

Detecting Errors in Financial Data: A Multi-Agent LLM and Synthetic Data Approach

by

Katherine Liu

B.S. Computer Science and Engineering
MIT, 2025

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

© 2025 Katherine Liu. This work is licensed under a [CC BY-NC-ND 4.0](#) license.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license
to exercise any and all rights under copyright, including to reproduce, preserve, distribute
and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Katherine Liu
Department of Electrical Engineering and Computer Science
May 9, 2025

Certified by: Amar Gupta
Research Scientist, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair
Master of Engineering Thesis Committee

Detecting Errors in Financial Data: A Multi-Agent LLM and Synthetic Data Approach

by

Katherine Liu

Submitted to the Department of Electrical Engineering and Computer Science
on May 9, 2025 in partial fulfillment of the requirements for the degree of

**MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE**

ABSTRACT

With the high volume of activity flowing through financial institutions, detecting potential errors remains a critical challenge. This paper addresses two key areas where errors may occur: business name registrations and transactions within valid accounts. Traditional string-matching methods struggle to accurately identify incorrectly written business names that closely resemble existing ones, while existing error detection models for transaction data often suffer from class imbalance, leading to reduced performance on minority incorrect transaction cases. To address these issues, this paper proposes two novel approaches. First, a hybrid method integrating multi-agent Large Language Models (LLMs) with existing string-matching techniques enhances the detection of incorrect business names by capturing subtle variations beyond conventional edit-distance metrics, improving the recall from 0.815 for the baseline model to 0.987 using the proposed method. Second, an improved tabular data generation method for credit card transactions is introduced, leveraging LLMs and class balancing to generate high-quality synthetic data. Using this data to train error detection systems results in a decrease of the false negative rate from 23.47% to 12.84%. Together, these methods enhance the performance of error detection systems, enabling financial institutions to enhance the experiences of their clients.

Thesis supervisor: Amar Gupta

Title: Research Scientist

Acknowledgments

I thank Connie Cao, Manaal Mohammed, Ryan Chin, and Aleksandar Jovanovic-Hacon for their assistance in conducting related works research and running experiments for the business name error detection task. Manaal Mohammed contributed to the development and evaluation of the CNN-based glyph similarity model, while Ryan Chin assisted with the webscraper implementation. Connie Cao and Aleksandar Jovanovic-Hacon provided collaborative support across multiple components, including contributing to the experimental setup and system design.

Contents

<i>List of Figures</i>	9
<i>List of Tables</i>	11
1 Introduction	13
2 Related Work	15
2.1 String Matching	15
2.2 Tabular Data Generation	17
2.3 Addressing Class Imbalance	17
3 Methods	19
3.1 Hybrid System for Business Name Error Detection	19
3.1.1 Current Approaches	19
3.1.2 Proposed System	20
3.1.3 Individual Methods	20
3.1.4 Aggregate Scoring	22
3.2 TabuLa for Transaction Error Detection	24
3.2.1 Proposed System	24
3.2.2 Class-Balanced Generation	24
3.2.3 Comparison with Traditional Resampling	24
3.2.4 Preprocessing	26
3.2.5 Model Training	26
4 Experiments	27
4.1 Datasets	27
4.1.1 Business Name Error Detection	27
4.1.2 Transaction Error Detection	28
4.2 Evaluation	28
4.2.1 Business Name Error Detection	28
4.2.2 Transaction Error Detection	29
5 Business Name Error Detection Results	31
5.1 Baseline: Fuzzy Matching	31
5.2 Individual Methods	33
5.3 Aggregate Results	34
5.3.1 LLM Prompting	34

5.3.2	Logistic Regression	36
5.3.3	Random Forest Classifier	36
6	Transaction Error Detection Results	39
6.1	Baseline	39
6.2	Normalization	40
6.3	Class Balancing	42
7	Conclusion	49
7.1	Future Work	51
7.1.1	Business Name Error Detection	51
7.1.2	Transaction Error Detection	51
	<i>References</i>	53

List of Figures

3.1	Business Name Error Detection System Pipeline	21
3.2	Transaction Error Detection System Pipeline	25
5.1	Fuzzy Matching Performance of 2 classes vs. 3 classes	32
5.2	Aggregate performance from LLM prompting	35
5.3	Aggregate performance from logistic regression	36
5.4	Aggregate performance from Random Forest classifier	37
6.1	Baseline ROC curve	40
6.2	Baseline confusion matrix	41
6.3	ROC curves for models trained without normalizing data vs. normalizing data	44
6.4	ROC curve for random undersampling	45
6.5	Confusion matrix for random undersampling	45
6.6	ROC curves for no class balancing vs. class balancing	46
6.7	Confusion matrices for no class balancing vs. class balancing	47
7.1	Transaction error detection performance comparison of baseline model vs. proposed model	50

List of Tables

5.1	Performance comparison of 2 classes vs. 3 classes	31
5.2	Performance comparison of individual methods	33
5.3	Performance comparison of aggregate scoring methods	34
5.4	Aggregate scores from LLM prompting	36
6.1	AUC values for each experiment without class balancing	40
6.2	AUC values for each experiment with class balancing	42
7.1	Business name error detection performance comparison of baseline model vs. proposed model	49

Chapter 1

Introduction

Error detection is of significant concern for financial institutions, as financial systems and activities continue to evolve in complexity, leading to innovative approaches that emphasize the need for robust, adaptable, and precise detection systems [1]. Two areas where it is important to detect such errors are account registration using incorrect business names and incorrect transactions under valid accounts.

Wrong business name registrations are of growing concern; one issue could result from such names being used to deceive customers by impersonating legitimate businesses through subtle differences in naming, including character substitutions, added words, or punctuation changes. This error could also result from benign misspellings and auto-correct substitutions. The methods discussed are also valuable in identifying mistakes in other contexts; they can be applied to detecting errors in URLs, product identifiers, or user-entered information during onboarding processes. This is important, since small typos in URLs could lead to broken links or misrouted traffic, and inconsistencies in product codes could disrupt inventory systems. Similarly, name mismatches across databases, which could result from formatting differences, abbreviations, or human error, can result in incorrect records. In each of these cases, the challenge lies in catching likely input errors. Detecting this kind of error requires distinguishing between possibly valid variations and mistakenly inputted or intentionally malicious naming, which is a challenging task due to some incorrect names having minor mistakes or potentially being deliberately designed to replicate legitimate names. It is necessary to determine what methods are capable of distinguishing which names are incorrect when they are intentionally similar to valid names. Current implementations use traditional methods like fuzzy matching and the edit distance, which struggle with scalability and lack the semantic or contextual nuances necessary for robust error detection. One study highlights the limitations of fuzzy matching techniques, noting that even when two strings appear similar to humans, fuzzy matching often struggles because it fails to adapt to the informativeness of character combinations [2]. They also involve using a database of known company names. This is both difficult to maintain, due to the large amount of companies and how often new companies are established, and not scalable, as new names must be compared to all existing company names in the database. It is necessary to design a method that can detect names that are similar but not exactly the same, which are likely to be incorrect, while also allowing for non-matches that could be smaller companies not in the database.

Additionally, even for valid accounts, it is essential to determine when a transaction could

be potentially erroneous. Error detection systems in banking frequently rely on tabular datasets to keep track of transaction information, like the transaction time and the amount. Machine learning models are often used to identify incorrect transactions, customer behaviors, or fraudulent entities. For instance, Raghavan et al. provides insights and benchmarks how these models are employed in identifying unusual activities [3]. Tabular data is essential in a variety of fields, including healthcare, finance, and marketing, where machine learning models are used for tasks ranging from protecting privacy in clinical decision support [4] to fraud detection [5]. However, in many of these applications, these datasets often suffer from class imbalance, where one class significantly outnumbers others, which can result in biased model training, leading to worse performance on minority classes and reducing the utility when applying the model to real-world scenarios [6]. Synthetic data generation has emerged as a promising approach to augmenting imbalanced datasets, allowing models to learn from more representative data distributions. Recent advances in Large Language Models (LLMs) have shown potential for producing high-quality synthetic tabular data. One challenge with using LLMs for tabular data synthesis is the complexity of encoding non-text data. Unlike data presented in textual form, tabular data relies on structured correlations between features, often encoded in numerical or categorical formats. EPIC [7] proposes a combination of CSV formatting, class balancing, and unique variable mappings to better capture correlations and underrepresented attributes. These prompt design strategies have shown improvements in generating data that aligns more closely with original tabular distributions, particularly in minority classes.

This dissertation proposes two methods that financial institutions can utilize in error detection, while addressing issues in currently implemented systems. First, a hybrid system is introduced, which combines multi-agent LLMs with existing string matching techniques to improve incorrect business name detection. Next, this dissertation proposes a tabular data generation method for credit card transaction data that builds on previous approaches while incorporating a class balancing component.

Together, these methods provide a more comprehensive error detection framework by improving the accuracy of identifying both business name errors and transaction errors. By leveraging advancements in NLP and synthetic data generation, this research aims to improve the ability of financial institutions to detect and prevent mistakes in increasingly complex data environments.

Chapter 2

Related Work

2.1 String Matching

String similarity metrics, such as the Levenshtein distance, are frequently used in error detection to assess how closely two strings resemble each other. Significant research has been conducted to enhance both the accuracy and efficiency of Levenshtein distance calculations for large-scale string comparisons [8]. Roy and Yedurupaka further explore the Levenshtein distance in fuzzy matching algorithms within AI, particularly their applications in Anti-Money Laundering (AML) processes [9]. They discuss techniques to enhance the efficiency of fuzzy matching, including optimization of algorithmic complexity and machine learning integration. Various alternative approaches have been proposed to improve string-matching algorithms, including hashing-based encoding of substrings and utilizing one-bit arrays for longest common subsequence identification [10]. However, despite these improvements, such methods are increasingly ineffective at addressing the growing diversity of error detection challenges. Without the ability to capture semantic relationships, these methods fail to distinguish between malicious name modifications and legitimate variations, like location tags, for instance.

To overcome these limitations, machine learning techniques have been explored as an alternative approach to approximate string matching. Large and diverse datasets play a critical role in advancing machine learning models for this task, leading to the development of extensive datasets consisting of both networked strings and images of strings [11, 12]. Several machine learning models have been evaluated in this context, including active learning classifiers, which outperform traditional methods [13]. Additionally, Long Short-Term Memory (LSTM) networks have been studied for their effectiveness in detecting similar strings, demonstrating superior performance over Random Forest classifiers [14]. Another promising approach involves the use of Siamese Convolutional Neural Networks (CNNs) to convert strings into images before analyzing them to detect homoglyphs, which are visually similar characters that can be exploited in phishing attacks [15]. Some studies suggest that attention-based CNNs can outperform Siamese CNNs in this task [16].

Hybrid models have also been proposed to leverage the strengths of multiple architectures. One such model integrates CNNs to capture local contextual information while incorporating LSTMs to analyze the broader context of a sentence, improving string similarity calculations

with a semantic perspective [17]. Another approach combines bidirectional LSTMs with multi-window CNNs to optimize performance in text-matching tasks by refining sequence representations and capturing interactions between compared text sequences [18]. These hybrid methods enhance the ability to detect complex string relationships, further improving error detection accuracy.

Recent advancements in error detection techniques for URLs provide further insights into string-matching methodologies. One promising approach integrates hash functions with machine learning models to identify homoglyph attacks, exploiting subtle textual variations designed to mislead users [19]. This method utilizes hash-based string representations, enabling more efficient machine learning detection of visual similarities that traditional techniques might miss. It has demonstrated strong performance, achieving an accuracy of 99.8%, particularly in cases where semantic or purely visual methods are insufficient. Additionally, the rise of deep learning and natural language processing has facilitated the use of transformer-based models in phishing detection. URLTran, for instance, is a transformer-based model designed specifically for phishing URL classification [20]. It fine-tunes BERT and RoBERTa models while incorporating a domain-specific transformer trained on custom vocabularies. The model processes URLs by tokenizing them, generating embedding vectors, and classifying them as phishing or non-phishing. It also incorporates adversarial training strategies to enhance robustness against evolving phishing techniques.

Based on developments in multi-agent LLM frameworks, LLM-based techniques could be useful when applied to error detection tasks. For instance, the MAKGED framework uses multiple LLM agents to collaboratively identify errors in knowledge graphs by analyzing subgraph information from diverse perspectives and engaging in multi-round discussions, leading to improved accuracy and transparency in error detection processes [21]. Similarly, the CoMM framework introduces a collaborative multi-agent, multi-reasoning-path prompting approach, where LLMs assume different expert roles to solve complex problems, showcasing the potential of role-based agent collaboration in improving reasoning capabilities [22]. Zhou et al. study the resilience of multi-agent collaboration in LLM systems, providing insights into how multiple language models can be orchestrated to evaluate error likelihood based on various similarity metrics [23]. These frameworks suggest the potential of multi-agent LLM prompting in tasks requiring nuanced understanding and error identification, such as string matching and business name error detection.

Integrating web scraping with LLMs has also emerged as a powerful approach for real-time data extraction and string validation. AutoScraper presents a two-stage framework that combines the adaptability of LLMs with traditional web scraping techniques, enabling efficient handling of diverse and constantly changing web environments [24]. Additionally, leveraging Retrieval-Augmented Generation (RAG) models with web scraping allows for accurate data extraction by combining the knowledge representation power of pre-trained LLMs with targeted information access [25]. These methods enhance the capability of systems to validate business legitimacy by accessing and analyzing real-time web data, reducing reliance on static databases.

2.2 Tabular Data Generation

LLMs have been shown to be useful in tasks involving tabular data. Fang et al. [26] describe characteristics of tabular data, reviews traditional, deep-learning, and LLM methods, introduces techniques for adapting tabular data for LLMs, and describes applications of LLMs in prediction tasks, data augmentation, and question answering tasks. Such techniques include serialization (text-based, embedding-based, graph/tree-based), table manipulation (compacting tables, additional information), prompt engineering (prompt format, in-context learning, chain-of-thought, RAG), and building end-to-end systems. To accurately generate synthetic tabular data, Fang et al. describe several methodologies that use LLMs such as GPT2 and DistilGPT2 for the task of tabular data synthesis.

GReAT (Generation of Realistic Tabular data) [27] is a method for synthesizing tabular data using transformer-based LLMs, specifically GPT-2 or DistilGPT-2. By leveraging textual encoding and feature order permutation during training, GReAT generates realistic tabular data while also allowing for conditional sampling while avoiding dimensionality issues. While achieving SOTA performance across diverse datasets, GReAT's primary limitation is its extensive training time.

TabuLa [28] extends GReAT and offers a faster training process by preprocessing tabular data to shorter token sequences while maintaining high quality synthetic data. This approach utilizes DistilGPT-2 and GPT-2, depending on computation resources, with randomly initialized weights to allow for faster convergence. This is due to the distinct format of tabular data. In order to use these natural language models, tabular data rows are converted into sentences, but the sentences are repetitive and do not match the typical language patterns in the training data of the models. Tabular data is converted into sentences using the "X Y" format (rather than "X is Y") since the former format is simpler and shorter, allowing models trained on this data to better recognize patterns in new tasks. Additionally, token sequence compression is used to improve training efficiency. This process involves abbreviating column names and categorical values into single tokens and applying the format simplification described earlier ("X Y" instead of "X is Y"). With these features, TabuLa reduces the training time per epoch by 46.2% and improves the synthetic data utility compared to current alternative LLM-based SOTA algorithms [28].

2.3 Addressing Class Imbalance

Several techniques have been developed for handling imbalanced datasets. One approach to addressing imbalanced datasets is to use Support Vector Machines (SVMs) [6], due to their versatility and robustness in handling imbalanced classes. However, traditional SVMs struggle with the asymmetrical class distribution, as the decision boundary may be biased toward the majority class. A novel taxonomy is proposed that categorizes techniques into data pre-processing, algorithmic structures, and hybrid methods, with advantages and limitations to each technique. Effective techniques include adaptive methods that integrate domain-specific knowledge and leverage local relationships to improve decision boundaries, like K-nearest neighbor-based weighted SVMs. Another important approach is to use ensemble-based techniques, which combine the strengths of multiple models to mitigate class imbalance.

Examples include Random Forest and Gradient Boosting. This paper suggests that generating synthetic data is a promising approach, although finding the perfect method is a challenge, and speed-up techniques can be useful for tackling computational complexity [29].

Recent work has explored more nuanced synthetic data generation strategies that address these limitations. For example, D’souza et al. propose a novel approach that introduces an “overlap class” to model the ambiguous boundary between minority and majority class regions during data generation [30]. This class is used during training of generative models to improve the quality of samples in these critical regions and then discarded when training the final classifier. The results demonstrate significant gains in classifier performance, especially for the minority class. This highlights how careful design of synthetic data can address issues of traditional data-level techniques.

Other studies have further investigated both data-level and algorithmic-level methods [31]. Data-level techniques have inferior performance, because they may lose important information through undersampling or overfitting with oversampling. Hybrid methods and algorithmic methods address class imbalance more effectively, achieving higher true positive rates (TPR) while maintaining balanced true negative rates (TNR). The algorithmic methods employed include binary cross entropy and focal loss.

Together, these different fields of research offer a rich foundation for designing more intelligent and context-aware error detection systems. However, challenges remain in integrating these techniques into scalable, real-world systems that are robust and adaptive to evolving financial systems. The methods proposed in this dissertation build upon these existing works, aiming to bridge gaps between traditional matching techniques and modern learning-based approaches while also introducing enhancements to handle the complexities of financial data.

Chapter 3

Methods

3.1 Hybrid System for Business Name Error Detection

Accurate detection of errors in business names is crucial for ensuring the reliability of financial systems. Since business interactions rely on user-submitted data for registration and verification processes, systems must be able to distinguish between valid variations and potentially incorrect entries. This task is made more difficult by the high volume of business entities and the subtlety of many errors, which can include character swaps, omissions, or stylistic changes. A variety of approaches have been developed to tackle this problem, but they often face limitations in scalability, accuracy, and adaptability. Below, I explore commonly used approaches, recent advancements in several other methods, and how they can be combined to improve system performance.

3.1.1 Current Approaches

Frequently used approaches involve running an iterative search within a database of known valid company names, and if the name is not found, it uses fuzzy matching to determine if the name is an accidental variation of a known company.

Iterative Search

Many systems process business names by searching a database of known legitimate companies. If an exact match is found, the name is classified as "Valid". Otherwise, a similarity assessment determines whether the name is an incorrect variation. This approach faces scalability challenges, as it relies on a linear search that becomes computationally expensive as the dataset grows. This inefficiency makes real-time error detection particularly difficult in fast-paced industries like finance and e-commerce. Additionally, maintaining the database is challenging as new businesses are established. To address scalability issues, several methods have been explored; for instance, Gschwind et al. present a scalable system that reduces linkage time by efficiently decomposing the search space using MinHash, achieving high linkage accuracy while scaling linearly with the number of nodes used [32].

Fuzzy Matching

If no exact match exists, the system applies fuzzy matching techniques, primarily the Levenshtein distance, to measure textual similarity [33]. While this method detects minor modifications, it struggles to differentiate between invalid variations, like mistakes or intentional fraud, and valid variations, such as branding differences or name extensions. Additionally, as the dataset expands, computing edit distances for every comparison becomes increasingly costly, leading to higher false positives and slower processing times.

3.1.2 Proposed System

The proposed system follows the pipeline illustrated in Figure 3.1. The input, which is a single business name, is first checked in the database for an exact match. If a match exists, the name is classified as "Valid"; otherwise, it is scored using each individual method. The individual scores are then aggregated to determine whether the name should be classified as "Valid" or "Wrong".

3.1.3 Individual Methods

To improve the accuracy, scalability, and contextual understanding of business name error detection systems, alternative methods that extend beyond traditional string matching are explored. While exact and fuzzy matching methods offer baseline functionality, they often fail to generalize across real-world variations, such as branding extensions, typographical errors, or fraudulent attempts. As datasets grow in size and complexity, these approaches face computational challenges. To combat this, I propose a system that integrates semantic embeddings, visual glyph analysis, large language model reasoning, and real-time web validation. These methods aim to capture deeper patterns in how incorrect names deviate from valid ones and to support scalable, real-time deployment in high-risk domains like finance and e-commerce.

Enhanced String Similarity Assessment

Cosine Similarity of Vector Embeddings. Instead of character-level similarity metrics such as edit distance, this approach represents each business name as a high-dimensional dense vector using embedding models like Word2Vec, FastText, or CompanyName2Vec, which focuses on learning company name semantics [34]. Given two business names a and b , the vector representations \vec{v}_a and \vec{v}_b are compared using cosine similarity: $\text{sim}(a, b) = \frac{\vec{v}_a \cdot \vec{v}_b}{\|\vec{v}_a\| \cdot \|\vec{v}_b\|}$. This method allows for semantic comparisons and captures similarities beyond surface-level edits. For instance, "Amazon Inc." and "Amazon Incorporated" can have high similarity despite a large edit distance. The effectiveness of this method can be improved by using domain-specific training.

BERT-Based Name Matching. This method uses contextual embeddings produced by transformer-based models such as BERT. Each business name is passed through the model as a sentence through a pre-trained model; specifically, I use "bert-base-uncased". One limitation is that this model might not detect errors caused by switching between upper and

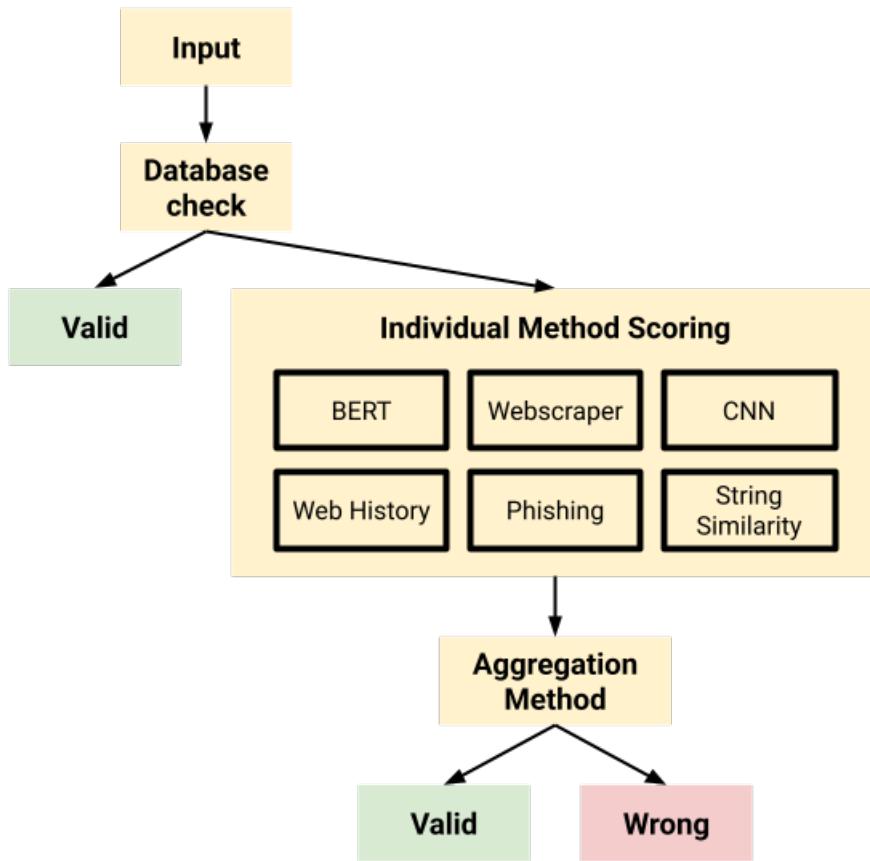


Figure 3.1: Business Name Error Detection System Pipeline. A business name is inputted into the system, where it is checked against the database. It is either classified as "Valid" or proceeds down the system, where it is scored individually using each method. The scores are aggregated to determine the final classification of "Valid" or "Wrong".

lower case, since the model is uncased. Given two names, cosine similarity, as described above, between their BERT embeddings is used to assess similarity [34]. BERT’s tokenization allows the model to handle out-of-vocabulary tokens and better understand uncommon names [35]. However, this method is computationally expensive when applied to large datasets [36].

Siamese CNN for Glyph-Based Matching. This approach analyzes the visual structure of business names by converting them to glyph-based images rendered using standard fonts. The model processes name pairs through two identical CNN branches with shared weights, with each branch passing one glyph image through convolutional and max-pooling layers for feature extraction. Cosine similarity is used to compare the features. The network eventually learns the similarity through distances in an embedding space. Incorrect names have low distances from, or high similarity scores to, the valid names they resemble, while valid names not in the database have high distances and low similarity scores. This method captures font-based differences, like if an "l" is replaced with an "I", since Siamese neural networks have been effectively used to detect homoglyph attacks by analyzing visual similarities between characters [37].

Improving Database Scalability

Multi-Agent LLM Prompting. This method leverages multiple language models, specifically GPT and Claude, to provide error likelihood scores based on web history, phishing probability, and string similarity, based on multiple similarity metrics (exact match probability, edit distance). Prompts include expert framing, where the LLMs are assigned expert roles to better detect error patterns. Past studies show that assigning agents dedicated roles and utilizing several agents to collaborate and hold unique perspectives enhances the system’s ability to analyze data [38, 39]. The LLMs output a probability score of how likely a business name is to be incorrect. This method bypasses the need for a static database by using models to access external knowledge. While expensive, utilizing a multi-agent approach has been shown to offer generalization, contextual awareness, and flexibility beyond traditional rule-based systems, especially in complex tasks [40].

Webscraper Validation. To validate business legitimacy, I call the DuckDuckGo API using the business name as the search query [41]. If no relevant or high-quality website is found, the name is flagged as "Wrong". An LLM evaluates whether the top search result corresponds to the business name, accounting for minor variations. In the use case of fraud detection, this method effectively detects fraudulent companies, as scams often lack an online presence or return irrelevant search results. By leveraging real-time web data, the webscraper reduces reliance on manually updated databases that could be incomplete or outdated.

3.1.4 Aggregate Scoring

Based on the effectiveness of individual methods, scores from the above methods can be combined into an aggregate score that provides the probability risk of a business name being erroneous with more context. There are several ways to determine an aggregate score.

LLM Prompting

An aggregation LLM synthesizes scores into a final error probability, dynamically adjusting weights based on contextual cues. The LLM is asked to take on the role of a bank fraud expert to convey to the LLM the importance of the task of error detection, and it uses the context of how each score was calculated to determine how much certain scores should be weighted. This allows for a flexible, adaptive approach that takes into account the advantages and drawbacks of each individual method and adjusts the weight of each feature, leading to more accurate, context-aware outputs [42].

Logistic Regression

An alternative that reduces the cost and runtime is to train a logistic regression model to find the optimal weights for each score. This is a simple, foundational model that does not require many computational resources [43]. The data is randomly split into 70% training data and 30% testing data before training the model.

One limitation of this method is that logistic regression models the decision boundary as a linear function of the input values [43, 44]. For several of the individual methods, like string similarity, for example, it is possible that the valid class has very high values due to exact matches in the database and low values due to smaller businesses being very different from the database names, while the incorrect class has scores in the middle since they are likely to be similar to known valid company names. Thus, a linear decision boundary may not accurately capture this nuance.

Random Forest Classifier

The Random Forest classifier provides a robust and flexible alternative to logistic regression, especially in this case where the relationships between features are complex and non-linear. Unlike logistic regression, which assumes a linear decision boundary, a random forest creates multiple decision trees that can capture more intricate patterns in the data. Each tree is trained on a random subset of the data, and the final decision is made through a majority vote of the individual tree predictions. Using random forests reduces the chance of overfitting compared to individual decision trees. A simulation study comparing random forest and logistic regression found that while logistic regression often achieved higher accuracy, random forests had a higher true positive rate under certain noisy conditions, highlighting their complementary strengths in different data scenarios [45]. The data is split into the same 70% for training and 30% for testing as used for logistic regression, with 100 estimators (decision trees) used.

By using the Random Forest classifier, the importance of the features can be evaluated by ranking the individual scores based on their contribution to the model's decision-making process, providing insights into which factors are most indicative of errors [46]. This can guide future model improvements and provide help in refining the input features for better error detection.

3.2 TabuLa for Transaction Error Detection

Detecting erroneous financial transactions poses challenges due to class imbalance, where valid transactions vastly outnumber fraudulent or erroneous ones. This imbalance hinders the performance of classification models, as they often learn to prioritize accuracy on the majority class at the expense of the minority. To address this, I extend TabuLa, a generative model built upon DistilGPT-2, by incorporating a parameter to manage class imbalance during the sampling process to preserve the underlying distribution of transaction data while correcting for class disparity [28, 47].

3.2.1 Proposed System

The error detection pipeline is shown in Figure 3.2. This hybrid approach combines class-aware generation, structured preprocessing, and robust classification to improve error detection performance for tasks like fraud detection and financial auditing. The system begins by taking in tabular transaction data and normalizing the data. The data generation process involves training on the original input data to generate synthetic data samples that are representative of the original data, ensuring that class imbalance is addressed, either through random undersampling or a class balancing component within TabuLa. A Random Forest classifier is then trained on the synthetic dataset, and it is used to classify transactions from the original input as "Valid" or "Wrong".

3.2.2 Class-Balanced Generation

To mitigate class imbalance during synthetic data generation, I introduce a class control mechanism into TabuLa’s sampling process. The proposed method calculates the target number of samples required for each class and enforces these limits during generation to produce only the required number of samples per class, ensuring a balanced synthetic dataset. This approach prevents overrepresentation of majority classes and mitigates biases that could otherwise propagate in downstream machine learning models.

3.2.3 Comparison with Traditional Resampling

To validate the effectiveness of this data generation strategy, I conduct comparative experiments against traditional techniques, such as undersampling and oversampling. Undersampling randomly removes samples from the majority class to match the minority class. While effective in balancing class proportions, it often results in information loss and reduced generalization. On the other hand, oversampling, which involves replicating samples from the minority class or generating samples through SMOTE, can lead to overfitting and less diverse synthetic data. By integrating class balancing directly into the generative model, I aim to balance data diversity and representativeness.

Additionally, I analyze the generated data across different values of the class balancing parameter to determine how to best incorporate this component to improve performance.

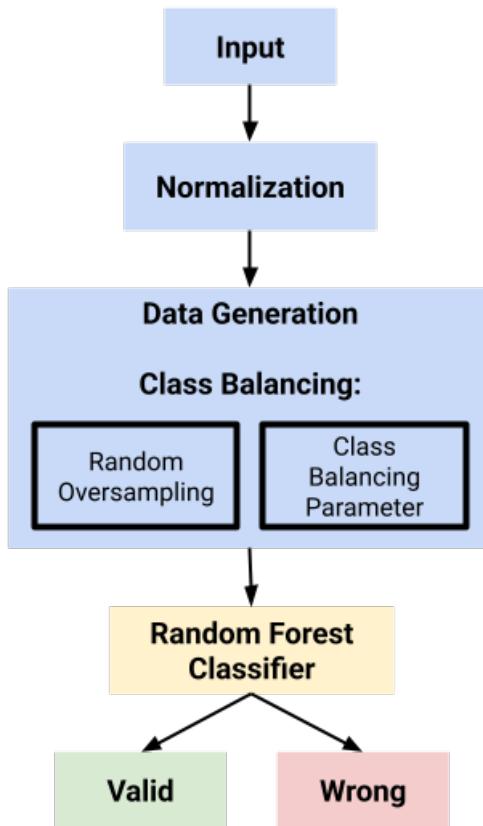


Figure 3.2: Transaction Error Detection System Pipeline. The input transaction data is normalized, and either random oversampling or a class balancing parameter within TabuLa is used to generate class-balanced synthetic data. A Random Forest classifier trained on the synthetic data is used to classify transactions as either "Valid" or "Wrong".

3.2.4 Preprocessing

In addition to class balancing, I explore preprocessing strategies, such as normalizing numerical features before training, to assess their influence on data generation and model robustness. These experiments will offer insights into how to best configure TabuLa, ensuring it effectively addresses class imbalance while preserving the statistical characteristics of the original data.

3.2.5 Model Training

Using the most effective configuration of preprocessing and class balancing methods, a synthetic dataset is generated, and a Random Forest classifier is trained to classify transactions as "Valid" or "Wrong". The models are trained with the following setup:

- Training/Test Split: 70% of the original data is used to train the data generation model, and the Random Forest classifier is trained on all of the synthetic data. The Random Forest classifier is tested on the other 30% of the original dataset.
- Dataset Size: Due to training resource limitations, most experiments will use 10k rows of data and generate 10k synthetic transactions.
- Number of Trees: 100 estimators to ensure a stable average and reduce overfitting.

The Random Forest model is selected for its ability to handle high-dimensional data and complex interactions between features. Feature importances derived from the trained model help interpret which attributes, like transaction amount or location, contribute most to classification. Different random states (42 and 21) are used to validate the results.

Chapter 4

Experiments

4.1 Datasets

To evaluate the robustness of the proposed error detection methods, I conduct experiments on two types of data: structured business name entries and real-world financial transactions. Each dataset is constructed or selected to address the specific challenges of its respective error detection system. For business name classification, it is necessary for the names to cover both valid names and the different kinds of incorrect names that users might enter during account registration, whether accidentally or maliciously. For transaction error detection, this process involves handling numerical data with severe class imbalance.

4.1.1 Business Name Error Detection

To ensure the model is able to handle all kinds of business names potentially seen during account registration, a dataset is constructed to capture three possible categories of business names: (1) authentic business names from established corporations, (2) variations of legitimate names representing accidental misspellings and malicious attempts to mimic real company names, and (3) unknown, yet plausible, business names that do not belong to any real entity. This structured dataset allows for the model to be tested in a realistic yet controlled setting.

The constructed dataset consists of 4,297 business names. Category 1 includes 1,000 names from the Fortune 500 lists to establish a reference set of real companies. Category 2 was generated using GPT-4o to create 3 variations of each of the 1000 valid names in Category 1. The modifications are intended to replicate potential typos and tactics used by fraudsters, like character substitutions, spacing adjustments, changes in capitalization, and punctuation variations. There are 2,961 names in Category 2 after removing 31 names containing invalid characters or exactly matching the original name. Category 3 was also generated using GPT-4o to ensure the model does not automatically classify unknown names as wrong. The prompt asked for professional-sounding names spanning a variety of industries, including single-word names, compound names, formal corporate names, and abbreviations. Since these names are intended to represent smaller, not widely recognized businesses, the prompt also ensured that generated names did not resemble any existing company.

These three categories of business names are necessary components of the testing dataset.

Category 1 ensures that real, verified companies are correctly classified as valid, serving as a baseline for legitimate entities. These must be classified correctly since they are known companies in the database, and misclassifying such companies could result in negative consequences for the bank, like losing clients and wasting resources on handling the perceived error. Category 2 represents the main goal of the error detection approach, which is identifying modified versions of real company names as incorrect. Detecting these subtle alterations is necessary in protecting other users from mistakes and potential scammers. Category 3 tests for unknown, but valid, companies to determine if the model can differentiate between incorrectly typed names and legitimate names, since both types of business names would not have a direct match to a company in the database. The proposed system should be able to draw two decision boundaries – one close to 100% similarity to distinguish between Categories 1 and 2 (completed through the database check), and one determined by the other learning techniques to classify Categories 2 and 3.

4.1.2 Transaction Error Detection

The MLG-ULB dataset, obtained from Kaggle¹, is used to evaluate the model. The dataset contains credit card transactions by European cardholders over two days, with 31 columns for each transaction and with fraud instances making up 0.172% of all 284,807 transactions. While the data is real rather than synthetic, to preserve confidentiality, the features are obtained with PCA. This dataset is currently being used since it is highly imbalanced and it has fewer rows and features than other datasets being considered, making it useful for decreasing training time for initial experiments. Since there are training resource limitations, 10k rows of this dataset are selected. The first 10k rows are used due to the natural temporal order of transactions. This preserves the sequence of back-to-back transactions, as these may contain important contextual or behavioral patterns. Maintaining the time order helps ensure that the model learns from realistic transaction flows, which is especially relevant in tasks like fraud detection where timing and sequence can influence predictive performance. To validate the initial experiments, they are repeated on different subsets of 10k consecutive transactions.

4.2 Evaluation

When evaluating the performance of error detection systems, it is important to consider the risks and trade-offs associated with misclassification. Since financial systems and business registration processes operate in high-stakes environments, both false positives (flagging correct entries as errors) and false negatives (not catching errors) can have significant consequences.

4.2.1 Business Name Error Detection

Since banks currently dedicate resources to manually checking potential incorrect accounts, I incorporate a confidence-based classification approach, where the model outputs “Valid” or

¹<https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data>

“Wrong” only when confidence is sufficiently high; otherwise, it assigns an “Unsure” label for manual review. Though this introduces a cost for human verification, it helps balance the tradeoff between reducing false positives and ensuring incorrect cases do not go undetected. While widening the thresholds to categorize more names as “Unsure” further reduces the number of misclassifications, it is not feasible to manually check most of the names. Thus, the threshold is set so that the percentage of valid names that are classified as “Unsure” is less than 5%. The percentage of incorrect names in this case is higher, but since real-world datasets tend to be imbalanced, with valid names outnumbering incorrect names, it is more important to limit the number of valid names that require manual checks. The calculations for recall, accuracy, and precision do not take into account the “Unsure” predictions, since any companies classified as “Unsure” should be correctly classified after manual review and should not signal that a wrong prediction was made. Treating “Unsure” predictions as incorrect within the accuracy calculations would make the model appear to perform worse than when using a 2-class model when it is actually more responsible to include the third class due to the high-risk nature of financial errors.

I assess the model’s performance primarily using **recall**, since correctly identifying incorrect names is crucial while minimizing false negatives, which are instances where errors go undetected. Recall is defined as the proportion of incorrect names correctly classified by the model, given by $\frac{TP}{TP+FN}$. This metric is particularly important because failing to detect errors can lead to significant financial and security risks.

Additionally, I evaluate **accuracy** to provide an overall measure of classification correctness, computed as $\frac{TP+TN}{TP+TN+FP+FN}$. While accuracy offers a general performance overview, it may be misleading for imbalanced datasets, as a model biased toward the majority class can still achieve high accuracy without effectively detecting errors.

I also measure **precision**, which quantifies how often predictions of errors are correct, given by $\frac{TP}{TP+FP}$. Precision is particularly relevant for minimizing false positives, where legitimate businesses are incorrectly flagged as wrong. While stricter error detection can be a protective measure, excessive false positives may negatively impact customer experience and operational efficiency.

Finally, I evaluate the **Area Under the Curve (AUC)** of the receiver operating characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR). A higher AUC indicates better model discrimination between incorrect and legitimate names, making it a valuable metric for assessing overall model effectiveness.

4.2.2 Transaction Error Detection

To evaluate the effectiveness of the proposed synthetic data generation method, I train a Random Forest classifier on each generated dataset and assess its performance on a test set composed of 20% of the original dataset. Model performance is primarily measured using the **area under the curve (AUC)** of the receiver operating characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR) at varying thresholds. A higher AUC value indicates better performance in distinguishing between incorrect and legitimate transactions. Additionally, I apply a 0.1% cutoff threshold to assess the model’s ability to make accurate predictions at high confidence levels, ensuring robust error detection in real-world scenarios.

Chapter 5

Business Name Error Detection Results

5.1 Baseline: Fuzzy Matching

Figure 5.1 shows how current approaches perform in detecting incorrect business names, using both 2-class and 3-class confusion matrices. The 3-class confusion matrix demonstrates the reduced number of false positives and false negatives by incorporating the "Unsure" class. Table 5.1 further shows the performance improvement from including the "Unsure" class using the evaluation metrics described.

Classes	Recall	Accuracy	Precision	AUC
2	0.815	0.813	0.906	0.185
3	0.851	0.846	0.910	0.384

Table 5.1: Performance comparison of 2 classes vs. 3 classes

The baseline model of using Fuzzy Matching performs well in detecting widely-known, legitimate business names that are considered as the ground truth in the database. Since company names are first checked for a direct match before applying fuzzy matching, every company in the database is correctly predicted as valid.

The model also generally performs well for invalid variations of legitimate names, with most predicted as wrong and some requiring a manual check. There were a few incorrect names predicted incorrectly as valid, but there is no clear pattern of what modification resulted in more incorrect predictions since such variations included space variations, case alterations, and character substitutions.

Most of the incorrect classifications are from predicting whether the synthetically generated business names from Category 3 are legitimate or wrong. Since fuzzy matching looks for matches within a provided database, business names that are not in the database have a low fuzzy matching score, resulting in mostly "Wrong" predictions or requiring a manual check. Additionally, the fuzzy matching scores for erroneous modifications and valid businesses not in the database are similar, with an average of 24.01 and 26.43, respectively, so it is difficult to determine the classification based on the fuzzy matching score alone.

While this baseline model performs well when distinguishing between legitimate names

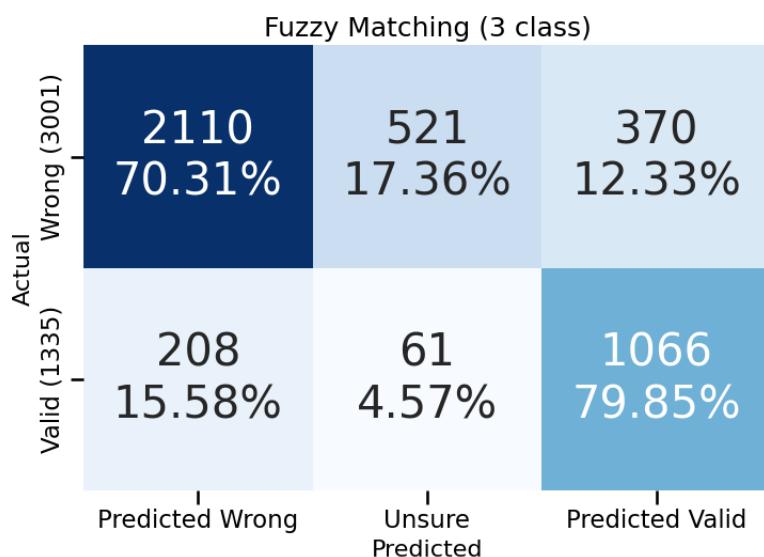
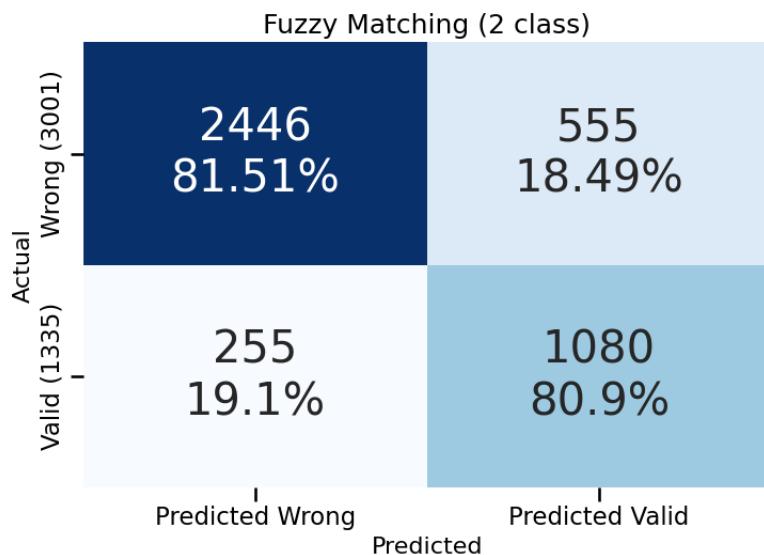


Figure 5.1: Fuzzy Matching Performance of 2 classes vs. 3 classes

and variations of legitimate businesses, it is not effective at determining whether unknown businesses—ones that could potentially be valid smaller companies—are legitimate or incorrect.

5.2 Individual Methods

The performance of each method individually is shown in Table 5.2. Each method produced a score from 0 to 100 for each business name, and the optimal threshold was calculated to find the decision boundary between valid and wrong names. The evaluation metrics discussed are also displayed.

Table 5.2: Performance comparison of individual methods

Method	Threshold	Recall	Accuracy	Precision	AUC
BERT	0.2004	0.9881	0.8954	0.8679	0.7744
Webscraper	65.1303	0.9488	0.7500	0.7380	0.7352
CNN	52.3046	0.9194	0.7257	0.7270	0.8112
Web History	0.0000	1.0000	0.6442	0.6442	0.4444
Phishing	18.0361	0.8956	0.8006	0.8136	0.8861
String Similarity	0.0000	1.0000	0.6442	0.6442	0.5810

The performance comparison highlights the strengths and weaknesses of each individual method in detecting mistakes. Among all methods, the BERT-based classifier achieved the highest accuracy (0.8954), recall (0.9881), and precision (0.8679), indicating its strong ability to correctly detect most incorrect cases with relatively few false positives. Its optimal decision threshold was determined to be 0.2004, suggesting that even moderately high BERT error probabilities are predictive of mistakes.

The Phishing score method showed the highest overall AUC (0.8861), indicating its capability in separating valid from wrong names across all thresholds. Its best threshold was relatively high at 18.0361, meaning only higher phishing scores contribute to accurate error detection. Similarly, both the CNN-based model and Webscraper scores performed competitively, with AUCs of 0.8112 and 0.7352, respectively, and recall above 0.9, indicating strong error sensitivity at thresholds of 52.3046 and 65.1303.

On the other hand, Web History and String Similarity methods defaulted to a threshold of 0.0000, classifying all entries as incorrect. This resulted in perfect recall (1.0000) but much lower accuracy (0.6442) and precision (0.6442), indicating many false positives. These features may provide limited discriminative power on their own and could be better suited for inclusion in ensemble methods. It is also possible that a single threshold does not accurately capture the score distribution for a method. For example, when examining the string similarity score, high values and low values could correspond more to valid names, since they represent either exact matches or smaller companies not in the database, while scores in the middle could correspond to incorrect names, since they represent minor mistakes. Score aggregation could be helpful in this context to avoid using a single threshold during classification.

While multiple methods demonstrate promising performance, BERT and Phishing scores stand out for their high recall, precision, and AUC. The chosen thresholds reveal how each method must classify to achieve peak performance, with lower thresholds indicating broader classification of cases as wrong, and higher thresholds suggesting more selective error detection.

5.3 Aggregate Results

Table 5.3 shows the evaluation metrics considered for each of the aggregation methods. While the flexibility and nuance offered by LLM Prompting is expected to improve performance by better capturing the complexities of the issue, logistic regression models and Random Forest classifiers outperform the LLM-based approach.

Aggregation Method	Recall	Accuracy	Precision	AUC
LLM Prompting	0.911	0.841	0.854	0.234
Logistic Regression	0.987	0.970	0.970	0.986
Random Forest Classifier	0.987	0.977	0.980	0.987

Table 5.3: Performance comparison of aggregate scoring methods

5.3.1 LLM Prompting

Using LLM prompting, more business names from both the valid and wrong groups are classified as wrong, as seen in Figure 5.2. This increases the amount of errors correctly classified as wrong from the baseline model, from 70.31% of wrong names to 88.25% of wrong names, and it decreases the false negative rate from 12.33% to 8.57%. However, it also decreases the amount of valid names correctly labeled as valid, from 79.85% to 67.82%, and increases the amount of valid names incorrectly labeled as incorrect, from 15.58% to 27.37%. These classifications were calculated with a threshold of 70 and 74, with risk scores over 74 classified as "Wrong", risk scores between 70 and 74 classified as "Unsure", and risk scores under 70 classified as "Valid". From the results, score aggregation using LLM prompting improves performance on the "Wrong" class but shows worse performance on the "Valid" class. While it is important to catch possible mistakes and errors, this could have negative consequences for customers with valid account names that are flagged as incorrect.

One consequence of using LLMs to calculate an aggregate score is that the calculation is not standardized between each score prediction. Table 5.4 shows several examples of business names from each category and how the LLM-based aggregation method performed, given their individual scores. Looking at two cases of valid business names, IBM and Home Depot, with similar scores in each category given to the LLM, the aggregate score is drastically different, with IBM having a risk score of 88 and Home Depot having a risk score of 34.

To consider which cases of the wrong names are incorrectly predicted as valid, examples of misclassified names and their scores are included. The modification that causes these misclassifications does not seem to be consistent, since the misclassifications cover a range of the name modifications, including punctuation and spacing changes. Since individual methods

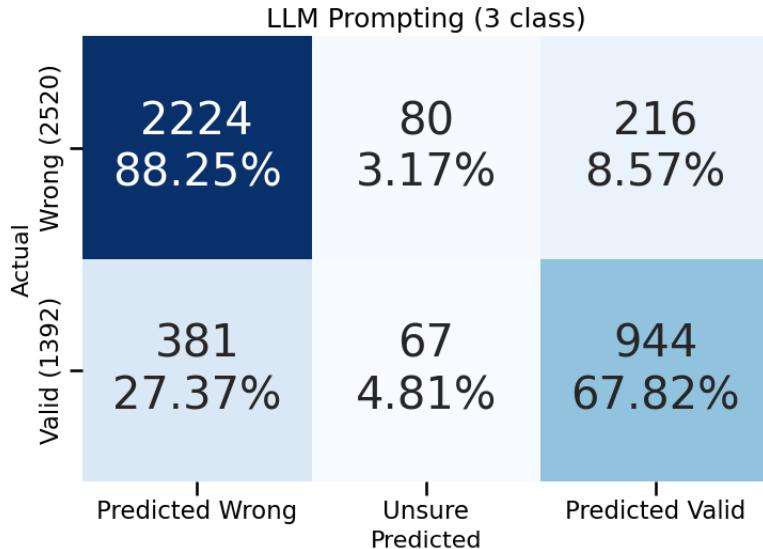


Figure 5.2: Aggregate performance from LLM prompting

are not as robust to these minor changes, this results in low risk scores for some methods, like a webscraper score of 5 for "eBay." and a BERT score of 0 for "W. W. Grainger". The LLM is unable to determine if individual scores are accurate, so errors in individual methods could result in a low aggregate score. While LLM prompting accurately detects most incorrect cases, there were still a large number of names that were misclassified in similar ways.

Finally, business names in Category 3 vary significantly within the individual scores, making it difficult for the LLM to utilize individual scores to accurately output an aggregate score. In the included examples, the individual scores differ especially for the webscraper and string similarity methods. This could be because during the dataset creation, due to the large number of companies that exist, the generated names might accidentally refer to small companies with some web presence that gets picked up by the webscraper. Additionally, depending on the industry and marketing strategies of these smaller companies, they could have varied levels of web presence. Regarding the differences in string similarity score, the names of smaller, less widely known companies might happen to be similar to larger companies; rather than accepting this as a valid company, the system might process this as a modification of the existing company in the database. When prompted for its reasoning behind the aggregate score, the LLM took into account the number of features with high scores, with some methods considered more strongly, based on what fraud experts consider most relevant and trustworthy, like web-related data. Thus, the variability in multiple scores, especially the webscraper score, results in worse performance for Category 3 names than for Category 1 and 2 names.

Company	Is Wrong	CNN	Webscraper	BERT	String Similarity	Phishing	Web History	Aggregate
IBM	0	88.33973	5.0	0.0	100.0	5.0	95.0	88.0
Home Depot	0	34.49308	0.0	0.0	100.0	5.0	95.0	34.0
eBay.	1	46.18839	5.0	69.431	65.0	5.0	95.0	63.0
W. W. Grainger	1	69.989876	85.0	0.0	20.0	5.0	85.0	69.0
Nexoria	0	44.147232	90.0	75.229	70.0	55.0	85.0	78.0
OmniVerse	0	33.013943	5.0	76.228	25.0	25.0	85.0	25.0
Crescendo Health	0	35.287697	60.0	75.842	25.0	5.0	85.0	63.0

Table 5.4: Aggregate scores from LLM prompting

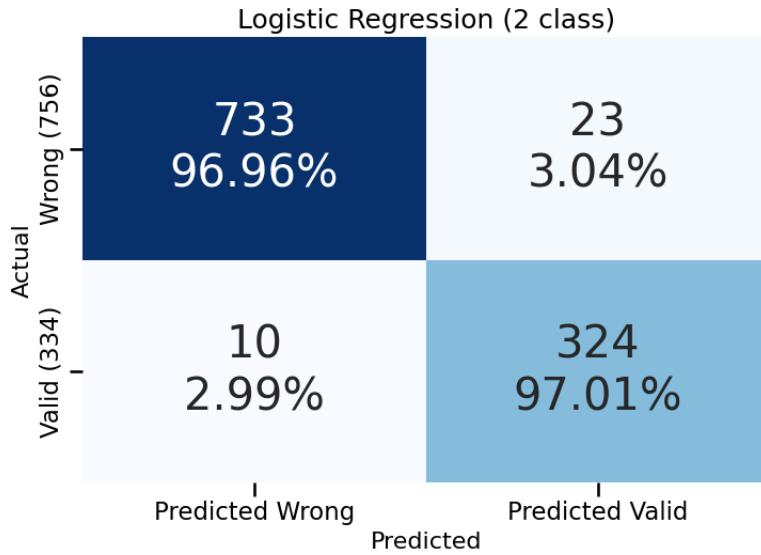


Figure 5.3: Aggregate performance from logistic regression

5.3.2 Logistic Regression

Figure 5.3 shows the significant improvement from using a logistic regression model, with a false negative rate of only 3.04%, compared to the false negative rate of 8.57% from the LLM-based aggregation method. While it can detect incorrect names with 98.65% accuracy, it struggles more with valid business names. This could be due to the smaller class size of valid names in the dataset since there is less data to train on, but it could also result from the non-linear decision boundary between valid and wrong business names.

5.3.3 Random Forest Classifier

As seen in Figure 5.4, using a Random Forest classifier has an improved ability to detect valid business names from the logistic regression model, resulting in a false negative rate of only 1.35%, a further improvement from the false negative rate of 3.04% from using logistic regression. This could be due to the non-linear decision boundary between scores for valid and wrong names, since there are valid names with low individual method scores that could

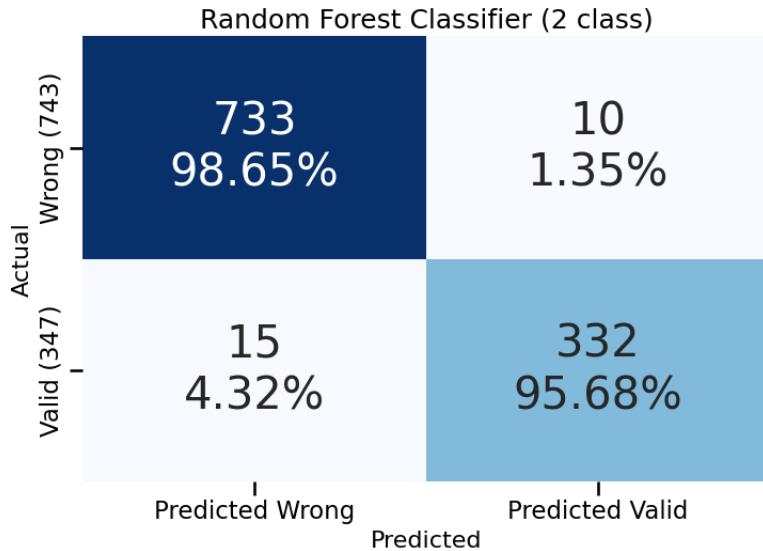


Figure 5.4: Aggregate performance from Random Forest classifier

be classified as wrong when using logistic regression.

Looking into the names classified incorrectly by both the logistic regression model and the Random Forest classifier, the misspelled names predicted as valid all include spacing variations. They also had low risk scores for the BERT model and the phishing prompt, suggesting that these individual methods are not robust in detecting spacing mistakes.

The Category 3 company names classified incorrectly as wrong all had multiple high risk scores. Upon searching some of these names, like "ARC Holdings" and "Synthesis Ventures", while they were generated in a way as to not resemble existing companies, some have legitimate company websites and articles. However, they still have a high webscraper risk score. This could indicate that while at first glance these companies might appear valid, DuckDuckGo might be able to provide more accurate insight on the validity of the companies based on the results; especially compared with large companies with many results, DuckDuckGo could consider smaller companies to be less trustworthy. Therefore, there is a chance that the company names were incorrectly generated to represent company names that are widely regarded as legitimate.

Chapter 6

Transaction Error Detection Results

Unlike the business name error detection task, which included a more nuanced three-class output, this task was framed as a binary classification problem, identifying transactions as either "Valid" or "Wrong". This binary framing simplifies the prediction objective but also requires the classifier to maintain high discrimination power across imbalanced classes. Table 6.1 shows the AUC values for various experimental setups, illustrating the model's performance under different preprocessing conditions. By comparing the AUC values, the table focuses on the model's ability to rank positive and negative instances correctly across all possible classification thresholds. This makes it particularly suitable for imbalanced datasets, where fixed-threshold metrics like accuracy can be misleading.

6.1 Baseline

To determine a baseline for assessing the effects of the proposed method in subsequent experiments, the baseline model was trained using the full, original version of the MLG-ULB dataset without any form of class balancing or feature normalization. A Random Forest classifier was employed due to its robustness and interpretability, as well as its relative insensitivity to unscaled features and class imbalance. The resulting ROC curve, shown in Figure 6.1, corresponds to an AUC of 0.796, as reported in Table 6.1. This performance indicates a strong capacity to separate the two classes based on the raw input features. However, from the confusion matrix in Figure 6.2, while the baseline model performs well on the valid transactions, it has significantly worse performance in detecting wrong transactions, with 23.47% of wrong transactions incorrectly labeled as valid, suggesting a tendency toward favoring the majority class.

This misclassification pattern reflects the effects of class imbalance in the training data, where valid transactions likely outnumber wrong ones significantly. Despite the random forest algorithm's inherent strategies for handling imbalance, such as bootstrapped aggregation and internal feature bagging, the model still exhibits bias toward the majority class [48]. These findings highlight the need for, and serve as a reference point for, evaluating the effectiveness of data preprocessing techniques like normalization and random undersampling, which can help reduce the skew and improve the classifier's performance. Random forests tend to handle class imbalance better than many linear models due to their ability to learn from subsets of

Table 6.1: AUC values for each experiment without class balancing

Experiment	Description	AUC
Baseline	Original MLG-ULB dataset, Random Forest classifier	0.796
Not normalized	2000 rows of MLG-ULB dataset, random undersampling, Random Forest classifier	0.530
Normalized	2000 rows of MLG-ULB dataset, random undersampling, normalized, Random Forest classifier	0.750

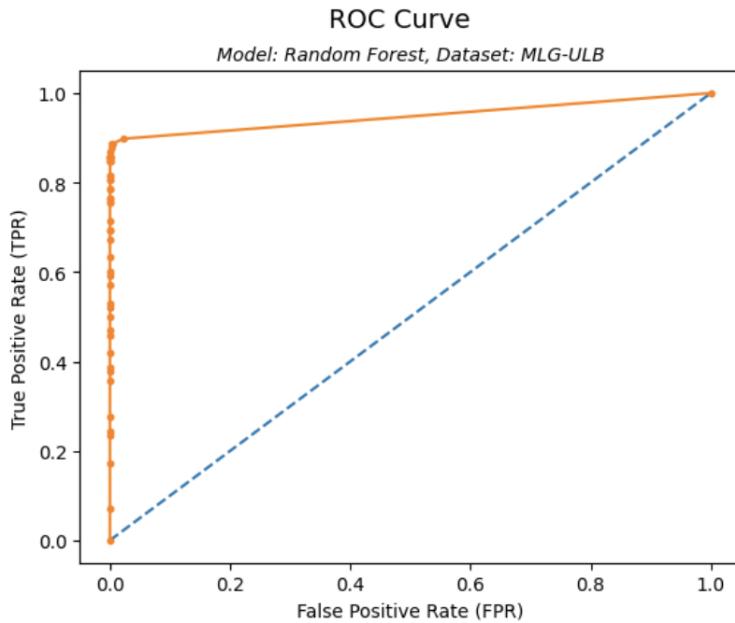


Figure 6.1: Baseline ROC curve

data and adjust thresholds internally, but improvements are expected using preprocessing and class balancing methods.

6.2 Normalization

Figure 6.3 shows the impact of normalizing the data features before training the model. Both the normalized and unnormalized models were trained for 400 epochs on a reduced version of the MLG-ULB dataset, consisting of 2000 rows obtained after randomly undersampling the majority class, due to resource and training time constraints. Using TabuLa, 2000 synthetic rows were generated. This allows for testing the performance of the Random Forest classifier on TabuLa-generated synthetic data, without the class balancing addition directly incorporated into the TabuLa data generation process.

The ROC curves demonstrate an improvement in the trade-off between the true positive rate (TPR) and the false positive rate (FPR), or an increased TP/FP ratio, when normalization

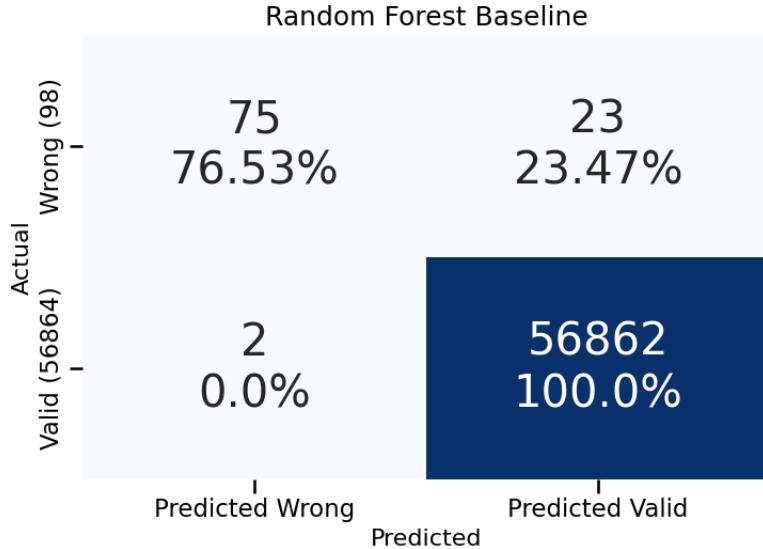


Figure 6.2: Baseline confusion matrix

is applied. The improvement is further reflected in the AUC values, from Table 6.1. The model trained on unnormalized data achieves an AUC of 0.530, indicating near-random performance, while the normalized model achieves an AUC of 0.750, showing significant improvements. This improvement suggests that normalization enables the model to better identify useful signals in the data by addressing scale discrepancies among features. Without normalization, features with larger numeric ranges may dominate the model’s decision-making process, masking the contribution of features on smaller scales. This is particularly problematic for decision-tree-based models when features with larger numeric ranges disproportionately influence the model, not because they carry more predictive information, but simply due to their scale.

It is important to note that these results were obtained using a small, balanced subset of the data. The significant drop in AUC for the unnormalized model compared to the baseline, which achieved an AUC of 0.796 on the full dataset, could result from several factors. Reducing the dataset size to only 2000 rows reduces the amount of training data available to the model, which limits its ability to generalize. Additionally, the use of random undersampling, while beneficial for improving performance on the minority class, could negatively impact the model’s ability to correctly classify the majority class by discarding potentially informative examples. This trade-off is more evident in the unnormalized experiment, where the model not only struggles with reduced data but also suffers from uncorrected feature scale issues. Together, these results support the continued use of normalization in future experiments to ensure more stable and meaningful learning across feature dimensions.

Table 6.2: AUC values for each experiment with class balancing

Experiment	Description	AUC
No class balancing	10k rows of MLG-ULB dataset, normalized, Random Forest classifier	0.745
Undersampled	10k rows of MLG-ULB dataset, normalized, random undersampling, Random Forest classifier	0.760
1%	balanced to 1% minority class	0.750
5%	balanced to 5% minority class	0.791
10%	balanced to 10% minority class	0.798
25%	balanced to 25% minority class	0.786

6.3 Class Balancing

Due to issues with long training times, only 10k rows were used for the initial experiments to determine the effects of class balancing. Before evaluating the class balancing component, I ran a baseline experiment using 10k rows of the original MLG-ULB dataset. To compare this method with traditional techniques, I also tried random undersampling of the majority class and ran an experiment using 10k rows of the resulting dataset. This was to determine if using LLM-based models to address class imbalance caused a significant enough improvement to justify the higher costs, resources, and time required to implement them. From Table 6.2, the AUC for the baseline with 10k rows is 0.745 and the AUC for using random undersampling is 0.760. Figures 6.4 and 6.5 show the ROC curve and confusion matrix for random undersampling. There is a slight improvement from the baseline, with a false negative rate of 23.47%, but the model is not able to catch 10.2% of incorrect transactions, which could have significant negative consequences for customers.

To assess the class balancing component, I trained on 10k rows of the normalized MLG-ULB dataset and generated 10k rows with and without class balancing. Figure 6.6 shows the ROC curves and Figure 6.7 shows the confusion matrices for generating data without class balancing and generating data with class balancing of the minority class to 10%. The second ROC curve rises more steeply, showing improved performance due to a higher true positive rate across false positive rates. The confusion matrices support this observation, with a false negative rate of 12.84% with class balancing, an improvement from a false negative rate of 20.41% without class balancing. From Table 6.2, the AUC for generating without class balancing is 0.760 and the AUCs for generating with class balancing shows an improvement when incorporating the class balancing component for higher percentages of the minority class, with the highest AUC, 0.798, corresponding to 10% transaction errors, or the minority class.

Comparing the methods of random undersampling and class balancing during data generation, class balancing results in a higher AUC of 0.798 compared to 0.760 using random undersampling, indicating that the class balancing component has stronger overall discriminative ability. However, this improvement in AUC comes at a cost to the false negative rate, which is a crucial metric in the context of rare but critical events such as transaction errors. The false negative rate is 10.20% for random undersampling and 12.84%

for 10% class balancing, as shown in Figures 6.5 and 6.7, implying that more actual errors go undetected when class balancing is introduced, resulting in worse performance for the minority class when incorporating the class balancing component. This tradeoff suggests that while class balancing improves the model's general classification capability, it could dilute sensitivity toward minority class instances. Depending on which metric is most relevant, random undersampling and the class balancing component provide different improvements to the baseline model; if reducing missed errors is the goal, undersampling could be preferred, but when maximizing general model performance, class balancing provides better holistic performance.

While these additions significantly improve the model's ability to detect errors, dropping the false negative rate from 20.41% to 12.84%, they also increase the number of false positives, or valid transactions incorrectly classified as "Wrong". This system takes on a more cautious approach that can protect customers from financial errors, but it could also lead to frustration if their valid transactions are reported as incorrect 3.81% of the time. This reflects the tradeoff commonly seen between precision and recall. By using a more conservative perspective and flagging more anomalies, the model reduces the number of errors that slip through the system, which is critical in maintaining customer trust and safety. However, this could compromise the user experience by sending false alerts that cause confusion or require unnecessary follow-up actions that drain resources. Due to the risks associated with undetected errors, the system adopts the more cautious stance.

The results from these experiments demonstrate that both normalization and class balancing play critical roles in improving model performance on the transaction error detection task. Normalization enhances the model's ability to interpret feature scales effectively, by ensuring that features are on comparable scales, which enhances the model's capacity to learn meaningful patterns without being biased by differing feature magnitudes. Class balancing, whether through traditional random undersampling or data generation with adjusted class ratios, directly addresses the skewed distribution of transaction datasets. Both approaches result in compromises; undersampling risks losing valuable majority class data and reduces the model's understanding of typical transactions, while data generation could introduce synthetic patterns that do not accurately reflect real-world transaction nuances. Future work could explore hybrid methods that combine undersampling with synthetic data generation, or more advanced sampling techniques, such as SMOTE, to further refine the balance between caution and accuracy in real-world financial systems.

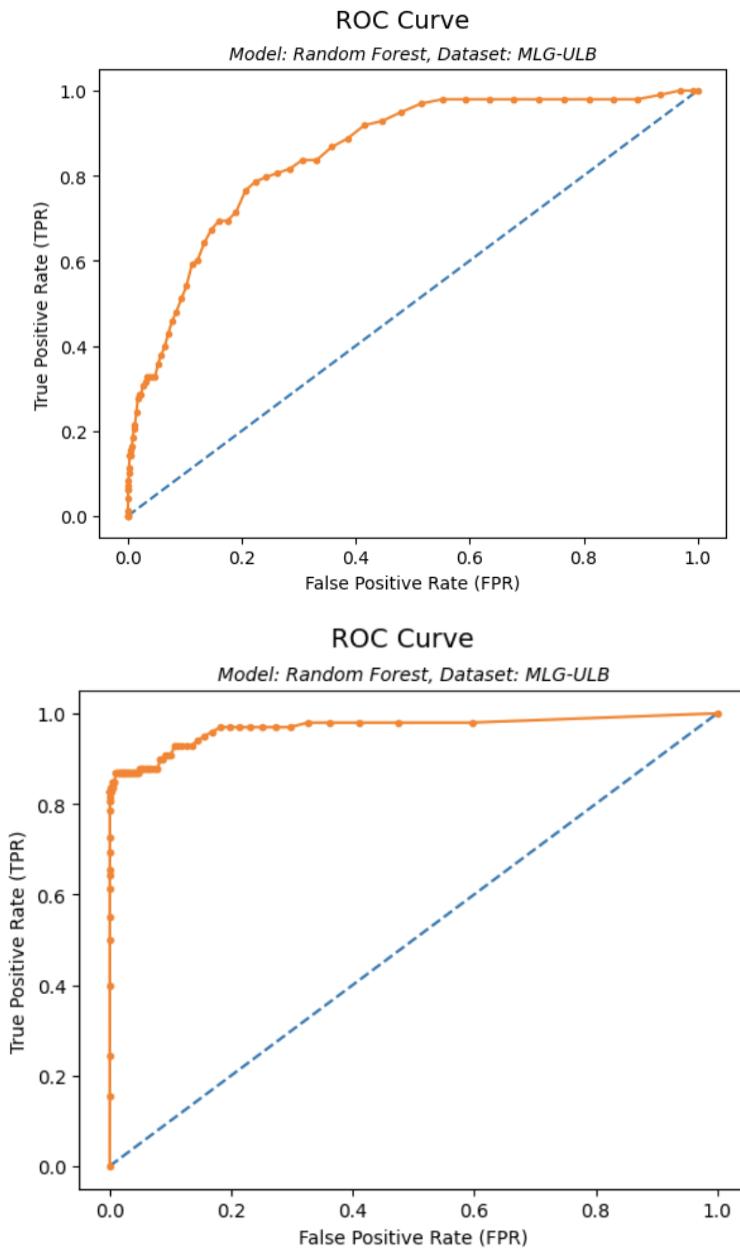


Figure 6.3: ROC curves for models trained without normalizing data (top) vs. normalizing data (bottom). The second curve shows the improved performance from normalizing the data.

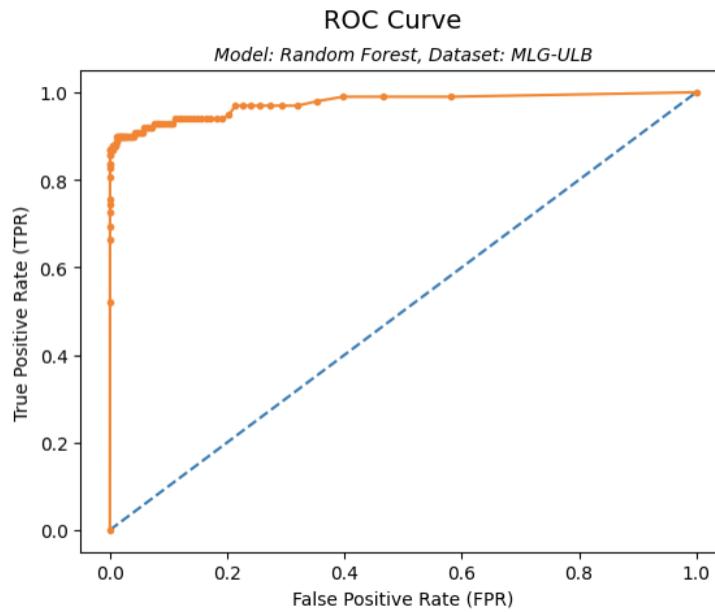


Figure 6.4: ROC curve for random undersampling. Random undersampling improves the performance from the baseline curve, as shown in Figure 6.1.

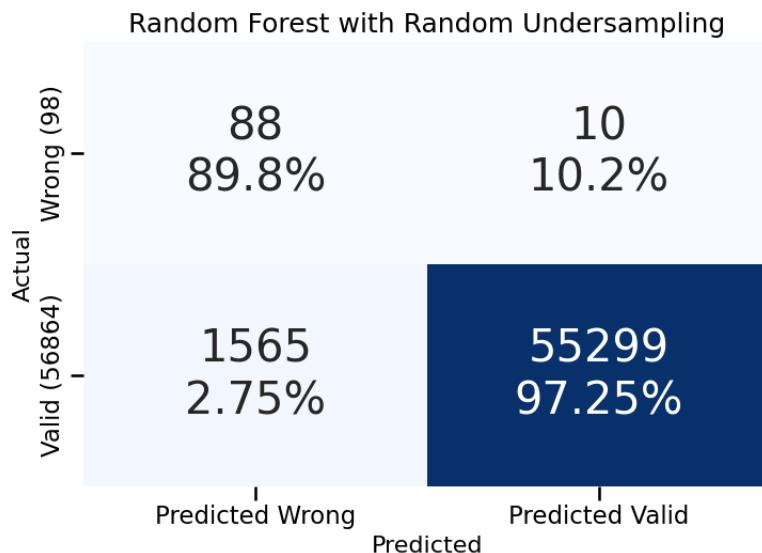


Figure 6.5: Confusion matrix for random undersampling. Random undersampling decreases the false negative rate from 23.47% in the baseline experiment (shown in Figure 6.2) to 10.2%.

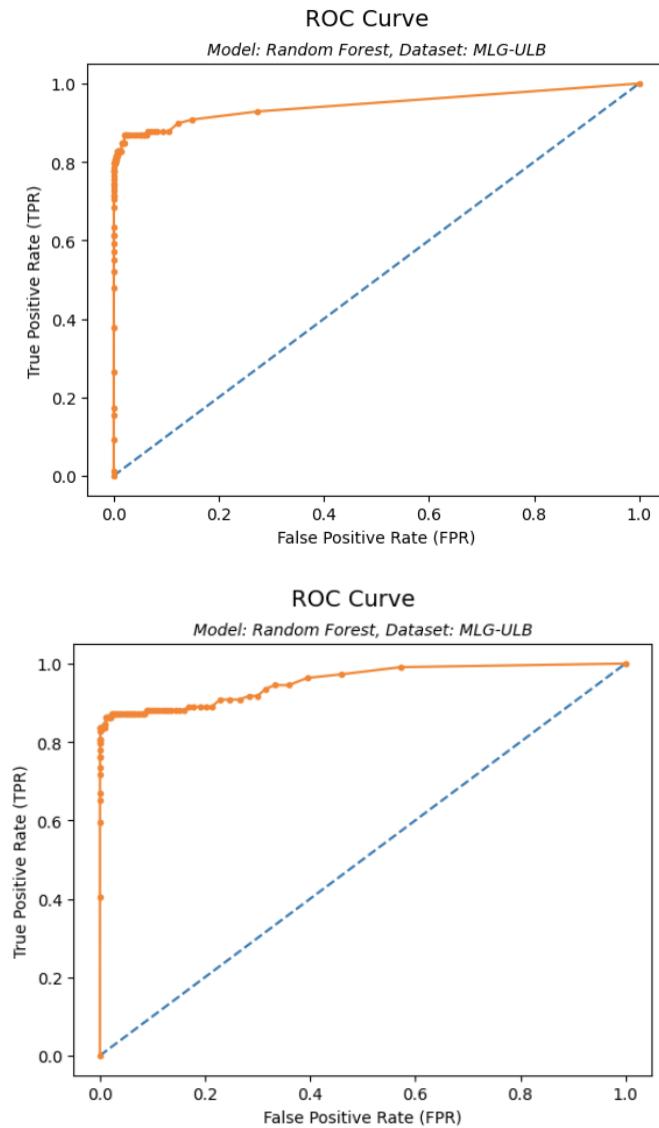


Figure 6.6: ROC curves for no class balancing (top) vs. class balancing using TabuLa (bottom). The second curve shows improved performance, with a higher true positive rate across most false positive rates, indicating that class balancing leads to better discrimination between valid and wrong transactions.

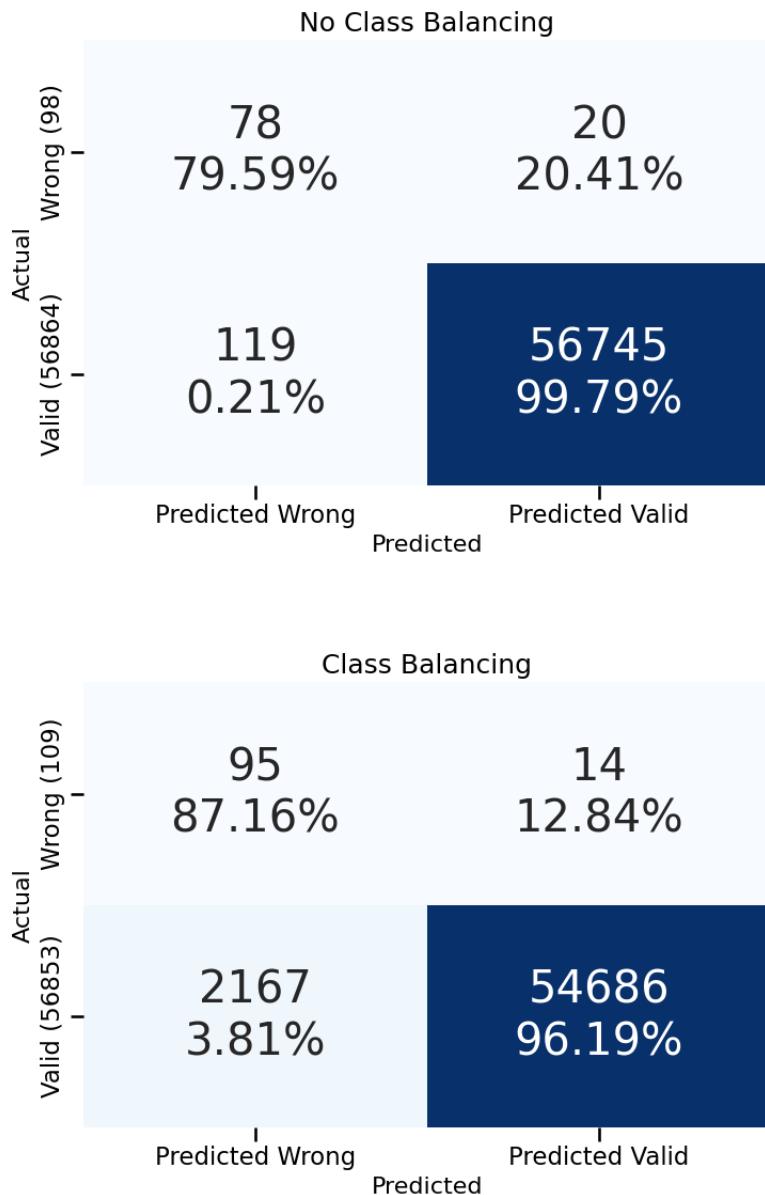


Figure 6.7: Confusion matrices for no class balancing (top) vs. class balancing using TabuLa (bottom). Adding the class balancing component decreases false negatives from 20.41% to 12.84%.

Chapter 7

Conclusion

The experiments conducted in both the business name error detection and transaction error detection tasks reveal the significant impact of the proposed methods on improving model performance. In the business name error detection task, the experiment results highlight the improved performance from the suggested methods, demonstrated by the increased recall from 0.815 to 0.987 while increasing the precision from 0.906 to 0.980, as shown in Table 7.1. This is critical in preventing serious financial or operational consequences. While using LLMs offers increased flexibility and context-awareness, simpler yet cost-efficient models like random forests also demonstrated impressive recall rates. This suggests that random forests are an effective choice when balancing performance with computational efficiency, especially in environments where resources or processing power are constrained.

Method	Recall	Accuracy	Precision	AUC
Baseline	0.815	0.813	0.906	0.185
Random Forest (Best Aggregate)	0.987	0.977	0.980	0.987

Table 7.1: Business name error detection performance comparison of baseline model vs. proposed model. The proposed system shows an increase in each of the evaluated metrics.

In the transaction error detection task, the development of a synthetic data generation pipeline greatly enhanced the model’s robustness in highly imbalanced settings, as shown in Figure 7.1. By employing techniques such as data normalization and class balancing, the model’s ability to identify minority class transactions, representing fraudulent or erroneous transactions, was significantly improved, decreasing the false negative rate from 23.47% using the baseline model to 12.84% using the proposed approach. Class balancing, especially when set to a 10% minority class ratio, resulted in the highest AUC score of 0.798. This suggests that increasing the representation of the minority class improves model discrimination. However, there is a tradeoff between the AUC and false positives and false negatives; while class balancing helped increase overall AUC, it had more false positives and false negatives than when random undersampling was used. This highlights the importance of choosing the right balance between these competing metrics, depending on the use case and desired model performance.

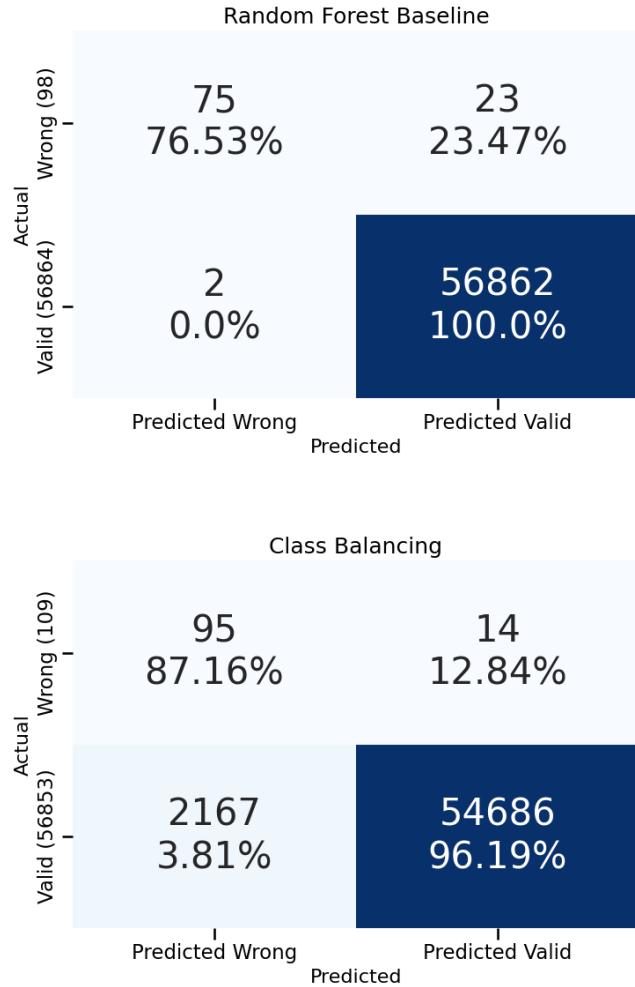


Figure 7.1: Transaction error detection performance comparison of baseline model (top) vs. proposed model (bottom). The proposed system decreases false negatives from 23.47% to 12.84% (top right cell).

In both tasks, the proposed systems demonstrate potential in enhancing error detection systems within financial contexts. The business name error detection approach not only reduces the risk of misclassification but also ensures that potentially costly mistakes are flagged for human review. The improvements made to the transaction error detection pipeline offer better capabilities of handling the complexities of real-world data, including class imbalance and large datasets. The suggested methods could be helpful in offering more robust, reliable, and scalable error detection systems that can be deployed in high-stakes environments, such as banking and finance.

7.1 Future Work

7.1.1 Business Name Error Detection

It would be helpful to validate the results on a more realistic dataset, since the limited size and GPT-generated data might not provide a completely accurate representation of the data the system might face. Additionally, realistic datasets would be imbalanced, with many more cases of valid names than wrong names, unlike the dataset used, which had 2961 wrong names, outnumbering the 1336 valid names. Further experiments on a real dataset could determine if the improvements in detecting wrong names are still present when wrong names make up a small minority of the dataset.

Additionally, if resource constraints are not considered, the LLM-based aggregation approach could be further developed by including more context in the prompt about each individual score; the nuances of each individual method and how reliable the score is can be explained to the LLM so that it can better determine how to weight each score.

Since web data is generally considered trustworthy, the webscraper could be further improved and given more consideration during aggregation. Webscraping from DuckDuckGo is constrained by a rate limit, and alternative sources like the Better Business Bureau (BBB) focus mainly on US companies. Future work could include expanding the web scraping functionality to include more global sources.

7.1.2 Transaction Error Detection

One challenge in transaction error detection is long training times. This makes it difficult to evaluate the model's performance on realistic datasets that have more features and larger datasets with more rows, including real bank data. I am currently using MIT SuperCloud to run the experiments [49]. With starting resource allocation, it takes a day to train on 10k transactions from a simplified dataset and generate 10k more samples. The classification process is shorter, but since many banks handle millions of transactions a day, it is necessary to ensure that the model checks each transaction, and in a reasonable amount of time so that clients do not face frustration during correct transactions and fraudsters, for instance, are caught quickly for incorrect transactions. To mitigate long training times, the TAEGAN model presents a GAN-based framework optimized for generating high-quality tabular data efficiently. By employing a masked auto-encoder as the generator, TAEGAN reduces computational complexity, allowing for large-scale transaction datasets [50].

Additionally, it is necessary to check that the generated data is valid and "business-possible". For example, an Android device cannot run iOS, and this must be enforced in synthetic data when considering devices and operating systems a transaction was made on. Current results are based on the MLG-ULB dataset, which provides features that have been obtained through PCA due to confidentiality issues. This makes it challenging to determine whether the generated data is reasonable and valid. When using real data, it is important to incorporate a validity check on the data during the synthetic data generation process. Past studies emphasize the importance of integrating data-centric AI techniques to guide synthetic data generation; by profiling data to capture complex nuances, these approaches ensure that generated data maintains logical consistency [51].

Future directions may also explore feature engineering by incorporating domain-specific features such as customer behavioral profiles or time-of-day patterns. Anomaly detection techniques, including isolation forests or autoencoders, could help better identify deviations from normal behavior [52]. Additionally, modeling transaction sequences over time using recurrent neural networks (RNNs) or temporal embeddings could improve detection of coordinated fraud or behavioral deviations. Integrating these techniques can lead to more accurate fraud detection systems.

Further experiments could also build on the techniques explored by testing hybrid approaches that use both undersampling and synthetic data generation methods. Other sampling techniques, like SMOTE, could also be incorporated to determine the best configuration of sampling and data generation to use to address the issue of class balancing.

References

- [1] D. Vallarino. *Detecting Financial Fraud with Hybrid Deep Learning: A Mix-of-Experts Approach to Sequential and Anomalous Patterns*. 2025. arXiv: [2504.03750](https://arxiv.org/abs/2504.03750) [cs.CR]. URL: <https://arxiv.org/abs/2504.03750>.
- [2] B. Libgober and C. T. Jerzak. *Linking Datasets on Organizations Using Half A Billion Open-Collaborated Records*. 2024. arXiv: [2302.02533](https://arxiv.org/abs/2302.02533) [cs.SI]. URL: <https://arxiv.org/abs/2302.02533>.
- [3] P. Raghavan and N. Gayar. “Fraud Detection using Machine Learning and Deep Learning”. In: Dec. 2019, pp. 334–339. DOI: [10.1109/ICCIKE47802.2019.9004231](https://doi.org/10.1109/ICCIKE47802.2019.9004231).
- [4] V. C. Pezoulas, D. I. Zaridis, E. Mylona, C. Androullos, K. Apostolidis, N. S. Tachos, and D. I. Fotiadis. “Synthetic data generation methods in healthcare: A review on open-source tools and methods”. In: *Computational and Structural Biotechnology Journal* 23 (2024), pp. 2892–2910. URL: <https://doi.org/10.1016/j.csbj.2024.07.005>.
- [5] Y. Lucas and J. Jurgovsky. *Credit card fraud detection using machine learning: A survey*. ArXiv e-prints. Oct. 2020. arXiv: [2010.06479](https://arxiv.org/abs/2010.06479).
- [6] S. Rezvani and X. Wang. “A broad review on class imbalance learning techniques”. In: *Applied Soft Computing* 143 (2023). URL: <https://doi.org/10.1016/j.asoc.2023.110415>.
- [7] J. Kim, T. Kim, and J. Choo. *EPIC: Effective Prompting for Imbalanced-Class Data Synthesis in Tabular Data Classification via Large Language Models*. ArXiv e-prints. Apr. 2024. arXiv: [2404.12404](https://arxiv.org/abs/2404.12404).
- [8] S. Zhang, Y. Hu, and G. Bian. “Research on string similarity algorithm based on Levenshtein Distance”. In: *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2017, pp. 2247–2251. DOI: [10.1109/IAEAC.2017.8054419](https://doi.org/10.1109/IAEAC.2017.8054419).
- [9] R. Roy and C. Yedurupaka. “IMPROVING THE EFFICIENCY OF FUZZY MATCHING ALGORITHMS IN AI AND ITS IMPLICATIONS IN AML”. In: *Shu Ju Cai Ji Yu Chu Li/Journal of Data Acquisition and Processing* 39 (Dec. 2024), p. 133. DOI: [10.5281/zenodo.7926784](https://doi.org/10.5281/zenodo.7926784).
- [10] S. Vaiwsri, T. Ranbaduge, and P. Christen. “Accurate and efficient privacy-preserving string matching”. en. In: *Int. J. Data Sci. Anal.* 14.2 (Aug. 2022), pp. 191–215. DOI: [{10.1007/s41060-022-00320-5}](https://doi.org/10.1007/s41060-022-00320-5).
- [11] B. Libgober and C. T. Jerzak. “Linking datasets on organizations using half a billion open-collaborated records”. en. In: *Polit. Sci. Res. Meth.* (Oct. 2024), pp. 1–20.

- [12] L. J. Sern, Y. G. Peng David, and C. J. Hao. “PhishGAN: Data Augmentation and Identification of Homoglyph Attacks”. In: *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*. IEEE, Nov. 2020, pp. 1–6. DOI: [10.1109/CCCI49893.2020.9256804](https://doi.org/10.1109/CCCI49893.2020.9256804). URL: <http://dx.doi.org/10.1109/CCCI49893.2020.9256804>.
- [13] N. An, L. Jiang, J. Wang, P. Luo, M. Wang, and B. N. Li. “Toward detection of aliases without string similarity”. In: *Information Sciences* 261 (2014), pp. 89–100. ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2013.11.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0020025513007974>.
- [14] A. Basile, R. Crupi, M. Grasso, A. Mercanti, D. Regoli, S. Scarsi, S. Yang, and A. C. Cosentini. “Disambiguation of company names via deep recurrent networks”. In: *Expert Systems with Applications* 238 (Mar. 2024), p. 122035. ISSN: 0957-4174. DOI: [10.1016/j.eswa.2023.122035](https://doi.org/10.1016/j.eswa.2023.122035). URL: <http://dx.doi.org/10.1016/j.eswa.2023.122035>.
- [15] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant. *Detecting Homoglyph Attacks with a Siamese Neural Network*. 2018. arXiv: [1805.09738 \[cs.CR\]](https://arxiv.org/abs/1805.09738). URL: <https://arxiv.org/abs/1805.09738>.
- [16] A. Gupta, L. S. Tomar, and R. Garg. *GlyphNet: Homoglyph domains dataset and detection using attention-based Convolutional Neural Networks*. 2023. arXiv: [2306.10392 \[cs.CR\]](https://arxiv.org/abs/2306.10392). URL: <https://arxiv.org/abs/2306.10392>.
- [17] E. L. Pontes, S. Huet, A. C. Linhares, and J. Torres-Moreno. “Predicting the Semantic Textual Similarity with Siamese CNN and LSTM”. In: *CoRR* abs/1810.10641 (2018). arXiv: [1810.10641](https://arxiv.org/abs/1810.10641). URL: [http://arxiv.org/abs/1810.10641](https://arxiv.org/abs/1810.10641).
- [18] D. Yang, X. Ke, Q. Yu, and B. Yang. “Enhanced LSTM: a Text Matching Aggregation Model”. In: *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*. Vol. 9. 2020, pp. 659–663. DOI: [10.1109/ITAIC49862.2020.9339154](https://doi.org/10.1109/ITAIC49862.2020.9339154).
- [19] A. M. Almuhaideb, N. Aslam, A. Alabdullatif, S. Altamimi, S. Alothman, A. Alhussain, W. Aldosari, S. J. Alsunaidi, and K. A. Alissa. “Homoglyph Attack Detection Model Using Machine Learning and Hash Function”. In: *Journal of Sensor and Actuator Networks* 11.3 (2022). ISSN: 2224-2708. DOI: [10.3390/jsan11030054](https://doi.org/10.3390/jsan11030054). URL: <https://www.mdpi.com/2224-2708/11/3/54>.
- [20] P. Maneriker, J. W. Stokes, E. G. Lazo, D. Carutasu, F. Tajaddodianfar, and A. Gururajan. “URLTran: Improving Phishing URL Detection Using Transformers”. In: *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*. 2021, pp. 197–204. DOI: [10.1109/MILCOM52596.2021.9653028](https://doi.org/10.1109/MILCOM52596.2021.9653028).
- [21] Y. Li, Y. Huang, G. Qi, J. Feng, N. Hu, S. Zhai, H. Xue, Y. Chen, R. Shen, and T. Wu. *Harnessing Diverse Perspectives: A Multi-Agent Framework for Enhanced Error Detection in Knowledge Graphs*. 2025. arXiv: [2501.15791 \[cs.AI\]](https://arxiv.org/abs/2501.15791). URL: <https://arxiv.org/abs/2501.15791>.

- [22] P. Chen, S. Zhang, and B. Han. “CoMM: Collaborative Multi-Agent, Multi-Reasoning-Path Prompting for Complex Problem Solving”. In: *Findings of the Association for Computational Linguistics: NAACL 2024*. Ed. by K. Duh, H. Gomez, and S. Bethard. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 1720–1738. DOI: [10.18653/v1/2024.findings-naacl.112](https://doi.org/10.18653/v1/2024.findings-naacl.112). URL: <https://aclanthology.org/2024.findings-naacl.112/>.
- [23] J.-t. Huang, J. Zhou, T. Jin, X. Zhou, Z. Chen, W. Wang, Y. Yuan, M. R. Lyu, and M. Sap. *On the Resilience of LLM-Based Multi-Agent Collaboration with Faulty Agents*. 2025. arXiv: [2408.00989 \[cs.AI\]](https://arxiv.org/abs/2408.00989). URL: <https://arxiv.org/abs/2408.00989>.
- [24] W. Huang, Z. Gu, C. Peng, Z. Li, J. Liang, Y. Xiao, L. Wen, and Z. Chen. *AutoScraper: A Progressive Understanding Web Agent for Web Scraper Generation*. 2024. arXiv: [2404.12753 \[cs.CL\]](https://arxiv.org/abs/2404.12753). URL: <https://arxiv.org/abs/2404.12753>.
- [25] A. Ahluwalia and S. Wani. *Leveraging Large Language Models for Web Scraping*. 2024. arXiv: [2406.08246 \[cs.CL\]](https://arxiv.org/abs/2406.08246). URL: <https://arxiv.org/abs/2406.08246>.
- [26] X. Fang, W. Xu, F. A. Tan, J. Zhang, Z. Hu, Y. Qi, S. Nickleach, D. Socolinsky, S. Sengamedu, and C. Faloutsos. *Large Language Models(LLMs) on Tabular Data: Prediction, Generation, and Understanding - A Survey*. ArXiv e-prints. Feb. 2024. arXiv: [2402.17944](https://arxiv.org/abs/2402.17944).
- [27] V. Borisov, K. Sessler, T. Leemann, M. Pawelczyk, and G. Kasneci. “Language Models are Realistic Tabular Data Generators”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: <https://openreview.net/forum?id=cEygmQNOeI>.
- [28] Z. Zhao, R. Birke, and L. Y. Chen. *TabuLa: Harnessing Language Models for Tabular Data Synthesis*. ArXiv e-prints. Oct. 2023. arXiv: [2310.12746](https://arxiv.org/abs/2310.12746).
- [29] Y. Lu, L. Chen, Y. Zhang, M. Shen, H. Wang, X. Wang, C. van Rechem, T. Fu, and W. Wei. *Machine Learning for Synthetic Data Generation: A Review*. 2025. arXiv: [2302.04062 \[cs.LG\]](https://arxiv.org/abs/2302.04062). URL: <https://arxiv.org/abs/2302.04062>.
- [30] A. D’souza, S. M, and S. Sarawagi. *Synthetic Tabular Data Generation for Imbalanced Classification: The Surprising Effectiveness of an Overlap Class*. 2025. arXiv: [2412.15657 \[cs.LG\]](https://arxiv.org/abs/2412.15657). URL: <https://arxiv.org/abs/2412.15657>.
- [31] Kanika and J. Singla. “Class Balancing Methods for Fraud Detection using Deep Learning”. In: *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*. 2022, pp. 395–400. DOI: [10.1109/ICAIS53314.2022.9742836](https://doi.org/10.1109/ICAIS53314.2022.9742836).
- [32] T. Gschwind, C. Miksovic, J. Minder, K. Mirylenka, and P. Scotton. *Fast Record Linkage for Company Entities*. 2019. arXiv: [1907.08667 \[cs.DB\]](https://arxiv.org/abs/1907.08667). URL: <https://arxiv.org/abs/1907.08667>.
- [33] SeatGeek. *FuzzyWuzzy: Fuzzy String Matching in Python*. <https://pypi.org/project/fuzzywuzzy/>. Version 0.18.0. 2023.
- [34] R. Ziv, I. Gronau, and M. Fire. *CompanyName2Vec: Company Entity Matching Based on Job Ads*. 2022. arXiv: [2201.04687 \[cs.SI\]](https://arxiv.org/abs/2201.04687). URL: <https://arxiv.org/abs/2201.04687>.

- [35] N. Poerner, U. Waltinger, and H. Schütze. *E-BERT: Efficient-Yet-Effective Entity Embeddings for BERT*. 2020. arXiv: [1911.03681 \[cs.CL\]](https://arxiv.org/abs/1911.03681). URL: <https://arxiv.org/abs/1911.03681>.
- [36] Z. Lin, J. Tan, D. Ou, X. Chen, S. Yao, and B. Zheng. “Deep Bag-of-Words Model: An Efficient and Interpretable Relevance Architecture for Chinese E-Commerce”. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD ’24. ACM, Aug. 2024, pp. 5398–5408. DOI: [10.1145/3637528.3671559](https://doi.org/10.1145/3637528.3671559). URL: <http://dx.doi.org/10.1145/3637528.3671559>.
- [37] V. R. and S. K.P. “Siamese neural network architecture for homoglyph attacks detection”. In: *ICT Express* 6.1 (2020), pp. 16–19. ISSN: 2405-9595. DOI: <https://doi.org/10.1016/j.ictex.2019.05.002>. URL: <https://www.sciencedirect.com/science/article/pii/S2405959519300025>.
- [38] M. W. U. Rahman, R. Nevarez, L. T. Mim, and S. Hariri. *Multi-Agent Actor-Critic Generative AI for Query Resolution and Analysis*. 2025. arXiv: [2502.13164 \[cs.MA\]](https://arxiv.org/abs/2502.13164). URL: <https://arxiv.org/abs/2502.13164>.
- [39] Q. Wang et al. *SchemaAgent: A Multi-Agents Framework for Generating Relational Database Schema*. 2025. arXiv: [2503.23886 \[cs.DB\]](https://arxiv.org/abs/2503.23886). URL: <https://arxiv.org/abs/2503.23886>.
- [40] R. Shu, N. Das, M. Yuan, M. Sunkara, and Y. Zhang. *Towards Effective GenAI Multi-Agent Collaboration: Design and Evaluation for Enterprise Applications*. 2024. arXiv: [2412.05449 \[cs.CL\]](https://arxiv.org/abs/2412.05449). URL: <https://arxiv.org/abs/2412.05449>.
- [41] *DuckDuckGoSearch Engine Results API*. URL: <https://serpapi.com/duckduckgo-search-api>.
- [42] S. Wang, S. Zhang, J. Zhang, R. Hu, X. Li, T. Zhang, J. Li, F. Wu, G. Wang, and E. Hovy. *Reinforcement Learning Enhanced LLMs: A Survey*. 2025. arXiv: [2412.10400 \[cs.CL\]](https://arxiv.org/abs/2412.10400). URL: <https://arxiv.org/abs/2412.10400>.
- [43] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown. “Text Classification Algorithms: A Survey”. In: *Information* 10.4 (Apr. 2019), p. 150. ISSN: 2078-2489. DOI: [10.3390/info10040150](https://doi.org/10.3390/info10040150). URL: [http://dx.doi.org/10.3390/info10040150](https://doi.org/10.3390/info10040150).
- [44] Y. Su and Y. Wu. *Robust Detection of LLM-Generated Text: A Comparative Analysis*. 2024. arXiv: [2411.06248 \[cs.CL\]](https://arxiv.org/abs/2411.06248). URL: <https://arxiv.org/abs/2411.06248>.
- [45] K. Kirasich, T. Smith, and B. Sadler. *Random Forest vs Logistic Regression: Binary Classification for Heterogeneous Datasets*. 2018.
- [46] A. Palczewska, J. Palczewski, R. M. Robinson, and D. Neagu. *Interpreting random forest classification models using a feature contribution method*. 2013. arXiv: [1312.1121 \[cs.LG\]](https://arxiv.org/abs/1312.1121). URL: <https://arxiv.org/abs/1312.1121>.
- [47] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *NeurIPS EMC²Workshop*. 2019.
- [48] A. A. Khan, O. Chaudhari, and R. Chandra. *A review of ensemble learning and data augmentation models for class imbalanced problems: combination, implementation and evaluation*. 2023. arXiv: [2304.02858 \[cs.LG\]](https://arxiv.org/abs/2304.02858). URL: <https://arxiv.org/abs/2304.02858>.

- [49] A. Reuther et al. “Interactive supercomputing on 40,000 cores for machine learning and data analysis”. In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE. 2018, pp. 1–6.
- [50] J. Li, Z. Zhao, K. Yee, U. Javaid, and B. Sikdar. *TAEGAN: Generating Synthetic Tabular Data For Data Augmentation*. 2024. arXiv: [2410.01933 \[cs.LG\]](https://arxiv.org/abs/2410.01933). URL: <https://arxiv.org/abs/2410.01933>.
- [51] L. Hansen, N. Seedat, M. van der Schaar, and A. Petrovic. *Reimagining Synthetic Tabular Data Generation through Data-Centric AI: A Comprehensive Benchmark*. 2023. arXiv: [2310.16981 \[cs.LG\]](https://arxiv.org/abs/2310.16981). URL: <https://arxiv.org/abs/2310.16981>.
- [52] J. Ye, Z. Tan, Y. Hu, X. Yang, G. Cheng, and K. Huang. *Disentangling Tabular Data Towards Better One-Class Anomaly Detection*. 2024. arXiv: [2411.07574 \[cs.LG\]](https://arxiv.org/abs/2411.07574). URL: <https://arxiv.org/abs/2411.07574>.