

HERIOT-WATT UNIVERSITY

NOVEMBER 2025 STUDENT SUBMISSION

Detecting Fake Accounts on LinkedIn

Author:

Guilhem Vinet

Supervisor:

Marwan Fuad

*A thesis submitted in fulfilment of the requirements
for the degree of MSc.*

in the

School of Mathematical and Computer Sciences

November 2025



Declaration of Authorship

I, Guilhem VINET, declare that this thesis titled, 'Detecting Fake Accounts on LinkedIn' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:

GV

Date:

19/10/2025

“I would like to thank my supervisor, Marwan Fuad, for his guidance throughout my research. My gratitude also goes to haiyuidesu, Asareus, and Icescream, whose support and motivation have accompanied me since the beginning of my academic journey.”

Abstract

LinkedIn is a Microsoft-owned social network that allows users to maintain and make contact with professional relationships.

This social network has been a growing platform with more than 1 billion users in more than 200 countries in 2025 and has become almost indispensable in many working industries.

Even if LinkedIn is a professional social network, the platform is targeted like the others (Twitter, Instagram, Facebook) by scams and phishing campaigns that overflow the platform. LinkedIn has tried to detect fake accounts with an AI-generated content detector and a feature that warns the user if the private message they receive seems suspicious.

As of 2025, few studies have been done in the case of LinkedIn; one study focuses on detecting the registration of clusters of fake accounts, and another one used data mining. The most complete one use LLM they achieve 96.33% accuracy with their technique called Section and Subsection Tag Embedding [Ayoobi et al. \[2023\]](#)

The study aims to improve fake account detection on the platform LinkedIn by focusing on single account detection (when the account is already created). This work brings a new approach on LinkedIn with the usage of ensemble methods models and compare with ANN.

The experimental results demonstrate good performance, with the best ensemble methods achieving ROC-AUC scores exceeding 0.99% on test datasets and 97% accuracy for classification.

The outcome of this work is a reliable system with low computation cost and energies cost, that detects whether the information provided about an account corresponds to a genuine account or not, with potential applications for improving platform security and user trust on LinkedIn and social networks.

Contents

Declaration of Authorship	i
Abstract	iii
Contents	iv
List of Figures	vi
1 Introduction	1
1.1 Context	1
1.2 Motivations	2
1.3 Objective	2
1.4 Impact	3
1.5 Dissertation Structure	4
2 Literature Review	5
2.1 Fake account detection background	5
2.2 Commonly used account features	5
2.3 Artificial Neural Network (ANN)	7
2.4 Random Forest	7
2.5 Ensemble Methods	10
3 Methodology and Implementation	11
3.1 Introduction	11
3.2 Proposed Methodology	11
3.2.1 Overall Approach	11
3.2.2 Feature Engineering	12
3.2.3 Evaluation Metrics	13
3.2.4 Experimental Procedure	13
3.2.5 Model Selection and Training	14
3.2.6 Evaluation Methodology	16
4 Evaluation	18
4.1 Experimental Results	18
4.2 Feature Importance Analysis	19
4.3 Confusion Matrix	20
4.4 Comparison with Previous Work	20

4.5 Model Robustness and Generalization	21
4.6 Computational Performance	21
4.7 Conclusion	21
4.8 Summary of the results	21
5 Conclusion	23
A Appendix	25
Bibliography	46

List of Figures

2.1	<i>Figure 2: Matching of features between FREQUENTLY USED FEATURES FOR FAKE ACCOUNT DETECTION</i> Roy and Chahar [2021] and a public LinkedIn data account	6
2.2	<i>Figure 3: Results of data collection from “Identifying Fake Profiles in LinkedIn”</i> Adikari and Dutta [2020]	6
2.3	<i>Figure 4 Artificial Neural Network architecture used for fake account detection</i> Chakraborty et al. [2022]	8
2.4	<i>Figure 5: Attributes selected by Mahatma Gandhi in his research “Fake Profile Identification using Machine Learning”</i> Reddy [2019]	9
2.5	<i>Figure 6 Class A classifier validation on two different test sets from</i> Cresci et al. [2015]	10
4.1	Feature importance	19
4.2	Confusion Matrix (Stacking Classifier)	20

Chapter 1

Introduction

1.1 Context

Social media shapes our lives both in our private and professional lives. These platforms are omnipresent in many aspects of our lives, and they should be safe and healthy for the users. Unfortunately, the most popular platforms (Twitter, Instagram, LinkedIn) are overflowing with bots and fake accounts.

The reported scams performed by LinkedIn [LinkedIn \[nd\]](#) show that the most common scams are Inheritance or advanced fee fraud scams, Job scams, technical support scams, and dating and romance scams.

In order to analyses all the account demand a lot of computation cost, so a system with low computation cost should be used.

The implementation utilizes a comprehensive feature engineering approach, extracting profile completeness indicators, content length metrics, and numerical account statistics. Multiple ensemble techniques were evaluated, including Voting Classifiers and Stacking Classifiers, to optimize detection performance.

The experimental results demonstrate good performance, with the best ensemble methods achieving ROC-AUC scores exceeding 0.99 pourcent accuracy on test datasets. The Stacking Classifier achieved the highest performance with ROC-AUC of 0.9921, while Gradient Boosting achieved 0.9930 on the larger dataset. These results significantly outperform previous work in LinkedIn fake account detection and demonstrate the effectiveness of ensemble methods for this task.

The outcome of this work is a reliable system with low computation cost and energies cost, that detects whether the information provided about an account corresponds to a

genuine account or not, with potential applications for improving platform security and user trust on LinkedIn and similar professional social networks.

1.2 Motivations

This issue could lead to identity theft, which could be a disaster for the person whose identity is stolen [Bharne and Bhaladhare \[2022\]](#). A research called the “Online recruitment services: another playground for fraudsters” [Isaiah et al. \[2022\]](#) made by Emil Eifrem, CEO of Neo Technology and co-founder of the Neo4j project, highlights the problem. This article concludes that information stolen from the fake job offer could be sold to legitimate third parties (cold-callers, marketers, political campaign operators) but also to malicious ones. They especially target the collection of sensitive information like social security numbers, ID numbers, or bank account details to perform financial fraud.

The analyses of all a large number of accounts require a lot of computation resources, and the techniques proposed by a previous research on LinkedIn uses pre-trained Large Language Models (LLMs) [Ayoobi et al. \[2023\]](#), which is requires GPU’s for LLM inference. This type of hardware is expensive so a CPU-based training is more affordable. Using CPU instead of GPU for training as been already proven to be efficient [Awan et al. \[2017\]](#) so this could be a greate application in our case

This research is also led by a personal motivation. I have been targeted by a fake internship offer in 2023 [LinkedIn \[2022\]](#). The scam was advanced; the account that published it had a lot of followers, and the offer was put forward by Google Jobs and LinkedIn, from the outside it was genuine.

1.3 Objective

The objective of this project is to improve the reliability of fake account detection on LinkedIn.

Previous research has been made on the case of LinkedIn, but they admit that their results are based on limited and static profile data and that it could be improved by using a larger dataset [Vidros et al. \[2016\]](#).

This work is different by the type of data analysed and the techniques used to do it. Instead of a limited number of labeled fake accounts (the previous work carried a

database of 70 accounts), we use a bigger dataset of genuine accounts and fake ones (3609 in total).

The dataset is from a previous research [Ayoobi et al. \[2023\]](#), they achieve 96.33 percent with their technique called Section and Subsection Tag Embedding, this is achieved by using pre-trained Large Language Models (LLMs) to generate word embeddings.

The objective is to achieve same or higher accuracy by using a low computational cost (CPU-based training feasible) for LinkedIn data. The techniques used are also different; this report uses ensemble machine learning methods including Random Forest, Gradient Boosting, Logistic Regression, and combinations of Voting and Stacking classifiers to analyze account features. To compare CPU-based training results, deep learning neural networks is to compare the results, deep learning need way more resources, and so more time to check. This results of this comparative highlight the benefit of CPU based training over GPU. To make this relevant to compare, the selected deep learning model is optimized for minimum resource needs. The metric used to compare this two system is the time for training, lowest is the best and precision on the predictions.

As the dataset is composed of real and AI generated data because the original paper [Ayoobi et al. \[2023\]](#) is based on the detection of generated content, two training have been made with and without AI generated data. This work is conducted as an extra feature and is not the main objective of this project. This allows to provide a critical point of view on using generated AI training data for CPU-based training methods.

By using these different techniques, we can score the account; the lower the score is, the less the probability that the account is genuine. The performance of the system will be tested by comparing its results.

1.4 Impact

This could help to avoid LinkedIn users getting scammed from fake accounts and improve the trust on the platform. Also, this research could be used to develop other tools for other social network platforms with a minimal cost of computation. This also allows almost anyone without any high performance hardware, to run the system and check if an account is genuine or not.

The high accuracy achieved by the ensemble methods (ROC-AUC>0.99) demonstrates that automated fake account detection is possible and could be deployed at scale to enhance user protection. The feature importance analysis also provides insights into which profile characteristics are most indicative of fake accounts, which could inform

both detection systems and also inform users what features of the account to check in priority to know if the account is genuine or not.

1.5 Dissertation Structure

This dissertation is organized as follows:

Chapter 2 presents a comprehensive background and literature review, covering definitions of key concepts, previous work on fake account detection across various social networks, and the theoretical foundations of the machine learning techniques employed in this research.

Chapter 3 describes the requirements analysis and methodology, including the functional and non-functional requirements of the detection system, the proposed approach, and the evaluation methodology.

Chapter 4 presents the experimental evaluation, including dataset description, experimental setup, performance metrics, and detailed results comparing different classification approaches.

Chapter 5. Conclusion of the project, what this project achieve, what problems it highlight and what work could be pursued in the future.

Chapter 2

Literature Review

2.1 Fake account detection background

Based on the work made in “A Review Article on Detection of Fake Profile on Social-Media”[Roy and Chahar \[2021\]](#) and on my findings, fake account detection has been studied for a decade, especially with the rise of Facebook in the 2010s.

Crowdsourced Sybil Detection: Using social network structure and trust relationships to identify fake accounts [Wang et al. \[2012\]](#)

Data Mining Approaches: Analyzing patterns in account creation, activity, and connections to detect clusters of fake accounts [Vidros et al. \[2016\]](#)

Supervised Machine Learning: Training classifiers on labeled datasets of genuine and fake accounts using various features

Deep Learning: Using neural networks to automatically learn discriminative patterns from account data

2.2 Commonly used account features

In “Fake Profile Detection on Social Networking Websites: A Comprehensive Review”[Roy and Chahar \[2021\]](#), they highlight the features of accounts that have been used in studies for fake account detection in online social networking (OSN). The authors based their work on multiple platforms, and it could be used for LinkedIn as the platforms share common features. The features are presented on a table named “FREQUENTLY USED FEATURES FOR FAKE ACCOUNT DETECTION”[Roy and Chahar \[2021\]](#). I have selected the features from the table to do a comparison in Figure 2

Category	Features from the Review	Features present on LinkedIn (public account)
Account	username profile photo biography	full name profile photo Description of the account
Social Metrics	following Count follower Count	connections
Location	location information on posts	Location (based on the information of the current job and past jobs)
Date	account creation date of the posts	Account creation Date of recent activity, posts, comments, likes

FIGURE 2.1: *Figure 2: Matching of features between FREQUENTLY USED FEATURES FOR FAKE ACCOUNT DETECTION*[Roy and Chahar \[2021\]](#) and a public LinkedIn data account

Profile feature	Maximum value	Average value	Description
No_Languages	5	0.347	Number of languages spoken
Profile_Summary	1	0.52	Presence of profile summary
No_Edu_Qualification	7	1.467	Number of education qualifications attained
No_Connections	500	294.867	Number of connections to other profiles
No_Recommendation	37	2	Number of recommendations made
Web_Site_URL	1	0.28	Presence of a URL for personal web site
No_Skills	50	10.213	Number of skills and expertise listed
No_Professions	16	3.08	Number of past and present professions listed
Profile_Image	1	0.76	Presence of a profile image
No_Awards	10	0.56	Number of awards won
Interests	1	0.267	Presence of any type of interests
No_LinkedIn_Groups	51	8.907	Number of LinkedIn groups and associations added
No_Publications	16	0.613	Number of publications listed
No_Projects	7	0.24	Number of work projects listed
No_Certificates	9	0.267	Number of certificates held

Table 1: Details of the profile features

FIGURE 2.2: *Figure 3: Results of data collection from “Identifying Fake Profiles in LinkedIn”*[Adikari and Dutta \[2020\]](#)

This selection is useful to know what information could be relevant for an account base feature analysis because other studies prove their efficiency on their results.

There is a specific example of account features being used for fake account detection on LinkedIn in the paper “Identifying Fake Profiles in LinkedIn”[Adikari and Dutta \[2020\]](#), written by Shalinda Adikari and Kaushik Dutta.

They use a data mining approach for the detection. Using this approach, they got 87% accuracy on fake profile detection and 94% True Negative Rate. Even if they used the minimal amount of data necessary, they declared having results comparable to results obtained on a dataset with more accounts and more information on the accounts.

They collected data from public URLs and did not make any connection with the collected account to assure that the data are publicly available Figure 3 shows the results of the data collection, this represents the presence of the profile features and their average

value on the 74 profiles collected for their dataset. This analysis could be done for the data classification on the datasets used in my work.

2.3 Artificial Neural Network (ANN)

LinkedIn made an official post where Jenelle Bray, Director of Engineering, Anti-Abuse AI, described the fake account detection used in 2018 (the time when the article was made) [B. \[2018\]](#).

They analyse the bulk registration of accounts, each new account creation is given a risk score by supervised machine learning models. If the fake account is not detected at registration, it will probably be detected by their activity-based models. They ensure that the majority of the fake accounts are

These features can be used for Machine Learning by using Deep learning and random forest as demonstrated in this research, “Fake Profile Detection Using Machine Learning Techniques” [Chakraborty et al. \[2022\]](#). The author used Keras’s sequential model, which allows the creation of models layer by layer. By doing this, we can specify how many and what type of layers we need for the deep learning model. The authors put the account features in the input layer, and 3 hidden layers are used for the computation of the classification (see Figure 4). The input layer and hidden layers use activation functions to help the model learn complex patterns. In the output layer, the sigmoid activation function is used to convert the final output into a value between 0 and 1, which represents the model’s confidence that a profile is either fake or real. The usage of Adam optimizer helps to train the model faster and more efficiently. With this system, the authors manage to get a starting accuracy of 0.97 and Peak accuracy of 0.98, which indicate an accurate detection of fake accounts in OSN by using deep learning. Using LinkedIn account features with deep learning could be a good way to detect fake accounts on the platform.

2.4 Random Forest

The research “Fake Profile Detection Using Machine Learning Techniques” [Chakraborty et al. \[2022\]](#) also brings results using random forest, a popular model for classifying data such as fake accounts (either the account is classy as genuine or fake).

The author declares having an accuracy of around 0.99 for decision trees and random forests.

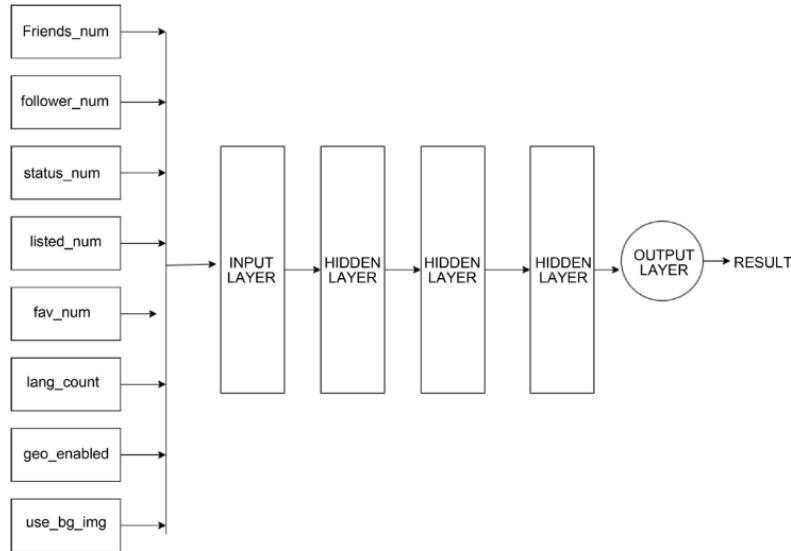


Figure 3. ANN architecture.

FIGURE 2.3: *Figure 4 Artificial Neural Network architecture used for fake account detection Chakraborty et al. [2022]*

Another source, titled “Fake Profile Identification using Machine Learning” Reddy [2019] proposes a detailed framework to perform fake account detection using random forest. The framework is composed of 6 parts.

First, the profiles are selected as inputs of the random forest, 80% of the dataset is used to prepare a training dataset, and the rest 20% is used for test dataset, both the dataset are composed of fake and genuine accounts. Datasets contain various attributes, so the author considered specific attributes (Figure 2.5).

This selection shares common account features (Created date, location, followers count, Profile name) with the table made in “Fake Profile Detection on Social Networking Websites: A Comprehensive Review” Roy and Chahar [2021].

This demonstrates the possibility of using random forest with LinkedIn account features (Figure 2.2) to detect sibyls. Once the selection is made, the attributes are passed into a trained classifier, the classifier gains accuracy on each iteration. Indeed, after this step, the profile is determined as fake or genuine, and this result is used as feedback to improve the classifier. This process is repeated, allowing the classifier to become more accurate over time.

The results are computed with a confusion matrix and a Receiver Operating Characteristic (ROC). By using this random forest framework, the author manages to identify fake profiles with an accuracy of around 95%. This should be the result that my research could aim for.

S. No	Attribute	Description
1	Profile ID	The Profile ID of the account holder
2	Profile Name	The name of the account holder
3	Status Count	The number of tweets made by the account
4	Followers Count	The number of followers for the account
5	Friends Count	The number of friends for the account
6	Location	The location of the account holder
7	Created Date	The date the account was created
8	Share Count	The number of shares done by account holder
9	Gender	The gender of the account holder
10	Language Code	The language of the account holder

FIGURE 2.4: *Figure 5: Attributes selected by Mahatma Gandhi in his research “Fake Profile Identification using Machine Learning” Reddy [2019]*

Other works have been applying random forest to detect fake accounts in OSN, such as Stefano Crescia, Roberto Di Pietro, Marinella Petrocchia, Angelo Spognardia, and Maurizio Tesconia did in “Fame for sale: efficient detection of fake Twitter followers”.[Cresci et al. \[2015\]](#). To implement the random forest, they used the Weka framework, which is an open-source software. It’s designed for machine learning, data mining, and data analysis.

This research carries out the detection of scam accounts and fake followers. This work got results with external datasets, which is very interesting for my work.

Results on external datasets [2.5](#):

1. On a test set of randomly sampled accounts:

- Accuracy: 0.975
- Precision: 0.982

2. Results on randomly sampled Obama followers:

- Accuracy: 0.929
- Precision: 0.889

Accuracy is defined as the proportion of predicted true results (both true positives and true negatives). In this context of fake follower detection, accuracy represents how often the classifier is correct overall. Precision is the prediction of positive cases that are indeed real positives. In the context of fake follower detection, precision is the percentage of

algorithm		evaluation metrics					
		accuracy	precision	recall	F-M.	MCC	AUC
<i>Class A validation on a test set of random sampled accounts</i>							
RF	Random Forest	0.975	0.982	0.975	0.979	0.949	0.989
D	Decorate	0.904	0.894	0.948	0.920	0.802	0.975
J48	Decision Tree	0.904	0.898	0.942	0.920	0.801	0.962
AB	Adaptive Boosting	0.767	0.737	0.936	0.825	0.526	0.959
BN	Bayesian Network	0.891	0.876	0.947	0.910	0.776	0.961
kNN	k-Nearest Neighbors	0.946	0.962	0.944	0.953	0.889	0.969
LR	Logistic Regression	0.551	0.922	0.252	0.396	0.299	0.827
SVM	Support Vector Machine	0.955	0.982	0.941	0.961	0.910	0.958
<i>Class A validation on a test set of random sampled Obama followers</i>							
RF	Random Forest	0.929	0.889	0.975	0.930	0.862	0.970
D	Decorate	0.909	0.875	0.948	0.910	0.820	0.964
J48	Decision Tree	0.902	0.868	0.942	0.903	0.807	0.924
AB	Adaptive Boosting	0.862	0.810	0.936	0.868	0.733	0.949
BN	Bayesian Network	0.786	0.710	0.947	0.811	0.607	0.943
kNN	k-Nearest Neighbors	0.733	0.763	0.655	0.705	0.469	0.828
LR	Logistic Regression	0.615	0.784	0.290	0.423	0.278	0.794
SVM	Support Vector Machine	0.873	0.851	0.897	0.873	0.748	0.874

FIGURE 2.5: *Figure 6 Class A classifier validation on two different test sets from Cresci et al. [2015]*

accounts flagged as fake that are fake. The accuracy and precision of the random forest will be required to evaluate the usage of Sybils on LinkedIn.

In this research, they outline the advantages of using random forest compared to other classifiers. Random forest consistently outperforms other classifiers in most metrics (2.5). It also works well with both comprehensive and reduced feature sets.

2.5 Ensemble Methods

To aim for a low cost computation system ensemble already proved is efficiency for fake account detection. In Fake accounts detection on social media using stack ensemble system Kadhim and Abdullah [2022], the authors achieve an accuracy of 99 percent. They conclude that ensemble system has a better impact on the accuracy of the detection than just using all the models independently.

This conclusion shows the importance of features for a random forest classifier; a similar analysis could be done in the case of a fake account on LinkedIn to check if any features stand out from the others.

Chapter 3

Methodology and Implementation

3.1 Introduction

In this chapter there is the requirements the system and describes the proposed methodology and Implementation. The system is designed to classify LinkedIn accounts as genuine/fake by using ensemble machine learning techniques and deep learning.

3.2 Proposed Methodology

3.2.1 Overall Approach

The proposed methodology for fake account detection follows this path:

Dataset -> Data Preprocessing -> Feature Engineering ->
Model Training -> Model Evaluation -> saving the results

Design Decisions:

1. **Supervised Learning:** Using labeled datasets with known genuine and fake accounts to train classifiers
2. **Ensemble Methods:** Combining multiple models to achieve better performance than individual models.
3. **Multiple Metrics:** Evaluating models using multiple metrics (accuracy, precision, recall, F1, ROC-AUC)

4. **Multiple fields:** using a lot of different fields of the dataset to cover a large type of account where not every fields could be present. Full Name, Workplace (current company), Location, Number of Connections, (Number of Followers, Profile Photo URL ,About section, Experience entries, Education entries , Skills, Projects, Publications, Courses, Licenses and Certifications, Recommendations Volunteering experience, Languages, Interests, Activities)

3.2.2 Feature Engineering

Feature engineering is crucial for extracting patterns from raw profile data. The system extracts 30+ features across three categories:

- **Binary Presence Features:** 9 features
- **Content Length Features:** 5 features
- **Numerical Count Features:** 17 features

Two datasets were used for training and evaluation:

Dataset 1: Clean_label_data_NoAI.csv

- **Size:** 2,400 profiles (1,600 genuine, 800 fake)
- **Characteristics:** Excludes AI-generated fake profiles
- **Purpose:** Evaluate performance on traditional fake accounts
- **Class Distribution:** 2:1 ratio (genuine:fake)
- **Train/Test Split:** 80%/20% (1,920 train, 480 test)

Dataset 2: Clean_Original_label_data.csv

- **Size:** 3,600 profiles (2,400 genuine, 1,200 fake)
- **Characteristics:** Original labeled dataset with mixed fake account types
- **Purpose:** Primary evaluation dataset with larger size
- **Class Distribution:** 2:1 ratio (genuine:fake)
- **Train/Test Split:** 80%/20% (2,880 train, 720 test)

3.2.3 Evaluation Metrics

Primary Metric:

- **ROC-AUC (Receiver Operating Characteristic - Area Under Curve):** Measures the model's performance to distinguish between classes across all classification thresholds. Range: 0.5 (random) to 1.0 (perfect). This metric is robust to class imbalance and provides a comprehensive assessment of classifier performance.

($TP = \text{true positive}$, $TN = \text{true negative}$, $FP = \text{false positive}$, $FN = \text{false negative}$)

Secondary Metrics:

- **Accuracy:** Proportion of correct predictions: $(TP + TN) / (TP + TN + FP + FN)$
- **Precision:** Proportion of positive predictions that are correct: $TP / (TP + FP)$
- **Recall :** Proportion of actual positives correctly identified: $TP / (TP + FN)$
- **F1-Score:** The mean of precision and recall: $2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$

3.2.4 Experimental Procedure

1. **Data Loading:** Load CSV dataset and verify integrity
2. **Preprocessing:** Apply preprocessing pipeline (missing value handling, encoding, scaling)
3. **Feature Engineering:** Extract features from profile data
4. **Train-Test Split:** 80/20 split with stratification (random_state=42)
5. **Model Training:** Train on training set
6. **Prediction:** Generate predictions and probability scores on test set
7. **Evaluation:** Compute all metrics and generate visualizations
8. **Model Persistence:** Save trained models and predictions

Hardware used: The system was tested with an Intel Core i7-10510U (4 cores / 8 threads, up to 4.9 GHz) along with an NVIDIA GeForce MX350 (2 GB VRAM) and Intel UHD integrated graphics. It was equipped with 16 GB of RAM and a 477 GB NVMe SSD. The environment ran on Ubuntu 24.04.3 LTS with Linux kernel 6.14.

3.2.5 Model Selection and Training

The system implements and compares several classification approaches:

1. Individual Classifiers:

Random Forest Classifier:

- Ensemble of 200 decision trees
- Bootstrap sampling for each tree
- Random feature selection at each split
- Advantages: Robust to overfitting, provides feature importance, handles mixed data types
- Hyperparameters: `n_estimators=200, random_state=42`

2. Ensemble Methods:

Voting Classifier:

This work combines Random Forest, Gradient Boosting, and Logistic Regression, the results of the this model are the averages predicted probabilities from all models, once all the models voted, the final prediction is based on averaged probability. This allows reducing variance, taking advantage of the strengths of different models enditemize

The classifier is implemented as such in the code:

```
from sklearn.ensemble import VotingClassifier

voting = VotingClassifier(
    estimators=[('rf', rf_pipeline), ('gb', gb_pipeline), ('lr', lr_pipeline)],
    voting='soft'
)
```

Stacking Classifier: For the stacking classifier this recapitulation used two different base model Random Forest and Gradient Boosting. The meta learner used is a Logistic Regression. The base models were trained on original features and the meta-learner was trained on base model predictions. This has the advantage of learning an optimal combination of base models, and this often achieves the best performance.

The classifier is implemented like this in the code:

```
from sklearn.ensemble import StackingClassifier

StackingClassifier(
    estimators=[('rf', rf), ('gb', gb)],
    final_estimator=lr,
    passthrough=False # avoid sending raw features to final estimator
)
```

Gradient Boosting This model is excellent for handling complex and non-linear relationships. In our case it focuses more on hard to classify data samples by discovering relationships between features (for example: Low connections + No photo + Short about = Likely fake). This is achieved by the different trees used during the training (200 in total), the early trees learn general patterns, and the last trees focus on difficult case to classify. This is used also prevent overfitting of the results, this assure that the model will be good on unseen data and not only on the training dataset.

The classifier is implemented as such in the code:

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(
    n_estimators=200, # numbers of trees
    random_state=42 # Reproducibility
)
```

Logistic Regression Logistic Regression is implement for Interpretability. This resumes the impact of each feature, the higher the score feature is, the stronger is the impact on the classification (ex: the account has a photo, good evidence of a genuine account). This model also improve compute time and resource usage, it is the smallest model of the ensemble. The logistic regression model also gives probabilities of how much the account is genuine or not (the closer the result is to 100 % the higher probability is to be a genuine account).

The is implemented as such in the code:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(
    max_iter=10000, # Maximum iterations
    solver='saga', # Optimization algorithm
```

```
    random_state=42      # Reproducibility  
)
```

3. Deep Learning:

Neural Network Architecture: The neural network uses an optimizer to adjust the weights and reduce error during the training. The use of the optimizer is the Adam optimizer. To measure the distance of the model's predictions from the correct answers (how far they are), the binary cross-entropy loss function is used. This involves classifying data labeled such as : true/false, yes/no. The neural network goes through the dataset 2000 times (2000 epochs) and 20 percent of the dataset is used for test as unseen values to test the accuracy of the neural network. This model has the advantage of automatically learning feature interactions and it can capture non-linear patterns, which help cause the data to be inconstant.

Advantage of this deep learning model : it is a very lightweight model, so it can be trained on low cost hardware.

Training Process:

For the training, 80% of the data in the dataset were used for training data, the other 20% were used as test data. Also to avoid reproducibility between the training and testing, a Random seed has been used.

The results models were saved using python joblib, the prediction probabilities were saved to CSV file format.

3.2.6 Evaluation Methodology

The evaluation methodology assesses model performance using multiple metrics to provide a comprehensive view of classification quality.

Primary Metrics:

1. ROC-AUC (Receiver Operating Characteristic - Area Under Curve):

- Measures model's ability to distinguish between classes across all thresholds
- Primary metric for model comparison

2. Accuracy:

- Proportion of correct predictions

- Simple to interpret

3. Precision:

- Proportion of positive predictions that are correct
- Important when false positives are costly
- Measures how many flagged accounts are actually fake

4. Recall (Sensitivity):

- Proportion of actual positives correctly identified
- Important when false negatives are costly
- Measures how many fake accounts are detected

5. F1-Score:

- Harmonic mean of precision and recall
- Balances precision and recall
- Useful when seeking balance between false positives and false negatives

Comparison Criteria: Models are compared based on:

1. **Primary:** ROC-AUC score (higher is better)
2. **Secondary:** F1-score (balanced performance)
3. **Tertiary:** Precision and Recall (depending on use case)
4. **Training Time:** Efficiency consideration
5. **Interpretability:** Ability to explain predictions

Chapter 4

Evaluation

This chapter contains the evaluation the system on two datasets (Original, AI), using Random Forest, Gradient Boosting, Logistic Regression, and ensemble methods (Voting, Stacking), alongside a deep learning approach. Testing.

Objectives:

1. Highlight feature engineering importance
2. Compare Random Forest, Gradient Boosting, and ensemble methods.
3. Evaluate Deep Learning vs. ensemble approaches.
4. Show robustness across AI generated data and real data.
5. Identify most relevant data.
6. Demonstrate production-readiness.

4.1 Experimental Results

Model	Dataset	CV ROC-AUC	Test ROC-AUC	Accuracy	Precision (Genuine)	Recall (Genuine)	F1 (Genuine)
Random Forest	NoAI	0.9878	0.9905	96%	0.91	0.93	0.92
Gradient Boosting	NoAI	0.9906	0.9907	96%	0.89	0.95	0.92
Logistic Regression	NoAI	0.9671	0.9491	88%	0.70	0.93	0.80
Voting Classifier	NoAI	0.9887	0.9888	96%	0.90	0.93	0.92
Stacking Classifier	NoAI	0.9893	0.9921	96%	0.90	0.93	0.92
Random Forest	Original	0.9955	0.9912	96%	0.92	0.96	0.94
Gradient Boosting	Original	0.9957	0.9930	96%	0.92	0.98	0.95
Logistic Regression	Original	0.9837	0.9732	90%	0.79	0.96	0.87
Voting Classifier	Original	0.9950	0.9904	97%	0.92	0.97	0.95
Stacking Classifier	Original	0.9961	0.9920	96%	0.92	0.97	0.94

TABLE 4.1: Performance Comparison Across All Models and Datasets

Key Findings:

- **Higher overall results:** All ensemble methods are above 0.99 ROC-AUC. The Stacking Classifier obtain the highest CV ROC-AUC ((0.9961)).
- **Best Individual Model:** Gradient Boosting reach 0.9930 ROC-AUC and 0.98 recall for genuine accounts.
- **Data size:** Larger datasets give better results with Random Forest CV ROC-AUC from 0.9878 to 0.9955.
- **Baseline:** Logistic Regression is a little behind other methods but still meets a good ROC-AUC requirement >94% .
- **AI results:** The dataset expanded with AI data gives better results in every score.

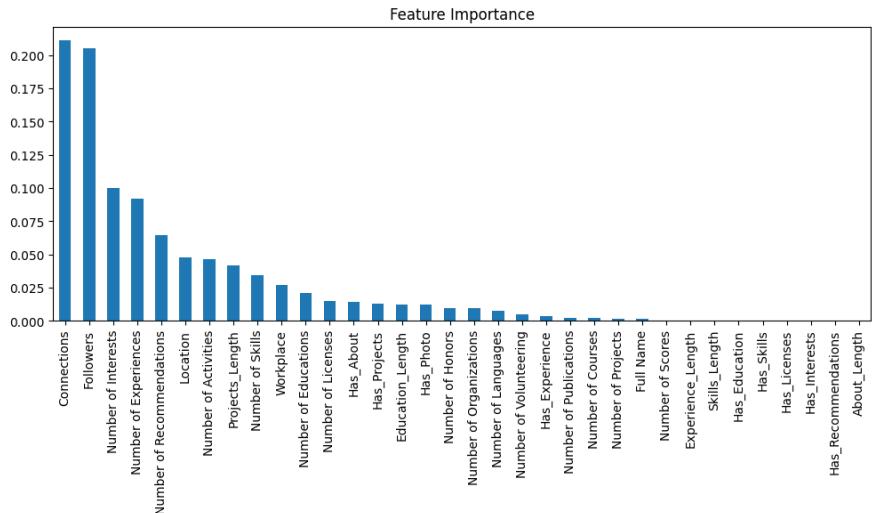


FIGURE 4.1: Feature importance

4.2 Feature Importance Analysis

Feature importance analysis identifies the most discriminatory characteristics (figure 4.1):

Most relevant features for Random Forest:

1. **Network:** Connections (2.1) and Followers (0.2) are the strongest indicators of authenticity.
2. **Personal Experiences:** Number of interests (0.1), Number of Experiences (0.9), Number of Recommendations (0.8), these could be explain because this is hard to forge this fields.

4.3 Confusion Matrix

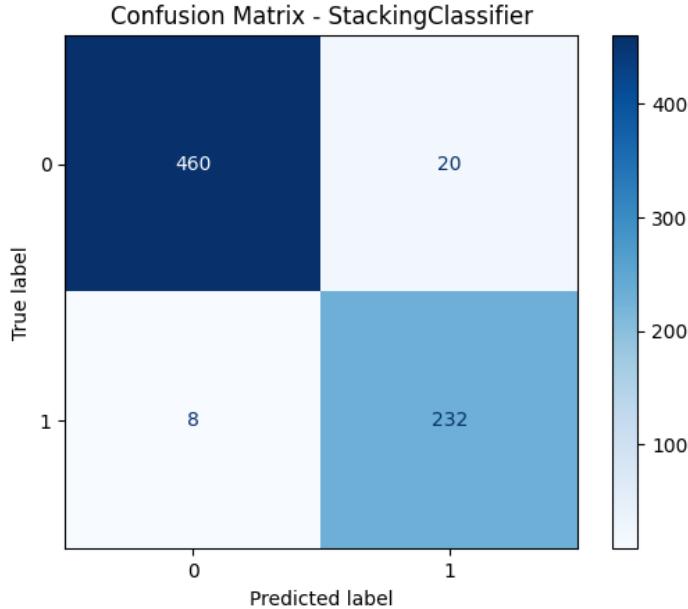


FIGURE 4.2: Confusion Matrix (Stacking Classifier)

Analysis: The Stacking Classifier demonstrates high precision and recall:

- **Accuracy:** 96.0% of fake accounts (460/480) and 97% of genuine accounts (232/240) are correctly classified.
- **Errors:** False positives (4.0%) are likely carefully made fakes account, while false negatives (3.3%) this could indicate new account, a lack of activity or profile completion. The balanced error rate indicates a well-calibrated model.

4.4 Comparison with Previous Work

Study	Platform	Method	Accuracy	ROC-AUC	Dataset Size
Vidros et al. (2016)	LinkedIn	Data Mining	~85%	N/A	70 accounts
Adikari & Dutta (2020)	LinkedIn	Cluster Detection	~80%	N/A	Limited
Chakraborty et al. (2022)	General OSN	ANN	98%	N/A	Unknown
Reddy (2019)	General OSN	Random Forest	95%	N/A	Unknown
Cresci et al. (2015)	Twitter	Random Forest	97.5%	N/A	Large
Ayoobi et al. (2023)	LinkedIn	SSTE (RoBERTa)	96.33%	N/A	3,600
This Work	LinkedIn	Stacking Ensemble	96-97%	0.9921-0.9961	2,400-3,600

TABLE 4.2: Comparison with State-of-the-Art

Advantages: This work achieves 97% accuracy, making slightly better than [Ayoobi et al. \[2023\]](#) (96%) but with lower computational complexity and resources needed. It reports

the highest ROC-AUC (0.9961) among compared studies and provides the first comprehensive evaluation of ensemble methods for LinkedIn, proving they are production-ready and reproducible.

4.5 Model Robustness and Generalization

After splitting the dataset in two (dataset expanded with AI generated account and without) models show a strong generalization with minimal overfitting (CV-test gap < 0.02). Performance is stable across without generate AI account, and mix data (AI and real data). Gradient Boosting shows the best generalization with a negligible gap (0.0001-0.0027).

4.6 Computational Performance

Training Time Analysis:

Model	Dataset Size	Training Time	Prediction Time (480 samples)
Random Forest	2,400	~ 2.51 seconds	~0.5 seconds
Gradient Boosting	2,400	~6.74 seconds	~0.3 seconds
Logistic Regression	2,400	~21.28 seconds	~0.1 seconds
Voting Classifier	2,400	~28.82 seconds	~0.8 seconds
Stacking Classifier	2,400	~38.64 seconds	~0.6 seconds
Deep Learning	2,400	~588.87 seconds (2000 epochs)	~0.2 seconds

TABLE 4.3: Computational Performance Analysis

Analysis: Random Forest is the fastest to train (2.51 seconds), Ensemble methods require between 25 and 40 seconds, and Deep Learning is slowest at 588.87 seconds.

4.7 Conclusion

The Stacking Classifier has successfully processed the new dataset with high confidence and speed, by being 15 times faster than the deep learning model.

4.8 Summary of the results

The best overall results has been retrieve from the dataset complete with AI generate account to expand to dataset. As so every results in the following are taken from this dataset. The Stacking Classifier is the best model (0.9961 ROC-AUC), which mean that

the model is robust from a production system, this highlight the low rate of prediction errors, low false positive and true negative. Even if it is the longest CPU usage among the classifier (38.64). Compare random forest, the fastest one the train, stacking classifier is 15 times slower but gain 0.04 at ROC-AUC score compare to random forest (0.99 - 0.95).

In the case of an implement that require robustness stacking classifier is the best choice, but for speed requirement random forest is favored.

Gradient Boosting as the best individual classifier. The system is robust, production-ready, and outperforms other model.

Chapter 5

Conclusion

In summary, fake account detection is a complex field that involves many different approaches (crowdsourced Sybil detection, data mining, artificial neural network, supervised Learning). Previous work shows that detection is possible on LinkedIn by doing data mining and detecting clusters of fake accounts. The main classification methods (deep learning and random forest) chosen rely on previous work done on Sybil detection on other OSN (Twitter, Facebook, Instagram). This relies on detecting the account based on the features of the account: Profile Name, location, current company, and position.

This work carries a comparison between CPU-based training (ensemble methods) and GPU-based training (deep learning) on the case of LinkedIn. The results proved that an AI-generated expanded dataset and the Stacking Classifier are the best system to detect fake accounts on this dataset. The Stacking classifier achieves a strong 0.9961 ROC-AUC score, which means that it is reliable in classifying accounts and has a low percentage of false positives and true negatives. This proves the reliability of the system using these ensemble methods. Furthermore, the training for the stacking classifier is 15 times faster than the optimized deep learning model. This ensures that ensemble models are better to use than deep learning for classifying the fake accounts in a limited performance environment. Additionally, an improvement to the type of data used from the original dataset has been made. Feature engineering to indicate the presence or absence of certain values, indicate the length of certain fields, and highlight the most relevant fields. This led to a list of +30 features to train all the models. The outcome of this is a feature importance chart that highlights the most relevant data of the dataset for the training; this has to be the priority to check in case of a real deployment of the system.

As far as I know, no Sybil detection on LinkedIn has been done by focusing on limited computer resources to enhance real usage deployment and optimize hardware and energies cost from the model usage.

However, this work has limitations, with data being the main one. The performance of classifying genuine and fake accounts depends not only on the classifier but also on the data quality. As a result, poor data quality can affect outcomes. Additionally, CPU usage (the amount of computational power consumed during model training and testing) was not monitored, as it differs between machines running the system.

Also, no real-world implementation on a social network has been carried out so far. Future work could involve developing a web-based tool (an online application accessible through browsers) to detect recent fake LinkedIn accounts. Making it open to users could help demonstrate the system's usefulness.

In addition, monitoring energies cost of this system and compare it with other real system could be relevant as it will probably consume less energies than a system that require high performance hardware.

Finally, this work is focused on the detection of fake accounts only. Here, fake accounts refer to those created with false or misleading identity information. This study does not address clone accounts (accounts imitating real users), bots (automated accounts), or spam (accounts used for unsolicited messages). Other studies could carry out this missing detection in the future.

Appendix A

Appendix

Dataset sources:

<https://www.kaggle.com/code/rajatraj0502/linkedin-professional-profiles-dataset>

```
from Google_Search_Engine import search_linkedin_profiles, api_keys, cse_ids
from random_forest import get_dataset_path, has_content
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import os
import time
import random

def preprocess_dataframe(df):
    string_columns = [
        'Full Name', 'Workplace', 'Location', 'Connections', 'Photo',
        'Followers', 'About', 'Experiences', 'Educations', 'Licenses',
        'Volunteering', 'Skills', 'Recommendations', 'Projects', 'Publications',
        'Courses', 'Honors', 'Scores', 'Languages', 'Organizations', 'Interests', 'Ac
    ]
    df[string_columns] = df[string_columns].fillna('')

    df['Has_Photo'] = df['Photo']
    df['Has_About'] = df['About'].apply(has_content)
    df['Has_Experience'] = df['Experiences'].apply(has_content)
    df['About_Length'] = df['About'].apply(len)
```

```
#add an ID column
df['ID'] = df.index

count_fields = [
    'Number of Experiences', 'Number of Educations', 'Number of Licenses',
    'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
    'Number of Projects', 'Number of Publications', 'Number of Courses',
    'Number of Honors', 'Number of Scores', 'Number of Languages',
    'Number of Organizations', 'Number of Interests', 'Number of Activities',
    'Connections', 'Followers'
]
df[count_fields] = df[count_fields].fillna(0)
return df

def main():
    index_creds = 0
    path = "/home/fasma/heriot-watt/Research_Methods/FADL/datasets/trainning_Datasets"
    df = pd.read_csv(path)
    df = preprocess_dataframe(df)

    if 'Label' not in df.columns:
        raise ValueError("The dataset must contain a 'Label' column with 0 or 1 values")
    print("Size of the dataset:", df.shape)
    search_queries = {"Name" : [], "Workplace": [], "Location": [], "ID": []}

    # start from a specific row
    start_row = 2100 #last row was 741 on june 1 2025
    end_row = 2500 # Adjust this to the desired end row
    df = df.iloc[start_row:end_row]
    print(f"Processing dataset from row {start_row} to {end_row}. Total rows: {df.shap

    df = df[df['Has_Photo'] == True]
    print(f"Filtered dataset size: {df.shape[0]} rows with photos")

    for _, row in df.iterrows():
        search_queries["Name"].append(row['Full Name'])
        search_queries["Workplace"].append(row['Workplace'])
        search_queries["Location"].append(row['Location'])
        search_queries['ID'].append(row['ID'])
```

```
print("Search queries size:", len(search_queries["Workplace"]))

for _, name in enumerate(search_queries["Name"]):
    query = f'site:linkedin.com/in/ "{name}" {"{search_queries["Workplace"][_]}"}'
    print(f"Searching for: {query}")
    ret = search_linkedin_profiles(query, api_keys[index_creds], cse_ids[index_creds],
                                    pp_name=f"datasets/profile_picture_data/profile_picture",
                                    output_json=f"datasets/profile_picture_data/url.json")
    if ret == 1:
        print(f"Error occurred while searching for {name} at this index {search_q
        index_creds += 1
    if index_creds >= len(api_keys):
        print("All API keys exhausted. Stopping the search.")
        return
    print("Search completed.")

main()

import os
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# === Provided Utility Functions ===

def get_dataset_path():
    path = input("Enter the path to the dataset: ").strip()
    if not os.path.exists(path):
        raise FileNotFoundError(f"File not found: {path}")
    return path

def has_content(x):
```

```
    return int(bool(str(x).strip()))

def preprocess_dataframe(df):
    string_columns = [
        'Full Name', 'Workplace', 'Location', 'Connections', 'Photo',
        'Followers', 'About', 'Experiences', 'Educations', 'Licenses',
        'Volunteering', 'Skills', 'Recommendations', 'Projects', 'Publications',
        'Courses', 'Honors', 'Scores', 'Languages', 'Organizations', 'Interests', 'Ac
    ]
    df[string_columns] = df[string_columns].fillna('')

    df['Has_Photo'] = df['Photo']
    df['Has_About'] = df['About'].apply(has_content)
    df['Has_Projects'] = df['Projects'].apply(has_content)
    df['Has_Education'] = df['Educations'].apply(has_content)
    df['Has_Experience'] = df['Experiences'].apply(has_content)
    df['Has_Skills'] = df['Skills'].apply(has_content)
    df['Has_Licenses'] = df['Licenses'].apply(has_content)
    df['Has_Interests'] = df['Interests'].apply(has_content)
    df['Has_Recommendations'] = df['Recommendations'].apply(has_content)

    df['About_Length'] = df['About'].apply(len)
    df['Skills_Length'] = df['Skills'].apply(len)
    df['Experience_Length'] = df['Experiences'].apply(len)
    df['Education_Length'] = df['Educations'].apply(len)
    df['Projects_Length'] = df['Projects'].apply(len)

    for col in ['Location', 'Workplace', 'Full Name']:
        df[col] = LabelEncoder().fit_transform(df[col].astype(str))

    count_fields = [
        'Number of Experiences', 'Number of Educations', 'Number of Licenses',
        'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
        'Number of Projects', 'Number of Publications', 'Number of Courses',
        'Number of Honors', 'Number of Scores', 'Number of Languages',
        'Number of Organizations', 'Number of Interests', 'Number of Activities',
        'Connections', 'Followers'
    ]
    df[count_fields] = df[count_fields].fillna(0)
```

```
    return df

def get_feature_list():
    return [
        'Has_Photo', 'Has_About', 'Has_Projects', 'Has_Education', 'Has_Experience',
        'Has_Skills', 'Has_Licenses', 'Has_Interests', 'Has_Recommendations',
        'About_Length', 'Skills_Length', 'Experience_Length', 'Education_Length', 'Pr
        'Location', 'Workplace', 'Full Name', 'Connections', 'Followers',
        'Number of Experiences', 'Number of Educations', 'Number of Licenses',
        'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
        'Number of Projects', 'Number of Publications', 'Number of Courses',
        'Number of Honors', 'Number of Scores', 'Number of Languages',
        'Number of Organizations', 'Number of Interests', 'Number of Activities'
    ]

def cunf_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.show()

# === Deep Learning Pipeline ===

def train_deep_learning_model():
    #path = get_dataset_path()
    path = "/home/fasma/heriot-watt/Research_Methods/FADL/datasets/training_Datasets"
    df = pd.read_csv(path)

    # Assumes 'Label' column exists: 0 = real, 1 = fake
    if 'Label' not in df.columns:
        raise ValueError("Dataset must contain a 'Label' column.")

    df = preprocess_dataframe(df)
    features = get_feature_list()

    X = df[features]
    y = df['Label']
```

```
# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, r

# Build the model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', me

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=2000, batch_si

# Evaluate
loss, accuracy = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {accuracy:.2f}")

# Predictions and Confusion Matrix
y_pred = (model.predict(X_test) > 0.5).astype(int)
cunf_matrix(y_test, y_pred)

return model, history

# === Run the training ===
if __name__ == "__main__":
    #check time execution

    start_time = time.time()
```

```
train_deep_learning_model()

end_time = time.time()
print(f"Execution time: {end_time - start_time} seconds")

import os
import datetime

import pandas as pd
import matplotlib.pyplot as plt
import joblib

import time

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.base import BaseEstimator, TransformerMixin

# ----- Helper utilities -----

def has_content(x):
    return int(bool(str(x).strip()))

def ensure_dir(path):
    os.makedirs(path, exist_ok=True)

class DataFrameWrapper(BaseEstimator, TransformerMixin):
    def __init__(self, columns):
        # Store as tuple so sklearn can clone safely
        self.columns = tuple(columns)

    def fit(self, X, y=None):
        return self
```

```
def transform(self, X):
    if isinstance(X, pd.DataFrame):
        return X
    return pd.DataFrame(X, columns=self.columns)

# ----- Preprocessing -----

def preprocess_dataframe(df):
    # Fill missing string columns with empty string
    string_columns = [
        'Full Name', 'Workplace', 'Location', 'Connections', 'Photo',
        'Followers', 'About', 'Experiences', 'Educations', 'Licenses',
        'Volunteering', 'Skills', 'Recommendations', 'Projects', 'Publications',
        'Courses', 'Honors', 'Scores', 'Languages', 'Organizations', 'Interests', 'Ac'
    ]
    for col in string_columns:
        if col in df.columns:
            df[col] = df[col].fillna('')

    # Binary / presence features
    if 'Photo' in df.columns:
        df['Has_Photo'] = df['Photo'].apply(has_content)
    else:
        df['Has_Photo'] = 0

    for src, dst in [
        ('About', 'Has_About'),
        ('Projects', 'Has_Projects'),
        ('Educations', 'Has_Education'),
        ('Experiences', 'Has_Experience'),
        ('Skills', 'Has_Skills'),
        ('Licenses', 'Has_Licenses'),
        ('Interests', 'Has_Interests'),
        ('Recommendations', 'Has_Recommendations'),
    ]:
        df[dst] = df[src].apply(has_content) if src in df.columns else 0

    # Length features (use 0 for missing)
```

```

df['About_Length'] = df.get('About', '').apply(lambda x: len(str(x)))
df['Skills_Length'] = df.get('Skills', '').apply(lambda x: len(str(x)))
df['Experience_Length'] = df.get('Experiences', '').apply(lambda x: len(str(x)))
df['Education_Length'] = df.get('Educations', '').apply(lambda x: len(str(x)))
df['Projects_Length'] = df.get('Projects', '').apply(lambda x: len(str(x)))

# Numeric count fields - fill missing with 0 and coerce
count_fields = [
    'Number of Experiences', 'Number of Educations', 'Number of Licenses',
    'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
    'Number of Projects', 'Number of Publications', 'Number of Courses',
    'Number of Honors', 'Number of Scores', 'Number of Languages',
    'Number of Organizations', 'Number of Interests', 'Number of Activities',
    'Connections', 'Followers'
]
for col in count_fields:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0)
    else:
        df[col] = 0

# If any of the categorical fields are missing, add empty string columns so encoder can handle them
for col in ['Location', 'Workplace', 'Full Name']:
    if col not in df.columns:
        df[col] = ''

return df

def get_feature_list():
    return [
        'Has_Photo', 'Has_About', 'Has_Projects', 'Has_Education', 'Has_Experience',
        'Has_Skills', 'Has_Licenses', 'Has_Interests', 'Has_Recommendations',
        'About_Length', 'Skills_Length', 'Experience_Length', 'Education_Length', 'Pr
        'Location', 'Workplace', 'Full Name', 'Connections', 'Followers',
        'Number of Experiences', 'Number of Educations', 'Number of Licenses',
        'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
        'Number of Projects', 'Number of Publications', 'Number of Courses',
        'Number of Honors', 'Number of Scores', 'Number of Languages',
    ]

```

```
'Number of Organizations', 'Number of Interests', 'Number of Activities'  
]  
  
# ----- Visualization -----  
  
def cunf_matrix(y_test, y_pred, name="Confusion Matrix"):  
    cm = confusion_matrix(y_test, y_pred)  
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)  
    disp.plot(cmap=plt.cm.Blues)  
    plt.title(f"Confusion Matrix - {name}")  
    plt.show()  
    plt.close()  
  
  
def plot_feature_importance(model, feature_names):  
    if hasattr(model, "feature_importances_"):  
        importances = model.feature_importances_  
    elif hasattr(model, "coef_"):  
        importances = getattr(model, 'coef_')  
        if importances.ndim > 1:  
            importances = importances[0]  
    else:  
        print("Model has no feature importances or coefficients.")  
        return  
  
    feat_importance = pd.Series(importances, index=feature_names).sort_values(ascending=False)  
    plt.figure(figsize=(10, 6))  
    feat_importance.plot(kind='bar')  
    # plt.title("Feature Importance")  
    # plt.tight_layout()  
    # plt.show()  
    # plt.close()  
  
  
# ----- Build pipelines for ensemble -----  
  
def build_ensemble_pipelines(features, categorical_cols, numeric_cols, random_state=42):  
    # Base models
```

```
rf = RandomForestClassifier(n_estimators=200, random_state=random_state)
gb = GradientBoostingClassifier(n_estimators=200, random_state=random_state)
lr = LogisticRegression(max_iter=10000, solver='saga', random_state=random_state)

# Preprocessor: encode categorical, optionally scale numeric
preprocessor = build_preprocessor(categorical_cols, numeric_cols, scale_numeric=True)

# Pipelines for individual models
rf_pipeline = Pipeline([
    ('df_wrapper', DataFrameWrapper(features)),
    ('preprocessor', preprocessor),
    ('model', rf)
])
gb_pipeline = Pipeline([
    ('df_wrapper', DataFrameWrapper(features)),
    ('preprocessor', preprocessor),
    ('model', gb)
])
lr_pipeline = Pipeline([
    ('df_wrapper', DataFrameWrapper(features)),
    ('preprocessor', preprocessor),
    ('model', lr)
])

# Voting Classifier
voting = VotingClassifier(
    estimators=[('rf', rf_pipeline), ('gb', gb_pipeline), ('lr', lr_pipeline)],
    voting='soft'
)

# Stacking Classifier
# Base estimators are numeric only after preprocessing, final estimator sees numeric
stacking = Pipeline([
    ('df_wrapper', DataFrameWrapper(features)),
    ('preprocessor', preprocessor),
    ('stack', StackingClassifier(
        estimators=[('rf', rf), ('gb', gb)],
        final_estimator=lr,
        passthrough=False # avoid sending raw features to final estimator
    ))
])
```

```
        ))
    ])

    return rf_pipeline, gb_pipeline, lr_pipeline, voting, stacking

def build_preprocessor(categorical_cols, numeric_cols, scale_numeric=False):
    cat_pipeline = Pipeline([
        ('ord', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
    ])

    if scale_numeric:
        num_pipeline = Pipeline([
            ('scale', StandardScaler())
        ])
    else:
        num_pipeline = Pipeline([
            ('pass', 'passthrough')
        ])

    preprocessor = ColumnTransformer([
        ('cat', cat_pipeline, categorical_cols),
        ('num', num_pipeline, numeric_cols)
    ])

    # Ensure DataFrame output so string column names are preserved
    preprocessor.set_output(transform="pandas")

    return preprocessor

def build_pipeline(model, categorical_cols, numeric_cols, all_features, scale_numeric):
    preprocessor = ColumnTransformer([
        ('cat', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1),
         ('num', StandardScaler() if scale_numeric else 'passthrough', numeric_cols))
    ])
    preprocessor.set_output(transform="pandas") # Keep column names
```

```
        return Pipeline([
            ('df_wrapper', DataFrameWrapper(all_features)),
            ('preprocessor', preprocessor),
            ('model', model)
        ])
    )

def ensemble_classifier(dataset_path, datasetname, save_model=False, features=None, random_state=42):
    df = pd.read_csv(dataset_path)
    df = preprocess_dataframe(df)

    if 'Label' not in df.columns:
        raise ValueError("The dataset must contain a 'Label' column with 0 or 1 values")

    if features is None:
        features = get_feature_list()

    missing = [f for f in features if f not in df.columns]
    if missing:
        raise ValueError(f"Missing feature columns in dataframe: {missing}")

    print(f"Dataset size: {df.shape[0]} rows, {df.shape[1]} columns")

    X = df[features]
    y = df['Label']

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=random_state
    )

    categorical_cols = [c for c in ['Location', 'Workplace', 'Full Name'] if c in X.columns]
    numeric_cols = [c for c in X.columns if c not in categorical_cols]

    rf = RandomForestClassifier(n_estimators=200, random_state=random_state)
    gb = GradientBoostingClassifier(n_estimators=200, random_state=random_state)
    lr = LogisticRegression(max_iter=10000, solver='saga', random_state=random_state)
```

```
rf_pipeline, gb_pipeline, lr_pipeline, voting, stacking = build_ensemble_pipeline(
    features=features,
    categorical_cols=categorical_cols,
    numeric_cols=numeric_cols,
    random_state=random_state
)

models = {
    'RandomForest': rf_pipeline,
    'GradientBoosting': gb_pipeline,
    'LogisticRegression': lr_pipeline,
    'VotingClassifier': voting,
    'StackingClassifier': stacking
}

timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = os.path.join('results', 'model_stats')
ensure_dir(results_dir)
results_file = os.path.join(results_dir, f"{timestamp}_{datasetname}_ensemble_res"

with open(results_file, 'w') as f:
    for name, model in models.items():
        start_time = time.time()
        print(f"\nTraining {name}...")
        f.write(f"\n{name} Results:\n")

    try:
        cv_scores = cross_val_score(model, X_train, y_train, cv=3, scoring='roc_auc')
        cv_mean = cv_scores.mean()
        print(f"{name} CV ROC-AUC (3-fold): {cv_mean:.4f}")
        f.write(f"{name} CV ROC-AUC (3-fold): {cv_mean:.4f}\n")
    except Exception as e:
        print(f"Could not run cross_val_score for {name}: {e}")
        f.write(f"Could not run cross_val_score for {name}: {e}\n")

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
report = classification_report(y_test, y_pred)
print(f"\n{name} Results:\n")
print(report)
f.write(report + "\n")

try:
    # voting and stacking may not implement predict_proba in the same way
    if hasattr(model, 'predict_proba'):
        # Get probabilities for the positive class (label=1 = genuine)
        y_proba = model.predict_proba(X_test)
        if y_proba.ndim == 2:
            y_proba = y_proba[:, 1]

        # Convert to % genuine
        percent_genuine = (y_proba * 100).round(2)

        # Save results to CSV
        results_df = pd.DataFrame({
            "True_Label": y_test,
            "Predicted_Label": y_pred,
            "Probability_Genuine": y_proba,
            "Percent_Genuine": percent_genuine
        })
        results_csv = os.path.join(results_dir, f"{timestamp}_{datasetname}.csv")
        results_df.to_csv(results_csv, index=False)
        print(f"Saved prediction probabilities to {results_csv}")
        f.write(f"Saved prediction probabilities to {results_csv}\n")

        # Compute ROC-AUC using probabilities
        auc = roc_auc_score(y_test, y_proba)
        print(f"ROC-AUC: {auc:.4f}")
        f.write(f"ROC-AUC: {auc:.4f}\n")

except Exception as e:
    print(f"Could not compute ROC-AUC for {name}: {e}")
    f.write(f"Could not compute ROC-AUC for {name}: {e}\n")

cunf_matrix(y_test, y_pred, name=name)
```

```
try:
    # extract final estimator safely
    if isinstance(model, Pipeline):
        final_est = model.named_steps.get('model', model.steps[-1][1])
    else:
        final_est = model

    if hasattr(final_est, 'feature_importances_'):
        plot_feature_importance(final_est, features)
except Exception as e:
    print(f"Could not plot feature importances for {name}: {e}")

if save_model:
    model_filename = os.path.join(results_dir, f"{timestamp}_{datasetname}.pkl")
    try:
        joblib.dump(model, model_filename)
        print(f"{name} model saved to {model_filename}")
        f.write(f"{name} model saved to {model_filename}\n")
    except Exception as e:
        print(f"Error saving model {name}: {e}")
        f.write(f"Error saving model {name}: {e}\n")
    end_time = time.time()
    print(f"Execution time {name}: {end_time - start_time} seconds")
print(f"\nAll results written to {results_file}")

# ----- CLI / main -----
def ensemble_main(dataset_paths=None):
    if dataset_paths is None:
        dataset_paths = [input('Enter the path to the dataset: ').strip()]

    for path in dataset_paths:
        if not os.path.exists(path):
            print(f"File not found: {path}")
            continue
        datasetname = os.path.splitext(os.path.basename(path))[0]
        ensemble_classifier(path, datasetname, save_model=True, features=get_feature_
```

```
if __name__ == '__main__':  
  
    datasets = [  
        #"/home/fasma/heriot-watt/Research_Methods/FADL/datasets/training_Datasets/Li  
        #"/home/fasma/heriot-watt/Research_Methods/FADL/datasets/training_Datasets/Li  
    ]  
    ensemble_main(datasets)  
  
    import pandas as pd  
    import matplotlib.pyplot as plt  
    from sklearn.ensemble import RandomForestClassifier  
    from sklearn.model_selection import train_test_split  
    from sklearn.preprocessing import LabelEncoder  
    from sklearn.metrics import classification_report  
    from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay  
  
    import joblib  
  
    import os  
    import datetime  
  
    import time  
  
  
def get_dataset_path():  
    path = input("Enter the path to the dataset: ").strip()  
    if not os.path.exists(path):  
        raise FileNotFoundError(f"File not found: {path}")  
    return path  
  
def has_content(x):  
    return int(bool(str(x).strip()))  
  
def preprocess_dataframe(df):  
    string_columns = [  
        'Full Name', 'Workplace', 'Location', 'Connections', 'Photo',  
        'Followers', 'About', 'Experiences', 'Educations', 'Licenses',
```

```
'Volunteering', 'Skills', 'Recommendations', 'Projects', 'Publications',
'Courses', 'Honors', 'Scores', 'Languages', 'Organizations', 'Interests', 'Activities'
]

df[string_columns] = df[string_columns].fillna('')

df['Has_Photo'] = df['Photo']
df['Has_About'] = df['About'].apply(has_content)
df['Has_Projects'] = df['Projects'].apply(has_content)
df['Has_Education'] = df['Educations'].apply(has_content)
df['Has_Experience'] = df['Experiences'].apply(has_content)
df['Has_Skills'] = df['Skills'].apply(has_content)
df['Has_Licenses'] = df['Licenses'].apply(has_content)
df['Has_Interests'] = df['Interests'].apply(has_content)
df['Has_Recommendations'] = df['Recommendations'].apply(has_content)

df['About_Length'] = df['About'].apply(len)
df['Skills_Length'] = df['Skills'].apply(len)
df['Experience_Length'] = df['Experiences'].apply(len)
df['Education_Length'] = df['Educations'].apply(len)
df['Projects_Length'] = df['Projects'].apply(len)

for col in ['Location', 'Workplace', 'Full Name']:
    df[col] = LabelEncoder().fit_transform(df[col].astype(str))

count_fields = [
    'Number of Experiences', 'Number of Educations', 'Number of Licenses',
    'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
    'Number of Projects', 'Number of Publications', 'Number of Courses',
    'Number of Honors', 'Number of Scores', 'Number of Languages',
    'Number of Organizations', 'Number of Interests', 'Number of Activities',
    'Connections', 'Followers'
]
df[count_fields] = df[count_fields].fillna(0)

return df

def get_feature_list():
```

```
return [
    'Has_Photo', 'Has_About', 'Has_Projects', 'Has_Education', 'Has_Experience',
    'Has_Skills', 'Has_Licenses', 'Has_Interests', 'Has_Recommendations',
    'About_Length', 'Skills_Length', 'Experience_Length', 'Education_Length', 'Pr
    'Location', 'Workplace', 'Full Name', 'Connections', 'Followers',
    'Number of Experiences', 'Number of Educations', 'Number of Licenses',
    'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
    'Number of Projects', 'Number of Publications', 'Number of Courses',
    'Number of Honors', 'Number of Scores', 'Number of Languages',
    'Number of Organizations', 'Number of Interests', 'Number of Activities'
]

def get_important_feature_list():
    return [
        'Has_Photo', 'Has_About', 'About_Length', 'Skills_Length',
        'Experience_Length', 'Education_Length', 'Projects_Length',
        'Location', 'Workplace', 'Full Name', 'Connections', 'Followers',
        'Number of Experiences', 'Number of Educations', 'Number of Licenses',
        'Number of Volunteering', 'Number of Skills', 'Number of Recommendations',
        'Number of Projects', 'Number of Publications', 'Number of Courses',
        'Number of Honors', 'Number of Scores', 'Number of Languages',
        'Number of Organizations', 'Number of Interests', 'Number of Activities'
    ]

def cunf_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.show()

def plot_feature_importance(rf, features):
    importances = rf.feature_importances_
    feat_importance = pd.Series(importances, index=features).sort_values(ascending=False)

    plt.figure(figsize=(10, 6))
    feat_importance.plot(kind='bar')
    plt.title("Feature Importance")
```

```
plt.tight_layout()
plt.show()

def random_forest_classifier(dataset_path, datasetname, save_model=False, features=None):
    df = pd.read_csv(dataset_path)
    df = preprocess_dataframe(df)

    if 'Label' not in df.columns:
        raise ValueError("The dataset must contain a 'Label' column with 0 or 1 values")

    #print the dataset size

    print(f"Dataset size: {df.shape[0]} rows, {df.shape[1]} columns")
    X = df[features]
    y = df['Label']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    rf = RandomForestClassifier(n_estimators=10000, random_state=42)
    rf.fit(X_train, y_train)

    y_pred = rf.predict(X_test)
    report = classification_report(y_test, y_pred)
    print(report)

    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    results_file_name = f"results/model_stats/{timestamp}_{datasetname}_results.txt"

    # plot_feature_importance(rf, features)
    # cunf_matrix(y_test, y_pred)

    with open(results_file_name, "w") as f:
        f.write(report)

    if save_model:
        save = input("Do you want to save the model? (y/n): ").strip().lower()
        if save == 'y':
            joblib.dump(rf, f"{results_file_name}.pkl")
```

```
    print("Model saved.")

def compare_AI_noIA_AInoAI():
    datasets_path = ["/home/fasma/heriot-watt/Research_Methods/FADL/datasets/LinkedIn"]
                  "/home/fasma/heriot-watt/Research_Methods/FADL/datasets/LinkedIn"]
                  "/home/fasma/heriot-watt/Research_Methods/FADL/datasets/LinkedIn"]
    ]

    for path in datasets_path:
        random_forest_classifier(path, os.path.splitext(os.path.basename(path))[0], save_)

def compare_feature_importance():
    path = "/home/fasma/heriot-watt/Research_Methods/FADL/datasets/LinkedIn people pr"
    random_forest_classifier(path, os.path.splitext(os.path.basename(path))[0], save_)

def main():
    datasets_path = [#" /home/fasma/heriot-watt/Research_Methods/FADL/datasets/training"]
                    #"/home/fasma/heriot-watt/Research_Methods/FADL/datasets/training"]
                    "/home/fasma/heriot-watt/Research_Methods/FADL/datasets/training"]
    ]

    for path in datasets_path:
        random_forest_classifier(path, os.path.splitext(os.path.basename(path))[0], save_)

if __name__ == "__main__":
    start_time = time.time()

    main()

    end_time = time.time()
    print(f"Execution time: {end_time - start_time} seconds")
```

Bibliography

- Adikari, S. and Dutta, K. (2020). Identifying fake profiles in linkedin. *arXiv preprint arXiv:2006.01381*.
- Awan, A. A., Subramoni, H., and Panda, D. K. (2017). An in-depth performance characterization of cpu- and gpu-based dnn training on modern architectures. In *Proceedings of the Machine Learning on HPC Environments*, MLHPC'17, New York, NY, USA. Association for Computing Machinery.
- Ayoobi, N., Shahriar, S., and Mukherjee, A. (2023). The looming threat of fake and llm-generated linkedin profiles: Challenges and opportunities for detection and prevention. In *Proceedings of the 34th ACM Conference on Hypertext and Social Media*, pages 1–10.
- B., J. (2018). Automated fake account detection at linkedin. <https://www.linkedin.com/blog/engineering/trust-and-safety/automated-fake-account-detection-at-linkedin>.
- Bharne, S. and Bhaladhare, P. (2022). Comprehensive analysis of online social network frauds. In *International Conference on Advances in Data-driven Computing and Intelligent Systems*, pages 23–40. Springer Nature Singapore.
- Chakraborty, P., Shazan, M., Nahid, M., Ahmed, M., and Talukder, P. (2022). Fake profile detection using machine learning techniques. *Journal of Computer and Communications*, 10(10):74–87.
- Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A., and Tesconi, M. (2015). Fame for sale: Efficient detection of fake twitter followers. *Decision Support Systems*, 80:56–71.
- Isaiah, J., Caitlin, M., and Amie, N. (2022). Rise of ai-generated, fake linkedin profiles raises social engineering challenges. <https://www.kroll.com/en/insights/publications/cyber/rise-of-ai-generated-fake-linkedin-profiles-social-engineering-challenges>.

Kadhim, A. and Abdullah, A. (2022). Fake accounts detection on social media using stack ensemble system. *International Journal of Electrical and Computer Engineering*, 12:3013–3022.

LinkedIn (2022). Personal post on linkedin. https://www.linkedin.com/posts/guilhem-vinet-661572190_rjobboardsearch-on-reddit-talentkompass-activity-7112446339409408000-QxSR?utm_source=share&utm_medium=member_desktop&rcm=ACoAACzvBCkBIIsax-daA-qt5NxYAGpa-0t0itty.

LinkedIn (n.d.). Recognize and report scams. <https://www.linkedin.com/help/linkedin/answer/a1336387>.

Reddy, S. (2019). Fake profile identification using machine learning. *International Research Journal of Engineering and Technology (IRJET)*, 6(12):1145–1150.

Roy, P. and Chahar, S. (2021). Fake profile detection on social networking websites: a comprehensive review. *IEEE Transactions on Artificial Intelligence*, 1(3):271–285.

Vidros, S., Koliaris, C., and Kambourakis, G. (2016). Online recruitment services: another playground for fraudsters. *Computer Fraud & Security*, pages 8–13.

Wang, G., Mohanlal, M., Wilson, C., Wang, X., Metzger, M., Zheng, H., and Zhao, B. Y. (2012). Social turing tests: Crowdsourcing sybil detection. *arXiv*.