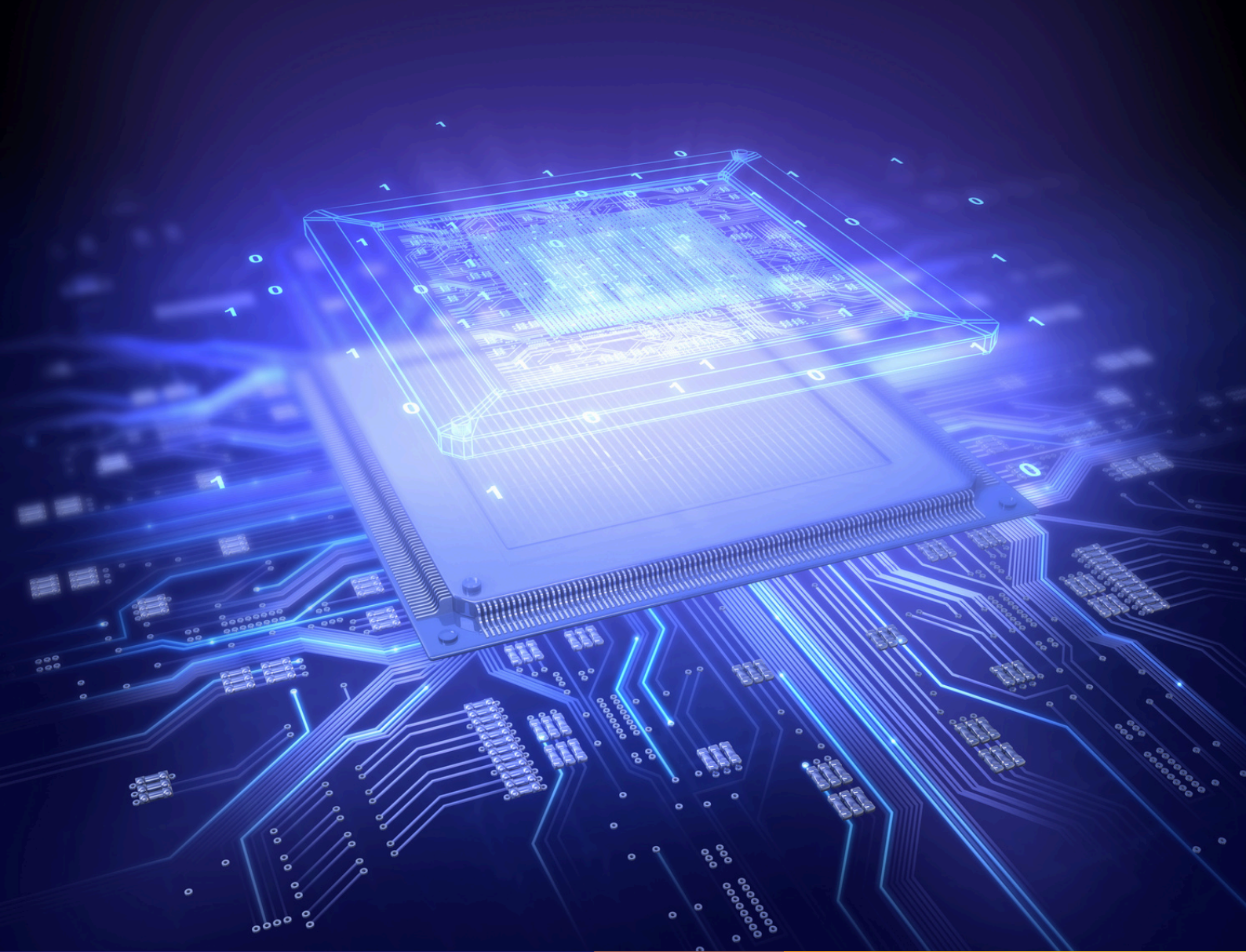




BS (CS) FINAL YEAR PROJECTS

CATALOGUE

(CLASS OF 2026)



Preface

This catalogue presents the Final Year Projects of BS(CS) students for the academic year 2025–2026. The projects reflect the innovation, technical expertise, and research capabilities of our students under the guidance of faculty members and industry mentors.

Dr. Shakeel Khoja

Dean, School of Mathematics
& Computer Science (SMCS)

Table of Contents

Enhanced SIEM with AI-Driven Security Analytics and Natural Language Querying	01
Memorable.io – Optimization and Recommendation Framework for Memorable Advertising	02
RAHAT-ONE: Intelligent HR Automation and Orchestration Platform	03
Branch Performance Analytics Dashboard with Predictive Footfall Forecasting	04
Visual QA	05
Active-Ledger: Trust-Gated Diffusion and Behavioural Contribution Scoring for Adversarially Robust Federated Learning on Clinical IoT Non-IID Data	06
AI Based Cost Management System	07
Awaaz se Alfaaz: Enhancing Urdu ASR through Whisper Fine-Tuning and SLM-Based Post-Processing	08
Vocalytics	09
AI Agent-Orchestrated Digital Shadow Framework for Multilingual Healthcare Workflow Automation	10
PrivCap: Benchmarking VLMs on Cross-Modal Privacy Amplification	11
AI Workflow Automator	13
Smart Payment Reconciliation System for PayPro	14
BlockLease	15
Autonomous agentic pentesting	16
AstroRoute: A Real-Time Campus Bus Tracking and Route Management System	17
Reimagining University Enrollment: A Preference-Driven Course Allocation Framework for IBA	18
Clinical Copilot Using Knowledge Graphs	20
4see: Foresee your seats before booking.	22
MatchIQ	23

Table of Contents

Enhancing Wazuh SIEM with STIX/TAXII Threat Intelligence and Machine Learning Predictive Analytics	25
Padel Pro	27
AI driven road Quality Assessment in Rural Areas using Satellite Images	29
Benchmarking VQE: From Pipeline Analysis to Adaptive Ansatz Optimization	31
Organizational Analyst Bot	32
Automated Morphological Analysis and Classification of Neutrophils in Blood Smears	34
Intelligent Document Processing System	36
Visual Question Answering (with multimodal explanations) for Gastrointestinal Imaging	38
Legal Sahara: Legal AI Assistant	40
WazuhBot	42
AI Talent Matching Platform	44
Machine Learning Based Cancer-Risk Prediction Using Structural and Sequential Data	46
Industry Project (with Alchemy Technologies)	48
AI-Powered eKYC	49
INTELLIQUERY: AI-powered Conversational Business Intelligence Dashboard Platform	50
FedAdaPriv-CPU: Adaptive Differential Privacy with Frozen Backbone for Resource Constrained Federated Learning in Medical Imaging	52
Recallable	53
Financial News Sentiment Analyzer for Market Intelligence	54
Researching Face-Voice Association in Multilingual Contexts	55
IBA Exam Scheduler	56
Cardiac Digital Shadow	57

Enhanced SIEM with AI-Driven Security Analytics and Natural Language Querying

Group Members

Shazain (27115)
Syed Shayan Hussain (26289)
Fatima Shahid (26905)

Supervisors

Supervisor (PI): Dr. Faisal Iradat
Industry Mentor: Nasiruddin Mahmood
Industry Company: folio3

Knowledge Area: Cybersecurity

Project Overview

This project enhances the open-source Wazuh Security Information and Event Management (SIEM) platform with AI-driven capabilities to address the cybersecurity challenges faced by Pakistani organisations, particularly universities and small-to-medium enterprises. The system extends Wazuh through six independently deployable components: a conversational interface powered by a large language model that enables natural-language threat analysis directly within the security dashboard, a Natural Language Query (NLQ) search plugin that translates plain-English queries into Wazuh DSL for alert investigation, a real-time network graph visualisation showing agent connectivity coloured by alert severity, a unified compliance dashboard covering PCI DSS, HIPAA, GDPR, NIST 800-53, TSC, and PECA 2016 simultaneously, custom PECA 2016 detection rules mapping security events to Pakistan's Prevention of Electronic Crimes Act, and a bilingual Urdu/English localisation toolbar with dark-mode support. The entire system is packaged into a single automated installation script that deploys all components on a clean Ubuntu environment, making enterprise-grade AI-enhanced security monitoring accessible to resource-constrained organisations without requiring deep cybersecurity expertise.

Technologies Used

Wazuh SIEM, OpenSearch Dashboards, OpenSearch ML Commons, Python, FastAPI, LangChain, LangChain-MCP-Adapters, Model Context Protocol (MCP), opensearch-mcp-server-py, Google Gemini API, Groq API, OpenAI API, AWS Bedrock, JavaScript, Node.js, webpack 5, D3.js, React, Ubuntu 24.04 LTS, systemd, Docker, AWS EC2, Bash, npm, HTTPS/TLS, REST API

Methodology

The project followed a modular, feature-driven development approach. Each component was developed and tested independently before integration into the main system. Development began with an analysis of the existing Wazuh architecture to identify integration points for each feature. Custom OpenSearch Dashboards plugins were built using vanilla JavaScript and webpack 5 to avoid dependency conflicts with Wazuh's internal React environment. The conversational interface was implemented using the MCP architecture — a three-component pipeline consisting of an OpenSearch MCP Server, an MCP-LLM Gateway, and OpenSearch ML Commons — allowing the LLM to query live Wazuh indices through structured tool calls rather than direct index access. The NLQ Search pipeline uses a deterministic Sec-IR intermediate representation layer between the LLM and the Wazuh DSL transpiler, reducing hallucination in generated queries through schema validation and self-correction. All features were first tested locally on a Linux Mint VM, then validated on a clean Wazuh Docker container to confirm reproducibility, before final deployment to AWS EC2. A single master installation script with feature flags was written to automate the entire setup process across all six components.

Specifications

Enhanced Wazuh SIEM is built on Wazuh 4.14.3 (Manager, Indexer, and Dashboard — all-in-one) running on Ubuntu 24.04 LTS, deployed to an AWS EC2 instance for the demo. The Wazuh Dashboard runs on OpenSearch Dashboards 2.19.4 and is accessible over HTTPS on port 443, while the Wazuh Indexer listens internally on port 9200 and the Wazuh API on port 55000. The AI assistant component consists of two Python services: the OpenSearch MCP Server (opensearch-mcp-server-py 0.8.0) running on port 9900, and the MCP-LLM Gateway built with FastAPI 0.128.0 and LangChain 0.3.27 running on port 9912 — both managed via systemd. The gateway supports Groq, Google Gemini, OpenAI GPT-4o, and AWS Bedrock (Claude) as interchangeable LLM backends. All custom Dashboard plugins were built with webpack 5 on Node.js 18.19.0. The minimum recommended hardware is 4 vCPU, 8 GB RAM, and 30 GB storage.

Keywords

Security Information and Event Management, English to Query Language Translation, Localized Regulatory Alignment

Memorable.io - Optimization and Recommendation Framework for Memorable Advertising

Group Members

Mahnoor Adeel (26913)
Kisa Fatima (27076)
Muhammad Ibrahim Ayoubi (26269)

Supervisors

Supervisor (PI): Dr. Atif Tahir
Industry Advisor: Umer Murtaza
Industry Company: Folio3
Industry Mentor: Ibadullah Shaikh

Knowledge Area: Artificial Intelligence based systems

Project Overview

The project presents a multimodal advertisement intelligence system that analyzes and predicts ad memorability using visual, audio, and textual representations extracted from video content. It leverages deep learning-based embeddings and modality-specific models to generate a unified memorability score along with interpretable contributions from each modality. In parallel, the system uses a large language model to generate and refine advertisement ideas based on retrieved similar ads through a retrieval-augmented generation approach, enabling creative ad synthesis grounded in real examples. Its objective is to bridge the gap between creative content generation and measurable audience recall by integrating deep learning-based memorability prediction with retrieval-augmented large language model generation.

Technologies Used

Python, PyTorch, Scikit-learn, SentenceTransformers (all-mpnet-base-v2), OpenAI Whisper, Google Gemini API, pgvector, PostgreSQL (Supabase), FastAPI, NumPy, Pandas, OpenCV, VGGish, ResNeXt, BERT, Hugging Face Transformers, React

Specifications

The system is implemented in Python and runs on a standard machine with an Intel i5 (or equivalent), 8-16 GB RAM, with optional NVIDIA GPU support for faster model inference and embedding extraction.

Methodology

The project follows a multimodal machine learning pipeline for advertisement memorability analysis and creative generation. The system processes video advertisements by extracting visual, audio, and textual information using specialized models and feature extractors. Visual features are obtained through deep convolutional architectures such as ResNeXt and BERT-based embeddings for semantic understanding of scene-level descriptions, while audio features are extracted using VGGish and Whisper for speech-to-text transcription and acoustic representation. These heterogeneous features are then encoded into dense embeddings and used to train separate regression models, along with traditional machine learning models such as XGBoost and Random Forest, to predict modality-specific memorability scores. To unify these signals, a weighted fusion mechanism combines individual modality predictions into a final memorability score, while also providing interpretability through modality-level contribution analysis.

In parallel, a retrieval-augmented generation (RAG) pipeline is built using MPNet-based embeddings stored in a vector database (pgvector), enabling the system to retrieve similar high-performing advertisements as contextual grounding. A large language model (Gemini) then generates new advertisement scripts based on user input and retrieved examples, producing structured, scene-wise ad narratives. This combined approach enables both analytical evaluation of advertisement effectiveness and data-driven creative generation within a single integrated framework.

The system is deployed using a full-stack architecture where the frontend is built in React to provide an interactive interface for users to input prompts, view generated advertisements, and analyze memorability predictions and explanations. The backend is developed using FastAPI, which serves as the core orchestration layer connecting all components of the system, including feature extraction modules, machine learning models, retrieval-augmented generation pipeline, and the database layer. FastAPI handles API requests for ad generation and prediction, manages communication with the LLM (Gemini), and integrates retrieval from the vector database (pgvector).

Keywords

Advertisement Analytics, Multimodal Learning, Ad Memorability Prediction

RAHAT-ONE: Intelligent HR Automation and Orchestration Platform

Group Members

Zuhair Farhan (27100)
Sahil Kumar (27149)
Syed Muhammad Hussain (26992)

Supervisors

Supervisor (PI): Saiyed Shahab Ahmed
Co-PI: Adil Saleem

Knowledge Area: Software Development

Project Overview

RAHAT-ONE is an HR orchestration platform built for the clothing brand Outfitters Stores Pvt Ltd. The system we have built is placed as a middle layer between Odoo, an open source HRMS platform, and the actual employee and HR portals. This layer automates five separate business processes, including employee onboarding, leave management, expense reimbursement, attendance and branch management, and general employee and manager assignments. In addition, RAHAT-ONE also includes a fraud detection pipeline, hosted on Modal, that is paired with the automated expense reimbursement module. The fraud pipeline consists of three primary steps, namely MD5 hash deduplication, OCR-based total amount verification via Chandra-OCR vision model, and statistical anomaly detection against the employee's claim history. All the workflows are orchestrated via the n8n open-source platform, and the automated processes retain HITL (Human-In-The-Loop) where necessary, in addition to error paths in case some flows fail to run. Each step in all flows is also audited and stored in an independent PostgreSQL table hosted on Neon. The HR staff and employees interact with the system through a purpose-built React portal.

Technologies Used

Python, React, Material-UI, Node.js, Express.js, n8n, Odoo, Modal, Neon, Microsoft Azure, Docker, Railway, Vercel, Git

Specifications

RAHAT-ONE uses Odoo 17 (community edition) hosted on a Microsoft Azure VM, which is the HRMS and primary data store and record. The frontend was made with React and deployed on Vercel, whereas the backend was made with Node.js and Express.js, being hosted on Railway, with the n8n automation workflows also being hosted via cloud. Our fraud detection pipeline is hosted and run on Modal, and our Audit PostgreSQL table is hosted on Neon. Thus, RAHAT-ONE is entirely cloud-based with no local hardware required.

Methodology

Development for RAHAT-ONE was spread across two academic terms, with work being split into phases. The initial phase was setting up infrastructure, such as hosting Odoo and setting up the React frontend and Node.js+Express.js backend. During this phase, we also conducted meeting with HR from Outfitters to understand their current process and inefficiencies. Then overall system design and process flow development began, followed by beginning development. We incrementally implemented each of the five targeted business processes, completing one process end to end with testing before moving to the next. During development of the expense reimbursement module, the fraud pipeline was being developed side by side, and after successful initial testing, was integrated to the complete reimbursement process and automated workflow. Each workflow was developed with HITL in extreme cases, whereas for general cases seamless automation was the goal. Once all processes were complete, audit nodes in all flows were added to connect and log the complete processes to an independently hosted PostgreSQL table. Git was used throughout with commits between functional milestones, and separate branches for each process.

Keywords

HR Process Automation, RPA Workflow Orchestration, HITL Automation

Branch Performance Analytics Dashboard with Predictive Footfall Forecasting

Group Members

Afnan Anwer Kiani (26900)
Ali Uz Zaman (26941)
Hussain Ali Shah (27131)

Supervisors

Supervisor (PI): Ms. Tasbiha Fatima
Industry Advisor: Malik Qamar Hayat
Industry Company: Habib Metro Bank

Knowledge Area: Artificial Intelligence based systems

Project Overview

The Branch Performance Analytics Dashboard with Predictive Footfall Forecasting is a web-based analytics platform developed for Habib Metro Bank to improve branch management through real-time monitoring and predictive intelligence. The project aims to provide branch managers and executives with actionable insights into operational performance, customer traffic patterns, transaction activity, and staff utilization through interactive dashboards and data visualization tools. By integrating machine learning models such as ARIMA and Prophet for customer footfall forecasting, the system enables proactive resource allocation, optimized staffing, and data-driven decision-making. The project contributes by combining secure banking-grade software engineering practices with predictive analytics to enhance operational efficiency, reduce costs, and improve customer service quality within banking operations.

Technologies Used

Python, FastAPI, React.js, PostgreSQL, ARIMA, Facebook Prophet, scikit-learn, statsmodels, RESTful APIs, Chart.js, Material-UI, Axios, React Router, Node.js, npm, Webpack, Nginx, Apache, Linux (Ubuntu), AWS, Microsoft Azure, Google Cloud, bcrypt, SSL/TLS, PostgreSQL psycopg2 driver

Specifications

The system is designed to operate on a Linux-based server environment (Ubuntu 20.04 LTS or later) using Python 3.9+ with FastAPI for backend development, PostgreSQL 13+ for database management, and React.js 18+ for frontend development. It supports deployment through Nginx or Apache web servers and is compatible with cloud platforms such as AWS, Microsoft Azure, and Google Cloud. The client-side requirements include modern web browsers such as Chrome 90+, Firefox 88+, Safari 14+, and Edge 90+, with a minimum screen resolution of 1366×768 and internet connectivity of at least 5 Mbps. The system uses RESTful APIs, Axios, Chart.js, Material-UI, React Router, Node.js, npm/yarn, and Webpack. Security specifications include TLS 1.2+ encryption, bcrypt password hashing with a minimum of 12 rounds, RBAC-based authorization, MFA authentication, AES-256 encryption for sensitive data, and audit logging. Recommended server hardware includes a multi-core processor (minimum 4 cores), 16GB RAM, 500GB SSD storage, and 1Gbps network connectivity.

Methodology

The project follows a structured software engineering methodology combining requirements analysis, system design, agile-based development, and machine learning model implementation. The development process began with the creation of a comprehensive Software Requirements Specification (SRS) to define functional and non-functional requirements, user roles, system constraints, and interface specifications. The system architecture was designed using a modular client-server approach with React.js for the frontend, FastAPI for backend services, and PostgreSQL for database management. Machine learning methodologies, including ARIMA and Facebook Prophet time-series forecasting models, were used to analyze historical branch data and predict future customer footfall patterns. The project also incorporates secure software development practices such as role-based access control, MFA authentication, audit logging, and encryption to ensure compliance with banking security standards. Continuous testing, validation of prediction accuracy, and iterative feature improvements were planned throughout the development lifecycle to ensure reliability, scalability, and usability.

Keywords

Predictive Analytics, Machine Learning, Banking Dashboard

Visual QA

Group Members

Agha Salik Ali (26986)
Simra Sheikh (27135)
Shaikh Muhammad Sharjeel (26932)

Supervisors

Supervisor (PI): Dr. Imran Rauf
Industry Advisor: Mr. Saad Mughal
Industry Company: AlphaVenture

Knowledge Area: Artificial Intelligence based systems

Project Overview

Visual QA is an AI powered design quality assurance platform that automates the detection of visual discrepancies between Figma design mockups and live web implementations. Design drift, the gradual accumulation of undetected layout, style, and content deviations between intended designs and deployed interfaces, erodes brand fidelity and creates a slow, error prone feedback cycle for development teams. The platform addresses this by providing an automated QA workflow: a custom Figma plugin exports design frame data to the system, which captures live website screenshots, performs a multi layer comparative analysis using vision language models, and produces a structured report with issues classified by severity, category, and confidence score. Beyond detection, an autonomous code fixing agent locates the source of each discrepancy, applies targeted changes, and submits a GitHub pull request, enabling complete resolution without manual developer involvement. Visual QA delivers a scalable, production ready framework that reduces the cost of maintaining design fidelity across iterative software development.

Technologies Used

Frontend: React 19, Vite, Laravel Echo, Pusher.js
Backend: Laravel 10, PHP 8.2
Database: MySQL
Real-time & WebSocket: Laravel Reverb
Browser Automation: Playwright, Chromium
AI & Vision Models: Claude Sonnet 4.6, Claude Sonnet 4.5
API & Integration: OpenRouter API, GitHub REST API
Figma Plugin: TypeScript, esbuild

Specifications

The backend requires PHP 8.2, MySQL, and a queue worker environment. The frontend is a single page application accessible via any modern browser. Browser automation is handled by a Playwright managed Chromium instance. Vision language model inference is routed through the OpenRouter API with a rate limit of 10 requests per 60 second window. GitHub branch creation, commits, and pull request submission use the GitHub REST API. The Figma plugin is a standalone TypeScript application deployed through the Figma client. Real time progress events are delivered over WebSocket via Laravel Reverb using the Pusher.js protocol on the client side.

Methodology

The system is structured around a four phase sequential job pipeline. In the first phase, a large language model processes exported Figma JSON to identify and extract named design sections. The second phase uses Playwright to launch a headless Chromium browser and capture screenshots of the target website segmented to match each section. The third phase runs a two layer VLM analysis per section pair: Layer A evaluates structural and content fidelity using screenshots alongside extracted HTML and Figma JSON, while Layer B assesses visual style including typography, colors, and spacing using computed CSS and Figma JSON. A deduplication call then removes cross layer duplicate findings before the report is compiled. The optional fourth phase runs an autonomous agent loop that iterates over flagged issues, using file system and code search tools to locate and fix responsible code, with GitHub integration to commit changes and open a pull request. Each phase broadcasts real time progress to the frontend via WebSocket.

Keywords

Vision Language Models, Design Automation, Visual Quality Assurance

Active-Ledger: Trust-Gated Diffusion and Behavioural Contribution Scoring for Adversarially Robust Federated Learning on Clinical IoT Non-IID Data

Group Members

Muhammad Maaz Siddiqui (27070)
Muhammad Ibrahim Farid (27098)
Muhammad Ibrahim Iqbal (27085)

Supervisors

Supervisor (PI): Dr. Faisal Iradat

Knowledge Area: Security

Project Overview

Federated learning (FL) is a natural fit for clinical electrocardiography: hospital data never leaves the hospital, while the shared model still benefits from every site. A clinical deployment has to survive two problems at once. Hospital data is highly non-independent and identically distributed (non-IID), and adversaries that get a foothold will preferentially target the rare, clinically dangerous classes. This paper introduces Active-Ledger, a blockchain-orchestrated FL system whose defence is behavioural rather than geometric and which also closes a second attack surface that classical robust aggregation never looks at: the augmentation channel that opens when a diffusion generator is added to repair rare class deficit. The Proof-of-Contribution score proposed here is a simple, on-chain, history-based contribution score; it is not a cryptographic proof. It is paired with a trust-gated classconditional 1D latent diffusion model (LDM) generator, and the ledger records every aggregation and every generation as an immutable event. Active-Ledger is evaluated on MIT-BIH arrhythmia classification under three attacks (Gaussian weight poisoning, static label-flip, and a sleeper backdoor-critical-layer flip that activates mid-training), against six baselines (FedAvg, Krum, Multi-Krum, Median, Trimmed-Mean, Bulyan). Across all three attacks, PoC is the only defence that keeps atrial-premature-beat (APB, Class 3) F1 above the 0.75 deployability line. Adding trust-gated diffusion then pushes APB-F1 to 0.879 under the sleeper attack, beating an un-gated Multi-Krum + Blind LDM baseline on both APB-F1 and final-round backdoor success rate.

Technologies Used

Python, PyTorch, Flower (Federated Learning), HuggingFace Diffusers, Solidity, Web3.py, Ethereum (Ganache), NumPy, Scikit-learn, SciPy, WFDB, Pandas, Matplotlib, Seaborn, Git, CUDA

Specifications

Hardware: NVIDIA Tesla T4 GPU (16 GB VRAM) for model training and diffusion inference; experiments executed across isolated cloud GPU sessions
Software: Python 3.10+, PyTorch 2.0+ with CUDA 11.8, Flower 1.7+ (federated learning framework), HuggingFace Diffusers 0.20+ (DDPM scheduler and UNet1DModel).
Blockchain: Solidity 0.8 smart contract compiled via py-solc-x, deployed on Ganache 7.0 local Ethereum network (15 accounts, 30M gas limit), interfaced through Web3.py 6.0+.
Dataset: MIT-BIH Arrhythmia Database from PhysioNet (48 ECG recordings, 360 Hz sampling rate, ~100K segmented heartbeat windows across 5 AAMI classes).
APIs/Libraries: WFDB for PhysioNet data retrieval, Scikit-learn for evaluation metrics (F1, precision, recall), SciPy for statistical analysis, Matplotlib/Seaborn for visualization.
Configuration: 40 FL rounds, 10 clients (8 honest + 2 Byzantine), 3 local epochs, batch size 128, Adam optimizer ($\text{lr}=0.001$), fixed random seed 42 for full reproducibility

Methodology

We adopt an experimental research methodology centered on a federated learning simulation framework. The MIT-BIH Arrhythmia Database is preprocessed into 360-sample ECG windows, labeled into 5 AAMI classes, and partitioned across 10 simulated clients using a Dirichlet-based non-IID distribution to replicate real-world clinical data heterogeneity. A CNN-LSTM classifier serves as the shared global model, trained via Flower-based federated rounds with class-weighted cross-entropy loss to handle extreme class imbalance (82:1 ratio). We implement and benchmark 9 aggregation strategies — FedAvg, Krum, Multi-Krum, Coordinate-Wise Median, Trimmed Mean, Bulyan, Proof-of-Contribution (PoC), Multi-Krum with Blind Latent Diffusion, and the full Active-Ledger system, under three Byzantine attack scenarios (Gaussian noise, label-flip, and sleeper attacks with 20% adversary ratio). Blockchain integration uses an Ethereum smart contract (Solidity) deployed on Ganache for immutable audit logging and EMA-weighted PoC scoring. A class-conditioned 1D DDPM UNet generates synthetic ECG signals for minority class augmentation, authorized via on-chain trust gating. All experiments use fixed seeds for reproducibility, with sequential method execution and isolated GPU environments to prevent cross-experiment data leakage.

Keywords

Federated learning, Blockchain, Byzantine robustness

AI Based Cost Management System

Group Members

Huzaiifa Ahmed Khan (26485)
Ali Saeed (27153)
Muhammad Bilal Adnan (27151)

Supervisors

Supervisor (PI): Dr. S.M. Faisal Iradat
Industry Advisor: Danish Hussain Baig
Industry Company: Herbion Naturals
Industry Mentor: Danish Hussain Baig

Knowledge Area: Artificial Intelligence based systems

Project Overview

This project is an AI-Based Cost Management System built to automate the entire procurement document processing pipeline for organizations. The system takes scanned invoices, Purchase Orders and Goods Receipts in PDF or image format, extracts structured financial fields using a multi-engine OCR pipeline combining Tesseract, GOT-OCR and fine-tuned LayoutLMv3 and Donut transformer models, and then performs three-way matching to validate invoices against their corresponding POs and GRs. Anomaly detection runs through both a rule-based checker and an unsupervised Isolation Forest model, with a privacy-preserving upgrade built on top via a Federated Learning framework that trains a neural anomaly detector across distributed clients using FedAvg with Differential Privacy, making sure no raw financial data ever leaves client premises. The full system is exposed through a FastAPI backend and visualized through a Streamlit analytics dashboard and is trained and evaluated on a mix of real-world scanned document datasets to make it robust enough for actual enterprise deployment.

Technologies Used

Python, FastAPI, Streamlit, PyTorch, Hugging Face Transformers, LayoutLMv3, Donut (Document Understanding Transformer), PEFT/LoRA, Tesseract OCR (pytesseract), GOT-OCR, scikit-learn, RapidFuzz, Federated Learning (FedAvg), Differential Privacy, Isolation Forest, OpenCV, Pillow, pdfplumber, pandas, SQLite, openpyxl, ReportLab, Kaggle (GPU training environment)

Methodology

The project follows a pipeline-based development methodology split across five phases. Phase 1 covers data preparation for Purchase Orders, Goods Receipts and invoices. Phase 2 builds the document extraction layer where a multi-engine OCR pipeline attempts spatial PDF parsing, Tesseract and GOT-OCR in sequence, with a fine-tuned LayoutLMv3 via PEFT and LoRA acting as the primary field extractor for structured key-value recognition. Phase 3 implements three-way matching using fuzzy string matching to reconcile invoice fields against PO and GR records and flag any discrepancies. Phase 4 adds anomaly detection by combining a rule-based checker with an Isolation Forest model trained on historical invoice features. Phase 5 brings in the Federated Learning layer, where an MLP anomaly detector is trained using FedAvg with Gaussian Differential Privacy across simulated distributed clients, so the model keeps improving without any raw financial data ever being centralized. The whole system is then served through a FastAPI REST API and visualized on a Streamlit dashboard.

Specifications

Backend: Python 3.10+, FastAPI with Uvicorn, SSE (Server-Sent Events) for real-time progress streaming

Dashboard: Streamlit 1.30+

ML Models: LayoutLMv3 (fine-tuned via PEFT/LoRA), Donut, Isolation Forest, custom MLP

Federated Learning: Custom FedAvg server with secure aggregation simulation and Gaussian Differential Privacy (ϵ -DP)

OCR Engines: Tesseract 5.x, GOT-OCR (llama-3 family)

Database: SQLite (via custom db module)

Data Formats: PDF, PNG/JPG (scanned), CSV, XLSX

Training Environment: Kaggle Notebooks (T4 GPU) for LayoutLMv3 and Donut fine-tuning

Hardware Requirements: 8 GB RAM minimum; CUDA-compatible GPU recommended for inference with transformer models

Testing: pytest (unit tests for extraction, matching, and detection modules)

Keywords

Invoice Processing, Federated Learning, Optical Character Recognition (OCR)

Awaaz se Alfaaz: Enhancing Urdu ASR through Whisper Fine-Tuning and SLM-Based Post-Processing

Group Members

Zehra Ahmed (26965)
Farah Inayat (26912)
Zuha Aqib (26106)

Supervisors

Supervisor (PI): Dr. Sajjad Haider

Knowledge Area: Artificial Intelligence based systems

Project Overview

This project focuses on improving Automatic Speech Recognition for Urdu, a low-resource language where existing models still struggle with accents, background noise, code-switching, and inconsistent Urdu spellings. We fine-tuned OpenAI Whisper Large-v3 and Whisper Large-v3-Turbo using LoRA on Urdu datasets including Common Voice v23, FLEURS, and CSaLT. The system was evaluated not only on benchmark datasets but also on a real-world YouTube evaluation set containing two-speaker Urdu news discussions with noise, interruptions, and natural speech patterns. We also tested decoder-level optimization and Small Language Model based post-processing to reduce transcription errors and improve spelling consistency. The project contributes updated Urdu ASR benchmarks, a realistic multi-speaker evaluation setup, and an end-to-end transcription pipeline aimed at making Urdu audio easier to transcribe, search, and analyze.

Technologies Used

Python, PyTorch, Hugging Face Transformers, OpenAI Whisper Large-v3, Whisper Large-v3-Turbo, LoRA, PEFT, Unsloth, jiwer, FFmpeg, Common Voice v23, FLEURS, CSaLT, YouTube audio dataset, Qwen3-14B, Tiny Aya Fire, Qalb-1.0-8B-Instruct, Gemma-2-9B, FastAPI, HTML, CSS, JavaScript, GitHub, Ubuntu VM, NVIDIA A40 GPU

Methodology

We first reviewed and replicated prior Urdu ASR benchmarking work to understand existing model performance and limitations. After that, we prepared Urdu speech datasets from Common Voice v23, FLEURS, and CSaLT by standardizing audio format, normalizing Urdu text, and creating consistent train, validation, and test splits. Whisper Large-v3 and Whisper Large-v3-Turbo were evaluated in zero-shot form and then fine-tuned using LoRA to adapt them better to Urdu speech. After fine-tuning, we performed decoder-level optimization by testing different inference settings such as beam width and length penalty. To evaluate real-world usefulness, we created a YouTube-based evaluation set containing 30 Urdu news clips with two speakers, background noise, and natural conversation flow. Finally, we applied Urdu-capable Small Language Models as a post-processing step to correct minor spelling and recognition errors without changing the meaning or structure of the transcript. Performance was measured mainly using Word Error Rate, and for SLM post-processing we also used CER, MER, and WIL.

Specifications

The experiments were conducted on an Ubuntu 22.04.5 LTS GPU environment with 32 vCPUs, 62 GB RAM, and 1 NVIDIA A40 48GB GPU. Audio files were converted to mono channel, 16 kHz sampling rate, and 16-bit PCM format before processing. The models used were Whisper Large-v3 and Whisper Large-v3-Turbo, fine-tuned using LoRA with mixed precision FP16 training. Turbo used LoRA rank 32, alpha 64, and dropout 0.05, while Large-v3 used rank 16 and alpha 32. Evaluation was performed using the jiwer library with consistent text normalization and metric calculation across all experiments.

Keywords

Urdu ASR, Whisper Fine-Tuning, SLM Post-Processing

Group Members

Zainab Rehman (25987)
Ameera Ali (26910)
Areeba Fatima (25919)
Zehra Imam (26946)

Supervisors

Supervisor (PI): Dr. Sajjad Haider

Knowledge Area: Artificial Intelligence based systems

Project Overview

Vocalytics is an AI powered platform that makes voice-based interviews easier, faster, and more reliable. Traditional interviews are slow, limited to one person at a time, and require manual recording and analysis, which can lead to mistakes and extra work. Our platform solves these problems by allowing multiple interviews at once, providing real time transcription, automatic analysis, and instant reports, all accessible in both Urdu and English. Users can easily create interviews, share links, and track responses on a simple dashboard. Behind the scenes, technologies like WebRTC, voice activity detection, and smart AI models ensure conversations feel natural, handle off topic or long answers, and keep the data accurate. With a fast and secure backend, Vocalytics stores responses efficiently and is ready to scale for many users. All in all, by combining artificial intelligence with voice based real time communication Vocalytics transforms how interviews are conducted across multiple domains like Healthcare, Hospitality, Human Resources, Market Research, and more.

Technologies Used

WebRTC, WebSockets, Supabase, Next.js (React), Streamlit, LiveKit, Python, Flask, Deepgram, Groq, Azure Cognitive Services.

Specifications

Vocalytics works on any standard laptop, desktop, or mobile device with a web browser and an internet connection. Users need a working microphone to participate in interviews. The system uses Whisper Large via Deepgram API key for transcribing speech, LLaMA 3.3 70B via Groq API for deciding interview flow logic, Azure Cognitive Services for converting text to speech, and LiveKit framework to manage the real time communication pipeline. Silero VAD is used to detect when a respondent stops or starts speaking. All responses are stored securely in Supabase. The platform supports both Urdu and English, making it easy for users to conduct and participate in interviews naturally.

Methodology

Vocalytics was developed through an iterative prototyping approach. We identified problems with traditional interviews via discussions with people in the market research and hospitality industry. These discussions helped us identify Functional and Non-Functional requirements that we then organized on Figma. By using a visual board, we could simultaneously define the system architecture and the user journey, as well as set priority for each task. To manage the development lifecycle, we utilized GitHub for version control and collaborative coding. We used WebRTC protocols to achieve low-latency and natural-sounding conversations and implemented Voice Activity Detection (VAD) to accurately track speaker transitions and maintain context. We also developed logic to handle asynchronous stream management, resolving technical hurdles such as duplicate tokens and interrupted response strings, to maintain coherency in responses. Throughout the development lifecycle, we performed rigorous testing for Bilingual Natural Language Understanding, ensuring the system remained performant in both Urdu and English. Ultimately, this methodology enabled us to transition from a single agent prototype to a modular framework capable of deploying custom agents for diverse industry needs.

Keywords

Agentic Interviewer Designer, Automated Interview System, Conversational Interview Agents

AI Agent-Orchestrated Digital Shadow Framework for Multilingual Healthcare Workflow Automation

Group Members

Ansh Dev Udassi (26903)
Ashal Ibrahim Khan (26977)
Hamza Moosani (26945)

Supervisors

Supervisor (PI): Dr. Imran Khan

Knowledge Area: AI AGENT, VOICE AGENT, HEALTHCARE

Project Overview

Modern healthcare systems face significant operational inefficiencies, with professionals spending substantial time on manual documentation and administrative coordination rather than direct patient care. This paper proposes an AI agent-orchestrated digital shadow framework for healthcare workflow automation, designed to streamline routine clinical operations while maintaining reliability and transparency. The framework integrates multi-agent orchestration with a Digital Shadow—a real-time digital representation of administrative workflows that can observe, coordinate, and automate routine tasks. At its core is a modular orchestration layer where a centralized agent functions as an intent classifier and workflow coordinator, dynamically routing requests to specialized sub-agents: a triage agent conducting structured patient interviews and generating History of Present Illness (HPI) notes, an appointment booking agent managing scheduling through real-time database and API integration, and a general query agent utilizing Retrieval-Augmented Generation (RAG) against a verified medical knowledge base. To ensure longterm robustness, the framework incorporates a reflective monitoring component that continuously analyzes interaction patterns to detect model drift and workflow inefficiencies, enabling incremental improvement without disrupting live operations. To evaluate the framework’s core architectural claims, we conducted an offline benchmark on a dataset of 50 expert-annotated bilingual queries.

Technologies Used

Python, langchain, langgraph, Huggingface, fastAPI,Ollama, ElevenLabs, groq, medgemma

Specifications

GPU

Keywords

AI Agent, Voice agent, Healthcare, Automation

Methodology

The proposed Digital Shadow framework is a modular multi-agent healthcare system designed to automate patient intake, appointment management, and pre-consultation data collection while supporting local-language interactions. A bilingual Leader Agent serves as the primary interface, coordinating specialized agents through Model Context Protocol (MCP) tool-calling rather than rule-based routing. To improve understanding of Pakistani clinical vocabulary, incoming Urdu, English, and code-switched patient utterances are first processed through a RAG-based local language dictionary that maps colloquial expressions to standardized clinical terminology. Based on conversational context, the Leader Agent dynamically invokes specialized agents for doctor recommendation, appointment booking, symptom elicitation, and general information retrieval. The Symptom Elicitation Agent conducts adaptive history-taking interviews, gathering structured History of Present Illness (HPI) information through context-aware follow-up questions, while a Diagnostic Summarization Model converts the collected information into a concise pre-consultation report for physicians. All agents operate behind standardized interfaces, enabling hot-swappable deployment and independent upgrades without service interruption. To ensure privacy and data sovereignty, the framework utilizes self-hosted open-weight language models (8B–14B parameters) that can run entirely on local clinic infrastructure. A passive Reflective Monitoring Component continuously analyzes completed interactions, tracking ambiguity, missing information, routing uncertainty, and unrecognized vocabulary, and surfaces flagged cases to administrators for review and future system improvement without automated retraining.

PrivCap: Benchmarking VLMs on Cross-Modal Privacy Amplification

Group Members

Alina Siddiqui (26555)
Fiza Hussain (26749)
Muhammad Ahsanuddin Ahmed (27134)

Supervisors

Supervisor (PI): Sajjad Haider
Co-PI: Sualeh Ali
Industry Mentor: Irtiza Zaidi

Knowledge Area: Artificial Intelligence based systems

Project Overview

PrivCap: Benchmarking VLMs on Cross-Modal Privacy Detection evaluates how well modern Vision-Language Models can detect privacy risks in content of the kind shared daily on social media, where an image paired with a caption can together reveal far more personal information than either alone. The project replicates and extends the benchmarking methodology of Tsaprazlis et al. (2025), "Assessing Visual Privacy Risks in Multimodal AI," using a 14-category taxonomy-grounded privacy framework across three benchmark datasets: VISPR (7,995 images), PrivacyAlert (1,554 images), and DIPA2 (265 unanimously annotated images). In the first phase, we replicate the paper's three standardized tasks: direct privacy-risk detection, taxonomy-guided detection, and multi-label attribute recognition, evaluating four recent VLMs: Qwen3-VL-8B, LLaMA-3.2-11B-Vision-Instruct, Gemma-3-4B-IT, and Ministral-3-3B across image-only conditions. In the second phase, we introduce a curated multimodal dataset pairing VISPR images with structured captions across two PII leakage levels (explicit and no-leak), constructed via a multi-stage Gemini Flash-Lite pipeline. Models are re-evaluated on the same three tasks with captions included, and a novel fourth task is introduced measuring cross-modal source attribution: whether models can correctly identify if a privacy violation originates from the image, the caption, or both. This phase directly tests whether VLMs can integrate visual and textual signals to detect privacy risks that neither modality would expose alone.

Methodology

The project follows an experimental benchmarking methodology structured in two phases. In the first phase, we mapped annotations from three datasets: VISPR, PrivacyAlert, and DIPA2, to a shared 14-category privacy taxonomy derived from Tsaprazlis et al. (2025). Each image was prepared with ground truth privacy labels, and four VLMs were evaluated on three tasks: direct privacy-risk detection (Task 1), taxonomy-guided binary detection (Task 2), and taxonomy-guided multi-label attribute recognition (Task 3). Each image was passed to the model with task-specific hierarchical prompts, and predictions were collected across three runs using majority voting to improve reliability. Two temperature settings (0.1 and 1.0) were tested to study response consistency. Outputs were evaluated using Macro F1, precision, recall, per-category F1, and accuracy, and exported to JSON and Excel for analysis. In the second phase, a multimodal dataset was curated by pairing VISPR images with structured captions generated through a multi-stage Gemini Flash-Lite pipeline, producing image-text pairs across two PII leakage levels: explicit and no-leak. Ground truth was reconstructed to reflect both image-grounded and caption-induced privacy labels, enabling evaluation of cross-modal signal integration. Models were re-evaluated on Tasks 1 through 3 with captions injected into prompts, and a novel Task 4 was introduced requiring models to attribute privacy violations to the image channel, the caption channel, or both. Results across both phases were compared to establish where zero-shot VLMs succeed and fail under multimodal conditions, motivating future fine-tuning work.

PrivCap: Benchmarking VLMs on Cross-Modal Privacy Amplification

Technologies Used

Python, Google Colab, LM Studio, HuggingFace Transformers, PyTorch, CUDA, NVIDIA RTX 4050, A100 GPU, T4 GPU, L4 GPU, Qwen-VL/Qwen3-VL, LLaMA-3.2-11B-Vision-Instruct, Gemma-3-4B-IT, Ministral-3-3B, Gemini API, Florence-2, LM Studio, VISPR dataset, PrivacyAlert dataset, DIPA2 dataset, NumPy, pandas, scikit-learn, openpyxl, Google Drive, JSON annotations, Excel result export, bfloat16 inference, majority voting, temperature sampling

Keywords

Vision-Language Models, Multimodal Privacy Detection, Personally Identifiable Information, Visual Privacy Benchmarking, Cross-Modal PII Leakage, Privacy Taxonomy, Zero-Shot Evaluation, Social Media Privacy, Attribute Recognition, Source Attribution

Specifications

The project was implemented using Python notebooks across two compute environments. Primary inference was run on Google Colab using A100 (40GB) and L4 (24GB) GPUs via the HuggingFace Transformers library, with models loaded in bfloat16 full precision. LM Studio was used during an early pilot phase for local quantized model testing on a limited subset before full-scale evaluation. The evaluation datasets consist of VISPR (7,995 images), PrivacyAlert (1,554 images), and DIPA2 (265 images), each with JSON annotations and taxonomy-mapped labels, along with curated image-caption pairs for the multimodal evaluation phase. The pipeline supports image preprocessing with longest-edge resizing, hierarchical prompt-based inference, repeated runs with majority voting, temperature-based consistency testing, automatic metric calculation, checkpointed incremental saving to Google Drive, and result export in JSON and Excel formats. Caption generation for the multimodal dataset was performed using the Gemini Flash-Lite API. All models: Qwen3-VL-8B, LLaMA-3.2-11B-Vision-Instruct, Gemma-3-4B-IT, and Ministral-3-3B, were evaluated at full scale across thousands of images with outputs parsed using robust JSON extraction and fallback heuristics to handle malformed model responses.

AI Workflow Automator

Group Members

Fizza Zehra (26944)
Ahmad Murtaza (24478)
Fatima Jawaid (25943)

Supervisors

Supervisor (PI): Ms. Abeera Tariq

Knowledge Area: Software Development

Project Overview

This project addresses the burden of repetitive administrative work faced by university educators on Learning Management Systems (LMS) by delivering two integrated AI-powered systems. The first is a browser automation agent that accepts natural-language instructions and uses Playwright with multi-LLM support (OpenAI, Anthropic Claude, Google Gemini) to autonomously navigate and perform tasks on browsers, such as uploading materials and managing courses, eliminating manual interaction. The second is an automated paper marking system that uses AI vision to evaluate scanned exam scripts against uploaded rubrics, generate per-question marks, and export results, reducing hours of manual grading to minutes. Together, the systems form a teacher productivity platform that demonstrates practical LLM-driven automation applied to real educational workflows, backed by a shared Supabase database for authentication and persistent data.

Technologies Used

Python, FastAPI, Uvicorn, Playwright, OpenAI API, Anthropic Claude API, Google Gemini API, React 18, Next.js 14, TypeScript, Tailwind CSS, Supabase (PostgreSQL), WebSockets, Pydantic, PyPDF2, pdf-to-img, Radix UI, Vite

Specifications

Backend: Python 3.10+, FastAPI + Uvicorn, Playwright (Chromium), multi-LLM APIs (OpenAI GPT-5.4, Claude 4.5 Sonnet, Gemini 3.0 Flash)
Frontend (Browser Agent): React 18, Vite, Tailwind CSS, WebSocket client
Frontend (Paper Marking): Next.js 14, TypeScript, Tailwind CSS, Radix UI components
Database: Supabase (PostgreSQL) with Row-Level Security and JWT-based auth
File handling: PyPDF2 / pdf-to-img for PDF parsing; Sharp for image processing; XLSX/Papaparse for result export
Real-time: WebSocket for streaming agent step events to the UI
Hardware: Standard Windows 11 development machine; no GPU required (all inference via cloud APIs)

Methodology

The project followed an iterative, feature-driven development approach. Work was broken into phases: (1) core Playwright browser automation engine with LLM tool-calling integration; (2) FastAPI backend refactored into modular routers for browser control, auth, LMS data, and flow history; (3) React frontend with real-time WebSocket event streaming; (4) Supabase auth and LMS onboarding wizard so teacher credentials auto-inject into every agent task; (5) a separate Next.js paper marking application using AI vision to process PDF exam scripts and Jupyter Notebooks against marking rubrics. Both systems were developed in parallel and share the same Supabase project for auth and storage.

Keywords

Agentic AI, Automated Assessment, Educational Technology

Smart Payment Reconciliation System for PayPro

Group Members

Abdul Nafay (26951)
Alina Zindani (27114)
Zain Mufti (26920)

Supervisors

Supervisor (PI): Miss Tasbiha Fatima
Industry Advisor: Mr. Iqar Memon
Industry Company: PayPro

Knowledge Area: Software Development

Project Overview

The Smart Payment Reconciliation System for PayPro is an industry-deployed automated reconciliation pipeline built to replace PayPro's entirely manual payment reconciliation process across multiple payment gateways. Before this project, PayPro's finance team spent a full working day per reconciliation cycle doing line-by-line spreadsheet comparisons across Meezan Bank (MBL) and lLink gateway reports, with no systematic audit trail and 3 to 5 data entry errors per week. The system implements a three-layer pipeline architecture covering data ingestion via Google Drive file triggers, a Python-based processing engine that handles gateway-specific parsing, aggressive ID normalization, set-based and bidirectional matching algorithms, and settlement summary generation with bank code enrichment, and finally an output and delivery layer that produces timestamped CSV discrepancy reports and Excel settlement summaries delivered via automated email. Operational visibility is provided through a Streamlit dashboard and a custom desktop Drive Uploader application. The delivered system achieves a 99.7% match accuracy rate, reduces end-to-end reconciliation time from a full working day to under 5 minutes per gateway, eliminates all manual data entry and bank code mapping errors, accelerates the merchant settlement cycle by one full business day, and saves an estimated 110 to 120 person-hours monthly while establishing a modular, gateway-agnostic foundation that can be extended to additional payment channels.

Technologies Used

Python, pandas, regex, Zapier, Google Drive API, Streamlit, Gmail (automated notifications), Microsoft Excel (openpyxl/formula-protected outputs), CSV, Google Drive (file-based trigger and storage)

Methodology

We developed our system using an Agile-Scrum sprint-based approach, starting from gathering requirements and data analysis for gateway-specific implementation and deployment. The architecture follows a stateless three-layer pipeline: a file-based ingestion layer using Google Drive folder monitoring, a Python processing engine with custom per-gateway parsers, ID normalization, matching logic, and settlement aggregation, and an output and delivery layer producing CSV reports, Excel summaries, automated email notifications, and a dated folder audit trail. Key design principles include stateless processing with no persistent database dependency, sequential execution for dependency safety, per-step error isolation, and a gateway-agnostic output format. The originally proposed database-centric approach using SQL Server and Power Automate was replaced with this serverless file-based architecture after discovering that MBL provides no programmatic API access and lLink's FTP uses undocumented authentication, making file-based processing the only practical path to full automation.

Specifications

Automation platform: Zapier serverless workflows with 15-minute polling on Google Drive triggers. Processing language: Python with pandas and regex-based stream parsing. Gateway coverage includes MBL, handled through ID-based set matching, settlement aggregation by Account number and MID key, and bank code for 40+ Pakistani financial institutions, and lLink, handled through regex stream parsing of unstructured DB Alert text, bidirectional set comparison, and T+1 settlement date calculation. Output formats include CSV discrepancy reports and Excel formula-protected settlement summaries using "0234" syntax to preserve leading zeros, with automated Gmail notifications on success or failure. The UI is a Streamlit-based reconciliation dashboard showing match rate, discrepancy counts, merchant-level filtering, and transaction trend visualization, alongside a custom desktop Drive Uploader application for guided, error-free file uploads. The system completes end-to-end processing in under 5 minutes per gateway and operates statelessly with no persistent database.

Keywords

Payment Reconciliation, FinTech, Process Automation

Group Members

Saad Imam (27079)
Syed Rafay Ahmed (27071)
Ali Siddiqi (26902)

Supervisors

Supervisor (PI): Dr. Ahmed Akhtar
Co-PI: Sir Adil Saleem

Knowledge Area: Blockchain Technology

Project Overview

BlockLease is a full-stack, blockchain-powered real estate investment platform designed to address the high entry barriers, inefficiencies, and lack of transparency in traditional property investment, particularly in Pakistan where real estate ownership is often inaccessible to middle-income earners, young professionals, and expatriates due to large capital requirements. The project aims to democratize real estate investment by enabling fractional ownership of properties, allowing users to invest with significantly lower amounts while benefiting from secure, transparent, and automated asset management through blockchain technology and smart contracts. The platform also incorporates a secondary marketplace for P2P and PMM trading, along with user and market analytics and agentic insights.

Technologies Used

Solidity, Hardhat, ethers.js, OpenZeppelin Contracts, Docker + Docker Compose, TypeScript, Node.js, React, Nginx, Prisma, PostgreSQL, Python, Ollama, Apache Spark, Apache Airflow, Selenium, Clickhouse, Redis, IPFS (Kubo), Jest, Chai, Mocha

Specifications

The complete project was deployed on IBA's High Performance Computing (HPC) cluster and operated within a fully dockerized environment using Docker Compose. The system followed a monorepo architecture managed with Nx and required Node.js ($\geq 18.0.0$), npm ($\geq 8.0.0$), Python 3.11 for the AI agent, and Docker for deployment and orchestration. The frontend was developed using React 19, Vite 7, Tailwind CSS, and ethers.js for Web3 integration, while the backend used Node.js with TypeScript, Express, Prisma ORM, Kafka, Redis, and JWT-based authentication. Smart contracts were developed in Solidity using Hardhat and implemented ERC-1155 token standards along with ERC-4337 account abstraction. The AI agent was built using FastAPI, LangChain, Ollama, and ChromaDB. Supporting infrastructure included PostgreSQL, ClickHouse, IPFS, Nginx, Apache Spark, Apache Airflow, and a local Hardhat blockchain network for smart contract interaction and testing.

Methodology

The project followed an Agile Scrum methodology with 2-week sprints, chosen for its iterative approach that enabled frequent testing, evaluation, and refinement of blockchain components throughout the development process. Sprint planning was centered around deliverable modules, with each sprint producing functional components that could be demonstrated, tested, and improved incrementally. Bi-weekly standups were conducted to maintain team coordination and track progress, while sprint retrospectives were used to identify and resolve technical challenges related to blockchain development and system integration.

Keywords

Fintech, Blockchain/Tokenization, Fractional Real Estate

Autonomous agentic pentesting

Group Members

Omar Ashraf Khan (26985)
Abdul Wasay Imran (27126)
Deebaj Haider Kazmi (26012)

Supervisors

Supervisor (PI): Dr Faisal Iradat
Industry Advisor: Musab Zuberi
Industry Company: Talent Pop

Knowledge Area: Cyber Security and Agentic AI

Project Overview

APEX is an extensible, event-driven orchestration framework for automated penetration testing and red-team workflows that coordinates modular agents (recon, VAPT, triage, reporting) and LLM-assisted decision logic to perform reproducible, resumable security assessments; it provides a CLI and a FastAPI server (SSE) for live streaming, MongoDB-backed checkpointing for pause/resume, and contributes a reusable agent/graph orchestration model and tooling for auditable, automated security engagements.

Technologies Used

Python, FastAPI, Uvicorn, LangChain, LangGraph, langchain-anthropic, langchain-google-genai, MongoDB, PyMongo, langgraph-checkpoint-mongodb, Docker, Docker Compose, Pydantic, HTTPX, BeautifulSoup4, Rich, ReportLab, Paramiko, Pillow, PyYAML, Readchar, MkDocs (mkdocs-material), Git.

Methodology

Modular agent-based design with graph-driven workflows (LangGraph), event-driven architecture (in-process event bus + SSE), background-thread runner with deferred imports, MongoDB checkpointing for resumability, iterative development with containerized integration, and LLMs (LangChain providers) for orchestration and decision-making.

Specifications

Python 3.10+ (3.11 recommended), & Docker Compose for containerized deployment, MongoDB instance for checkpointing, API keys for Anthropic/Google GenAI when using those providers, network access to target systems, recommended runtime: 2+ CPU cores and 4+ GB RAM (higher for heavy LLM/parallel scans), FastAPI default port (uvicorn/8000) for API/SSE endpoints. Docker

Keywords

Automated penetration testing, Agent-based orchestration, Checkpointed, resumable

AstroRoute: A Real-Time Campus Bus Tracking and Route Management System

Group Members

S.M Abdullah Shabbir (27075)
Ibrahim Khalil Ahmad (27003)
Abdur Rehman Nasir (27067)

Supervisors

Supervisor (PI): Dr. Muhammad Saeed
Industry Advisor: Muhammad Saleem
Industry Company: Digital Information Systems (Pvt) Ltd.
Industry Mentor: Umair Ahmad

Knowledge Area: Software Development

Project Overview

AstroRoute is a web-based bus tracking and route management platform designed for educational institutions. It provides real-time GPS tracking of campus buses across student, driver, and administrator interfaces, enabling students to view live bus locations, drivers to conduct and manage trips, and administrators to oversee routes, vehicles, stops, and personnel. The system includes a live interactive map, boarding event logging, analytics dashboards, and route planning tools, delivered through a Progressive Web App with real-time updates via WebSocket communication. By centralizing transport operations and improving visibility for all users, AstroRoute aims to make campus bus services more organized, transparent, and efficient.

Technologies Used

Python, Flask, PostgreSQL, Socket.IO, JWT, bcrypt, Leaflet.js, HTML, CSS, JavaScript

Specifications

The backend of AstroRoute was built using Python 3.11 and Flask REST APIs, with PostgreSQL used to store and manage the system data across 9 relational tables. For real-time bus location updates, the system uses Socket.IO over WebSocket communication. Login and user security were handled through JWT-based authentication and bcrypt password hashing. The map feature was developed using Leaflet.js with OpenStreetMap tiles. AstroRoute was designed as a Progressive Web App, so it can be used on modern desktop and mobile browsers without requiring a separate mobile application.

Methodology

AstroRoute was developed through an iterative and practical development approach. We first identified the main users of the system, including students, drivers, and administrators, and then planned the features required for each role. The project was built step by step, starting with the basic structure and then adding modules such as login, route management, live bus tracking, trip management, and admin dashboards. After developing each feature, we tested it, improved it where needed, and then integrated it with the rest of the system. The frontend and backend were connected through APIs, while real-time bus location updates were handled using WebSocket communication. This approach helped us improve the system gradually and ensure that the final application was functional, organized, and easy to use.

Keywords

Bus Tracking, Route Management, Real-Time GPS Tracking

Reimagining University Enrollment: A Preference-Driven Course Allocation Framework for IBA

Group Members

Azka Aqeel (27068)
Rohan Riaz (26916)
Muhammad Hammad Khan (26917)

Supervisors

Supervisor (PI): Dr. Imran Rauf

Knowledge Area: Software Development

Project Overview

Currently, the Institute of Business Administration (IBA), Karachi relies on a first-come, first-serve (FCFS) course enrollment system that creates systemic inequities, overwhelming administrative burden, and significant student stress. Students with slower internet connections or those unable to register at the precise opening moment are systematically disadvantaged, regardless of their academic needs. Our final year project proposes a preference-driven course allocation framework that replaces the FCFS model with an ordinal preference submission system. Students rank up to ten course sections during a submission window and a priority scoring model which is composed of four weighted components: relevancy, standing, starvation, and rank, quantifies each student's priority score on every requested section. Two allocation algorithms were explored: Approximate Competitive Equilibrium from Equal Incomes (A-CEEI), adapted from the Wharton CourseMatch system, and the Stable Matching via Deferred Acceptance algorithm. Both were evaluated across eight metrics including feasibility, blocking pairs, average preference rank, top-choice satisfaction percentages, fully allocated student count, average courses per student, fairness variance, and runtime. Stable Matching outperformed A-CEEI across every dimension, producing zero constraint violations, zero blocking pairs, and completing in approximately 21 seconds versus 451 seconds. Following allocation, a facilitation week module allows students to add, drop, or swap sections under server-enforced validation covering capacity limits, schedule conflicts, prerequisite requirements, and course load boundaries. The system is built using React with TypeScript on the frontend, Python Flask on the backend, and MySQL for persistent storage, with an integrated hybrid Retrieval-Augmented Generation chatbot serving as a student support assistant.

Methodology

The final implementation uses the Deferred Acceptance (DA) algorithm as the core allocation engine. Before allocation begins:

Students submit ranked course preference lists during the submission window.

Preference lists are locked by the administrator after the deadline.

The system computes and freezes all priority scores to ensure a fixed ordering throughout the allocation process.

The allocation engine operates iteratively as follows:

Students propose to their highest preferred remaining section.

Sections tentatively retain the highest-priority students up to capacity.

Lower-priority applicants are rejected and continue proposing to their next preferred section.

Proposals that would create:

Timetable conflicts,

Duplicate course enrollments, or

Course-load violations are automatically skipped to maintain feasibility.

Student priority is determined using a weighted composite score consisting of four factors:

Relevancy: measures how closely a student's academic history aligns with the requested course.

Academic Standing: rewards senior students with higher priority.

Starvation: increases priority for students repeatedly been denied the same course in previous semesters.

Preference Rank: reflects the ordering of the student's submitted preferences.

To prevent long-term disadvantage, the system performs a post-allocation starvation counter update, where denial counters are incremented for every requested course a student failed to obtain. These counters persist across semesters and directly influence future allocation runs.

In cases where students receive identical composite scores, a single lottery-based tie-breaking mechanism is applied before the allocation begins to establish a strict total ordering.

Additional system components include:

A Facilitation Week module supporting add, drop, and swap operations under enforced validation constraints.

A hybrid RAG-based chatbot that assists students with allocation-related queries.

Testing included:

Unit-level validation of individual allocation engine components.

End-to-end evaluation using seed datasets to verify stability, feasibility, fairness, and performance.

Reimagining University Enrollment: A Preference-Driven Course Allocation Framework for IBA

Technologies Used

Python, Flask, Flask-CORS, React, Vite, JavaScript, HTML, CSS, TailwindCSS, Material-UI, Radix UI, Node.js, MySQL, SQLAlchemy, PyMySQL, bcrypt, cryptography, ChromaDB, FAISS, Sentence-Transformers, rank-bm25, Groq API, BeautifulSoup4, lxml, NumPy, Requests, Recharts, Framer Motion, React Router, React Hook Form, REST APIs, Git, GitHub, npm, Pytest, Virtual Environments (venv)

Keywords

Preference-Driven Course Allocation, Stable Matching via Deferred Acceptance, Approximate Competitive Equilibrium from Equal Incomes (A-CEEI)

Specifications

Hardware: Multi-core CPU, minimum 8GB RAM

Software/Infrastructure: MySQL 8.0+, Node.js, Python 3.10+

Frontend developed using React and Vite with TailwindCSS, Material-UI, Radix UI, HTML, CSS, Recharts, Framer Motion, React Router, and React Hook Form

Backend developed using Flask, Flask-CORS, SQLAlchemy ORM, PyMySQL, bcrypt, cryptography, and MySQL database integration through REST APIs

AI chatbot/recommendation service implemented using Sentence-Transformers, ChromaDB, FAISS, BM25 ranking, Groq API (llama-3.3-70b-versatile), BeautifulSoup4, lxml, NumPy, and Requests

Frontend dependencies installed using npm install from package.json

Backend and chatbot dependencies installed using Python requirements.txt files inside virtual environments (venv and botenv)

Database initialized using App/backend2/reset_db_data.bat, which automates database creation, data loading, and password hashing/setup processes

Allocation algorithm implemented using native Python libraries integrated with SQLAlchemy models

Version control and collaboration managed using Git and GitHub

Backend testing performed using Pytest

Clinical Copilot Using Knowledge Graphs

Group Members

Qamar Raza (27140)
Muddasir Javed (24463)
Abdullah Khalid (26969)

Supervisors

Supervisor (PI): Maria Rahim
Industry Company: Folio3
Industry Mentor: Irtiza Hussain

Knowledge Area: Artificial Intelligence based systems

Project Overview

Clinical Copilot is an all-in-one ecosystem designed to enhance clinical workflow with the help of Artificial Intelligence. The system eases and assists doctors' workflow by helping them manage their clinical documentations, diagnostic support, and patient data management. It sits at the heart of the doctor's room, transcribing real time doctor-patient conversation in both English and Urdu languages, and then intelligently extracts facts from the conversation. This helps identify possible misdiagnosis between patient medicines, past illnesses, or even past checkups against the current conversation and then notifies the doctor in real time to prevent any future medical issues. This feature is backed by knowledge graph, a RAG-based approach to provide better accuracy and reasoning, helping the system understand the complex relationships between medications, diseases and patient reports to generate accurate alerts. This is supported by the fact that patients can also upload their medical textual based reports which are processed through OCR and securely ingested into the patient's database, helping the system for future analysis and diagnostic alerts. The system also allows doctors to automatically generate After Visit Summary (AVS) and customizable SOAP notes tailored to doctor's preferences. It also provides an integrated conversational assistant, "Shifa Scribe," which acts as an intelligent interface connected to the entire clinical ecosystem, enabling doctors to retrieve patient insights, summaries, historical data, and contextual recommendations through natural language interaction. By combining speech processing, natural language understanding, OCR, knowledge graphs, and AI driven clinical reasoning, the proposed Clinical Copilot project aims to reduce administrative burden, improve documentation efficiency, support early detection of diagnostic inconsistencies, and enhance continuity of patient care while maintaining HIPAA compliance data management for clinics and hospitals.

Methodology

We identified major pain points in clinical workflows, including excessive documentation time, fragmented patient records, limited contextual diagnostic support, and inefficient data retrieval. Based on these challenges, the platform architecture was divided into interoperable modules, utilizing a modular full-stack methodology to integrate real-time transcription, RAG, Agentic Workflow-based reasoning, and automated clinical documentation into a unified workflow.

A patient data processing pipeline was developed to consolidate medical records, laboratory reports, and prescription data into structured representations that support efficient querying and analysis. During clinical consultations, a real-time audio processing module captured doctor-patient conversations and generated transcriptions using voice activity detection, speaker diarization, and segmentation techniques was added. Information extracted from these conversations was utilized to generate clinical alerts, SOAP notes, and After Visit Summaries.

To support clinician decision-making, a hybrid retrieval framework combines structured database queries with semantic vector-based search, enabling access to both numerical patient data and contextual medical records. Retrieved information was then used to provide grounded responses through a clinician-facing copilot interface called "Shifa-Scribe".

A knowledge graph was incorporated to model relationships between diseases, medications, allergies, symptoms, and patient history. This enabled contextual reasoning for tasks such as drug-allergy conflict detection, drug-disease interaction analysis, and alternative medication recommendations. Finally, schema-constrained generation and validation techniques were employed to ensure consistent documentation outputs, which were validated, formatted, and exported as standardized clinical reports. The overall system was evaluated based on retrieval performance, tool-selection accuracy, and documentation quality.

Clinical Copilot Using Knowledge Graphs

Technologies Used

TypeScript, Python, Next.js, Tailwind CSS, shadcn/ui, Prisma, PostgreSQL, FastAPI, OpenAI Whisper Large-V3, PyanNote, Pinecone, Neo4j, Zod, Clerk, Qwen 3.5 9B, GLM-OCR

Keywords

Clinical Copilot - using knowledge Graph, Natural language Processing (NLP), Agentic Healthcare AI

Specifications

The platform is developed using Next.js and Python. To maintain strict HIPAA compliance and zero data leakage, all models run entirely locally using the vLLM inference engine, currently hosted on our own isolated cloud environment using Modal. The frontend and backend is currently hosted on vercel and a python transcription pipeline hosted on Modal, however the system is made flexible as to be hosted anywhere given that these requirements are met:

Full Stack Framework: Next.js
Programming Languages: Python3, JavaScript, TypeScript
Database Systems: PostgreSQL for SQL DB, Pinecone for Vector DB, Neo4j for Graph DB
Authentication & User Management: Clerk
Processor: Any modern dual-core or quad-core processor
RAM: Minimum 8 GB, Storage: Minimum 16 GB
Microphone and Internet Connectivity required
GPU: NVIDIA CUDA-enabled GPU

4see: Foresee your seats before booking.

Group Members

Kanza Saud (25226)
Laiba Sajid (30502)
Aleena Meraj (27034)

Supervisors

Supervisor (PI): Dr. Syed Ali Raza
Industry Advisor: Adeen Atif
Industry Company: Ticketwala

Knowledge Area: Software Development

Project Overview

4See is an interactive event booking and venue management platform developed to solve the problem of “blind ticketing,” where customers purchase tickets without being able to properly visualize the venue map, seating arrangement, or seat quality i.e., if it is accessible or damaged etc., beforehand. Traditional ticketing systems in Pakistan often rely on simple ticket lists or manually assigned categories, which creates confusion for attendees and increases management difficulties for organizers.

The primary objective of this project is to provide a smart, visual (2D), and user-friendly ticketing experience that benefits both event organizers and customers. The system enables organizers to create and manage venues digitally by uploading venue floor plans, defining sections, detecting seats with above 90% precision and an average of 95.7% precision for geometric and non-geometric seats detected incrementally. The organizers can even start from scratch by adding seats in straight or curved rows. The platform also allows assigning pricing tiers, row labels, and managing events through an integrated dashboard.

At the same time, attendees can browse events, explore interactive venue maps, view seat availability in real time, and reserve seats directly from a visual layout instead of relying on text-based ticket categories. A major contribution of the project is the integration of venue-map processing with real-time seat reservation and booking management, helping reduce manual configuration effort while improving booking accuracy and transparency. The platform also introduces tools that support scalability for large events, including simultaneous seat reservations, category-based pricing, and organizer-focused venue management utilities allowing dynamic pricing for the same venue but different events. By combining interactive visualization with smart seat-selection features, the project enhances customer confidence during ticket purchases and provides organizers with a more efficient, fast, and modern approach to event management within the growing digital entertainment ecosystem.

Technologies Used

React, TypeScript, Vite, Tailwind CSS, Shadcn UI, Radix UI, React Router, Node.js, Express.js, MySQL, Python, FastAPI, Uvicorn, OpenCV, Pillow/PIL, Redis, Multer, Vitest, Zod, PNPM, Git.

Specifications

The system requires Node.js v16+, Python 3.8+, MySQL Server, Redis for seat locking, and a modern web browser. It uses a React/Vite frontend, an Express.js REST API backend, and a FastAPI Python service for smart seat/map image processing. Key APIs include authentication, event management, venue creation, seat layout editing, booking, map image upload, and real-time seat availability. MySQL stores users, venues, floors, sections, seats, categories, events, and bookings, while Multer handles file uploads. No specialized hardware is required; a standard laptop/PC with at least 8 GB RAM is recommended for development. Keywords: Interactive Seat Maps, Smart Seat Detection, Event Booking

Methodology

The project followed an iterative full-stack development methodology, beginning with requirement analysis for organizers, attendees, and the industry partner that will use the codebase in its business operations. Development was carried out in repeated cycles of planning, implementation, testing, feedback, and refinement. Frequent meetings with the industry supervisor were conducted throughout the project to validate requirements, review progress, identify practical business needs, and ensure that the product remained aligned with functional expectations. The system was designed in modular components, including authentication, venue creation, event management, seat mapping, smart seat detection, and booking. The frontend was developed using React and TypeScript with reusable UI components, while the backend APIs were built with Express and MySQL to manage users, venues, events, seats, and bookings. Smart seat detection was integrated through a Python-based image processing service, allowing uploaded venue maps to be converted into editable interactive seat layouts. Each feature was implemented, reviewed, tested, and improved in stages based on supervisor feedback and usability considerations, ensuring that the final system supports real industry workflows, maintains data consistency, enables real-time seat availability, and provides a smooth booking experience.

Keywords

Interactive Seat Maps, Smart Seat Detection, Event Booking

Group Members

Ikhlas Khan (27096)
Musab Suhail (26923)
Zainab Hasan (27127)

Supervisors

Supervisor (PI): Sir Imran Rauf

Knowledge Area: Computer Vision

Project Overview

MatchIQ is an AI powered football analytics platform designed to make professional match analysis accessible to lower league clubs, academies, and grassroots teams, especially in countries like Pakistan where advanced sports analytics tools are often too expensive. The platform takes a single match video as input and automatically processes it using computer vision and machine learning techniques to track players and the ball, identify teams through jersey colours, and map player movements onto a real pitch. It then generates useful insights such as player speed, distance covered, sprint counts, fatigue levels, goal scoring probability, and match outcome predictions. The system is powered by a FastAPI backend and presented through an interactive Next.js dashboard that includes live video streaming, a real time pitch minimap, and detailed post match analytics. MatchIQ aims to remove the need for expensive hardware, manual tagging, and costly data subscriptions by providing affordable and automated football analysis using only a consumer GPU and a single video file.

Technologies Used

Python, FastAPI, Uvicorn, OpenCV, PyAV, PyTorch, CUDA, YOLOv8, Roboflow Inference, OCSORTTracker, supervision, sports (Roboflow), SiglipEncoder, KMeans, NumPy, pandas, SciPy, WebSockets, MJPEG, Next.js, React, TypeScript, Tailwind CSS, Recharts, Framer Motion, Lucide React, WSL2, NVIDIA CUDA 12.1

Methodology

MatchIQ was developed using an iterative and pipeline based approach that combined computer vision research, backend development, and frontend web technologies. The project started by identifying the main analytical features required, including player tracking, movement analysis, fatigue estimation, goal probability, and match outcome prediction. These were divided into separate processing stages where each stage used the CSV output from the previous one, making the system easier to test and improve over time. For the computer vision pipeline, pre trained YOLOv8 models from Roboflow were tested on football footage, while homographic projection was used to convert player positions from pixels into real world pitch coordinates. The tracking system was later improved by replacing ByteTrack with OCSORTTracker to reduce identity switching during player occlusion. Team classification was achieved using KMeans clustering on jersey colours without requiring labelled training data. The backend was developed with FastAPI and used MJPEG streaming and WebSockets for real time updates. The frontend was built in Next.js with reusable components and fallback mock data to ensure the interface remained functional during development. Git version control was used throughout the project to manage progress and track feature development.

Keywords

Football Analytics, Player Tracking, Affordable Sports AI

Specifications:

Hardware

NVIDIA GeForce GTX 1660 Super (6 GB VRAM) or similar consumer GPU; standard laptop or desktop system supported. No specialised sports capture hardware is required. Input works with a single static wide angle elevated camera video.

Operating System and Environment

Windows 10/11 with WSL2 running Ubuntu. Python 3.12 virtual environment located at `/mnt/d/MatchIQ/venv/`.

GPU and CUDA Support

CUDA 12.1 and cuDNN 9.1 are used for GPU acceleration. PyTorch 2.5.1 with CUDA 12.1 and Torchvision 0.20.1 are required. ONNX Runtime 1.21.1 provides full CPU fallback when GPU is not available.

Backend

FastAPI 0.115.14 with Uvicorn 0.44.0 running on `http://localhost:8000`. Supports MJPEG video streaming, WebSocket real time updates, HTTP Range video seeking, and ZIP download of CSV outputs.

Computer Vision and ML Libraries

OpenCV contrib 4.10.0.84, PyAV 14.2.0, Roboflow Inference 1.2.1 (CPU and GPU versions), OCSORTTracker, supervision, and sports library from Roboflow (installed via Git).

Data Processing

NumPy 1.26.4, Pandas 2.3.3, SciPy 1.17.1.

External APIs

Roboflow Inference API for YOLOv8 models football player ball detection and pitch keypoints detection. Hugging Face SiglipEncoder for jersey feature extraction. API keys stored securely in `.env` file.

Frontend

Next.js 16.2.3, React 19, TypeScript 5, Tailwind CSS 4. Runs on `http://localhost:3000`.

Input Specification

Single continuous MP4 football match video. Requires static or near static elevated wide angle camera view. Exactly two teams with clearly distinguishable jersey colours.

Enhancing Wazuh SIEM with STIX/TAXII Threat Intelligence and Machine Learning Predictive Analytics

Group Members

Syed Saadan Uddin (26980)
Abdullah Rehman (27074)
Saim Wajid (26907)

Supervisors

Supervisor (PI): Dr. Faisal Iradat

Knowledge Area: Security

Project Overview

The overall purpose of Wazuh-TI is to address the pervasive issue of alert fatigue in modern Security Operations Centers (SOCs) by transforming raw, disjointed security logs into prioritized, actionable intelligence. The primary objective is to seamlessly integrate standardized external threat intelligence, utilizing the STIX 2.1 data model and TAXII protocol alongside global OSINT feeds, directly into the Wazuh SIEM ecosystem. By bridging this gap, the project's main contribution is the development of a proactive, dual-engine analytical framework: it introduces a custom Machine Learning pipeline to automatically calculate probabilistic risk scores and map threat behaviors to the MITRE ATT&CK framework, combined with a Conversational AI Analyst powered by Google Gemini that enables security teams to execute complex forensic investigations using natural language. Ultimately, this integration shifts security operations from reactive monitoring to predictive threat hunting, significantly reducing the time required to triage and mitigate critical cyber threats.

Technologies Used

Python, JavaScript, FastAPI, SQLAlchemy, React, Vite, Tailwind CSS, Recharts, Scikit-learn, Google Gemini AI, Wazuh SIEM, Docker, Docker Compose, SQLite, STIX 2.1, TAXII 2.1, AlienVault OTX, AbuseIPDB, ThreatFox

Keywords

Threat Intelligence, SIEM (Security Information and Event Management), Predictive Analytics

Methodology

The development of the threat intelligence platform followed an iterative Agile methodology combined with a modular architectural approach to enable the continuous integration and testing of complex components. The lifecycle began with establishing a microservices architecture using FastAPI for high performance asynchronous data processing, while Docker was utilized to containerize the environment and ensure parity across development and deployment stages. Following this foundation, the intelligence ingestion engine was engineered to query TAXII servers and external REST APIs, allowing incoming threat data to be systematically parsed, standardized into the STIX data model, deduplicated, and securely persisted using SQLAlchemy. Analytical capabilities were subsequently developed through a data driven machine learning lifecycle, where a Random Forest Classifier was trained to calculate probabilistic risk scores based on host criticality, and the Google Gemini SDK was integrated to empower a conversational AI analyst with natural language forensic capabilities. Finally, the presentation layer was built using React and Tailwind CSS with a strong focus on user experience, and the entire integrated system was subjected to rigorous validation using simulated attack scenarios and mock agents to verify the accuracy of machine learning predictions and the overall stability of the real time data synchronization pipeline.

Enhancing Wazuh SIEM with STIX/TAXII Threat Intelligence and Machine Learning Predictive Analytics

Specifications

Hardware & System Requirements:

Processor: Multi-core CPU (4 cores or higher recommended to handle concurrent data processing and machine learning inference).

Memory (RAM): Minimum 8 GB (16 GB highly recommended to smoothly run the Wazuh Manager, Docker containers, and the React frontend simultaneously).

Storage: Minimum 20 GB of available SSD storage for the relational database, Docker images, and threat intelligence logs.

Operating System: Platform-agnostic (Windows, macOS, or Linux), provided Docker and Docker Compose are natively supported.

Software & Core Dependencies:

Containerization: Docker Engine (v24+) and Docker Compose for deployment.

Backend Environment: Python 3.11+, utilizing the FastAPI framework and SQLAlchemy ORM for asynchronous API routing and database management.

Frontend Environment: Node.js ecosystem utilizing React 19, Vite, and Tailwind CSS.

SIEM Infrastructure: An active Wazuh Manager deployment (v4.x) configured to expose local logs and rulesets.

Database: SQLite (default for development) or PostgreSQL (for production scalability) to persist the STIX/TAXII models and ML predictions.

External APIs & Integrations:

TAXII 2.1 Server Endpoints: Integrated to systematically poll and ingest standardized STIX 2.1 threat bundles and adversary campaigns.

AlienVault OTX API: Used to fetch community-driven threat pulses and indicators of compromise (IOCs).

AbuseIPDB API: Interrogated for real-time IP address reputation scoring and contextual data.

ThreatFox & URLhaus APIs: Utilized to automatically ingest high-confidence malware hashes and malicious URLs into the enrichment pipeline.

Padel Pro

Group Members

Sharjeel Chandna (24384)
Dawood Naeem (18720)
Omer Shahid (25129)
Abaan Noor (24911)

Supervisors

Supervisor (PI): Tasbiha Fatima

Knowledge Area: Software Development

Project Overview

Padel is one of the fastest growing sports worldwide, and its popularity in Pakistan, especially in Karachi, continues to rise with new courts being built regularly. Despite this growth, booking a court remains inefficient, with players relying on scattered WhatsApp groups and having no central view of availability or pricing. Finding a competitive game is just as frustrating, as players struggle to find opponents at their skill level while dealing with buried messages, unreliable self-proclaimed skill levels, and coordination that falls apart over texts. Padel Pro facilitates this growing community through a centralized web platform where players and club owners can create accounts. Owners add their courts, and players directly book timeslots from their PCs, easily comparing options and confirming bookings without back-and-forth. The platform's main draw is skill-based matchmaking, which incorporates player ratings and reviews from past opponents to create a much more consistent and streamlined competitive environment. Padel Pro also features an integrated marketplace for buying and selling equipment, a community hub for discussions, and a chatbot for FAQs, turning the disorganization of WhatsApp into an easy to navigate, competitive ecosystem where everything padel related exists in one place.

Technologies Used

React.js & JavaScript: The main programming language and framework used to build the interactive web pages, buttons, and user interfaces.

Tailwind CSS: The design tool used to style the application and make it look clean, modern, and mobile friendly.

Firestore: our "behind the scenes" engine. It handles user logins safely (Authentication) and stores all your data like court bookings, player profiles, and marketplace items (Firestore Database).

Vite: The engine that runs behind the scenes during development to compile your code incredibly fast.
Vercel: The hosting platform where your live website is currently deployed, allowing anyone to access it via your `padelproapp.vercel.app` link.

Methodology

The development process was carried out sequentially and component-by-component, where we worked on one large subsystem at a time and not on all at once. Firstly, we designed the wireframes for all screens and flows (bookings, matchmaking, market, and profiles) in order to create an understanding of the navigation and information architecture of the application without having to write code. Once the wireframes were finalized, we then worked on the frontend development of the screens and linking them together with dummy data in order to validate the navigation and user experience. Once we were satisfied with our frontend architecture, we finally turned our attention to the backend and implemented the microservices, databases, and API endpoints required to power the frontend screens we designed earlier.

Keywords

Sports Platform, Competitive Matchmaking, Court Reservation System

Specifications:

1) Client System Requirements:

- Browser: Any modern web browser (Google Chrome)
- Device: Responsive web application accessible via mobile phones, tablets, and desktop computers.
- Network: An active internet connection.

2) Cloud Infrastructure & Hardware:

- Serverless Architecture: No physical hardware or local servers were required.
- Frontend Hosting: Deployed on Vercel cloud infrastructure for fast, global delivery.
- Backend Database: Powered by Google Cloud's Firebase infrastructure, ensuring high availability and scalability.

3) APIs & Third Party Services:

- Firebase Authentication API: Used to securely handle user registration, login, and session management.
- Firebase Cloud Firestore API: A real-time database API used to instantly sync court availability, matchmaking data, and marketplace listings.
- Firebase Cloud Storage API: Used to securely upload, store, and retrieve media files, such as marketplace equipment photos and user profile pictures.

AI driven road Quality Assessment in Rural Areas using Satellite Images

Group Members

Isht dev (27164)
Ghulam Murtaza Tunio (27150)
Hammadullah Muazam (27033)

Supervisors

Supervisor (PI): Tasbiha Fatima
Co-PI: Abdul Wahab
Industry Advisor: Arbaz Khan
Industry Company: Folio3
Industry Mentor: Muhammad Nasiruddin Mahmood

Knowledge Area: Artificial Intelligence based systems

Project Overview

Roadlytics is a road-quality assessment platform that combines deep learning over Sentinel-2 satellite imagery with a deployed web application for analysts working on rural infrastructure in Sindh, Pakistan. The analytical pipeline operates in two stages. The first performs binary semantic segmentation of road pixels using a DeepLabV3+ architecture with an SE-ResNeXt-101-32x4d encoder; through six sequential experiments, test IoU was raised from a 31.4% baseline to 66.2%, satisfying the project's target band of 65–75%. The single most consequential intervention was a training-time 3×3 morphological dilation of one-pixel road labels, which addressed a topological degeneracy in the ground truth more effectively than shallower encoders or alternative decoder families such as U-Net. The second stage classifies each road pixel as Good, Unpaved, or Damaged using an EfficientNet-B2 trained on K-Means pseudo-labels derived from spectral features, achieving 83% accuracy and a 0.83 macro F1 on held-out test patches. The pipeline is delivered as a production web application in which analysts upload georeferenced imagery, run jobs on in-process inference workers, and inspect results as interactive map overlays alongside connectivity metrics and downloadable reports. A retrieval-augmented assistant, built on a persistent vector database with deterministic local SHA-1 hash embeddings and a Gemini generation backend, sits over job metadata, artifact manifests, analytics snapshots, and report text to produce grounded summaries, failure diagnostics, and report drafts while explicitly refraining from unsupported claims about field truth, pavement safety, or repair cost.

Technologies Used

The platform is implemented as a Python and TypeScript codebase spanning a browser client and a backend service. The frontend uses Next.js 15 (App Router), React 19, TypeScript, Leaflet, and react-leaflet, and is built as a static export served by nginx. The backend is implemented in FastAPI with Uvicorn, running on Python 3.11, with Pydantic, an in-process async worker queue, and SQLite (Python standard-library sqlite3) for job, artifact, event, and analytics persistence. The application has no authentication layer in the current no-auth MVP. Object storage is handled by Azure Blob Storage via the azure-storage-blob SDK, with a local filesystem fallback when no Azure connection string is configured. The inference pipeline runs in the same container as the backend and uses PyTorch, segmentation_models_pytorch, rasterio, GDAL, NumPy, OpenCV, scikit-learn, albumentations, scipy, geopandas, shapely, pyproj, fiona, and networkx. The deep-learning models are DeepLabV3+ with an SE-ResNeXt-101-32x4d encoder for road segmentation, and EfficientNet-B2 (torchvision) modified for four-channel input for condition classification; both were trained on four-band Sentinel-2 Level-2 GeoTIFF tiles covering the Blue, Green, Red, and Near-Infrared bands. A PakOSM segmentation alternative rasterizes Pakistan OpenStreetMap road geometries onto the GeoTIFF grid without requiring learned weights. Connectivity analytics are computed with networkx. The retrieval-augmented assistant is built on ChromaDB for vector storage, a deterministic local SHA-1 hash embedding function, and the Google Gemini API accessed through the google-genai SDK (model: gemini-2.5-flash), with an extractive fallback path when no API key is configured. Supporting tooling includes npm, ESLint, TypeScript, ruff, mypy, pytest with pytest-asyncio, and Docker Compose for orchestration. The map view renders user-toggleable raster overlays and GeoJSON layers above a fixed OpenStreetMap basemap via Leaflet, with tile endpoints served by rio-tiler.

AI driven road Quality Assessment in Rural Areas using Satellite Images

Methodology

Development proceeded along two parallel tracks. The machine-learning track adopted sequential ablation as its core methodology: beginning with a U-Net/ResNet-34 baseline at 31.4% test IoU, each subsequent experiment isolated a single variable, such as batch size, label representation, encoder depth, or decoder family, so that observed improvements could be attributed unambiguously to that change. The severe foreground-background imbalance characteristic of road segmentation, in which road pixels constitute under two percent of the image, was addressed through stratified tile filtering, a combined Focal-Dice loss, and a reduced prediction threshold. The most significant intervention was a 3×3 morphological dilation applied to the one-pixel road labels within the data loader, which provided the network with a topology-preserving learning target and lifted test IoU to 66.2% in a single experiment. The final deployed segmentation model is DeepLabV3+ with an SE-ResNeXt-101-32x4d encoder, which superseded the U-Net experiments and runs sliding-window inference over 1024×1024 tiles with a 512-pixel stride. A PakOSM alternative directly rasterizes Pakistan OpenStreetMap road geometries onto the input GeoTIFF grid as a no-weights segmentation option. For road-condition assessment, an unsupervised-to-supervised strategy was adopted, in which K-Means clustering over spectral features produced pseudo-labels at scale and an EfficientNet-B2 modified for four-channel input was trained on 32×32 patches sampled around road pixels with class-balanced batching. The software-engineering track followed a contract-driven methodology in which the system was decomposed into a sequence of self-contained modules, each with explicit dependencies, file boundaries, and acceptance criteria, enabling parallel development without breaking inter-module interfaces. The runtime architecture centres on a FastAPI backend with an in-process async worker queue; job submissions are placed on the queue, progress is persisted in SQLite, and the browser polls the `/jobs/{id}` endpoint for status updates. When Azure Blob Storage is configured, client uploads proceed directly to object storage via SAS tokens, bypassing the application server; otherwise, uploads proxy through the backend to local storage.

Specifications:

Model training was conducted on an NVIDIA A100 40 GB GPU through Google Colab Pro+, with random seed 42 fixed across NumPy, PyTorch, and Python to support reproducibility. Inputs are four-band Sentinel-2 Level-2 GeoTIFFs at 10 metre ground sampling distance covering the Blue, Green, Red, and Near-Infrared bands, accepted at sizes up to two gigabytes per upload. The segmentation pipeline tiles scenes into 1024×1024 patches with a 512-pixel stride using a sliding-window approach, running DeepLabV3+ with an SE-ResNeXt-101-32x4d encoder at a prediction threshold of 0.5. The condition classifier uses EfficientNet-B2 (torchvision) with its first convolutional layer modified to accept four channels, the additional channel initialized from the mean of the pretrained RGB filters; inference extracts 32×32 patches centered on every road pixel and processes them in batches of 256. Inference runs in-process within the FastAPI backend container via an async worker service that re-queues recoverable jobs on startup. The public API is a REST interface implemented in FastAPI with Pydantic schema validation and standardized error responses; there is no authentication layer in the current MVP. Progress is delivered by HTTP polling against the `/jobs/{id}` endpoint. Persistent state resides in SQLite via Python's standard-library sqlite3, with separate tables for jobs, artifacts, job events, and analytics snapshots. Raster artifacts and reports are stored in Azure Blob Storage, with SAS tokens issued for direct client uploads and time-limited downloads; a local filesystem backend is used when no Azure connection string is configured. The retrieval-augmented assistant uses a persistent ChromaDB collection with a deterministic SHA-1 hash embedding function and a Gemini generation backend (gemini-2.5-flash), with an extractive fallback when no API key is configured. The system is packaged as two Docker containers orchestrated by Docker Compose: the backend service exposing port 8000, and the frontend static site served by nginx on port 80.

Keywords

Computer Vision, Geospatial Analysis, MLOps, Generative AI, Deep Learning

Benchmarking VQE: From Pipeline Analysis to Adaptive Ansatz Optimization

Group Members

Muhammad Hashir Ilyas (26972)
Muhammad Imad Raza (26953)

Supervisors

Supervisor (PI): Dr. Jibran Rashid

Knowledge Area: Quantum Computing / Quantum Chemistry

Project Overview

The Variational Quantum Eigensolver (VQE) is a hybrid quantum-classical algorithm tailored to the Noisy Intermediate-Scale Quantum (NISQ) era, with its principal application in quantum chemistry: estimating molecular ground-state energies, which underpins the understanding of chemical properties. While exact classical methods such as Full Configuration Interaction scale exponentially with system size, VQE leverages variational principles and parameterized quantum circuits to approximate these quantities with potential advantages on suitable hardware. This project began as a comprehensive benchmarking study of the full VQE pipeline across basis sets, fermion-to-qubit mappings, ansatzes, optimizers, and backends. Through initial experimentation and consultation with researchers in the field, it became clear that optimal pipeline configurations are molecule-specific and do not generalize, making broad benchmarking a less productive direction than anticipated. The project therefore pivoted to a more focused contribution: investigating the practical resource efficiency of ADAPT-VQE on NISQ hardware by combining multiple methods and advancements made on ADAPT-VQE, to demonstrate the practical viability of adaptive ansatz construction on real quantum hardware. Specifically focusing on integrating the Coupled Exchange Operator (CEO) pool with the Overlap-ADAPT-VQE selection criterion and the TETRIS-ADAPT-VQE framework for shallower circuit construction.

Technologies Used

Python, Qiskit, PennyLane, Jax, OpenMP

Specifications

We used the Qiskit and PennyLane API. The hardware requirements included a 64 core HPC with 128 GB RAM and 512 GB SSD

Keywords

Quantum Computing, Quantum Chemistry, Variational Algorithms

Methodology

The project was conducted in two stages reflecting its change in direction. In the first stage, a broad benchmarking pipeline was implemented and evaluated on the hydrogen molecule, sweeping combinations of basis sets, fermion-to-qubit mappings, ansatzes, and optimizers to identify high-performing configurations. This stage established the core infrastructure and produced baseline results, but also revealed a fundamental limitation: configurations that performed well on H₂ did not reliably transfer to other molecules, and no consistent general-purpose configuration emerged. Following discussions with researchers in the field and the project supervisor, the scope was narrowed to a focused investigation of hardware resource efficiency within the ADAPT-VQE framework. The second stage follows a three-configuration experimental design built around a shared Overlap-Vanishing Paired Compact Excitation Operator (OVP-CEO) pool, ensuring that performance differences are attributable solely to the selection criterion and ansatz construction strategy. The baseline configuration runs CEO-ADAPT-VQE with standard gradient-based selection. The second configuration replaces the selection criterion with the overlap-based approach of Feniou et al., running an initial phase that builds the ansatz by maximizing overlap with a pre-computed reference state before switching to energy gradient selection for the remaining iterations. The third configuration extends this by additionally applying the TETRIS framework, selecting multiple operators with disjoint qubit supports per iteration to reduce circuit depth without increasing CNOT count. All three configurations are evaluated on CH₄ and H₂O at stretched bond geometries using the LinAlg exact statevector simulator, with performance measured in terms of operators and CNOT gates required to reach chemical accuracy of 1.6 mHa from the FCI reference energy.

Organizational Analyst Bot

Group Members

Asna Farooqui (26918)
Saad Thaplawala (27172)
Uzair Nadeem (24928)

Supervisors

Supervisor (PI): Muhammad Zain Uddin
Co-PI: Muhammad Adil Saleem
Industry Advisor: Haris Aqeel
Industry Company: Folio3

Knowledge Area: Artificial Intelligence based systems

Project Overview

This project presents an AI-powered Business Intelligence and Analytics platform designed to enable intelligent querying, automated insight generation, and secure semantic analysis of enterprise data. The system supports data ingestion through PostgreSQL database connectors and Power BI (.pbix) file imports. To minimize production database load and improve performance, selected metadata and sampled records are fetched and stored locally using DuckDB for offline analytical processing.

The platform employs semantic modelling techniques to construct a semantic layer containing relationships, schema mappings, KPI definitions, and contextual business metadata. Retrieval-Augmented Generation (RAG) combined with vector databases is used to provide contextual understanding and improve Natural Language Processing (NLP)-based query interpretation. Users can interact with the system using natural language queries, which are converted into optimized SQL or DAX queries through NLP-to-SQL/DAX generation models.

The solution provides advanced analytics capabilities including executive KPI reporting, trend and seasonality analysis, anomaly detection, forecasting, and correlation analysis. Large Language Models generate explainable summaries and insights grounded in retrieved semantic context. The system can also automatically generate charts and lightweight dashboards based on user intent.

A local-first deployment architecture is adopted to ensure enterprise data privacy and security by keeping all sensitive organizational data within the client environment. The proposed system aims to simplify business analytics workflows, reduce dependency on manual dashboard creation, and enhance decision-making through AI-driven insights and semantic understanding of data.

Methodology

The project follows an AI-driven Business Intelligence and Analytics methodology focused on secure local data processing and semantic understanding of enterprise data. The system first ingests data either through direct MySQL database connectors or Power BI (.pbix) file imports. For SQL databases, metadata and selected top-N rows are fetched initially to reduce latency and avoid heavy production database load. Required tables and columns are identified using metadata-driven recommendation models, after which the selected data is stored locally in DuckDB for offline querying and processing.

The ingested data then undergoes preprocessing and cleaning before semantic modelling techniques are applied to create a semantic layer containing relationships, schemas, KPI definitions, column mappings, and contextual business meanings. For PBIX files, metadata extraction models are used to extract semantic and structural information similarly.

The semantic layer is integrated with a Retrieval-Augmented Generation (RAG) pipeline and vector database to enable natural language querying. User queries are embedded and matched against contextual metadata and business rules to optimize prompt generation and improve NLP-to-SQL/DAX conversion accuracy. Query generation models dynamically generate optimized SQL or DAX queries to retrieve relevant information.

The analytics engine performs KPI analysis, trend detection, anomaly detection, forecasting, and correlation analysis using statistical and machine learning techniques. Large Language Models generate explainable insights, summaries, and recommendations grounded in retrieved business context. The system can additionally generate charts, KPIs, and lightweight dashboards automatically based on the query intent.

The entire application is designed as a downloadable local deployment solution to ensure user data privacy and prevent sensitive enterprise data from leaving the client environment.

Organizational Analyst Bot

Technologies Used

FastAPI, Node.js, React, MySQL, DuckDB, Power BI (.pbix) Integration, SQL, DAX, NLP-to-SQL, NLP-to-DAX, Retrieval-Augmented Generation (RAG), Vector Database, Semantic Modelling, Metadata Extraction, Machine Learning Models, Large Language Models (LLMs), Pandas, NumPy, REST APIs, Embedding Models, Local Storage Processing, Data Cleaning & Preprocessing, Automated Query Generation, KPI Analytics, Trend & Anomaly Detection, Dashboard Visualization, Docker, Git, Cloud APIs, LM Studio

Keywords

AI-Powered Business Intelligence, Local-First Analytics, Intelligent Data Querying

Specifications

Hardware (development and testing): PC with 16 GB RAM and an NVIDIA GeForce RTX 3050 GPU.
Operating system: Developed on Windows; backend and frontend stacks are cross-platform where dependencies allow.

Backend runtime: Python 3 with FastAPI (REST API), served via Uvicorn (default port 8000 in this repository).

Frontend stack: Node.js; React 18 with TypeScript; Vite (build/dev server); Tailwind CSS; Radix UI / shadcn-style components.

Core databases and storage: PostgreSQL client (psycopg2) for connecting to external source databases; DuckDB for local ingested/analytical storage (~askdata.duckdb); file-based ChromaDB for vector/RAG indexes (optional LanceDB support in dependencies).

AI / NLP: Retrieval-Augmented Generation (RAG) using Sentence Transformers for embeddings (e.g. BGE-family models); Google Gemini API (~google-gemini) for LLM tasks; OpenAI-compatible HTTP clients for local LM Studio or NVIDIA NIM-style endpoints; hybrid retrieval includes BM25 (~rank-bm25).

Other notable backend libraries: Pandas/NumPy for data handling; scikit-learn, statsmodels, Prophet, pmdarima for analytics and forecasting; Plotly for visualization APIs; geopy for geolocation-related cleaning; cryptography for secure settings handling.
External connectivity: Network access needed when using cloud LLMs or remote PostgreSQL; optional offline/local models when configured (e.g. bundled embedding weights).

Automated Morphological Analysis and Classification of Neutrophils in Blood Smears

Group Members

Muhammad Annas Shaikh (26919)
Muhammad Haaris (27083)

Supervisors

Supervisor (PI): Dr. Rizwan Ahmed Khan
Co-PI: Dr. Rizwan Alam

Knowledge Area: Artificial Intelligence based systems

Project Overview

Automated white blood cell classification and morphological analysis in peripheral blood smears sits at the intersection of two critical unmet needs in modern healthcare: the inaccessibility of conventional hematology analyzers in resource-constrained clinical environments due to their prohibitive cost, and the well-documented failure of deep learning models to generalize across clinical sites due to variations in staining protocols and imaging hardware, a phenomenon known as covariate shift. This project directly addresses both challenges through a comprehensive pipeline spanning data curation, morphometric analysis, domain adaptation, and vision-language modelling. First, in collaboration with Quaid-i-Azam University, Islamabad, we curated a novel, high-quality peripheral blood smear dataset from local Pakistani clinical sources, featuring precise cell-wall and nucleus polygon annotations, segmentation masks, and morphological diversity across multiple stimulation conditions, filling a significant gap in regional representation in public hematology datasets. Second, we developed an automated cell annotation and morphometric framework that computes instance-wise Nucleus-to-Cytoplasm ratios from annotated microscopy images, reducing what previously demanded months of expert manual effort to a matter of hours. Third, we propose SER-DA-MMD, a parameter-efficient unsupervised domain adaptation framework that inserts a lightweight Squeeze-Excitation Residual adapter into a frozen backbone guided by Maximum Mean Discrepancy alignment, achieving substantial accuracy gains over direct transfer with no target-domain labels and no backbone retraining required. Finally, we fine-tuned a Vision-Language Model on cell morphology captions to enable natural language understanding of leukocyte structure, with a clinician-facing desktop application integrating all components currently under active development for point-of-care deployment.

Methodology

Our methodology follows a clear four-stage pipeline designed for real clinical impact. First, we curated a novel peripheral blood smear dataset from local Pakistani sources in collaboration with Quaid-i-Azam University, Islamabad. We collected microscopy images across three separate experiments under both Control and stimulated conditions using PMA and FMLP at varying concentrations. Expert hematologists meticulously annotated each image by drawing polygon boundaries around cell walls and nuclei, providing perfect ground-truth for training and evaluation. Second, building on these annotations, we built an automated cell annotation and NC-ratio computation framework. What previously demanded months of manual effort is now completed in few days with the tool, or in a few hours when assisted by our AI model. Third, to overcome the devastating performance drop caused by covariate shift across different clinical sites, we developed SER-DA-MMD, a parameter-efficient unsupervised domain adaptation framework. We keep the entire pre-trained backbone frozen and insert a lightweight trainable Squeeze-Excitation Residual adapter just before the classifier head. This adapter uses channel-wise attention to suppress stain-sensitive artefacts while amplifying morphologically informative structures. Training is guided by a hybrid loss that combines cross-entropy on labelled source data with Maximum Mean Discrepancy (MMD) alignment on unlabelled target data. For CNN architectures, we additionally apply Adaptive Batch Normalisation (AdaBN) as a post-training step. Fourth, we fine-tuned a Florence-style vision-language model on haematology-specific image-caption pairs, enabling the system to generate natural language descriptions of cell morphology. A desktop application integrating all modules is currently under active development for direct clinical deployment.

Automated Morphological Analysis and Classification of Neutrophils in Blood Smears

Technologies Used

PyTorch, timm, Hugging Face Transformers, Gemma Vision-Language Model, scikit-image, tifffile, GradCAM, Squeeze-and-Excitation Networks, Maximum Mean Discrepancy with multi-kernel RBF kernels, Adaptive Batch Normalization, ResNet50, VGG16-BN, InceptionV3, Vision Transformer ViT-Base, AdamW optimiser, NVIDIA Tesla T4 GPU, Kaggle

Keywords

Cellular Morphological Explainability, Domain Adaptation, Annotated Hematology Dataset Curation

Specifications

The system was trained on an NVIDIA Tesla T4 GPU across four architectures: VGG16-BN, ResNet50, InceptionV3, and ViT-Base. Three publicly available datasets were used, with PBC (Barcelona) serving as the labelled source and Raabin-WBC and AML Cytomorphology as unlabelled cross-site targets. Our custom Pakistani Local PBS dataset, curated with Quaid-i-Azam University across three experiments and multiple PMA/FMLP stimulation concentrations, currently comprises 1,573 hematologist-verified polygon-annotated images covering cell walls and nuclei, making it a first-of-its-kind growing regional resource. Our domain adaptation approach, SER-DA-MMD, uniquely keeps the entire backbone frozen while a lightweight adapter learns to suppress stain-sensitive artifacts and align feature distributions across clinical sites. This achieves up to +23.8 percentage points accuracy gain over direct transfer and closes the gap to full fine-tuning within approximately 5 percentage points, all without requiring a single target-domain label.

Intelligent Document Processing System

Group Members

Fatima Naeem (26933)
Juweriya Bint Nasir (26787)
Maheen Muhammad Rizwan (27118)

Supervisors

Supervisor (PI): Sir Rizwan Ahmed
Industry Advisor: Sir Hamza Usman Ghani
Industry Company: VentureDive

Knowledge Area: Artificial Intelligence based systems

Project Overview

Enterprise document processing still relies heavily on manual data entry, while many existing Intelligent Document Processing (IDP) systems are expensive, cloud-dependent, or unreliable for financial-grade auditing. This project presents FADE (Financial Agentic Document Extraction), an end-to-end intelligent document processing platform designed to convert heterogeneous business documents into validated, structured, and auditable data. The system follows a “fast by default, intelligent by exception” architecture. A deterministic high-speed extraction layer using OCR and spatial heuristics processes the majority of well-structured documents locally and efficiently, while a multimodal reconciliation layer is invoked only when validation checks detect inconsistencies such as missing values or decimal-slip errors. This framework guarantees financial accuracy, minimizes needless expenses related to AI inferences, and permits scalability. Audit logs are used to track each step of the procedure, guaranteeing that dependability and compliance are upheld. FADE has been created by trial and error. Early iterations with OCR + LLM have shown that local document extraction is possible, but there were certain limitations associated with flat text processing and vision-based only approaches for structured financial documents. This insight contributed to the design of the final two-tiered model. Beyond extraction, the platform includes role-based access control, human-in-the-loop review, and analytics. The platform is also multi-domain and schema-driven, allowing the same platform to support invoices, contracts, HR records, and other enterprise documents without code modifications.

Methodology

The project was executed in two phases combining research exploration and iterative development. The initial phase explored Intelligent Document Processing inspired by JPMorgan DocLLM for layout-aware financial extraction. Since DocLLM was not deployable, core ideas were replicated using Open-DocLLM (OCR + LLM). An offline pipeline was built using Tesseract OCR, Mistral 7B via Ollama, and FastAPI orchestration. Experiments included pypdfium2 for PDF parsing and LLaVA 7B/13B for vision-language inference. OCR+LLM performed reasonably on English receipts but lost layout in flat-text form, multilingual performance degraded, and vision models hallucinated on financial data. A zone-aware 3×3 spatial extraction using TSV coordinates was introduced, improving structural awareness but remaining sensitive to parsing errors. In the second phase, we explored GOT-OCR 2.0, IBM Docling, and Qwen2-VL-7B to build a more advanced pipeline. However, the single-tier design was inefficient because every document was processed through a heavy VLM, making it costly and slow for simple cases. This highlighted the need to move toward a more efficient, hybrid pipeline instead of a fully model-driven one. It follows a “fast by default, intelligent by exception” design. Tier 1 uses PaddleOCR PP-OCRv5 Mobile (CPU), spatial heuristics, regex, and bounding-box matching to extract fields in under 800ms. A Financial Auditor performs decimal-safe arithmetic, consistency checks, and anomaly detection. If discrepancies are detected, Tier 2 triggers Gemini 2.5 Flash with BAML types for targeted reconciliation. LangGraph orchestrates a deterministic state machine (preprocess, OCR, audit, reconcile, persist) with JSONB traceability logs while HITL corrections become labeled data for calibration and future fine-tuning.

Intelligent Document Processing System

Technologies Used

Used: Python, JavaScript, SQL, Bash, FastAPI, SQLAlchemy, Pydantic, LangGraph, BAML, uvicorn, JWT, passlib/bcrypt, PostgreSQL 16, PaddleOCR, Tesseract, EasyOCR, Docling, pdfplumber, poppler, LayoutParser, Detectron2, spaCy, HuggingFace Transformers, Gemini 2.5 Flash, Qwen2-VL-7B, bitsandbytes, vLLM, pandas, NumPy, scikit-learn, statsmodels, Plotly, ARIMA, Prophet, Random Forest, Isolation Forest, React, react-router-dom, recharts, Context API, plain CSS (BEM-lite), nginx, Docker, Docker Compose, BuildKit, Git, GitHub, pytest, VS Code.

Specifications

: The system runs on macOS, Linux, or Windows (WSL2) with 16 GB RAM and ~15 GB free disk. It is deployed as a three-container Docker Compose stack: a React frontend, a FastAPI backend, and PostgreSQL 16. The backend exposes a JWT-secured REST API with role-based access (admin, reviewer, enterprise_user) and integrates with Google Gemini 2.5 Flash for multimodal reconciliation, with an optional self-hosted Qwen2-VL-7B endpoint for domain-tuned inference. Supported uploads are PDF, PNG, and JPEG up to 10 MB. Performance targets include sub-800 ms Tier 1 extraction and strict decimal arithmetic throughout the financial pipeline.

Keywords

Enterprise Document Automation, Intelligent Document Processing, Data Extraction

Visual Question Answering (with multimodal explanations) for Gastrointestinal Imaging

Group Members

Itbaan Safwan 26197,
Abdullah Ahmedani 25896
Ramail Khan 26924

Supervisors

Supervisor (PI): Dr. Atif Tahir

Knowledge Area: Artificial Intelligence based systems

Project Overview

This project presents a Visual Question Answering (VQA) system for gastrointestinal (GI) endoscopy images, designed to answer clinical questions about abnormalities and anatomical landmarks while providing interpretable visual evidence for its predictions. Standard VQA fine-tuning was found insufficient due to a lack of visual grounding, degraded captioning ability, and a tendency to produce correct answers for incorrect visual reasons. To address these limitations, two solutions were developed. The first introduces multi-task learning combining VQA, segmentation mask prediction via SegCLIP-generated pseudo-masks, and rich explanation generation using Gemma 3. While this improved over the baseline, it introduced noisy pseudo-masks and hallucinated explanations. The second solution replaces pseudo-masks with Grad-CAM activation maps derived from a CNN classifier trained on approximately five million GI images, and decouples answer generation from explanation by separately captioning the identified region of interest. This produces more faithful grounding and reduces hallucination. The system is built on Florence-2, a compact vision-language model suitable for clinical deployment. A remaining limitation is the tendency to localise only a single lesion instance in multi-focal cases, for which DINOv2 patch-level similarity is proposed as a future fix.

Methodology

The methodology begins with Florence-2 as the base vision-language model, initially fine-tuned on a Kvasir VQA X1 dataset using standard VQA training, which revealed failures in visual grounding and captioning quality. To address this, a multi-task learning approach was introduced, jointly training the model on VQA, segmentation mask prediction using SegCLIP-generated pseudo-masks, and rich explanation generation using Gemma 3 API responses as targets. However, this introduced noisy masks and hallucinated explanations, leading to a second solution that replaced pseudo-masks with Grad-CAM activation maps from a dedicated CNN classifier trained on approximately five million GI images, and decoupled explanation from answering by separately captioning the identified region of interest rather than generating explanations end-to-end. Finally, to handle multi-focal lesions where only a single instance was being grounded, the use of DINOv2 patch-level similarity matrices is proposed to surface additional visually similar regions across the image.

Visual Question Answering (with multimodal explanations) for Gastrointestinal Imaging

Specifications

The project runs on Linux with Python 3.12.6 and requires a CUDA-compatible GPU, using fp16 mixed precision during training and float32 during inference. The core model is Microsoft Florence-2-base, fine-tuned via LoRA (PEFT) with a final configuration of $r=128$, $\alpha=256$, $\text{dropout}=0.05$ targeting attention and MLP layers across both the vision tower and language model encoder/decoder. Training uses a batch size of 2 with 3 gradient accumulation steps (effective batch size 6), a learning rate of $5e-5$, warmup ratio of 0.1, over 1 epoch. Inference uses beam search ($\text{num_beams}=3$, $\text{max_new_tokens}=1024$) with a custom top-k stability confidence score ($k=5$). The fine-tuned model weights are hosted on Hugging Face Hub (peeache/FL2_VQA_MIX_128_256), the training dataset is sourced from Kaggle (itbaansawan/fyp-gi-vqa-mtl-data) and Hugging Face Datasets (SimulaMet/Kvasir-VQA-x1), experiment tracking is done via WandB, visual ROI captions are generated using the Google Gemini API (gemma-3-27b-it, $\text{temperature}=0.2$, $\text{max_tokens}=120$), and a pre-trained GastroNet-5M (ResNet-50 + DINOv1) backbone is included. Evaluation covers BLEU, ROUGE-1/2/L, and METEOR for VQA, and IoU for segmentation tasks.

Technologies Used

Python, PyTorch, Hugging Face Transformers, PEFT (LoRA), Microsoft Florence-2, Gemma 3 27B, Google Gemini API, Hugging Face Hub, Hugging Face Datasets, OpenCV (cv2), NumPy, Pandas, Pillow (PIL), Matplotlib, NLTK, BLEU, ROUGE, METEOR (evaluate library), WandB, KaggleHub, Jupyter Notebook, GastroNet-5M (DINOv1/ResNet-50), Kvasir-VQA-x1 Dataset, CUDA, dotenv

Keywords

Multi-Task Learning (MTL), Visual Question Answering (VQA), Gastrointestinal (GI) Endoscopy

Legal Sahara: Legal AI Assistant

Group Members

Rafsha (26639)
Hamna Inam Abro (27113)
Zara Masood (26928)

Supervisors

Supervisor (PI): Dr. Solat Sheikh

Knowledge Area: Artificial Intelligence based systems

Project Overview

Pakistan's legal profession is facing an acute problem of access to justice as millions of people are unable to hire lawyers, legal research is still largely manual and filing of petitions requires expertise beyond the reach of ordinary citizens and junior lawyer. Legal Sahara is Pakistan's first agentic AI-powered legal intelligence platform, purpose-built for Pakistani courts, designed to address these systemic challenges through three specialized AI agents: a RAG Agent for intelligent case discovery and legal Q&A over a Supreme Court judgments database, a Petition Drafter Agent that autonomously generates court-ready petitions across five formats and five petition types, and a Summarizer Agent that produces structured legal briefs from uploaded documents including scanned PDFs. It features a hybrid retrieval architecture that integrates BAAI/bge-large-en-v1.5 dense embeddings, BM25 keyword search, and Reciprocal Rank Fusion, all managed by LangGraph stateful agentic graphs that repeatedly execute plans and re-plan until completion. It also incorporates a multi-layer LLM-as-judge evaluation chain to avoid hallucination errors and a multi-model verification strategy using LLaMA and Gemma. Legal Sahara provides a robust, specialized solution that addresses the specific legal context in Pakistan, with a focus on citation hierarchy (PLD > SCMR > YLR > CPLA), offering a secure and specialized alternative to generalist LLMs for high-stakes legal situations in the country—promoting legal technology research and fostering fair access to justice in Pakistan.

Methodology

Legal Sahara was developed using an agentic AI architecture orchestrated through LangGraph, with three specialized agents each built as a stateful graph with conditional edges, retry logic, and self-correction capabilities. The RAG Agent follows a plan-execute-replan loop where Gemma 3 routes queries, LLaMA 3.3 generates answers, and Gemma 3 independently verifies outputs through a two-gate faithfulness check, preventing hallucinated citations by retrieving case IDs directly from ChromaDB metadata rather than generating them. The Petition Drafter operates as an 11-node LangGraph pipeline that classifies the petition type, checks for missing information, detects PPC severity red flags, retrieves supporting precedents via hybrid search, ranks citations by legal hierarchy, drafts the petition using LLaMA 3.3 70B, and then validates the output using a 5-dimensional LLM-as-judge scoring rubric, triggering an automatic revision if thresholds are not met. The Summarizer Agent processes uploaded documents through an 8-node sequential pipeline covering classification, facts, issues, holding, reasoning, ratio decidendi, synthesis, and evaluation, with OCR auto-detection for scanned PDFs at 300 DPI and exponential backoff retry logic throughout. All agents are exposed via a FastAPI REST backend, containerized with Docker Compose, and deployed on AWS EC2 with persistent ChromaDB and HuggingFace model cache volumes.

Technologies Used

Used Python, FastAPI, React.js, LangGraph, LangChain, LLaMA 3.3 70B (Groq), LLaMA 3.1 8B (Groq), Gemma 3 27B (Google Gemini), ChromaDB, BAAI/bge-large-en-v1.5, rank_bm25, Reciprocal Rank Fusion (RRF), pypdf, pytesseract, pdf2image, python-docx, Pillow, Docker, Docker Compose v3.9, AWS EC2, Nginx, Git, GitHub

Keywords

Agentic AI , Legal Technology , Natural Language Processing(NLP)

Legal Sahara: Legal AI Assistant

Specifications:

LLMs: LLaMA 3.3 70B via Groq API (drafting, extraction, answer generation), LLaMA 3.1 8B via Groq (classification, fallback), Gemma 3 27B via Google Gemini (routing, RAG verification)

Retrieval: ChromaDB PersistentClient (vector store), BAAI/bge-large-en-v1.5 (1024-dim embeddings, cosine distance), rank_bm25 BM25Okapi (keyword retrieval), RRF fusion at k=60

Document Processing: pypdf (text extraction), pytesseract + pdf2image (OCR at 300 DPI), python-docx (DOCX parsing), Pillow (image preprocessing); OCR auto-detected when avg text/page < 100 chars; warning triggered if avg OCR confidence < 70%

Backend: FastAPI with REST endpoints — POST /rag, POST /draft, POST /summarize, GET /health

Frontend: React.js served via Nginx on port 3001

Deployment: Docker Compose v3.9, AWS EC2 Ubuntu instance, private bridge network (legal_net), persistent Docker volumes for ChromaDB embeddings and HuggingFace model cache, unless-stopped restart policy

Court Formats Supported: Sindh HC, Lahore HC, Islamabad HC, Sessions Court Karachi, Sessions Court Lahore

Petition Types: Post-Arrest Bail, Pre-Arrest Bail, Habeas Corpus, Quashment of FIR, Property/Encroachment

Evaluation Results: RAG — Precision@3: 0.79, MRR: 0.88; Drafter — avg score 8.3/10 (Gemini Pro judge); Summarizer — LLM-judge mean 80/100, Human mean 79.3/100 across 10 cases

Group Members

Tooba Iqbal Aswani (26958)
Minahil Siddiq (27005)
Muhammad Jawad Maqsood (27080)

Supervisors

Supervisor (PI): Dr. Faisal Iradat

Knowledge Area: Security

Project Overview

WazuhBot is an agentic AI-powered Security Operations Assistant built on top of Wazuh, an open-source SIEM/XDR platform, designed to reduce the burden on security analysts by enabling natural-language interaction with live security data. The system allows analysts to investigate alerts, identify network anomalies, and receive remediation guidance through a conversational interface instead of manually navigating dashboards. Unlike a simple chatbot, WazuhBot operates agentially — it reasons across multiple steps, queries live Wazuh data via a Model Context Protocol(MCP) bridge, retrieves from a curated security knowledge base using Retrieval-Augmented Generation (RAG), and runs a locally trained ML model to classify network traffic as benign or a specific attack type (DoS, DDoS, brute force, port scan, etc.). All LLM inference runs locally via Ollama, ensuring no sensitive security data leaves the network. The project contributes a complete, deployable agentic security operations tool that integrates LLM reasoning, RAG, ML-based anomaly detection, and live SIEM tooling into a unified analyst interface.

Methodology

The project followed an iterative, component-driven development approach. The system was decomposed into five independently developed and tested services: a Vite.js frontend, a Node.js/Express REST API gateway, a Python Flask chatbot service (LLM + RAG + MCP bridge), a Python Flask ML inference service, and a Wazuh MCP server. The ML model (Random Forest classifier) was developed separately using a standard supervised learning pipeline — data exploration and feature engineering in Jupyter Notebook on the NF-UQ-NIDS-v2 dataset, followed by training, evaluation, and export of model artifacts. The RAG knowledge base was built by curating security documents (attack explanations, Wazuh rules, remediation playbooks, hardening guides) and indexing them into ChromaDB using sentence-transformer embeddings. The agentic chatbot was implemented using Ollama's tool-calling API, where the LLM dynamically decides whether to query Wazuh live data, retrieve from the knowledge base, invoke the ML classifier, or combine all three. Integration was validated end-to-end: user registers with Wazuh credentials, sends a natural-language security question, and receives a context-aware response synthesised from live SIEM data and the knowledge base.

Specifications

Hardware: Standard development machine; LLM inference requires minimum 8 GB RAM (16 GB recommended for 8B models); no GPU required (CPU-only Ollama supported)

Software Requirements: Node.js ≥ 18, Python ≥ 3.11, MySQL 8.0, Ollama (local LLM runtime)

LLM: Any Ollama-compatible tool-calling model (recommended: LLaMA 3.1 8B, Hermes3 8B, Qwen2.5 7B)

ML Model: Random Forest classifier trained on NF-UQ-NIDS-v2 (network intrusion detection dataset, 40+ features, **multi-class:** benign, DoS, DDoS, port scan, brute force, etc.)

APIs: Wazuh Manager REST API, Wazuh Indexer (OpenSearch) REST API, Ollama local API

Ports: Frontend :5173, Express API :5000, Chatbot service :5002, ML inference service :5001

Database: MySQL (users, sessions, chat history, per-user encrypted Wazuh credentials)

Vector Store: ChromaDB (local, embedded) for RAG knowledge base

Protocols: HTTP/REST (all inter-service communication), MCP (Model Context Protocol) for Wazuh tool bridge, JWT for authentication, bcrypt for password hashing

Technologies Used

Python, Flask, Node.js, Express.js, Vite.js, Vanilla JavaScript, MySQL, Ollama, LLaMA 3.1 (8B), ChromaDB, Sentence-Transformers, Wazuh SIEM/XDR, OpenSearch (Wazuh Indexer), Model Context Protocol (MCP), scikit-learn, Random Forest, NumPy, pandas, joblib, bcryptjs, JSON Web Tokens (JWT), Nodemailer, CORS, dotenv, NF-UQ-NIDS-v2 Dataset, Jupyter Notebook

Keywords

Agentic AI, Security Information and Event Management (SIEM), Network Intrusion Detection

AI Talent Matching Platform

Group Members

Haseeb Ahmed (26077)
Abdullah Iqbal (26904)
Hussain Abbas (27089)

Supervisors

Supervisor (PI): Miss Tasbiha Fatima
Industry Advisor: Wajeeha Javed
Industry Company: VentureDive

Knowledge Area: Artificial Intelligence based systems

Project Overview

Reading hundreds of resumes by hand to find the right candidate takes too long and good people often get missed. Jobify is an AI based hiring platform made for VentureDive that fixes this problem by doing the whole process automatically. It reads resumes and job postings, pulls out information like skills, education, and experience using language models, and then ranks candidates based on how well they fit the job. The ranking uses both meaning based matching and rules like skill coverage and experience level. Recruiters can also see exactly why a candidate was ranked in a certain position through SHAP based explanations. On top of that, VenD profile generation turns raw candidate data into clean ready to view profiles for recruiters. The goal of this project is to build a full working hiring system that saves time, reduces human error, and makes the recruitment process more fair and easy to understand. The biggest contribution of this work is bringing together resume parsing, smart candidate matching, flexible ranking, and clear explanations into one platform that replaces slow manual hiring with something that actually works at scale.

Methodology

Adopted an iterative and component-based development strategy that involved creating a three-layered microservices architecture made up of the front end, back-end API, and artificial intelligence services. Developed the solution in several stages, starting with the creation of an ETL pipeline responsible for text extraction, layout analysis, data cleansing, and structured data extraction from the LLMs. Developed an entity embedding and semantic matching capability using vector embeddings and a vector database for similarity searches. Created a ranking engine incorporating various ranking strategies including semantic ranking alone, hybrid ranking, RRF, cross-encoder reranking, LambdaMART, and LLM ranking. Optimized the ranking strategies using surrogate measures such as NDCG, precision, MRR, and latency. Designed scoring parameters as dynamic variables that can be changed during execution rather than hardcoding them to enable experimentation. Added explainability features to the system by tracking per-entity and per-feature scores for transparent recruitment. Enabled reproducibility and scalability by packaging the entire system using Docker Compose. Automated tests were performed on text processing pipelines, ranking models, and middleware services.

AI Talent Matching Platform

Specifications

The system is made up of three services that all run together using Docker Compose. The main backend is built with Node.js, Express, and Prisma ORM and handles all the business logic and API requests. The AI service is built with Python and FastAPI and takes care of resume parsing, semantic matching, and candidate ranking. Structured data like user accounts, jobs, and profiles is stored in PostgreSQL while candidate and job embeddings are stored in Qdrant which powers the semantic search. For extracting information from resumes and job postings the system calls the Groq API using Llama 3.3 70B as the main language model and Google Gemini 2.0 as a backup. The two backend services communicate over HTTP with API key authentication, request ID tracking, and automatic retries built in. Security across the platform is handled through JWT authentication, role based access control for candidates, recruiters, and admins, rate limiting, CORS, and structured logging using Winston. To run the system locally you need Docker, Docker Compose, and environment variables set for the Groq API key, Google API key, database connection, JWT secrets, and the internal AI service key.

Technologies Used

Python, FastAPI, Node.js, Express.js, React, Next.js, TypeScript, PostgreSQL, Prisma, Qdrant (vector database), Docker, Docker Compose, BAML, Groq (Llama 3.3 70B), Google Gemini 2.0, sentence-transformers (all-MiniLM-L6-v2), LambdaMART / LightGBM, MLflow, cross-encoder re-rankers, BM25, Pydantic, Winston, JWT authentication, pytest, REST APIs

Keywords

Automated Recruiting System, Ranking Explainable AI, Variable Candidate

Machine Learning Based Cancer-Risk Prediction Using Structural and Sequential Data

Group Members

Darshna Luhana (27116)
Afaf Irfan (26456)
Ali Hamas Detho (26995)

Supervisors

Supervisor (PI): Dr Taslim Murad

Knowledge Area: Bioinformatics with Machine Learning

Project Overview

Proper classification of BRCA1 and BRCA2 gene mutations plays a crucial role in estimating the risk of hereditary breast and ovarian cancers. However, existing computational approaches to variant pathogenicity prediction are still limited due to their inability to utilize non-sequence-based features and interpret results clinically. The current research aims to develop an efficient machine learning approach based on integrating sequence-based and structural properties for improving classification accuracy and providing clinical interpretation. Variant information was downloaded from ClinVar, and 74 sequence features were obtained via Ensembl VEP. In addition, 5 three-dimensional structural features were calculated for each mutation using crystal structures of PDB protein models, FoldX, UCSF ChimeraX, and Biopython software. Finally, three feature sets were examined within the following machine learning models: Random Forest, XGBoost, Decision Tree, KNN, and Logistic Regression. It was shown that combining both sequence and structure information resulted in higher classification performance than using only one of the feature sets. Overall, Random Forest provided the best results with a classification accuracy of 0.892 and ROC-AUC of 0.931. Analysis via SHAP explainability showed that both types of features—sequence-based features like ClinPred and SIFT, and structural features like SASA and DDG—fell into the top ten features that were the most impactful, thus proving the significance of combining the two data sources. This finding implies that using both kinds of data in developing predictors results in more robust mutation predictors with clear biological interpretation, which could potentially impact clinical counseling and cancer prediction.

Technologies Used

Python, Clinvar, Foldx, chimeraX, Uniport, PDB, SHAP, BioPython

Keywords

BRCA1/BRCA2 Variant Classification , Explainable AI, 3D Protein Structure Analysis

Specification

Python with biopython installed
Min 8gb ram

Machine Learning Based Cancer-Risk Prediction Using Structural and Sequential Data

Methodology: Data Collection

ClinVar was utilized as the primary data source for extracting both sequence and structural variations. Human variants associated with the BRCA1 and BRCA2 genes were isolated. The target label utilized for this classification task is the IMPACT attribute, which comprises four distinct levels: High, Moderate, Low, and Modifier. This target variable was carefully selected and isolated to prevent any potential data leakage during downstream preprocessing and modeling.

Feature Extraction from Sequence Pipeline:

The sequence-based feature extraction pipeline initiated by processing raw data on ClinVar variants through Ensembl's Variant Effect Predictor (VEP). This step yielded an initial genomic dataset consisting of 85 feature columns across 33,745 sample rows.

Structural Feature Extraction:

For the structural pipeline, three-dimensional macromolecular data for BRCA1 and BRCA2 were extracted. The original, raw Protein Data Bank (PDB) files were processed and cleaned utilizing the UCSF ChimeraX software tool, which successfully excluded all water molecules and non-protein ligands. To compute the relevant thermodynamic alterations induced by the identified point mutations, the FoldX empirical force field approach was systematically employed.

Integration of Datasets:

To fuse the distinct biological modalities, a specialized mutation mapping key was derived from the unprocessed data generated by the VEP. Concurrently, a custom Python standardization script was executed to remove chain identifiers from the structural output files, establishing a standardized, uniform mutation key column. An inner merge operation was subsequently executed via the Pandas library to seamlessly integrate the sequence and structural datasets into a unified master dataset.

Machine Learning and Evaluation:

Three distinct data configurations were derived from the master integrated dataset for comparative analysis: a sequence-only subset, a structure-only subset, and the combined multimodal dataset. For each of these three configurations, five distinct machine learning classification algorithms were implemented: Decision Tree, Random Forest, XGBoost, KNN, and Logistic Regression. Each model configuration was rigorously validated and evaluated using a comprehensive suite of performance metrics, including accuracy, Receiver Operating Characteristic Area Under the Curve (ROC-AUC), precision, recall, and F1-score. SHAP was used as an explainable AI tool to evaluate the results.

Industry Project (with Alchemy Technologies)

Group Members

Shahzain Ali Khan (26937)
Fatima Mahmood (24314)

Supervisors

Supervisor (PI): Jawwad Farid
Industry Mentor: Agnes Paul

Knowledge Area: Data Analytics

Project Overview

The purpose of this project is to transform raw deposit balance data into meaningful behavioral intelligence that supports decision-making across multiple banking stakeholders, including treasurers, risk managers, branch managers, relationship managers, product managers, and senior management. Traditional banking analysis often relies on static balance snapshots, which fail to capture how deposits behave over time, especially under changing or stressful conditions. To address this, the project introduces a layered analytical framework called the Deposit Behavior Engagement Model, which processes large-scale daily end-of-day deposit data into reusable account-level and segment-level behavioral metrics such as inflows/outflows, volatility, drawdowns, seasonality, persistence, probability, and trigger-based risk analysis. The project contributes by designing a scalable and optimized database architecture capable of handling over 10 million rows of financial time-series data while generating stakeholder-ready analytical reports, on both the frontend and excel, efficiently within practical time constraints.

Technologies Used

Oracle SQL, Pro*C, TypeScript, JavaScript, Node.JS, React, Excel

Specifications

The project was developed using Oracle Database for large-scale data storage and analytical processing, with Oracle SQL used for querying, optimization, and behavioral metric calculations. The reporting and dashboard layer was built using TypeScript, while Pro*C and libxlsxwriter were used for generating Excel-compatible analytical reports. The system processes over 10 million rows of daily time-series deposit data and relies heavily on precomputed analytical layers and optimized database structures for performance. Hardware requirements include sufficient memory (RAM) and storage depending on the size of the dataset, the number of metrics involved, and the types of reports required.

Methodology

The project follows a layered database architecture and analytical pipeline approach designed to transform large-scale raw deposit data into reusable behavioral analytics and stakeholder-ready reports. The methodology begins with importing raw daily end-of-day deposit balances, which are then normalized into a relational schema using mapping and standardization layers to ensure consistency and reduce redundancy. To improve analytical performance and minimize repeated joins, an optimized denormalized account master layer was introduced. On top of this, reusable scratch/base analytical tables were created to precompute account-level and segment-level behavioral metrics such as daily changes, inflows/outflows, volatility, seasonality, and drawdowns. The system further separates parameter-independent computations from parameter-dependent analytics to avoid recalculating expensive logic repeatedly. Finally, reporting and view layers expose business-ready outputs for different banking stakeholders. This methodology emphasizes scalability, modularity, performance optimization, and reusability, enabling efficient analysis of millions of rows of financial time-series data.

Keywords

Deposit Behavior Analytics, Financial Data Engineering, Liquidity Risk Modeling

AI-Powered eKYC

Group Members

Syed Ali Rizwan 26906
Syed Hamza Asif 26975

Supervisors

Supervisor (PI): Syed Ali Raza
Industry Advisor: Muhammad Hassan Khan
Industry Company: TPS Worldwide

Knowledge Area: Artificial Intelligence based systems

Project Overview

This project presents an AI-powered electronic Know Your Customer (eKYC) platform delivered as a modular SaaS solution that automates and streamlines customer identity verification for banks, fintechs, telcos, and lending platforms. The system integrates document verification via OCR and authenticity analysis, biometric authentication through face matching and liveness detection, external database screening, and behavioral biometrics into a unified, configurable onboarding pipeline. Key contributions include a no-code drag-and-drop flow builder for compliance and product teams, real-time guided document capture SDKs for mobile and web, an intelligent risk and decision engine with explainable outputs, and a manual review console with full audit trails.

Technologies Used

Technologies Used: Python, TensorFlow/PyTorch (deep learning models), ArcFace/FaceNet (face matching), OCR models (multi-language), React Native, iOS SDK, Android SDK, JavaScript/Web SDK, REST APIs, gRPC, RabbitMQ/Kafka (message queues), Docker/Kubernetes (containerization and orchestration), PostgreSQL/Document DB/Graph DB/Time-series DB (polyglot persistence), AES-256, TLS 1.3, CI/CD pipelines, Vector Database (face embeddings).

Specifications

Platforms: Web (Chrome, Edge, Firefox — latest 2 major versions), Android (API 24+), iOS (iOS 13+), React Native
Viewport support: 360px (mobile) to 1920px (desktop)
Image formats: JPEG, PNG, HEIC, WebP, multi-page PDF
Max upload size: 8MB per image (configurable);
chunked/resumable upload support
Face embedding: 512-dimensional float vector (cosine/Euclidean similarity scoring)
Camera preview latency: <150ms on target devices
Quality hint refresh rate: ≥5Hz real-time preview
Performance: End-to-end decision within ~30 seconds; first meaningful paint within 1s on 3G; reviewer actions persist within 2s
Availability: 90% uptime target for core APIs
Security: TLS encryption, AES-256 encrypted storage, RBAC (Admin, Editor, Reviewer, Auditor, SRE), session timeout default 15 minutes
APIs: REST endpoints (start flow, submit node result, get instance status), webhook callbacks, SDK keys with fine-grained scopes
Compliance: WCAG AA accessibility, immutable audit logs, exportable compliance reports

Methodology

The project follows an Agile development methodology with short, focused sprints delivering incremental and testable value. Development is organized into eight sequential phases: (1) Foundation and CI/CD setup, (2) Core microservices and data persistence, (3) Capture flows and data pipeline, (4) Baseline ML model development (MVP), (5) Model tuning and risk/decision engine integration, (6) QA, security hardening, and penetration testing, (7) User Acceptance Testing (UAT) and production deployment with observability, and (8) Final documentation and stakeholder handover. Testing is layered (unit → integration → end-to-end) with automated regression runs in the CI pipeline, and sprint planning, daily stand-ups, and retrospectives are used for project management and risk control.

Keywords

Electronic KYC, Customer Onboarding, Virtual KYC

INTELLIQUERY: AI-powered Conversational Business Intelligence Dashboard Platform

Group Members

Izma Khan (26926)
Sameed Ahmad (26956)
Shaikh Muhammad Umair (26409)

Supervisors

Supervisor (PI): Abeera Tariq
Industry Advisor: Muhammad Naeem
Industry Company: Folio3

Knowledge Area: Artificial Intelligence based systems

Project Overview

IntelliQuery is an AI-powered natural language to SQL agent built on a multi-node LangGraph workflow that enables non-technical business users to query a PostgreSQL (Northwind) database hosted on AWS RDS simply by typing plain English questions. The system contributes a fully automated pipeline: Schema Inspector, SQL Planner, Query Validator, SQL Executor, and Output Formatter, where each node is intelligently connected by conditional routing edges that handle retries, error recovery, and security enforcement; the Planner node leverages Google Gemini LLM combined with ChromaDB semantic vector search over 23 curated few-shot examples to generate contextually accurate SQL, while the Validator enforces security (blocking injections and destructive statements) and business-logic correctness before any query reaches the database, ultimately democratizing data access by letting users get formatted, business-ready results from complex relational data without writing a single line of SQL.

Technologies Used

Python, LangChain, LangGraph, LangChain-Google-GenAI, Google Gemini (LLM), LangChain-Groq, Groq, Ollama, FastAPI, Uvicorn, Pydantic, PostgreSQL, AWS RDS, psycopg2, ChromaDB, FastEmbed, Redis, PyJWT, MCP (Model Context Protocol), LangChain-MCP-Adapters, python-dotenv, asyncssh, DuckDuckGo Search, pypdf, docx2txt, React (frontend), Vite, Node.js

Specifications

APIs: Groq API (LLaMA 3.3 70B), AWS RDS PostgreSQL, Redis
Database: PostgreSQL (Northwind, 13 tables) on AWS RDS, ChromaDB (vector store), FastEmbed (local embeddings)
Backend: FastAPI, Uvicorn, JWT authentication, MCP server
LLM Config: LLaMA 3.3 70B via Groq, temperature=0, max 3 retries
Dev Environment: Python 3.x, VSCode, Windows 11, python-dotenv

Methodology

The project follows an Agentic AI development methodology built around a phased, iterative approach. The architecture is designed as a multi-node LangGraph directed graph, where each node (Schema Inspector, Planner, Validator, Executor, Output Formatter) is developed, tested, and validated independently before being wired into the full workflow, a modular-first strategy that allows unit-level testing at the tool, node, flow, and business-logic layers progressively. The development is structured across 7 planned stages, starting with a working NLP-to-SQL baseline (Stage 1, current) and incrementally adding self-correction, hierarchical memory, multi-tool ecosystems, context optimization, and finally PowerBI visualization integration, ensuring each stage delivers a functional system before complexity is added. The SQL generation uses a Retrieval-Augmented Generation (RAG) pattern, where ChromaDB with FastEmbed performs semantic similarity search over 23 curated few-shot examples to ground the Gemini LLM's output in domain-specific query patterns. A multi-layer validation strategy (security, syntax, business logic, database EXPLAIN) acts as a quality gate, and conditional edge routing enables automatic retry loops back to the Planner on failures, making the system self-correcting without human intervention. The overall design philosophy is domain-aware AI, where the agent is pre-loaded with hand-crafted business context, table relationship mappings, and JOIN patterns specific to the Northwind dataset, reducing hallucination and improving SQL accuracy over a generic LLM approach.

Keywords

Natural Language to SQL, Agentic AI, Conversational BI

FedAdaPriv-CPU: Adaptive Differential Privacy with Frozen Backbone for Resource Constrained Federated Learning in Medical Imaging

Group Members

Anusha Randhawa (27095)
Hamza Zaman (27162)
Zain Sharjeel (26922)

Supervisors

Supervisor (PI): Ms. Tasbiha Fatima

Knowledge Area: Security

Project Overview

Federated learning enables hospitals to collaboratively train AI models without sharing sensitive patient data, making it particularly valuable for privacy-preserving healthcare applications. However, existing differentially private federated learning methods typically apply uniform privacy budgets across clients, overlooking differences in data volume, class distribution, and learning progress, which often leads to poor minority-class performance and high computational costs.

To address these challenges, we propose FedAdaPriv-CPU, a CPU-efficient federated learning framework that combines a frozen DenseNet121 backbone with adaptive per-client privacy budget allocation. Privacy budgets are dynamically assigned using three signals: data volume, convergence state, and round progression, while privacy accounting is enforced through a Rényi Differential Privacy accountant.

Evaluated on 21,165 chest X-ray images across four diagnostic classes under severe non-IID conditions, FedAdaPriv-CPU achieved 85.99% accuracy and a macro-F1 score of 0.810 at $\epsilon = 4.0$. The framework improved macro-F1 by 11.5 percentage points over a fixed-budget baseline and increased Viral Pneumonia F1 from 0.420 to 0.722. These results demonstrate that adaptive privacy allocation can substantially improve both privacy-preserving learning and minority-class performance while remaining practical for deployment on standard CPU-based hospital infrastructure.

Methodology

FedAdaPriv-CPU is a privacy-preserving federated learning framework designed for multi-class chest X-ray classification in resource-constrained healthcare environments. At the start of training, each hospital performs a one-time local feature extraction step using a frozen DenseNet121 backbone, generating 1024-dimensional feature vectors that remain stored on-device. Only a lightweight multilayer perceptron classification head is subsequently trained, significantly reducing computational requirements compared to full-model federated learning. During each communication round, clients evaluate the current global model on their local data and compute class-wise performance deficiency scores. These are combined with dataset volume and training progress signals to determine an adaptive privacy budget for each client through a three-signal allocation mechanism.

Local training is performed using differentially private optimization with per-sample gradient clipping, Gaussian noise injection, focal loss, and FedProx regularization. A client-side Rényi Differential Privacy accountant tracks cumulative privacy expenditure and enforces a global privacy budget of $\epsilon = 4.0$ at $\delta = 10^{-5}$. To further protect institutional information, aggregation weights derived from client data volume are privatized using the Laplace mechanism before transmission. The central server receives only the model update and privatized aggregation weight from each hospital and performs privacy-coupled weighted aggregation to produce the next global model. This architecture enables collaborative medical AI development while preserving patient privacy and remaining practical for deployment on standard CPU-based hospital infrastructure.

FedAdaPriv-CPU: Adaptive Differential Privacy with Frozen Backbone for Resource Constrained Federated Learning in Medical Imaging

Specifications

Dataset: COVID-19 Radiography Database containing 21,165 chest X-ray images across four disease classes: COVID-19 (3,616 images), Lung Opacity (6,012 images), Normal (10,192 images), and Viral Pneumonia (1,345 images); 85% train/15% test stratified split..

Federated Setup Configuration: Realistic non-IID data distributions through Dirichlet- $\alpha=0.5$ partitioning with class dominance patterns. Client A holds 6,158 images with 70% COVID-19 dominance, Client B contains 9,249 images with 70% Lung Opacity dominance, and Client C maintains 2,581 images with 70% Viral Pneumonia dominance, reflecting real-world hospital specialization patterns.

Training Configuration:

Privacy budget: $\epsilon=4.0$, $\delta=10^{-5}$ (RDP accountant)

Rounds: 5 warmstart + 30 main adaptive rounds

Batch size: 64 physical, 256 logical (gradient accumulation)

Gradient clipping norm: $C=1.0$

Hardware: CPU-only deployment on 2.4 GHz Intel Xeon processors; no GPU required. Experimentations done on T10 GPU.

Performance: ~4 hours per federated round; macro-F1=0.810, accuracy=85.99% at $\epsilon=4.0$.

Technologies Used

Python, PyTorch, Opacus, torchvision, NumPy, scikit-learn, pandas, matplotlib, MobileNetV2, DenseNet121, Pillow (PIL), Jupyter Notebook, Kaggle, Rényi Differential Privacy (RDP) Accountant, ImageNet (pretrained weights), COVID-19 Radiography Database, Federated Learning, Differential Privacy (DP-SGD), CUDA (optional GPU acceleration)

Keywords

Federated learning, differential privacy, medical imaging, chest X-ray classification, resource-constrained deployment, adaptive privacy budgets, transfer learning, non-IID data

Recallable

Group Members

Maham Junaid (26909)
Maryam Ihsan (27152)
Talha Shahid (26948)

Supervisors

Supervisor (PI): Syed Ali Raza

Knowledge Area: Artificial Intelligence based systems

Project Overview

Recallable is an intelligent mobile application designed to transform university-level learning by integrating Learning Management System (LMS) scraping, AI-powered content generation, and spaced repetition learning algorithms. The system automates the extraction and organization of course materials from institutional LMS platforms, generates comprehensive lectures and revision materials using generative AI, and provides personalized study scheduling through the SM-2 spaced repetition algorithm. The application offers interactive chat-based Q&A, dynamically generated flashcards and multiple-choice questions, and real-time progress tracking across multiple courses. By combining LMS integration with intelligent content generation and adaptive learning techniques, this project addresses the challenge of knowledge retention and effective study planning in higher education, enabling students to achieve mastery through friction-less, AI-enhanced learning experiences.

Technologies Used

Frontend: Flutter, Dart, Firebase Auth
Backend: FastAPI, Python, SQLAlchemy, JWT Authentication
Database & Cloud: Firebase Firestore, Google Cloud Storage, AWS EC2
AI & Machine Learning: Claude (Anthropic), Groq API (Llama models)
LMS & Content Processing: Selenium, BeautifulSoup4, PyPDF
Infrastructure: Docker, Uvicorn, Google Cloud Scheduler

Specifications

API Response Time: <2 seconds for standard queries
Content Generation: 30-120 seconds per lecture/summary (depends on AI model)
Session Storage: Support for 500+ concurrent users
File Upload: Maximum 50MB per document
Authentication: JWT tokens with 24-hour expiry
Sync Frequency: Incremental LMS sync every 6 hours
Hardware Used (Development):
Development Machine: Intel i7/Ryzen 7, 16GB RAM
Deployment Server: AWS EC2 (t2.large instance with 8GB RAM)
Database: Google Cloud Firestore (managed, scalable infrastructure)

Methodology

The development followed an iterative, full-stack agile approach with the following phases:

1. Requirements Analysis & Architecture Design – Defined system requirements through stakeholder interviews and designed microservices-based architecture separating concerns into presentation, application, and data layers.
2. LMS Integration Development – Built web scraper using BeautifulSoup4 to automate course material extraction from institutional LMS platforms, handling authentication and maintaining incremental sync capabilities.
3. Backend API Development – Implemented RESTful API using FastAPI with JWT-based authentication, role-based access control, and comprehensive error handling.
4. Content Processing Pipeline – Developed automated pipeline for PDF extraction, text processing, and AI-powered content generation (lectures, summaries, flashcards, MCQs).
5. Frontend Development – Built responsive mobile application using Flutter with offline-first design, real-time state management, and seamless API integration.
6. Learning Algorithm Integration – Implemented SM-2 spaced repetition algorithm with topic-level progress tracking and adaptive scheduling for optimal knowledge retention.
7. Testing & Iteration – Conducted unit testing, integration testing, and user acceptance testing with iterative refinements based on feedback.
8. Deployment – Deployed backend on AWS EC2 with Docker containerization, configured Firebase for real-time data synchronization, and tested cross-platform frontend.

Keywords

Microlearning, Spaced Repetition, Active Recall

Financial News Sentiment Analyzer for Market Intelligence

Group Members

Syed Baryal Shah (27128)
Aaraiz Masood (26988)
Syed Ammar Ali Zaidi (26964)

Supervisors

Supervisor (PI): Ms. Samreen Kazi

Knowledge Area: Artificial Intelligence based systems

Project Overview

The Financial News & Report Sentiment Analyzer is an intelligent system designed to help investors and financial analysts make informed decisions by automating the analysis of financial news. Its primary objective is to collect real-time news from multiple sources, process it using the FinBERT natural language processing model to determine sentiment (positive, negative, or neutral), and correlate these sentiment scores with actual stock market movements. The project contributes an accessible, educational platform that demonstrates the relationship between news sentiment and market behavior while transparently communicating the limitations of sentiment-based market predictions.

Technologies Used

Python, JavaScript, TypeScript, SQL, FastAPI, React, PostgreSQL, Redis, FinBERT, spaCy, Hugging Face Transformers, PyTorch, pandas, NumPy, SciPy, Chart.js, Recharts, Google News API, Yahoo Finance API, Alpha Vantage API.

Specifications

The system integrates with the Google News, Yahoo Finance, and Alpha Vantage APIs to aggregate real-time financial news and market data. On the server side, it requires a minimum of a 4-core CPU (8 cores recommended), 8 GB of RAM, and 100 GB of SSD storage, running on either Linux (Ubuntu 20.04 LTS+) or Windows Server 2019+. The software stack relies on Python 3.8+, Node.js 16.x+, PostgreSQL 12+, and Redis 6.0+. Client-side requirements specify any modern processor, at least 4 GB of RAM, a minimum display resolution of 1280x720, and a modern JavaScript-enabled web browser. Performance-wise, the system is designed to process news articles within 5 seconds of collection, sustain a throughput of up to 1000 articles per hour, achieve at least 80% sentiment classification accuracy, and maintain a 95% uptime during business hours.

Methodology

The project employs a phased development methodology that prioritizes establishing a functional core system before integrating advanced capabilities. Phase 1 focuses on building this core pipeline, encompassing API-driven data collection, FinBERT-based sentiment analysis, and basic dashboard visualization. Phase 2 emphasizes system enhancements, refined statistical analysis, and comprehensive performance evaluation, while Phase 3 targets advanced features like source credibility verification, contingent on time constraints. Architecturally, the system utilizes a Layered Architecture combined with Microservices principles, leveraging an event-driven flow to manage asynchronous background tasks, such as news collection, alongside synchronous processing for user-initiated requests.

Keywords

Sentiment Analysis, Market Intelligence, Natural Language Processing

Researching Face-Voice Association in Multilingual Contexts

Group Members

Noor Un Nisa Shaukat (26971)
Hamna Usman (26990)
Anushe Ali (26418)

Supervisors

Supervisor (PI): Dr. M. Atif Tahir

Knowledge Area: Machine learning

Project Overview

This project presents a multilingual multimodal biometric verification system that improves identity authentication by combining facial recognition and voice analysis within a unified framework. The system addresses the limitations of existing face-voice association approaches, which are typically trained in monolingual settings and often fail to maintain reliable performance when users speak different languages or switch between them. By incorporating language-aware multi-task learning, the framework extracts and fuses facial and vocal embeddings while considering language variations to achieve robust identity matching across multilingual and noisy environments. The project evaluates multiple fusion strategies, benchmarks performance on multilingual datasets, and examines fairness across demographic factors such as gender. In addition to its research contribution toward cross-lingual face-voice verification, the project also includes a proof-of-concept prototype interface demonstrating practical applications in biometric authentication, secure access control, and forensic investigations.

Technologies Used

Python, PyTorch, TensorFlow, OpenCV, CNN, Vision Transformer (ViT), wav2vec 2.0, x-vectors, multimodal learning, transformers, face recognition, speaker verification, multimodal fusion, contrastive learning, Kaggle,

Specifications

The project is developed using Python with frameworks such as PyTorch and TensorFlow. OpenCV is used for face processing, while wav2vec 2.0 and x-vector models are used for voice feature extraction. Training and experimentation are performed on Kaggle using GPU-enabled notebooks and high-memory environments to efficiently process large multilingual audiovisual datasets. The system also includes a simple web-based interface for face-voice verification and uses evaluation metrics such as Accuracy, ROC-AUC, and Equal Error Rate (EER).

Methodology

The project follows a research-oriented multimodal deep learning methodology focused on multilingual face-voice verification. First, a multilingual audiovisual dataset is prepared and divided into train and test sets using an unseen-speaker and unseen-language protocol to ensure fair evaluation. Facial and vocal features are then extracted using deep learning-based embedding models, where face encoders generate facial identity representations and speech models generate speaker embeddings while preserving language-related characteristics. These embeddings are fused using different multimodal fusion strategies, including early fusion, late fusion, and shared embedding approaches, combined with language-aware multi-task learning to jointly perform identity verification and language classification. The system is trained and optimized in GPU-based environments and evaluated using metrics such as accuracy, ROC-AUC, and Equal Error Rate (EER) under multilingual, noisy, and cross-accent conditions. Finally, a prototype interface is developed to demonstrate real-time face-voice matching and confidence-based verification results.

Keywords

Multimodal Biometrics, Face-Voice Verification, Deep Learning

IBA Exam Scheduler

Group Members

Safwan Adnan (27136)
Sana Arshad (27093)
Umaima Raheel (27159)

Supervisors

Supervisor (PI): Dr. Ahmed Akhter

Knowledge Area: Software Development

Project Overview

The IBA Exam Scheduler is a web app built to take the pain out of scheduling university exams. It generates conflict-free timetables by working through the usual headaches: room capacities, student enrollments, and available time slots. The interface is clean and straightforward, and the scheduling logic draws from the open-source UniTime solver. The goal was to cut down on manual admin work, reduce the mistakes that come with doing this by hand, and make the whole process a bit fairer for students and faculty alike.

Technologies Used

Next.js, React, TypeScript, Prisma ORM, PostgreSQL, SQLite, Tailwind CSS, NextAuth.js, Radix UI

Specifications

System Requirements: Node.js runtime environment (v20+), Web browser client.
Database: PostgreSQL (production) or SQLite (development) managed via Prisma ORM.
Architecture & APIs: Next.js App Router architecture with internal REST API routes. Supports integration with a dedicated backend Solver API service for heavy computational scheduling tasks.
Authentication: Secured using NextAuth.js with encrypted sessions (bcryptjs).
Export Capabilities: Includes PDF generation (jsPDF) and Excel parsing (xlsx) for reporting and data ingestion.

Methodology

The project uses Next.js for both frontend and backend, which keeps the stack simple, one framework handles the UI, server-side rendering, and API routes together. Data is managed through Prisma ORM, which makes migrations and database queries straightforward. The scheduling logic is the more interesting part: it's a constraint satisfaction model, loosely based on the UniTime engine, that works through room conflicts, enrollment clashes, and time-slot restrictions until it lands on a workable timetable.

Keywords

Automated Scheduling, Constraint Satisfaction, Exam Timetabling

Cardiac Digital Shadow

Group Members

Umar Hussain (27132)
Rija Faisal (26943)
Syed Saqlain (26869)

Supervisors

Supervisor (PI): Dr. Imran Khan

Knowledge Area: Artificial Intelligence based systems

Project Overview

Cardiac Digital Shadow is an AI powered ECG analysis platform designed to make clinical grade arrhythmia detection accessible through a simple, interactive web interface. Traditional ECG interpretation requires trained cardiologists and specialist software, creating a bottleneck in research and educational settings where rapid, automated analysis is needed. This platform addresses that gap by allowing users to upload a twelve-lead ECG recording and receive an instant AI driven cardiac analysis across 71 distinct diagnostic and rhythm conditions through an intuitive dashboard without any specialist tooling. The system processes recordings through a deep learning model trained on the PTB XL database, identifies detected conditions with severity classifications, and explains its predictions by highlighting the exact portions of the ECG waveform that influenced each diagnosis. Beyond classification, the platform computes clinical measurements including heart rate, PR interval, QRS duration, and QTc, adjusts interpretations based on patient demographics, and generates structured clinical notes. Users can also simulate cardiac pathologies and treatments, interact with an AI assistant to ask questions about the analysis in plain language, and export a formatted clinical report. The primary contribution of this project is bringing together deep learning inference, signal level explainability, demographic aware clinical reasoning, and an accessible web interface into a single cohesive platform aimed at supporting medical education, ECG research, and pre-clinical decision support.

Technologies Used

Uvicorn, PyTorch, Keras, scikit-learn, NumPy, SciPy, pandas, NeuroKit2, WFDB, HeartPy, PyWavelets, joblib, ReportLab, Groq API, LangGraph, ChromaDB, sentence-transformers, RAG, Pydantic, Docker, React, TypeScript, Vite, Tailwind CSS, uPlot

Specifications

The backend uses Python 3.11+, FastAPI, and Uvicorn, with a PyTorch xresnet1d101 model trained on PTB-XL for 12-lead ECG classification. It supports WFDB and CSV uploads. The RAG assistant uses ChromaDB, sentence-transformers, and the Groq API. The frontend is a React 19 + TypeScript SPA built with Vite, using uPlot for ECG visualization and Tailwind CSS for styling. PDF reports are generated with ReportLab, and the system is containerized with Docker and Docker Compose. Recommended hardware is an 8 GB RAM multi-core CPU, with optional CUDA GPU acceleration.

Keywords

ECG Arrhythmia Classification, Explainable AI, Clinical Decision Support

Methodology

The project was developed using a modular, pipeline driven approach where each component was built and validated independently before integration. Development began with the ECG data pipeline, establishing support for both WFDB and CSV formats with automatic format detection, lead standardization, amplitude normalization, and resampling. The deep learning inference pipeline centers on an xresnet1d101 model trained on PTB XL for 71 class multi label cardiac classification. Baseline models including SVM, Logistic Regression, and a Transformer encoder were benchmarked against the primary model to evaluate performance gains. A NeuroKit2 based delineation layer extracts clinical measurements including PR interval, QRS duration, QTc, and HRV metrics, with a signal quality gate applied before inference to reject corrupt recordings. Grad CAM was implemented on the xresnet1d101 model to produce temporal saliency maps over the raw ECG, making predictions interpretable and clinically grounded. A demographic aware threshold engine personalises interpretations based on patient age, sex, and BMI. The AI assistant was built as a Retrieval Augmented Generation pipeline using ChromaDB and sentence transformers, with the 71 condition ECG knowledge base as the retrieval corpus, enabling contextually grounded responses to clinical queries. The frontend was developed in React 19 with TypeScript using uPlot for waveform rendering, and the entire system was containerised using Docker and Docker Compose.



Main Campus

University Enclave, University Road,
Karachi - 75270 Pakistan
Phone : +92-(21) 3810-4700

 www.iba.edu.pk

 +92 21 111 422 422

City Campus

Plot # 68 & 88 Garden / Kayani
Shaheed Road, Karachi - 74400 Pakistan
Phone : +92-(21) 3810-4701

      [ibakhiofficial](#)