Final Project Report

(CS5800 Algorithms Fall 2023 Semester)

Real-Time Fraud Detection in Financial Transaction Using Cuckoo Filters

Arya Dhorajiya (NUID: 002780002)
Spandan Maaheshwari (NUID: 001525706)
Sudarshan Paranjape (NUID: 002817988)
Sumit Hawal (NUID: 002979933)

Presentation Link: https://www.youtube.com/watch?v=pcWoxIym0is
Github Code: https://github.com/arya1234/Cuckoo_Filter

Project Context

Project context in the view of Arya Dhorajiya:
Embarking on a journey from my undergraduate days to the rigorous LeetCode grind, my fascination with algorithms transcended theoretical puzzles to recognize their pivotal role in our daily lives. Intrigued by the complexities of financial institutions, fueled by a love for math and the allure of colossal numbers, I, a self-proclaimed data science and algorithm enthusiast, proposed a project to my peers. The objective was clear: to go beyond theoretical elegance and make a tangible impact in the financial sector, specifically in fraud detection, through the application of machine learning. The project took an unexpected turn when we delved into the realm of Cuckoo filters, realizing their potential to revolutionize fraud detection by efficiently handling set-membership queries. This technology promised to enhance the storage and search capabilities of both whitelisted and blacklisted transactional behaviors, providing a dynamic augmentation to the existing financial infrastructure. The strategic choice of Cuckoo filters, with their efficiency and memory optimization, highlighted their significance in addressing the real-world challenges of fraud detection. As our project unfolds, it embodies the harmonious blend of passion, technical acumen, and a genuine commitment to reshaping the landscape of financial technology. Incorporating Cuckoo filters isn't just about creating a project; it's about crafting a solution that navigates the intricacies of financial transactions with accuracy and resource efficiency, bridging the gap between algorithmic enthusiasm and practical demands.

Project context in the view of Spandan Maaheshwari:
My journey in the field of electronics has always been marked by a fascination with how complex systems manage and process data with remarkable efficiency. This curiosity has now led me to the exploration of Cuckoo Filters, a sophisticated data structure known for its

dynamic capabilities in handling data sets. Unlike the static nature of Bloom Filters, Cuckoo Filters offer the unique advantage of allowing additions and deletions, making them particularly suited for environments where data is continually changing. This feature of Cuckoo Filters aligns perfectly with my growing interest in hashing and its myriad applications across various computer systems.

Our project aims to leverage the dynamic capabilities of Cuckoo filters for real-time fraud detection in financial transactions. We focus on their ability to quickly adapt to changing fraud patterns, due to their efficient and precise set membership testing. The goal is to develop a prototype that promptly identifies and responds to potential fraud. We will explore the working and application of Cuckoo filters, inspired by their use in tech giants like Meta. Ultimately, this project seeks to innovate in financial security, enhancing the reliability of such systems.

Project context in the view of Sudarshan Paranjape:
Fraud detection has grown in importance across a number of businesses, such as e-commerce, insurance, and banking. The ubiquity of internet banking and online transactions needs increased attention to prevent against fraudulent operations. The main goal of real-time fraud detection is to quickly detect and stop fraudulent transactions as they happen. Using transaction data, an algorithm that efficiently identifies and reduces fraudulent activity is to be developed. Similar to this, we attempt to use Cuckoo Filter to build a Fraud/Unauthorized Transaction detection system. Cuckoo filters can be used to detect fraud by maintaining hash table fingerprints of known fraudulent IDs. A new transaction's fingerprint is generated and compared to the fingerprints stored in the hash table upon receipt. If the fingerprint is found in the hash table, then the transaction is flagged as potentially fraudulent else shown as a whitelist recipient.Eventually, this research aims to innovate in financial security by improving the reliability of such systems.

Project context in the view of Sumit Hawal:
I have always been fascinated by the scale of financial operations. I am amazed by the scale and reliability of services these institutions provide us with. Even thought, my background in finance has been minimal, I have tried to learn it from different books and courses. As someone with a background in computer engineering and passion for data science, I was extremely excited to tackle this interesting problem of fraud detection in financial transactions. In my view it is an extremely challenging but important issue to be addressed, as there are millions of transactions happening at any given instance and not being able to identify these frauds could lead to catastrophic loss to the lives of people. Our plan was to implement a data structure and leverage the capabilities of machine learning to identify anomalies in the financial transactions. The structure would aid the machine learning model in making a decision in somewhat an stacked ensemble way.

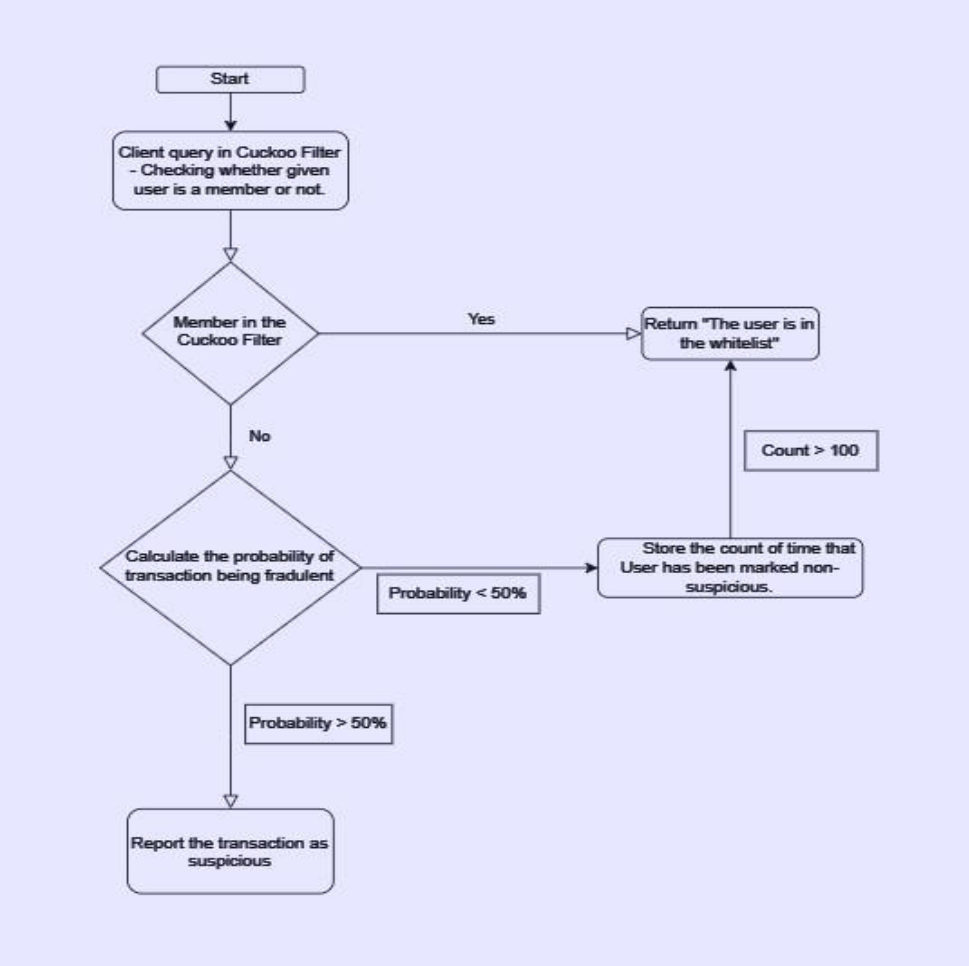
Question

There are millions of transactions happening at any instance in the world of finance, and it is highly likely that there may be small transactional frauds happening in those millions of transactions. According to Statista, the amount of fraudulent payments in United States almost doubled from 2014 to 2021. In total, fraudulent payments reached 32 billion dollars in 2021. It is therefore important not only to identify them, but also do them in real-time. There can be two approaches to this problem, one is rule-based and second is Machine Learning Based. But as we know, rule base can never generalize to a problem well like how a machine learning model can. The issue with ml models is their high dependencies on computational resources. And from this project we aim to check if using a hash-map data structure like 'Cuckoo Filter' can aid our prediction of fraudulent transactions.

Analysis & Overview

Given the time constraints we are trying out are implementation on a small dataset. But the idea can be extrapolated to large scale problems in finance. We use the Cuckoo filter as a preprocessing step before we implement our machine learning model. If the particular transaction is a part of cuckoo filter, then we ignore the transaction. If it is not, then we test our model against the machine learning model to check its probability of being a fraud. If the probability of fraud is greater than 50% then we mark the transaction as fraudulent and if not, we mark it as 'not fraudulent'. Along with that we keep a list of count number of time user was marked non-suspicious and if it is greater than 100, we return the user to 'whitelist'.

The ml model we use here is random forest classifier, due to its ability to model complex data, minimum preprocessing requirements, high accuracy. Below the flowchart of our process and short explanation of the cuckoo filter.



Requirements

- Operation system: Windows/Linux/OS.
- Language: Python 3.x.
- Python Library: Numpy, Pandas, Scikit Learn.

Input

Here is a snapshot of our dataset below,

transaction_id	hour_of_day	category	amount(usd)	merchant	job	First_Last
fff87d4340ef756a592eac652493cf6b	15	misc_pos	194.51	fraud_Towne LLC	Public relations account executive	James_Strickland
d0ad335af432f35578eea01d639b3621	15	health_fitness	52.32	fraud_Friesen Ltd	Retail merchandiser	Cynthia_Davis
87f26e3ea33f4ff4c7a8bad2c7f48686	1	shopping_pos	6.53	fraud_Mohr Inc	Systems developer	Tara_Richards
9c34015321c0fa2ae6fd20f9359d1d3e	20	home	7.33	fraud_Gaylord-Powlowski	Cytogeneticist	Steven_Faulkner
198437c05676f485e9be04449c664475	5	gas_transport	64.29	fraud_Christiansen, Goyette and Schamberger	Product/process development scientist	Kristen_Allen
2203dd06764180b6314dca2b21df2494	10	shopping_pos	129.8	fraud_Schuppe, Nolan and Hoeger	Agricultural consultant	Jeremy_Roberson
7b44d808a40f8ac414736be6a038e83c	9	grocery_pos	111.1	fraud_Kuhic Inc	Broadcast presenter	John_Chandler

We have 7 attributes and 200000 rows. The attributes are:

- transaction_id: Unique identifier to separate each transaction.
- hour_of_day: Time of transaction.
- category: domain of transaction {eg: ‘misc_pos’, ‘shopping’,…}.
- amount(usd): amount involved in transaction.
- merchant: merchant of payment.

- job: type of job.
- First_Last: first and last name of the person, separated by underscore '_'.

Cuckoo Filter

A cuckoo filter is probabilistic data structure which is used to test the set membership. The unique aspect about it is, 'False Positives' are possible but 'False Negatives' are not. In case of transactions that would be, normal transactions can be flagged as fraud, but fraud will never be flagged as normal. A cuckoo filter is also better in terms of deleting existing terms and achieving lower space overhead than bloom filters.

Working

A cuckoo filter uses hash table based on cuckoo hashing to store the cuckoo fingerprints of the items. Cuckoo Hashing is a hashing technique with worst-case constant lookup time. The basic idea of this hashing is to resolve the collisions by using two hash functions instead of 1, providing two possible locations in the hash table for each key. Operations of cuckoo hashing include:

- **Hash Functions Selection:** Choose multiple independent hash functions that are uniform in distribution to minimize the probability of collisions, which is crucial for the efficient operation of Cuckoo filters.

- **Bucket Array Initialization:** Initialize a fixed-size bucket array, where each bucket can hold a small, fixed number of elements. The size and number of buckets are determined based on the expected load factor and the acceptable false positive rate.

- **Element Insertion:** The new item to be inserted, first slot of table T1 is checked. If it is empty, the item is inserted and if it is not, then the placed item is removed and new item is inserted in the place.
  The removed item is then placed in table T2. To avoid infinite iteration of looking for empty space, a max-loop threshold is specified. Sample tables are given below.

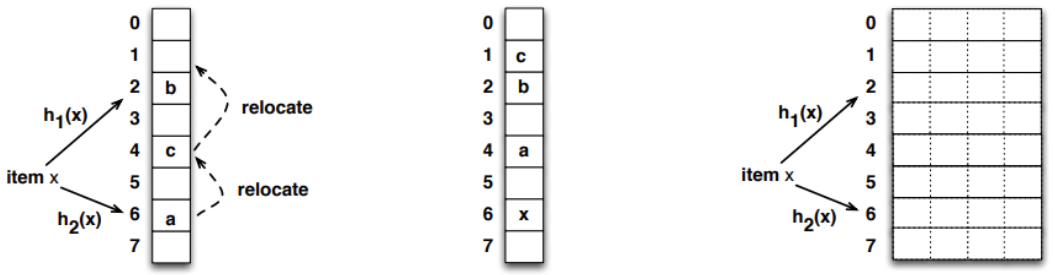
- **Element Lookup:** For lookup operations, apply the hash functions to the username and check the corresponding buckets for its presence. If the username is found in any of the buckets, it is considered present in the filter.
  Checking if the same key is present in both tables T1 and T2.
  Worst case time complexity = O(1).

- **Element Deletion:** Delete a username by hashing it to find its buckets and then removing it from the appropriate bucket. This operation distinguishes Cuckoo Filters from Bloom Filters, as it allows for precise deletion without affecting other usernames. Ignoring the cost of shrinking the table.
  Worst case time complexity = O(1).

- **Count Operations:** Counting Cuckoo filters enhance the basic by incorporating counters within the buckets. Each entry not only stores the username of the item but also includes a small counter that records how many times the username has been inserted.

- **False Positive Handling:** Manage false positives by fine-tuning the number of buckets and the size of each bucket. A lower load factor generally results in a lower false positive rate, but this must be balanced against the space efficiency of the filter.

- **Dynamic Space Re-optimization:** Re-optimizing the space substantially reduce the load factor, maintaining space efficiency and operational performance over time.



(a) before inserting item x (b) after item x inserted (c) A cuckoo filter, two hash per item and functions and four entries per bucket

1. table for h(k)

		Key inserted									
k		20	50	53	75	100	67	105	3	36	39
h(k)		9	6	9	9	1	1	6	3	3	6
Hash table entries	0										
	1					100	67	67	67	67	100
	2										
	3								3	36	36
	4										
	5										
	6		50	50	50	50	50	105	105	105	39
	7										
	8										
	9	20	20	20	75	75	75	53	53	53	75
	10										

2. table for h′(k)

		Key inserted									
k		20	50	53	75	100	67	105	3	36	39
h′(k)		1	4	4	6	9	6	9	0	3	3
Hash table entries	0									3	3
	1				20	20	20	20	20	20	20
	2										
	3										
	4			53	53	53	53	50	50	50	53
	5										
	6							75	75	75	67
	7										
	8										
	9						100	100	100	100	105
	10										

Img src: https://en.wikipedia.org/wiki/Cuckoo_hashing

Conclusion

Successfully implemented Cuckoo filters for fraud detection in financial transactions and took it a step further by integrating them with a random forest classifier, creating a robust two-step mechanism. Throughout our CS5800 class, we drew inspiration from the 'cuckoo filter' data structure and the binary tree-based 'random forest classifier,' conducting a thorough evaluation considering both time and space complexity.

Our learning process highlighted key areas for future improvement. Firstly, we recommend exploring the use of a larger dataset to enhance the effectiveness of the implemented pipeline. Secondly, considering the dynamic nature of financial transactions, incorporating an incoming stream of data could refine our fraud detection system further. Lastly, we propose investigating state-of-the-art hash data structure-based filters, such as 'Ribbon,' to stay abreast of the latest advancements in the field.

Completion Context:

Arya Dhorajiya:

So, after diving into this project—mixing machine learning with Cuckoo filters for sniffing out fraud in financial transactions—I've seen some real magic happen in the security game of financial institutions. The Cuckoo filters turned out to be the secret sauce, cranking up the efficiency and accuracy of our fraud detection system. Now, it's on point, quickly nailing both the good and the shady transaction vibes. This move didn't just sped up what we already had; it added this cool adaptability to keep up with the ever-changing fraud game. The drop in false positives is a game-changer, making our fraud detection way more dependable. My tag team effort, blending algorithm smarts with machine learning mojo, birthed a solution that tackles legit challenges. The result? A slick fraud detection system that's not just advanced and scalable but also marks the win in my mission to shake up the financial tech scene. Plus, let's not forget—the tricks and skills picked up along the way are like gold for what's coming next. They're the solid base for tackling whatever challenges pop up in the ever-evolving world of data science and tech.

Spandan Maaheswari:

In this project, I explored Cuckoo filters for detecting fraud, which was a new and exciting challenge for me. Cuckoo filters are a type of data structure known for their accuracy and efficiency in managing large amounts of information. Our task was to adjust these filters to specifically identify fraudulent activities in vast financial data sets. This required tweaking their design to handle constant updates and ensure they could quickly and accurately flag potential frauds. The main hurdle was balancing the complexity of Cuckoo filters with the need for fast and reliable fraud detection.

The experience was incredibly enriching, both in terms of technical skills and practical understanding. Working with Cuckoo filters enhanced my knowledge of how advanced data

structures can be applied in real-world scenarios, especially in critical areas like cybersecurity. Hashing was something that I always wanted to work on, and this project has served that purpose really well. This project was a major step forward in my journey in computer science, teaching me valuable lessons in handling intricate computational challenges and preparing me for future projects that involve large-scale data and security concerns.

Sudarshan Paranjape:
All of us are very interested when it comes to hashing, as it is used in almost every field of computer applications and Cuckoo filters seem like a very interesting topic for all of us. Cuckoo filter is a data structure designed to tell you, rapidly and memory-efficiently, whether an element is present in a set and update the list if it's not present. Cuckoo filter has many interesting applications and if we can implement it for one field, we can always extend it for other use cases. It is also interesting to note and learn how top MNCs like Meta implement filters in their applications.

The training was extremely beneficial, providing a substantial increase in technical proficiency as well as real-world understanding. My expertise of implementing sophisticated data structures in real-world scenarios has improved as a result of working with Cuckoo filters, especially in crucial fields like privacy and security. I've always wanted to learn more about hashing, and this project has shown to be the perfect way to satisfy that curiosity. It represents an important milestone in my computer science education, teaching me valuable lessons in solving difficult computational problems and preparing me for projects involving large amounts of data and security concerns in the future.

Sumit Hawal:
By implementing this project, I had the opportunity to learn about different hashing techniques currently used like 'Cukoo', 'Bloom', 'Ribbon' filters. Along with data structures I learned out how leverage different techniques within different domains of computer science together to create powerful solutions. Implementation of our technique was also a great experience as I got to learn the challenges faced while implementing an idea. My soft skills were also greatly benefitted from working in a team environment like communication, teamwork, delegating task, etc. My interest in finance and the application of knowledge of computer science to finance was something I really enjoyed. Therefore, this project turned out to be a great learning experience for all of us.

Appendix

Code

```
# Define a function to create and populate a Cuckoo Filter with legitimate users
def Filter(legit_users_list):
  # Define a CuckooFilter class with insert, contains, delete methods
  class CuckooFilter:
    # Constructor to initialize Cuckoo Filter parameters
    def __init__(self, capacity, fingerprint_size, bucket_size):
        self.capacity = capacity
        self.fingerprint_size = fingerprint_size
        self.bucket_size = bucket_size
        self.buckets = [None] * capacity

    # Hash function to generate an index for a given data
    def hash_function(self, data):
        sha256_hash = hashlib.sha256(data.encode()).hexdigest()
        return int(sha256_hash, 16) % self.capacity

    # Extracts a fingerprint from data
    def fingerprint(self, data):
        sha256_hash = hashlib.sha256(data.encode()).hexdigest()
        return int(sha256_hash[:self.fingerprint_size], 16)

     # Insert a fingerprint into the Cuckoo Filter
    def insert(self, data, max_retries=500):
        fp = self.fingerprint(data)
        index1 = self.hash_function(data)
        index2 = (index1 ^ self.hash_function(str(fp))) % self.capacity

        for _ in range(max_retries):
            if self.buckets[index1] is None:
                self.buckets[index1] = [0] * self.bucket_size

            if self.buckets[index2] is None:
                self.buckets[index2] = [0] * self.bucket_size
```

```
            if 0 in self.buckets[index1]:
                self.buckets[index1][self.buckets[index1].index(0)] = fp
                return True
            elif 0 in self.buckets[index2]:
                self.buckets[index2][self.buckets[index2].index(0)] = fp
                return True
            else:
                index = index1 if len(self.buckets[index1]) < len(self.buckets[index2]) else index2
                evicted_fp = self.buckets[index][0]
                self.buckets[index][0] = fp
                fp = evicted_fp
                index1 = self.hash_function(str(evicted_fp))
                index2 = (index1 ^ self.hash_function(str(fp))) % self.capacity

        return False

    # Check if a fingerprint is present in the Cuckoo Filter
    def contains(self, data):
        fp = self.fingerprint(data)
        index1 = self.hash_function(data)
        index2 = (index1 ^ self.hash_function(str(fp))) % self.capacity

        return self.buckets[index1] and fp in self.buckets[index1] or \
               self.buckets[index2] and fp in self.buckets[index2]

    # Delete a fingerprint from the Cuckoo Filter
    def delete(self, data):
        fp = self.fingerprint(data)
        index1 = self.hash_function(data)
        index2 = (index1 ^ self.hash_function(str(fp))) % self.capacity

        if self.buckets[index1] and fp in self.buckets[index1]:
            self.buckets[index1][self.buckets[index1].index(fp)] = 0
            return True
        elif self.buckets[index2] and fp in self.buckets[index2]:
            self.buckets[index2][self.buckets[index2].index(fp)] = 0
            return True

        return False
  # Read legitimate user IDs from a file
  import_file_path = legit_users_list
```

```
    import_file_path = legit_users_list
    with open(import_file_path, 'r') as file:
        top_50_legitimate_users = [line.strip() for line in file]

    # Cuckoo Filter to maintain a whitelist of legitimate users
    cuckoo_filter = CuckooFilter(capacity=1000, fingerprint_size=2,bucket_size=3)

    # Assuming you have a list of legitimate user IDs
    legitimate_users = top_50_legitimate_users
    # Insert legitimate user IDs into the Cuckoo Filter
    for user_id in legitimate_users:
        cuckoo_filter.insert(user_id)

    return cuckoo_filter

def main(csv,user_list):
    cuckoo_filter = Filter(user_list)

    # Read CSV data and preprocess
    X_1 = pd.read_csv(csv)
    X_1.drop(columns=['transaction_id'],inplace=True)
    enc = OrdinalEncoder(dtype=np.int64)
    enc.fit(X_1.loc[:, ['category', 'merchant', 'job']])
    X_1.loc[:, ['category', 'merchant', 'job']] = enc.transform(X_1[['category', 'merchant', 'job']])

    # Function to get transaction data for a user (replace this with your actual function)
    def get_transaction_data_for_user(user_id):
        # Replace this with your logic to fetch transaction data for the given user_id
        # This function should return the relevant features of the transaction
        transaction_data =  (X_1[X_1['First_Last'] == user_id].iloc[0]).copy()
        f = transaction_data.drop(['First_Last'])
        return np.array(f)

    # Set a fraud threshold
    fraud_threshold = 0.5
    i = 0
    #Load a pre-trained Random Forest model
    rf_model = joblib.load('C://Users//smeet//Desktop//files//Algo project//random_forest_model.joblib')

    # Check if a user ID is in the whitelist before making a prediction
    user_to_check = list(set(np.array(X_1['First_Last'])))
```

```
    # Take user input for user_to_check list
    user_input = input("Enter user IDs (comma-separated) to check (leave empty for default list): ")
    if user_input:
        user_to_check = user_input.split(',')
    else:
        user_to_check = list(set(np.array(X_1['First_Last'])))
    for user in user_to_check:
        if cuckoo_filter.contains(user):
            print(f"{user} is in the whitelist")
        else:
            transaction_data = get_transaction_data_for_user(user)
            prediction_proba = rf_model.predict_proba(transaction_data.reshape(1, -1))[:, 1]
            fraud_probability = prediction_proba  # Probability of being fraudulent
            if fraud_probability > fraud_threshold:
                i = i + 1
                print(f"Suspicious activity detected by {user}! Probability of fraud: {fraud_probability}")
                if i > 10:
                    break

import joblib
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
import pandas as pd
import numpy as np
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")

# Import hashlib for hash functions
import hashlib

# Set file paths for user-list and csv_data
user_list = 'C://Users//smeet//Desktop//files//Algo project//top_50_legitimate_users.txt'
csv_data = 'C://Users//smeet//Desktop//files//Algo project//X_1.csv'

#Calling the Main function
if __name__ == '__main__':
    main(csv_data,user_list)
```

Output:

```
C:\Users\Suyash\Desktop>text.py
Enter user IDs (comma-separated) to check (leave empty for default list): Austin_Phillips,Michelle_Gregory,Sandra_Davies,Angela_Hodges
Suspicious activity detected by Austin_Phillips! Probability of fraud: [0.825]
Michelle_Gregory is in the whitelist
Suspicious activity detected by Sandra_Davies! Probability of fraud: [0.99]
Angela_Hodges is in the whitelist

C:\Users\Suyash\Desktop>
C:\Users\Suyash\Desktop>
```

```
C:\Users\Suyash\Desktop>text.py
Enter user IDs (comma-separated) to check (leave empty for default list):
Suspicious activity detected by Misty_Hart! Probability of fraud: [0.985]
Angela_Hodges is in the whitelist
Suspicious activity detected by Austin_Phillips! Probability of fraud: [0.825]
Sally_Moore is in the whitelist
Suspicious activity detected by Kelly_Lawrence! Probability of fraud: [0.97]
Suspicious activity detected by Phyllis_Powell! Probability of fraud: [0.64]
Suspicious activity detected by Colleen_Morris! Probability of fraud: [0.98]
Keith_Sanders is in the whitelist
Suspicious activity detected by Melissa_Rodriguez! Probability of fraud: [0.97]
Suspicious activity detected by Juan_West! Probability of fraud: [0.985]
Tammy_Ayers is in the whitelist
Suspicious activity detected by Sydney_Morales! Probability of fraud: [0.77]
Jennifer_Black is in the whitelist
Suspicious activity detected by Jerry_Kelly! Probability of fraud: [0.99]
Mary_Wall is in the whitelist
Sara_Harris is in the whitelist
Suspicious activity detected by Savannah_Clark! Probability of fraud: [0.885]
Michelle_Gregory is in the whitelist
Lauren_Anderson is in the whitelist
Lauren_Torres is in the whitelist
Suspicious activity detected by Sandra_Davies! Probability of fraud: [0.99]

C:\Users\Suyash\Desktop>
```

References:

1. https://en.wikipedia.org/wiki/Cuckoo_filter
2. https://www.cs.cmu.edu/~dga/papers/cuckoo-conext2014.pdf
3. Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies (CoNEXT '14). Association for Computing Machinery, New York, NY, USA, 75–88.
4. https://doi.org/10.1145/2674005.2674994 5. Peir, Jih-Kwon & Lai, Shih-Chang & Lu, Shih-Lien & Stark, Jared & Lai, Konrad. (2002). Bloom filtering cache misses for accurate data speculation and prefetching.
5. Proceedings of the International Conference on Supercomputing. 189-198. 10.1145/514191.514219. 6. P. K. Sadineni, "Detection of Fraudulent Transactions in Credit Card using Machine Learning Algorithms," 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2020, pp. 659-660, doi: 10.1109/I-SMAC49090.2020.9243545.