, BASE_CONFIGURATION, BATTERSIDE, PITCHERTHROWS.

Actor identifiers

CATCHER_ID, PITCHER_ID, UMPIRE_ID, GAMEID, TEAM IDs, RUNNER IDs, STADIUM_ID.

Outcome

PITCHCALL ∈ {BallCalled, StrikeCalled}

2.2 Filtering

We filter the dataset to only called pitches and define:

- IS_STRIKE = 1 when StrikeCalled
- IS_STRIKE = 0 when BallCalled

2.3 Preventing Leakage of Catcher/Umpire Effects

To keep the model neutral to players and umpires, the following columns are excluded from features:

- PITCH_ID
- GAMEID
- CATCHER_ID
- PITCHER_ID
- UMPIRE_ID
- RUNNER_ON_FIRST_ID
- RUNNER_ON_SECOND_ID
- RUNNER_ON_THIRD_ID
- HOME_TEAM_ID
- PITCHER_TEAM_ID
- STADIUM_ID

This ensures the learned probability reflects what an average catcher would receive.

3. Stage 1 — Baseline Called Strike Probability Model

3.1 Objective

Estimate Strike Called Probability

$$P\left(\text{StrikeCalled} \mid \text{pitch characteristics, location, situation}\right)$$

This gives a baseline strike probability (CS_PROB) independent of framing skill.

3.2 Feature Selection

To improve how the model understands pitch context and zone location, I add a set of simple, catcher-agnostic engineered features.

1. Count and Leverage Features

- PITCH_COUNT = BALLS + STRIKES
- IS_TWO_STRIKES: indicator for two-strike counts
- IS_FULL_COUNT: Indicator for a 3–2 count. Full-count pitches are especially high-stakes and may be called differently.
- IS_HIGH_LEVERAGE: full count, runners on base, or two outs

2. Strike Zone Geometry and Normalization

- PLATE_Z_NORM: vertical distance from zone center, scaled by zone height
- PLATE_X_NORM: horizontal location scaled by half-plate width
- PLATE_Z_NORM_SQ / PLATE_X_NORM_SQ: squared terms for distance-from-center

3. Edge / Borderline (Shadow Zone) Features

- IN_ZONE: inside both vertical & horizontal boundaries
- IS_EDGE_VERT / IS_EDGE_HORZ: near top/bottom or inside/outside edges
- IS_SHADOW_PITCH: pitch on either edge

4. Directional Features: High/Low, Inside/Outside, Glove/Arm Side

- IS_HIGH_PITCH / IS_LOW_PITCH
- IS_INSIDE / IS_OUTSIDE (relative to batter handedness)
- IS_GLOVE_SIDE / IS_ARM_SIDE (relative to catcher)

5. Interaction Features (Key Contexts for Framing)

- EDGE_TWO_STRIKES = IS_SHADOW_PITCH × IS_TWO_STRIKES
- EDGE_HIGH_LEVERAGE = IS_SHADOW_PITCH × IS_HIGH_LEVERAGE
- VERT_BREAK_LOW / VERT_BREAK_HIGH = induced break × pitch height

Different pitch types get called differently in the same spot, so this helps LightGBM capture those differences.

3.3 Model Choice: LightGBM

LightGBM is selected because:

- It models nonlinear boundary effects crucial for shadow-zone predictions.
- Natively supports categorical inputs.
- Produces well-calibrated probabilistic outputs.

- Fast and efficient for large pitch-level datasets.

3.4 Training Setup

- StratifiedKFold cross-validation with 5 folds.
- Early stopping based on validation AUC/logloss.
- Final n_estimators derived by averaging best_iteration across folds.
- Final LightGBM model is trained on the full dataset.

Performance metrics: AUC, LogLoss, RMSE

4. Stage 2 - From Probabilities to Framing

This stage converts pitch-level predictions into catcher-year framing values.

4.1 Pitch-Level Residuals

For each pitch, calculate:

Residual = IS_STRIKE – CS_PROB

Interpretation:

- Positive → catcher got more strikes than expected.
- Negative → catcher got fewer strikes than expected.

Residuals are aggregated later to measure framing.

4.2 Aggregate to Catcher-Year

Group residuals by:

- CATCHER_ID
- GAME_YEAR

For each catcher-year, compute:

- Opportunities = number of taken pitches
- ActualCalled = sum(IS_STRIKE)
- ExpectedCalledStrikes = sum(CS_PROB)
- mean_resid = average residual for that group

mean_resid is the raw per-pitch framing effect.

However, raw effects have problems:

- Catchers with few opportunities have noisy, unstable values.
- Year-to-year variance is inflated.
- We need shrinkage toward league-average for stability.

This motivates the Bayesian hierarchical model.

4.3 Model Structure

For each catcher-year i:

- n_i = the number of called-pitch opportunities
- τ_i = observed mean residual
- $theta_i$ = true underlying framing ability

The hierarchical normal-normal model is:

Level 1: Latent catcher-year effect

$$\theta_i \sim \mathcal{N}(0, \tau)$$

Level 2: Observed mean residual

$$r_i \sim \mathcal{N}\left(\theta_i,\ \frac{\sigma}{\sqrt{n_i}}\right)$$

Where:

- τ controls variance between catcher-years.
- θ captures pitch-level random variation.

Priors:

- $\tau \sim \mathrm{HalfNormal}(0.2)$
- $\sigma \sim \mathrm{HalfNormal}(1.0)$

Intuition Behind the Model

- Catcher-years with huge sample sizes (n_i large) have precise estimates; posterior ≈ raw.
- Catcher-years with small sample sizes shrink strongly toward 0.
- The resulting effects are more predictive and less noisy.

4.4 Framing Metric

$$\mathrm{CalledStrikesAdded}_i = \mathrm{PosteriorEffect}_i \times \mathrm{Opportunities}_i$$

$$\mathrm{CSA\backslash\_per\backslash\_100}_i = 100 \times \mathrm{PosteriorEffect}_i$$

These are the final framing metrics delivered to Baseball Operations.

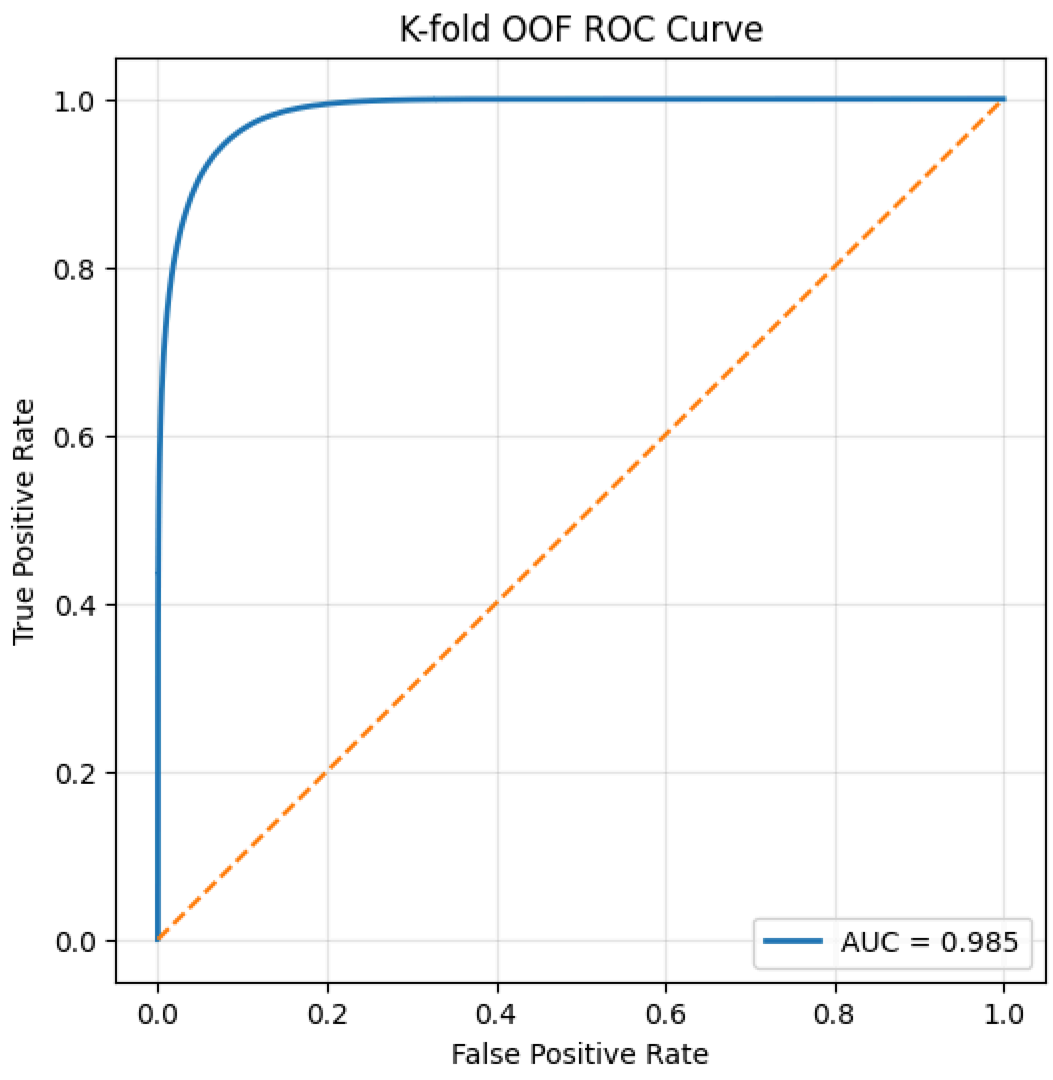
5. Data Visualizations

Stage 1 — Baseline Called Strike Probability Model K-fold cross-validation AUC curve.

In [14]:
```
import numpy as np
import pandas as pd
import lightgbm as lgb
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

def load_and_prepare_data(csv_path: str):
    df = pd.read_csv(csv_path)

    valid_calls = ["BallCalled", "StrikeCalled"]
    df = df[df["PITCHCALL"].isin(valid_calls)].copy()
    df["IS_STRIKE"] = (df["PITCHCALL"] == "StrikeCalled").astype(int)

    id_like_cols = [
        "PITCH_ID",
        "GAMEID",
        "CATCHER_ID",
        "PITCHER_ID",
        "UMPIRE_ID",
        "RUNNER_ON_FIRST_ID",
        "RUNNER_ON_SECOND_ID",
        "RUNNER_ON_THIRD_ID",
        "HOME_TEAM_ID",
        "PITCHER_TEAM_ID",
        "STADIUM_ID",
    ]
    target_cols = ["PITCHCALL", "IS_STRIKE", "GAME_YEAR"]
    drop_cols = set(id_like_cols + target_cols)

    candidate_cols = [c for c in df.columns if c not in drop_cols]

    cat_cols = [
        c for c in candidate_cols
        if df[c].dtype == "object" or str(df[c].dtype).startswith("category"
    ]
    num_cols = [c for c in candidate_cols if c not in cat_cols]

    for c in cat_cols:
        df[c] = df[c].astype("category")

    feature_cols = num_cols + cat_cols

    X = df[feature_cols]
    y = df["IS_STRIKE"].values
    return df, X, y, feature_cols, cat_cols

csv_path = "ML_TAKES_ENCODED.csv"
df_all, X_all, y_all, feature_cols, cat_cols = load_and_prepare_data(csv_pat

def plot_roc_curve(y_true, y_prob, title="ROC Curve"):
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 6))
    plt.plot(fpr, tpr, linewidth=2, label=f"AUC = {roc_auc:.3f}")
    plt.plot([0, 1], [0, 1], linestyle="--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(title)
    plt.legend(loc="lower right")
    plt.grid(alpha=0.3)
    plt.show()

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
oof_pred = np.zeros(len(X_all))

cat_idx = [feature_cols.index(c) for c in cat_cols]

fold_aucs = []

for fold, (train_idx, valid_idx) in enumerate(skf.split(X_all, y_all), start
    print(f"=== Fold {fold} ===")
    X_train, X_valid = X_all.iloc[train_idx], X_all.iloc[valid_idx]
    y_train, y_valid = y_all[train_idx], y_all[valid_idx]

    model = lgb.LGBMClassifier(
        objective="binary",
        boosting_type="gbdt",
        n_estimators=2000,
        learning_rate=0.05,
        num_leaves=63,
        random_state=42 + fold,
        n_jobs=-1,
    )

    model.fit(
        X_train,
        y_train,
        eval_set=[(X_valid, y_valid)],
        eval_metric="auc",
        categorical_feature=cat_idx,
        callbacks=[
            lgb.early_stopping(100),
            lgb.log_evaluation(period=50),
        ],
    )

    y_valid_pred = model.predict_proba(X_valid)[:, 1]
    oof_pred[valid_idx] = y_valid_pred
    # per-fold AUC
    fpr_f, tpr_f, _ = roc_curve(y_valid, y_valid_pred)
    fold_auc = auc(fpr_f, tpr_f)
    fold_aucs.append(fold_auc)
    print(f"Fold {fold} AUC: {fold_auc:.4f}")

print(f"\nMean AUC across folds: {np.mean(fold_aucs):.4f} ± {np.std(fold_auc

plot_roc_curve(y_all, oof_pred, title="K-fold OOF ROC Curve")
```

```
=== Fold 1 ===
[LightGBM] [Info] Number of positive: 279532, number of negative: 607824
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.006338 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3426
[LightGBM] [Info] Number of data points in the train set: 887356, number of
used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.315017 -> initscore=-0.776
769
[LightGBM] [Info] Start training from score -0.776769
Training until validation scores don't improve for 100 rounds
[50]    valid_0's auc: 0.983615 valid_0's binary_logloss: 0.17313
[100]   valid_0's auc: 0.984748 valid_0's binary_logloss: 0.148806
[150]   valid_0's auc: 0.985162 valid_0's binary_logloss: 0.14529
[200]   valid_0's auc: 0.985297 valid_0's binary_logloss: 0.144393
[250]   valid_0's auc: 0.985275 valid_0's binary_logloss: 0.145191
[300]   valid_0's auc: 0.985306 valid_0's binary_logloss: 0.144726
Early stopping, best iteration is:
[224]   valid_0's auc: 0.985307 valid_0's binary_logloss: 0.14431
Fold 1 AUC: 0.9853
=== Fold 2 ===
[LightGBM] [Info] Number of positive: 279533, number of negative: 607824
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.008542 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3426
[LightGBM] [Info] Number of data points in the train set: 887357, number of
used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.315018 -> initscore=-0.776
765
[LightGBM] [Info] Start training from score -0.776765
Training until validation scores don't improve for 100 rounds
[50]    valid_0's auc: 0.983502 valid_0's binary_logloss: 0.174168
[100]   valid_0's auc: 0.984565 valid_0's binary_logloss: 0.149775
[150]   valid_0's auc: 0.984984 valid_0's binary_logloss: 0.146179
[200]   valid_0's auc: 0.985081 valid_0's binary_logloss: 0.146299
[250]   valid_0's auc: 0.984876 valid_0's binary_logloss: 0.150807
Early stopping, best iteration is:
[199]   valid_0's auc: 0.985125 valid_0's binary_logloss: 0.145222
Fold 2 AUC: 0.9851
=== Fold 3 ===
[LightGBM] [Info] Number of positive: 279533, number of negative: 607824
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.009054 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3425
[LightGBM] [Info] Number of data points in the train set: 887357, number of
used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.315018 -> initscore=-0.776
765
[LightGBM] [Info] Start training from score -0.776765
Training until validation scores don't improve for 100 rounds
```

```
[50]    valid_0's auc: 0.983275 valid_0's binary_logloss: 0.174545
[100]   valid_0's auc: 0.984385 valid_0's binary_logloss: 0.15047
[150]   valid_0's auc: 0.984815 valid_0's binary_logloss: 0.14691
[200]   valid_0's auc: 0.984939 valid_0's binary_logloss: 0.146378
[250]   valid_0's auc: 0.984875 valid_0's binary_logloss: 0.148074
Early stopping, best iteration is:
[182]   valid_0's auc: 0.98493  valid_0's binary_logloss: 0.146166
Fold 3 AUC: 0.9849
=== Fold 4 ===
[LightGBM] [Info] Number of positive: 279533, number of negative: 607824
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.008989 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3427
[LightGBM] [Info] Number of data points in the train set: 887357, number of
used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.315018 -> initscore=-0.776
765
[LightGBM] [Info] Start training from score -0.776765
Training until validation scores don't improve for 100 rounds
[50]    valid_0's auc: 0.983494 valid_0's binary_logloss: 0.174034
[100]   valid_0's auc: 0.984548 valid_0's binary_logloss: 0.149857
[150]   valid_0's auc: 0.984927 valid_0's binary_logloss: 0.146434
[200]   valid_0's auc: 0.984993 valid_0's binary_logloss: 0.146784
[250]   valid_0's auc: 0.985077 valid_0's binary_logloss: 0.145804
Early stopping, best iteration is:
[198]   valid_0's auc: 0.985061 valid_0's binary_logloss: 0.14554
Fold 4 AUC: 0.9851
=== Fold 5 ===
[LightGBM] [Info] Number of positive: 279533, number of negative: 607824
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.055676 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3424
[LightGBM] [Info] Number of data points in the train set: 887357, number of
used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.315018 -> initscore=-0.776
765
[LightGBM] [Info] Start training from score -0.776765
Training until validation scores don't improve for 100 rounds
[50]    valid_0's auc: 0.983524 valid_0's binary_logloss: 0.17416
[100]   valid_0's auc: 0.984613 valid_0's binary_logloss: 0.14962
[150]   valid_0's auc: 0.985021 valid_0's binary_logloss: 0.145983
[200]   valid_0's auc: 0.985182 valid_0's binary_logloss: 0.144937
[250]   valid_0's auc: 0.985212 valid_0's binary_logloss: 0.145012
[300]   valid_0's auc: 0.985121 valid_0's binary_logloss: 0.146306
Early stopping, best iteration is:
[210]   valid_0's auc: 0.98519  valid_0's binary_logloss: 0.144879
Fold 5 AUC: 0.9852

Mean AUC across folds: 0.9851 ± 0.0001
```



6. Poduction Pipeline

The production.py script:

- Loads the saved LightGBM model and feature column list.
- Loads new_data.csv and validates schema.
- Ensures categorical columns match model training.
- Computes CS_PROB for all pitches.
- Writes pitch_level_predictions.csv.
- Aggregates pitch-level data to catcher-year.
- Fits Bayesian hierarchical model with PyMC.
- Writes new_output.csv containing:
    - CatcherID
    - Year
    - Opportunities
    - ActualCalledStrikes
    - CalledStrikesAdded

- CalledStrikesAdded_per_100

This ensures reproducible framing metrics across future datasets.

7. Limitations & Future Work

Limitations

- Cross-validation is not year-based
- Umpire effects are not explicitly modeled

Future Work

- Explore GAMS, XGBoost for CS_PROB prediction
- Region-specifc calibration for shadown zone accuracy.

```
In [ ]:
```