Archives available at journals.mriindia.com

International Journal on Advanced Computer Engineering and Communication Technology

ISSN: 2278-5140

Volume 15 Issue 01, 2026

Citizen-Led AI Audit Platform for Transparency and Accountability in Automated Decision-Making

[1]Pradeep Arun Patil, [2]Rutuja Sunil Jadhav, [3]Prajakta Jagtap, [4]Kartik Dhanaji Thorat, [5]Hrishikesh Kamlakar patil

[1]Prof, Computer Engineering, Sandip Institute Of Technology and Research Centre Nashik(SITRC)

[2,3,4,5] B.E – Student, Computer Engineering, Sandip Institute Of Technology and Research Centre Nashik(SITRC)

Email: [1]mail2pradippatil@gmail.com, [2]rutujaj2003@gmail.com, [3]prajaktajagtap2004@gmail.com, [4]kartikthorat011@gmail.com, [5]hrishipatil193@gmail.com

Peer Review Information

Submission: 25 Jan 2026

Revision: 12 Feb 2026

Acceptance: 26 Feb 2026

Keywords

Responsible AI, Algorithmic Transparency, Citizen-Led Auditing, Ethical AI Governance, Bias Detection

Abstract

Artificial Intelligence (AI) and automated decision-making systems are increasingly embedded in critical areas of governance such as housing allocation, welfare distribution, recruitment, healthcare, and immigration. While these systems promise efficiency and scalability, they often operate as opaque "black boxes," producing decisions that lack explainability or recourse for affected citizens. This opacity undermines public trust and accountability in digital governance.

This review paper examines global efforts toward Responsible AI and highlights the urgent need for citizen-led auditing mechanisms that operationalize fairness, transparency, and accountability in practice. Drawing insights from recent literature on algorithmic transparency, fairness auditing, and privacy-preserving governance frameworks, the paper identifies key gaps—namely the absence of citizen-sourced evidence pipelines, cross-domain bias mapping, and measurable audit effectiveness. A conceptual framework and layered functional architecture are proposed to integrate citizen reporting, NLP-based anonymization, structured metadata storage, and visualization dashboards for systemic bias detection. The study bridges theoretical Responsible-AI principles with practical citizen-centric accountability models, offering a scalable foundation for participatory and ethical AI governance.

Introduction

Artificial Intelligence and automated decision-making systems are increasingly integrated into digital platforms that support governance, public services, and large-scale administrative processes. Modern software systems are designed to process high volumes of applications, evaluate eligibility criteria, classify user profiles, and generate decisions with minimal human intervention. These intelligent systems rely on structured data processing, rule-based logic, and machine learning models to improve efficiency, reduce manual workload, and enable scalable service delivery.

1. Background and Context

Traditionally, decision-making in governance and administrative systems was performed manually by officials following documented procedures. Records were maintained in physical files or basic digital databases, and decisions could be explained by referencing predefined rules or documented evaluation criteria. Although manual systems were time-consuming, they

provided a degree of interpretability and human oversight.
With the advancement of digital infrastructure, many institutions transitioned to centralized software platforms for application processing and eligibility verification. Modern systems typically operate using multi-layered architectures that include:

- A user interface layer for data submission
- A processing layer for validation and rule execution
- A database layer for structured storage
- Analytical components for reporting and monitoring

2. Problem Definition

The primary problem addressed in this project is the absence of a structured, auditable, and traceable software framework for collecting, processing, and analyzing outcomes generated by automated decision-making systems.
Existing systems focus primarily on efficiency and scalability, but they do not provide:

- A standardized mechanism for recording user experiences
- Structured metadata extraction from unstructured inputs
- Integrated anonymization workflows for sensitive data
- Controlled taxonomy-based classification
- Aggregated visualization for pattern identification
- Comprehensive audit logging across system events.

3. Expected Outcome and Impact

The expected outcome of this project is a functional working prototype that demonstrates a structured and auditable workflow for processing and analyzing automated decision-related records.
The system is expected to provide:

- A modular and traceable software architecture
- Improved data integrity through structured storage and logging
- Privacy-aware data handling mechanisms
- Consistent taxonomy-based classification
- Aggregated visualization of structured records
- A clear separation between processing, validation, and analytics layers.

Literature Survey

The purpose of this literature survey is to study existing systems, architectural models, and technological approaches relevant to the problem domain addressed in this project. Various conventional and advanced software systems were examined to understand how they manage data processing, automation, monitoring, and accountability. The review focuses on identifying strengths, limitations, and areas where improvement is required. This analysis helps in understanding how current solutions partially address the problem and where structured redesign or integration is necessary. The insights gained from this study form the foundation for identifying research gaps and designing the proposed system architecture presented in the next chapter.

1. Overview of Existing Systems

A) Traditional Systems in the Domain:
Traditional systems in many software domains operate using centralized architectures. These systems typically include a user interface for data entry, a backend processing unit, and a centralized database for storage. Workflows are generally linear: user input is validated, processed based on predefined rules, and stored in the system.

B) Secure or Optimized System Variants:
To overcome performance and security limitations, optimized system variants have been developed. These systems integrate additional features such as encryption mechanisms, automated validation tools, monitoring modules, and rule-based or intelligent processing algorithms.

C) Distributed or Advanced Architecture-Based Systems:
Modern systems increasingly adopt distributed architectures or modular design frameworks. These systems separate responsibilities into independent modules such as data ingestion, processing, analytics, and visualization layers. Some systems use scalable architectures to handle high data volume and concurrent users.

D) Logging, Monitoring, and Audit Systems:
Many software platforms incorporate logging and monitoring mechanisms to track system activities. These systems record events such as user actions, system errors, and database transactions. Monitoring tools may provide dashboards to visualize operational metrics.

System Design

1. System Architecture and Diagrams

System Architecture

This figure presents the layered architecture of the Citizen-Led AI Audit Platform. It clarifies system boundaries, principal actors, major subsystems (UI, anonymization/NLP, storage,

analytics, validation, visualization), and the cross-cutting Audit Log & Taxonomy Manager. An architectural view is used to communicate decomposition (what the system is made of), allocation of responsibilities, and high-level data/control flow before drilling down into process detail.

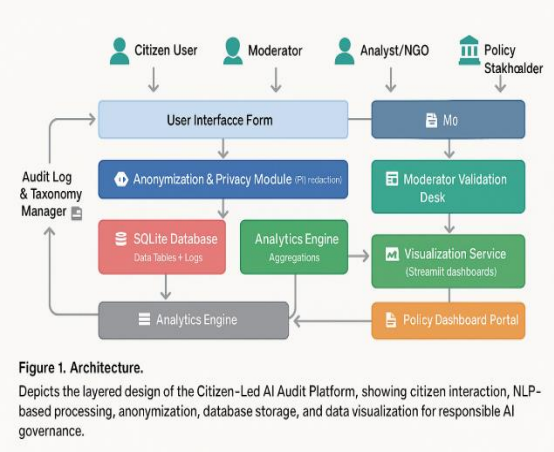


Figure 1. System Architecture.

Component-wise Explanation:

- Actors (Citizen User, Moderator, Analyst/NGO, Policy Stakeholder): External roles that interact with the platform through the UI, validation desk, dashboards, and policy portal.
- User Interface Form (UI Layer): Entry point for case submission and structured metadata. Enforces input validation, consent capture, and rate-limiting.
- Anonymization & Privacy Module: Performs PII detection and redaction on narrative text and attachments. Includes rule-based filters (regex, dictionaries) and ML NER models. Produces a redaction log.
- Moderator Validation Desk: A gated interface where moderators review redactions, approve/return cases, and curate taxonomy mappings.
- SQLite Database (Data Tables + Logs): System of record for cases, metadata, attachments, and audit trails. Chosen for embedded deployment and ACID semantics.
- Analytics Engine (Aggregations): Computes descriptive and exploratory analytics—counts, frequencies, co-occurrence, geospatial buckets, trend lines—feeding visualizations and the policy portal.
- Visualization Service (e.g., Streamlit dashboards): Renders charts/maps/tables and exposes filterable, role-aware views.
- Policy Dashboard Portal: Curated, export-ready views for decision makers, including metrics, heatmaps, and downloadable reports.
- Audit Log & Taxonomy Manager (cross-cutting): Centralizes event logging (create/approve/export), maintains controlled vocabularies (category, sub-type, reason, synonyms), and governs evolution of labels used across the stack.

Data/Interaction Flow:

1. Submission: Actors use the UI to submit narrative + metadata.
2. Privacy Processing: The submission flows to Anonymization & Privacy; PII is detected/redacted, a redaction log is written, and a safe record is produced.
3. Persistence: The redacted record and logs are stored in SQLite.
4. Moderation: The Validation Desk retrieves candidate records and the current taxonomy, enabling approve/reject and label corrections; decisions are persisted.
5. Analytics: Approved data is aggregated by the Analytics Engine (counts, topic frequencies, geo/time slices).
6. Visualization & Policy: Aggregates feed the Visualization Service and Policy Portal; stakeholders browse, filter, and export.
7. Governance: All significant events plus taxonomy updates route through Audit Log & Taxonomy Manager for traceability and consistency.

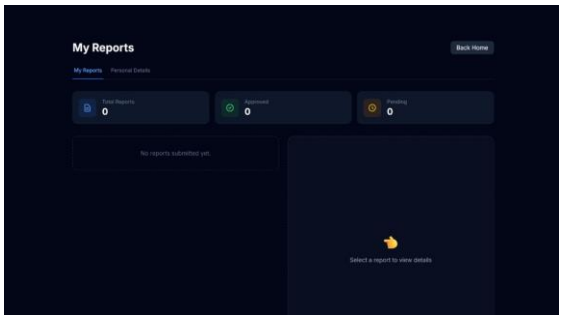
Theoretical Foundation

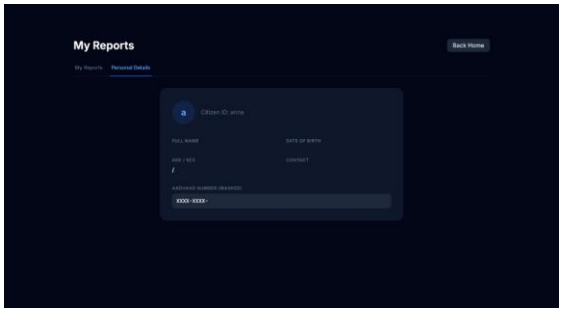
- ETL pipeline: Ingest (UI) → Transform (anonymize, extract metadata, map to taxonomy) → Load (SQLite) → Analyze (aggregations) → Present (dashboards).
- NLP/Privacy: Rule-based + NER for PII; metadata extraction via tokenization, POS/NER, and pattern mining; taxonomy mapping as controlled-vocabulary normalization.
- Data Governance: Immutable audit logs (append-only), provenance, and role-based access control.
- Analytics/BI: OLAP-like aggregation over denormalized views; visualization grounded in analytic tasks (overview, filter, zoom, detail-on-demand).
- Software Architecture: Layered style, separation of concerns, and cross-cutting concerns (logging/governance) via a dedicated manager.

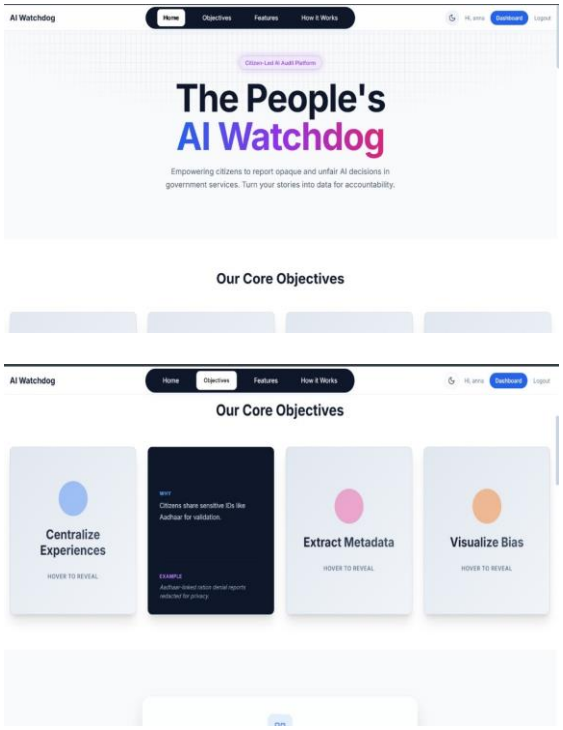
Context in the Project

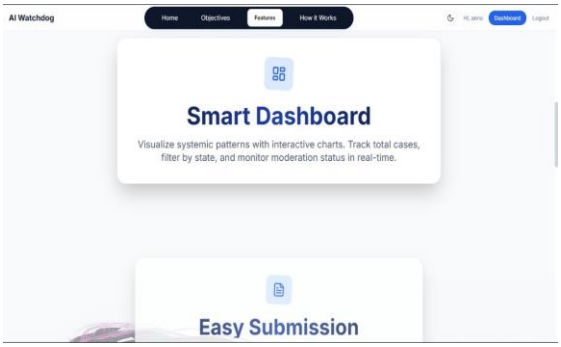
The architecture situates every subsequent diagram: DFDs detail its flows, the ER diagram specifies its persistence, the class diagram encodes its abstractions, and sequence/activity diagrams capture dynamic behavior across these components.

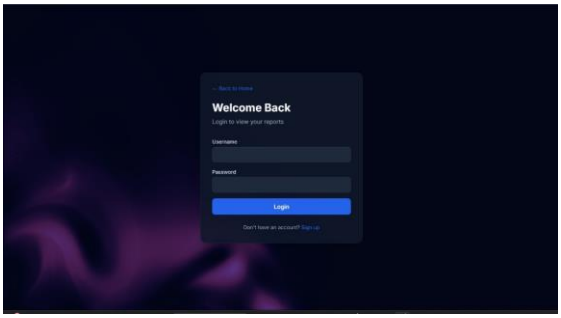
Screenshots











Conclusion

The architecture defines a privacy-first, auditable pipeline that transforms sensitive citizen input into governance-grade analytics. It emphasizes modularity, accountability, and traceability—key quality attributes for an AI audit platform in the SDLC's inception and elaboration phases.

Technology and Tools Overview

1. Frontend

The frontend relies on several key packages to build a modern web interface. It uses Next.js and React to structure pages and create interactive UI components, and React-DOM to render those components in the browser. Axios handles communication with the backend, while Framer Motion adds animations and Lucide React provides icons. Libraries like Recharts and Three.js are used for charts and 3D graphics, and Tailwind CSS is used for styling. Next-themes manages light/dark themes, and TypeScript helps catch errors early by adding types to the code. All dependencies are listed in a JSON file, which makes it easy to install them with npm (Node Package Manager).

For the environment, the project requires Python 3.8 or newer to run backend services, and Node.js 18 or higher along with npm for installing frontend packages and starting the development setup. A version control tool like Git is also needed to track and manage changes in the codebase.

2. Backend

The backend uses several Python libraries to build and run the API. FastAPI is the main

framework for creating API endpoints quickly and efficiently, and Uvicorn is the server that runs the FastAPI application. python-multipart allows the backend to handle file uploads, while pandas is used to work with and analyze data. scikit-learn provides machine-learning tools, and plotly is used for creating visual charts. pytest is included for testing the code, and passlib with bcrypt ensures secure password hashing. python-jose is used for handling and verifying JSON Web Tokens (JWT) for authentication, and tqdm adds progress bars for command-line tasks.

3. Database or Storage Mechanism

The project uses SQLite as the main database because it is lightweight and easy to set up. The database is configured to use Write-Ahead Logging (WAL), which speeds up writes and lets multiple read operations happen at the same time without blocking each other, improving performance and concurrency compared to the default rollback mode. Two tables are defined in the schema: one for storing user information like usernames, hashed passwords, roles and personal details, and another for storing audit reports with fields such as category, location, description, status, timestamps, and associated user ID. Files related to the application are organized into folders, with the SQLite database file (reports.db), trained machine-learning model (model.pkl), and logs stored under the backend, while static assets for the frontend are in their own folder. Data like AI models are saved using pickle serialization, user sessions are managed with JWT tokens, file uploads are temporarily processed in memory, and logs are rotated with timestamps to keep track of activity over time.

4. Development Platform and Environment setup

The development environment includes separate setups for backend and frontend. For the backend, a Python virtual environment is created, activated, and all required packages are installed. For the frontend, npm installs all the necessary JavaScript packages and starts the development server. Common tools include Visual Studio Code with extensions for Python and TypeScript, formatting tools like Black and Prettier, and linters such as ESLint and Pylint to keep code clean and consistent. Git is used for version control, and environment variables are configured to store important settings like the database URL, secret keys, API base URL, and runtime mode.

5. Programming Languages and Frameworks

The backend is built with Python using FastAPI, a modern web framework that is fast, supports asynchronous (non-blocking) code, and automatically generates API documentation from the code itself. FastAPI also integrates well with Python tools for data processing and machine learning, making development efficient and reliable. For the frontend, Next.js with TypeScript is used because Next.js enhances React with features like server-side rendering and optimized loading, while TypeScript adds type safety that helps catch errors early and keeps code more maintainable. Additionally, libraries like Pandas and Scikit-learn are used for data manipulation and machine learning tasks, as they offer powerful, easy-to-use tools for working with structured data and building predictive models.

Conclusion

The platform uses SQLite as its main database with Write-Ahead Logging (WAL) enabled to improve performance and support better concurrency than the default rollback mode. In WAL mode, writes are appended to a separate log file, and readers can access the database at the same time without blocking each other, which results in faster operations and smoother access for multiple users.

The database schema includes two main tables. The users table stores credentials and profile information such as username, hashed password, role, contact details, and personal data. The reports table captures audit entries with fields like category, location, tags, status, description, timestamps, and the user who submitted the report.

The project's file structure stores the SQLite database file (reports.db), the trained AI model (model.pkl), and application logs in the backend folder, while frontend static files are in their own directory. AI models are persisted using Pickle serialization, user sessions are managed with JWT tokens stored in local storage, file uploads are processed in memory without saving to disk, and logs are kept in rotating files with timestamps to track activity.

References

Budhewar AS, Hatkar SS. Visual Cryptography Identity Specification Scheme. International Journal of Computer Sciences and Engineering. 2019;7(4):1148-52.

Budhewar AS, Patil PG, Kale SM. Neighbour-Aware Cooperation For Semi-Supervised Decentralized Machine Learning. Educational Administration: Theory and Practice. 2024;30:2039-47.

Jagneet AG, Budhewar AS, Kadam GR, Jadhav NU, Loharkar KB, Mohabe YR. Face Recognition-

Based Attendance System Using Group Photos. International Journal. 2024 Sep;8(9).

Budhewar AS, Patil PG, Patel MJ, Roy AV, Phapale NE, Rathod DS. SECURE CARE HUB: A BLOCKCHAIN-ENABLED PLATFORM FOR STREAMLINED HEALTHCARE SERVICES. INTERNATIONAL JOURNAL. 2024 Sep;8(9).

Budhewar PG, Patil1 Anmol S. RANSOMWARE DETECTION AND CLASSIFICATION USING MACHINE LEARNING. INTERNATIONAL JOURNAL. 2024 Sep;8(9).

Atti, Hima Bindu, et al. "Integrating Multi-Omics Data Using AI: Foundations for Systems Pharmacology and Precision Care." *Revolutionizing Drug Research and Personalized Medicine Through AI and Machine Learning,* edited by Adinarayana Andy, et al., IGI Global Scientific Publishing, 2026, pp. 397-430. https://doi.org/10.4018/979-8-3373-6400-1.ch014

Veernapu, K. K., Patil, B. K., Tharayil, A. S., Sindhura, C., Andy, A., Budhewar, A., Somwanshi, S. K., Kavya, K., Bharadwaj, M., & Raj, P. (2026). Real-time AI in Clinical Decision Support: Bridging Research and Practice. In A. Andy, A. Tharayil, & A. Budhewar (Eds.), *Revolutionizing Drug Research and Personalized Medicine Through AI and Machine Learning* (pp. 431-452). IGI Global Scientific Publishing. https://doi.org/10.4018/979-8-3373-6400-1.ch015

S. D. Veeravalli, P. A. Patil, T. Porwal, V. S. Karwande, A. S. Budhewar and B. M. Nanche, "Adaptive-Personalised Federated Deep Learning for Privacy-Aware NAFLD Screening," *2025 International Conference on Innovations in Intelligent Systems: Advancements in Computing, Communication, and Cybersecurity (ISAC3)*, Bhubaneswar, India, 2025, pp. 1-7, doi: 10.1109/ISAC364032.2025.11156569.