Machine Learning Based Real-Time Ad Click Fraud Detection and IP Mitigation System

R Radhika[1], A Kameswara Rao[2], J Sujay[3]

[1]Assistant Professor, [2,3] UG Scholars
Department of CSE, Sri Chandrasekharendra Saraswathi Viswa Maha Vidyalaya
(SCSVMV Deemed to be University), India.

DOI: 10.64823/ijter.2604016

***Abstract*:** The rapid expansion of digital advertising has created a lucrative target for fraudulent actors who exploit the pay-per-click model by generating illegitimate clicks through automated bots, click farms, and malicious scripts. These fraudulent activities distort campaign analytics, exhaust advertiser budgets, and progressively erode trust in online advertising platforms. Rule-based detection systems — which rely on fixed thresholds and manually defined heuristics — are easily circumvented by adversaries who deliberately engineer their traffic to remain below detection limits while still causing meaningful financial damage.

This paper presents a modular, data-driven system that addresses this challenge by combining a 21-feature behavioural representation with a trained XG-Boost gradient boosting classifier to detect fraudulent clicks in real time. The system handles the severe class imbalance inherent to fraud datasets through cost-sensitive learning and threshold optimization guided by the precision-recall curve. Transparency is embedded at the core: SHAP-based explainability generates per-prediction feature-level rationales that are surfaced directly on the advertiser dashboard, converting opaque model decisions into actionable human-readable insights. The complete solution is implemented as a full-stack web application with a React frontend, Node.js/Express backend, Python Flask ML microservice, and MongoDB data layer. Evaluated on the large-scale TalkingData AdTracking benchmark, the deployed XGBoost model achieves an AUC of 0.9549 and a recall of 0.7673, while the hybrid LightGBM–XGBoost ensemble reaches 0.9815 AUC, demonstrating strong predictive performance and practical deployability.

Keywords: Ad Click Fraud, XGBoost, LightGBM, SHAP, Feature Engineering, Real-Time Detection, IP Mitigation.

INTRODUCTION

Digital advertising operates on a foundational assumption: that every recorded click represents a deliberate, genuine action by a real user. Advertisers commit substantial budgets based on this assumption, trusting that click counts, click-through rates, and conversion metrics faithfully reflect authentic audience engagement. Ad click fraud violates this contract at scale. Automated bots simulate browsing behavior across thousands of IP addresses; organized click farms employ workers to manually trigger advertisements; competing advertisers deliberately

generate clicks to exhaust rivals' daily budgets. Industry estimates place global annual losses from click fraud in the billions of dollars, making it one of the most financially damaging threats in the digital economy .

The consequences extend beyond direct financial loss. When a significant fraction of recorded clicks are fraudulent, key performance indicators such as cost-per-click, click-through rate, and return on ad spend lose their diagnostic value. Advertising campaigns are then optimized against corrupted signals rather than genuine engagement, leading to systematic misallocation of marketing investment. Over time, widespread fraud erodes advertiser confidence in digital channels as a whole, threatening the economic sustainability of the ecosystem that supports publishers, app developers, and content platforms.

Conventional detection relies on rule-based systems that flag clicks when predefined conditions are met — for example, more than a threshold number of clicks per minute from a single IP, or a click-through rate above a fixed ceiling. These systems are computationally inexpensive and straightforward to interpret, but their rigidity is precisely what adversaries exploit. Fraudsters distribute activity across large pools of IP addresses, introduce randomized inter-click delays, rotate device and browser identifiers, and calibrate click volumes to remain just below detection thresholds. Against an adaptive, deliberate adversary, static rules degrade rapidly in effectiveness and require continuous manual maintenance.

This work contributes an end-to-end system that integrates high-performance fraud detection with real-time deployment infrastructure and transparent explainability. The system processes each incoming click through a two-stage pipeline: explicit rule-based filters handle obviously fraudulent patterns without incurring ML inference cost, while a trained XGBoost model scores the remainder using a 21-dimensional behavioral feature vector. SHAP-generated explanations identify the specific features driving each fraud decision. Detected fraudulent IP addresses are immediately blocked from future clicks, completing the loop from detection to mitigation. The entire solution is deployed as a functional full-stack web application with a near real-time advertiser dashboard — demonstrating that research-grade fraud detection can be structured as an operational, user-facing platform.

I. Related Work

1.1 Early Statistical and Rule-Based Approaches

The earliest contributions to click fraud detection were primarily empirical and descriptive. Stone-Gross et al. conducted one of the first rigorous analyses of fraudulent activity in online ad exchanges, characterizing the behavioural signatures of botnets and click farms and establishing a foundational taxonomy of attack types. Metwally et al. addressed the narrower problem of duplicate click detection in click streams, proposing deduplication filters for the most straightforward repeated-click attacks. While valuable for establishing the problem space and vocabulary, both approaches depended on manually specified rules and fixed thresholds that offered no mechanism for adapting to evolving adversarial strategies.

1.2 Classical Machine Learning Methods

As labelled clickstream datasets became available, researchers applied supervised learning methods to fraud classification. Linear models including Logistic Regression and Support Vector Machines, implemented using scikit-learn, demonstrated improved generalization over rule-based baselines by learning patterns from behavioural features such as session duration, device consistency, and click frequency. Hoang and Nguyen demonstrated that machine learning approaches applied to mobile advertising environments yield strong fraud detection results, confirming the cross-platform validity of the technique. However, the linearity of these models

constrained their ability to capture the complex, higher-order feature interactions that sophisticated fraud patterns exploit — a gap that ensemble methods subsequently addressed.

1.3 Gradient Boosting and Ensemble Methods

Tree-based ensemble models marked the most significant advance in fraud detection capability. Random Forest and Gradient Boosted Decision Trees proved highly effective for structured click data by naturally modeling nonlinear relationships and feature interactions. Wang and Dong demonstrated that Gradient Boosted Trees combined with behavioral feature engineering achieve top-tier results on click fraud benchmarks. LightGBM and XGBoost further advanced the state of the art through efficient histogram-based tree construction that scales to hundreds of millions of records. Solutions built on these frameworks dominated the TalkingData AdTracking Fraud Detection challenge , which remains the primary public benchmark for this problem. Zhou provides a comprehensive theoretical treatment of ensemble methods that underlies the design choices made in this work.

1.4 Feature Engineering and Imbalance Handling

Iqbal et al. established through systematic study that feature engineering contributes at least as much to detection performance as model selection. Temporal aggregation features, count-based behavioural summaries, and target encoding of categorical fraud rates collectively provide discriminative power that raw click attributes cannot supply. Li et al. demonstrated that inter-click timing patterns from the same IP or device are particularly strong fraud indicators. Micci-Barreca introduced target encoding — replacing high-cardinality categorical variables with historical outcome rates — which has since become a standard component of fraud detection feature pipelines. For class imbalance, Chawla et al. proposed SMOTE-based oversampling, while He and Garcia provided a comprehensive comparative analysis of imbalanced learning strategies. Davis and Goodrich and Sokolova and Lapalme established the theoretical basis for precision-recall evaluation that guides the threshold optimization approach used in this work.

1.5 Model Explainability

The SHAP (SHapley Additive Explanations) framework provides a game-theoretic approach to attributing model predictions to individual input features. Lundberg and Lee demonstrated that SHAP values satisfy desirable axiomatic properties — consistency, local accuracy, and missingness — that earlier feature importance methods lack. In fraud detection specifically, SHAP bridges the gap between predictive accuracy and operational transparency, enabling investigators to understand why a specific click was flagged rather than accepting an opaque binary verdict. The reviewed literature collectively establishes strong foundations in modeling and feature engineering, but almost universally focuses on predictive performance metrics without addressing system integration, real-time deployment, or user-facing infrastructure. This work directly targets that gap.

II. METHODOLOGY

2.1 Dataset and Preprocessing

The system is trained and evaluated on the TalkingData AdTracking Fraud Detection dataset , one of the largest and most widely cited public benchmarks for click fraud research. The dataset contains approximately 184 million click records with six raw attributes per event: IP address, application identifier, device type, operating system, advertising channel, and click timestamp. The binary label indicates whether the click was followed by a genuine app installation within the attribution window; non-converting clicks are treated as potentially fraudulent.

Fraudulent clicks represent only approximately 0.264% of total records — an extreme class imbalance that makes standard accuracy evaluation misleading and demands specialized training strategy.

To balance computational feasibility with representative coverage, the 50 most recent million records are selected and sorted chronologically. A temporal 90/10 train-validation split is applied: the first 45 million records form the training set and the final 5 million the validation set. This ordering strictly mirrors real-world deployment conditions, where the model is always evaluated on events that occur after its training window, preventing data leakage and ensuring that validation performance reflects genuine temporal generalization. Preprocessing removes records with missing or inconsistent values, applies ordinal encoding to categorical attributes, and drops columns unavailable at inference time — including attribution timestamps and intermediate derived fields that would constitute lookahead.

2.2 Handling Class Imbalance

A naive classifier that labels every click as genuine would achieve over 99.7% accuracy while detecting no fraud. Accuracy is therefore disqualified as a primary evaluation metric for this problem. The system addresses class imbalance through two complementary mechanisms. First, the scale_pos_weight parameter in both LightGBM and XGBoost is set equal to the ratio of negative-class to positive-class samples computed from the training set. This multiplies the loss contribution of every fraudulent sample during tree construction, biasing the model toward sensitivity to the minority class without any modification to the raw training data. Second, a post-training threshold sweep evaluates candidate classification thresholds from 0.01 to 0.30 on the validation set, selecting the value that maximizes the F1-score across the precision-recall curve. The deployed threshold is set at 0.5, applied after rule-based pre-filters have already removed the most obvious fraud events.

2.3 Feature Engineering

The feature engineering pipeline transforms raw click log attributes into a 21-dimensional behavioral feature vector. Features are organized into six functional categories, each designed to capture a distinct aspect of fraudulent behavior. Table 1 provides a complete summary of all feature categories along with the signals they capture and the fraud patterns they are designed to detect.

Table 1: Feature Engineering — Categories, Signals, and Targeted Fraud Patterns

Category	Features Included	What It Measures	Fraud Signal
Time Context	hour, day, day_of_week	When the click occurred	Bots often active at unusual hours; bursts on specific days
Time Gaps	ip_time_diff, ip_app_time_diff	Speed of successive clicks from same IP	Gap < 0.5 s is physically impossible for a human user
Click Volumes	ip_count, ip_app_count, ip_device_os_count	Total historical clicks per IP / combination	Abnormally high counts signal automated volume generation
Behavioral Diversity	unique_app/channel/device_per_ip, ip_hour_clicks	Range of activity breadth per IP	Low diversity + high burst rate indicates focused bot activity

Category	Features Included	What It Measures	Fraud Signal
Target Encoding	app_te, device_te, os_te, channel_te	Historical fraud rate for each category value	High encoded rate is the strongest single fraud predictor
Raw Identifiers	ip, app, device, os, channel	Base categorical attributes from click log	Low standalone power; amplified by engineered features above

Target encoding is the most impactful individual step in the pipeline. Each of the four categorical identifiers — application, device, OS, and channel — is replaced with its historical fraud rate computed from the training set. This encoding embeds the cumulative fraud history of every categorical value into a single continuous signal that directly informs the model's fraud probability estimate. At inference time, categorical values not present in the training data are assigned the global mean fraud rate as a conservative fallback, preventing cold-start failures. These four target-encoded features consistently rank as the top predictors in feature importance analysis, confirming the design hypothesis that historical fraud rate is the strongest predictor of future fraud likelihood for any given categorical attribute.

Time-gap features are computed by querying the 50 most recent clicks from the same IP address stored in MongoDB and measuring the elapsed seconds since the previous click. A gap below 0.5 seconds is physically impossible for a human user interacting with an advertisement and is intercepted by the rule-based pre-filter before the ML model is invoked. Larger gap values serve as a consistent indicator of genuine human browsing behavior and appear in SHAP analysis as one of the most reliable fraud-reducing signals. This complementarity between the rule filter and the ML model — where the rule handles the extreme tail and the model handles the nuanced middle ground — is a deliberate architectural choice that improves both efficiency and sensitivity.

2.4 Model Training and Configuration

Both LightGBM and XGBoost are trained on the 45-million-record training partition. Table 2 summarizes the full configuration of both models side by side.

Table 2: Model Configuration — LightGBM vs XGBoost (Deployed)

Parameter	LightGBM	XGBoost (Deployed)
Boosting Rounds / Estimators	1,200 rounds with early stopping on AUC	1,000 estimators (fixed)
Tree Structure	128 leaves — leaf-wise growth	Max depth = 8 — level-wise growth
Learning Rate	0.05	0.05
Tree Construction	Histogram-based (fast, memory-efficient)	Histogram-based (hist method)
Feature / Sample Subsampling	80% at each boosting round	80% at each boosting round
Imbalance Handling	scale_pos_weight = neg/pos ratio	scale_pos_weight = neg/pos ratio
Evaluation Metric	AUC on validation set	AUC on validation set

A hybrid ensemble prediction is produced as a weighted blend of both model outputs: ensemble score = 0.4 × LightGBM score + 0.6 × XGBoost score. The 0.6 weight assigned to XGBoost reflects its more stable probability calibration across different threshold settings. This ensemble achieved a validation AUC of 0.9815. On a separate temporal holdout of 2 million records drawn from row 128 million onward — representing a genuinely distinct future time window — the hybrid achieved an AUC of 0.9414, confirming robust generalization across temporal distribution shifts. XGBoost alone is selected for the deployed system due to its lower inference latency, superior stability under threshold variation, and native compatibility with the SHAP TreeExplainer, which requires a single underlying tree structure to compute consistent Shapley value attributions.

2.5 Hybrid Detection Pipeline

The detection pipeline operates in two sequential stages. Stage 1 applies three explicit rule-based filters that intercept obviously fraudulent clicks without invoking the ML model, reducing computational overhead and reserving probabilistic inference for behaviourally ambiguous events. Table 3 describes each filter, its trigger condition, and the specific fraud pattern it targets.

Table 3: Stage 1 — Rule-Based Fraud Filters

Filter Name	Trigger Condition	Fraud Pattern Intercepted
Rapid Click	ip_time_diff < 0.5 seconds	Bot-speed clicking — sub-second response is physically impossible for a human user interacting with an ad
Burst Attack	More than 10 clicks from same IP within any 10-second window	High-frequency automated bursts targeting a single campaign to rapidly exhaust budget
Hourly Flood	ip_hour_clicks exceeds 40 in the current hour	Sustained high-volume fraud that mimics heavy but believable user activity over a longer window

If no rule-based condition fires, Stage 2 forwards the full 21-feature vector to the Flask ML microservice. The microservice applies runtime target encoding by substituting categorical IDs with preloaded fraud-rate maps, runs the XGBoost inference pass, applies the 0.5 classification threshold, and computes SHAP values using the TreeExplainer. The top three contributing features are selected by absolute SHAP magnitude, converted to human-readable text using a template formatter, and returned alongside the fraud probability and binary verdict. The application backend stores the complete click record — including prediction, probability, and explanation — in MongoDB, inserts the IP address into the blocked IP collection if fraud is detected, and updates the advertiser dashboard. Subsequent clicks from a blocked IP are rejected at the gateway before any feature computation occurs, eliminating repeat-offender processing cost entirely.

2.6 System Architecture

The full system is organized into six independently replaceable modular layers. Table 4 presents each layer with its technology stack, primary responsibility, and a key implementation detail.

Table 4: System Architecture — Modular Layer Summary

Layer	Technology	Primary Responsibility	Key Implementation Detail
Client Layer	React.js	Advertiser UI — click capture, fraud dashboard, campaign management	Dashboard polls backend every 5 s; fraud charts rendered via Recharts
Application Gateway	Node.js / Express	Authentication, feature computation, rule filters, pipeline orchestration	Port 5000; computes all 21 features from last 50 IP clicks in MongoDB
Inference Layer	Python Flask	ML model serving, SHAP computation, fraud response formatting	Port 5001; isolated from public internet — internal access only
Model Layer	XGBoost + SHAP	Fraud probability scoring and feature-level attribution	UBJ-format model; TreeExplainer initialized once at service startup
Data Layer	MongoDB	Persistent storage of all system entities	4 collections: users, advertisements, click logs with predictions, blocked IPs
Visualization Layer	Recharts Dashboard	Fraud analytics and campaign performance reporting	Per-ad metrics, time-series fraud rate trends, session-level activity

The architectural separation of the Flask ML microservice from the main Node.js application server is a deliberate security decision: the model inference endpoint is reachable only via internal service communication, not directly from the public internet. This isolation limits the attack surface and prevents unauthorized model queries or adversarial probing of the fraud decision boundary. All incoming click payloads are validated against a strict schema at the application gateway before any processing begins, rejecting malformed or anomalous inputs before they can propagate into the feature computation pipeline.

2.7 SHAP Explainability Integration

The SHAP TreeExplainer is initialized a single time when the Flask microservice starts, loading the serialized XGBoost model and computing the background dataset statistics required for efficient Shapley value computation. For every click event that reaches Stage 2, SHAP values are computed across all 21 features. The three features with the largest absolute SHAP contributions are selected and formatted into natural-language explanations delivered to the dashboard alongside each fraud verdict. A representative explanation for a detected fraud event might read: 'High historical fraud rate for this app (app_te = 0.84) strongly increased fraud probability. Rapid inter-click timing from the same IP (ip_time_diff = 0.3 s) provided an additional strong fraud signal. Elevated hourly click count from this IP (ip_hour_clicks = 37) further contributed to the fraud classification.'

These explanations serve multiple practical functions. For individual click review, they allow an advertiser or fraud analyst to evaluate the plausibility of a specific fraud decision and make an informed judgment about maintaining or lifting an IP block. At the aggregate level, patterns in SHAP explanations across many fraud

detections reveal which fraud types are most active in a given campaign — whether the dominant signal is high-volume burst activity, specific high-risk app targets, or unusual device-channel combinations. This aggregated intelligence supports proactive advertiser response beyond automated blocking, enabling campaign-level adjustments to placements, targeting parameters, or bid strategies.

III. Results and Discussion

3.1 Performance Summary

The XGBoost model was evaluated on the temporal validation partition. Table 5 presents a complete performance summary covering both the deployed standalone XGBoost model and the hybrid ensemble, along with the confusion matrix breakdown.

Table 5: Performance Summary — Deployed XGBoost and Hybrid Ensemble

Metric	Score	Interpretation
AUC — Hybrid Ensemble (Validation)	0.9815	Best combined performance of LightGBM + XGBoost blend
AUC — XGBoost Standalone (Validation)	0.9549	Deployed model; strong discriminative separation
AUC — Temporal Holdout (2M records)	0.9414	Confirms generalization across future time shifts
Recall at Deployed Threshold	0.7673	76.7% of all fraudulent clicks correctly intercepted
True Positives	3,317	Fraudulent clicks correctly flagged and IP blocked
True Negatives	1,968,379	Legitimate clicks correctly passed through
False Positives	27,298	Legitimate clicks wrongly flagged — recoverable via unblock
False Negatives	1,006	Missed fraud — most costly error; direct budget loss

The deployed XGBoost model achieves an AUC of 0.9549, reflecting strong discriminative separation between fraudulent and legitimate clicks across all classification thresholds. The ROC curve rises steeply in the low false-positive-rate region, indicating that the model successfully flags a large proportion of fraud while incorrectly blocking very few legitimate users. At the deployed threshold of 0.5, the recall of 0.7673 means that more than three-quarters of all fraudulent clicks in the validation set are correctly intercepted and the originating IP addresses are automatically blocked.

The 1,006 False Negatives represent the 23.3% of actual fraud that evades detection at the current threshold — the most costly error type, as each represents budget consumed by an undetected fraudulent click. The 27,298 False Positives represent legitimate clicks incorrectly flagged, a false positive rate of approximately 1.4% among genuine clicks. While any false positive is undesirable, this rate is manageable in practice: legitimate users can be unblocked through manual review or automated appeal processes, and the economic cost of a false positive is substantially lower than that of a missed fraud event.

3.2 Feature Importance Analysis

Feature importance computed using the XGBoost gain metric reveals a clear predictive hierarchy. Target encoding features dominate: app_te ranks first with the highest gain score, followed by channel_te and device_te. This confirms that the historical fraud rate of an application or channel is the most powerful individual predictor of whether a new click from that entity is fraudulent — a finding consistent with the literature [11][9] and with the intuition that fraud tends to concentrate around specific high-value or low-quality inventory targets.

Count-based behavioral features — ip_app_count and ip_count — rank fourth and fifth, reflecting that high cumulative click volumes from a given IP or IP-app combination reliably indicate automated activity. The raw IP address identifier contributes moderate importance, as certain IP ranges exhibit persistent elevated fraud rates that the model learns to associate with risk. Time-gap features (ip_time_diff, ip_app_time_diff) contribute lower average gain in the importance metric, but SHAP analysis reveals that their extreme values carry disproportionately strong directional influence on individual predictions — very small gaps are powerful fraud signals, while very large gaps are equally powerful legitimacy signals.

3.3 SHAP Analysis and Behavioral Insights

The global SHAP summary plot, computed across the full validation set, surfaces two key behavioral insights. First, higher values of all four target-encoded features consistently push predictions toward fraud, confirming that applications, channels, devices, and operating systems with historically high fraud rates continue to generate fraudulent activity in the validation period. This temporal stability of categorical fraud rates is what makes target encoding such a powerful feature: fraud tends to be concentrated around specific inventory that either attracts fraudulent traffic structurally or has been deliberately targeted by fraud operators.

Second, larger values of ip_time_diff reduce predicted fraud probability — genuine users exhibit natural inter-click delays consistent with reading ad content, navigating to a destination, and returning, while automated systems cannot simulate this variability at scale without sacrificing the click volume needed to be economically effective. The SHAP analysis also validates the division of labor between the two detection stages: the rule-based filters intercept clicks at the extreme left tail of the ip_time_diff distribution (below 0.5 seconds), while the ML model handles the more nuanced region where timing is suspicious but not definitively impossible, relying on the combined evidence of multiple features to reach a verdict.

IV. Challenges and Future Work

4.1 Identified Limitations

Static Target Encodings. Target encoding maps are computed offline from the training dataset and remain fixed throughout the deployment lifecycle. As fraud patterns shift — new fraudulent applications emerge, previously clean channels become compromised, bot operators migrate to new device profiles — static encodings become progressively misaligned with current conditions. Since target encoding features are simultaneously the most predictive and the most temporally sensitive components of the feature vector, encoding staleness poses the highest risk of model degradation over time. Continuous encoding refresh through streaming computation is the primary mitigation for this limitation.

Partial Training Data. Hardware memory constraints limited training to 50 million of the 184 million available records. While the model demonstrates strong generalization to temporal holdout data, training on the full dataset would expose the model to a wider range of fraud patterns — particularly rare but distinct attack types that appear

infrequently in the 50-million subset. Distributed training using Apache Spark with MLlib or XGBoost's native distributed mode would make full-dataset training computationally tractable without per-machine memory limits.

Absence of Online Learning. The deployed model is static between scheduled retraining cycles and cannot adapt in real time to newly emerging fraud strategies. In an adversarial environment where fraud operators actively probe detection boundaries and adjust their behavior in response to blocking patterns, static models will eventually develop exploitable blind spots. Incremental gradient boosting, where new labeled fraud events are used to update model weights without full retraining, or complementary unsupervised anomaly detection layers that do not require labeled examples of new patterns, would significantly improve adaptive resilience.

Fixed Classification Threshold. The uniform 0.5 threshold applies across all campaigns and advertiser contexts. Different campaigns have fundamentally different tolerance profiles for false positives and false negatives: a performance campaign on a high-cost keyword may prioritize maximum recall to minimize fraudulent spend, while a broad awareness campaign may accept a higher false positive rate to preserve reach. Per-campaign or dynamically adjusted thresholds, configurable through the advertiser dashboard, would better align detection sensitivity with individual business requirements.

4.2 Future Development Directions

Streaming feature engineering using Apache Kafka or Apache Flink would enable continuous updating of target encoding maps and count-based aggregations as new click data arrives, keeping the feature representation aligned with current fraud patterns without requiring full retraining cycles. Containerized cloud deployment using Docker and Kubernetes would provide the elasticity, fault tolerance, and geographic distribution needed for production-scale traffic, with autoscaling inference replicas to handle volume spikes during campaign launches or coordinated fraud attacks. Incorporating real-time anomaly detection — particularly unsupervised methods capable of flagging statistically unusual click patterns not seen during training — would extend the system's detection coverage beyond the fraud patterns represented in historical labeled data, improving resilience against genuinely novel attack strategies.

V. Conclusion

This paper has presented a modular, end-to-end system for real-time ad click fraud detection and automated IP mitigation. The system integrates a 21-feature behavioral representation, a trained XGBoost classifier, SHAP-based explainability, and a full-stack web application into a cohesive operational platform. Evaluated on the large-scale, highly imbalanced TalkingData AdTracking benchmark, the deployed model achieves an AUC of 0.9549 and a recall of 0.7673, while the hybrid LightGBM–XGBoost ensemble reaches an AUC of 0.9815 — results that confirm both strong predictive performance and robust temporal generalization.

The two-stage hybrid detection pipeline — rule-based pre-filters for obviously fraudulent patterns followed by probabilistic ML inference for ambiguous cases — balances computational efficiency with detection sensitivity. SHAP explainability transforms fraud decisions from opaque binary verdicts into transparent, feature-level rationales accessible to advertisers directly through the dashboard, embedding interpretability as a core system capability rather than an analytical afterthought. The modular six-layer architecture — React, Node.js/Express, Flask, XGBoost, MongoDB, Recharts — demonstrates that each component can be independently maintained or upgraded without disrupting the overall pipeline.

The four identified limitations — static target encodings, partial training data, absent online learning, and fixed threshold — define a concrete and prioritized roadmap for future development. Addressing these through streaming feature updates, distributed training on the full dataset, incremental model adaptation, and per-campaign threshold configuration will progressively strengthen the system's resilience against adaptive adversaries. The results and architecture presented here establish a strong, deployable foundation and confirm that machine learning-based fraud detection can be built not merely as an accurate model but as a practical, transparent, and operationally coherent system.

VI. REFERENCES

[1] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," Proc. 22nd ACM SIGKDD, pp. 785–794, 2016.

[2] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," NeurIPS, vol. 30, pp. 3146–3154, 2017.

[3] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," Annals of Statistics, vol. 29, no. 5, 2001.

[4] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.

[5] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," NeurIPS, vol. 30, 2017.

[6] S. M. Lundberg et al., "From Local Explanations to Global Understanding with Explainable AI for Trees," Nature Machine Intelligence, vol. 2, no. 1, 2020.

[7] N. V. Chawla et al., "SMOTE: Synthetic Minority Over-sampling Technique," JAIR, vol. 16, pp. 321–357, 2002.

[8] H. He and E. A. Garcia, "Learning from Imbalanced Data," IEEE TKDE, vol. 21, no. 9, pp. 1263–1284, 2009.

[9] D. Wang and J. Dong, "Ad Click Fraud Detection Using Gradient Boosting with Behavioral Features," Proc. DMBD, pp. 215–225, 2019.

[10] T. Hoang and C. Nguyen, "Detection of Click Fraud in Mobile Advertising Using ML," IJACSA, vol. 11, no. 8, 2020.

[11] M. Iqbal et al., "Feature Engineering for Fraud Detection in Large-Scale Click Streams," IEEE Access, vol. 9, 2021.

[12] F. Li, Y. Li, and J. Zhao, "Click Fraud Detection Based on Temporal Behavior Analysis," Computers & Security, vol. 109, 2021.

[13] B. Stone-Gross et al., "Understanding Fraudulent Activities in Online Ad Exchanges," ACM SIGCOMM IMC, 2011.

[14] A. Metwally, D. Agrawal, and A. El Abbadi, "Duplicate Detection in Click Streams," Proc. WWW, 2005.

[15] TalkingData, "AdTracking Fraud Detection Challenge," Kaggle, 2018. https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection

[16] M. Micci-Barreca, "Preprocessing Scheme for High-Cardinality Categorical Attributes," ACM SIGKDD Explorations, vol. 3, no. 1, 2001.

[17] J. Davis and M. Goadrich, "The Relationship Between Precision-Recall and ROC Curves," Proc. ICML, 2006.

[18] M. Sokolova and G. Lapalme, "Performance Measures for Classification Tasks," IPM, vol. 45, no. 4, 2009.

[19] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," JMLR, vol. 12, 2011.

[20] W. McKinney, "Data Structures for Statistical Computing in Python," Proc. SciPy, 2010.
[21] C. R. Harris et al., "Array Programming with NumPy," Nature, vol. 585, 2020.
[22] XGBoost Developers, "XGBoost Documentation," 2024. https://xgboost.readthedocs.io
[23] LightGBM Contributors, "LightGBM Documentation," Microsoft, 2024. https://lightgbm.readthedocs.io
[24] SHAP Contributors, "SHAP Documentation," 2024. https://shap.readthedocs.io
[25] Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms, CRC Press, 2012.
[26] Pallets Projects, "Flask Documentation," 2024. https://flask.palletsprojects.com
[27] OpenJS Foundation, "Express.js Documentation," 2024. https://expressjs.com
[28] MongoDB Inc., "MongoDB Manual," 2024. https://www.mongodb.com/docs/manual/
[29] Meta Open Source, "React Documentation," 2024. https://react.dev