SENTINELPAY: A HYBRID AI-DRIVEN SYSTEM FOR REAL-TIME FRAUD DETECTION IN DIGITAL PAYMENT SYSTEMS

Balasubramanian N[1*], Thillai Nirmal K[2], Muraleeshwar V[3], Muhammed Danish Z[4]

[1]Department of Computer Science & Engineering Mohamed Sathak Engineering College Ramanathapuram, India*
balasubramanian_n@msec.org.in
[2]Department of Computer Science & Engineering Mohamed Sathak Engineering College Ramanathapuram, India
thillainirmal.javatechies@gmail.com
[3]Department of Computer Science & Engineering Mohamed Sathak Engineering College Ramanathapuram, India
muraleeshwarech@gmail.com
[4]Department of Computer Science & Engineering Mohamed Sathak Engineering College Ramanathapuram, India
muhammed2005danish@gmail.com

Corresponding Author

Abstract**—***Financial system fraud goes undiscovered, even when it would be most beneficial for them to do so. UPI transactions typically complete in less than two seconds, making it nearly impossible to detect any mistakes after the fact. SentinelPay circumvents this issue by classifying all transactions prior to the payment gateway approval at the authorization step. The architecture was developed using an event-driven microservices approach, and each part was designed to maintain the latency below acceptable boundaries. A few design choices shape the system. Apache Kafka takes care of transaction events to keep submission and scoring from being mixed up. There are various levels in the detection pipeline. After discovering statistical outliers and starting with the basic threshold criterion, Spring AI was used to make decisions based on the situation. There was no layer that was missing because they all depend on each other. This makes sure that the detection process works well and finds any mistakes. Redis may maintain user activity profiles, so the inference engine can acquire data in milliseconds.*

After looking at 50,000 entries in total, SentinelPay found real UPI fraud trends. We looked at the following numbers: recall (94.6%), detection accuracy (95.2%), F1-score (94.2%), and precision (93.8%). The throughput stays the same at 1,200 TPS, even when there is a lot of traffic. The average end-to-end latency is about 50 ms. This method gives results that are 13.2 points more accurate if you start with a rule. There are two other benefits: the frequency of false positives was down by 65%, and the time it took to get findings went down by 75%. You do not have to change your UPI settings to use SentinelPay.

Index Terms*—Real-time fraud detection, Apache Kafka, microservices, Spring AI, Redis, event-driven architecture, digital payments, anomaly detection, secure transaction validation*

INTRODUCTION

Detecting fraud in digital payments is an area that is constantly changing. When a payment method becomes popular, hackers can get into your account more easily, leading to increased risks of fraud and financial loss for users who may not be aware of the security measures they need to take.

UPI, mobile wallets, and purchasing online have all gotten significantly more popular in the past ten years. Because of these developments, a lot of people can simply get loans. Stopping fraud can be hard since the attack surface gets bigger [6], [13]. The company loses hundreds of billions of dollars a year because of online payment fraud, but things do not seem to be getting better. When there are more transactions, criminals pay more attention to the system.

Upon further examination, it is clear that many detection systems adhere to principles from a time when transaction contexts were less unpredictable. When fraud tendencies are less unpredictable, rule-based methods, such as velocity checks, constraints on transaction size, and geographic filters, are often sufficient [7], [10]. However, dynamic rules frequently cause an excessive number of false alerts because modern attackers are infamous for continually changing their tactics to avoid detection, which can overwhelm security teams and lead to missed genuine threats. In reality, it is difficult and expensive to regain user trust. Supervised ML models perform better than predefined rules in many cases, although most of them only work when they are processed in batches or offline. According to [13], [14], when external scoring is incorporated into a realtime authorization step, the latency becomes extremely high for procedures that require less than 100 ms to complete.

A better-coordinated strategy is therefore required. Although distributed event streaming technologies, such as Kafka, can handle a large number of simultaneous transactions, standard message queues, which are systems that manage the transmission of messages between services, cannot. It has been mentioned in [6] that behavioral AI can detect subtle signs of fraud that manual rules cannot detect. The literature rarely combines these strengths. To prioritize throughput, event streaming systems can lose some of their adaptability, and the integration of AI models outside the real-time authorization process introduces latency [1], [13].

There is a very low chance that a pre-authorization microservice that completely uses event-driven streaming, contextual AI inference, and behavioral caching with a latency of less than one millisecond exists. This is also the reason why SentinelPay was developed. Kafka monitors the progress of transactions, and a three-stage hybrid detection pipeline makes a decision before the payment is processed. The goal of this design is to keep the speed and accuracy of detection high.

II. LITERATURE REVIEW

If you look closely enough at the research, you can discover that the majority of fraud detection solutions just address a small part of the bigger issue. Limits on transaction amounts, speed limits, and blacklists of merchant categories were some of the criteria that were used by many of the initial systems [7], [10]. These approaches worked for a while, especially when con artists performed the same tricks again and over again. But when opponents got wiser, adding more rules did not work very well. There was better coverage, but there were too many false positives to deal with. Because of this, the processes cost more and needed more advanced algorithms to be more accurate.

In the last few years, supervised machine learning has made it much easier to find things. Models can find patterns that rule sets usually miss by looking at past transactions [11], [13]. On the other hand, it has proven hard to turn these well-trained models into systems that can work without any problems in real time.

When inference is performed utilizing remote model servers, round-trip delays are frequently greater than the 100 ms timeframe permitted by permission [14]. Because of these problems, many businesses choose post-authorization or batch-based scoring over real-time classification. Therefore they cannot stop fraud from happening in the first place. Some researchers are looking into an event-driven architecture as a possible answer. Acharjee et al. showed that using Kafka-based pipelines to quickly detect credit card fraud is useful [2].

Despite their high throughput, adaptive reasoning was not factored into their score, which may limit the effectiveness of the Kafka-based pipelines in accurately detecting credit card fraud in real-time scenarios. Although Mandlik et al. [1]—their UPI-centric architecture, which used device fingerprinting and user expenditure profiles, made a big difference in accuracy— their system was largely built for offline inference. Suggula and Goli [3], Venkatasubbu [4], and Gogineni [5] show that event-driven systems can still meet sub-second classification targets, even when they have a lot of work to do. But contextual inference and real-time behavioral caching were usually absent.

It is pertinent to note that contemporary methodologies, like explainable machine learning pipelines [7] and graph neural networks [8], have streamlined the comprehension of fraud networks and the algorithms employed for predictive generation. However, these approaches are not fast enough for realtime approval and frequently presume that the environment is down.

No one has yet discovered a way to integrate (i) eventdriven streaming, (ii) in-memory behavioral caching for profile retrieval times less than one millisecond, and (iii) adaptive, contextual AI inference into a single, pre-authorization microservices architecture [8], [15] during their investigation. To address this gap, the SentinelPay technique was developed.

III. PROPOSED METHODOLOGY

SentinelPay's operation is primarily based on timing. Prior to the payment gateway returning an authorized response, categorization must occur. Any approach of identifying fraud requires this. When a user initiates a transaction, the API Gateway confirms their identification and routes the request to the Transaction Service. Please be aware that this

transaction service does not give ratings to its own transactions. Instead, a separate Kafka topic keeps track of each transaction as an event. The user path only becomes responsive when submission and detection do not happen at the same time. After this, the real classification process runs in the background [3], [5].

The Fraud Detection Service will next investigate and sort through these Kafka events. At this point, feature extraction occurs. It evaluates the user's location, device information, previous actions, and transaction amount, among other factors. The min-max normalization procedure is applied to numerical fields like amount and frequency. Classification elements are encoded with one-hot encoding. To further reduce latency, Spring AI inference is conducted on the same microservice instance. According to [6], [14], this protects the system from encountering network trips, which previously added 30–60 ms of latency and perhaps exceeded real-time limits.

A closer study into behavioral context shows that it is quite important for making correct estimations. Knowing how often a customer buys something, how much they usually spend, and which gadgets they trust is really important. Retrieving each transaction from a relational database like MySQL would greatly increase the delay because there are so many of them. We decided to use Redis [18] for our in-memory storage because of this issue. Less than one nanosecond is all it takes to obtain information [9]. Three sequential detection stages form the hybrid pipeline: Stage 1 uses threshold rules to quickly discover clear violations; Stage 2 identifies transactions that are unusual but yet meet the basic parameters; and Stage 3 applies Spring AI's contextual reasoning to refine the risk assessment by examining the user's behavioral fingerprint. This strategy is not just a list of steps; it also depends on how the tasks are done and in what order [2], [6]. Fig. 1 indicates that the payment procedure does not end until the final classification choice is made.

A. Hybrid AI-Based Fraud Detection Model

At the core of SentinelPay lies the hybrid detection model. Feature weights w_i are determined dynamically according to each feature's discriminative contribution to fraud identification, derived from historical transaction patterns and continuously refined through Spring AI contextual analysis. Features exhibiting strong empirical correlation with fraudulent activity—including anomalous transaction amounts, abnormally high transaction frequency, and device-profile inconsistencies—receive elevated weights.

Weighted feature aggregation yields a scalar risk score, which is then passed through a sigmoid activation to produce the posterior fraud probability. The Stage 3 Spring AI inference component subsequently refines this score through dynamic contextual reasoning that generalizes beyond static rule boundaries—a characteristic that sets the hybrid model apart from purely rule-based or static classifiers, delivering consistently high precision and recall across diverse fraud scenarios [6], [14].

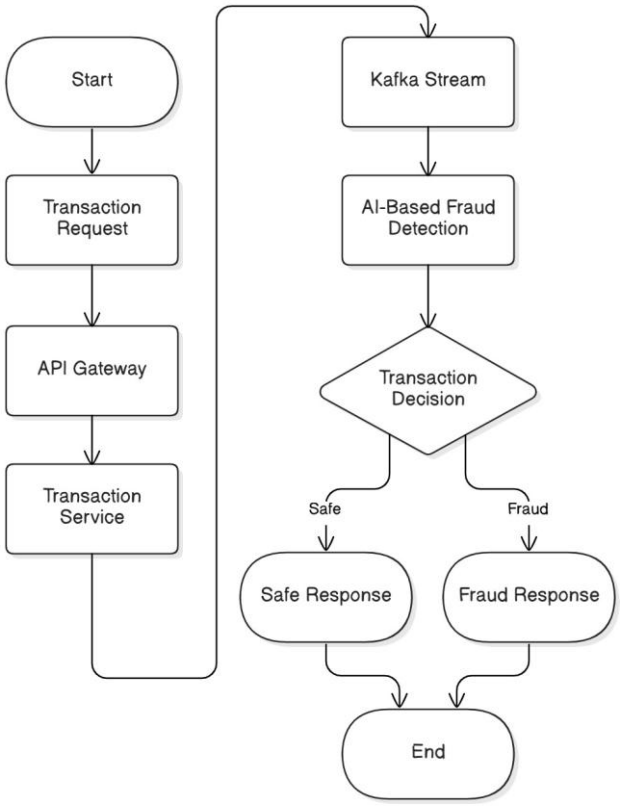

Fig. 1. End-to-end transaction workflow of SentinelPay, illustrating the sequence from transaction initiation through API Gateway authentication, event streaming, in-memory behavioral profile retrieval, Spring AI risk scoring, threshold-based classification, and real-time user notification dispatch.

B. Spring AI Inference Specification

Within the Fraud Detection Service, Spring AI functions as a feature-augmented behavioral scoring module. Learned behavioral embeddings—extracted from historical transaction patterns held in the in-memory caching tier—are applied to adaptively reweight the risk score R delivered by Stages 1 and 2, using a sigmoid-activated linear classifier selected for sub-millisecond inference time. A key architectural property of the Spring AI layer is *model-agnosticism*: the underlying inference engine is interchangeable—ranging from a lightweight logistic regressor for latency-critical deployments to a transformer-based sequence model for higheraccuracy scenarios—without any modification to the microservice pipeline or event streaming topology [6], [14].

IV. MATHEMATICAL MODEL

Each transaction is encoded as a multi-dimensional feature vector:

$T = \{x_1, x_2, \ldots, x_n\}$ (1)

where x_i captures individual transaction attributes including transaction amount, behavioral metrics, device information, and geolocation.

Fraud identification is cast as a binary classification problem:

$$f(T) = \begin{cases} 1, & \text{Fraudulent} \\ 0, & \text{Legitimate} \end{cases} \quad (2)$$

A weighted linear combination of feature vector components yields the scalar risk score R:

$$R = \sum_{i=1}^{n} w_i \cdot x_i \quad (3)$$

where $w_i \in \mathrm{R}$ is the learned weight for the i-th feature, reflecting its discriminative contribution to fraud likelihood.

Each transaction receives a label according to:

$$D(T) = \begin{cases} \text{Fraud}, & R > \theta \\ \text{Legitimate}, & R \le \theta \end{cases} \quad (4)$$

where θ is an empirically calibrated risk threshold.

Mapping R through a sigmoid function gives the posterior fraud probability:

$$P(\text{Fraud} \mid T) = \frac{1}{1 + e^{-R}} \quad (5)$$

The dual-output formulation accommodates both hard decisions through $D(T)$ and probability-calibrated scoring through $P(\text{Fraud} \mid T)$, allowing operators to tune the operating threshold to their specific risk appetite.

V. FRAUD DETECTION ALGORITHM

Algorithm 1 details the complete hybrid AI-based fraud detection procedure, tracing all stages from transaction initiation through classification decision and user notification.

Algorithm 1 Hybrid AI-Based Real-Time Fraud Detection

Input: Transaction $T = \{x_1,\ldots,x_n\}$; threshold θ Output: Classification — *Fraud* or *Legitimate*

1. User initiates transaction via the frontend interface.
2. API Gateway authenticates and routes the request to the Transaction Service.
3. Transaction Service publishes the event to the designated streaming topic.
4. Fraud Detection Service consumes the event from the streaming topic.

5. Extract feature vector T: amount, frequency, device ID, geolocation.
6. Retrieve user behavior profile from the in-memory caching tier.
7. Stage 1 — Rule-based validation: flag if any hard threshold is violated.
8. Stage 2 — Statistical anomaly detection: score $R \leftarrow \sum_{i=1}^{n} w_i \cdot x_i$ distributional outliers.
9. Compute weighted risk score: .
10. Stage 3 — Spring AI contextual reasoning: refine R adaptively.
11. Compute posterior probability: $P \leftarrow \frac{1}{1+e^{-R}}$.
12. if $R > \theta$ then

▷ Classify as *Fraud*; block transaction; dispatch alert.

13. else

▷ Classify as *Legitimate*; authorize and execute.

14. Return classification result and notification to frontend.

VI. SYSTEM ARCHITECTURE

SentinelPay was designed to be flexible, so you can add and remove services as needed. The REST API Gateway is the only means for the React.js frontend and backend services to talk to each other. It handles all user interactions [15]. You can only get in through this gateway. It uses JWT to verify users, sends requests to the right places, and spreads the load evenly across the backend.

Each service—Transaction and Fraud Detection—works on its own. The only thing they all have in common is the Kafka event bus. This intentional split is there to keep problems with one service from affecting the other. Once the transaction is confirmed, it is transmitted to Kafka. The Fraud Detection Service's detection pipeline looks at the scenario and makes a quick decision. Keeping behavioral profiles near to the detection logic shortens search times, especially when service demand is high [3], [9]. MySQL allows you to make permanent records and audit trails.

There are six main parts to the system: the front end or user interface, the API Gateway, the microservice processing, the event streaming (via Kafka), the Spring AI inference, and the persistent storage. Each tier can be launched or scaled separately so that changes in one segment do not affect the whole stack. Fig. 2 shows the general layered structure and how different problems are grouped.

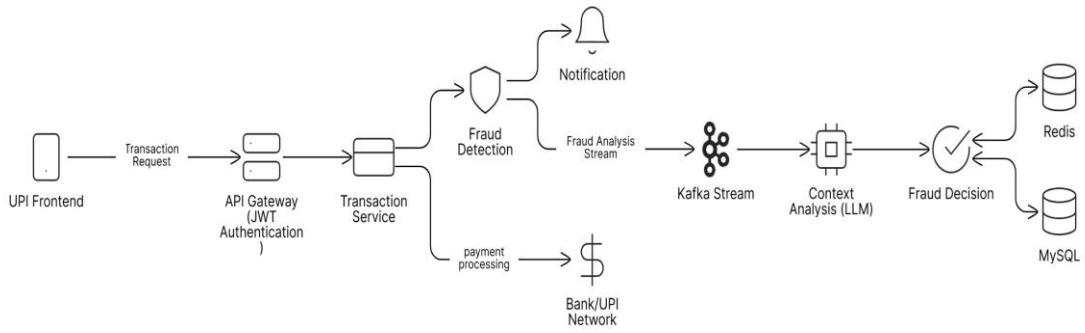

Fig. 2. Layered, security-aware microservice architecture of SentinelPay, delineating the frontend presentation tier, API Gateway, Transaction and Fraud Detection services, event streaming bus, in-memory caching tier, Spring AI inference engine, and MySQL persistent storage layer.

VII. IMPLEMENTATION DETAILS

In real life, people have employed SentinelPay technology. MySQL is a database that people use to store data for a long period. We utilize Redis [18] to store behavior and Apache Kafka [16] to handle event streams. You can use Spring Boot [17] to build microservices. The Fraud Detection Service is much better now. For instance, it might now use Spring AI to tell the difference between features, figure out how risky something is, and make choices according to the circumstance. This technique cuts down on network latency by processing everything on one workstation, like prior versions did, instead of using remote scoring sites.

It was possible to execute useful tests because each warning and transaction had its own Kafka topic. Warnings were always provided, even when there were more trades, because separate groups of clients were watching each broadcast [3]. There was increased demand, but we were able to keep things running smoothly this way. The API Gateway will always validate all incoming data to make sure that the users are who they say they are.

You may log in, buy goods, and see what is new on the front page. Prometheus uses metrics to keep a check on services, and Grafana presents real-time data including throughput, detection rates, and latency. The service level is where the

propagation happens. The API Gateway, Transaction Service, Notification Service, and Fraud Detection Service all work in their respective Docker containers. This kind stops problems from growing worse and helps things grow. Docker Compose manages the networks and service lifecycles for two different services: Kafka and Redis. The stateless fraud detection service can grow to handle more transactions as they come in.

VIII.DEPLOYMENT ARCHITECTURE

SentinelPay uses Docker containerization to make sure that environments can be moved, operations may grow, and services can be kept separate. The API Gateway, Transaction Service, and Fraud Detection Service are all separate services that run in their containers. This keeps the problems to the system, which lets it grow horizontally without affecting other services.

The in-memory caching service and the event streaming broker are set up as separate containerized parts. By putting all of the infrastructure in one deployment environment, Docker Compose takes care of networking and the lifecycle of services across containers. When there are a lot of transactions, especially during busy times, service replicas are made as needed to keep the service running smoothly and with high uptime.

The coordinated stack is made up of the API Gateway, Transaction Service, Notification Service, Authentication Service, Event Stream Broker, in-memory caching layer, MySQL, Grafana, and Prometheus. Each portion has its own host port, which makes it easier for them to talk to each other. Moving the containerized system to the cloud should be easy. After that, with a few changes, it can do a lot of financial duties on Azure, Amazon Web Services (AWS), and Google Cloud Platform.

IX. DATASET DESCRIPTION

We gave SentinelPay a go using a sham dataset designed to mimic actual digital payments, with an emphasis on UPI-based services. The need to safeguard data privacy in the absence of a publicly available labeled real-time transaction stream and the requirement for a controlled and repeatable class distribution that corresponds to the approximately 5% fraud rate observed in actual payment systems were the two main motivating factors in the development of synthetic generation [12]. The characteristics of production transactions as they pertain to attack patterns are accurately shown by the synthesis class distribution when using the current UPI fraud taxonomy.

The final corpus contains 50,000 records, with a fraud margin of 5%. There are 2,500 false records for every 47,500 genuine ones.

The feature vector of a transaction includes the following information: the amount, the timestamp, the user ID, the device information (ID, operating system, and type), the geolocation (latitude and longitude), the merchant ID, the transaction category (transfer, recharge, or bill payment), and metrics that show the average amount and frequency of transactions for each user in the past.

People may construct phony records that looked legitimate with anomaly insertion. Attack vectors that were a problem included timing signatures that did not match, transaction volumes that were too high, transactions that went too quickly, worries about geolocation, and worries about finding devices. Before training, we cleaned up the data by encoding everything once, getting rid of duplicates, and changing the widths of the features. Then we used min-max to make everything the same. These tactics helped the model come together quite quickly and correctly. The in-memory cache layer keeps the merged user profiles so that they can be accessed more quickly during inference.

A. Integration with Real-World Datasets

SentinelPay is architecturally compatible with established fraud detection benchmarks, including the IEEE-CIS Fraud Detection dataset [12], which shares core feature dimensions— transaction amounts, device fingerprinting, and temporal patterns—with the synthetic benchmark used herein. For production deployment, preprocessing historical transaction records and ingesting them into the event streaming pipeline is sufficient; the AI-based inference architecture requires no structural modification. The dataset integration pipeline is depicted in Fig. 3.

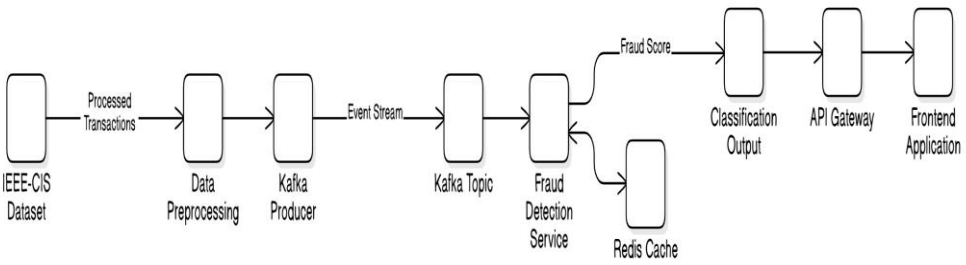

Fig. 3. Dataset integration pipeline depicting preprocessing, normalization, feature extraction, and streaming ingestion stages, applicable to both the synthetic benchmark and real-world sources such as the IEEE-CIS Fraud Detection dataset [12].

X. PERFORMANCE EVALUATION

The standard in Section IX is used to test SentinelPay. This benchmark has measurements and categories for the whole system. Accuracy is the percentage of transactions that are correctly identified.

SentinelPay can tell the difference between fraud and real transactions 95.2% of the time, although there is a big 5% class imbalance. Precision (93.8%) shows that the number of false alarms that could stop real user transactions has been cut down. It shows how many of the fraud forecasts are correctly identified as fraudulent. The recall rate of 94.6% accurately shows how many real fraud cases were successfully identified, which shows that the detection coverage is quite good and there are very few missed fraud episodes. The F1-score (94.2%) is a good way to measure how well a classification system works when the data is not evenly distributed.

Latency in processing: Three synchronized architectural strategies are used to achieve end-to-end classification in about 50 ms: (i) an in-memory caching layer is used to access behavioral profiles in less than one millisecond, and (ii) stateless event streaming is used to allow inference to happen asynchronously. SentinelPay lowers user-facing friction while keeping detection coverage the same, which leads to a 65.0% operational gain over the rule-based baseline with a 3.5% false positive rate. Using stateless, horizontally scalable microservice execution and concurrent event streaming, the system can handle more than 1,200 TPS of traffic at the same time. Fig. 4 shows how important performance metrics changed throughout the course of the evaluation trials, showing that the system is reliable in real time under a normal load.

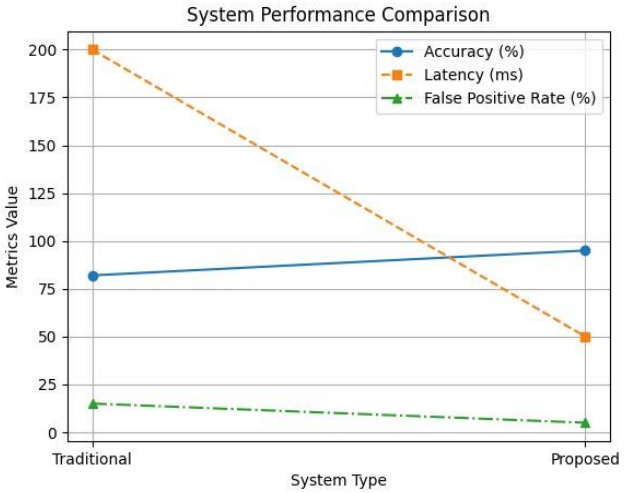

Fig. 4. Performance trends of SentinelPay across evaluation runs, showing concurrent trajectories of detection accuracy (95.2%), F1-score (94.2%), processing latency (~50ms), and transaction throughput (1,200 TPS), confirming stable convergence within target operational bounds.

XI. RESULTS AND DISCUSSION

We can tell how well SentinelPay works by looking at 50,000 transactions, 5% of which are false. Finding objects 95.2% of the time is not as good as recall (94.6% accuracy) and precision (93.8% accuracy) because the data is not evenly distributed. You are more likely to catch someone who lies a lot. About 20% of trades that seem shady are true, based on the accuracy rate.

Every second, systems process 1,200 transactions and stop 42 legitimate payments. This is a huge improvement over the old rule-based system, which would stop more than 120 times a second. It cuts down on the necessity for more investigation around 65% of the time. The F1 test works well in the real world, where class distributions are different. This is shown by the fact that 94.2% of people pass it.

Latency is difficult to resolve. Many parts need to work together for the 50 ms requirement to be met. The Fraud Detection Service uses Spring AI inference directly, which cut down on network expenses by a lot. Redis fixed database issues, and Kafka's ability to process events in real time made it straightforward to add new entries. It will be harder to accomplish the delay goal if any of these improvements are taken away. Changing the size of the scale was also crucial. The throughput stayed at 1,200 TPS since there were so many copies of the Fraud Detection Service running. Kafka was employed in each of these situations. Because of this growth plan, the team was able to keep up its high level of performance even as demand for services grew.

Table I makes it easy to see how numbers stack up against a rule-based norm. SentinelPay is now 13.2% more reliable, 75% faster, and 3.5% less likely to give false positives (from more than 10% to less than 5%). Making a distributed microservices design could be difficult, but the benefits for management are worth the effort. Table II reveals that rulebased detection is correct 83.4% of the time, but it also detects many false positives. Giving points for statistical mistakes makes the process more accurate and cuts down on the number of false positives. With contextual AI, the accuracy goes up to 95.2% and the number of false positives goes down to 3.5%.

TABLE I QUANTITATIVE COMPARISON: RULE-BASED BASELINE [7], [10] VS. SENTINELPAY (FULL SYSTEM)

Metric	Baseline	SentinelPay	Gain
Processing Latency	~200ms	~50ms	↓75%
Detection Accuracy	82.0%	95.2%	+13.2pp
False Positive Rate	>10%	3.5%	↓65%
Throughput	Low (Batch)	~1,200 TPS	↑High
F1-Score	N/A†	94.2%	—
Precision	N/A†	93.8%	—
Recall	N/A†	94.6%	—
Scalability	Limited	High	—
Processing Mode	Batch	Real-Time	—

†Rule-based systems report aggregate accuracy only; per-class P/R/F1 are undefined in threshold-only architectures.

XII. USER INTERFACE DESIGN

The front end of SentinelPay is built with React.js and looks like a real-world digital payment system like UPI. REST APIs routed through the API Gateway are used for communication on the backend. They quickly tell you what kind of fraud each transaction is.

Fig. 5 depicts the user interface workflow in three steps: user authentication, transaction submission, and fraud detection result. Fig. 5(a) shows the login screen, which needs

TABLE II STAGE-WISE PERFORMANCE COMPARISON OF THE THREE-STAGE HYBRID DETECTION PIPELINE

Configuration	Acc.	F1	FPR	Latency
Stage 1: Rule-Based only	83.4%	N/A*	11.2%	~15ms
Stage 1+2: Rule + Statistical	89.7%	87.6%	7.1%	~28ms
Stage 1+2+3: Full System	95.2%	94.2%	3.5%	~50ms

*Stage 1 applies hard threshold decisions; per-class P/R/F1 metrics are not applicable.

JWT-based authentication. The API Gateway checks session tokens. Fig. 5(b) shows the transaction submission screen that customers will see after they log in. This is a structured multifield form that gathers the transaction ID, amount, location, device, and kind of merchant. It also has a panel for searching judgments. Fig. 5(c) shows the outcome dashboard right away. It shows that the system can find fraud in less than a second from start to finish, with a high-confidence fraud prediction blocking the transaction at 85% fraud confidence on the AI risk-score gauge.

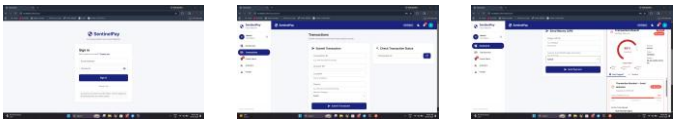

Fig. 5. SentinelPay user interface workflow: (a) JWT-secured login with API Gateway session token management, (b) transaction submission form with multi-field input and real-time verdict retrieval, and (c) fraud outcome dashboard displaying a blocked transaction at 85% confidence on the AI riskscore gauge.

XIII. CONCLUSION

SentinelPay uses a security-aware, event-driven microservice architecture to discover fraud in secure digital payment environments in real time, before it happens. The key new thing about it is a three-stage detection pipeline that uses rule-based validation, statistical anomaly detection, and Spring AI adaptive contextual reasoning in a pre-execution, microservice-native design. The authors assert that this combination has not been previously established in the fraud detection literature. Combining high-throughput event streaming, sub-millisecond in-memory behavioral caching, and adaptive AI inference leads to both high detection accuracy and endto-end latency that works in the real world.

Measured results show that the system has a detection accuracy of 95.2%, an F1 score of 94.2%, a precision of 93.8%, a recall of 94.6%, an end-to-end latency of about 50 ms, and a sustained throughput of 1,200 TPS. This is an improvement of 13.2 percentage points in accuracy, 75% in latency, and 65% in false positive rate over rule-based baselines. Table II shows that each pipeline stage makes a measurable, independent contribution. Stage 1 enforces deterministic border control, Stage 2 expands coverage to distributional anomalies, and Stage 3 facilitates adaptive inference across novel fraud patterns. This shows that the cascade design is a complete and necessary part of engineering.

SentinelPay is a platform that can grow, work well, and be used in production to stop financial fraud in the future. You can use it directly with UPI payment systems, mobile banking services, and high-volume e-commerce payment gateways that handle millions of transactions every day. The pre-execution, security-aware design has been tested for accuracy, latency, and throughput all at once. It allows for complicated and safe transaction validation to happen in real time when

it comes to today's digital financial systems. It also gives you a design template that you can use to develop fraud protection systems that are ready to be used in banking and fintech.

XIV. FUTURE WORK

We will soon use more complex deep learning structures, such as transformer-based sequence models and graph neural networks [8], to better grasp the similarities and distinctions between lying and transactions.

Through the introduction of additional possibilities to conduct fraud and contribute data into the model, adaptive online learning has the potential to improve inference at the Stage 3 development stage.

To guarantee that rules are adhered to, an explainable artificial intelligence (XAI) technique is employed [7]. The General Data Protection Regulation (GDPR) tells us how to manage data. The Reserve Bank of India (RBI) and the Payment Card Industry Data Security Standard (PCI-DSS) both give information on how to process payments and get clearance. The goal of this form of teamwork is to make it easier for businesses to employ cloud computing systems.

To confirm that the notion is compatible with a wide range of payment systems, we will put it through its paces by testing it with data taken from the real world, such as the IEEE-CIS Fraud Detection standard [12].

ACKNOWLEDGMENT

The authors thank the Department of Computer Science & Engineering, Mohamed Sathak Engineering College, Ramanathapuram, India, for providing the infrastructure and institutional support that facilitated this research.

REFERENCES

[1] A. S. Mandlik, M. S. Ganeshpure, C. N. Kadadas, L. Korra, J. U. Kidav, and M. A. Lavadkar, “AI-driven real-time threat detection for UPI transactions,” in *Proc. 2025 Int. Conf. Applications of Machine Intelligence and Data Analytics (ICAMIDA)*, Aurangabad, India, 2025, doi: 10.1109/ICAMIDA64673.2025.11209290.

[2] S. Acharjee et al., “Streaming-based anomaly detection for real-time scalable credit card fraud prevention: A comparative study,” in *Proc. 2025 Int. Conf. Engineering Innovations and Technologies (ICoEIT)*, 2025.

[3] V. Suggula and R. R. Goli, “Real-time fraud detection in banking transactions using Kafka streaming and machine learning ensemble,” in *Proc. 2025 Int. Conf. Advancement in Futuristic Technologies (ICAFT)*, 2025.

[4] G. Venkatasubbu, “Streaming intelligence: Real-time fraud detection in retail payments using machine learning,” in *Proc. 2025 Int. Conf. Engineering and Emerging Technologies (ICEET)*, 2025.

[5] V. Gogineni, “Apache Kafka for low-latency AI-enhanced data analytics in fraud detection systems,” in *Proc. 2025 IEEE Int. Seminar on Artificial Intelligence, Networking and IT (AINIT)*, 2025.

[6] S. K. A. Ramesh et al., “Real-time AI-enabled anomaly detection system for preventing financial fraud,” in *Proc. 2025 Int. Conf. Information Systems and Computer Networks (ISCON)*, 2025.

[7] A. Okonkwo, “Real-time fraud detection in digital banking using explainable machine learning and data engineering pipeline,” in *Proc. 2025 Int. Conf. Electrical and Computer Engineering Researches (ICECER)*, 2025.

[8] N. Chhaparia et al., “Real-time UPI fraud detection using graph neural networks,” in *Proc. 2025 IEEE Pune Section Int. Conf. (PuneCon)*, 2025.

[9] R. K. Mallidi et al., “Streaming platform implementation in banking and financial systems,” in *Proc. 2022 Asian Conf. Innovation in Technology (ASIANCON)*, 2022.

[10] A. Kumar et al., “Real-time SOA-based credit card fraud detection system using machine learning techniques,” in *Proc. 2021 Int. Conf. Computing Communication and Networking Technologies (ICCCNT)*, 2021.

[11] K. Kumar et al., “Machine learning solutions for investigating stream data using distributed frameworks: A literature review,” in *Proc. 2021 IEEE Asia-Pacific Conf. Computer Science and Data Engineering (CSDE)*, 2021.

[12] IEEE Computational Intelligence Society, “IEEE-CIS fraud detection dataset,” Kaggle, 2019. [Online]. Available: https://www.kaggle.com/c/ ieee-fraud-detection

[13] S. Roy et al., “Deep learning for financial fraud detection: A survey,” in *Proc. 2023 IEEE Int. Conf. Big Data (BigData)*, Sorrento, Italy, 2023, pp. 1–10.

[14] A. K. Jain et al., “Real-time anomaly detection in financial transactions using machine learning,” in *Proc. 2022 IEEE Int. Conf. Data Science and Advanced Analytics (DSAA)*, Shenzhen, China, 2022, pp. 1–9.

[15] M. Gupta et al., “Event-driven microservices architecture for scalable financial systems,” in *Proc. IEEE Int. Conf. Cloud Computing*, 2021.

[16] Apache Software Foundation, “Apache Kafka documentation,” [Online]. Available: https://kafka.apache.org/documentation/

[17] VMware, “Spring Boot documentation,” [Online]. Available: https:// spring.io/projects/spring-boot

[18] Redis Ltd., “Redis documentation,” [Online]. Available: https://redis.io/ docs/