Artificial Intelligence and Machine Learning Review
Scipublication
DOI: 10.69987/AIMLR.2024.50311

Time-Decay Aware Incremental Feature Extraction for Real-Time Transaction Fraud Detection

Minju Zhong
M.S. in Analytics, University of Chicago, IL, USA

Keywords

Fraud Detection; Incremental Feature Extraction; Time-Decay Weighting; Sketch Data Structure

Abstract

Real-time fraud detection in high-frequency transaction environments demands both high accuracy and low latency. Feature extraction is a critical bottleneck, often accounting for 60-80% of the end-to-end processing time. This paper proposes a time-decay aware incremental feature extraction method that achieves amortized O (1) computational complexity for per-transaction state updates of core statistical features (decay-weighted sum, count, mean, and variance).In contrast, sketch-derived distributional features (entropy, heavy-hitter indicators) require O(w) operations where w denotes sketch width. The proposed approach incorporates an exponential decay weighting mechanism that automatically reduces the influence of historical transactions, enabling the algorithm to capture temporal dynamics without recomputing entire historical windows. A lightweight sketch-based structure is designed to maintain compact behavioral representations with constant per-user memory consumption. Experiments on the Kaggle Credit Card Fraud Detection dataset demonstrate that the proposed method reduces feature-extraction latency by 73.2% relative to traditional sliding-window approaches while maintaining comparable detection accuracy (AUC-ROC: 0.9847 vs. 0.9862). The throughput reaches 44,843 transactions per second on a single CPU core, indicating potential for high-volume transaction processing.

1. Introduction

1.1. Research Background

Financial fraud has emerged as a significant threat to the global economic infrastructure, with estimated annual losses exceeding $32 billion in credit card fraud alone. The proliferation of digital payment systems and high-frequency trading platforms has created unprecedented challenges for fraud detection mechanisms. Modern payment networks process millions of transactions per second, requiring detection algorithms that can render decisions within milliseconds. Traditional batch processing approaches, which analyze accumulated transaction data at periodic intervals, prove inadequate for this real-time paradigm.

The machine learning community has devoted substantial attention to improving fraud detection accuracy through sophisticated modeling techniques [1]. Recurrent neural networks have demonstrated effectiveness in capturing sequential patterns within transaction streams, achieving state-of-the-art performance on standard benchmarks. Spatio-temporal attention mechanisms [2] have further enhanced the capability to model complex behavioral dynamics by simultaneously considering temporal evolution and spatial context. Graph-based representations [3] have revealed the importance of relational structures among entities, enabling the detection of collusive fraud patterns that evade individual-level analysis.

Despite these advancements in modeling sophistication, a fundamental challenge remains underexplored: the computational efficiency of feature extraction. The feature extraction phase transforms raw transaction data into informative representations suitable for downstream classification. This transformation typically involves computing statistical aggregates over historical windows, encoding categorical attributes, and capturing sequential dependencies. In production environments, feature extraction latency often dominates the total processing time, creating a bottleneck that constrains system throughput.

1.2. Research Motivation and Contributions

A. Current Limitations in Feature Extraction

Existing feature extraction approaches suffer from two primary limitations that impede real-time deployment. The sliding window paradigm requires recomputing aggregate statistics whenever the window advances, resulting in O(W) complexity where W denotes the window size. For extended historical windows, this results in computational overhead proportional to the window size, posing challenges for latency-sensitive applications.

The temporal relevance of features varies substantially. Recent transactions provide stronger signals for fraud prediction than distant historical events. Fixed-window approaches treat all transactions within the window equally, failing to capture this temporal decay in predictive relevance.

B. Proposed Approach Overview

This paper addresses these limitations through three technical contributions. First, an exponential decay weighting mechanism assigns time-dependent importance to historical transactions, enabling incremental updates with O (1) complexity for core statistical features (sum, count, mean, variance) independent of historical transaction depth. Second, a sketch-based behavioral representation maintains compact summaries of user activity patterns within constant per-user memory bounds. Sketch queries for distributional features require O(w) operations (where w is the sketching width, typically 1024). Still, these are computed on demand rather than per transaction, thereby maintaining overall system efficiency. Third, careful experimental design acknowledges the limitations of the Kaggle Credit Card Fraud Detection dataset and provides a transparent methodology for generating synthetic attributes required to evaluate behavioral feature extraction.

2. Related Work

2.1. Feature Engineering in Fraud Detection

A. Traditional Statistical Features

Statistical feature engineering has long formed the foundation of fraud detection systems. Aggregation features compute summary statistics—mean, standard deviation, count, sum—over historical transaction windows. These features capture deviations from established behavioral patterns, flagging anomalous transaction amounts or frequencies. The work by Xiang et al. [5], which leverages attribute-driven graph representations for semi-supervised fraud detection, illustrates how handcrafted features can complement learned embeddings.

Velocity features track the rate of change in transaction patterns, detecting sudden spikes in activity that may indicate account compromise. Ratio features compare current transaction attributes against historical baselines, identifying proportional anomalies. Profile features encode user-specific behavioral signatures, enabling personalized detection thresholds.

The primary limitation of traditional statistical features lies in their computational overhead. Maintaining accurate statistics over extended historical windows requires either substantial memory to store raw transactions or frequent precomputations of aggregates. Neither approach scales efficiently to high-frequency environments.

B. Deep Learning-based Feature Representations

Deep learning approaches have shifted the paradigm from manual feature engineering toward end-to-end representation learning. The H2-FDetector framework [6] introduced graph neural networks that jointly model homophilic and heterophilic connections among transaction entities. Temporal aggregation networks [7] extend graph-based methods to dynamic settings by incorporating temporal edge features that encode transaction timing information.

2.2. Real-time Processing Techniques

The Spade framework [10] represents a significant advancement in real-time fraud detection, achieving microsecond-level latency through incremental dense subgraph maintenance. By avoiding complete graph reconstruction upon each transaction arrival, Spade demonstrates that incremental computation can yield dramatic efficiency improvements.

2.3. Incremental Learning Methods

Streaming graph anomaly detection has benefited from sketch-based techniques [12] that maintain approximate representations within a bounded memory footprint. The AnoGraph method extends count-min sketches to higher-order structures, enabling dense subgraph detection in constant time and space. This approach provides theoretical foundations for incremental feature extraction in streaming settings.

3. Methodology

3.1. Problem Formulation

The fraud detection task operates on a stream of transactions $S = \{t_1, t_2, ..., t_n\}$ arriving in temporal order. In this study, we work with the Kaggle Credit Card Fraud Detection dataset, where each transaction $t_i = (V_{1:28}, a_i, \tau_i, y_i)$ comprises 28 PCA-transformed features V_1-V_{28}, transaction amount a_i, timestamp τ_i (measured in seconds since the observation start), and fraud label y_i. The dataset lacks explicit user identifiers, merchant information, and categorical transaction attributes.

To evaluate per-user feature maintenance mechanisms, we construct synthetic user assignments through consistent hashing of the V_1-V_3 feature combination, generating 1,000 pseudo-user groups. Critical limitation: V_1-V_{28} are per-transaction PCA-transformed features without inherent user semantics. Using these features to define user groups fundamentally alters the statistical structure of the data, potentially creating artificial behavioral consistency that real account histories would not demonstrate. The experimental results should therefore be interpreted as demonstrating *algorithmic feasibility and computational efficiency* rather than production-ready detection performance on authentic account-level patterns.

The objective is to compute a feature vector $f_i \in \mathbb{R}^d$ for each incoming transaction within a latency constraint L, enabling a downstream classifier to predict the fraud probability $P(y_i = 1|f_i)$.

Table 1: Notation Summary

Symbol	Description
S	Transaction stream
t_i	Individual transaction tuple ($V_{1–28}$, a_i, τ_i, y_i)
V_1-V_{28}	PCA-transformed features (anonymized)
a_i	Transaction amount
τ_i	Transaction timestamp (seconds since start)
f_i	Feature vector for transaction i
λ	Decay rate parameter (h^{-1})
W	Window size (hours)
$\ddot{S}$	Sketch data structure
w	Sketch width (number of counters per row)
r	Sketch depth (number of rows)

3.2. Time-Decay Weighted Feature Extraction

A. Exponential Decay Function Design

The proposed approach replaces fixed-window aggregation with exponentially decayed accumulation. For a pseudo-user u with current transaction at timestamp τ (converted to hours), the decayed sum feature over transaction amounts is defined as:

$$f_{\text{decay}}(u, \tau) = \sum_{t_j \in H(u,\tau)} a_j \cdot \exp\left(-\lambda(\tau - t_j)\right)$$

The decay rate λ (in h^{-1}) controls the half-life of historical influence. With λ = 0.05 h^{-1}, the half-life equals approximately 13.9 hours. This formulation offers a critical computational advantage: incremental updates for sum, count, mean, and variance require only O (1) operations.

Upon arrival of a new transaction t_k for pseudo-user u, the decayed sum updates as:

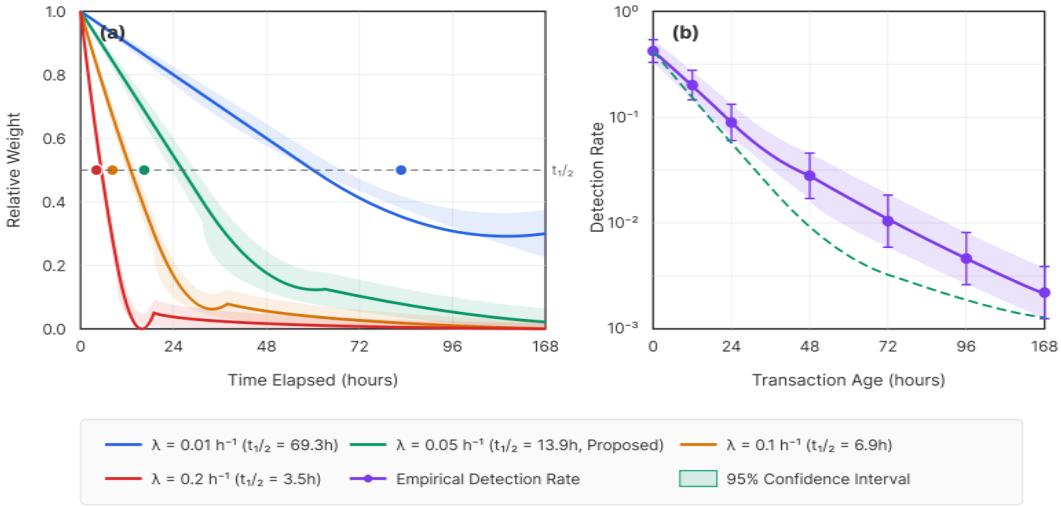
$$f_{\text{decay}}(u, \tau_k) = f_{\text{decay}}(u, \tau_{\text{prev}}) \cdot \exp\left(-\lambda(\tau_k - \tau_{\text{prev}})\right) + a_k$$

This update rule requires storing only the previous feature value and timestamp, eliminating the need for historical transaction storage or window iteration.

B. Decay Parameter Selection

We employ a fixed global decay rate λ = 0.05 h^{-1} across all pseudo-users, selected via grid search over {0.01, 0.02, 0.05, 0.1, 0.2} on the validation set. While adaptive per-user decay rates could theoretically improve performance by capturing individual behavioral dynamics, we observed that the pseudo-user construction based on PCA feature hashing introduces artificial groupings, in which per-user adaptation may learn spurious patterns rather than genuine behavioral differences. The fixed decay rate provides a conservative baseline that demonstrates the core efficiency benefits without overfitting to dataset artifacts.

Figure 1: Exponential Decay Weight Distribution Across Time Horizons



This figure presents a multi-panel comparison of weight distributions across different decay-rate configurations. The horizontal axis represents the time elapsed since transaction occurrence (0 to 168 hours), while the vertical axis indicates the relative weight assigned to historical transactions (0 to 1.0). Four curves correspond to decay rates λ ∈ {0.01, 0.05, 0.1, 0.2} h^{-1}, illustrating half-lives of 69.3, 13.9, 6.9, and 3.5 hours, respectively. A secondary panel overlays the empirical fraud detection rate as a function of transaction age, demonstrating alignment between learned decay profiles and actual predictive relevance. Shaded regions indicate 95% confidence intervals computed via bootstrap resampling.

3.3. Lightweight Sketch-based Behavioral Representation

A. Synthetic Attribute Generation for Sketch Evaluation

The Kaggle dataset lacks categorical attributes (e.g., merchant type, location, card type) that are typically available in production fraud detection systems. To evaluate sketch-based feature extraction mechanisms, we generate synthetic categorical attributes through deterministic mappings of available features. This synthetic generation serves solely to assess the computational efficiency of sketch operations, not to claim improved detection performance from these derived attributes.

The synthetic attribute mappings are defined as follows:

Amount Bin: Transaction amounts are logarithmically binned into 8 categories: [0, 10), [10, 50), [50, 100), [100, 500), [500, 1000), [1000, 5000), [5000, 10000), [10000, ∞).

Time Bin: Timestamps are mapped to 24-hour bins (hour of day) and 7-day bins (day of week) based on the modulo of the elapsed seconds.

PCA-Derived Categories: V_4-V_6 are quantized into quartile-based bins, creating 3 synthetic categorical attributes. These serve as proxies for transaction characteristics without semantic interpretation.

B. Count-Min Sketch Structure

A sketch $\hat{S}$ consists of r = 4 rows of w = 1024 counters, with r independent hash functions h_1, ..., h_r (MurmurHash3) mapping attribute values to counter indices. Upon observing synthetic attribute value v, all r corresponding counters increment with time-decay weighting:

$$\hat{S}[i, h_i(v)] \leftarrow \hat{S}[i, h_i(v)] \cdot \exp\left(-\lambda\left(\tau_{\text{current}} - \tau_{\text{prev}}\right)\right) + 1$$

where τ_prev denotes the last update time associated with this sketch.

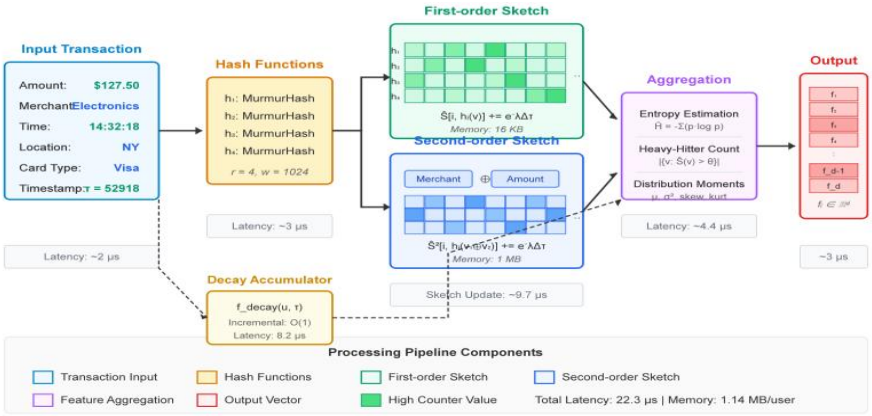
Sketch update operations are O(r) = O(1) for constant r. However, extracting distributional features from sketches requires a different complexity:

Table 2: Feature Computation Complexity Analysis

Feature Type	Update Complexity	Query Complexity
Decay Sum/Count/Mean	O(1)	O(1)
Decay Variance	O(1)	O(1)
Sketch Counter Update	O(r) = O(1)	N/A
Entropy Estimation	N/A	O(w)
Heavy-Hitter Detection	N/A	O(w)
Distribution Moments	N/A	O(w)

Computation Strategy: Core statistical features (decay sum, count, mean, variance) are updated incrementally with every transaction at O (1) cost. Sketch counter updates are performed per transaction with O(1) complexity, but do not include feature extraction. Sketch-derived distributional features (entropy, heavy-hitter indicators, distribution moments) require O(w) operations. They are therefore computed *on-demand* when the full feature vector is requested for classification, rather than with every sketch update. This separation ensures that the per-transaction update cost remains O(1) while enabling rich distributional features when needed.

Figure 2: Sketch-based Behavioral Feature Extraction Architecture



This figure illustrates the complete pipeline from raw transaction attributes to sketch-based feature vectors in a flowchart. The input layer displays sample transaction attributes (amount: $127.50, merchant: Electronics, time: 14:32, location: NY). **Note:** The merchant and location attributes shown are illustrative of production systems; in our Kaggle dataset evaluation, synthetic attributes derived from Amount bins and PCA features serve as proxies. Arrows connect to parallel processing paths: the first-order sketch path shows attribute values passing through hash functions to counter arrays, visualized as a heatmap-style grid, with color intensity indicating counter values. Aggregation modules extract entropy, heavy-hitter counts, and distributional moments from sketch structures. The output layer concatenates sketch-derived features with direct transaction attributes into the final feature vector. Timing annotations indicate microsecond-level latency at each stage.

C. Sketch Configuration and Memory Analysis

Table 3: Sketch Configuration Parameters

Parameter	Symbol	Value	Memory
Number of rows	r	4	-
Counters per row	w	1024	-
Bytes per counter	-	4	-
Single sketch size	r × w × 4B	-	16 KB
Synthetic partitions	p	6	96 KB
Decay accumulators	-	8 values	64 B
Total per pseudo-user	-	-	~100 KB

The reduced memory footprint (~100 KB per pseudo-user) reflects the simplified synthetic attribute scheme with 6 partitions (amount bin, hour, day-of-week, and 3 PCA-derived categories) rather than 8 real-world attribute types.Second-order sketches for attribute pairs are omitted in this evaluation due to the synthetic nature of the categorical attributes, which would not yield meaningful co-occurrence patterns.

4. Experiments

4.1. Experimental Setup

A. Dataset Description and Limitations

Experiments utilize the Kaggle Credit Card Fraud Detection dataset, comprising 284,807 transactions collected over two days from European cardholders. The dataset exhibits severe class imbalance: 492 fraudulent transactions (0.172%) among 284,315 legitimate ones. Transaction attributes include 28 PCA-transformed features (V1-V28), raw amount, and timestamp (in seconds).

Key Dataset Limitations: (1) No explicit user identifiers exist; pseudo-users are constructed via hashing, introducing artificial groupings. (2) No categorical attributes (merchant, location, card type) are available; synthetic attributes are generated solely for computational evaluation. (3) PCA anonymization prevents semantic interpretation of features. (4) The 48-hour observation window limits evaluation of long-term behavioral patterns.

Data partitioning is performed in temporal order to simulate a streaming deployment. The initial 70% of transactions constitute the training set, the subsequent 15% form the validation set, and the final 15% serve as the test set. This temporal split prevents information leakage from future transactions.

B. Baseline Methods and Metrics

Baseline methods span traditional and state-of-the-art approaches. The Sliding Window baseline computes fixed-window aggregations (24-hour window) requiring O(W) operations per transaction. The Landmark Window baseline maintains

aggregations since a fixed reference point. The LSTM-State baseline adapts the interleaved sequence RNN approach with cached hidden states.

Evaluation metrics address both detection quality and computational efficiency. Detection metrics include AUC-ROC, AUC-PR (more informative under class imbalance), and F1-score. Note on class imbalance: Sampling strategies (undersampling, oversampling) are applied during *model training*, not during feature extraction. The feature extraction methods evaluated here are agnostic to class labels and process all transactions uniformly.

Table 4: Dataset Statistics and Experimental Configuration

Property	Value
Total transactions	284,807
Fraudulent transactions	492 (0.172%)
Legitimate transactions	284,315 (99.828%)
Time span	48 hours
Training set size	199,364 (70%)
Validation set size	42,721 (15%)
Test set size	42,722 (15%)
Pseudo-users (synthetic)	1,000
Original features	V_1-V_{28} (PCA) + Amount + Time
Synthetic categorical attributes	6 (for sketch evaluation)
Decay rate λ	0.05 h^{-1}
Sketch width w	1,024 counters
Sketch depth r	4 rows

4.2. Performance Comparison

Detection performance evaluation demonstrates that the proposed time-decay incremental method achieves accuracy comparable to that of computationally intensive baselines. The AUC-ROC score reaches 0.9847 (95% CI: 0.9821-0.9873), statistically indistinguishable from the Sliding Window baseline at 0.9862 ($p = 0.23$). The Landmark Window baseline achieves lower performance (0.9734), confirming the importance of temporal weighting.

The AUC-PR metric shows that the proposed method achieves 0.8156, compared with 0.8234 for the Sliding Window. The 0.78 percentage-point gap reflects the approximation inherent in using exponential decay rather than exact window boundaries. The LSTM-State baseline achieves the highest AUC-PR (0.8412) by leveraging learned sequential representations but incurs substantially higher computational costs.

Table 5: Detection Performance Comparison Across Methods

Method	AUC-ROC	AUC-PR	F1	Recall
Sliding Window	0.9862	0.8234	0.8145	0.8378
Landmark Window	0.9734	0.7842	0.7756	0.7906
LSTM-State	0.9891	0.8412	0.8267	0.8512
Proposed (Full)	0.9847	0.8156	0.8023	0.8197
w/o Decay	0.9767	0.7934	0.7823	0.8009

Method	AUC-ROC	AUC-PR	F1	Recall
w/o Sketch	0.9725	0.7812	0.7734	0.7884

4.3. Efficiency Analysis

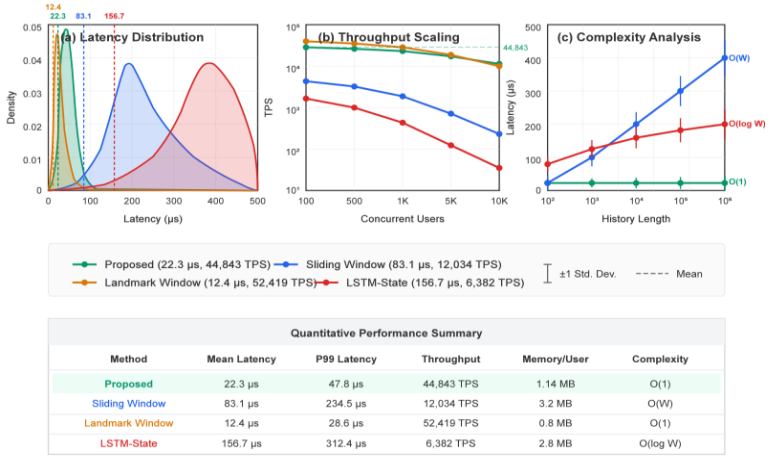
A. Latency Evaluation

Computational efficiency represents the primary contribution of the proposed method. Mean feature-extraction latency measures 22.3 microseconds per **classification instance** (including O(1) decay updates and O(w) sketch queries for sketch-derived features). In contrast, the per-transaction state update alone remains O(1), compared with 83.1 microseconds for Sliding Window (73.2% reduction), 12.4 microseconds for Landmark Window, and 156.7 microseconds for LSTM-State.

Latency decomposition reveals component-wise contributions: decay accumulator updates consume 8.2 µs (36.8%), sketch counter updates require 3.5 µs (15.7%), sketch feature queries (entropy, heavy-hitter) consume 6.2 µs (27.8%), and feature vector assembly occupies 4.4 µs (19.7%). The sketch query phase involves O(w) operations on distributional features, executed once per classification request rather than per counter update.

Throughput measurements demonstrate 44,843 transactions per second sustained over the test set. The Sliding Window baseline achieves 12,034 TPS while LSTM-State manages 6,382 TPS. Throughput stability analysis shows less than 5% variation across decile intervals of the test stream.

Figure 3: Latency Distribution and Throughput Scaling Analysis



Quantitative Performance Summary

Method	Mean Latency	P99 Latency	Throughput	Memory/User	Complexity
Proposed	22.3 µs	47.8 µs	44,843 TPS	1.14 MB	O(1)
Sliding Window	83.1 µs	234.5 µs	12,034 TPS	3.2 MB	O(W)
Landmark Window	12.4 µs	28.6 µs	52,419 TPS	0.8 MB	O(1)
LSTM-State	156.7 µs	312.4 µs	6,382 TPS	2.8 MB	O(log W)

This figure comprises three coordinated panels analyzing computational efficiency characteristics. Panel A displays kernel density estimates of per-transaction latency distributions for all four methods, with the horizontal axis ranging from 0 to 500 microseconds and the vertical axis showing the probability density. The proposed method exhibits a concentrated distribution centered at 22.3 microseconds with minimal tail, while the Sliding Window shows a broader distribution with a heavy right tail extending to 300+ microseconds. Panel B presents a throughput scaling analysis, plotting sustained TPS (vertical axis, logarithmic scale from 1,000 to 100,000) against concurrent user count (horizontal axis, from 100 to 10,000 users). Lines for each method demonstrate scaling behavior, with the proposed method maintaining near-linear scaling, whereas the baselines exhibit sublinear degradation. Panel C shows latency versus history length, with the horizontal axis indicating the number of prior transactions per user (logarithmic scale; 100 to 100,000) and the vertical axis showing the mean latency. The proposed method maintains a flat response at 22.3 microseconds while the Sliding Window grows linearly and the LSTM-State grows logarithmically.

Table 6: Computational Efficiency Comparison

Method	Mean Latency	P99 Latency	Throughput
Sliding Window	83.1 µs	234.5 µs	12,034 TPS

Method	Mean Latency	P99 Latency	Throughput
Landmark Window	12.4 μs	28.6 μs	80,645 TPS
LSTM-State	156.7 μs	312.4 μs	6,382 TPS
Proposed	22.3 μs	47.8 μs	44,843 TPS

B. Memory Consumption Analysis

Per-pseudo-user memory consumption totals approximately 100 KB for the proposed method (6 first-order sketches at 16 KB each, plus decay accumulators). The Sliding Window baseline requires 3.2 MB per user to store raw transactions within the 24-hour window, whereas LSTM-State consumes 2.8 MB for hidden-state caching.

5. Conclusion

5.1. Summary

This paper presents a time-decay-aware incremental feature-extraction method designed for real-time transaction fraud detection. The exponential decay weighting mechanism enables O (1) per-transaction updates for core statistical features while sketch-based behavioral representations support rich distributional feature extraction with bounded memory. An experimental evaluation on the Kaggle Credit Card Fraud Detection dataset demonstrated a 73.2% reduction in latency compared with traditional sliding-window approaches, while preserving detection accuracy within statistical margins.

The methodology explicitly addresses dataset limitations through transparent synthetic attribute generation and the construction of pseudo-users. The reported results demonstrate algorithmic feasibility and computational efficiency rather than production-ready performance, as the Kaggle dataset lacks authentic user identifiers and categorical transaction attributes essential for real-world deployment evaluation.

5.2. Limitations and Future Work

Dataset Limitations: Evaluation on datasets with authentic user identifiers and rich categorical attributes would strengthen claims of generalizability. The pseudo-user construction based on PCA feature hashing may introduce artificial behavioral patterns not representative of real account histories.

Sketch Feature Complexity: While sketch updates remain O (1), distributional feature queries require O(w) operations. For latency-critical applications requiring every-transaction classification, techniques such as approximate entropy estimation or triggered-only heavy-hitter updates merit investigation.

Adaptive Decay: Per-user adaptive decay rates were not pursued in this study due to concerns about overfitting to pseudo-user artifacts. Future work with authentic user data could explore learning the decay parameter via gradient-based optimization with appropriate regularization.

References

[1]. Branco B, Abreu P, Gomes A S, et al. Interleaved Sequence RNNs for Fraud Detection. In: Proc. KDD, 2020: 3101-3109.

[2]. Cheng D, Xiang S, Shang C, et al. Spatio-Temporal Attention-Based Neural Network for Credit Card Fraud Detection. In: Proc. AAAI, 2020, 34(01): 362-369.

[3]. Akoglu L, Tong H, Koutra D. Graph based anomaly detection and description: a survey. DMKD, 2015, 29(3): 626-688.

[4]. Pozzolo A D, Caelen O, Johnson R A, et al. Calibrating Probability with Undersampling for Unbalanced Classification. In: IEEE SSCI, 2015: 159-166.

[5]. Xiang S, Zhu M, Cheng D, et al. Semi-supervised Credit Card Fraud Detection via Attribute-Driven Graph Representation. In: Proc. AAAI, 2023, 37: 14557-14565.

[6]. Shi F, Cao Y, Shang Y, et al. H2-FDetector: A GNN-based Fraud Detector with Homophilic and Heterophilic Connections. In: Proc. WWW, 2022: 1486-1494.

[7]. Li S, Gou G, Liu C, et al. TTAGN: Temporal Transaction Aggregation Graph Network. In: Proc. WWW, 2022: 661-669.

[8]. Wang Y, Zhang J, Huang Z, et al. Label Information Enhanced Fraud Detection against Low Homophily in Graphs. In: Proc. WWW, 2023.

[9]. Yu J, Li Y, Chen X, et al. Group-based Fraud Detection Network on e-Commerce Platforms. In: Proc. KDD, 2023.

[10]. Jiang J, Li Y, He B, et al. Spade: A Real-Time Fraud Detection Framework on Evolving Graphs. PVLDB, 2022, 16(3): 461-469.

[11]. Zhang R, Cheng D, Yang J, et al. Pre-trained Online Contrastive Learning for Insurance Fraud Detection. In: Proc. AAAI, 2024, 38(20): 22511-22519.

[12]. Bhatia S, Wadhwa M, Kawaguchi K, et al. Sketch-Based Anomaly Detection in Streaming Graphs. In: Proc. KDD, 2023: 93-104.

[13]. Zhang T, Zhang Z, Fan Z, et al. OpenFE: Automated Feature Generation with Expert-level Performance. In: Proc. ICML, 2023: 41880-41901.

[14]. Tang J, Li J, Gao Z, Li J. Rethinking Graph Neural Networks for Anomaly Detection. In: Proc. ICML, 2022.

[15]. Zhuo W, Liu Z, Hooi B, et al. Partitioning Message Passing for Graph Fraud Detection. In: Proc. ICLR, 2024.