

AI-ASSISTED LEDGER-BASED PAYMENT PROCESSING SYSTEM

V. Ruhee¹, S. Srihita², S. Keerthan³, T. Jashwanth⁴

¹²³⁴Students, Department of Computer Science and Engineering, HITAM, Telangana, India Project Guide: P. Ramesh

Abstract - Digital payment systems require secure transaction processing, reliable accounting, and timely identification of suspicious activity. This paper presents an AI-Assisted Ledger-Based Payment Processing System that combines wallet-based payments, double-entry ledger accounting, and an AI-assisted fraud analysis service. The proposed architecture contains a mobile application, API Gateway, Payment Service, Ledger Service, AI Fraud Service, and PostgreSQL database. Payment requests are authenticated and validated before transaction processing. The fraud service analyses transaction amount, payment frequency, account age, and historical transaction patterns to generate a risk assessment. After authorization, the Ledger Service records matching debit and credit entries. The architecture separates probabilistic fraud analysis from deterministic financial recording, supporting transaction traceability, consistency, fraud awareness, and auditability.

Keywords: Digital Payment, Fraud Detection, Artificial Intelligence, Double-Entry Ledger, Digital Wallet, PostgreSQL, Transaction Security, Payment Processing

1. INTRODUCTION

Digital payment applications allow users to transfer money rapidly through wallets and online accounts. A reliable payment platform must do more than move an amount from one account to another. It must authenticate the user, verify authorization, validate the request, maintain correct balances, preserve transaction history, and provide a traceable record of each financial event. These requirements become more important when payment activity is frequent and suspicious behavior can change over time.

Fraud detection is challenging because legitimate and suspicious transactions may share similar characteristics. A transaction can become unusual because of its amount, frequency, account age, or relationship to previous activity. Static rules are useful for known conditions, but behavioral analysis can provide additional information before a payment is finally authorized.

The proposed project combines an AI-assisted fraud analysis service with a double-entry ledger architecture. The AI component supports risk assessment before authorization, while the ledger component remains responsible for financial recording. This separation prevents a predictive model from directly modifying accounting records and allows the fraud analysis method to evolve independently.

The system follows a modular service-oriented design. The Mobile Application provides the user interface. The API Gateway is the controlled entry point. The Payment Service coordinates validation and authorization. The AI Fraud Service evaluates transaction behavior. The Ledger Service records the financial consequence of an approved payment, and PostgreSQL stores persistent data.

1.1 BACKGROUND

Modern payment systems must handle several operations within a short transaction interval. A request may involve authentication, wallet lookup, balance validation, risk assessment, authorization, database updates, and response generation. If these operations are not coordinated correctly, duplicate payments, inconsistent balances, incomplete records, or weak audit trails may result.

A double-entry ledger provides a structured way of representing a financial event. For a transfer of amount X from a source wallet to a destination wallet, the system records a debit of X on the source side and a credit of X on the destination side. Equality between the two sides provides a consistency invariant that can be checked independently of the fraud model.

1.2 NEED FOR THE PROPOSED SYSTEM

The project addresses the need for a payment workflow that combines security-aware authorization with reliable financial recording. Instead of treating fraud detection and accounting as one operation, the design establishes a sequence in which risk is assessed first and the ledger is updated only after the payment is authorized. This provides a clear boundary between decision support and deterministic transaction processing.

1.3 SYSTEM MOTIVATION

The motivation for combining these mechanisms is to make payment processing both risk-aware and accounting-aware. The AI component can examine behavior before authorization, while the ledger provides a deterministic record after authorization. The two functions can therefore be improved and tested independently without weakening the core financial invariant.

1.4 KEY DESIGN CONSIDERATIONS

The design gives priority to traceability and controlled state changes. Each payment is associated with a transaction identifier so that the request, risk assessment, authorization result, and ledger entries can be connected. The architecture also separates persistent financial data from the user interface, reducing the chance of direct client-side manipulation.

Another consideration is recoverability. A payment that fails before commit should not leave a partial financial record. The combination of explicit transaction states, database transactions, idempotency, and audit information provides a structured basis for handling retries and failures.

2. PROBLEM STATEMENT

The main problem is to design a wallet-based payment system that can assess unusual transaction behavior before authorization while maintaining consistent, traceable, and balanced financial records. The workflow must prevent unauthorized access, validate the sender and receiver, verify available balance, coordinate fraud analysis, and avoid partial ledger updates when a transaction fails.

A simple wallet implementation may update a sender balance and a receiver balance without maintaining a detailed representation of the event. Such an approach can make reconciliation and auditing more difficult. The proposed system instead records the financial event using matching debit and credit entries linked to the same transaction identifier.

The system must also address repeated requests. Network delays or client retries may cause the same payment request to reach the server more than once. An idempotent transaction identifier and a unique database constraint can be used to prevent the same financial event from being posted twice.

2.1 OBJECTIVES

- Provide secure registration and authentication for users.
- Support wallet-to-wallet digital payments.
- Maintain balanced debit and credit ledger entries.
- Analyze transaction behavior before final authorization.
- Generate an explainable fraud-risk assessment.
- Maintain transaction history and audit-oriented information.
- Preserve database consistency through PostgreSQL transactions.

- Separate payment coordination, fraud analysis, and ledger recording.

2.2 SCOPE

The scope includes user registration, authentication, wallet management, balance checking, wallet-to-wallet payments, transaction validation, fraud-risk assessment, authorization, ledger posting, transaction history, and persistent storage. The design also considers security controls and consistency testing for payment operations.

The project is a prototype-level architecture. It does not claim integration with banking networks, regulatory settlement systems, or a production-scale payment switch. A fraud model also requires a representative dataset and experimental validation before numerical performance claims can be made.

2.3 SYSTEM REQUIREMENTS

Functional requirements: authenticate users, create wallets, initiate payments, validate sender and receiver information, check balance, perform risk analysis, authorize or reject transactions, post ledger entries, and display transaction history.

Non-functional requirements: preserve confidentiality, integrity, availability, consistency, traceability, and maintainability. Payment operations should be atomic, API access should be authenticated and authorized, and sensitive data should be protected in transit and at rest.

3. LITERATURE REVIEW

Research has explored artificial intelligence and machine learning for payment-card and transaction fraud detection. Ryman-Tubb, Krause, and Garn surveyed AI and machine-learning approaches for payment card fraud and discussed the challenges of applying research techniques in operational environments [1]. Their work supports the use of transaction and behavioral information as inputs to fraud analysis.

Carcillo et al. investigated the combination of supervised and unsupervised learning for credit-card fraud detection [2]. Their study illustrates how anomaly information can complement supervised learning when fraudulent behavior is not represented by one fixed pattern. This motivates the proposed use of historical transaction behavior alongside current payment attributes.

3.1 FRAUDS-DETECTION CONSIDERATIONS

Fraud datasets are commonly imbalanced because legitimate transactions greatly outnumber fraudulent transactions. Therefore, evaluation should consider precision, recall, F1-score, false-positive rate, and false-

negative rate instead of relying only on overall accuracy. These measures can be applied when the proposed fraud component is experimentally evaluated.

3.2 TRANSACTION INTEGRITY

From a system-design perspective, prediction alone cannot guarantee financial correctness. A model can produce a risk assessment, but the payment platform still requires deterministic validation, authorization, and accounting. The proposed architecture therefore keeps the AI Fraud Service separate from the Ledger Service.

3.3 TRANSACTIONS-SECURITY IMPLICATIONS

The research also highlights the importance of evaluating false positives and false negatives. In a payment workflow, an overly sensitive detector may interrupt legitimate activity, while a weak detector may miss suspicious behavior. The prototype therefore treats the model output as decision support and keeps authorization policy explicit.

3.4 LITERATURE SYNTHESIS

The literature indicates that fraud detection benefits from combining transaction-level information with behavioral evidence, while reliable payment processing requires deterministic validation and accounting. The proposed architecture follows both observations by placing risk analysis before authorization and ledger posting after authorization.

3.5 PRACTICAL IMPLICATION

For a project prototype, the separation of responsibilities simplifies debugging and testing. Developers can test the fraud service using historical transaction features and independently test the ledger service using controlled payment events. The Payment Service then becomes the point at which the two outputs are coordinated.

3.6 RESEARCH GAP

Existing payment-security research and fraud-detection research provide methods for recognizing suspicious behavior, while accounting systems provide deterministic mechanisms for representing financial events. The project focuses on connecting these two functions within a single payment workflow while keeping their responsibilities separate.

The research gap addressed by the project is the integration of pre-authorization fraud assessment with double-entry transaction recording. The Payment Service acts as the coordinator: it requests risk analysis before authorization and invokes the Ledger Service only when the transaction is permitted. This creates a clear control

boundary between probabilistic analysis and financial consistency.

The separation also supports maintainability. A future fraud model can be trained or replaced without redesigning the ledger structure. Likewise, ledger controls can remain stable while the fraud component is improved.

4. PROPOSED SYSTEM

The proposed system consists of six major components: Mobile Application, API Gateway, Payment Service, AI Fraud Service, Ledger Service, and PostgreSQL Database. Each component performs a defined role in the payment lifecycle.

4.1 COMPONENT RESPONSIBILITIES

Component	Responsibility
Mobile Application	Registration, authentication, wallet operations, payment initiation, balance and transaction history.
API Gateway	Authentication boundary, routing, request validation and controlled API exposure.
Payment Service	Validation, balance checks, fraud coordination, authorization and workflow control.
AI Fraud Service	Analysis of transaction and historical features; returns a risk assessment.
Ledger Service	Creates matching debit and credit entries and maintains financial traceability.
PostgreSQL	Persistent storage for users, wallets, payments, ledger entries and fraud assessments.

4.2 ARCHITECTURAL PRINCIPLE

The AI component recommends or flags while the Ledger Service records the approved financial consequence. The model is not permitted to write arbitrary financial entries. This separation reduces coupling and allows the system to be evaluated in stages.

4.3 DESIGN PRINCIPLES

The proposed architecture follows modularity, separation of concerns, traceability, and deterministic financial recording. The user-facing application does not directly modify ledger entries. The Payment Service controls the workflow, and the Ledger Service remains the only component responsible for creating the financial event.

The same principle applies to fraud analysis. The AI Fraud Service receives defined inputs and returns a risk assessment. It does not directly authorize or reject a

transaction outside the policy enforced by the Payment Service.

PROPOSED SYSTEM ARCHITECTURE

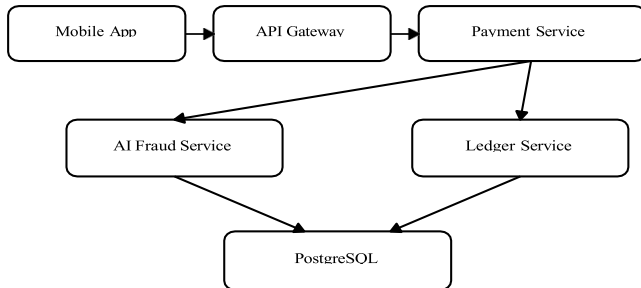


Fig. 1. Proposed AI-assisted ledger-based payment processing architecture.

The architecture separates user interaction, request routing, payment orchestration, risk analysis, financial posting, and persistence. This modular structure makes it possible to test each service independently while preserving the complete transaction workflow.

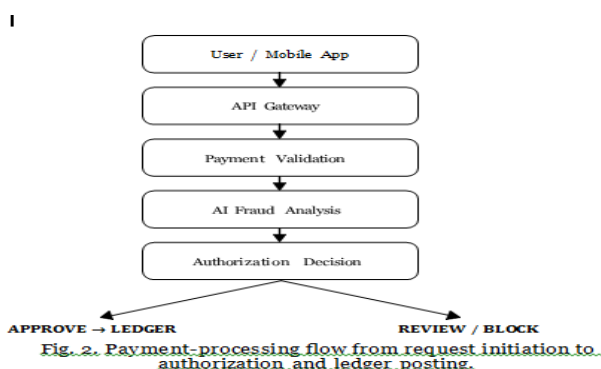
4.4 DATA FLOW AND SERVICE BOUNDARIES

The data path begins with the payment request and ends with a persisted financial event. The API Gateway is the external boundary, the Payment Service is the workflow coordinator, the AI Fraud Service provides risk information, and the Ledger Service creates the accounting entries.

5. PAYMENT PROCESSING METHODOLOGY

A payment begins when an authenticated user submits the sender wallet, receiver wallet, amount, and transaction request. The API Gateway verifies the request context and routes it to the Payment Service. The Payment Service validates the request, checks wallet status and available balance, and prepares the transaction features required for risk assessment.

PAYMENT PROCESSING FLOW



The flow ensures that the ledger is not updated before authorization. If risk analysis or another validation step rejects the payment, the transaction can be marked rejected without creating a completed financial event.

5.1 TRANSACTION STATES

- INITIATED: request received and assigned a transaction identifier.
- VALIDATED: authentication, wallet and amount checks succeed.
- RISK ASSESSED: fraud service returns a risk assessment.
- AUTHORIZED: configured payment policy permits the transfer.
- POSTED: balanced ledger entries are committed.
- REJECTED: payment is not permitted and no completed ledger event is created.
- FAILED: processing cannot complete and financial posting is not partially committed.

5.2 FAILURE HANDLING AND IDEMPOTENCY

Payment processing should be designed so that a failure before database commit does not leave only one side of the ledger. PostgreSQL transactions provide the database-level mechanism for grouping related changes. If an operation fails before commit, the transaction can be rolled back so that the financial state remains consistent.

Idempotency protects against duplicate requests. A client can retry a request after a timeout without knowing whether the original request reached the server. The Payment Service can use a unique transaction identifier or idempotency key to determine whether an equivalent event already exists.

5.3 LEDGER RECONCILIATION

After posting, the system can verify that the debit and credit sides are balanced, update the corresponding wallet state, and preserve the transaction reference for later audit. A failed reconciliation should prevent the system from treating the event as successfully completed.

5.4 AUDIT TRAIL

An audit-oriented record can retain transaction identifiers, timestamps, status transitions, ledger entry identifiers, and the associated risk assessment reference. These records support investigation and reconciliation while keeping sensitive information controlled.

5.3 DOUBLE-ENTRY LEDGER

DOUBLE-ENTRY LEDGER MODEL



$$\Sigma \text{ Debit} = \Sigma \text{ Credit}$$

Fig. 3. Double-entry representation of an approved wallet transfer.

For an approved payment of amount X, the source wallet receives a debit entry of X and the destination wallet receives a credit entry of X. Both entries are associated with the same payment event. The invariant $\Sigma \text{Debit} = \Sigma \text{Credit}$ can be checked after posting.

5.4 LEDGER RECORD STRUCTURE

A ledger entry can contain an entry identifier, transaction identifier, wallet identifier, entry type, amount, timestamp, and status. The transaction identifier links the two sides of the transfer. The ledger should be append-oriented for auditability, with controlled compensating entries used when a correction is required rather than silently changing historical events.

6. DATABASE DESIGN

PostgreSQL is used as the persistent database. The conceptual data model contains User, Wallet, PaymentTransaction, LedgerEntry, and FraudAssessment entities. A user may own one or more wallets, while a payment identifies sender and receiver wallets and stores the amount, timestamp, and transaction state.

CONCEPTUAL DATABASE MODEL

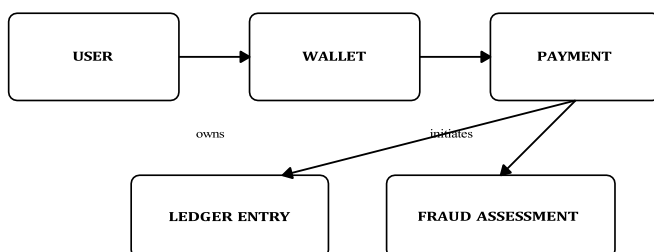


Fig. 4. Conceptual database relationships for the payment system.

The database model links the payment transaction with its financial entries and risk assessment. Foreign-key

relationships can prevent references to nonexistent users or wallets, while unique identifiers can support idempotency and traceability.

6.1 DATA CONSISTENCY

Database constraints should enforce primary keys, foreign keys, valid status values, and appropriate amount ranges. Financial amounts should use an exact numeric representation rather than floating-point arithmetic. Indexes on transaction identifiers and timestamps can support history and audit operations.

6.2 TRANSACTION SEQUENCE

PAYMENT TRANSACTION SEQUENCE

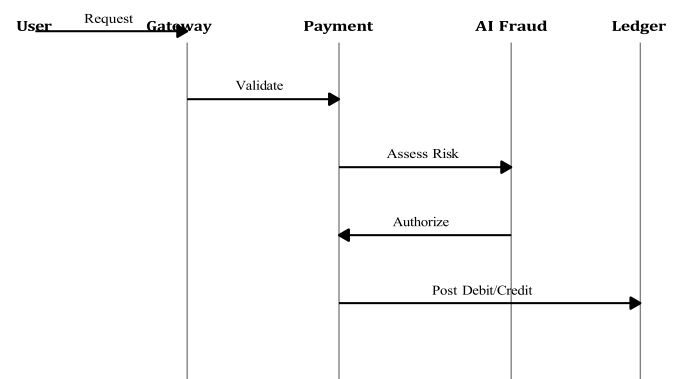


Fig. 5. Simplified transaction sequence among the client and backend services.

The sequence illustrates the separation of responsibilities. The client submits the request, the gateway routes it, the payment service coordinates validation, the AI service provides risk information, and the ledger service performs final financial posting after authorization.

6.3 AUDITABILITY

Transaction history should retain enough information to reconstruct the payment lifecycle. A common transaction identifier can connect the original request, risk assessment, authorization result, ledger entries, and final status. This supports investigation and reconciliation without requiring the AI model to modify accounting history.

6.4 CONCURRENCY CONTROL

Concurrent payment requests require careful balance handling. If two requests attempt to spend the same available balance at the same time, the system should use appropriate transaction isolation and locking or equivalent controls. The objective is to prevent the database from accepting a state that violates the wallet's financial rules.

6.5 DATABASE TRANSACTION LIFECYCLE

A payment-related database operation can be viewed as begin, validate, update, commit, and audit stages. The transaction should be committed only when all required financial changes succeed. If any mandatory step fails, the database transaction should be rolled back.

The AI Fraud Service examines current transaction attributes together with historical behavior. The project identifies transaction amount, payment frequency, account age, and historical transaction patterns as important features.

AI FRAUD-ANALYSIS PIPELINE

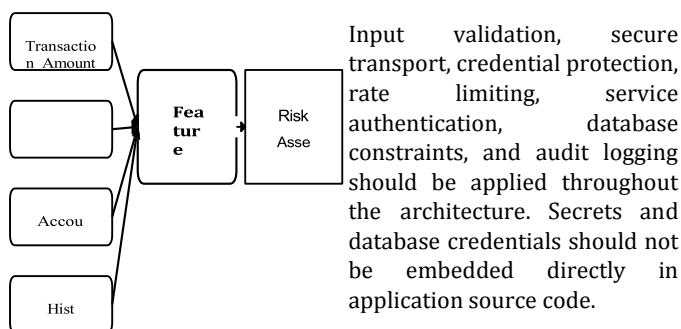


Fig. 6. AI-assisted fraud-analysis pipeline.

7. AI-ASSISTED FRAUD ANALYSIS

The fraud service converts transaction and behavioral information into a risk assessment. The result is returned to the payment workflow rather than directly modifying the ledger.

7.1 FEATURE ENGINEERING

Transaction amount provides information about the scale of the payment. Frequency captures repeated activity over a selected period. Account age provides context for account maturity, while historical patterns describe how the current transaction compares with previous behavior.

7.2 MODEL DECISION POLICY

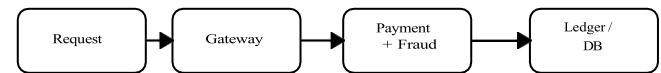
The risk result should be mapped to an explicit authorization policy. The policy can distinguish automatic approval, additional verification, manual review, or rejection. The threshold values are configuration choices and require experimental calibration using representative data.

7.3 MODEL MONITORING

After deployment, the distribution of transaction features and model decisions should be monitored for changes. Periodic evaluation can identify shifts in transaction

behavior and indicate when the model requires retraining or recalibration.

TRANSACTION DATA FLOW



Request-> validation-> risk-> authorization-> posting.

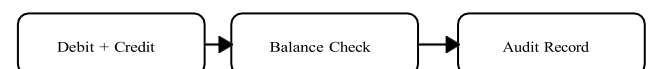
Fig. 8. Data movement across the major payment services.

8. SECURITY ANALYSIS

Security must protect identity, authorization, transaction integrity, and sensitive financial information. Authentication verifies the user, while authorization verifies whether that user is permitted to access a particular wallet or transaction object. Server-side ownership checks are important because a valid account should not automatically grant access to another user's financial resources.

8.1 THREATS AND CONTROLS

Threat	Control
Unauthorized access	Authentication, token/session protection and rate limiting.
Object-level authorization	Server-side wallet and transaction ownership checks.
Duplicate payment	Idempotency key and unique transaction constraint.
Partial ledger update	Atomic database transaction for financial posting.
Data tampering	Access control, integrity constraints and audit records.
Suspicious transaction	AI risk assessment plus authorization policy.



Balanced event-> wallet update-> traceable audit entry

Fig. 9. Reconciliation path for a posted ledger event.

SECURITY CONTROL LAYERS

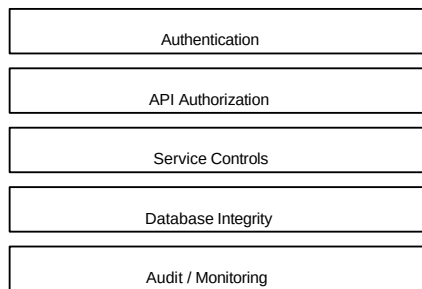


Fig. 10. Layered security controls for the payment platform.

8.2 API AND DATA SECURITY

API endpoints should validate input at the server, enforce authentication and authorization, restrict sensitive operations, and return controlled error information. Database access should be limited to services that require it. Logs should avoid exposing passwords, tokens, or unnecessary financial details.

8.3 SECURITY TESTING

- Test invalid credentials and unauthorized wallet access.
- Test duplicate requests and replayed transaction identifiers.
- Test insufficient balance and invalid amount conditions.
- Verify that failed payments do not create partial ledger events.
- Verify that sensitive fields are not exposed through API errors or logs.

9. IMPLEMENTATION METHODOLOGY

The prototype can be implemented using Java and Spring Boot for backend services, an Angular-based web interface or equivalent client layer, and PostgreSQL for persistence. The services should expose well-defined APIs and maintain clear boundaries between user interaction, payment coordination, fraud analysis, and ledger posting. Implementation can be organized into stages. The first stage establishes registration, authentication, and wallet management. The second stage implements payment validation and transaction-state handling. The third stage implements the double-entry ledger schema and atomic posting. The fourth stage integrates the AI Fraud Service using selected transaction features. The final stage applies security controls and performs functional, consistency, concurrency, and model evaluation.

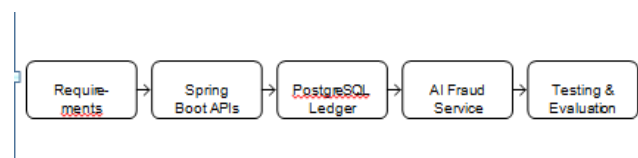
9.1 TESTING STRATEGY

- Functional testing of registration, authentication, wallet creation, balance retrieval, and payment initiation.
- Negative testing for invalid credentials, insufficient balance, invalid amounts, duplicate requests, and blocked transactions.
- Ledger consistency testing to verify matching debit and credit entries for every posted payment.
- Concurrency testing for simultaneous payment requests against the same wallet.
- Security testing for authentication, authorization, input validation, session handling, and API access control.
- Fraud-model evaluation using a representative labelled or appropriately constructed transaction dataset.

9.2 IMPLEMENTATION STACK

The service layer can use Java and Spring Boot, with REST-style APIs for communication between the client and backend services. Angular can provide a structured web interface for registration, wallet operations, payment initiation, and transaction history. PostgreSQL provides persistent storage for transactional and ledger data.

IMPLEMENTATION WORKFLOW



Build->integrate->validate->evaluate

Fig. 11. Prototype implementation and evaluation workflow.

The implementation moves from requirements to service development, ledger/database integration, AI integration, and testing. Each stage can be validated independently before the complete payment workflow is evaluated.

9.3 DEPLOYMENT CONSIDERATIONS

A future deployment should isolate services, protect configuration secrets, monitor database health, record controlled application logs, and apply backup and recovery procedures.

10. EXPECTED RESULTS AND EVALUATION

The expected outcome is a secure and traceable wallet payment workflow in which suspicious transactions can be identified before final authorization and approved payments are recorded as balanced ledger events. The

system should allow a payment to be traced from request through risk assessment, authorization, and ledger posting.

Evaluation should measure transaction-processing latency, successful transaction completion rate, ledger-balance consistency, duplicate-payment prevention, authorization correctness, and fraud-model performance. These are evaluation criteria, not completed experimental results. Numerical values should be reported only after implementation and testing.

10.1 LIMITATIONS

- The architecture is a project-level prototype rather than a production banking deployment.
- Fraud-model performance depends on dataset quality, class balance, and temporal characteristics.
- Risk thresholds require experimental calibration; no unverified fraud-detection accuracy is claimed.
- External payment rails, settlement systems, and regulatory integrations are outside the current scope.

10.2 FUTURE SCOPE

- Evaluate supervised, unsupervised, and hybrid fraud-detection approaches.
- Investigate explainable AI and analyst feedback mechanisms.
- Evaluate additional behavioral features and realistic concurrency workloads.
- Extend deployment with observability, secure key management, automation, and controlled external-service integration.

11. CONCLUSION

The AI-Assisted Ledger-Based Payment Processing System combines wallet-based payments, AI-assisted fraud analysis, and double-entry ledger accounting in a modular service-oriented architecture. AI supports risk assessment before authorization, while the Ledger Service remains responsible for deterministic financial recording. The workflow emphasizes authenticated requests, validation, pre-authorization risk assessment, atomic ledger posting, traceability, and database consistency.

12. ACKNOWLEDGEMENT

The authors thank the Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management (HITAM), and the project guide, P. Ramesh, for guidance and support.

13. REFERENCES

- 1) S. Ryman-Tubb, P. Krause, and W. Garn, "How artificial intelligence and machine learning research impacts payment card fraud detection: A survey and industry benchmark," *Engineering Applications of Artificial Intelligence*, vol. 76, pp. 130–157, 2018, doi: 10.1016/j.engappai.2018.07.008.
- 2) G. Carcillo, Y.-A. Le Borgne, O. Caelen, and G. Bontempi, "Combining unsupervised and supervised learning in credit card fraud detection," *Information Sciences*, vol. 557, pp. 317–331, 2021, doi: 10.1016/j.ins.2019.05.042.
- 3) PostgreSQL Global Development Group, "Transactions," PostgreSQL Documentation.
- 4) PostgreSQL Global Development Group, "Transaction Isolation," PostgreSQL Documentation.
- 5) OWASP Foundation, "OWASP API Security Top 10 – 2023," 2023.