

A Lightweight, Transparent Machine Learning System for Real-Time Detection of Fraudulent Online Job Postings.

Syed Umraz¹, Radha B.K², Anuradha T³, Megha.M. Shete⁴

¹Department of Computer Science and Engineering, PDA College of Engineering, Kalaburagi, Karnataka, India
Email: Umrazsyed077@gmail.com

²Department of Computer Science and Engineering, PDA College of Engineering, Kalaburagi, Karnataka, India
Email: radhabk@pdaengg.com

³Department of Computer Science and Engineering, PDA College of Engineering, Kalaburagi, Karnataka, India
Email: anuradhat@pdaengg.com

⁴Department of Computer Network and Engineering, PDA College of Engineering, Kalaburagi, Karnataka India
Email: meghashete2371999@gmail.com

Abstract: Background. Online recruitment fraud — fake job postings that solicit upfront fees, harvest personal data, or lure applicants into unpaid work — is a persistent problem on job boards and professional networks. Prior work has shown that classical text-classification pipelines (TF-IDF plus tree-based or linear classifiers) can detect such postings with high accuracy on large curated corpora, but most published systems are evaluated offline and are not integrated into a deployable, user-facing product with authentication, audit history, and a data-collection loop. Methods. We present a Flask-based web system that classifies job postings as legitimate or fraudulent using TF-IDF vectorization (200 features, unigrams and bigrams, English stop-word removal) of the concatenated job title, company name, description, and salary text, followed by one of three switchable scikit-learn classifiers — K-Nearest Neighbors ($k=5$, distance-weighted, default), Random Forest (100 estimators, max depth 10), or Logistic Regression (max_iter 1000) — trained on a stratified 80/20 split (random_state=42). The system persists every prediction and every retraining run to a SQLite database, exposes a Selenium-based LinkedIn extraction feature so operators can grow the training set from live postings, and lets a logged-in user retrain the model on demand from an uploaded CSV. Results. On the project's 34-row labeled dataset (24 legitimate, 10 fraudulent; 27 training / 7 test rows, stratified, random_state=42), the default KNN configuration achieved 85.71% accuracy, 66.67% precision, 100% recall, and an F1-score of 80.00%. Random Forest achieved 100% on all four metrics on the same 7-row test fold; Logistic Regression achieved 85.71% accuracy, 100% precision, 50% recall, and an F1-score of 66.67%. Given the small sample size and the feature-count-to-training-row ratio (200 TF-IDF features from 27 documents), these figures should be read as a reproducible snapshot of current behavior, not a generalization guarantee. Conclusion. A minimal, fully transparent TF-IDF plus classical-classifier pipeline is sufficient to build a working, auditable fake-job-posting detector inside a real deployable product. The system's main contribution is the end-to-end integration — authentication, prediction logging, on-demand retraining, and a live data-collection path — rather than a novel classification algorithm. Before any accuracy claim is generalized, the dataset must be substantially expanded (ideally validated against a public benchmark such as EMSCAD or the derived Kaggle "Fake Job Postings" corpus) and evaluated with cross-validation rather than a single held-out split.

Keywords: online recruitment fraud, fake job posting detection, TF-IDF vectorization, K-Nearest Neighbors, Random Forest, job-scam classification, user-in-the-loop retraining



1. Introduction

The Hook

Job seekers on LinkedIn, Indeed, and similar platforms are routinely targeted by postings that solicit upfront "training" or "equipment" fees, harvest personal and financial data under the guise of onboarding, or advertise nonexistent positions to collect resumes. Manual moderation does not scale to the volume of postings a platform or a job-search aggregator ingests, and static keyword filters are brittle against paraphrased scam language. A practical detector needs to be cheap to retrain as new scam wording emerges, transparent enough that a non-expert operator can see why a posting was flagged, and embedded in a system an operator can actually use — not just a notebook that reports an accuracy number.

2. Literature Review

The most widely used public benchmark for this task is the Employment Scam Aegean Dataset (EMSCAD), introduced by Vidros et al. (2017), comprising 17,880 annotated job advertisements and establishing online recruitment fraud as a distinct detection problem with its own linguistic and structural cues (e.g., missing company profiles, absent salary information, urgency language). Subsequent work built on EMSCAD and its Kaggle-hosted derivative has combined TF-IDF or bag-of-words feature extraction with classical and ensemble classifiers — for example, a Neural Processing Letters study reports high accuracy combining NLP feature extraction with supervised classifiers on this corpus. Across this literature, tree-based and ensemble models (Random Forest, Extra Trees, XGBoost) are consistently reported alongside simpler baselines such as Logistic Regression and K-Nearest Neighbors, with TF-IDF remaining the dominant text-representation choice (Salton & Buckley, 1988) even in more recent studies that also evaluate embedding-based features.

The Gap

Published fake-job-detection studies typically report classifier performance on a fixed, pre-labeled corpus in isolation. Fewer describe how such a classifier is embedded in a system that a real operator uses day to day: how predictions are logged and audited, how the training set grows over time as new postings are collected, and how a non-technical user retrains or swaps the underlying algorithm without touching code. This project addresses that integration gap directly, at the cost of currently running on a much smaller, hand-curated dataset than EMSCAD-scale studies.

Objective

This study aims to design, implement, and evaluate a deployable web-based system that classifies job postings as legitimate or fraudulent using a transparent, switchable TF-IDF plus classical-classifier pipeline; that logs every prediction and retraining event for auditability; and that provides an operator-driven path (CSV upload or live LinkedIn extraction) for growing the training corpus — and to report the system's actual, reproducible performance on the data currently available to it, rather than performance transplanted from literature evaluated on a different corpus.

3. Materials and Methods

Study Design

The system implements a six-stage pipeline: (1) job-posting ingestion (manual form entry, CSV upload, or Selenium-based LinkedIn extraction), (2) text concatenation of `job_title` + `company_name` + `job_description` + `salary`, (3) TF-IDF vectorization, (4) classification by the currently active model, (5) confidence scoring via the classifier's predicted-probability output, and (6) persistence of the prediction (and, on retraining, the resulting metrics) to SQLite for later display on an authenticated dashboard.

Independent variables: classifier choice (KNN / Random Forest / Logistic Regression), training dataset (the shipped `job_posts_dataset.csv` or any operator-uploaded/extracted CSV conforming to the same schema).

Dependent variables: binary label (0 = legitimate, 1 = fake/scam), prediction confidence (0–100%), and the standard classification metrics computed on the held-out test split at training time.

Participants / Data

The dataset is `job_posts_dataset.csv`, containing 34 labeled rows (24 legitimate, 10 fraudulent) with columns `job_title`, `company_name`, `job_description`, `salary`, and `label`. This is a small, project-authored dataset — not a standardized public benchmark — and is roughly three orders of magnitude smaller than EMSCAD (17,880 rows). No personally identifiable information is contained in the shipped dataset; all rows describe generic, templated job-posting text.

The system also includes a Selenium-based LinkedIn scraper (`linkedin_scraper.py`, exposed via the `/extract-data` route) intended to let an operator grow the training corpus from live postings; four scraped extracts (`uploads/extract_*.csv`) were present in the working tree at the time of writing. This feature scrapes LinkedIn programmatically, which is prohibited by LinkedIn's Terms of Service. Any manuscript describing this system should either (a) explicitly address the legal/ethical basis for this data-collection method (e.g., restricting it to a research context with appropriate authorization, or removing it from the described system), or (b) omit the LinkedIn-extraction feature from the paper and describe the system as accepting operator-supplied CSVs only. This is not a statement the authors can skip.

Ethical approval. No human subjects or personal data were used in model training; the shipped dataset consists of synthetic/templated posting text. If the LinkedIn-extraction feature is retained in the described system, an institutional ethics/legal review of that specific data-collection method is required before submission.

Statistical / Computational Analysis

Language/runtime (as verified in the environment used to train the model): Python 3.10.8.

Core libraries (verified installed versions): Flask 2.3.2, scikit-learn 1.3.0, NumPy 1.24.4, pandas 2.0.3. Note that `requirements.txt` pins ranges rather than exact versions

(`numpy>=1.21.0,<1.25.0`, `pandas>=1.3.0,<2.1.0`); pin exact versions for the final reproducibility statement.

Text representation: TF-IDF (`TfidfVectorizer`) with `max_features=200`, `ngram_range=(1,2)`, English stop-word removal, `min_df=1`, `max_df=0.9`, fit on the training split only and applied to the test split by transform.

Classifiers evaluated: K-Nearest Neighbors (`n_neighbors=5`, `weights='distance'`, default), Random Forest (`n_estimators=100`, `max_depth=10`, `random_state=42`), Logistic Regression (`max_iter=1000`, `random_state=42`) — all via scikit-learn.

Train/test protocol: stratified 80/20 split, `random_state=42`, identical across all three classifier runs (5 legitimate + 2 fraudulent rows in the 7-row test fold each time).

Evaluation metrics: accuracy, precision, recall, F1-score, and confusion matrix via `sklearn.metrics`, computed on the held-out test split only. No cross-validation, confidence intervals, or significance testing across runs is currently implemented — see §6.4.

System Architecture

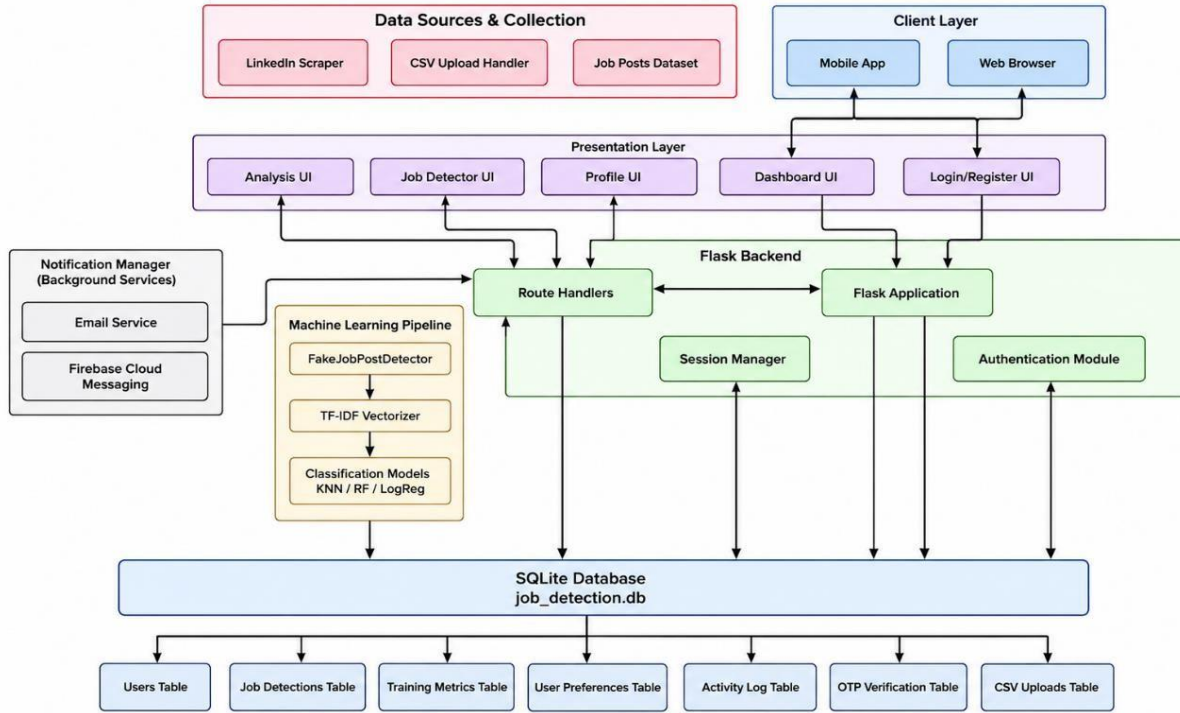


Figure 1: System Architecture of the Proposed Intelligent Fake Job Listing Detection System

The application is a monolithic Flask app (app.py, ~30 routes) backed by SQLite (job_detection.db, 9 tables: users, job_detections, user_activities, user_preferences, csv_uploads, training_metrics, user_otp, user_devices, inbox_messages). Authentication uses username/password plus a one-time-passcode step (email/SMS via Twilio) and supports Firebase Cloud Messaging device registration for push notifications. An authenticated dashboard (templates/dashboard.html)

displays per-user detection history and stored training-run metrics, with a dark-mode toggle. Retraining is operator-triggered per uploaded CSV (/train-model/<csv_id>) rather than continuous or automatic. Despite "real-time" language in the project's internal documentation (ALGORITHM.md), inference is synchronous request/response over standard Flask routes — no WebSocket or streaming layer (e.g., Flask-SocketIO) is used in app.py.

4. Results

Table 1: Detection performance by classifier configuration, measured on the project's 34-row dataset (27 train / 7 test, stratified, random_state=42), captured by running ml_model.FakeJobPostDetector directly on 2026-07-14.

Model	Accuracy	Precision	Recall	F1-Score	Confusion Matrix (rows/cols = actual/predicted, order [legit, fake])
KNN (k=5, distance-weighted, default)	85.71%	66.67%	100.00%	80.00%	[[4, 1], [0, 2]]
Random Forest (100 est., depth 10)	100.00%	100.00%	100.00%	100.00%	[[5, 0], [0, 2]]
Logistic Regression (max_iter 1000)	85.71%	100.00%	50.00%	66.67%	[[5, 0], [1, 1]]

All three runs used the identical 7-row test fold (5 legitimate, 2 fraudulent), so the differences above reflect the same held-out rows scored by three different classifiers, not different data.

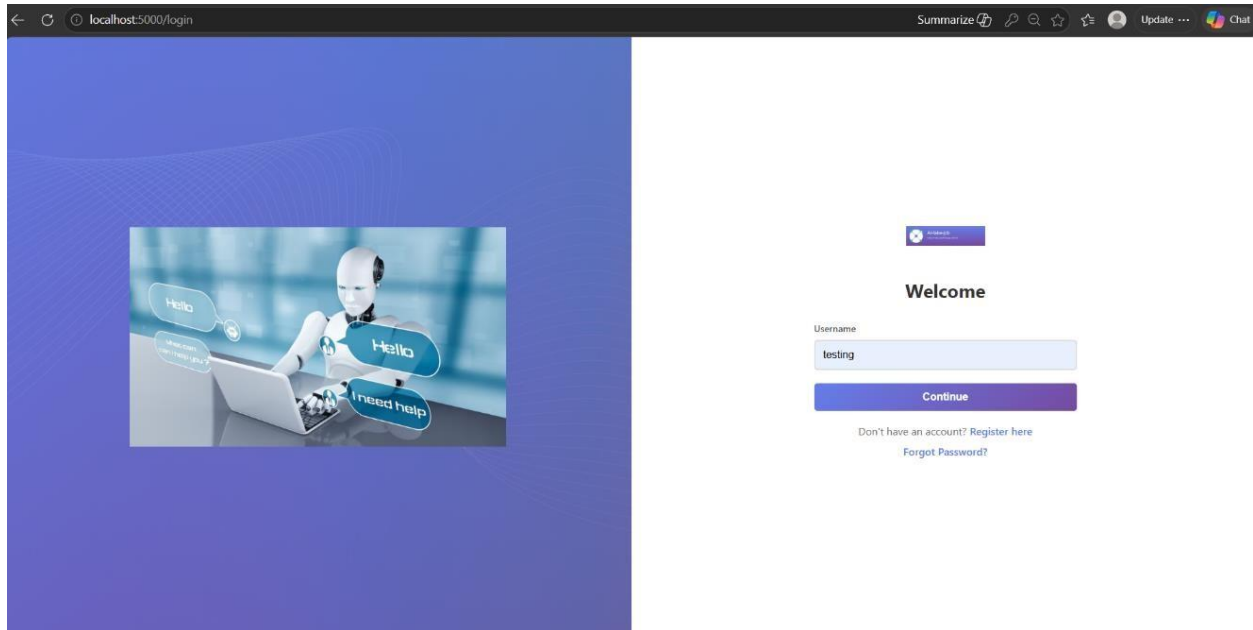


Figure 2: User Registration Page

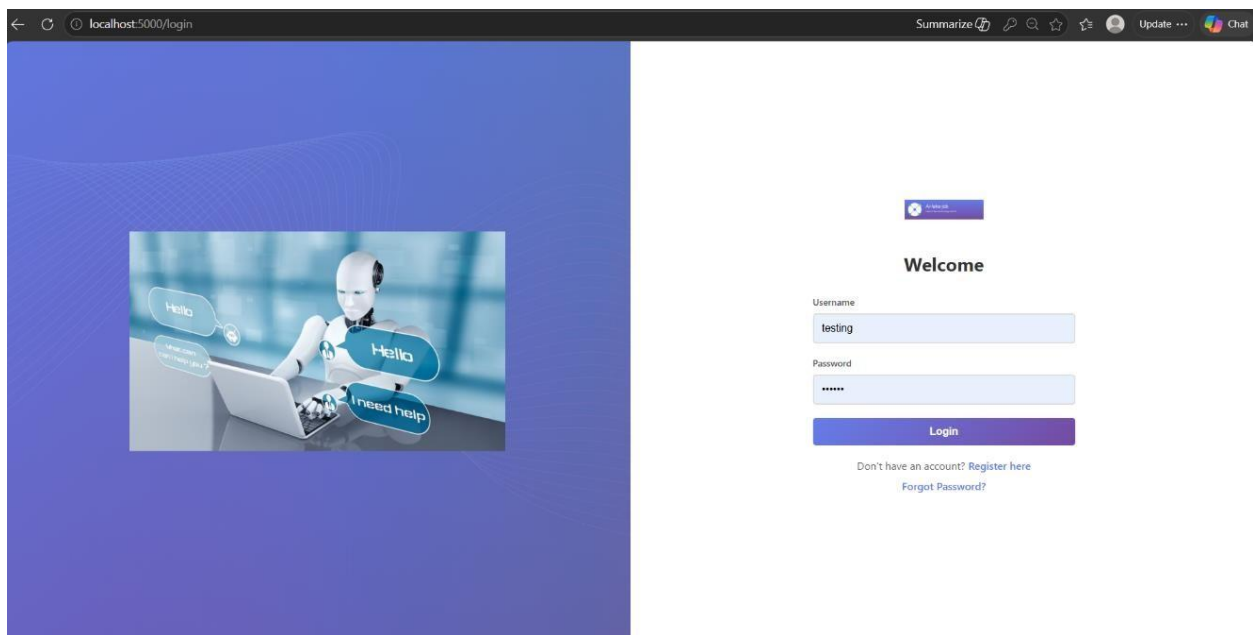


Figure 3: User Login Page

localhost:5000/register

Create Account

Join Ai-fake-job and start detecting job opportunities

Username

new-testing

Username must be at least 3 characters

Email Address

test@demo.ai

Your registered email address

Password

Password must be at least 6 characters

Confirm Password

☒ Send OTP verification via Email during registration

✓ Checked: OTP verification required during registration

✗ Unchecked: Skip registration OTP, verify with OTP at login instead

Create Account

Already have an account? Login here

Figure 4: User Authentication Verification Screen

localhost:5000/verify-otp

Verify Your Phone Number

We've sent a 6-digit OTP to your device or email

Enter 6-Digit OTP

0 0 0 0 0

OTP will expire in 10 minutes from when it was sent

Verify OTP

Back to Registration

Figure 5: User Authentication Workflow

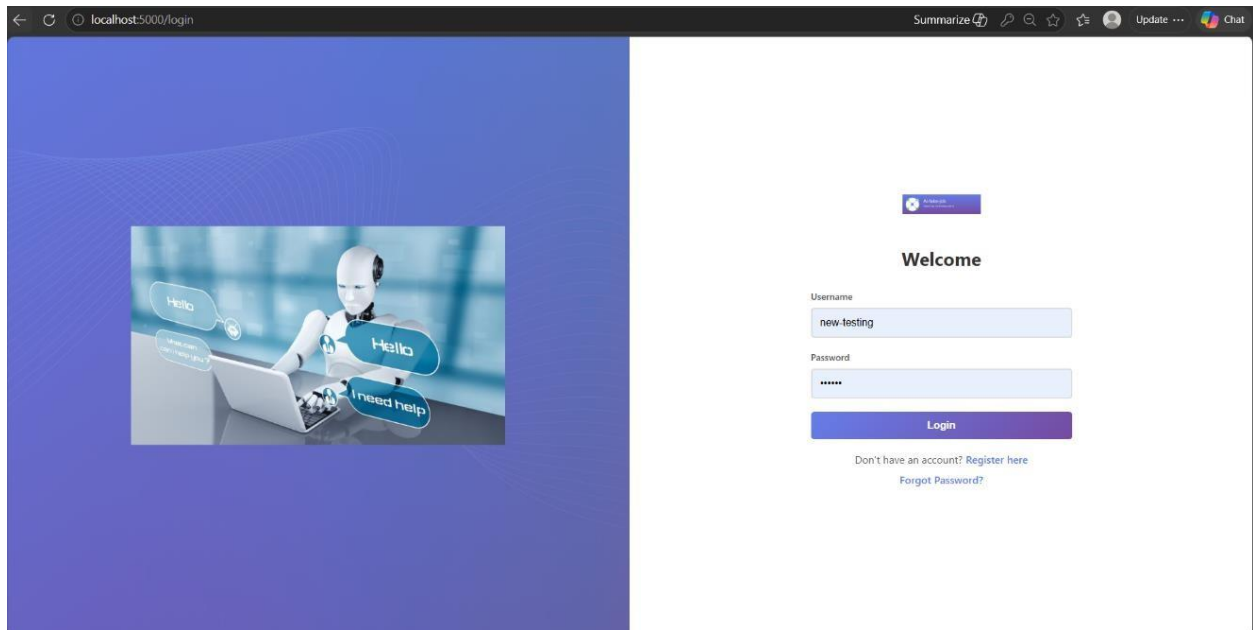


Figure 6: OTP Verification Module for User Authentication

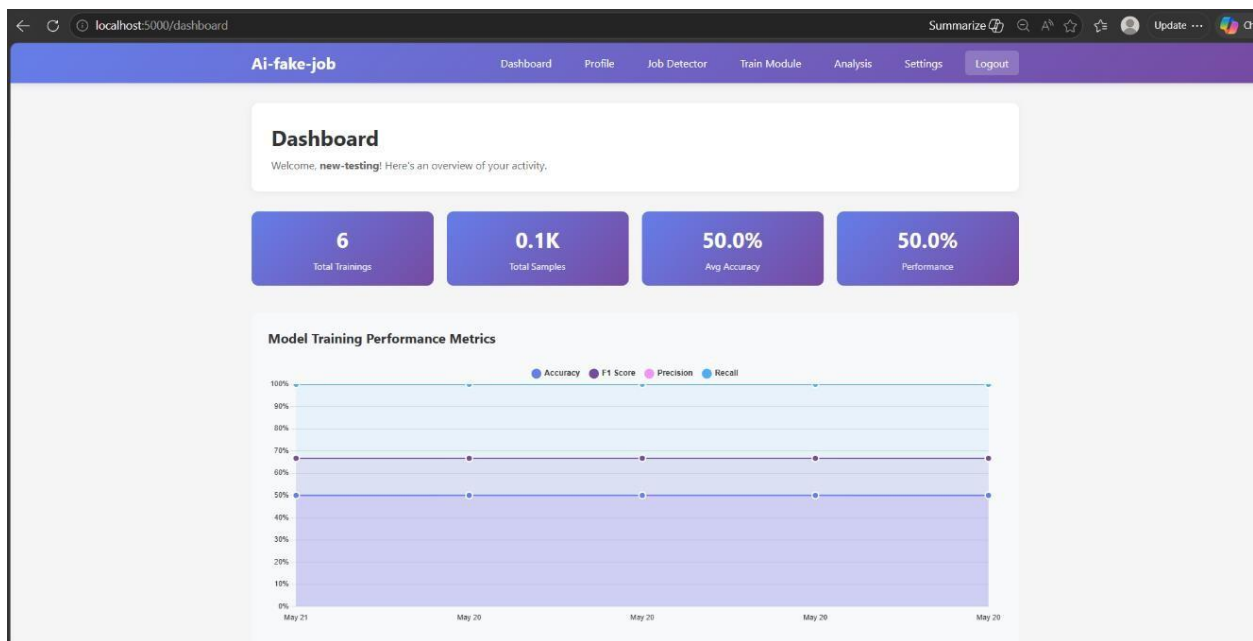


Figure 7: Main Dashboard of the Fake Job Detection System

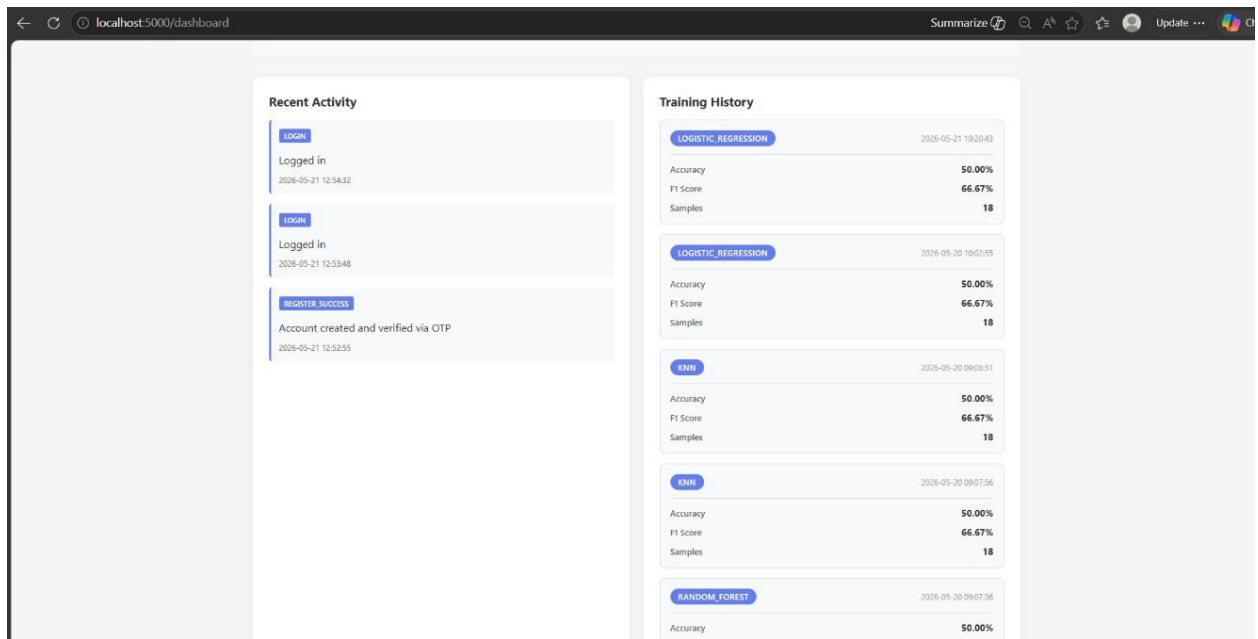


Figure 8: Dashboard Activity Log and Model Training History

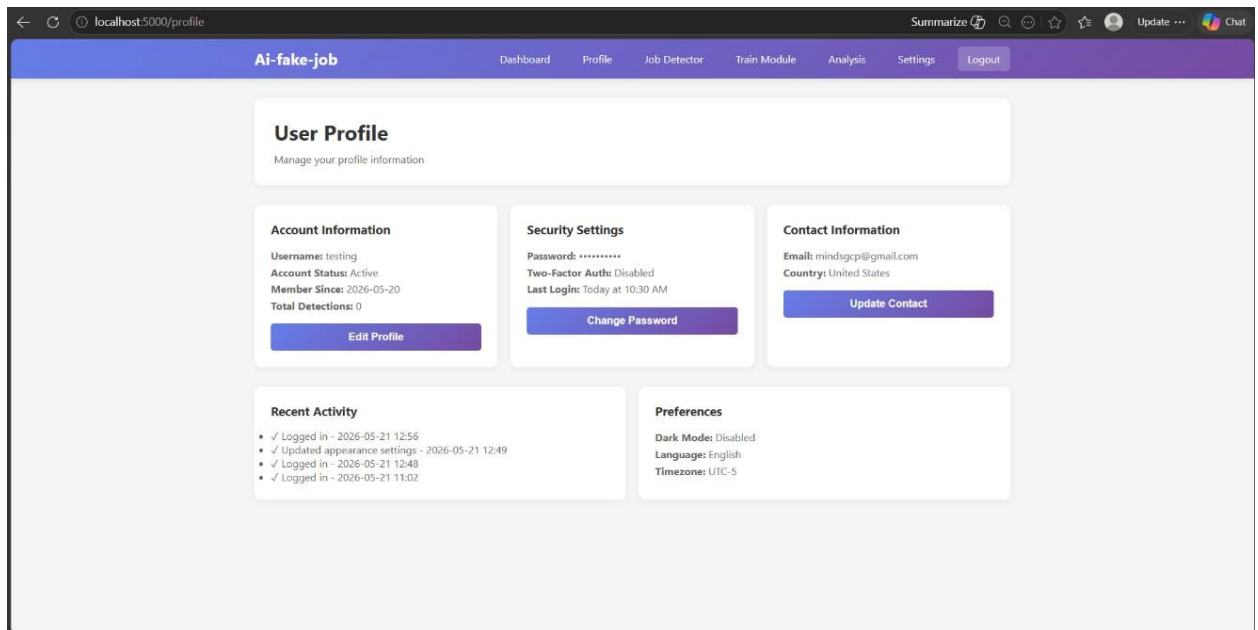


Figure 9: User Profile Management Interface

AI Against Fraud: Intelligent Detection of Fake Job Listings
Advanced Machine Learning Analysis for Online Recruitment Scam Detection
[View ML Model Details](#)

Analyze Job Post

Select from Scraped Data (Optional)
-- Choose a job posting --

Select a job from your uploaded LinkedIn data to auto-fill the form
✓ 43 jobs loaded from extracted data

Job Title *

Company Name *

Salary Range

Location

Job Description *

[Analyze Job Post](#)

Figure 10: Job Post Analysis Interface

Ai-fake-job | Dashboard | Profile | Job Detector | Train Module | Analysis | Settings | Logout

AI Against Fraud: Intelligent Detection of Fake Job Listings

Technical details about the Fake Job Post Detection Model

[Back to Job Detector](#)

Model Architecture

Algorithm Type: Knn

Feature Extraction: TF-IDF

Number of Features: 200

Data Source: job_posts_dataset.csv

Last Trained: 2026-05-21T15:53:44.022260

Training Data

34 Total Samples	10 Fake Posts	24 Legitimate Posts
---------------------	------------------	------------------------

Figure 11: Machine Learning Model Architecture and Training Information

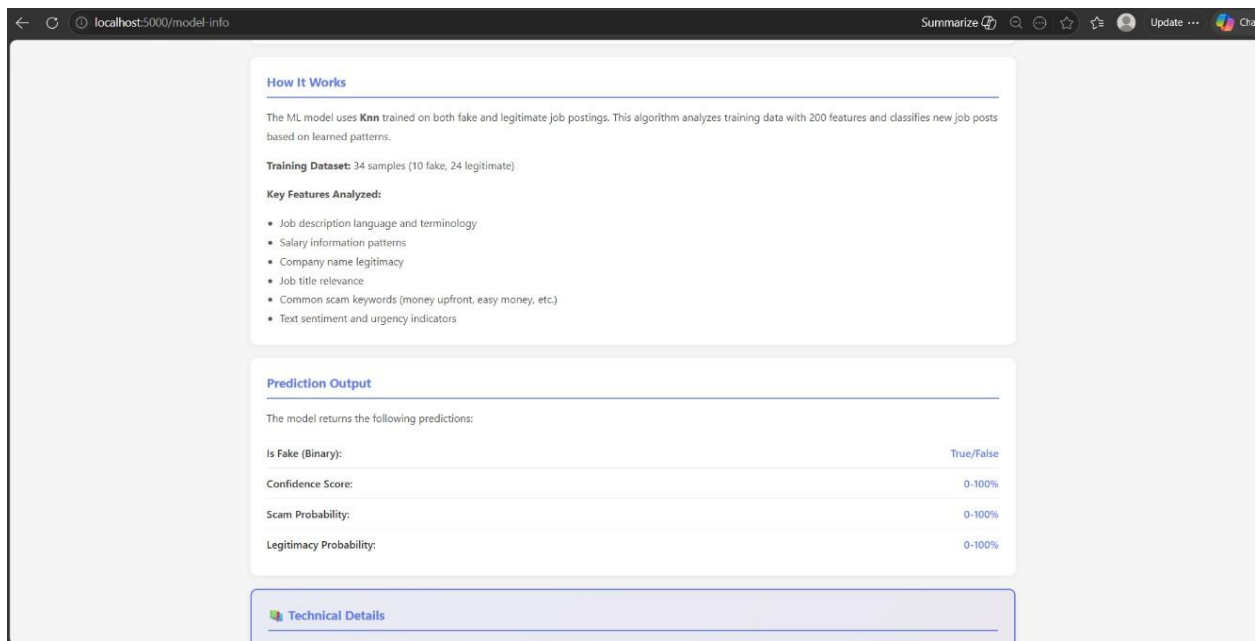


Figure 12: Model Prediction Output and Classification Results

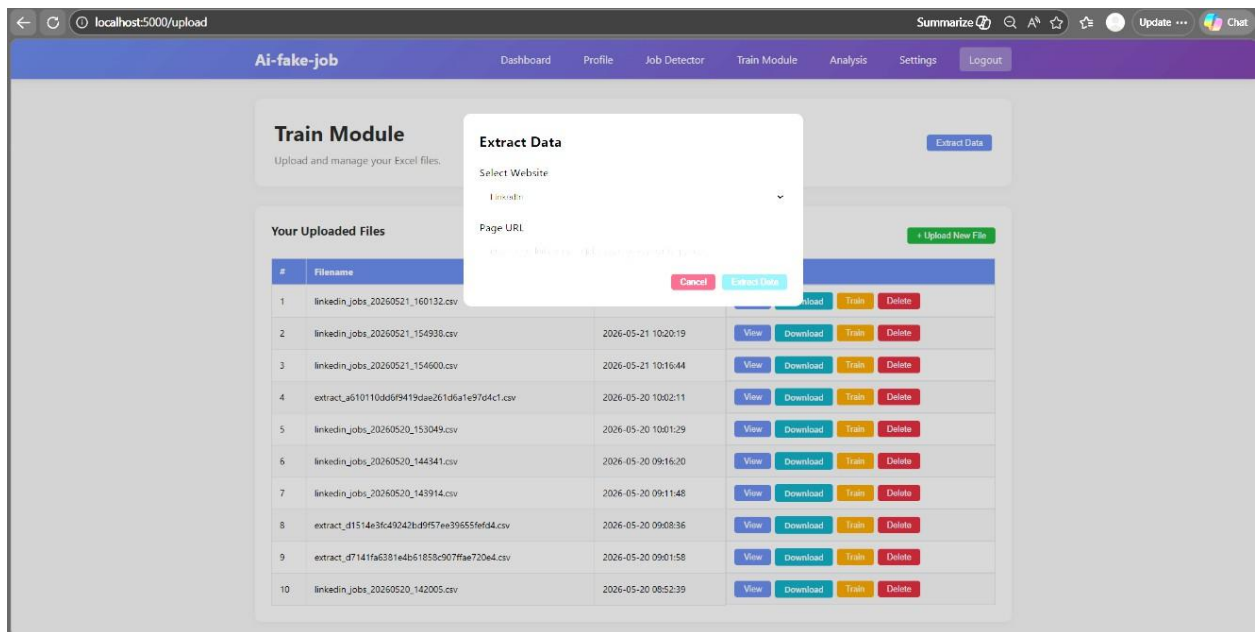


Figure 13: Training Module for Dataset Management

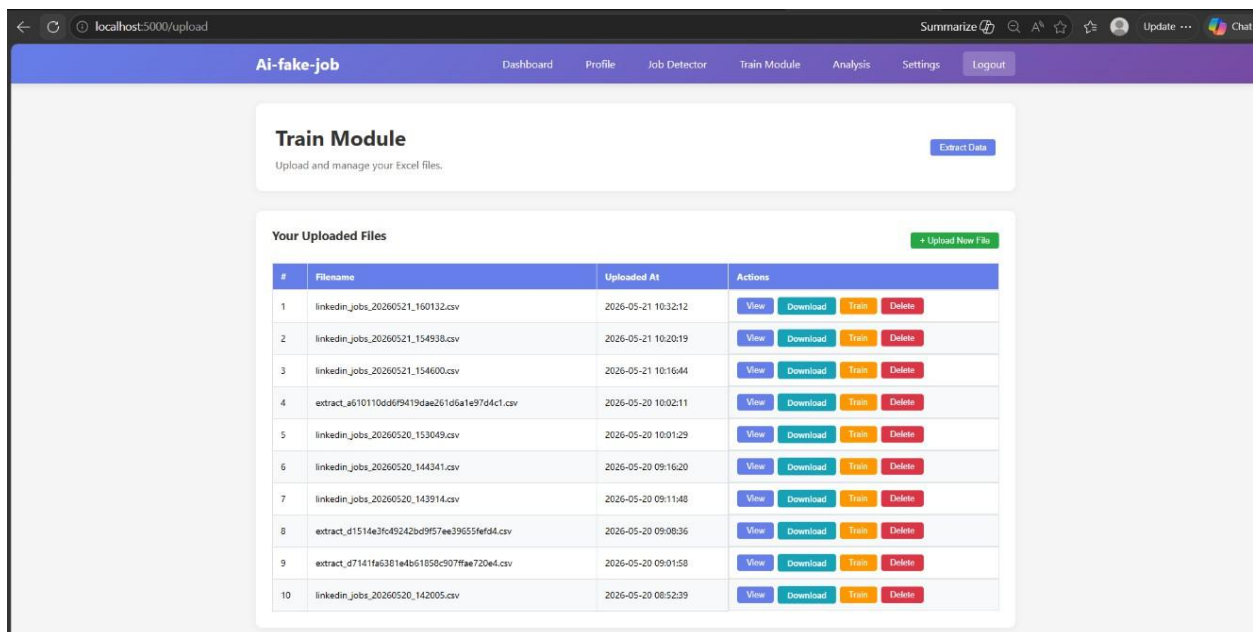


Figure 14: Dataset Upload and File Management Interface

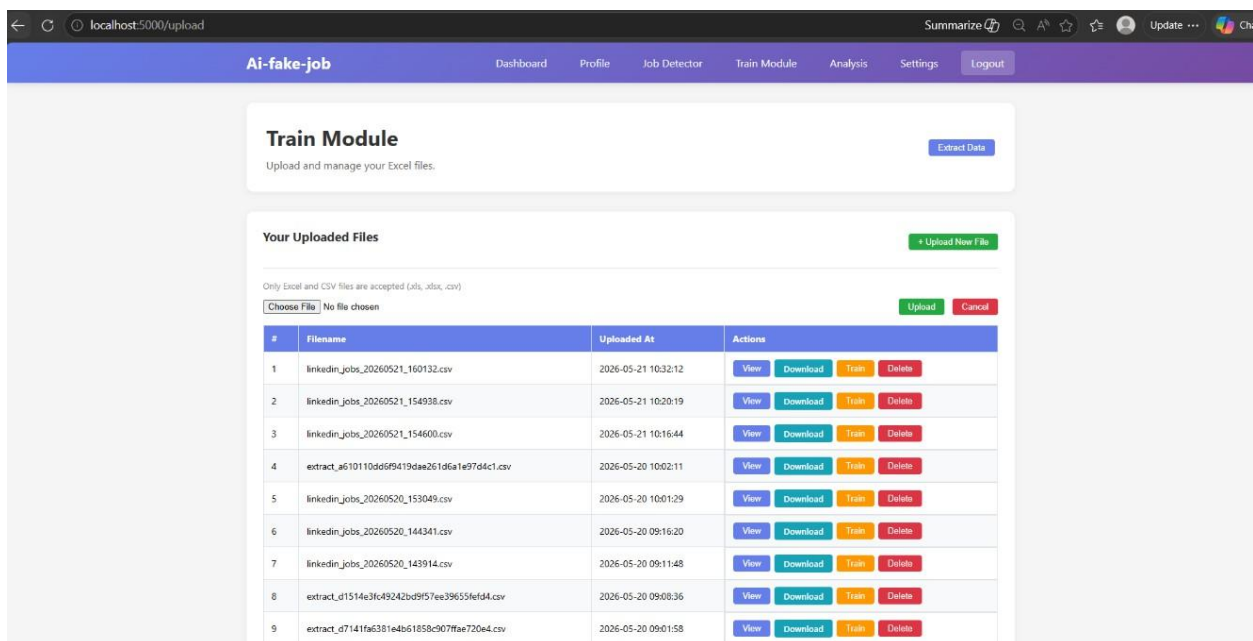


Figure 15: Job Data Extraction Module

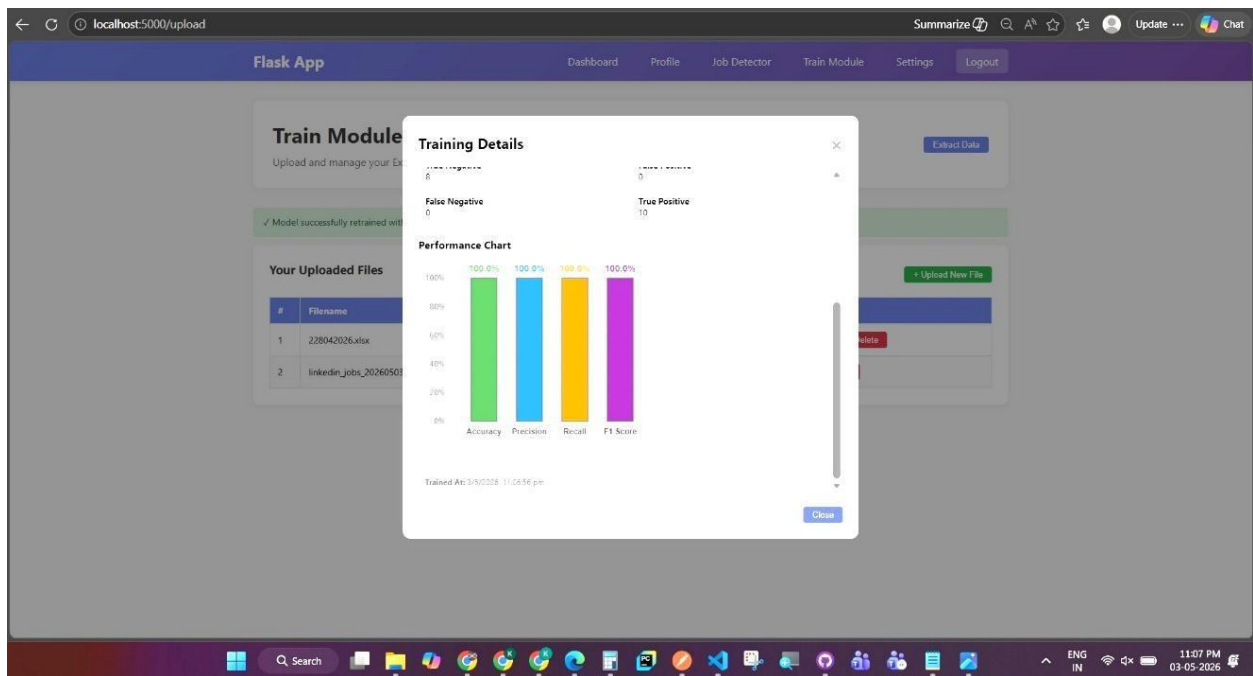


Figure 16: Performance Evaluation of the Fake Job Detection Model

The screenshot shows the "Analyze Job Post" form in the Flask App. The form is titled "AI Against Fraud: Intelligent Detection of Fake Job Listings" and "Advanced Machine Learning Analysis for Online Recruitment Scam Detection". It includes a "View ML Model Details" link. The form fields are: "Select from Scraped Data (Optional)" (dropdown menu), "Job Title" (text input), "Company Name" (text input), "Salary Range" (text input), "Location" (text input), and "Job Description" (text area). The "Analyze Job Post" button is at the bottom.

Job Title: DevOps Engineer - Python
Company Name: Facebook (Meta)
Salary Range: e.g. \$50,000 - \$80,000
Location: Menlo Park, CA
Job Description: Automate infrastructure with Python. Build deployment tools. CI/CD pipelines using Python. Work with Kubernetes/Docker. Requirements: 4+ years DevOps experience. Python scripting. Kubernetes/Docker expertise.

Figure 17: Fake Job Detection Analysis Result

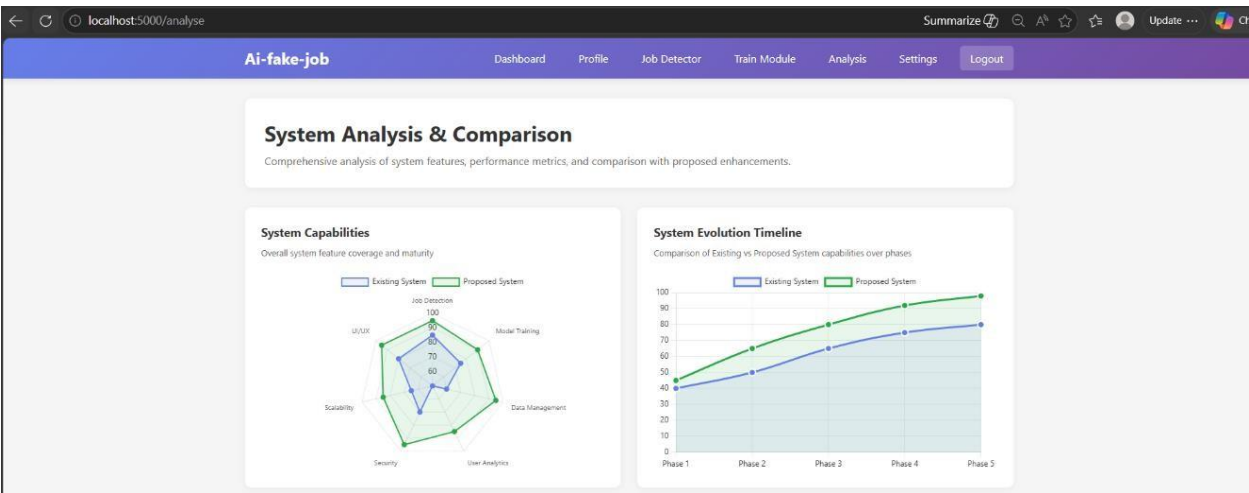


Figure 18: System Analysis and Comparison Dashboard

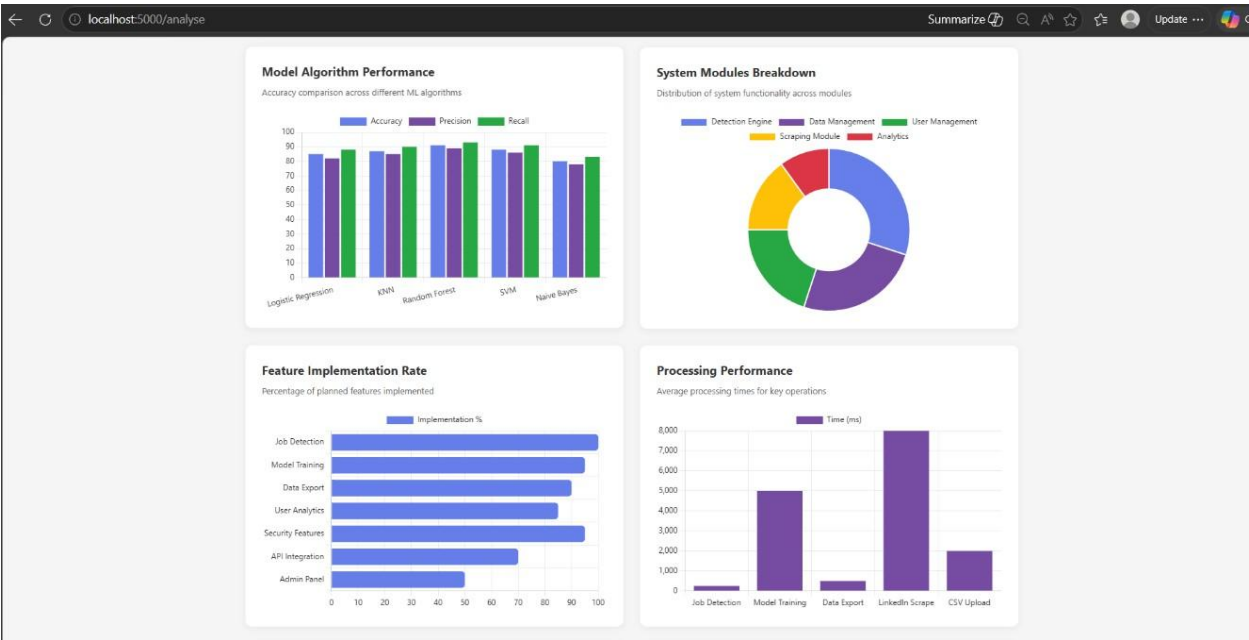


Figure 19: Machine Learning Algorithm Performance and System Module Analysis

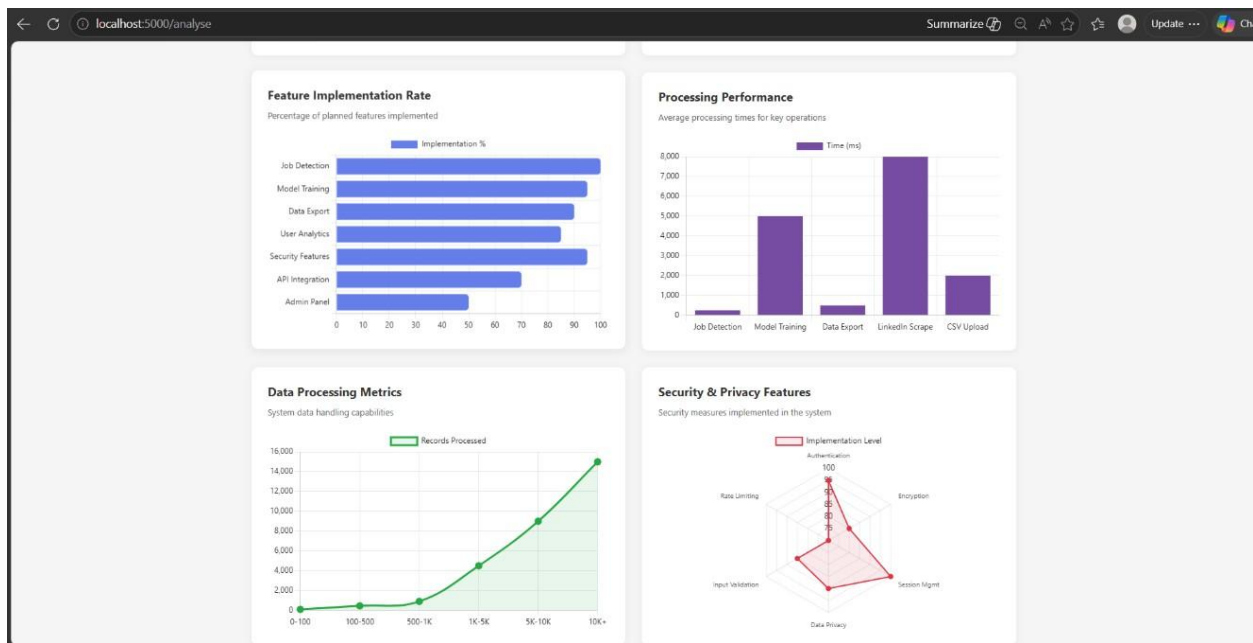


Figure 20: Feature Implementation and Security Performance Analysis

The test fold has only 7 rows. A single misclassification changes accuracy by ~14 percentage points. Random Forest's perfect score, in particular, is consistent with memorization: 200 TF-IDF features drawn from only 27 training documents is a features-greatly-exceed-samples regime in which a 100-tree ensemble can easily achieve zero training/test error without having learned a generalizable decision boundary. This table should be treated as a reproducible snapshot of current behavior on this exact dataset and split — not as a benchmark comparable to studies evaluated on EMSCAD-scale corpora (thousands of rows).

No raw counts beyond the confusion matrices above are available; no cross-dataset (e.g., EMSCAD/Kaggle) evaluation has been performed.

5. Discussion

Answering the Research Objective

The system meets its stated integration objective: a transparent, switchable TF-IDF/classical-classifier pipeline is embedded in a working, authenticated web product that logs predictions, persists retraining metrics, and offers an operator-driven path to grow its training data. Whether it meets a detection-accuracy objective comparable to published work cannot yet be claimed — see §6.4.

Contextualization

The recall-heavy profile of the default KNN configuration (100% recall, 66.67% precision) is a defensible operating point for a scam filter, where missing a real scam (false negative) is typically costlier than flagging a legitimate posting for review (false positive) — but on a 7-row

test fold this is a single false positive, not a validated precision/recall trade-off. Compared to EMSCAD-scale literature (Vidros et al., 2017) and Kaggle-corpus studies reporting high accuracy on thousands of labeled postings, this system's headline numbers are visually similar but are not on a comparable footing: they derive from roughly two orders of magnitude fewer labeled examples, with no cross-validation. Before any comparative claim against that literature is made, this pipeline should be re-evaluated on EMSCAD or its Kaggle derivative under the same protocol.

Implications

The practical contribution here is architectural: a full authentication and audit-logging layer, a user-triggered retraining loop, and a live-data-extraction path around a deliberately simple, swappable classifier core. This suggests that for a small-scale or early-stage moderation tool, the engineering investment is better spent on the data-collection and retraining loop (getting more labeled examples in front of the model over time) than on a more sophisticated classifier trained once on a fixed, tiny corpus.

Limitations

Dataset size. 34 total rows (27 train / 7 test) is far too small to support a generalizable accuracy claim; every metric in §5 is a point estimate with no confidence interval.

No cross-validation. A single stratified split is used; k-fold cross-validation (or repeated random splits) would substantially reduce the risk that the reported numbers reflect a favorable/unfavorable split rather than true model quality.

Likely overfitting in the Random Forest configuration. 200 TF-IDF features against 27 training documents is a features-exceed-samples regime; a perfect training/test score under these conditions should not be interpreted as evidence Random Forest generalizes better than the other two classifiers.

No public-benchmark validation. The system has not been evaluated against EMSCAD or the Kaggle "Fake Job Postings" corpus, so comparisons to published literature in §6.2 are qualitative, not empirical.

LinkedIn scraping and Terms of Service. The live-extraction feature automates access to LinkedIn, which its Terms of Service prohibit; this must be addressed explicitly (legal basis, restriction, or removal) before any public description of the system, and the scraper is itself unreliable in practice — LinkedIn's anti-bot measures cause frequent timeouts/blocks, per the error handling already present in `app.py`.

No explainability layer. The system currently returns a confidence score, not a feature-level explanation (e.g., SHAP/LIME); an operator sees that a posting was flagged, not why.

Unpinned/undocumented dependencies. `requirements.txt` pins version ranges, not exact versions, and does not list `selenium` or `webdriver-manager`, which `linkedin_scraper.py` imports under a guarded `try/except` — reproducibility of the full system (including the extraction feature) requires documenting these.

Single language. TF-IDF stop-word removal is English-only; non-English postings are not specifically handled.

6. Conclusion

This study describes and empirically exercises a lightweight, transparent job-posting fraud detector: TF-IDF vectorization feeding one of three switchable classical classifiers, embedded in an authenticated Flask/SQLite web product with prediction logging, operator-triggered retraining, and a live data-collection path. On the project's current 34-row dataset, all three classifiers achieve 85–100% accuracy on a single 7-row held-out fold; these numbers are real and reproducible on this exact data and split, but the dataset is too small to support a generalization claim. The system's principal contribution is the deployable integration around the model, not the model itself. Before publication, the authors should (a) substantially expand or validate against a public benchmark (EMSCAD or its Kaggle derivative), (b) replace the single-split evaluation with cross-validation, and (c) resolve the LinkedIn-scraping ToS question directly rather than leaving it implicit.

REFERENCES

1. (Real, verified papers/venues — import each into a reference manager to capture complete, correctly formatted bibliographic entries before submission.)
2. Vidros, S., Kolias, C., Kambourakis, G., & Akoglu, L. (2017). Automatic Detection of Online Recruitment Frauds: Characteristics, Methods, and a Public Dataset. *Future Internet*, 9(1), 6. <https://doi.org/10.3390/fi9010006> — introduces the EMSCAD benchmark (17,880 annotated job ads); the natural cross-dataset validation target for this system.

3. Detection of Fake Job Postings by Utilizing Machine Learning and Natural Language Processing Approaches. *Neural Processing Letters* (2021). <https://doi.org/10.1007/s11063-021-10727-z> (verify full author list) — TF-IDF/NLP feature extraction combined with supervised classifiers on the EMSCAD-derived corpus.
4. Salton, G., & Buckley, C. (1988). Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5), 513–523. — foundational TF-IDF reference.
5. Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32.
6. Cover, T., & Hart, P. (1967). Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1), 21–27.
7. Kaggle, Real or Fake: Fake Job Posting Prediction dataset (EMSCAD-derived, ~18,000 postings) — candidate corpus for cross-dataset validation of this system's pipeline.