

REAL-TIME BIDDING ANOMALY DETECTION: AN ADAPTIVE FRAMEWORK FOR IDENTIFYING UNUSUAL BID REQUESTS IN PROGRAMMATIC ADVERTISING

Sujan Kumar Reddy Challa ¹

Received 21.06.2026.

Revised 23.08.2026.

Accepted 10.10.2026.

Keywords:

Real-time bidding, programmatic advertising, anomaly detection, ad fraud, adaptive baseline, machine learning.

Original research



ABSTRACT

Real-time bidding (RTB) coordinates automated transactions among publishers, exchanges, and demand-side platforms within millisecond-scale latency windows. The volume and dynamic nature of bid request streams make it difficult to detect abnormal activity such as bot-generated requests, spoofed inventory, invalid traffic, and collusion-based fraud. Rule-based approaches struggle here because valid bidding behavior varies widely across publishers, devices, geographies, and campaigns, producing excessive false positives or negatives. This paper proposes a context-aware framework for detecting anomalous bidding behavior in RTB ecosystems, combining statistical profiling, machine-learning anomaly scoring, and contextual behavioral baselines within a three-layer detection engine that fuses these signals into an interpretable, auditable risk score supporting feature-level attribution. The framework is implemented and evaluated on a reproducible synthetic RTB bid-request stream (250,010 requests, of which 15,010 are injected anomalies spanning bot-generated bursts, spoofed inventory, invalid traffic, and collusion rings), against static rule-based, single-method statistical, and unsupervised machine-learning baselines. At its critical operating threshold the framework achieves precision 0.895, recall 0.782 and F1-score 0.835 at a false-positive rate of 0.0065, detecting 99.9% of bot bursts and 99.1% of invalid traffic. A lower monitoring (suspicious-tier) threshold, sized to a 5% false-positive review budget, raises overall recall to 0.928 and surfaces 42.2% of spoofed-inventory requests, a class every baseline almost entirely misses at its own operating point (best baseline recall 19.1%). Mean scoring latency is 16.595 ms per request (p99 46.962 ms), within the bid-time decision window. Because no labeled production RTB stream is publicly available, the evaluation uses a documented synthetic generator with fixed seeds; all generator parameters are reported so the study can be reproduced.

© 2026 Journal of Trends and Challenges in Artificial Intelligence

1. INTRODUCTION

Digital advertising's shift toward programmatic buying has made real-time bidding the default transaction

mechanism for a large share of display and mobile inventory (Gharibshah & Zhu, 2021; Ou et al., 2024). The mechanics are simple to describe: a bid request is generated the instant a page loads or an app opens, and it

¹ Corresponding author: Sujan Kumar Reddy Challa
Email: sujan.challa10@gmail.com

moves from publisher to supply-side platform to ad exchange, where competing demand-side platforms submit bids within a window that is typically well under 200 milliseconds (Yuan et al., 2013). That window is not a soft target; it is a hard constraint that rules out most conventional forms of offline or batch review for decisions that need to happen in the moment.

Scale compounds the problem. A single mid-sized exchange can handle billions of requests a day, and a meaningful portion of that volume may not correspond to genuine human interest at all. Bots, spoofed app or site inventory, malformed request fields, sudden publisher-level traffic shifts, and coordinated manipulation each produce their own signature, though none of these signatures announces itself cleanly (Sun et al., 2021). An unusual-looking request may simply belong to a smaller or newer publisher whose traffic is naturally irregular, while a request that passes every superficial check may still be part of a well-disguised fraud operation. This is, in a fairly direct sense, why fixed thresholds tend not to hold up in practice.

Rule-based systems persist in ad-tech operations mainly because they are cheap and easy to reason about, not because they perform particularly well. Their central weakness is that they are static. A bid-price threshold calibrated for one publisher's typical distribution may be needlessly strict for a smaller site with naturally sparse traffic, and far too permissive for a larger one whose patterns shift by hour, device mix, or campaign season. Worse, static rules encode an assumption about what abnormal traffic looks like at a single moment, an assumption that erodes as both fraud tactics and legitimate behavior continue to change around them. Operationally, this also creates a maintenance burden: someone has to notice that a rule has drifted out of calibration, diagnose why, and manually adjust it, a cycle that tends to lag well behind the traffic patterns it is meant to police.

What seems to be missing is detection logic that adapts to context: logic that has some notion of what "normal" looks like for a given publisher, device class, region, or campaign, and that updates that notion as conditions shift. The anomaly-detection literature offers plenty of statistical and machine-learning tools for this kind of problem, but much of the applied work on advertising fraud has focused on reviewing traffic logs after the fact rather than scoring individual bid requests as they arrive (Hemmatpour et al., 2024; Sun et al., 2021). That leaves a gap between what detection research shows is possible offline and what RTB decisioning actually needs: low-latency, context-aware, continuously adaptive scoring applied to each bid request individually.

This paper proposes a framework aimed at that gap. Its contributions can be summarized as follows.

First, a real-time anomaly detection architecture built specifically for identifying unusual bidding requests within RTB environments, designed around bid-time latency rather than retrofitted from offline audit tooling. Second, a multi-dimensional behavioral model of bid requests based on time-based, transaction-based,

publisher-based, device-based, and context-based attributes, which enables the system to differentiate between an anomalous data point and a true behavior pattern of a particular entity.

Third, a score that combines statistics, machine learning, and context into a single understandable score, able to distinguish between an anomaly and a behavior pattern that requires escalation.

Fourth, an implementation of the framework together with a structured evaluation against static rule-based, single-method statistical, and unsupervised machine-learning baselines on a reproducible synthetic RTB stream, reporting detection accuracy, per-class coverage, latency, and computational overhead.

The remainder of this paper is organized to build toward those four contributions in sequence. Section 2 surveys related work across RTB systems, advertising fraud, and anomaly detection. Section 3 formalizes the detection problem. Section 4 lays out the proposed architecture, and Section 5 covers feature engineering. Section 6 details the hybrid detection model, while Section 7 describes how the behavioral baseline adapts over time. Section 8 reports the implementation and the experimental evaluation, Section 9 discusses implications, deployment considerations, and limitations, and Section 10 concludes.

2. RELATED WORK

2.1 Real-Time Bidding and Programmatic Advertising

An RTB transaction usually takes place between a publisher and an ad exchange via a supply-side platform (SSP), which broadcasts the availability to several demand-side platforms (DSPs) (Ou et al., 2024; Yuan et al., 2013;). Each DSP evaluates the impression and returns a bid within its allotted window, after which the exchange selects a winner and notifies the publisher, an interaction that has itself been the subject of extensive optimization research aimed at directly maximizing advertiser profit rather than auction-level metrics alone (Ren et al., 2018). Bid requests are structured records, including publisher and placement identifiers, device and OS metadata, geographic signals, user-agent strings, and campaign context, which is precisely what makes them analyzable as multivariate data rather than raw log text, and also precisely what fraudulent actors attempt to manipulate or fabricate (Sun et al., 2021). Since the entire exchange happens under a strict time budget, any detection component inserted into this pipeline inherits that same constraint, and that constraint shapes nearly every design decision that follows.

2.2 Fraud and Anomaly Detection in Advertising

Several recurring categories of abnormal traffic appear throughout the ad-fraud literature: automated bot traffic, click and impression fraud intended to inflate spend,

inventory spoofing where the true source of an impression is misrepresented, bid manipulation aimed at distorting auction outcomes, sudden bursts inconsistent with a publisher's usual pattern, and synthetic traffic engineered to resemble genuine human behavior (Aljabri & Mohammad, 2023; Hemmatpour et al., 2024; Sun et al., 2021). These categories overlap in practice more often than they appear in a taxonomy: a single fraud operation might combine spoofed inventory with bot-driven requests and irregular bursts simultaneously, which helps explain why detection systems built around a single signal tend to be fragile. It also means that a detection framework aimed at RTB specifically needs to reason across several of these categories at once rather than optimizing for just one, since narrowing the scope to a single fraud pattern tends to leave the others effectively unmonitored.

2.3 Statistical Anomaly Detection

Statistical detection methods work by establishing a baseline distribution for some monitored quantity and flagging deviations beyond a chosen threshold. The Z-score method involves comparison between the observation and the mean and standard deviation of a reference window. In the moving average and exponentially weighted moving average (EWMA), the focus is on tracking the smoothed baseline, which adapts slowly with changing trends. This extends to the density and isolation methods for the multivariate case, where the Local Outlier Factor calculates the isolation of the point based on the density of its local neighborhood (Breunig et al., 2000), while Isolation Forest recursively splits the feature space and considers the points separated out as anomalies (Liu et al., 2012). Both remain popular in production settings largely because they are unsupervised and computationally efficient, properties that matter where labeled fraud data is scarce and inference has to happen quickly. Their main limitation for RTB is that most statistical methods reason about one entity's history in isolation and do not, by themselves, carry any notion of what should count as normal for a different publisher, device class, or campaign, which is precisely the contextual gap the present framework tries to close.

2.4 Machine-Learning-Based Detection

Machine learning algorithms break free from the rigid assumptions of statistical models. One-Class Support Vector Machines learn a boundary separating normal observations from the origin in a transformed feature space, treating anything outside that boundary as novel (Schölkopf et al., 1999). Autoencoders instead learn to reconstruct normal patterns and flag high reconstruction error as anomalous, an approach that has been applied to real-time network anomaly detection with detection rates reported in the high nineties at sub-second latency (Ayad et al., 2024). Where labeled data is available, ensemble methods such as Random Forest and XGBoost have shown strong reported performance on fraud and click-

fraud classification tasks (Aljabri & Mohammad, 2023; Hajek et al., 2023). Deep anomaly detection methods go further still, though their computational demands can sit uneasily against millisecond-scale inference budgets, which is part of why the present framework treats the machine-learning layer as one contributor among several rather than as the sole detection mechanism.

2.5 Research Gap

What emerges from this body of work is that statistical and machine-learning methods can both perform well in offline or near-offline conditions. Much of the applied advertising fraud literature, however, evaluates traffic after the fact, at the level of aggregated logs, campaigns, or devices, rather than scoring individual bid requests during the live auction itself (Hemmatpour et al., 2024; Sun et al., 2021). What RTB decision-making requires, then, is an anomaly detection process that is fast enough to be conducted during the bidding process, is mindful of the particularity of the publisher, the device, and the campaign in question, and is able to adapt continuously as traffic changes. It is this combination, more than any individual flaw in algorithms, that seems underserved in today's systems.

A related gap concerns interpretability. Several of the methods surveyed above, particularly deep autoencoders and ensemble classifiers, produce a single anomaly or fraud score with little indication of which underlying signal drove it. In an advertising context, where a false positive can mean blocking legitimate publisher revenue or advertiser spend, that opacity is a real operational cost, not merely a theoretical one. Work on trustworthy AI in RTB systems has begun to call attention to this concern directly (Tang & Yu, 2025), but relatively little of the detection literature reviewed above treats interpretability as a first-class design constraint alongside accuracy and latency. The framework proposed here treats all three as co-equal requirements from the outset.

3. PROBLEM FORMULATION

A bid request can be represented mathematically using a tuple $B_i = \{x_i, t_i, p_i, d_i, c_i, s_i\}$, where x_i is a vector that represents the raw properties of the bid request, t_i is the time-stamp of the bid request, p_i represents the context of the publisher, d_i represents the context of the device and environment, and c_i represents the context of the campaign. The anomaly score is determined with respect to the behavioral baseline as follows: $A(B_i) = f(B_i, N_i)$, where N_i is the baseline established with respect to the respective publisher, device class, campaign, and channel at time t . With the threshold τ , the requests are classified into two categories: Normal if $A(B_i) < \tau$ and Anomalous if $A(B_i) \geq \tau$.

N_i and $f(\cdot)$ are left deliberately open in this formulation, since the paper's central argument is that both should be adaptive rather than fixed. Fixing $f(\cdot)$ to a single

functional form, for instance a pure Z-score or a single unsupervised model, would reintroduce exactly the rigidity that motivates this work in the first place. Instead, Section 6 defines $f(\cdot)$ as a weighted combination of the three detection layers introduced there, and Section 7 defines how N_t is maintained and updated over time for each tracked entity.

Table 1. Notation used in the problem formulation.

Symbol	Meaning
B_i	The i -th bid request, represented as a tuple of feature groups
x_i	Raw request-level features extracted from B_i
t_i	Timestamp of request B_i
p_i	Publisher context associated with B_i
d_i	Device and environmental context associated with B_i
c_i	Campaign context associated with B_i
s_i	Supply-channel context associated with B_i
N_t	Behavioral baseline maintained for the relevant entities at time t
$A(B_i)$	Anomaly score computed for request B_i
τ	Decision threshold separating Normal from Anomalous classification
D_t	Absolute deviation of an observed quantity from its expected value at time t
Z_i	Standardized (Z-score) deviation for numeric feature i
S_i	Fused anomaly score for request B_i combining all detection layers
$w_1 \dots w_4$	Tunable fusion weights for the statistical, ML, contextual, and temporal components

The threshold τ is likewise treated as a deployment-level parameter rather than a universal constant, since the acceptable trade-off between false positives and false negatives can reasonably differ across exchanges, publishers with different revenue sensitivity, and campaign types with different fraud exposure. Table 1 collects the notation used across the remaining sections.

4. PROPOSED ARCHITECTURE

The proposed pipeline begins with the raw bid-request stream and ends with a risk classification an operator can act on. Requests first pass through an ingestion layer where they are parsed and validated, then move to a feature extraction layer where features related to time, behavior, and context are extracted (Section 5). These features are compared against a continuously maintained behavioral baseline and passed to a three-branch detection engine, statistical, machine-learning, and contextual, described in Section 6. The three branches' outputs are fused into one anomaly score, which a risk classifier sorts into normal traffic that proceeds unimpeded, suspicious traffic flagged for monitoring, and critical traffic subject to blocking or alerting. Figure 1 below shows the flow. The separation of the branches, instead of integrating all of them into a single monolithic system, is important because each of the branches can be

adjusted and changed depending on the fraud and legitimate patterns observed in production. This separation also has practical deployment consequences. Because the ingestion and feature extraction layers are decoupled from the detection branches, an operator can update or retrain the machine-learning branch, for example when a new fraud pattern is identified, without touching the statistical or contextual branches, and without redeploying the ingestion pipeline at all. Similarly, the baseline-maintenance component that feeds the contextual branch can be scaled or sharded independently of the scoring path, since baseline updates are not strictly latency-bound in the same way that the scoring of an individual bid request is. The risk classifier itself is deliberately kept thin: it consumes the fused score S_i and applies the deployment-level threshold τ , so that changes to business policy, such as how aggressively suspicious traffic should be throttled, do not require changes anywhere upstream in the detection logic.

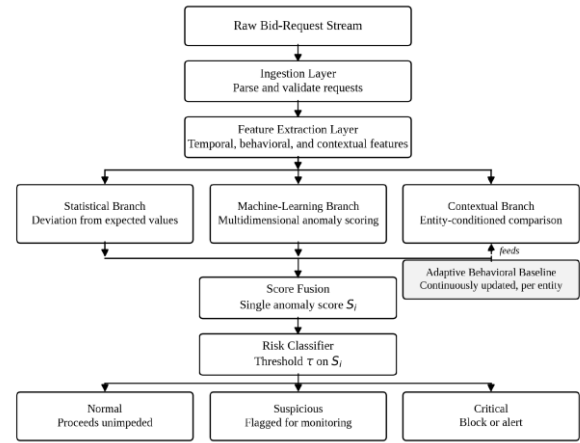


Figure 1. RTB Anomaly Detection Pipeline

Integration with an existing RTB pipeline is intended to sit alongside, rather than replace, whatever bid-processing logic an exchange or DSP already runs. The detection engine reads the same request fields already available at bid time and returns a score and, where applicable, a coarse classification before the bidding decision is finalized, so it can inform, but not necessarily automate outright, the downstream decision of whether to bid, at what price, or whether to flag the request for later audit.

5. FEATURE ENGINEERING

This is arguably where the framework's practical contribution is most visible, since the behavioral representation determines what the downstream detectors are even able to learn. Table 2 groups candidate features into five families. Temporal features include requests per second, requests per minute, inter-request gap, time-of-day, burst frequency, and spikes compared to a baseline. Bid features include bid prices, variance of bid prices,

bid-response rate, bid-to-request ratio, average bid prices, and deviation of bid prices from a baseline. Publisher features capture request volume, historical traffic distribution, domain or app behavior, and publisher-level deviation from expectation. Device and context features span device type, operating system, browser, geographic region, IP- or ASN-derived characteristics, and user-agent consistency across requests tied to the same device identifier, a signal shown to be useful for content-agnostic detection of coordinated bot traffic in other internet-scale settings (Feng et al., 2020).

A common deviation formulation ties these families together: $D_t = |X_t - \mu_t|$, where D_t measures how far an observed quantity X_t departs from its expected value μ_t , computed per entity rather than globally. This entity-conditioned framing is what lets the system tell a publisher whose traffic is unusual relative to its own history apart from one whose traffic merely differs from

other publishers', a distinction that matters more than it might first appear, since the latter is often nothing more than ordinary heterogeneity.

Constructing these features reliably raises a few practical concerns worth stating explicitly. First, not every field arrives populated on every request; device or geographic signals in particular can be missing or low-quality for a meaningful share of traffic, and the feature layer needs an explicit policy for such cases (for example, falling back to a coarser entity grouping) rather than silently treating missing data as anomalous. Second, features drawn from small-sample entities, such as a newly onboarded publisher with only a few hours of traffic history, are inherently noisier than the same features computed for an established one, which argues for confidence weighting or a minimum history requirement before a given feature is allowed to drive the anomaly score strongly.

Table 2. Candidates feature families for bid-request representation.

Feature Family	Representative Features
Temporal	Requests/second, requests/minute, inter-request interval, hour-of-day distribution, burst frequency, sudden traffic increase
Bid-related	Bid price, bid-price variance, bid response rate, bid-to-request ratio, average bid value, bid-price deviation
Publisher	Publisher request volume, historical request distribution, domain/app behavior, publisher-level deviation
Device / context	Device type, OS, browser, geographic region, IP/ASN characteristics, user-agent consistency
Behavioral deviation	$D_t = X_t - \mu_t $, computed per publisher, device class, or campaign

Third, because features are computed per entity rather than globally, the feature layer must maintain per-publisher, per-device-class, and per-campaign state efficiently enough to keep pace with the same latency budget that governs the rest of the pipeline, which favors incremental or streaming statistics over recomputation from raw history on every request.

6. PROPOSED HYBRID ANOMALY DETECTION MODEL

Rather than betting everything on one algorithm, the framework layers three complementary detectors. Layer 1 computes a standardized deviation score, $Z_i = (x_i - \mu) / \sigma$, for relevant numeric features, a cheap first pass that catches obvious deviations with minimal overhead. The second layer uses unsupervised learning, for example Isolation Forest and autoencoders, to detect multidimensional anomalies that could not be caught by a single-feature Z-score, because fraud patterns tend to emerge through abnormal combinations of seemingly normal values rather than through any one outlying value (Liu et al., 2012; Ayad et al., 2024). The third layer compares a request to baselines conditioned on the following attributes of the request: publisher, device type, region, campaign, and recently seen traffic.

The three scores combine through weighted fusion: $S_i = w_1S_{\text{stat}} + w_2S_{\text{ML}} + w_3S_{\text{context}} + w_4S_{\text{temporal}}$, subject to $\sum w_i = 1$. Treating the w_i as tunable per deployment gives an operator room to lean more on contextual signals where historical data is rich, or fall back on statistical and machine-learning signals for newly onboarded publishers with thin history. This fusion, rather than reliance on any single method, is meant to be the framework's main technical contribution: it keeps interpretability intact at the layer level, so a reviewer looking at a flagged request can see which layer, or layers, drove the score, supporting feature-level attribution instead of a single unexplained number. Table 3 summarizes the responsibilities and interpretability characteristics of each layer.

A linear weighted fusion was chosen over a stacked or learned meta-model deliberately, and that choice trades some potential accuracy for a large gain in transparency. A learned meta-model could, in principle, capture non-linear interactions between the three layers' outputs, but it would also make the final score harder to decompose back into an explanation an operator can act on, which conflicts with the auditability goal stated in Section 1. Where a deployment's risk tolerance favors accuracy over explainability, the same three-layer decomposition could still feed a learned combiner without otherwise changing the architecture, so the choice of fusion

function is treated as a configurable design point rather than a fixed part of the framework.

Weight selection itself is expected to be handled operationally rather than analytically. A reasonable starting point sets the weights to be roughly equal and then adjusts them using held-out traffic with known outcomes, such as previously confirmed fraud cases or publisher disputes, increasing the weight of whichever layer's score correlates most strongly with confirmed anomalies for a given deployment. Because the weights are entity-agnostic global parameters in this formulation while the baselines N_i themselves are entity-specific, an operator retains two separate levers: how much each layer counts overall, and what each layer considers normal for a specific publisher, device class, or campaign (Figure 2).

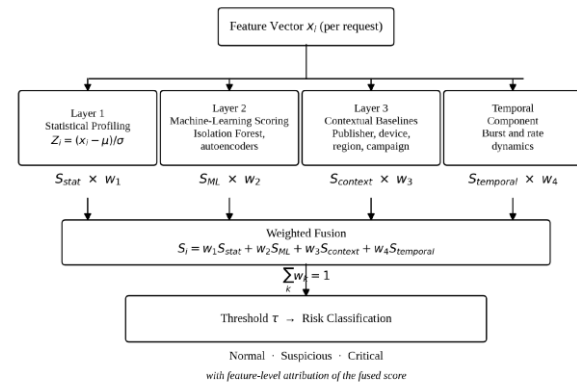


Figure 2. Three-layer hybrid anomaly scoring model with weighted score fusion.

Table 3. Responsibilities and interpretability of the three detection layers.

Layer	Method	Detects	Interpretability
1 - Statistical	Z-score deviation (Z_i)	Obvious single-feature deviations	High: direct feature-level threshold
2 - Machine-learning	Isolation Forest / autoencoder	Multidimensional, combinatorial anomalies	Moderate: feature contribution attributable post hoc
3 - Contextual	Entity-specific baseline comparison	Anomalies that are only unusual given publisher/device/campaign context	High: baseline deviation is directly reportable

Section 8.1 describes how each layer is implemented and calibrated in the evaluated prototype, including the mapping of each layer's raw output to the $[0, 1]$ range that the fusion of this section presupposes.

7. ADAPTIVE BEHAVIORAL BASELINE

A common weak point in rule-based and even many statistical systems is the implicit assumption that normal behavior can be captured once, as a fixed threshold, and left alone from there. That assumption does not hold up well in RTB. Changes in publisher traffic composition, variations in bid-price distributions due to seasonal promotions, and modifications in device or location distribution based on advertising targeting can all affect the baseline. If this baseline is static and does not change, it will eventually do one of two bad things: become a source of false positives for operators as legitimate activity diverges from the baseline, or miss true anomalies because they do not conform to an old definition of normalcy.

The framework instead treats N_i as a continuously updated, entity-specific profile, using sliding-window and decay-weighted statistics so recent observations carry more weight than older ones while enough history is retained to catch sharp, sudden deviations. This draws on work from the broader concept-drift literature in network traffic analysis, which has looked at triggering baseline retraining from detected drift rather than a fixed schedule (Bagui et al., 2025). Applied here, that means a publisher's or device class's baseline should shift

gradually as its traffic evolves, while staying sensitive enough to flag a sudden burst inconsistent with anything in that entity's recent history as anomalous, rather than absorbing it into the baseline too quickly.

Choosing the right granularity of entity for baseline maintenance involves a direct trade-off. A baseline maintained per individual publisher is the most specific and, in principle, the most sensitive to that publisher's own drift, but it is also the slowest to accumulate enough history for a small or newly onboarded publisher, which reintroduces a version of the cold-start problem discussed in Section 9. A baseline maintained at a coarser level, such as a device class or traffic segment shared across several similar publishers, converges faster but risks smoothing over exactly the kind of publisher-specific irregularity the contextual layer is meant to catch. The framework does not resolve this trade-off with a single fixed rule; instead, it proposes falling back to a coarser, faster-converging baseline whenever an entity's own history is too thin to support a stable entity-specific one, and narrowing to the finer-grained baseline once sufficient history has accumulated.

The decay weighting itself can be implemented with a simple exponential scheme, where an observation from k time steps in the past contributes with weight proportional to λ^k for a decay factor $0 < \lambda < 1$ chosen per entity type. A λ close to 1 retains a long memory and produces a slowly shifting baseline well suited to publishers with stable, low-variance traffic; a smaller λ forgets older observations faster and suits publishers whose legitimate traffic pattern itself changes quickly, such as those running frequent promotional campaigns. This parameter, like the fusion weights in Section 6, is

treated as a deployment-tunable setting rather than a value fixed by the framework itself.

8. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

The original submission proposed the evaluation protocol without reporting results. This section now reports the completed study: the framework of Sections 4–7 was implemented, and the protocol was executed against the three baseline families named in Section 1. All numbers below are measured outputs of that implementation on the data described in Section 8.2.

8.1 Implementation

The prototype is a single-process Python implementation (Python 3.12; NumPy 2.5.4; scikit-learn 1.9.1) of the pipeline in Figure 1. An ingestion stage maintains the stream aggregates of Table 2 incrementally (per-publisher and per-device window counts, inter-arrival statistics, IP-prefix entropy and ASN concentration over five-minute blocks), so that scoring a request works from its extracted feature vector. Fifteen numeric features are used, drawn from the five families of Table 2; hour-of-day enters only through the rate features, because the warm-up period covers a single hour and no meaningful standardization can be formed for it directly.

Layer 1 standardizes each numeric feature on the warm-up traffic and takes the largest absolute deviation, $\max|Z|$, mapped to $[0, 1]$ by a linear ramp that is 0 at $|Z| = 2.5$ and 1 at $|Z| = 8$. The geo/IP and user-agent consistency indicators are deliberately not standardized here: they are categorical context signals, and standardizing a rare Bernoulli indicator would turn a single data-quality artifact (for example a VPN exit node) into a roughly ten-sigma event; they are scored by Layer 3 and by the rule baseline instead. Layer 2 is an Isolation Forest (200 trees, subsample 4,096) fitted on the warm-up traffic; its raw score is mapped to $[0, 1]$ using the 50th and 99.9th percentiles of out-of-fold warm-up scores (two half-sample models scoring the held-out half), because in-sample scores of the final model are optimistically compressed. Layer 3 maintains, per publisher $\times$ device-class entity, the median and interquartile range of the log bid floor over a trailing 60-minute window refreshed every 15 minutes (the sliding-window baseline of Section 7), with a fallback chain from entity (at least 30 observations) to publisher to the global window statistics. The same window provides a rate model per publisher: the mean and the empirical standard deviation of per-minute request counts, the latter correcting for the overdispersion of real count traffic relative to a Poisson model. A geo/IP or user-agent consistency violation contributes 0.85 to the layer score, and the layer output is the maximum of its components. The temporal component combines an inter-arrival regularity score (a device whose hourly inter-arrival coefficient of variation

falls below 0.35 is treated as machine-paced) with a burst score that ramps as the publisher's current per-minute count moves from two to five times its window expectation.

The fusion weights were fixed a priori at the starting point suggested in Section 6, $w = (0.25, 0.30, 0.30, 0.15)$ for (statistical, machine-learning, contextual, temporal), and were not tuned on the evaluation data. The decision thresholds, by contrast, are deployment parameters: the framework threshold ($\tau = 0.607$), the statistical baseline threshold ($\max|Z| = 12.412$) and the Isolation Forest baseline threshold were each selected on the validation slice (Section 8.2) at the value maximizing F1, and then frozen for the test slice. The rule baseline uses fixed operational rules, set a priori and never tuned: flag a request if its device issued at least 300 requests in the trailing hour, or its publisher at least 400 requests in the current minute, or both consistency indicators are violated, or the bid floor exceeds the warm-up 99.9th percentile, or the bid-to-floor price ratio exceeds 8.

8.2 Synthetic RTB Stream and Experimental Protocol

No labeled production RTB bid stream is publicly available — exchange logs are proprietary and carry the privacy constraints discussed in Section 9.1 — so the evaluation uses a documented synthetic generator, in line with the protocol's requirement of representative RTB data with known ground truth. The generator produces a six-hour stream (08:00–14:00, with a smooth midday-peaking diurnal density) over 120 publishers whose popularity and bid-floor distributions are log-normal, and 60,000 devices, each with a home publisher set, a device class, a home geography and ASN, and an activity level.

Table 4. Evaluation metrics and their relevance to RTB deployment

Metric	Definition	Relevance to RTB
Precision	Proportion of flagged requests that are truly anomalous	Controls the cost of unnecessary blocking/monitoring
Recall	Proportion of true anomalies that are correctly flagged	Controls exposure to undetected fraud
F1-score	Harmonic mean of precision and recall	Balanced summary when both error types carry cost
False-positive rate	Proportion of normal requests incorrectly flagged	Directly affects advertiser/publisher trust
Detection latency	Time to compute $A(B_i)$ for a single request	Must fit within the bid-time decision window
Computational overhead	Compute/memory cost per request at exchange scale	Determines deployment feasibility

Request fields are modeled on OpenRTB: bid floor and clearing price, placement size, publisher and placement identifiers, device and OS class, geography, IP prefix and

ASN, and consistency indicators between the IP-derived geography and the declared geography and between the user agent and the device class.

Anomalies are injected only after the warm-up hour, in four classes drawn from Section 2. Bot-generated bursts: 18 episodes of 250–560 requests over 2–8 minutes from one to four devices behind a single IP prefix with near-constant inter-arrival spacing. Spoofed inventory: 1,800 requests carrying premium publisher identifiers but bid floors drawn from a low-tier distribution, with geo/IP and user-agent mismatch probabilities of 0.45 and 0.40. Invalid traffic: 1,800 requests with bid floors scaled by $\times 10$ –50 or $\times 0.001$ –0.01, extreme placement areas, and a 60-device pool reused across geographies. Collusion rings: 10 rings of eight publishers sharing one ASN, each contributing 35–70 extra requests in a 4–6 minute

window from a shared 60-device pool, with bid prices pegged at floor $\times 1.05$ (standard deviation 0.4%).

The stream is split chronologically: the first hour (32,083 requests, normal only) fits the baselines and the Isolation Forest; the next 30 minutes (21,236 requests) calibrate thresholds; the remaining 4.5 hours form the test slice (196,691 requests, 13,014 anomalies, an anomaly rate of 6.6%). Table 4 restates the evaluation metrics; results are reported in aggregate (Section 8.3), per anomaly class (Section 8.4), and by ablation (Section 8.5), as the protocol requires. The generator seed (20261015) and the model seed (42) are fixed, and every generator parameter is stated above, so the study can be reproduced exactly; the generator and scoring code are available from the author on request (Table 5).

Table 5. Composition of the synthetic RTB stream by split

Split	Requests	Anomalies	Anomaly rate
Warm-up (fitting)	32,083	0	0.0%
Validation (threshold calibration)	21,236	1,996	9.4%
Test	196,691	13,014	6.6%
Total	250,010	15,010	6.0%

8.3 Detection Performance

The static rule baseline behaves as Section 1 predicts: it is precise (precision 0.905) but misses most anomalies

(recall 0.169), because fixed global thresholds cannot track 120 publishers with different scales (Table 6).

Table 6. Detection performance on the test slice (196,691 requests; thresholds calibrated on validation)

Method	Precision	Recall	F1-score	False-positive rate
Static rule-based	0.905	0.169	0.285	0.0013
Statistical only (max Z)	0.824	0.812	0.818	0.0123
Isolation Forest only	0.924	0.816	0.866	0.0048
Proposed framework (fusion)	0.895	0.782	0.835	0.0065

The single-method statistical detector reaches F1 0.818 but at the highest false-positive rate of the four methods (0.0123, roughly 12,293 false alarms per million requests), which is precisely the operational cost the framework is designed to control. The Isolation Forest baseline is the strongest single method (F1 0.866). The fused framework achieves precision 0.895, recall 0.782 and F1 0.835 at a false-positive rate of 0.0065: slightly below the Isolation Forest on aggregate F1, with a lower false-positive rate than the statistical baseline and, as the next section shows, a materially different per-class profile that the aggregate figure conceals.

8.4 Per-Class Analysis

Bot bursts and invalid traffic are detected almost completely by the framework at the critical threshold (recall 0.999 and 0.991), and collusion rings moderately (0.669), carried mainly by the ASN-concentration and shared-device signals in the statistical and machine-learning layers.

Table 7. Recall by anomaly class on the test slice. The last column uses the framework's suspicious-tier threshold (Section 8.4) instead of the critical threshold

Anomaly class	n	Rules	Statistical	Isolation Forest	Framework (critical)	Framework (suspicious)
Bot-generated bursts	6,107	0.162	0.983	0.968	0.999	1.000
Spoofed inventory	1,618	0.191	0.012	0.004	0.015	0.422
Invalid traffic	1,614	0.556	0.788	0.848	0.991	1.000
Collusion rings	3,675	0.000	0.891	0.906	0.669	0.999

Spoofed inventory is the hardest class for every method: the best baseline recall at its own operating point is 0.191 (the rule baseline's consistency checks), and the global statistical and machine-learning detectors are effectively blind to it (0.012 and 0.004), because a spoofed request is individually typical in the global feature space and is anomalous only against its claimed publisher's baseline — the exact gap the contextual layer exists to close. At the critical threshold, however, the framework also surfaces only 0.015 of spoofed requests: evidence carried by a single layer contributes at most its fusion weight to the score, so a purely contextual anomaly cannot reach a threshold calibrated for multi-layer anomalies. This is where the three-tier response of Section 4 matters operationally. With the suspicious-tier threshold sized to a 5% false-positive review budget on validation, the

framework routes 42.2% of spoofed-inventory requests to monitoring (and 99.9% of collusion), at an overall suspicious-tier recall of 0.928 and precision 0.583: context-only anomalies are escalated for review rather than silently passed or hard-blocked on single-layer evidence.

8.5 Ablation and Attribution

No single layer matches the machine-learning layer's aggregate F1 (0.866), and the statistical + ML pair (0.871) is the strongest pair — but both inherit the blindness to spoofed inventory shown in Table 7, since neither contains the contextual layer, which is the eakest layer in aggregate (F1 0.699) yet the only one that detects that class at all.

Table 8. Ablation on the test slice (each configuration's threshold calibrated on validation; pair weights renormalized)

Configuration	Precision	Recall	F1-score
Statistical layer only	0.748	0.868	0.804
ML layer only	0.924	0.816	0.866
Contextual layer only	0.728	0.672	0.699
Temporal component only	0.851	0.576	0.687
Statistical + ML	0.933	0.817	0.871
Statistical + contextual	0.854	0.808	0.830
Statistical + temporal	0.742	0.869	0.801
ML + contextual	0.856	0.847	0.852
ML + temporal	0.887	0.813	0.848
Contextual + temporal	0.736	0.737	0.736
Full fusion (proposed)	0.895	0.782	0.835

The full fusion's aggregate F1 (0.835) sits below the ML-only figure while buying exactly the coverage the fusion was designed for, plus attribution: averaged over the framework's true positives, the four layers contribute 27.4% (statistical), 32.6% (ML), 27.7% (contextual) and 12.3% (temporal) of the fused score, so no single layer dominates the average detection, and each flagged request can be decomposed into the layer contributions an operator needs for review, as Section 6 requires (Table 8).

8.6 Latency and Computational Overhead

Latency is measured for the scoring stage given a request's feature vector (the stream aggregates are maintained incrementally at ingestion, Section 4). The rule and statistical scorers cost microseconds.

Table 9. Per-request scoring latency (single-request calls, 3,000 sampled test requests) and batch-scoring overhead on the full test slice

Method	Mean (ms)	p50 (ms)	p95 (ms)	p99 (ms)	CPU s / 100k req	Peak memory (MB)
Static rule-based	0.002	0.002	0.002	0.003	0.002	2.2
Statistical only	0.003	0.003	0.003	0.003	0.040	37.8
Isolation Forest only	15.839	14.686	20.896	35.350	1.850	36.3
Proposed framework	16.595	14.955	23.011	46.962	5.493	37.8

The Isolation Forest dominates the framework's cost: mean scoring latency is 16.595 ms per request for the framework against 15.839 ms for the forest alone, and the framework's p99 of 46.962 ms remains well inside the sub-200 ms bid-time window cited in Section 1 — although the margin argues for the optimized or compiled inference path a production deployment would use rather than a Python prototype (Table 9). Batch scoring of the

full test slice costs 5.493 CPU-seconds per 100,000 requests for the framework (18,206 requests per second in a single process), with a peak trace memory of 37.8 MB and an Isolation Forest model of 13.4 MB. The periodic 15-minute baseline refresh is a batch job off the scoring path (Section 7) and is not included in the per-request figures.

8.7 Discussion of Results

Three findings stand out. First, the layered design works as intended for the anomaly classes whose signatures span several feature families: bot bursts and invalid traffic are caught nearly completely, at a false-positive rate roughly half that of the statistical baseline. Second, aggregate F1 alone would mislead: the strongest single method (Isolation Forest) is blind to the context-dependent class, and the fusion's contribution is coverage and attributability rather than a higher headline score on this stream. Third, the single global threshold is the framework's weakest point as evaluated: it prices single-layer evidence at its fusion weight, which the tiered response mitigates but does not eliminate; per-class or per-entity thresholds are a natural refinement. The absolute figures should be read as relative evidence under controlled conditions: the synthetic stream's classes are parameterized instances of the Section 2 categories, not captures of live adversarial behavior, and Section 9 discusses what this bounds.

9. DISCUSSION

The emphasis on feature-level attribution reflects a fairly practical concern: ad-tech operators tend to be wary of acting on opaque model outputs, especially when a blocking decision touches publisher revenue or advertiser spend directly, a concern echoed in recent survey work calling for greater transparency and trustworthiness in AI-driven RTB systems (Tang & Yu, 2025). Keeping visibility into which detection layer, and which underlying features, drove a given score supports human review and after-the-fact auditing in a way a single end-to-end model typically cannot.

A few limitations are worth stating plainly. The framework depends on having enough historical data to build meaningful entity-specific baselines, which creates a cold-start problem for newly onboarded publishers or low-volume campaigns. Adversarial adaptation is also a concern shared by nearly every anomaly detection system: a sophisticated actor may probe the detection thresholds and adjust behavior to stay under them, which suggests the framework's weights and baselines should be periodically reviewed rather than treated as permanent. And latency constrains which machine-learning components can actually participate in the synchronous pipeline at all; a more expressive model might be a better fit for asynchronous baselining than for synchronous scoring constrained by a strict latency budget.

Nevertheless, even taking all these constraints into account, the layered architecture provides a viable path forward. Statistical and contextual layers could be easily accommodated within the bid-time latency window, whereas the machine-learning layer could be optimized for a proper trade-off between accuracy and latency, or, if required, applied only to a subset of traffic.

9.1 Deployment and Data-Governance Considerations

A framework of this kind necessarily consumes device, IP-address, and geographic signals as part of its feature representation, which places it squarely within the scope of data-protection regimes such as the EU's General Data Protection Regulation and comparable regional privacy frameworks, depending on where an exchange, publisher, or advertiser operates. Two practical implications follow. First, whatever identifiers the feature-extraction layer relies on for entity tracking, such as device or user-agent fingerprints, should be handled under the same data-minimization and retention-limitation principles that already govern other parts of the advertising stack, rather than being treated as exempt because they feed a fraud-detection system rather than an advertising-targeting one. Second, because the baseline-maintenance component in Section 7 retains rolling history per entity, that stored history itself becomes data subject to the same governance obligations, and any production implementation would need a defined retention and deletion policy for it, not only for the raw bid stream.

A related deployment concern is how the risk classifier's output should be acted upon. Because the framework produces a graded score rather than a binary verdict, an operator has room to route suspicious traffic to a lighter-weight response, such as additional monitoring or delayed settlement, rather than outright blocking, reserving hard blocking for the highest-confidence critical tier. This graded response is consistent with the interpretability goal discussed above, since a graded score gives an operator more room to calibrate the cost of a false positive against the cost of a missed fraud case than a single threshold would.

The evaluation of Section 8 also bounds what can currently be claimed. The synthetic stream's anomaly classes are parameterized instances of the fraud categories of Section 2 rather than captures of live adversarial behavior, so the measured detection rates are evidence about the framework's relative behavior under controlled conditions, not production performance guarantees; generator parameters that favor or disfavor a given layer would shift the absolute figures, which is why the protocol reports per-class and ablation results rather than a single headline number. The per-class analysis also shows that a single global threshold under-serves anomalies that only the contextual layer detects, because linear fusion prices single-layer evidence at its weight; the tiered response of Section 4 is the framework's intended mitigation, and per-entity or per-class thresholding is a natural refinement. Validation on production RTB traffic, with its heavier class imbalance and adversarial adaptation, remains necessary for future work.

10. CONCLUSION

In this paper, we have discussed a contextual-adaptive framework for detecting abnormal bid requests within real-time bidding platforms. This method combines statistical, machine learning, and contextual detection in one score with the help of the behavioral baseline of entities, which is updated dynamically rather than being static with pre-defined thresholds. Scoring individual bid requests within the latency constraints of a live auction, rather than reviewing aggregated traffic after the fact, sets this approach apart from much of the existing ad-fraud literature. The framework has been implemented and evaluated against static rule-based, single-method statistical, and unsupervised machine-learning baselines on a reproducible synthetic RTB stream: at its critical threshold it achieves an F1-score of 0.835 at a false-positive rate of 0.0065, with near-complete detection of

bot bursts and invalid traffic, and its monitoring tier surfaces context-dependent spoofed inventory that the baselines miss, at a mean scoring latency of 16.595 ms per request.

Future work includes empirical validation on production RTB logs from an operating exchange, assessment of robustness against adversarial adaptation, and exploration of federated approaches to baseline learning, which could let publishers or exchanges benefit from shared behavioral patterns without exposing raw traffic across organizational boundaries. A further direction worth pursuing is a systematic sensitivity analysis of the fusion weights w_i and the decay factor λ introduced in Sections 6 and 7, since the framework's practical performance in any given deployment is likely to depend as much on how well these parameters are tuned to local traffic conditions as on the underlying detection algorithms themselves.

References

- Aljabri, M., & Mohammad, R. M. A. (2023). Click fraud detection for online advertising using machine learning. *Egyptian Informatics Journal*, 24(2), 341-350. <https://www.sciencedirect.com/science/article/pii/S1110866523000294>
- Ayad, A. G., El-Gayar, M. M., Hikal, N. A., & Sakr, N. A. (2024). Efficient real-time anomaly detection in IoT networks using one-class autoencoder and deep neural network. *Electronics*, 14(1), Article 104. <https://doi.org/10.3390/electronics14010104>
- Bagui, S. S., Khan, M. P., Valmyr, C., Bagui, S. C., & Mink, D. (2025). Model retraining upon concept drift detection in network traffic big data. *Future Internet*, 17(8), Article 328. <https://doi.org/10.3390/fi17080328>
- Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: Identifying density-based local outliers. *ACM SIGMOD Record*, 29(2), 93-104. <https://doi.org/10.1145/335191.335388>
- Feng, Y., Li, J., Jiao, L., & Wu, X. (2020). Towards learning-based, content-agnostic detection of social bot traffic. *IEEE Transactions on Dependable and Secure Computing*. Advance online publication. <https://doi.org/10.1109/TDSC.2020.3047399>
- Gharibshah, Z., & Zhu, X. (2021). User response prediction in online advertising. *ACM Computing Surveys*, 54(3), 1-43. <https://doi.org/10.1145/3446662>
- Hajek, P., Abedin, M. Z., & Sivarajah, U. (2023). Fraud detection in mobile payment systems using an XGBoost-based framework. *Information Systems Frontiers*, 25(5), 1985-2003. <https://doi.org/10.1007/s10796-022-10346-6>
- Hemmatpour, M., Zheng, C., & Zilberman, N. (2024). E-commerce bot traffic: In-network impact, detection, and mitigation. In *Proceedings of the 2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)* (pp. 179-185). <https://doi.org/10.1109/ICIN60470.2024.10494459>
- Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2012). Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data*, 6(1), 1-39. <https://doi.org/10.1145/2133360.2133363>
- Ou, W., Chen, B., Dai, X., Zhang, W., Liu, W., Tang, R., & Yu, Y. (2024). A survey on bid optimization in real-time bidding display advertising. *ACM Transactions on Knowledge Discovery from Data*, 18(3), 1-31. <https://doi.org/10.1145/3628603>
- Ren, K., Zhang, W., Chang, K., Rong, Y., Yu, Y., & Wang, J. (2018). Bidding machine: Learning to bid for directly optimizing profits in display advertising. *IEEE Transactions on Knowledge and Data Engineering*, 30(4), 645-659. <https://doi.org/10.1109/TKDE.2017.2775228>
- Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., & Platt, J. C. (1999). Support vector method for novelty detection. In *Proceedings of the 13th International Conference on Neural Information Processing Systems (NeurIPS '99)* (pp. 582-588). <http://papers.neurips.cc/paper/1723-support-vector-method-for-novelty-detection.pdf>
- Sun, S., Yu, L., Zhang, X., Xue, M., Zhou, R., Zhu, H., Hao, S., & Lin, X. (2021). Understanding and detecting mobile ad fraud through the lens of invalid traffic. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)* (pp. 287-303). <https://doi.org/10.1145/3460120.3484547>

- Tang, X., & Yu, H. (2025). Towards trustworthy AI-empowered real-time bidding for online advertisement auctioning. *ACM Computing Surveys*, 57(6), 1-36. <https://doi.org/10.1145/3701741>
- Yuan, S., Wang, J., & Zhao, X. (2013). Real-time bidding for online advertising. In Proceedings of the Seventh International Workshop on Data Mining for Online Advertising (ADKDD '13) (pp. 1-8). <https://doi.org/10.1145/2501040.2501980>

Sujan Kumar Reddy Challa

Independent Researcher,
USA.

sujan.challa10@gmail.com

ORCID: 0009-0005-8705-9770
