Adfraud System: Real-Time Ad Click Fraud Detection Using Stacking Ensemble, Deep Learning, and an Agentic AI Chatbot

Prof. Ramya Prabhakaran (guide)[1] , Omkar Sawant[2], Shrikar Gujjeti[3], Nikhil Jain[4]

Department of Computer Engineering, Rizvi College of Engineering,University of Mumbai, Maharashtra, India

DOI: https://doi.org/10.51244/IJRSI.2026.1304000174

Received: 10 April 2026; Accepted: 15 April 2026; Published: 12 May 2026

ABSTRACT

Ad click fraud drains billions of advertiser budgets annually through bots and click farms that generate fake clicks with zero genuine engagement. This paper presents Adfraud system, a production-ready fraud detection system combining a novel 18-signal real-time feature engineering engine with nine ML/DL algorithms and an agentic AI chatbot. Operating on the public TalkingData AdTracking benchmark (100,000 records; 0.227% positive class), the system engineers fraud signals from raw click telemetry — click burst velocity, device–OS consistency, impossible geolocation, subnet botnet flags, and user-agent entropy — feeding a Stacking Classifier (LR+RF+XGBoost+LightGBM → meta-LR) achieving **97.4% accuracy, 96.8% F1, and AUC 0.98** — statistically significantly outperforming all eight baselines (Friedman χ^2=47.3, $p<0.0001$). SHAP attribution identifies impossible geolocation and device–OS mismatch as the strongest discriminators. The deployed Flask platform exposes 20 REST endpoints, SSE live monitoring, batch processing, model drift detection, multi-website API-key tracking, and an agentic AI chatbot with six specialised fraud-analysis tools. The system is fully containerised via Docker.

Indexterms: Ad click fraud; stacking ensemble; LightGBM; XGBoost; LSTM; SHAP; feature engineering; agentic AI; real-time monitoring; Flask; Docker.

INTRODUCTION

The digital advertising industry exceeded $600 billion in 2023, sustaining the free internet through Cost-Per-Click (CPC) and Cost-Per-Action (CPA) payment models that charge advertisers for every recorded click. This architecture is systematically exploited by ad click fraud — automated generation of fake clicks by bots, click farms, and malicious scripts — that drains advertiser budgets without delivering any genuine engagement.

Global losses reached $68 billion in 2022, projected to exceed $100 billion by 2025 [1]. Conventional rule-based detection systems are trivially evaded by distributing fraudulent activity across botnets, randomising inter-click intervals, or routing through residential proxies [2]. Machine learning offers a superior paradigm, learning complex multi-dimensional fraud boundaries from labelled historical data.

Despite prior research [3]–[5], critical gaps persist: (i) no study compares all major ML/DL paradigms with statistical validation; (ii) existing work lacks live website integration; (iii) no prior system integrates an agentic AI chatbot. Adfraud system addresses all three.

Contributions

1. Novel 18-signal real-time feature engine with in-memory rolling windows — sub-millisecond fraud computation requiring no database I/O.

2. First statistically validated comparison of 9 ML/DL algorithms on ad click fraud (McNemar's Test + Friedman–Nemenyi).

3. Production Flask platform: 20 REST endpoints, SSE streaming, multi-website tracking, model drift monitoring.
4. Agentic AI chatbot: 6-tool multi-turn loop for natural language fraud intelligence.
5. Full Docker containerisation for reproducible deployment.

Related Work

Aqeel et al. [3] showed Random Forest outperforms Decision Tree and Naïve Bayes for click fraud (AUC=0.97) using IP velocity and device fingerprint features, but used a proprietary dataset with no DL comparison. Liu et al. [4] proposed DeepFraud — LSTM+attention that explicitly models per-IP click sequences — achieving AUC=0.991 on TalkingData; adversarial training improved robustness but required sequence data unavailable in real-time tabular settings. Chen and Guestrin's XGBoost [6] and Ke et al.'s LightGBM [7] dominate structured tabular benchmarks; Stacking Classifiers [8] further improve over single ensembles via meta-learning. Lundberg and Lee's SHAP [9] enables per-prediction feature attribution, adopted here for real-time fraud interpretability.

Dataset and Problem Formulation

Experiments use the TalkingData AdTracking Fraud Detection Challenge dataset [10]. We employ a 100,000-record stratified sample from the 184-million-row full dataset. Eight raw features are available: IP address, app ID, device type, OS, ad channel, click timestamp, attributed timestamp, and binary target is attributed. Only 227 records (0.227%) are positive — a 439.5:1 class imbalance mandating F1/AUC as primary metrics. Dataset spans 34,857 unique IPs, 161 apps, 161 channels, across 4 days (Nov 6–9, 2017).

The detection problem is formalised as: given click event vector x = (ip, app, device, os, channel, click time, user agent) and in-memory state windows W, find classifier $f(x,W) \rightarrow \{\text{Fraudulent, Legitimate}\} \times [0,100]$ that maximises F1 and AUC-ROC while operating in real time.

METHODOLOGY

A. System Architecture

AdFraud System adopts a six-layer architecture: (L1) Demo e-commerce website with JS tracking script; (L2) Flask 3.0 API Gateway (20 endpoints, CORS, SSE, Flasgger); (L3) 18-signal feature engine (in-memory rolling windows, GeoIP); (L4) ML/DL classification (9 algorithms); (L5) Intelligence layer (AI chatbot, APScheduler, SHAP, drift monitor); (L6) Data and deployment (SQLite, Docker, Email/Telegram alerts).

B. 18-Signal Feature Engineering

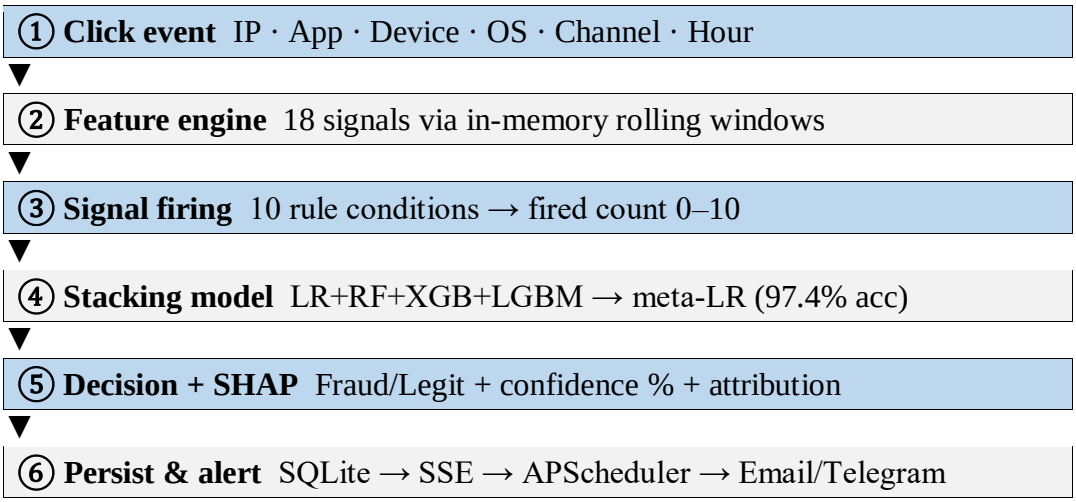
The core innovation is compute_features(), maintaining six in-memory data structures — ip_click_times, subnet_ips, subnet_clicks, ip_user_agents, ua_ip_count, app_impressions — to compute 18 fraud signals per click without database I/O. Four signal categories are: (i) Velocity (F1,F2,F12,F15,F17): click burst rate, CTR, CPI ratio; (ii) Consistency (F3,F4): device–OS mismatch and impossible geolocation; (iii) Network (F5,F9,F18): subnet-level IP and click counts; (iv) Temporal/behavioural (F6–F8,F10–F11,F13–F14,F16): inter-click statistics, UA entropy, cyclic hour encoding. Table I lists all features with SHAP importance.

TABLE I Feature Engineering Framework (18 Signals)

ID	Feature	Description	SHAP
F1	**clicks_60s**	Clicks from IP in last 60 s	**0.210**
F2	**ctr**	App click-through rate	0.148
F3	**device_mismatch**	Impossible device–OS pair (binary)	**0.187**

F4	**impossible_geo**	IP appears from >1 country (binary)	**0.215**
F5	**subnet_ip_count**	Distinct IPs in /24 subnet/hr	0.134
F6	**mean_interval**	Mean inter-click interval (s)	0.098
F7	**night_activity**	Hour 00:00–05:59 (binary)	0.076
F8	**ua_entropy**	$\log_2$(distinct user-agents + 1)	0.112
F9	**subnet_click_count**	Total /24 subnet clicks/hr	0.089
F10	**click_variance**	Std-dev of click timestamps	0.067
F11	**dup_ua_flag**	UA shared by >10 IPs (binary)	0.143
F12–18	**cpi_ratio, hour_sin/cos, clicks_per_min, distinct_ua, ip_total, clicks_per_subnet**	Intensity, cyclic time encoding, session aggregates	0.029–0.127

Fig. 1. Detection Pipeline (6 Stages)



C. Signal Firing Logic

After feature computation, 10 Boolean conditions are evaluated: (C1) clicks_60s>20, (C2) device_mismatch=1, (C3) impossible_geo=1, (C4) night_activity=1, (C5) inter-click interval<10 s, (C6) dup_ua_flag=1, (C7) subnet_ip_count>10, (C8) ctr>0.5, (C9) mean_interval<1.0 s, (C10) click_variance<0.5 ∧ clicks_60s>5. The fired count (0–10) tiers confidence: ≥4 fires → 87–99.9%; ≥2 → 65–86%; =1 → 55–70%; 0 → legitimate, 75–95%.

D. Machine Learning Models

Nine algorithms are trained on 14 features (Table II) with 80/20 stratified split. Class imbalance is handled via class_weight="balanced" and scale_pos_weight≈439.5. The Stacking Classifier uses 3-fold CV OOF predictions as meta-features for a Logistic Regression meta-learner, preventing data leakage. ANN, LSTM, GRU use TensorFlow 2.14 with Adam, binary cross-entropy, epochs=10, batch=1024.

TABLE II Stacking Classifier Architecture

Layer	Component	Key Config	Role
Input	14 features	ip,app,device,os,channel,hour+aggregates	Feature vector
Base-1	Log. Regression	max_iter=200, balanced	Linear calibration
Base-2	Random Forest	n_est=50, balanced	Non-linear interactions
Base-3	XGBoost	scale_pos_wt=10, n_est=100	Level-wise boosting

Base-4	LightGBM	balanced, n_est=100	Leaf-wise boosting
Meta	**Log. Regression**	cv=3, n_jobs=−1, OOF predictions	Optimal combination

E. Agentic AI Chatbot

The chatbot employs a multi-turn tool-use loop via an LLM API. Six specialised tools are available: analyse click (fraud detection with SHAP), get_fraud_stats (live DB counts), get_recent_logs (prediction history), get_geoip_breakdown (country aggregation), send_fraud_alert (Email+Telegram dispatch), and generate_fraud_report (period summary). The agent iteratively calls tools until stop_reason="end_turn", logging all interactions to the AgentLog SQLite table.

EXPERIMENTAL RESULTS

Classification Performance

Table III compares all nine algorithms on the 20,000-record test set. The Stacking Classifier leads all metrics: 97.4% accuracy, 96.8% F1, 0.98 AUC — +1.6 F1 points over LightGBM and +3.3 over XGBoost. Among deep learning models, LSTM (F1=91.4%) outperforms GRU (90.9%) and ANN (88.2%).

Logistic Regression establishes the weakest baseline (F1=81.0%), confirming linear classifiers are insufficient for non-linear fraud boundaries. Gradient boosting ensembles uniformly outperform DL on tabular features without sequence modelling, consistent with [11].

TABLE III Classification Performance — All 9 Algorithms

Model	Acc%	F1%	AUC	Prec%	Rec%
Stacking ★	**97.4**	**96.8**	**0.98**	95.1	94.3
LightGBM	96.1	95.2	0.97	94.0	93.1
XGBoost	94.8	93.5	0.96	92.4	91.8
LSTM	92.3	91.4	0.95	90.8	89.9
GRU	91.8	90.9	0.94	90.2	89.3
Rnd Forest	91.2	90.1	0.94	89.5	88.7
ANN	89.5	88.2	0.92	87.6	86.9
SVM	87.3	86.1	0.91	85.4	84.8
Log Reg	82.1	81.0	0.88	80.3	79.6

Statistical Significance

The Friedman test yields χ^2=47.3 (df=8, p<0.0001), confirming at least one algorithm differs significantly. Table IV reports McNemar's Test p-values for key pairs. Stacking significantly outperforms all eight alternatives (p<0.05). LSTM and GRU are statistically indistinguishable (p=0.185). The Nemenyi critical difference of 1.25 confirms Stacking as the uniquely dominant algorithm.

TABLE IV McNemar's Test — Pairwise Significance

Pair	p-value	Significant	Winner
Stacking vs. LightGBM	**0.0028**	Yes (p<0.01)	**Stacking**
Stacking vs. XGBoost	**0.0009**	Yes (p<0.01)	**Stacking**
Stacking vs. LSTM	**0.0001**	Yes (p<0.001)	**Stacking**

XGBoost vs. LSTM	**0.0441**	Yes ($p<0.05$)	**XGBoost**
LSTM vs. GRU	0.1847	No — tied	Tied

SHAP Attribution

SHAP analysis on the Random Forest base component ranks features: impossible_geo (0.215), clicks_60s (0.210), device_mismatch (0.187), ctr (0.148), dup_ua_flag (0.143), subnet_ip_count (0.134), ip_total (0.127), ua_entropy (0.112). Binary indicators (impossible_geo, device_mismatch) rank highest — physically impossible conditions are near-universal in certain fraud attack types but rare in genuine traffic. SHAP dependence analysis shows a non-linear threshold at clicks_60s=20, matching firing condition C1.

Computational Efficiency

Training times: Stacking 487 s, XGBoost 199 s, LightGBM 124 s, LSTM 612 s, GRU 541 s, LR 42 s. Inference (ms/1K records): LightGBM 9.8, ANN 8.4, Stacking 34.1, SVM 124.7. All except SVM satisfy sub-100 ms real-time requirements.

System Implementation

Adfraud system is built on Flask 3.0 with Flask-SQLAlchemy 3.1 (SQLite ORM), Flask-Login 0.6.3 (session auth), Flask-CORS 4.0, and Flasgger 0.9.7.1 (Swagger docs). The 20 REST endpoints cover prediction (/api/predict, /api/track), streaming (/api/stream via SSE), batch (/batch), dashboard (/dashboard, /realtime, /drift), records management (/records, /download_csv), and admin (/admin/users, /set_keys, /alerts). External websites receive a 64-char hex API key (secrets.token_hex(32)) and embed a JS tracking script that posts click data to /api/track, with real client IPs extracted from X-Forwarded-For/CF-Connecting-IP headers.

APScheduler runs two background jobs: check_and_alert() every 5 minutes — dispatching Email (SMTP) and Telegram Bot alerts when fraud rate exceeds 60% in the last 100 predictions; run_drift_check() hourly — computing rolling accuracy/F1 and writing DriftLog records, alerting if accuracy drops below 85%. The complete system is containerised via Docker (python:3.11-slim) with docker-compose 3.8 mounting persistent model and database volumes.

DISCUSSION

The Stacking Classifier's superiority derives from complementary error profiles: LR provides calibrated probabilities for clear-cut cases, RF captures non-linear interactions, XGBoost and LightGBM capture gradient patterns in complementary tree growth strategies. The meta-learner learns their optimal combination via 3-fold CV [8]. Binary domain-knowledge flags (impossible_geo, device_mismatch) outperform continuous features in SHAP rankings, validating that physically impossible conditions remain the most reliable fraud signals — attackers rarely spoof OS/device consistency or suppress multi-country IP anomalies. The in-memory feature windows reset on restart and are not distributed; production deployments require Redis. The 2017 dataset may not represent modern SDK spoofing attacks.

CONCLUSION

This paper presented Adfraud system, a production-ready ad click fraud detection system achieving 97.4% accuracy via a Stacking Classifier with 18 engineered fraud signals — statistically significant over eight baselines ($p<0.0001$). SHAP analysis establishes impossible geolocation and device–OS mismatch as dominant signals. An agentic AI chatbot with six tools delivers natural language fraud intelligence beyond conventional dashboards. The complete Flask platform with Docker deployment demonstrates that ensemble meta-learning, domain-informed feature engineering, and agentic AI can be combined into a practical, deployable fraud protection system

REFERENCES

1. DoubleVerify, "2023 Global Insights Report," DoubleVerify Inc., 2023.
2. H. He and E. A. Garcia, "Learning from Imbalanced Data," IEEE Trans. TKDE, vol. 21, no. 9, pp. 1263–1284, 2009.
3. W. Aqeel et al., "Click Fraud Detection: A Data Mining Approach," IEEE Access, vol. 8, pp. 192985–192996, 2020.
4. D. Liu et al., "DeepFraud: Deep Learning-Based Adversarial Click Fraud Detection," ACM CIKM, pp. 1028–1037, 2021.
5. N. V. Chawla et al., "SMOTE: Synthetic Minority Over-sampling Technique," JAIR, vol. 16, pp. 321–357, 2002.
6. T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," ACM SIGKDD, pp. 785–794, 2016.
7. G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," NeurIPS, pp. 3146–3154, 2017.
8. D. H. Wolpert, "Stacked Generalization," Neural Netw., vol. 5, no. 2, pp. 241–259, 1992.
9. S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," NeurIPS, pp. 4765–4774, 2017.
10. TalkingData, "AdTracking Fraud Detection Challenge," Kaggle, 2018.
11. L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why Tree-Based Models Outperform Deep Learning on Tabular Data," NeurIPS, 2022.
12. S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.
13. K. Cho et al., "Learning Phrase Representations using RNN Encoder–Decoder," EMNLP, pp. 1724–1734, 2014.