**Profile Authenticity Scorer — Detailed Report**

**Phase 1: Project Setup and Planning**

**Directory Structure**

arduino

CopyEdit

profile\_scorer/

├── app/

│ ├── api/

│ ├── core/

│ ├── models/

│ ├── utils/

│ └── main.py

├── data/

├── models/

├── tests/

├── config.json

├── Dockerfile

├── requirements.txt

└── README.md

**Configuration File: config.json**

* Contains thresholds for scoring authenticity
* List of regex buzzwords to flag common spam/exaggeration patterns
* Model fallback toggle for future use

**Requirements File: requirements.txt**

text

CopyEdit

fastapi

uvicorn

pydantic

scikit-learn

pandas

numpy

nltk

joblib

textstat

**Phase 2: Dataset Collection and Preparation**

* Real bios collected internally or scraped
* Fake bios generated via ChatGPT or manually exaggerated
* Data labeled as authentic or likely\_fabricated
* Saved as final\_dataset.json or CSV

**Phase 3: Model Development**

**Data Loading and Preprocessing**

* Loaded JSON data into a pandas DataFrame
* Combined headline and bio text into one feature column
* Cleaned text (lowercasing, removing non-alphanumeric characters)
* Computed readability scores using textstat.flesch\_reading\_ease
* Calculated buzzword overlap from a predefined list
* Created binary labels: 1 for authentic, 0 for fake/exaggerated

**Feature Extraction**

* Extracted TF-IDF features with unigrams and bigrams, removing stopwords using NLTK’s English stopwords list
* Combined TF-IDF sparse matrix with numeric features (readability, buzzword overlap)

**Model Training**

* Train/test split: 80/20 stratified split
* Model: Logistic Regression with max 1000 iterations
* Metrics: Accuracy, Precision, Recall, F1, ROC AUC
* Saved the trained model and vectorizer as model2.pkl and vectorizer2.pkl

**Phase 4: API Development (FastAPI)**

**app/main.py**

* FastAPI app initialized
* Pydantic models ProfileData and ProfileRequest to validate inputs
* Routes:
  + GET /health: returns status OK
  + GET /version: returns app version
  + POST /check-profile: takes user profile and returns authenticity score and verdict

**app/predictor.py**

* Loads the saved model and vectorizer
* Loads configuration from config.json
* Helper functions:
  + clean\_text(text): cleans input text
  + regex\_flag\_count(text): counts regex pattern hits for buzzwords
* Main function predict\_profile(headline, bio):
  + Cleans and vectorizes combined headline and bio
  + Computes readability score
  + Counts buzzword regex hits separately for headline and bio
  + Combines TF-IDF features with numeric features for model input
  + Predicts probability that profile is authentic
  + Applies penalty to score based on buzzword count
  + Determines verdict: authentic, borderline, or likely\_fabricated based on thresholds
  + Provides detailed reason string explaining verdict
  + Flags specific fields (headline, bio) for buzzwords, empty text, or low readability

**Detailed Function Descriptions**

**1. clean\_text(text)**

* Converts input text to lowercase
* Removes non-alphanumeric characters (punctuation, symbols)

**2. regex\_flag\_count(text)**

* Uses precompiled regex patterns from config.json to count how many buzzword phrases appear in the text

**3. predict\_profile(headline, bio)**

* Combines headline and bio into a single string
* Cleans text and transforms with TF-IDF vectorizer
* Computes readability using textstat
* Counts buzzword hits in headline and bio separately and sums them
* Forms final input by horizontally stacking vectorized text and numeric features
* Predicts probability of authenticity using the logistic regression model
* Applies a penalty of 0.05 per buzzword hit to the score to reduce authenticity confidence
* Determines verdict according to thresholds:
  + = 0.75: authentic
  + = 0.5: borderline
  + else: likely fabricated
* Builds a reason string citing buzzword hits and readability
* Flags fields if empty, contain buzzwords, or have low readability
* Returns a dictionary with authenticity score, verdict, reason, and flagged fields

**How to Run**

1. **Install dependencies:**

nginx

CopyEdit

pip install -r requirements.txt

1. **Train the model:**  
   Run the Jupyter notebook or training script (training\_model.ipynb) to generate model files in models/.
2. **Run the API server:**

nginx

CopyEdit

uvicorn app.main:app --host 0.0.0.0 --port 8000

1. **Test endpoints:**
   * Health check: GET /health
   * Version check: GET /version
   * Profile check: POST /check-profile with JSON payload:

json

CopyEdit

{

"user\_id": "user\_123",

"profile\_data": {

"headline": "Visionary Thought Leader",

"bio": "Innovative professional with a proven track record..."

}

}