

Medical Insurance Claim Audit System

B. Pranitha¹

¹Computer Science and Engineering, Mahatma Gandhi Institute of Technology

Abstract - Manual processing of medical insurance claims can lead to delayed verification and document-level fraud such as duplicate submissions. This paper presents the Medical Insurance Claim Audit System, a Flask-based web application that automates document extraction, policy validation, and fraud analysis for insurance claim auditing. Uploaded documents, including Aadhaar cards, medical reports, and proposal forms, are processed using Tesseract OCR through the pytesseract library to extract document identifiers and treatment information. The extracted data is validated against predefined insurance policy rules covering coverage limits, treatment eligibility, claim dates, and duplicate document checks. A rule-based fraud scoring mechanism classifies claims as approve, review, or reject based on detected anomalies. The system also incorporates Security PIN and OTP-based user authentication, while audit records are stored in SQLite for claim traceability. Evaluation using simulated claim scenarios showed that the system correctly enforced policy validation rules and detected duplicate document submissions. The proposed system provides a lightweight rule-based approach for automated insurance claim auditing and fraud analysis.

Key Words: Medical insurance claims, OCR, Tesseract OCR, fraud detection, rule-based validation, insurance claim auditing, healthcare claim verification, Flask web application, audit logging, SQLite database

1.INTRODUCTION

The healthcare industry has witnessed sustained growth in the number and complexity of medical insurance claims over the past decade. Insurance providers must process large volumes of claims efficiently while simultaneously safeguarding against erroneous

submissions and deliberate fraud. Traditional claim processing pipelines rely substantially on manual document inspection, structured data entry, and human judgment, resulting in processing delays, inconsistencies, and limited scalability. These limitations are compounded by the inherent difficulty of verifying identity and treatment data from heterogeneous, unstructured document sources such as government-issued identification cards, physician-generated medical reports, and handwritten or digitally filled insurance forms.

Academic literature on the subject is bifurcated into two distinct streams. One stream focuses on advanced machine learning and graph-based fraud detection, offering high detection accuracy but requiring substantial labeled datasets, computational infrastructure, and domain expertise. The other stream focuses on document management and audit logging in electronic health records but does not address automated claim decision-making. No lightweight, unified, and deployable system has been proposed that integrates OCR-based document processing, policy-driven claim validation, numerical fraud scoring, and multi-level authentication into a single coherent pipeline.

The current research attempts to fill the gap in question with the development of the Medical Insurance Claim Audit System, which is a Flask-built web application that performs the automatic insurance claim life cycle process ranging from submitting documents to providing an output with the results of audit decisions. The proposed tool is intended to have the characteristics of being highly interpretable and rule-transparent but efficient and not cloud-based, thus running on common computers and not requiring any special hardware or

machine learning techniques. An end-to-end automated insurance claim processing pipeline is presented within the framework of the proposed system, encompassing such stages as document recognition via optical character recognition, claims validation based on predefined rules, and calculating a score associated with potential fraudulence of the processed claim. To provide extra safety, the proposed system features a three-tier authentication architecture that combines credential-based authentication, security PIN, and OTP validation. As part of the policy-driven validation engine incorporated into the application, there is an option to dynamically load the insurance claim parameters from the dataset, thus allowing managing rules without changing the codebase.

2. Body of Paper

The remainder of this paper is organized as follows. Section 3 surveys related literature. Section 4 identifies the research gap and problem statement. Section 5 presents the system objectives. Section 6 describes the proposed system and its architecture. Section 7 details the methodology and algorithms. Section 8 presents implementation specifics. Section 8 reports testing and evaluation results. Section 9 provides a comparative discussion. Section 10 concludes the paper with future directions.

3. Related Work

Research on automated medical insurance claim processing spans several interrelated domains including fraud detection, document digitization, audit log analysis, and rule-based decision systems. This section surveys representative works organized by thematic focus.

3.1 Fraud Detection in Insurance Claims

Muhammad et al. [1] proposed a fraud detection framework leveraging Graph Neural Networks (GNNs)

to capture relational patterns among patients, physicians, and claims. Their model demonstrated strong detection accuracy and provided post-hoc explainability for identified fraudulent patterns. However, the framework requires large-scale structured graph datasets and significant computational resources, limiting its applicability in lightweight, real-time deployment contexts.

Du Preez et al. [2] conducted a systematic review of machine learning techniques applied to healthcare claim fraud detection, covering supervised classifiers, ensemble methods, and anomaly detection models. The review identified persistent challenges including class imbalance in training data, limited model interpretability, and inconsistent dataset availability across healthcare jurisdictions. These findings underscore the practical difficulty of deploying purely ML-based fraud detection in operational insurance systems.

3.2 Document Processing and Digitization

Arora [3] investigated intelligent document management systems for health insurance digitization, focusing on automated data extraction to reduce manual document handling. The study demonstrated efficiency gains in document throughput but did not extend to claim validation logic or fraud detection, leaving the system incomplete for end-to-end claim auditing purposes.

3.3 Audit Log Analysis in Healthcare Information Systems

Rule et al. [4] analyzed event logs from electronic health record (EHR) systems to identify workflow inefficiencies and usage patterns. While providing valuable operational insights, the study was scoped to system-level workflow optimization and did not address insurance claim validation or fraud analysis. Similarly, Kannampallil et al. [5] applied audit log data from EHR systems to study user behavior and system interaction

patterns, focusing on clinical workflows rather than claim-level decision-making.

Huilgol et al. [6] examined audit log utilization for healthcare quality improvement. Their work highlighted the value of audit data in performance monitoring but did not propose mechanisms for automated claim validation or fraud detection. Li and Chen [7] proposed a structured auditing framework for medical insurance fund management with an emphasis on financial transparency and risk control. However, their approach relied on conventional manual auditing processes without automation or real-time processing capabilities.

3.4 Rule-Based Systems and IT-Based Claim Processing

Umair Abdullah [9] proposed a data mining-driven rule-based expert system for medical billing compliance. The system combined mined historical patterns with predefined compliance rules to detect billing anomalies, achieving improved accuracy over purely manual review. However, the system was scoped to structured billing data and lacked document-level OCR extraction or real-time fraud detection capabilities.

Wojtusiak et al. [11] demonstrated a rule-based approach for predicting Medicaid claim payment outcomes using structured claim data. The study established the viability of rule-based decision logic for consistent claim assessment. However, the system operated exclusively on pre-structured tabular data and did not incorporate fraud detection or document processing. Park [12] discussed IT-based systems for standardizing claim review processes, noting improvements in evaluation consistency. The described systems nonetheless relied on predefined structured workflows and manual case handling for complex scenarios.

Wang and Zhang [12] addressed data integrity and consistency in insurance systems through data audit

techniques. Their approach targeted post-processing audit scenarios and did not extend to real-time automated claim decision-making. The Joint Learning Network publication [8] provided a governance-oriented toolkit for medical audit systems emphasizing structured guidelines for claim monitoring and compliance. However, the approach was inherently manual and did not incorporate intelligent automation, OCR, or fraud detection.

3.5 Summary of Literature Survey

Table 1: Summary of Literature Survey

Ref.	Authors & Year	Focus / Technique	Key Limitation
[1]	Muhammad et al. (2025)	GNN-based fraud detection for medical insurance claims	Requires large structured datasets; high computational cost
[2]	du Preez et al. (2024)	Systematic review of ML techniques for healthcare fraud	Data imbalance issues; limited interpretability
[3]	Arora (2024)	Document digitization via automated data extraction	No integrated validation or decision-making mechanism
[4]	Rule et al. (2023)	EHR event-log analysis for workflow optimization	No real-time claim validation or fraud detection
[5]	Kannampallil et al. (2022)	EHR audit log analysis for system and workflow analysis	Focuses on behavior analysis; lacks claim-level fraud detection
[6]	Huilgol et al. (2022)	Audit log analysis for healthcare quality improvement	No automation for claim validation or fraud detection
[7]	Li & Chen (2020)	Structured auditing framework for insurance fund management	Manual processes; no intelligent automation
[8]	JLN (2019)	Governance guidelines for medical audit systems	Largely manual; no OCR or real-time processing

[9]	Umair Abdullah (2015)	Data mining–driven rule-based system for billing compliance	No document-level OCR or real-time fraud detection
[10]	Park (2012)	IT-based claim review and assessment	Predefined workflows; no intelligent automation
[11]	Wojtusiak et al. (2011)	Rule-based prediction of Medicaid claim payments	Structured data only; no fraud detection mechanisms
[12]	Wang & Zhang (2010)	Data audit for medical insurance systems	Post-processing only; lacks automation and real-time detection

4. Research Gap and Problem Statement

4.1 Research Gap

A critical examination of the surveyed literature reveals a consistent pattern of fragmentation. Advanced fraud detection approaches, particularly GNN and ML-based systems [1][2], achieve strong detection accuracy under research conditions but require large labeled datasets, domain-specific feature engineering, and computational infrastructure that exceeds the capacity of typical insurance processing environments. Simultaneously, document digitization systems [3] address data extraction in isolation, without validating extracted data against policy rules or assessing fraud risk. Audit log and workflow studies [4][5][6] provide operational insights but do not address automated claim decision-making.

Rule-based systems [9][11] offer interpretable and consistent decision logic but operate exclusively on pre-structured, manually entered data. No existing lightweight system integrates OCR-based unstructured document processing, policy-driven rule validation, quantitative fraud scoring, multi-level authentication, and audit logging within a single deployable web application. This fragmentation results in either partial automation requiring significant manual intervention or

complex ML-dependent architectures impractical for lightweight operational deployment.

4.2 Problem Statement

Traditional medical insurance claim processing systems are characterized by heavy reliance on manual document verification, structured data workflows, and fragmented tool ecosystems. These systems fail to automate the extraction of relevant information from unstructured documents, apply policy rules dynamically, identify suspicious claim patterns in real time, or maintain comprehensive decision audit trails. There exists a need for a unified, interpretable, and deployable system that automates the complete claim verification lifecycle—from document upload and data extraction to policy compliance checking, fraud risk quantification, and timestamped decision logging—without requiring specialized ML infrastructure or manual data entry.

5. System Objectives

The Medical Insurance Claim Audit System is meant to facilitate the claim auditing procedure by introducing intelligent processes like document recognition, validation, fraud detection, and access controls. The automatic extraction of claim information from an image of a document through the use of Optical Character Recognition ensures efficiency through elimination of manual data entry. A dynamic and policy driven validation engine helps to validate claims according to various insurance parameters like limits of cover, treatments, waiting periods, and age. Fraud detection is achieved by means of an enhanced multi-criteria fraud detection engine that identifies fraudulent claims and assigns scores. Enhanced security levels are enforced through the use of triple factor authentication involving credential login, security PIN authentication, and OTP verification. According to the outcome of the validation and fraud analysis, the system makes deterministic decision on whether the claim is approved, rejected, or

needs a manual review. In addition, a detailed timestamped audit trail is generated for each claim. The system does not require any special computing power and can run on ordinary computer systems. There is no need of cloud infrastructure as it is not necessary for the system to be hosted in a server.

6. Proposed System

6.1 System Overview

The Medical Insurance Claim Audit System is a modular, web-based application built on the Python Flask framework. It operationalizes a sequential processing pipeline in which each stage performs a specific verification or analysis function. The pipeline ensures that claim data passes through authentication gates, document processing, policy validation, and fraud analysis before a decision is issued. The system is designed for local deployment on standard hardware and operates without requiring external APIs, cloud services, or GPU-based inference.

The system processes three categories of uploaded documents: government-issued identification (Aadhaar card), physician-generated medical reports, and completed insurance proposal forms. These document types collectively provide the identity, clinical, and policy context required for comprehensive claim evaluation.

6.2 System Architecture

The system follows a layered architecture comprising two principal operational layers:

6.2.1 Application Layer

The Application Layer constitutes the user-facing interface of the system, implemented as HTML-rendered Flask templates. It provides pages for user registration, credential-based login, document upload, security PIN entry, OTP verification, claim detail submission, and decision result display. The Application Layer is

designed to ensure smooth navigation across all workflow stages, presenting extracted OCR data and final decisions in a structured, readable format.

6.2.2 Processing Layer

The Processing Layer executes the backend operations of the system across six functional modules: Authentication, Document Processing, Claim Validation, Fraud Detection, Decision Engine, and Audit Logging. These modules operate sequentially, with each module receiving structured outputs from its predecessor and contributing to the final claim decision. Flask session management is employed to maintain user state across modules.

6.3 System Modules

The architecture comprises eight distinct functional modules, as described below and illustrated in Figure 1.

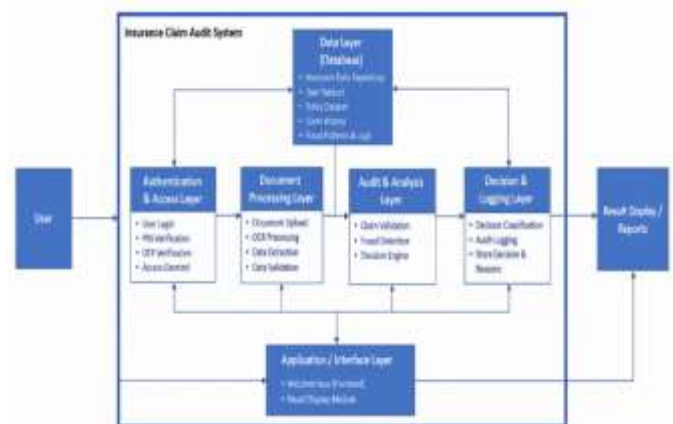


Figure 1: Module Interaction Diagram of the Medical Insurance Claim Audit System

6.3.1 User Interface Module

The User Interface Module provides the web-based interaction platform through which all user actions are initiated. It supports registration, login, document upload, claim submission, and result viewing. The module displays OCR-extracted data for user review prior to claim submission and presents the final decision along with supporting justifications.

6.3.2 Authentication Module

The Authentication Module enforces a three-factor identity verification process. At the first factor, the user provides a User ID and registered email address for credential verification against the SQLite database. At the second factor, the user must enter a Security PIN generated at registration time. At the third factor, a dynamically generated OTP is presented by the system and must be correctly entered to proceed. This layered authentication architecture prevents unauthorized claim submissions and protects user account integrity.

6.3.3 Document Upload Module

The Document Upload Module manages the receipt and handling of submitted document files. Users upload up to three document types—Aadhaar card, medical report, and insurance proposal form—through a structured interface. Uploaded files are processed directly from the file stream without persistent storage on disk, preserving system storage efficiency and reducing data exposure risk.

6.3.4 OCR Processing Module

The OCR Processing Module converts uploaded document images into machine-readable text using the Tesseract OCR engine through the pytesseract Python library. Extracted raw text undergoes post-processing using regular expressions to identify key fields including document identifiers and treatment type descriptors. Data normalization routines clean extracted values, standardize formats, and apply fallback extraction logic when primary patterns fail to match. The module outputs a structured data object containing extracted text, document ID, and treatment classification for downstream processing. Figure 2 illustrates the OCR processing workflow.

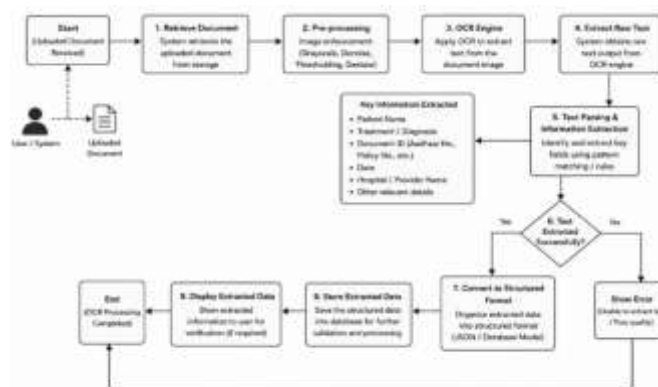


Figure 2: Workflow of the OCR Processing Module

6.3.5 Claim Validation Module

The Claim Validation Module applies a policy-driven rule set to evaluate whether a submitted claim satisfies insurance policy conditions. Policy parameters are loaded dynamically from a CSV dataset at validation time, allowing rule updates without code modification. The module evaluates the following conditions:

- Policy status verification (active/inactive).
- Treatment type eligibility against the policy's covered treatment list.
- Waiting period compliance based on policy start date and minimum waiting duration.
- Claim amount compliance against the maximum policy coverage limit.
- Age eligibility against the policy's defined minimum and maximum age bounds.

Each violated condition is recorded as a distinct issue, and a rejection flag is activated for critical violations. Figure 3 illustrates the claim validation logic.



Figure 3: Flow Diagram of the Claim Validation Module

6.3.6 Fraud Detection Module

The Fraud Detection Module analyzes the information provided by the insurance claims and uses several rule-based methods for identifying the fraud patterns. First of all, it checks for duplicate documents based on document identifier, meaning that there are some instances when the same document is submitted as an attachment for various claims. Secondly, it analyzes the cross-document consistency, meaning that it examines patient's name, treatment, etc., comparing the records provided within different documents. Thirdly, it does the missing/invalid data check, helping us to find those cases when we do not have any information regarding some of the fields in the records or the records include the data, such as NULL values, placeholders, and others. Another interesting aspect for analysis is represented by the high value claims since we focus on finding those insurance claims in which the submitted value is more than 90% of the maximum policy coverage limit, making them rather risky. Lastly, we analyze the frequent submission, focusing on counting the number of claims submitted by a particular user or document identifier.

Each detected anomaly contributes an incremental value to a cumulative fraud score. The numerical fraud score quantifies the aggregate risk level associated with a claim and is passed to the Decision Engine for final classification. Figure 4 illustrates the fraud detection logic.



Figure 4: Flow Diagram of the Fraud Detection Module

6.3.7 Decision Engine

The Decision Engine integrates the outputs of the Claim Validation Module and the Fraud Detection Module to produce the final claim decision. The decision logic follows a deterministic, threshold-based classification scheme. Table 2 presents the decision classification criteria.

Table 2: Decision Engine Classification Criteria

Condition	Fraud Score / Flag	Decision
Rejection flag triggered (policy violation)	Any	Rejected
No flag; fraud score > 50	> 50	Rejected
No flag; fraud score 21–50	21–50	Manual Review
No flag; fraud score ≤ 20	≤ 20	Approved

For every decision, the Decision Engine generates a structured explanation listing the specific policy violations or fraud indicators that influenced the outcome. This explanation is presented to the user and recorded in the audit log.

6.3.8 Audit Log Module

The Audit Log Module records all processed claim decisions to a dedicated table in the SQLite database. Each log entry includes the claim data, decision outcome, supporting explanation, and a timestamp generated via Python's datetime module. The audit log provides a complete, immutable record of system activity, enabling post-hoc traceability, compliance verification, and retrospective analysis of decision patterns.

7. Methodology

7.1 Development Methodology

The system was developed following a structured, phase-based development process comprising system analysis, system design, modular implementation, testing, and local deployment. The development methodology was iterative within each phase, with validation testing performed at each module boundary before integration into the complete pipeline.

7.2 Claim Processing Workflow

The process starts from registration and login, and after that, multi-level verification through Security PIN and OTP verification takes place for the purpose of ensuring security of access. After authentication, the user uploads the relevant documents, and the same will be processed through Optical Character Recognition (OCR). Claim details are submitted by the user, and claim validation will take place in the claim validation module according to policy rules. These rules may include things like eligibility, coverage, and claim limits. After the process of validation, the process of fraud detection starts with respect to the pattern of documents used, any data inconsistencies, and high-value claims. On the basis of validation and fraud detection results, the decision engine provides the final result in the form of approved or rejected or manual review.

Figure 5 illustrates the complete process flow diagram of the system.

Figure 5: End-to-End Process Flow Diagram of the Medical Insurance Claim Audit System

7.3 Core Algorithms

7.3.1 OCR Processing Algorithm

The OCR processing algorithm employs Tesseract as the core recognition engine. Upon receiving a document image file stream, the algorithm: (1) loads the image from the in-memory stream; (2) invokes pytesseract to

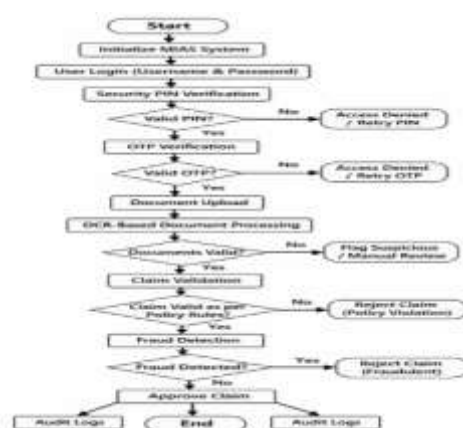
extract raw text; (3) applies a sequence of regular expression patterns to match document ID formats and recognized treatment descriptors; (4) normalizes extracted values by removing extraneous whitespace and standardizing case; and (5) executes fallback extraction logic when primary patterns fail. The algorithm returns a structured dictionary containing the raw OCR text, the extracted document ID, and the classified treatment type. Error handling mechanisms manage cases including missing OCR engine binaries, unreadable image formats, and empty OCR output.

7.3.2 Policy Loading Algorithm

The policy loading algorithm reads a CSV-formatted policy dataset at runtime. It iterates through the dataset rows to match the user-submitted policy type against available policy records. Upon finding a match, it extracts and type-converts key policy parameters: minimum and maximum age bounds, maximum claim amount, waiting period duration, and the list of covered treatment types. If no matching policy is found, the algorithm returns a null value, and the validation engine issues a policy-not-found rejection flag. This design decouples policy rules from application code, allowing rule updates through dataset modification without redeployment.

7.3.3 Claim Validation Algorithm

The claim validation algorithm receives as input the structured claim data (user details, policy parameters, OCR-extracted document data, and submitted claim



fields) and executes a sequential rule evaluation sequence. Each rule evaluation records a descriptive issue string on violation and conditionally sets a rejection flag. The algorithm also incorporates fraud scoring logic by evaluating conditions such as claim amount relative to the policy limit. Upon completing all rule checks, the algorithm returns the list of accumulated issues, the rejection flag state, and the computed fraud score for consumption by the Decision Engine.

8. Implementation

8.1 Technology Stack

The system is implemented using the following technologies:

- Python 3.8+: Primary programming language for all backend logic.
- Flask: Lightweight web framework managing routing, session handling, and template rendering.
- Tesseract OCR (via pytesseract): Engine for extracting text from uploaded document images.
- SQLite: Embedded relational database for storing user records, claim data, and audit logs.
- Python Standard Libraries: Used for regular expression processing (re), OTP/PIN generation (random), timestamp recording (datetime), and file I/O (io).
- Pytest: Testing framework used for unit test execution and code coverage analysis.
- VS Code / PyCharm: Integrated development environments used during implementation.

8.2 Implementation Details

The authentication module is implemented within the Flask application's app.py file. User registration and login routes interact with the SQLite database through parameterized SQL queries. Session tokens are managed using Flask's built-in session management to maintain authentication state across requests. The document

processing module processes files directly from the uploaded file stream, avoiding persistent disk storage. The pytesseract library is invoked with default configuration, and the resulting text string is processed through a pipeline of compiled regular expression patterns. The claim validation engine is encapsulated in claim_validator.py, which defines the rule evaluation logic as a function receiving structured input dictionaries and returning a results dictionary. The fraud detection logic is incorporated within the same module, computing the fraud score through a series of conditional checks. The decision engine is implemented as conditional logic at the tail of the validation function, consuming the rejection flag and fraud score to emit the final decision string.

8.3 Database Design

The system employs three primary SQLite tables. The Users table stores registration data including user ID (primary key), name, email, phone number, policy type, and security PIN. The Claims table stores submitted claim records including associated user ID, document identifiers, extracted OCR data, submitted claim fields, decision outcome, and decision explanation. The AuditLog table stores timestamped entries for each processed claim, linking the claim ID to the decision and the timestamp of processing. This schema supports both real-time decision queries and retrospective audit analysis.

8.4 Deployment Configuration

The system is deployed in a local execution environment. Deployment requires installation of Python dependencies (Flask, pytesseract, Pillow) via pip and installation of the Tesseract OCR binary with its path configured in the application. The Flask development server is invoked via the command python app.py, and the application is accessible at http://127.0.0.1:5000/ through a standard web browser. The system requires a minimum of 4 GB RAM and an

Intel i3 or equivalent processor. No external network connectivity or cloud services are required for operation.

9. Testing and Evaluation

9.1 Testing Approach

The system was evaluated through a combination of functional testing, validation testing, fraud scenario testing, usability evaluation, and unit testing with code coverage analysis. Functional testing verified the correctness of core operations including user registration and login, OCR extraction accuracy across varied document formats, and the correct execution of validation logic for diverse claim inputs. Fraud scenario testing presented the system with known fraudulent scenarios—duplicate document IDs, cross-document name mismatches, high-value claim submissions, and incomplete document data—and verified that the fraud detection module correctly identified each anomaly and computed an appropriately elevated fraud score. Usability testing confirmed smooth navigation across all interface pages and clear presentation of extracted data and decision outputs.

9.2 Module-Level Results

9.2.1 Registration and Login

The registration interface collected all required user details, stored them correctly in the SQLite database, and generated a unique user ID upon each successful registration. The login module correctly validated credentials against stored records and appropriately rejected invalid User ID and email combinations. Figure 6 and Figure 7 illustrate the registration and login interfaces respectively.

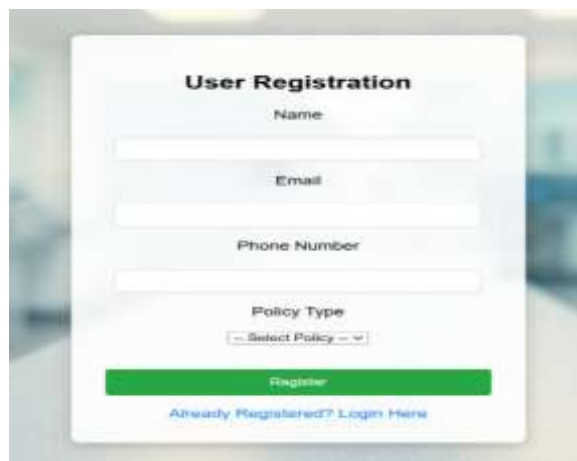
The image shows a 'User Registration' form. It has a title 'User Registration' at the top. Below the title are five input fields: 'Name', 'Email', 'Phone Number', 'Policy Type' (which is a dropdown menu with 'Select Policy' as the placeholder), and a green 'Register' button. At the bottom, there is a link that says 'Already Registered? Login Here'.

Figure 6: User Registration Interface

The image shows a 'User Login' form. It has a title 'User Login' at the top. Below the title are two input fields: 'User ID' and 'Email'. At the bottom, there is a blue 'Login' button.

Figure 7: User Login Interface

9.2.2 Document Verification and OCR Processing

The document upload interface accepted the required document types and successfully invoked the OCR processing pipeline. Extracted document identifiers and treatment descriptors were correctly identified for test document images with clearly rendered text. The panel also presented extracted data fields for user review before claim submission, enhancing transparency. Figure 8 illustrates the document verification and OCR processing interface.



Figure 8: Document Verification and OCR Processing Interface

9.2.3 Multi-Level Authentication

Security PIN verification correctly accepted valid PINs and rejected incorrect entries. OTP verification generated a unique numeric OTP for each session and correctly validated user-entered values. Both authentication steps functioned as effective access control gates, preventing progression to claim submission upon failed authentication.

9.2.4 Claim Submission and Decision Output

The claim details entry interface successfully collected all required fields and initiated the validation and fraud analysis pipeline upon submission.

The Decision Engine produced correct outcomes across three distinct test scenarios:

- Approval Scenario: A claim satisfying all policy rules and exhibiting no fraud indicators was correctly approved. Figure 9 illustrates this outcome.
- Rejection Scenario: A claim submitted with a previously used document ID was correctly identified by the duplicate detection mechanism and rejected with an explanatory message citing the fraud indicator. Figure 10 illustrates this outcome.
- Manual Review Scenario: A claim with a high claim amount contributing a fraud score above the review threshold but below the rejection threshold was correctly flagged for manual review. Figure 11 illustrates this outcome.

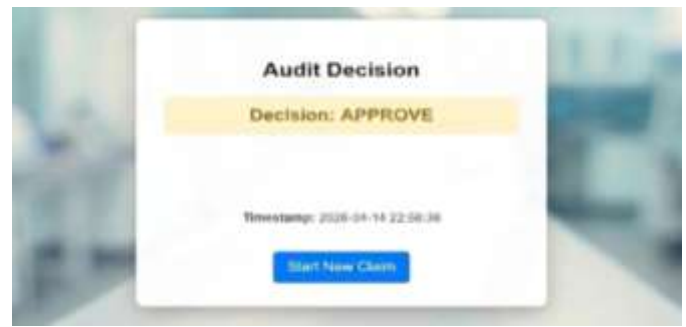


Figure 9: Audit Decision Interface Showing Approved Claim

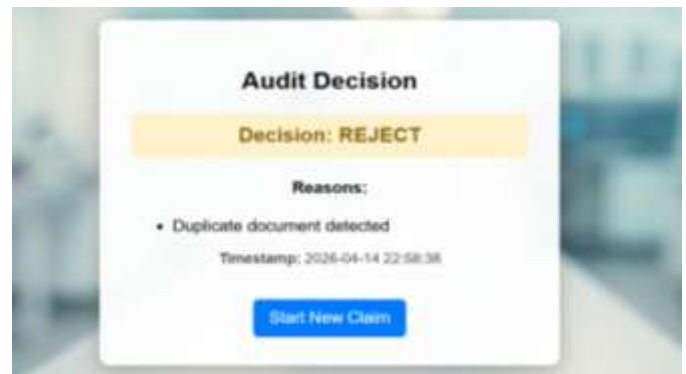


Figure 10: Audit Decision Interface Showing Rejected Claim

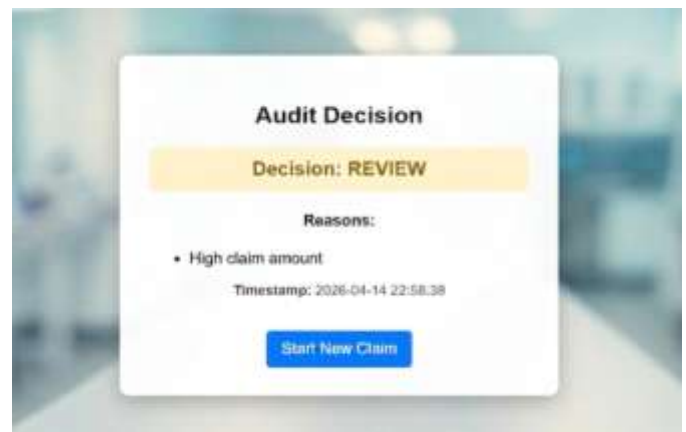


Figure 11: Audit Decision Interface Showing Manual Review Claim

9.3 Unit Testing and Code Coverage

Unit testing was conducted using the Pytest framework. A total of 100 test cases were executed, covering valid inputs, invalid inputs, boundary conditions, and edge cases across all major modules. All 100 test cases passed without failures. Code coverage analysis was performed to assess the proportion of the codebase exercised during testing.

The overall system achieved 87% code coverage, with the claim validation engine and policy loading module

achieving near-complete coverage. OCR processing coverage was constrained by the conditional handling of OCR engine availability and varied image formats during automated testing. Figure 12 illustrates the Pytest output and coverage report.

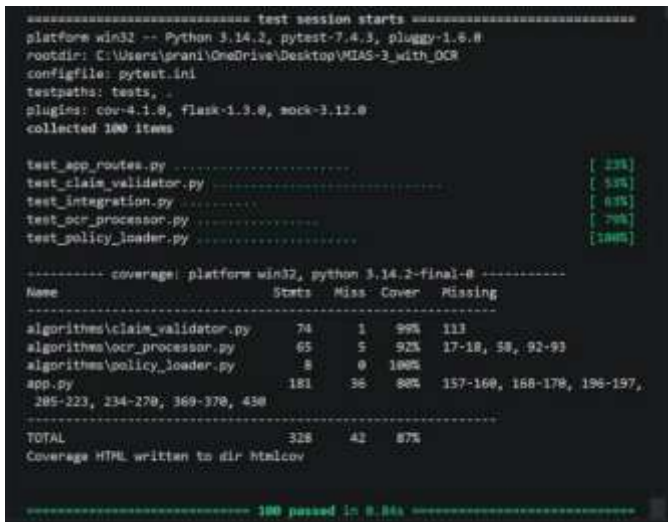


Figure 12: Unit Testing Results and Code Coverage Report (Pytest)

10. Comparative Discussion

Table 3 compares present system against existing enterprise claim systems and academic ML/GNN-based approaches across key system features.

Table 3: Feature Comparison of the system against Existing Approaches

Feature	Existing Enterprise Systems	ML/GNN Approaches	Proposed System
OCR-based document processing	Limited	Not addressed	Fully integrated
Rule-based claim validation	Partial (structured only)	Absent	Dynamic policy-driven
Multi-level authentication	Basic login	N/A	Login + PIN + OTP
Fraud detection mechanism	Limited / manual	ML classifiers	Rule-based scoring

Automated decision generation	Partial	Model-dependent	Deterministic (3-class)
Audit logging	Enterprise-grade	Not addressed	Timestamped SQLite
Lightweight deployment	No (complex infra)	No (GPU/cloud)	Local, standard hardware
Open / interpretable logic	No	Often opaque	Fully traceable

Enterprise systems such as those offered by Optum and Change Healthcare provide robust infrastructure for structured claim processing but do not natively integrate OCR-based document extraction, limiting their ability to handle heterogeneous, unstructured documents without preprocessing pipelines. ML and GNN-based approaches achieve strong fraud detection performance under research conditions but introduce opacity in decision logic and require substantial data and computational infrastructure, making them unsuitable for lightweight deployment contexts.

This system occupies a distinctive position by combining the interpretability and consistency of rule-based systems with document-level processing capabilities and multi-criteria fraud scoring. Unlike purely document-focused systems, the system extends extraction to full claim validation and decision generation. Unlike pure audit log systems, the system acts on extracted data in real time rather than analyzing historical patterns. The deterministic, threshold-based Decision Engine ensures that every decision is explainable and auditable—a property that is difficult to guarantee in ML-based classifiers.

10. CONCLUSION

This paper has presented the Medical Insurance Claim Audit System, a web-based application that automates the complete medical insurance claim verification lifecycle through an integrated pipeline of OCR-based

document processing, policy-driven rule validation, multi-criteria fraud detection, three-factor authentication, and timestamped audit logging. The system addresses a clear gap in the existing literature and tooling landscape by providing a unified, interpretable, and locally deployable claim auditing solution that does not require large-scale datasets, ML training pipelines, or cloud infrastructure.

The implemented system correctly processes all three targeted document types, applies dynamic policy rules loaded from a dataset, assigns quantitative fraud scores across five anomaly categories, and produces deterministic claim decisions with structured justifications. Unit testing with Pytest confirmed correct system behavior across 100 test cases with 87% overall code coverage. The system effectively demonstrates that the integration of proven technologies—OCR, rule-based engines, and relational databases—within a coherent modular architecture can deliver practical claim auditing capability suitable for small to medium insurance processing contexts.

Future enhancements may include: integration of machine learning classifiers to extend fraud detection beyond predefined rules; adoption of advanced OCR APIs to improve extraction accuracy on low-quality documents; migration to scalable database systems such as MySQL or PostgreSQL for production deployment; cloud-based deployment to enable concurrent access and improved availability; mobile application development for enhanced accessibility; implementation of encryption and role-based access control for strengthened security; and integration with hospital and insurance provider databases for real-time cross-verification of submitted claims.

REFERENCES

1. M. Muhammad et al., "Fraud Detection in Medical Insurance Claims Using Graph Neural Networks," arXiv preprint, 2025.
2. S. du Preez et al., "Machine Learning Techniques for Healthcare Claim Fraud Detection: A Systematic Review," *Journal of Healthcare Informatics*, 2024.
3. P. Arora, "Intelligent Document Management System for Health Insurance Digitization," *International Journal of Computer Applications*, 2024.
4. J. Rule et al., "Analysis of Electronic Health Record Event Logs for Workflow Optimization," *Journal of Biomedical Informatics*, vol. 140, 2023.
5. K. Kannampallil et al., "Exploring Electronic Health Record Audit Log Data for System Analysis," *Journal of the American Medical Informatics Association*, vol. 29, no. 4, 2022.
6. S. Huilgol et al., "Improving Healthcare Quality Using Audit Log Analysis," *Healthcare Informatics Research*, vol. 28, no. 2, 2022.
7. X. Li and Y. Chen, "Audit Supervision and Risk Control in Medical Insurance Fund Management," *International Journal of Health Policy and Management*, 2020.
8. Joint Learning Network, "Medical Audit System Strengthening Toolkit," JLN Publication, 2019.
9. U. Abdullah, "Data Mining Driven Rule Based Expert System for Medical Billing Compliance," 2015.
10. H. Park, "Health Insurance Claim Review and Assessment Using Information Technology," *Health Policy and Technology*, vol. 1, no. 2, 2012.
11. J. Wojtusiak et al., "Rule-Based Prediction of Medical Claim Payments Using Medicaid Data," *Artificial Intelligence in Medicine*, vol. 51, no. 1, 2011.
12. Y. Wang and Z. Zhang, "Data Audit Approach for Medical Insurance Systems," *Journal of Systems and Software*, vol. 83, no. 5, 2010.