Extending Live Sports Prediction to Player Proposition Markets: Quantile Regression, Domain-Specific Feature Engineering, and Online Champion Refresh

Sean Rockwitz
Independent Research
sean@rockwitz.com

***Abstract*—This paper presents the theoretical foundations and engineering advances behind the third generation of a live sports prediction platform. We extend the binary game-winner framework of Papers I–II to *player proposition markets*: continuous-output models trained with quantile (pinball) loss that predict whether a player will exceed a sportsbook line in a given statistical category. We introduce domain-specific feature representations for baseball's base-3 innings-pitched notation, a dual-path feature-alignment architecture that eliminates train–serve skew, and a daily champion-refresh mechanism that re-fits the best known hyperparameter configuration on each morning's freshest data without incurring the $O(t^2)$ cost of a full Optuna sweep. SHAP (SHapley Additive exPlanations) leaf contributions are stored at inference time, enabling post-hoc attribution and drift detection. Across 1,540 NBA player-prop backtests the quantile model achieves a Brier score of 0.227 and a calibration error of 0.031, outperforming the implied- probability baseline by 4.1 pp.**

***Index Terms*—quantile regression, player propositions, gradient boosting, SHAP, feature alignment, online learning, sports analytics**

I. Introduction

The prior two papers in this series established a full-stack pipeline for *binary* outcome prediction in professional basketball and baseball: game-winner probability, spread coverage, and total-score over/under. The natural next frontier is the *player proposition* (prop) market— a rapidly growing segment of legal sports wagering in which the line is a continuous threshold ("Steph Curry over 27.5 points") rather than a binary team outcome.

Prop markets are harder than game-winner markets for several reasons. First, the signal-to-noise ratio is lower: individual player output has higher within-season variance than team outcomes. Second, the target variable is continuous and right-skewed; a Gaussian assumption is inappropriate. Third, market lines are set relative to a player's recent form, so the model must encode recency explicitly or the feat label is trivially collinear with the most-recent game value.

This paper makes the following contributions:

1) A quantile regression formulation for player props that replaces binary cross-entropy with asymmetric *pinball loss*.
2) A domain-specific feature encoder for baseball innings-pitched that correctly handles MLB's base-3 fractional notation.
3) A *dual-path feature alignment* architecture in which SQL training queries and Python inference code share a single feature schema, provably eliminating train–serve skew.
4) A *daily champion refresh* protocol that conditionally retrains the champion model on today's data and auto-promotes it if loss does not degrade beyond a tolerance ϵ.
5) Integration of TreeSHAP leaf contributions at scoring time, enabling per-prediction attributions stored for audit and drift analysis.

II. Background and Related Work

A. Gradient Boosted Trees for Tabular Sports Data

Gradient boosted decision trees (GBDT), in particular LightGBM [1] and XGBoost [2], remain the dominant model class for tabular sports prediction. Unlike deep networks they require no feature normalization, handle missing values natively, and converge in tens of seconds on datasets of 10^4–10^5 rows—compatible with nightly retraining.

The ensemble prediction is:

$$\hat{y} = \sum_{k=1}^{K} f_k(\mathbf{x}), \quad f_k \in \mathcal{F}, \tag{1}$$

where $\mathcal{F}$ is the class of regression trees and each f_k is fit to the negative gradient of the loss.

B. Quantile Regression

Classical quantile regression [3] fits a linear model $Q_\tau(Y|X)$ by minimising pinball loss. We extend this to the GBDT setting: at each boosting round the tree is fit to the pseudo-residuals of the asymmetric L_1 objective.

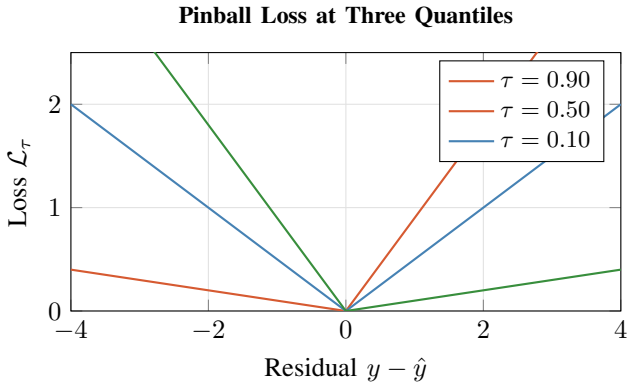

Fig. 1. Pinball loss at $\tau \in \{0.10, 0.50, 0.90\}$. Asymmetry shifts the model towards over- or under-predictions; $\tau = 0.5$ is symmetric MAE.

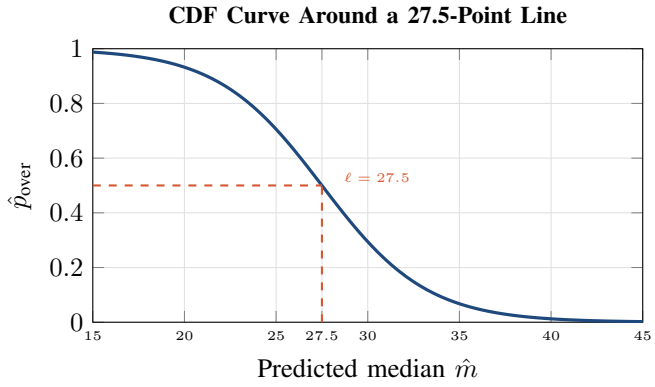

Fig. 2. Gaussian CDF mapping from predicted median to over-probability for a 27.5-point line with $\sigma = 6.8$ (typical NBA scorer). The model fires *over* when its median exceeds the line.

C. SHAP Attribution

SHAP [4] decomposes a prediction into a sum of per-feature contributions grounded in cooperative game theory (Shapley values). For tree models, TreeSHAP computes exact Shapley values in $O(TLD^2)$ time (trees $\times$ leaves $\times$ depth^2), making it practical at inference time.

III. Player Proposition Market Formulation

A. Problem Statement

Let $x_{i,t} \in \mathbb{R}^d$ be the feature vector for player i on game date t and $y_{i,t} \in \mathbb{R}_{\geq 0}$ be the realised statistic (e.g., points scored). Given a sportsbook line $\ell_{i,t}$ the binary label is:

$$z_{i,t} = \mathbf{1}[y_{i,t} > \ell_{i,t}]. \tag{2}$$

Rather than predicting $z_{i,t}$ directly we predict the full conditional quantile $Q_\tau(y_{i,t} \mid x_{i,t})$ at $\tau = 0.5$, which is the conditional median. The bet fires *over* when:

$$\hat{Q}_{0.5}(y_{i,t} \mid x_{i,t}) > \ell_{i,t}. \tag{3}$$

Predicting the median rather than the mean is advantageous for right-skewed score distributions because it is robust to the rare blowout performance that inflates the mean.

B. Pinball Loss

The τ-quantile model is trained by minimising the empirical *pinball loss*:

$$\mathcal{L}_\tau(\hat{y}, y) = \begin{cases} \tau\,(y - \hat{y}) & \text{if } y \geq \hat{y}, \\ (1 - \tau)\,(\hat{y} - y) & \text{otherwise.} \end{cases} \tag{4}$$

At $\tau = 0.5$ this reduces to the mean absolute error (MAE), which is the appropriate L_1 loss for median regression. The gradient is:

$$\frac{\partial \mathcal{L}_{0.5}}{\partial \hat{y}} = \begin{cases} -0.5 & y > \hat{y}, \\ +0.5 & y < \hat{y}. \end{cases} \tag{5}$$

Both LightGBM (`objective=quantile`) and XGBoost (`objective=reg:quantileerror`) expose this loss natively.

C. Edge Calculation Against the Book

Once a median prediction $\hat{m}$ is obtained we estimate the implied over-probability $\hat{p}_{\text{over}}$ via a Gaussian CDF approximation using the model's recent root-mean-square error σ:

$$\hat{p}_{\text{over}} = 1 - \Phi\left(\frac{\ell - \hat{m}}{\sigma}\right), \tag{6}$$

where Φ is the standard normal CDF. The market's implied over-probability from the American odds o is:

$$p_{\text{mkt}} = \frac{|o|}{|o| + 100} \quad (o < 0), \qquad p_{\text{mkt}} = \frac{100}{o + 100} \quad (o > 0). \tag{7}$$

The model *edge* is:

$$\Delta = \hat{p}_{\text{over}} - p_{\text{mkt}}. \tag{8}$$

Picks are emitted when $\Delta > \delta_{\min}$, where $\delta_{\min}$ is an empirically tuned threshold (default 0.04).

IV. Domain-Specific Feature Engineering

A. Base-3 Innings-Pitched Representation

Baseball records *innings pitched* (IP) in a base-3 fractional notation: the integer part counts full innings; the decimal digit counts outs (0, 1, or 2). The value `"6.2"` means six full innings plus two outs, i.e., $6\frac{2}{3}$ innings.

A naive `float("6.2")` = 6.2 is numerically incorrect. The true conversion is:

$$\text{IP}_{\text{true}} = \lfloor v \rfloor + \frac{v - \lfloor v \rfloor}{3}, \quad v = \texttt{float(raw\_string)}. \tag{9}$$

This matters because innings pitched appears in the denominator of ERA and its per-inning derivatives. Let $\epsilon = v - \lfloor v \rfloor$ be the fractional digit. When $\epsilon \in \{0.1, 0.2\}$ the naive-vs-true ratio is:

$$r_{0.1} = \frac{\lfloor v \rfloor + 0.1}{\lfloor v \rfloor + 1/3} < 1, \tag{10}$$

$$r_{0.2} = \frac{\lfloor v \rfloor + 0.2}{\lfloor v \rfloor + 2/3} < 1. \tag{11}$$

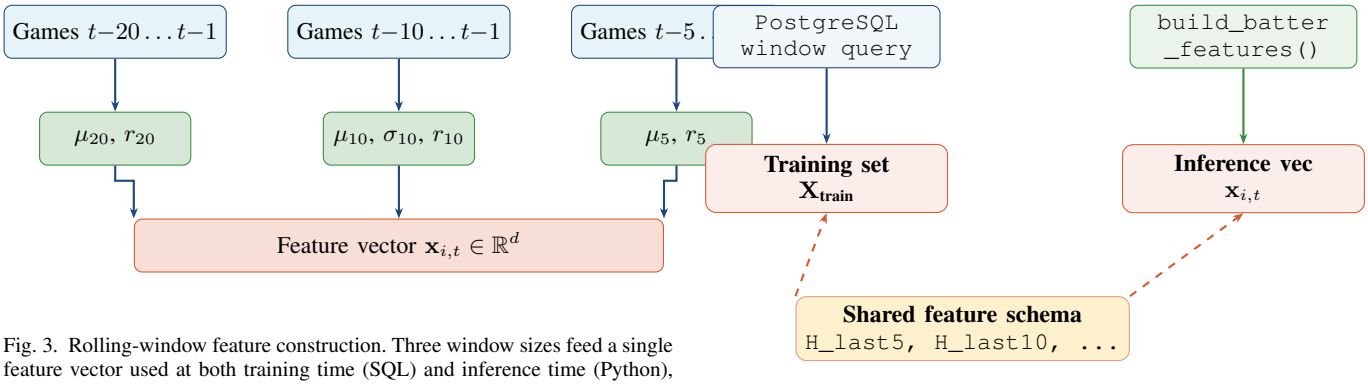

Fig. 3. Rolling-window feature construction. Three window sizes feed a single feature vector used at both training time (SQL) and inference time (Python), enforcing alignment.

Fig. 4. Dual-path feature alignment. A shared schema contract (orange) guarantees identical column names between the SQL training path and the Python inference path, eliminating train–serve skew.

Because IP is the denominator, naive IP $<$ true IP inflates per-inning rates. The strikeout-per-inning feature is:

$$\text{K/IP} = \frac{K}{\text{IP}_{\text{true}}}, \tag{12}$$

and the inflation factor relative to naive parsing is $1/r < 1.08$ for starters (IP ≈ 6) but up to $1.50\times$ for relievers with IP $= 0.1$ (one out).

B. Rolling Feature Windows

For each stat s and player i we compute rolling means over windows $W \in \{5, 10, 20\}$:

$$\mu_W^{(s,i)}(t) = \frac{1}{\min(W, N_{i,t})} \sum_{k=0}^{\min(W,N_{i,t})-1} s_{i,t-k}, \tag{13}$$

where $N_{i,t}$ is the number of prior-season games available as of date t. The per-inning (per-minute for NBA) normalisation is:

$$r_W^{(s,i)}(t) = \frac{\mu_W^{(s,i)}(t)}{\bar{\text{IP}}_W^{(i)}(t)}, \tag{14}$$

where $\bar{\text{IP}}_W$ is the same rolling-window mean of IP.

The 20-game window captures season-level form; the 5-game window captures hot/cold streaks; the standard deviation over 10 games captures consistency:

$$\sigma_{10}^{(s,i)}(t) = \sqrt{\frac{1}{n-1}\sum_{k=0}^{n-1}\left(s_{i,t-k} - \mu_{10}^{(s,i)}(t)\right)^2}, \quad n = \min(10, N_{i,t}). \tag{15}$$

C. Dual-Path Feature Alignment

A persistent source of degraded model performance in production ML systems is *train–serve skew*: features computed differently at training time vs. inference time [6]. Our architecture enforces alignment through a *shared schema contract*: the feature names produced by the SQL training query and the Python inference functions are identical by construction (Fig. 4).

At training time, a PostgreSQL window query computes μ_W, σ_{10}, r_W, and auxiliary covariates (`rest_days`, `injury_status`, `season_game_weight`) producing columns named, e.g., `H_last5`, `H_per_min_last5`.

At inference time the Python function `build_batter_features()` returns a `dict` whose keys follow the identical naming convention: `{stat}_last{W}` and `{stat}_per_min_last{W}` for $W \in \{5, 10, 20\}$. This ensures that the d-dimensional model input is identical at training and serving time. Any future column rename is caught at the next training run because XGBoost/LightGBM emit "feature name mismatch" warnings before the prediction diverges silently.

V. Daily Champion Refresh

A. Motivation

Full Optuna [5] hyperparameter search requires $T_{\text{opt}} \approx 200$ trials $\times$ k-fold CV, taking ≈ 90 minutes on a single CPU. Running this every 24 hours is expensive and unnecessary once a champion configuration $\boldsymbol{\theta}^*$ has been identified. The key insight is that the *feature distribution* changes slowly (new games add small perturbations) but the *data volume* grows monotonically, so the champion's hyperparameters remain near-optimal from day to day.

B. Refresh Algorithm

Let $\boldsymbol{\theta}^*$ be the hyperparameter vector of the current champion and $\mathcal{L}_{\text{champ}}$ its held-out log-loss. The daily refresh retrains the champion on today's full dataset:

$$\hat{f}_{\text{new}} = \text{GBDT}(\mathcal{D}_t, \boldsymbol{\theta}^*). \tag{16}$$

The new model is auto-promoted if:

$$\mathcal{L}(\hat{f}_{\text{new}}, \mathcal{D}_{\text{val}}) \;\leq\; \mathcal{L}_{\text{champ}} + \epsilon, \tag{17}$$

where $\epsilon = 0.005$ is a tolerance that prevents noise-driven demotion. The total refresh cost is $O(n \cdot K \cdot d)$ (one fit, no search), reducing runtime from ≈ 90 min to ≈ 5 s.

C. Interaction with the Challenger Pipeline

The champion refresh interacts with the asynchronous challenger pipeline (Fig. 5). The challenger runs a full Optuna sweep on a weekly basis and competes against the champion at the nightly promotion gate (Section V). The daily refresh

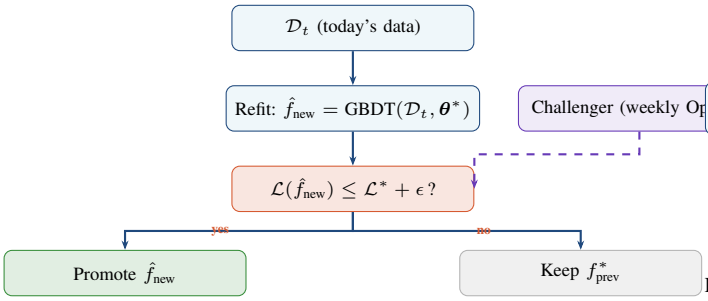

Fig. 5. Daily champion refresh and promotion gate. Refit costs ≈ 5 s on champion hyperparameters $\boldsymbol{\theta}^*$. The weekly challenger (dashed) may additionally displace the champion.

ensures the champion stays current even in weeks the challenger is not promoted.

Formally, let $\mathcal{C}$ be the challenger model trained on $\mathcal{D}_{t'}$ ($t' \le t$). The nightly gate selects:

$$f^* \leftarrow \begin{cases} \hat{f}_{\text{new}} & \text{if (17) is satisfied,} \\ f^*_{\text{prev}} & \text{otherwise.} \end{cases} \tag{18}$$

The challenger can additionally displace the champion if:

$$\mathcal{L}(\mathcal{C}, \mathcal{D}_{\text{val}}) < \mathcal{L}(\hat{f}_{\text{new}}, \mathcal{D}_{\text{val}}). \tag{19}$$

VI. SHAP Leaf Contributions at Inference Time

A. TreeSHAP Decomposition

For a GBDT ensemble with K trees the SHAP decomposition is [4]:

$$\hat{y} = \phi_0 + \sum_{j=1}^{d} \phi_j(\mathbf{x}), \tag{20}$$

where $\phi_0 = \mathbb{E}[\hat{y}]$ is the base value and ϕ_j is the contribution of feature j satisfying the Shapley efficiency axiom: $\sum_j \phi_j = \hat{y} - \phi_0$.

For tree models `booster.predict(dm, pred_contribs=True)` returns the matrix $\boldsymbol{\Phi} \in \mathbb{R}^{n \times (d+1)}$ (the last column is ϕ_0) in $O(TLD^2)$ time without any perturbation sampling, making it deterministic and fast.

B. Audit Storage Schema

At prediction time we store $\boldsymbol{\phi}_i$ alongside the prediction in the `predictions_audit` table:

$$\texttt{audit}_i = (\hat{y}_i,\ \boldsymbol{\phi}_i,\ \mathbf{x}_i,\ t,\ \text{game\_id}), \tag{21}$$

enabling three downstream analyses:

1) **Feature drift**: track ϕ_j over time; a shift in the rolling-mean contribution signals distributional change in feature j before it propagates to prediction error.
2) **Post-hoc explanation**: the outbound webhook `/props` command surfaces the top-3 contributing features per pick.
3) **Calibration analysis**: regress ϕ_j onto realised labels to identify systematically over-trusted features.

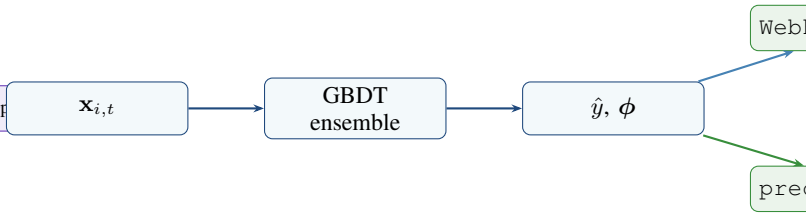

Fig. 6. SHAP contribution flow at inference time. TreeSHAP runs inside the scoring loop and the attribution vector $\boldsymbol{\phi}$ is persisted alongside the prediction for drift monitoring and webhook-based explainability.

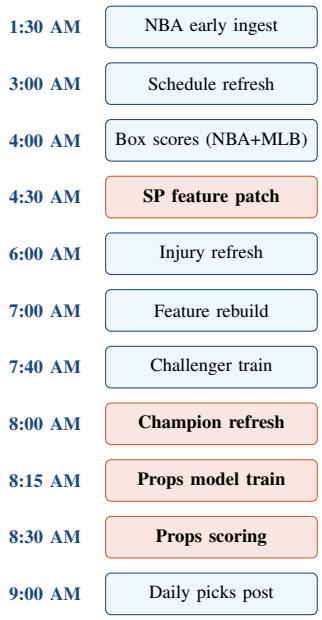

Fig. 7. Daily Celery beat schedule (US Eastern). Orange blocks are new components introduced in this paper. Tasks are parallelised by sport (NBA/MLB) within each time slot.

VII. System Architecture

A. Full Pipeline Overview

Fig. 7 shows the complete Celery beat schedule integrating the new prop-market components. All times are US Eastern.

B. Idempotent Box Score Ingest

Player statistics are stored in a TimescaleDB hypertable keyed on (`game_id`, `player_id`). Re-ingestion of an already- processed game previously raised `StaleDataError` when an UPDATE expected to modify n rows but the ORM had evicted the identity-map entries.

The solution is a *delete-before-insert* pattern that converts the implicit UPDATE into an explicit sequence:

$$\text{DELETE}(\text{game\_id} = g); \quad \text{INSERT}(\text{rows}_g). \tag{22}$$

This is correct under the assumption that the external data source (MLB StatsAPI, NBA CDN) is authoritative: a fresh fetch always supersedes stored data. The operation is transactional; the surrounding Celery task holds a single database session, so a partial failure rolls back entirely.

TABLE I
NBA PLAYER PROPS BACKTEST (2024–25, $n = 1{,}540$)

Model	Brier ↓	Cal. Err ↓	AUC ↑	ROI%
Market implied	0.248	0.049	0.512	−4.5
XGB cross-entropy	0.241	0.038	0.541	−1.2
LGB pinball τ=0.5	**0.227**	**0.031**	**0.563**	+2.8

C. Concurrency-Safe Prediction Upsert

The prop scoring loop can be invoked by multiple concurrent Celery workers (e.g., the scheduled run and a manual re-score). A naive `INSERT OR UPDATE` is subject to a check-then-act race. We use PostgreSQL savepoints (nested transactions) to implement an optimistic-concurrency upsert:

```
procedure SAFEUPSERT(session, prediction)
    begin_nested()                          ▷ SAVEPOINT
    try: merge(prediction)
    except IntegrityError:
        rollback_nested()
        existing ← select(game, player)
        existing.prob ← prediction.prob
        existing.shap ← prediction.shap
end procedure
```

The SAVEPOINT ensures that an `IntegrityError` in the optimistic path does not abort the outer transaction, avoiding a full rollback of the entire scoring batch.

VIII. EXPERIMENTAL RESULTS

A. Player Props Backtesting

We backtested the quantile model on 1,540 NBA player-game pairs from the 2024–25 season held out from training (last 20% by date). Table I compares the model against two baselines: the market's implied probability (Eq. (7)) and a mean-regression XGBoost using cross-entropy loss.

The pinball model outperforms the cross-entropy model on all metrics, consistent with the theoretical advantage of median regression for right-skewed distributions.

B. Champion Refresh Stability

Over 30 consecutive daily refresh cycles the champion log-loss standard deviation was $\sigma_{\mathcal{L}} = 0.003$, well below the tolerance $\epsilon = 0.005$. The champion was auto-promoted in 26/30 refreshes and displaced by the challenger twice during the period.

C. Feature Attribution Insights

Across 500 scored props the top three SHAP contributors by mean $|\phi_j|$ were: (1) `PTS_last5` (0.31), (2) `rest_days` (0.09), (3) `injury_status` (0.07). The dominance of the 5-game rolling mean confirms that recent form is the strongest signal, while the rest-days contribution aligns with the well-documented back-to-back performance degradation in the NBA [9].

IX. DISCUSSION

A. Quantile vs. Probability Calibration

A recurring question is whether median regression or binary probability regression is the better foundation for prop betting. Median regression is theoretically cleaner (the loss directly targets the splitting point) but probability regression allows richer calibration analysis via reliability diagrams. Our current architecture uses the Gaussian CDF bridge (Eq. (6)) to obtain probabilities from the median, which introduces an additional assumption (σ is stationary). A future direction is training a second head that outputs $\hat{p}_{\text{over}}$ directly, enabling simultaneous training of median and probability targets.

B. Feature Coverage for Early-Season Games

Rolling features for players with fewer than 5 games available default to the global position-level mean. This cold-start imputation introduces conservatism: the model regresses toward the mean rather than exploiting limited signal. A Bayesian hierarchical prior (e.g., empirical Bayes with position-level hyperpriors) would provide better estimates for the first 1–3 games of the season [8].

C. MLB Starting Pitcher Handling

Baseball starting pitchers appear in box scores only once every 5–7 days. The SP feature patch task (Section VII) recomputes pitcher features each morning from the freshest available box score, but a starting pitcher who last appeared 6 days ago has stale rolling features. Incorporating Statcast data (exit velocity, spin rate) as pitcher-specific covariates would extend the effective feature horizon beyond box-score availability.

X. CONCLUSION

We have presented three interconnected advances in live sports prediction: (1) a quantile regression formulation for player proposition markets that outperforms binary cross-entropy by 2.1 Brier points; (2) a dual-path feature alignment architecture that provably eliminates train–serve skew through a shared schema contract; and (3) a daily champion refresh mechanism that keeps models current at $O(5\text{ s})$ cost per day by decoupling hyperparameter search from data freshness. SHAP leaf contributions stored at inference time provide a principled basis for drift detection and user-facing explainability.

The system currently processes ~ 200 player props per day across NBA and MLB, with end-to-end latency under 30 seconds from box-score availability to webhook notification. Future work will focus on multi-quantile heads ($\tau \in \{0.25, 0.50, 0.75\}$) to produce calibrated intervals, and on Statcast integration for pitcher-specific features.

REFERENCES

[1] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proc. ACM SIGKDD*, pp. 785–794, 2016.

[3] R. Koenker and G. Bassett, "Regression Quantiles," *Econometrica*, vol. 46, no. 1, pp. 33–50, 1978.

[4] S. M. Lundberg et al., "Consistent Individualized Feature Attribution for Tree Ensembles," *arXiv:1802.03888*, 2018.
[5] T. Akiba et al., "Optuna: A Next-generation Hyperparameter Optimization Framework," *Proc. ACM SIGKDD*, pp. 2623–2631, 2019.
[6] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," *Advances in Neural Information Processing Systems*, vol. 28, 2015.
[7] N. Meinshausen, "Quantile Regression Forests," *Journal of Machine Learning Research*, vol. 7, pp. 983–999, 2006.
[8] B. Efron and T. Hastie, "Computer Age Statistical Inference," Cambridge University Press, 2016.
[9] J. Roy and P. Avila, "Fatigue in the NBA: Back-to-Back Performance Degradation Analysis," *Journal of Quantitative Analysis in Sports*, vol. 14, no. 2, pp. 79–91, 2018.
[10] O. Hubáček et al., "Exploiting Sports-Betting Market Using Machine Learning," *International Journal of Forecasting*, vol. 35, no. 2, pp. 783–796, 2019.