University of Zurich[UZH]

Restricted Verifier: Using ML/AI to Limit Data Disclosure

Junxiao Cao
Zurich, Switzerland
Student ID: 22-956-643

Supervisor: Daria Schumm, Weijie Niu, Prof. Dr. Burkhard Stiller,
Prof. Dr. Roger Wattenhofer
Date of Submission: August 29, 2025

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 29.08.2025

Signature of student

Abstract

Im Kontext dezentraler und selbstverwalteter Identitäten wächst die Sorge um die Offenlegung von Daten. Gesetzliche Regelungen und Richtlinien sollen Datenmissbrauch einschränken.

Nutzer, die über Informationen verfügen, haben jedoch keinen Zugriff, um die Datenanfrage des Dienstanbieters zu verifizieren.

Um dieser Herausforderung zu begegnen, schlägt diese Arbeit vor, maschinelles Lernen und künstliche Intelligenz zu nutzen, um Informationsinhabern bei der Entscheidung zu helfen, ob Dienstanbieter nur unbedingt notwendige Daten anfordern.

Konkret entwickelt diese Arbeit einen Prototyp eines eingeschränkten Verifizierers mit zwei Schlüsselkomponenten. Eine davon ist ein On-Chain-Smart Contract zur Erleichterung der Kommunikation zwischen Entitäten. Eine weitere ist ein Off-Chain-Checker, der aus einem maschinellen Lernmodell besteht, um die Notwendigkeit einer Datenanfrage zu ermitteln.

Die Effektivität und Effizienz des Systems werden durch Tests sowohl unüberwachter als auch überwachter Lernmodelle an synthetisch generierten Datensätzen bewertet.

In Leistungstests zeigte der Random-Forest-Algorithmus eine aussergewöhnliche Leistung bei der Erkennung übermässiger Datenanfragen. Darüber hinaus führte der Prototyp des eingeschränkten Verifizierers zu minimalen Latenzen, was auf seine Anwendung in der realen Welt hindeutet.

Abstract

In the context of decentralized identities and self-sovereign identities, concern over data disclosure rises up. Legal regulations and policies are implemented to limit data misuse.

However, users who are information holders do not have access to verify the data request from the service provider.

To address this challenge, this work propose to leverage machine learning and artificial intelligence technology to help information holders decide if service providers only request strictly necessary data.

Specifically, this work develops a restricted verifier prototype with two key components. One is on-chain smart contract to facilitate communication between entities. Another one is off-chain checker which consists of a machine learning model to determine the necessity of a data request.

The system's effectiveness and efficiency are evaluated by testing both unsupervised and supervised learning models on synthetically generated datasets.

In capability and performance tests, the Random Forest algorithm demonstrates exceptional performance in detecting excessive data requests. Furthermore, the restricted verifier prototype introduced minimal latency, indicating its application in the real world.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Daria Schumm and Weijie Niu. Without their guidance and mentorship, it is not possible to complete this work.

I would also like to acknowledge UZH Communication Systems Group and ETH Zürich DISCO group, who provide support and resources.

Finally, I wish to acknowledge my grandmother Xiuying Liang, my mother, Rulan Chen, and my fiancé, Jingyu Wang. Their support was a constant source of strength during the past three years, especially while I was thousands of miles away from home. I am deeply grateful for their encouragement.

After two decades of study, I finally will end this chapter. I will carry the confidence and resilience I have gained from this journey into the future.

Contents

Chapter 1

Introduction

1.1 Motivation

Governments and institutions are increasingly becoming digital. As a result, individuals who are unwilling to provide certain data to service providers are increasingly excluded from various services. For example, when an individual like Tom wishes to book a flight ticket, airline companies often require mandatory personal information from him, such as his gender, during the booking process. However, Tom might have privacy concerns and prefer not to disclose his gender to the flight company. In the current system, this information is typically a mandatory field during the registration or booking process. Due to this requirement, if Tom chooses to withhold his gender, he is unable to complete the transaction, thereby he is excluded from traveling by air. Another example is the mandatory requirement of displaying COVID-19 vaccination certificates when people travel during pandemic time. Those who refuse to disclose this information have no choice but to give up their travel plan [26].

Governments have led initiatives in Decentralized Identity (DI) and Self-Sovereign Identity (SSI) systems in recent years, which introduced complex challenges, particularly concerning the potential imposition of such systems on individuals. For example, the European Union published the eIDAS 2.0 regulation which introduced a European Digital Identity Wallet, enabling citizens to store and control their identity credentials securely and use them across member states of the European Union (EU) [8]. There is a concern that service providers might request more personal information than is strictly necessary. Without strong enforcement of data minimization principles, users could be pressured to disclose more data than required for a service. People who opt not to participate in the system because of the unwillingness of sharing unnecessary data are completely excluded from the service. Although data privacy policies such as the General Data Protection Regulation (GDPR) mandate that service providers should not require excessive data, users do not have strategies to verify data requests [9].

The reality is that individuals and service providers are not in an equitable position because service providers are the ones who actually control the data collection process. In the absence of solutions that empower users to verify if excessive data is requested from

the service providers, service providers still have access to request extensive personal information for other purposes, sometimes exceeding what is strictly relevant to the service. For example, in the famous Facebook-Cambridge Analytica data scandal, Cambridge Analytica collected millions of personal data from Facebook users for political advertising [18]. To limit data misuse, data minimization is one of the fundamental principles in DI/SSI frameworks. Service providers should not require more data than the service needs. Requesting unnecessary data may trigger various privacy issues. However, users are often unable to recognize or address when such a request is made.

This dynamic creates a power imbalance, effectively compelling users to consent to excessive data disclosure or face exclusion. While this issue may be somewhat mitigated in competitive sectors where users can choose from multiple service providers, such flexibility is often absent in contexts like government services or international mobility, where options are typically limited or nonexistent.

1.2 Thesis Goals

The primary goal of this thesis is to research how machine learning techniques and artificial intelligence solutions can be employed to achieve data minimization and limit data disclosure in DI and SSI systems. The focus is on defining and implementing the verifier role within DI and SSI frameworks. Research begins with establishing a foundational understanding of DI and SSI systems and their key components, such as issuers, holders, and verifiers. Building on this foundation, the thesis examines the verifier's role and authority in decentralized identity and self-sovereign identity systems. A comprehensive literature review is conducted to learn the essentials of the research.

The second goal is to explore the application of machine learning models to limit data disclosure in DI and SSI systems. This is achieved by developing a prototype that detects excessive data disclosure. This goal is broken down into three key subgoals. First, several use cases, including flight ticket purchase, student information management, restaurant reservation, and other general scenarios, are identified to provide concrete contexts for data collection and model training. Second, a generated synthetic dataset is used as input to train a suitable machine learning model. This dataset includes a labeling scheme for supervised learning and predefined criteria to address the inherent subjectivity of defining data necessity. The size of the dataset is determined by the amount needed to achieve sufficient model accuracy while remaining within computational limitations. Finally, suitable ML models are selected and trained based on their accuracy and efficiency in detecting unnecessary data requests. These models are evaluated to ensure they can effectively meet the data minimization requirements of the chosen use cases.

After model training and evaluation, the third goal is to design a verifier prototype that integrates the machine learning model into an existing decentralized identity system. This prototype demonstrates how data requests can be automatically checked if they meet the data minimization requirement. Finally, this thesis evaluates the restricted verifier from both a technical and a practical perspective, evaluating its feasibility, effectiveness, and alignment with the principles of decentralized identity and self-sovereign identity systems.

1.3 Methodology

This section details the methodology used in literature review, data generation, model training, and integration of the model into a decentralized identity system.

1.3.1 Literature review

To lay a solid theoretical foundation for this research, a comprehensive literature review is conducted to understand the whole landscape of decentralized identity frameworks, which is crucial for exploring various architectures and core concepts of the decentralized identity framework. We focus on SSI systems, especially their core concepts such as access control, data ownership, and privacy-preserving mechanisms. A critical part of this review is understanding the unique roles and authority of verifiers within DI/SSI systems. The review also concentrates on the intersection of identity systems with machine learning and artificial intelligence. This component of the review is crucial for understanding how artificial intelligence and machine learning help to detect unnecessary data request and sensitive information in different scenarios. This provides the conceptual framework for selecting and integrating machine learning models into decentralized identity infrastructure.

Following the PRISMA methodology, the literature review is conducted to identify, evaluate, and summarize academic and technical works related to DI/SSI frameworks and their intersection with ML and AI [28].

In the identification stage, we search major digital databases including IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, and Google Scholar using combinations of keywords such as ”decentralized identity”, ”self-sovereign identity”, ”verifiable credentials”, ”data minimization”, ”data disclosure”, ”access control”, ”zero-knowledge proofs”, ”machine learning in decentralized identity systems” and ”machine learning in self-sovereign identity systems”. The search is limited to publications between 2000 and 2025, and only English-language sources are considered.

After removing duplicates and irrelevant papers, we exclude publications that focus solely on centralized identity management or unrelated blockchain topics. We assess the remaining articles against the eligibility criteria: relevance to DI/SSI architectures, discussion of privacy-preserving mechanisms, and involvement of machine learning models in DI/SSI identity systems.

A total of 41 publications are selected. The final review focuses on (i) architectural models of decentralized identity systems, (ii) self-sovereign identity concepts such as data ownership and access control, (iii) the role of verifiers in DI/SSI frameworks, and (iv) the integration of AI/ML techniques to enhance data privacy, particularly in detecting unnecessary data requests.

This structured review provides the theoretical basis for selecting and integrating machine learning models into the DI/SSI infrastructure.

1.3.2 Data generation

Due to the absence of a publicly available dataset suitable for training and evaluating models that detect excessive data requests, we have to generate our own synthetic dataset. This dataset is used as the crucial input for the machine learning models. To ensure the generated data is representative and high-quality, it needs to simulate real-world data requests, including various fields and contexts.

For this purpose, we use the Synthetic Data Vault (SDV) library to generate synthetic data [29]. SDV is chosen for its outstanding capacity to produce high-quality, statistically comparable data with both categorical and multi-type fields, which is essential for simulating the complexity of data requests in real-world DI/SSI systems. We first define a schema to represent typical data requests issued by service providers. Each synthetic sample simulates a data request and includes fields such as name, gender, nationality, age, marital status, and purpose of service. These attributes are selected based on a review of the actual data fields requested in DI/SSI systems documented in academic papers and from real-life scenarios.

Multiple synthetic datasets are generated to reflect variations in service types, including flight ticket purchase, student information management, and restaurant reservation.

This unlabeled dataset is used to train our unsupervised learning models in order to detect abnormal data requests. The GaussianCopula model is selected based on its superior performance in modeling high-cardinality categorical fields.

To create more complex synthetic samples, we used a large language model (LLM). We choose Gemini, which is developed by Google for its strong performance in natural language reasoning and structured data synthesis [14]. Compared to rule-based or statistical generation tools, LLMs offer the advantage of producing semantically rich and varied samples, which closely resemble real-world data requests.

Each synthetic sample generated by Gemini simulates a data request, including fields such as name, gender, nationality, age, marital status, and purpose of service. To enable supervised machine learning, it is essential to create labeling rules to identify whether a data request is excessive for the selected use case. Consequently, we ask Gemini to provide a binary label indicating whether the data request is appropriate (minimal) or inappropriate (excessive). The size of the dataset is determined by the necessary amount of data needed to reach adequate model accuracy within the constraints of computational resources.

Gemini is chosen after a preliminary comparison with other publicly available LLMs (e.g., GPT-4, BERT, and Llama) in the early stage of this work. Compared with open-source models, Gemini is more precise in labeling excessive requests after manual verification. Compared to other business models, Gemini is more cost-effective.

This approach produced a labeled dataset suitable for training machine learning models to detect violations of data minimization in DI/SSI systems.

1.3.3 Model selection and training

To develop an effective checker for detecting unnecessary data requests in DI/SSI systems, we evaluate multiple unsupervised and supervised learning models. The selection criteria focus on the models' ability to handle high-dimensional data and detect outliers.

For unsupervised learning, we select Isolation Forest and Autoencoder. These models are chosen for their strong performance in efficiency in preliminary experiments. Isolation Forest, in particular, excels in identifying anomalies based on isolation properties without assumptions about the data distribution, while Autoencoders capture reconstruction errors as a proxy for detecting atypical data request patterns.

For supervised learning, we train and compare Random Forest, Logistic Regression, and Decision Tree classifiers. These are selected for their ability and effectiveness in binary classification and anomaly detection tasks.

All models are trained on the synthetic dataset described previously, using an 80/20 train-validation split. The feature vectors include binary indicators for each requested data field (e.g. name, age, nationality), along with contextual information such as the purpose of the service. For supervised learning models, the ground truth labels are generated using Gemini, which indicate whether a given data request was compliant with data minimization principles.

Instead of building every model from scratch, we import the model from standard Python libraries such as sklearn, since the goal of this work is not to develop novel machine learning algorithms.

To comprehensively assess model performance, we use the following metrics. We use Precision-Recall Curve and F1-Score to evaluate performance on potentially imbalanced dataset (e.g., where there are relatively few excessive data requests compared to normal data requests). The AUC-ROC curve is used to measure the trade-off between true positive and false positive rates. We also use accuracy to generally benchmark across all the models.

These metrics allow us to compare models not only in terms of overall accuracy, but also with regard to their ability to correctly detect excessive data requests, which is the primary goal of this work.

1.3.4 Integration

Integration is a crucial validation step. It allows the prototype to be tested in a real and operational environment, which helps us uncover potential challenges. This process validates the prototype as a genuine solution, not just a theoretical concept. Integrating into an existing SSI system also provides a strong proof of concept, demonstrating the functionality and performance of the machine learning-based checker.

CredChain is an ideal platform for integration, because its design principles align with the needs of this work. CredChain is built on the Ethereum blockchain, which provides the

decentralized and immutable ledger required for the on-chain components of the prototype [6]. The restricted verifier prototype is integrated into the CredChain infrastructure via a hybrid deployment. We deploy both on-chain and off-chain components to achieve better performance. The core verification logic is implemented as a smart contract deployed on the blockchain, while the synthetic data generator and machine learning model are hosted on an off-chain Flask-based web application to avoid huge latency on the blockchain.

This integration strategy ensures a balance between blockchain transparency and computational efficiency. This hybrid deployment makes restricted verifier well-suited for DI/SSI frameworks where privacy, verifiability, and performance are critical.

1.4 Thesis Outline

This thesis consists of 6 chapters, including an introduction, fundamentals of the work, architecture design, implementation, evaluation and the conclusion.

Chapter 1 begins with an introduction and motivation that the growing concerns about data disclosure in DI/SSI system are examined. This section also outlines the main research tasks and objectives.

Chapter 2 discusses about the background and related work, which provide a detailed overview of decentralized identity systems and the principles that are needed in this work. This helps readers understand the fundamentals of this work and also informs readers of the development of this area based on related works.

Chapter 3 talks about the conceptual framework of the restricted verifier. This is to help the reader grasp the entire architecture of the prototype.

Chapter 4 is the implementation chapter, which shows the details of the restricted verifier, including the dataset generation, model selection and challenges encountered during the experiments. The behind-the-scenes reasons of the steps is also discussed in detail.

Chapter 5 focuses on evaluating and validating the prototype and assessing its performance and effectiveness.

Chapter 6 concludes with a critical discussion of the advantages and disadvantages of the proposed solution. We also discuss about future work, including potential improvements to the model and broader applications of the restricted verifier in decentralized identity systems.

1.5 Contribution

Our work introduces a novel user-centric verifier that empowers the user to evaluate data requests, which is previously unavailable in DI and SSI systems. This approach directly addresses the problem of excessive data disclosure by complementing existing technologies

with a blockchain-based, restricted verifier. In contrast to previous systems where users could not verify if data request is necessary for its declared purpose, this prototype sends the request to a local checker. This machine learning-based classifier helps users identify unnecessary data requests, giving them the control to make an informed decision about data sharing.

Chapter 2

Fundamentals

2.1 Background

In modern society, no one is exempt from the concern of data privacy. Various international organizations and national governments have introduced the corresponding laws and regulations to regulate data privacy. For example, the European Union (EU) published the General Data Protection Regulation (GDPR) in 2016 focusing on regulating data collection, data transfer, and all forms of actions related to the data subject [9]. GDPR also becomes the basis for other similar laws or acts in other countries and regions. For example, the UK published the UK GDPR which is identical to the GDPR [17]. The California Consumer Privacy Act (CCPA) that came into effect in 2018 also shares many similarities with GDPR. In the United States, there are several acts to prevent data misuse [37]. The Personal Information Protection Law (PIPL) came into effect in China in 2021 aiming at protecting personal data [30]. When we compare all these privacy laws, it is easy to extract the core concepts of data privacy protection. These principles include fairness, transparency, and data minimization [5]. Data minimization means that data collectors can only require minimal data to provide the service and no other data should be requested from the user [3]. These laws and regulations show that more and more governments are paying attention to data disclosure.

2.1.1 Decentralized Identities

Decentralized identity(DI) and self-sovereign identity(SSI) represent the emerging trend in identity management that is shifting from centralized authorities to user-centric [32]. There are four key components in the self-sovereign identity architecture [24]. The first is decentralized identifiers(DIDs). The identifier is unique and is linked to the specific user for identification. The common way to generate such identifiers is asymmetric cryptography. By establishing the public and private key pairs, anyone on the blockchain can verify the identity of a specific user without relying on any external entities. The second component is the authentication. Authentication in the self-sovereign identity system is achieved through cryptographic methods, with which only authorized entities can access

or share the data. In this case, access control is achieved. The third part is verifiable claims(VCs), which is the core of self-sovereign identity. A claim is a statement about a specific subject. The verifiable claim is signed by the issuer. Users store such claims and present them to other parties who send requests without disclosing other personal information. The fourth component is storage. There are two ways to store verifiable claims. The most common way is off-chain storage, which is also known as private storage. Users can keep the data in their own storage. Another way is public storage, which means that the data is stored on-chain. For example, public keys are usually stored in public storage for the convenience of communication. In parallel, blockchain technology is widely used in decentralized identity systems and self-sovereign systems. The blockchain is a growing list of records known as blocks. Each block consists of the cryptographic hash of the previous block, a timestamp, and transaction data. The core part of the blockchain are the rules that are known as smart contracts. Smart contracts can be used to control the ownership of properties and make blockchain more applicable compared with other technologies [27]. The main challenges of using blockchain technologies to replace the current centralized database include cyber security threats such as adversarial machine learning and IoT vulnerabilities, data privacy concerns such as GDPR compliance and user consent, and technical issues such as scalability [39].

2.1.2 Data minimization

In recent years, data minimization has emerged as a critical principle in the design of privacy-preserving systems, particularly in decentralized identity systems. There are arguments that self-sovereign systems cannot let users fully control their data without addressing privacy at the network level [35]. However, other people believe SSI is feasible despite network concerns. Stokkink worked with the Dutch government to develop TCID, which satisfies functional requirements to guarantee the desirable system properties. They show that despite the latency caused by network-level anonymization, TCID is still applicable in real-life situations. Although DI and SSI are promising in protection data privacy, there are still challenges to apply them to real life. Giannopoulou pointed out that there are three challenges in data compliance when applying decentralized identity and self-sovereign identity systems [12]. First, there is ambiguity in roles and responsibilities. DI and SSI systems blur the traditional role in data management system. Secondly, although self-sovereign identity promotes user control over personal data, ensuring only necessary data is collected remains a huge challenge. Thirdly, one of the most critical feature of blockchain system is immutability which makes erasing after using not possible. Considering the challenge of data minimization in SSI, a growing trend of research addresses this challenge from various perspectives, including software engineering, machine learning, algorithmic fairness, and distributed learning. Senarath et al. proposed a data minimization model aimed at integrating privacy considerations into the software development life cycle. Their model helps developers understand data from a user-centric perspective and redesign data collection strategies accordingly [33]. Mukta conducted a survey on data minimization techniques in blockchain-based healthcare systems. They present a comparative analysis on the privacy properties of various methods. The first solution is data masking, which focuses on minimizing collection. During the collection stage, the service provider is responsible for collecting the data legitimately.

The decision to collect minimal patient data can be made at design time. For example, when collecting personal information, the service provider may make some data entry fields optional, giving the patient the choice not to expose some data to the system. The second solution is access delegation aiming at minimizing purpose. During the data usage phase, it is essential to ensure that the data is used strictly in accordance with the purpose for which it was originally collected. If secondary use, such as forwarding to third parties is intended, additional consent should be obtained from the user. However, in practice, the consent request remains static. Users are typically asked to consent prior to data collection, covering all anticipated purposes, with little to no follow-up. The third solution is access control, which focuses on minimizing storage and access. In the storage stage, service providers typically apply access control measures to limit who can access sensitive data. The key concern is data retention. Ideally, private data should be deleted at the point the service terminates. However, removing data from a blockchain-based system is challenging due to its inherent immutability. They offer a unique view of data minimization from both data holders and issuers. From the data owners' perspective, there are three solutions. The first solution is selective disclosure, which means minimizing data sharing. Users can decide what data they would like to share. Consent management and access control should also be owner-centric. The second is anonymization. This is to modify personal data in order to make it impossible to identify the data owner. The third solution is consent management. Data owners have full control over their data. Without the consent of data owner, no one can access the data. [25].

2.1.2.1 Federated learning

Recent works in federated learning (FL) has also contributed significantly to the practice of data minimization. By design, federated learning enables the training of decentralized models without transferring raw data to central servers [22]. For example, Bonawitz et al. designed an efficient and robust protocol to securely aggregate high-dimensional data. Their protocol allows a server to compute large-scale data vectors from endpoints in a secure manner without knowing the contribution of a specific user [4].

2.1.3 Machine Learning

Machine learning is a popular solution for various tasks like classification, recognition, and prediction. Whether excessive data are requested is a classic binary classification task.

Machine learning has become a widely adopted approach in various tasks, including classification, regression and pattern recognition. Its ability to learn from large datasets makes it particularly suitable for problems where rule-based or manual systems fall short.

In the context of data privacy, data minimization can be converted into a classification task that determining whether excessive data is requested. This can be framed as a binary classification problem, where the system must decide whether a given request for personal data is aligned with the principle of data minimization or whether it goes beyond what is necessary for the stated purpose.

2.1.3.1 Machine learning approaches

Machine learning approaches can be divided into three major categories, supervised learning, unsupervised learning, and semi-supervised learning. Supervised learning means that training data is labeled. Every input sample is matched with an output label. This direct association between input and output lets the model learn the pattern and be able to make predictions. However, if labeled data is not available, supervised learning is not feasible. In this case, unsupervised learning focuses on uncovering the inner logic and hidden patterns in the training dataset. Unsupervised learning strategies, such as clustering techniques and anomaly detection, could be used to identify unusual data. Unlike unsupervised learning and supervised learning, semi-supervised learning combines labeled and unlabeled data to train models. It leverages the strengths of both supervised and unsupervised learning, addressing the challenge of needing large amounts of labeled data, which can be expensive or time-consuming to acquire. Semi-supervising learning model is firstly trained on labeled dataset. Then it uses its learned patterns from the labeled data to predict labels for the unlabeled data, creating pseudo-labels. The pseudo-labeled data is then combined with the original labeled data. The model is retrained on this larger, enriched dataset. This process helps the model to improve overall performance.

2.1.3.2 Machine learning models

For classification tasks, we consider both unsupervised and supervised learning models.

For unsupervised learning, the most common strategy is clustering. For example, the famous K-Means algorithm. These algorithms group similar data points together to form a cluster. If a data point is too far away from the center of the cluster, it is an outlier. For anomaly detection, the autoencoder is designed to learn efficient compressed representations of input unlabeled data. Autoencoder consists of two main components, encoder and decoder. The encoder is used to compress the input into lower dimensions, while the decoder reconstructs the original input from the code. The model is trained to minimize the difference between the original input and its reconstruction. Autoencoders are especially useful for tasks like dimension reduction, feature learning, and anomaly detection. In anomaly detection, for example, the model is trained with data that are regarded normal. When the input is an anomaly sample, the reconstruction will be far from the original input, which is a flag for outliers. Apart from Autoencoder, one-class SVM(support vector machine) and isolation forest also have great performance in anomaly detection. They have similar learning strategy as autoencoder. They also learn from the normal data pattern and identify deviations as outliers. One-class SVM unlike autoencoders focuses on detect outliers and novelties within the dataset rather than assuming tasks like binary classification. Compared with Autoencoder which is a deep learning model, isolation forest is renowned for efficacy and efficiency. Isolation forest relies on recursive partitioning to detect anomalies.

For supervised learning, there are some popular models which are both efficient and effective in classification tasks. Logistic regression is widely used for binary classification tasks. Its main strengths are simplicity, efficiency, and easy to interpret. A decision tree is

a tree-like model which leverage partitioning to detect anomalies. Decision tree is robust to outliers and easy to visualize. Random forest consists of multiple decision trees. As a result, random forest overcomes the problem of decision tree such as overfitting. Random forest performs well in both balanced and unbalanced datasets.

2.1.3.3 Machine learning metrics

To evaluate the performance of machine learning models, we use various metrics, including F1 score, accuracy, AUC-ROC curve and precision-recall curve. In the following paragraphs, TP stands for true positive, TN stands for true negatives, FP stands for false positive, and FN stands for false negative. True positives are instances that are actually positive and are correctly predicted as positive by the model. True negatives are instances that are actually negative and are correctly predicted as negative by the model. False positives are instances that were actually negative, but were incorrectly predicted as positive by the model. This is also known as a Type I error. False negatives are instances that were actually positive, but were incorrectly predicted as negative by the model. This is also known as a Type II error.

Accuracy is the most straightforward metric for measuring machine learning models. Accuracy quantifies the proportion of correctly predicted instances that includes both positive and negative out of the total number of instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Although accuracy is straightforward, it can be misleading when the dataset is highly imbalanced. For example, the dataset consists of 95% of positive instances and 5% negative instances. The model is set to predict everything as positive. As a result, it achieves 95% accuracy which is not the case.

Precision, which is also known as Positive Predictive Value, measures the proportion of true positive among all instances predicted as positive. It answers the question: "Among all the instances the model predicted as positive, how much percentage were actually positive?"

$$Precision = \frac{TP}{TP + FP}$$

Recall which is also known as Sensitivity or True Positive Rate, measures the proportion of true positive predictions among all actual positive instances. It answers the question: "Of all the actual positive instances, how many did the model correctly identify?"

$$Recall = \frac{TP}{TP + FN}$$

F1 score is harmonic mean of precision and recall. F1 score shows the balance between precision and recall. It is high only if both are high.

$$Recall = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

ROC (Receiver Operating Characteristic) curve plots the True Positive Rate (Recall) against the False Positive Rate (FPR) at different classification thresholds.

AUC (Area Under the Curve) quantifies the overall ability of the model to discriminate between the positive and negative classes, across all thresholds.

True Positive Rate (TPR) = Recall

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (FPR)

$$FPR = \frac{FP}{FP + TN}$$

The ROC curve is made by varying the classification threshold from 0 to 1, and computing TPR and FPR at each point. The AUC is the area under the ROC curve, ranging from 0 to 1. 1.0 means perfect classifier. 0.5 means random guessing. Smaller than 0.5 means worse than random guessing.

2.2 Related Work

Recent research has explore the possibility of achieve data minimization goal in DI and SSI systems. Researchers focus on improving data minimization through decentralized data architectures. Battiston created an alternative solution to the standard cloud-centralized data architecture [2]. Specifically, instead of depending on centralized storage, they aim to put data under the control of the individual owners in decentralized personal data stores. Their primary goal is to improve data minimization. For example, they tried to keep more sensitive data under the control of users while not interrupting the normal function of the application. To achieve this goal, they added aggregating views over the decentralized data to the centralized part of the schema. They designed an extensive SQL language to provide privacy-preserving incremental view maintenance. Some studies aimed at leveraging decentralized identity-Based blockchain solution to prevent unauthorized data usage. Kang et al. design a new architecture that covers essential roles in the data-sharing ecosystem: the issuer of personal data, the individual holder of the personal data, a trusted data storage manager, a trusted license distributor, and the data consumer. The proof-of-concept implementation is based on the decentralized identity framework being developed by the Hyperledger Indy/Aries project [21]. Zyskind et al. introduced a novel protocol which can turn a blockchain system into an automated access-control manager.

In this scenario, third-parties are not needed to perform authentication and access control. They combine on-chain and off-chain storage to construct a privacy-centralized personal data management platform. The framework focuses on ensuring that users own and control their personal data while the service provides are only guests with delegated permissions [40].

Another key aspect of decentralized identity is access management. In order to enable individuals decide who can access their personal data and to let individuals to make use of their data for their own purposes, Hardjono proposed that a federated authorization architecture is required as a fundamental component of the decentralized identity solution. This federated component allow users to manage data access policies including granting access an revoking access to the distributed data repositories. This work showed the user managed access architecture and protocols that build the foundation of scalable federated authorization [16].

Recent works have also focused on developing new identity frameworks. Samunnisa et al. showed that traditional centralized identity systems have vulnerabilities that result in data breaches and lack of user control. Their study aims to develop and evaluate a decentralized identity management framework based on blockchain technology to enhance security, privacy, and efficiency in digital transactions. They proposed an integrated system that contains Decentralized Identifiers (DIDs), Verifiable Credentials (VCs), and smart contract verification. They also introduce a trust registry model to attest the creditability of the issuer. The system was tested with a Kaggle-based decentralized identity dataset in a simulated blockchain environment [31]. Similarly, NEXTLEAP proposes a new decentralized architecture which enhances data privacy. NEXTLEAP builds a federated identity systems. NEXTLEAP decentralized the traditional identities to secure both message and metadata [15].

Researchers also explore privacy-preserving way for authentication. Sucasas et al. propose a a pseudonym-based signature scheme that provides verifiable delegation. This scheme enables users to share data attributes according to the policies of the service providers but remain anonymous [36]. Gilani et al. pointed out that the self-sovereign identity concept gives users access to own and gain insights of their digital identities. They provided a comprehensive overview of the core concepts of self-sovereign identity, including the components of identity proofing and authentication solutions for different self-sovereign identity solutions [13].

In addition to selective disclosure, other privacy-enhancing technologies are being explored. Babel et al. pointed out that DI/SSI aims to offer a solution to manage their own credentials. However, when presented to a relying party, excessive data is revealed to the verifier and the activities of the end users are tracked by the relying party. Several academic works and practical solutions leverage zero-knowledge proofs (ZKPs) to reduce or avoid such excessive information disclosure. Strategies vary from simple selective disclosure to data-minimizing anonymous credentials. They demonstrated that the current SSI solutions built with anonymous credentials still lack critical features including revocation, certificate chaining, and security integration. They argued that general-purpose ZKPs can make security components possible in digital identity systems and prevent MitM attacks [1].

In the field of AI, researches are conducted on leveraging machine learning to limit data disclosure. Ganesh et al. proposed a data minimization framework for machine learning that meets the requirements of legal regulations by limiting the use of unnecessary data and purpose in the training pipeline [11]. To achieve the data minimization goal, it is important to identify and mitigate algorithmic biases in the recommendation system that could lead to discrimination. Clavell and her collaborators presented an algorithmic audit of a commercial application to demonstrate the feasibility of delivering a service without collecting sensitive data like gender, age, race, religion, or other protected attributes of users [10]. Staab et al. proposed a vertical data minimization workflow based on data generalization, ensuring that no full-resolution client data is exposed during training and deployment of models, in which case user privacy is under the umbrella during cyber attacks or data breaches [34].

When doing analysis, researchers may need to access microdata collected by organizations, while these microdate may contain personal information. To avoid the potential privacy leak, data centers usually only allow analysis to be completed using the software from the controller and the controller checks the outputs of the analyses before returning the outputs. Manual checking is time-consuming and expensive. Domingo-Ferrer et al. explored the use of machine learning techniques to partially automate output checking. They used rule-based approach and proved that machine learning models can be employed in safe access centers and decentralized data storage [7]. Jones et al. discusses about the ethnics of consenting in AI which fills the gap of leveraging AI in preserving privacy may trigger privacy concerns [19]. Although machine learning models do not have human involved, they still can have bias because of the training data they are fed. Veale et al. present that trusted third parties could selectively store data which is strictly necessary for discriminating and integrating fairness constraints when building the model in a privacy-preserving manner [38].

Chapter 3

Design

3.1 Prototype architecture

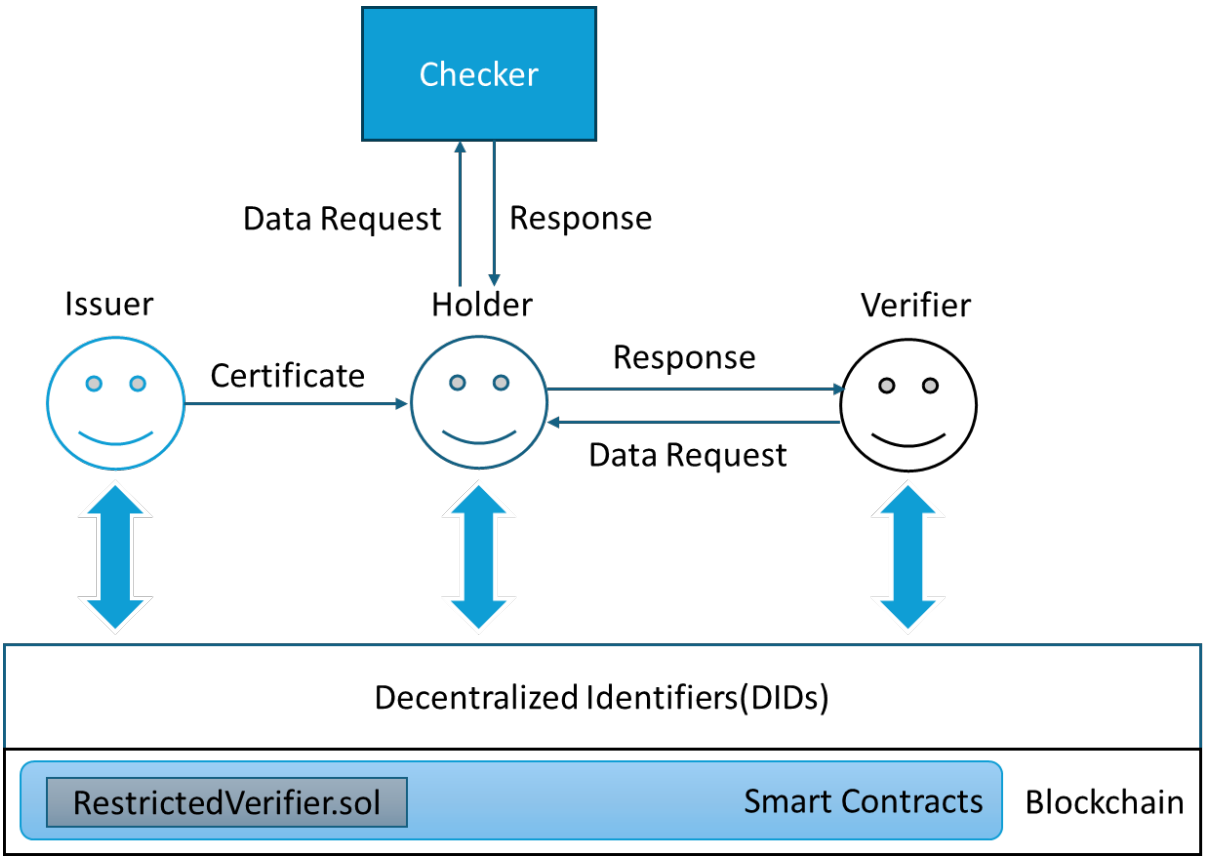

Figure 3.1: Architecture

The restricted verifier prototype consists of 5 major components: issuer, holder, verifier, checker and blockchain. As Figure 3.1 shows, the issuer issues the certificates to the holder. Issuers are usually governments or organizations that issues identity certificates to the holders. Holders are users who hold their personal information certificates or credentials given by the issuer. The verifiers are service providers who send data requests to

the holder to obtain the necessary information to provide certain services. The credentials issued by the issuers are usually encoded by cryptographic algorithms rather than plain-text for security reasons[23]. The restricted verifier prototype can be divided into two parts, on-chain and off-chain. The core of the on-chain part is the smart contract, which performs verification rules of handling the data request. The checker is a machine learning-based tool designed to analyze the data request and make a decision that if excessive data is requested in the data request. The machine learning model is embedded in the checker.

The checker component of this system is designed to belong to the holder (the user), allowing them to verify data requests from a verifier (the service provider) with the help of a machine learning-based automated tool. The holder and the checker communicate through a Web3 instance that connects to a local Ethereum node.

This checker is intentionally not deployed on the blockchain to prevent potential interruptions to normal network communication. The decision to keep the checker off-chain is based on the significant computational resources that machine learning models consume, which is highly inefficient and costly for on-chain execution [20]. For instance, the checker needs to perform several computationally intensive tasks, such as converting text-based inputs into numerical vectors and preprocessing the vectors for the models, followed by the actual prediction process. If these tasks are executed on the blockchain, the massive latency may interfere with the core functionality of blockchain. In contrast, computational resources are not a constraint when the checker is deployed off-chain.

To facilitate communication, the verifier sends a data request to the holder by emitting an event on the blockchain. Similarly, the holder's response to the verifier is also transmitted via an event emission. This event-driven communication model ensures that the interactions remain transparent and secure on the blockchain, while the heavy computational work is handled efficiently off-chain.

3.1.1 Data structure

The data structures are written with Solidity language which is a statically-typed curly-braces programming language designed for developing smart contracts that run on Ethereum.

```
event DataRequest(
    address indexed verifier,
    address indexed holder,
    string fields,
    string purpose,
    uint256 requestID
);
```

Listing 3.1: DataRequest

```
event DataRequestResponse(
    address indexed verifier,
    address indexed holder,
    bool approved,
    uint256 requestID
```

```
6 );
```

Listing 3.2: DataRequestResponse

```
1 uint256 private requestCounter;
```

Listing 3.3: RequestCounter

```
1 event VerificationRequest(
2     address verifier,
3     address holder,
4     string fields,
5     string purpose,
6     bool responded
7 );
```

Listing 3.4: VerificationRequest

From Listing 3.1, we can see that a data request consists of five components, the address of verifier, the address of holder, data fields requested, purpose of the request, and request ID. The data structure of `DataRequestResponse` is showed in Listing 3.2. Accordingly, `DataRequestResponse` contains the address of verifier, the address of holder, decision and request ID. The address of holder and verifier is an address variable which is a Solidity data type representing an Ethereum address.

Another variable prensented in both `DataRequest` and `DataRequestResponse` is `uint256 requestID`. `requestID` is a unique identifier for `DataRequest`. Each time `funchtion requestData` is called, `function requestCounter` is incremented, and the new value becomes the `requestID`. This ensures every request has a unique identifier. This helps off-chain systems like `checker`, track which request triggered the event. Here, we choose `uint256` rather than smaller data types like `uint32` or `uint8` for two reasons. On the one hand, when the smart contract runs for a long time, smaller data types may encounter the problem of running out of identifiers. In contrast, `uint256` can avoid this problem. On the other hand, `uint256` is the most resource-efficient type in Solidity even though it uses more bits than necessary for small numbers. The Ethereum Virtual Machine (EVM) stores data in 32-byte (256-bit) slots. When a `uint256` variable is declared, it occupies exactly one slot. For a smaller type like `uint8`, it still occupies a full 32-byte slot, but only uses a small portion of it. If EVM stores multiple small varibles in one slot, packing and unpacking consume more gas than reading and writing.

The `purpose` variable in `DataRequest` is a `string`. It can be a single word or short sentence to explain the purpose of the service. For example, the airline company may put flight booking as the purpose. The `approved` variable in `DataRequestResponse` is a `boolean` variable. It is a binary value. `True` represents that the data request is approved as no excessive data is required. `False` represents that the data request is denied by the checker.

The `requestCounter` in Listing 3.3 is a `uint256` variable used to tracker the number of requests. The `private` keyword is a visibility modifier in Solidity that restricts access to a variable or function to only the contract itself. It cannot be accessed by other contracts or off-chain systems. If `requestCounter` were `public`, anyone could read or modify it

directly. We add this keyword to prevent external manipulation. By making it private, we ensure that only the contract's functions can increment it, which means `requestId` values are sequential and unpredictable by external actors.

As Listing 3.4 shows, `VerificationRequest` contains an extra boolean variable apart from the data types discussed above. `responded` is used to track pending requests.

In these events, `fields` is a string consisting of several words separated by comma.

`indexed` is a keyword that marks a parameter as indexed in an event. Indexed parameters are hashed and stored in the event's topics, which is a special data structure in Ethereum logs. Through using `indexed`, parameters can be efficiently filtered by off-chain systems. For example, in `checker`, it is possible to filter events where the verifier is a specific address without scanning all events. Non-indexed parameters are stored in the data field of the log. Although indexed parameters can accelerate filtering, they also increase gas cost because the data must be hashed. In addition, only the first three parameters can be indexed, which is limited by the Solidity language. That is why we choose to index only parameters needed to filter by (e.g., verifier, holder) rather than indexing all parameters.

When verifier sends a data request to holder, holder will send the data request to checker. After making the decision, the checker will send the response back to the holder. The holder can decide to accept or reject the data request based on the response sent by the checker. Finally, the holder can send the response to the verifier.

Chapter 4

Implementation

In this chapter, details of implementation will be discussed in several aspects, including data structure, major functions, on-chain smart contract, off-chain flask application, how the on-chain part and off-chain part communicate, and model training.

4.1 Data preprocessing

When `checker` receives a `DataRequest`, `checker` will call `generate_field_vector` to convert the `fields` string into `field_vector_formatted` vector for further computation shown in Listing 4.1. Here, we choose to make `purpose` as `string` instead of `array` or `vector`, because *Web3.py* and many Ethereum clients do not support filtering for events with dynamic types, like `string[]`, in the event signature. This means that while the event is emitted and visible on-chain, the event filter in Web3.py will not catch it if the event contains a dynamic array. In the following code, `string fields` is converted into `field_vector_formatted` by the function `generate_field_vector`.

```
1  fields = "name, email, passport_number"
2  def generate_field_vector(fields, fields_dic):
3      # Convert the input fields string into a list (assuming comma-
           separated)
4      fields_list = fields.split(", ")
5
6      # Initialize an empty list of 0s with the same length as fields_dic
7      field_vector = [0] * len(fields_dic)
8
9      # Set the corresponding index to 1 for each field found in
           fields_list
10     for field in fields_list:
11         if field in fields_dic:
12             index = fields_dic.index(field)
13             field_vector[index] = 1
14     field_vector_formatted = np.array(field_vector).reshape(1, -1)
15     return field_vector_formatted
```

Listing 4.1: Convert string fields into formatted vector

4.2 On-chain smart contract

Apart from the data structure discussed in Chapter 4.1, smart contract `RestrictedVerifier.sol` consists of several functions to emit, handle and respond events.

```
1     function requestData(
2         address verifier,
3         string memory fields,
4         string memory purpose
5     ) public returns (uint256) {
6         require(bytes(fields).length > 0, "Must specify fields");
7         require(bytes(purpose).length > 0, "Must specify purpose");
8
9         requestCounter++;
10
11        requests[requestCounter] = VerificationRequest({
12            verifier: verifier,
13            holder: msg.sender,
14            fields: fields,
15            purpose: purpose,
16            responded: false
17        });
18
19        emit DataRequest(verifier, msg.sender, fields, purpose,
              requestCounter);
20
21        return requestCounter;
22    }
```

Listing 4.2: Request data function

Listing 4.2 shows the function `requestData`. This function is called by the verifier to request data from the holder. Therefore, the parameters are the address of verifier, fields requested and purpose. In line 7-8, there are two assertions to ensure both fields and purpose are not empty. After validate the data request, we increase the `requestCounter` by one to keep the counter updated. Then we store the request in memory after getting the address of the holder. Finally, we emit the event `DataRequest`. The return value is the index of the request from the counter.

```
1
2     function respondToRequest(uint256 requestId, bool approved) public
          {
3         VerificationRequest storage request = requests[requestId];
4         require(!request.responded, "Request already responded");
5         require(msg.sender == request.verifier, "Only verifier can
              respond");
6
7         request.responded = true;
8
9         emit DataRequestResponse(request.verifier, request.holder,
              approved, requestId);
10    }
```

Listing 4.3: Respond to request function

Function `respondToRequest` in Listing 4.3 is used to respond to the request by the verifier. Firstly, we get the request from storage. Then we check if the request has already been responded. If yes, no further response is needed. Then, the assertion is to ensure only verifier can respond. After the validation, we can actually change the status of the request to responded. Lastly, we emit the event `DataRequestRespond`.

```
    function getRequest(uint256 requestId) public view returns (
        address verifier,
        address holder,
        string memory fields,
        string memory purpose,
        bool responded
    ) {
        VerificationRequest storage request = requests[requestId];
        return (
            request.verifier,
            request.holder,
            request.fields,
            request.purpose,
            request.responded
        );
    }
}
```

Listing 4.4: Get request function

Listing 4.4 shows the function `getRequest` which is used to acquire the request. The parameter is the ID of the request. We get the request from the storage and return the needed information including the address of both verifier and holder, fields requested, purpose and the status of the request.

4.3 Off-chain Flask application

```
model = joblib.load('model.pkl')
# Connect to the Ethereum node
w3 = Web3(Web3.HTTPProvider('http://127.0.0.1:8545'))
print("Connected to Ethereum:", w3.is_connected())

# Load the ABI
with open('RestrictedVerifier_abi.json') as f:
    abi = json.load(f)

# Set the deployed contract address
contract_address = '0x0B306BF915C4d645ff596e518fAf3F9669b97016'

# Get contract object
verifier = w3.eth.contract(address=contract_address, abi=abi)
```

Listing 4.5: Flask application setup

Listing 4.5 shows the setup of the off-chain Flask application. Firstly, the machine learning is loaded using joblib which is a light-weight pipelining tool in python. Then, we

connect to the Ethereum node using web3. Through an output, we can check if the connection is successful. After the successful connection, we load the ABI file, which is a json description file to explain the interface of Solidarity smart contract. Then, the core to connect on-chain verifier and off-chain system is to set the contract address which is generated after deployment. With the contract address, we can get the verifier instance through web3.

```
# Function to handle new events
def handle_event(event):
    verifier = event['args']['verifier']
    request_id = event['args']['requestId']
    holder = event['args']['holder']
    fields = event['args']['fields']
    purpose = event['args']['purpose']

    print(f"Received Request from {holder} for {fields} with purpose '{
        purpose}'")

    decision = decide(fields, purpose)
    print(f"Decision: {decision}")

    # Call approveRequest
    tx_hash = verifier.functions.respondToRequest(request_id, decision)
        .transact({'from': verifier})
    receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
    print(f"Approval Transaction complete: {receipt.transactionHash.hex
        ()}")
```

Listing 4.6: Handle event

Listing 4.6 shows how `checker` handle the event. Firstly, all parameters needed are stored locally to avoid frequent call to access the event. Then, we call the function `decide` to get the decision. After getting the result, we call function `respondToRequest` to generate the hash which is passed to web3 for communication.

```
def generate_field_vector(fields, fields_dic):
    # Convert the input fields string into a list (assuming comma-
        separated)
    fields_list = fields.split(", ")

    # Initialize an empty list of 0s with the same length as fields_dic
    field_vector = [0] * len(fields_dic)

    # Set the corresponding index to 1 for each field found in
        fields_list
    for field in fields_list:
        if field in fields_dic:
            index = fields_dic.index(field)
            field_vector[index] = 1
    field_vector_formatted = np.array(field_vector).reshape(1, -1)
    return field_vector_formatted
```

Listing 4.7: Get field vector

This function is used to process the input string of `fields`. As it is separated by comma, we get `fields list` from the string `fields`. Then we match the items in the list to the `fields dic` which we created before. We convert the list to a binary list which lowers the computational complexity and is gas-efficient. At last, this function will return a formatted binary list.

```
#decision function
def decide(fields, purpose):
    # Example: if purpose contains "Marketing" reject it
    fields_dic = ['name', 'email', 'passport_number', 'birthdate', '
        phone_number', 'height', 'blood_type', 'shoe_size', '
        favorite_color', 'eye_color', 'pet_name']
    if "flight" in purpose.lower():
        input_vector = generate_field_vector(fields, fields_dic)
        result = model.predict(input_vector)
        if result == 1:
            return True
        else:
            return False
    return False
```

Listing 4.8: Decision function

The function `decide` in Listing 4.8 is the core part of the whole verification process. We set the `fields dic` first for the function `generate_field_vector` in Listing 4.7. Then we process the purpose variable to match the use case. At last, we can load the model and make the prediction. If the response is `TURE`, the the function will return `TRUE`. Otherwise, `FALSE` will be returned.

```
# Subscribe to new events
def log_loop(event_filter, poll_interval):
    print("Event Listener Thread Started...")
    while True:
        for event in event_filter.get_new_entries():
            print("New Event Detected")
            handle_event(event)
        time.sleep(poll_interval)
```

Listing 4.9: Log loop function

The function `log_loop` is for constantly listening to the port if any new event is detected.

```
# Create event filter
event_filter = verifier.events.DataRequest.create_filter(from_block=0)
```

Listing 4.10: Event filter

`event_filter` with `from_block=0` means it will listen for all occurrences of the DataRequest event from the very beginning of the blockchain.

```
app = Flask(__name__)
@app.route('/')
def home():
    return "Flask app running and listening to blockchain events!"

if __name__ == '__main__':

    # Start event listener thread
    thread = threading.Thread(target=log_loop, args=(event_filter, 2),
        daemon=True)
    thread.start()

    print("Started Flask app and Blockchain Event Listener!")

    # Start Flask app
    app.run(port=5000, threaded=True)
```

Listing 4.11: Main function

Listing 4.11 shows the main function of `checker`. We start event listener thread and Flask application.

4.4 Model training

4.4.1 Synthetic dataset generator

```
def generate_synthetic_data(
    n_normal=100,
    n_outliers=5,
    random_seed=42
):
    np.random.seed(random_seed)

    # Step 1: Define fields
    core_fields = ['name', 'email', 'passport_number', 'birthdate', '
        phone_number']
    optional_fields = ['height', 'blood_type', 'shoe_size', '
        favorite_color', 'eye_color', 'pet_name']
    all_fields = core_fields + optional_fields

    # Step 2: Create seed training data (core fields = 1, optional = 0)
    seed_data = []
    for _ in range(10):
        record = {field: 1 for field in core_fields}
        record.update({field: 0 for field in optional_fields})
        seed_data.append(record)
    seed_df = pd.DataFrame(seed_data)

    # Step 3: Define metadata
    metadata = SingleTableMetadata()
    metadata.detect_from_dataframe(seed_df)

    # Step 4: Fit synthesizer
```

```
26      synthesizer = GaussianCopulaSynthesizer(metadata)
27      synthesizer.fit(seed_df)
28
29      # Step 5: Generate normal companies
30      normal_data = synthesizer.sample(n_normal)
31
32      # Add tiny noise (some optional fields activated rarely)
33      for field in optional_fields:
34          normal_data[field] = normal_data[field].apply(lambda x: 1 if np
                .random.rand() < 0.05 else 0)
35      normal_data = synthesizer.sample(n_normal)
36      ################################################
37      # this is to make all data binary in order to perform pca
38      # Force all fields to be binary (some SDV models generate floats)
39      normal_data[all_fields] = normal_data[all_fields].applymap(lambda x
            : 1 if str(x).lower() in ['1', 'true'] else 0)
40
41      # Random tiny noise
42      for field in optional_fields:
43          normal_data[field] = normal_data[field].apply(lambda x: 1 if np
                .random.rand() < 0.05 else x)
44      ################################################
45
46      # Step 6: Inject outlier companies
47      outlier_data = []
48      for _ in range(n_outliers):
49          record = {field: 1 for field in core_fields}
50          # Choose 3 random weird fields to add
51          weird_fields = np.random.choice(optional_fields, size=3,
                replace=False)
52          record.update({field: 1 if field in weird_fields else 0 for
                field in optional_fields})
53          outlier_data.append(record)
54      outlier_df = pd.DataFrame(outlier_data)
55
56      return full_df
```

Listing 4.12: Synthetic dataset generator function

In the Listing 4.12, we can see the whole process of generating synthetic dataset depending on SDV. We set core fields and optional fields first. Then we use random seed to create seed training data. We define metadata and choose synthesizer following the steps of SDV. After the preparation steps, we can generate normal companies. We also add tiny noise to avoid bias. After getting the normal dataset, we inject outliers to complete the whole dataset. This function will return a full dataset for further use.

```
1 # Generate the data
2 core_fields_flight_ticket = ['name', 'email', 'passport_number', '
      birthdate', 'phone_number']
3 optional_fields_flight_ticket = ['height', 'blood_type', 'shoe_size', '
      favorite_color', 'eye_color', 'pet_name']
4 df_flight_ticket = generate_synthetic_data(core_fields=
      core_fields_flight_ticket,
5                                            optional_fields=
                                                optional_fields_flight_ticket,
```

```
6                                 n_normal=100, n_outliers=5)
7 df_flight_ticket.to_csv("output_flight_ticket.csv")
```

Listing 4.13: Generate datasets for use case: flight ticket purchase

Listing 4.13 show how we generate the synthetic dataset for the use case flight ticket purchase. Firstly, we configure the core fields as name, email, passport number, birth date and phone number. Then we set the optional fields as height, blood type, shoe size, favorite color, eye color and pet name. After setting up all the fields, we call the function `generate_synthetic_data` and set the parameters. At last, we export the dataset to a csv file `output_flight_ticket.csv`.

```
1 # Generate the data
2 core_fields_student = ['name', 'email', 'passport_number', 'birthdate',
      'phone_number', 'studentID', 'gender', 'age']
3 optional_fields_student = ['height', 'blood_type', 'shoe_size', '
    favorite_color', 'eye_color', 'pet_name']
4 df_student = generate_synthetic_data(core_fields=core_fields_student,
5                                 optional_fields=optional_fields_student,
6                                 n_normal=1000, n_outliers=5)
7 df_student.to_csv("output_student.csv")
```

Listing 4.14: Generate datasets for use case: student information management

Listing 4.14 show how we generate the synthetic dataset for the use case student information management. Firstly, we configure the core fields as name, email, passport number, birth date, phone number, studnet ID, gender and age. Then we set the optional fields as height, blood type, shoe size, favorite color, eye color and pet name. After setting up all the fields, we call the function `generate_synthetic_data` and set the parameters. At last, we export the dataset to a csv file `output_student.csv`.

```
1 # Generate the data
2 core_fields_restaurant = ['name', 'time']
3 optional_fields_restaurant = ['height', 'blood_type', 'shoe_size', '
    favorite_color', 'eye_color', 'pet_name']
4 df_restaurant = generate_synthetic_data(core_fields=
    core_fields_restaurant,
5                                 optional_fields=optional_fields_restaurant
                                      ,
6                                 n_normal=1000, n_outliers=5)
7 df_restaurant.to_csv("output_restaurant.csv")
```

Listing 4.15: Generate datasets for use case: restaurant reservation

Listing 4.15 show how we generate the synthetic dataset for the use case restaurant reservation. Firstly, we configure the core fields as name and time. Then we set the optional fields as height, blood type, shoe size, favorite color, eye color and pet name. After setting up all the fields, we call the function `generate_synthetic_data` and set the parameters. At last, we export the dataset to a csv file `output_restaurant.csv`.

```
1  def create_point_cloud_plot(data_file, title, n_normal, n_outliers,
       output_filename):
2      # Read the data
3      X_data = pd.read_csv(data_file, index_col=0)
4      X_data_raw = X_data.to_numpy()
5
6      # Use PCA to reduce dimensions for visualization
7      pca = PCA(n_components=2)
8      X_data_2d = pca.fit_transform(X_data_raw)
9
10     # Separate normal and outlier points
11     normal_points = X_data_2d[:n_normal]
12     outlier_points = X_data_2d[n_normal:]
13
14     # Create scatter plot
15     plt.figure(figsize=(12, 8))
16
17     # Plot normal points
18     plt.scatter(normal_points[:, 0], normal_points[:, 1],
19                 alpha=0.7, s=60, c='blue', label='Normal')
20
21     # Plot outlier points
22     plt.scatter(outlier_points[:, 0], outlier_points[:, 1],
23                 alpha=0.9, s=80, c='red', label='Outlier')
24
25     plt.title(title)
26     plt.legend()
27     plt.axis('off')  # Remove axis
28
29     # Save the plot
30     plt.savefig(output_filename, dpi=300, bbox_inches='tight')
31     plt.show()
32     plt.close()
```

Listing 4.16: Point cloud polt generation function

Listing 4.16 shows the function to generate point cloud for the datasets. We import the csv file to `X_data` using pandas library from python and set the parameter `index_col` to 0 to discard the index column in line 3. After importing the csv file using pandas, the datatype of `X_data` is `DataFrame`. In order to use it for model training, we convert `X_data` to `numpy` data type and get `X_data_raw`. In line 7, we initiate a two-component PCA instance. Then we call the function `fit_transform` of pca to convert `X_data_raw` into an `ndarray` variable. We use PCA to reduce dimensions for visualization. Then in line 11-12, we separate normal and outlier points and store in two variables. After the preprocessing, we can start to draw the point cloud plot. We choose `matplotlib.pyplot` because its outstanding capability of visualization. We use the function `pyplot.scatter` to plot both normal and outlier points. To make the plot more readable, we add title and legend. At last, we export the plot using the function `pyplot.savefig`

```
1  # Generate plots for all three datasets
2  # Flight ticket data
3  create_point_cloud_plot(
4      'output_flight_ticket.csv',
5      'Flight Ticket Data Points (PCA Visualization)',
```

```
6       100, 5,
7       'flight_ticket_point_cloud.png'
8   )
```

Listing 4.17: Point cloud plot for the use case: flight ticket purchase

In Listing 4.17, we call the function `create_point_cloud_plot` to draw cloud point plot for the use case: flight ticket purchase.

```
1   # Generate plots for all three datasets
2   # Student data
3   create_point_cloud_plot(
4       'output_student.csv',
5       'Student Data Points (PCA Visualization)',
6       1000, 5,
7       'student_point_cloud.png'
8   )
```

Listing 4.18: Point cloud plot for the use case: student information management

In Listing 4.18, we call the function `create_point_cloud_plot` to draw cloud point plot for the use case: student information management.

```
1   # Generate plots for all three datasets
2   # Restaurant data
3   create_point_cloud_plot(
4       'output_restaurant.csv',
5       'Restaurant Data Points (PCA Visualization)',
6       1000, 5,
7       'restaurant_point_cloud.png'
8   )
```

Listing 4.19: Point cloud plot for the use case: restaurant reservation

In Listing 4.19, we call the function `create_point_cloud_plot` to draw cloud point plot for the use case: restaurant reservation.

4.4.2 Unsupervised learning

```
1   X_raw = X.to_numpy()
2   model = IsolationForest(contamination=0.25, random_state=42)
3   model.fit(X_raw)
```

Listing 4.20: Training isolation forest

We convert the raw data to numpy. Then, we set the machine learning model and train it with our training data.

4.4.3 Supervised learning

This section discusses about the implementation of supervised learning models. We choose logistic regression, random forest and decision tree. In this section, we will show the code of preparation, model training and plot drawing

4.4.3.1 Preparation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression # Import Logistic Regression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import sys
from datetime import datetime
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.ensemble import RandomForestClassifier
# Redirect all output to a log file
log_filename = f'model_training_log_{datetime.now().strftime("%Y%m%d_%H%M%S")}.txt'
log_file = open(log_filename, 'w')
original_stdout = sys.stdout
sys.stdout = log_file

print(f"Model Training Log - Started at {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
print("=" * 80)

# --- 1. Load the noisy dataset ---
try:
    df = pd.read_csv('output_noisy.csv')
    print("\nDataset Info:")
    df.info()
    print("\nValue counts for 'Is_Excessive':")
    print(df['Is_Excessive'].value_counts())
except FileNotFoundError:
    print("Error: 'output_noisy.csv' not found. Please make sure the file is in the same directory as the script.")
    exit()

# --- 2. Separate features (X) and target (y) ---
X = df[['Requested_Data_Type', 'Purpose_of_Request']]
y = df['Is_Excessive']

print(f"\nFeatures (X) shape: {X.shape}")
print(f"Target (y) shape: {y.shape}")

```

```
41 # --- 3. Define the preprocessor for One-Hot Encoding ---
42 preprocessor = ColumnTransformer(
43     transformers=[
44         ('cat', OneHotEncoder(handle_unknown='ignore'), ['
              Requested_Data_Type', 'Purpose_of_Request'])
45     ])
46
47 # --- 4. Split the data into training and testing sets ---
48 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
      =0.2, random_state=42, stratify=y)
49
50 print(f"\nTraining set size: {len(X_train)} samples")
51 print(f"Testing set size: {len(X_test)} samples")
52 print("\n'Is_Excessive' distribution in training set:")
53 print(y_train.value_counts(normalize=True))
54 print("\n'Is_Excessive' distribution in test set:")
55 print(y_test.value_counts(normalize=True))
```

Listing 4.21: Preparation

In Listing 4.21, we can see the preparation steps before training supervised learning models. From line 1 to line 13, we import necessary python libraries, including `pandas`, `sklearn`, `seaborn`, `matplotlib`, `sys`, `datetime`. Among all these libraries, `sklearn` provides the core functions for model training.

Line 15-18 shows how the log files are generated. We use the `sys` library to generate logs. To make the whole process clear, we use `print` to record every step.

From line 24 to line 32, we can see that dataset is loaded through `pandas` and stored in a `DataFrame` object. Then we print the basic information of the dataset. In this section, we use the `try-catch` structure to catch the exception and errors. The second step is separate features (X) and the target (y). We store these attributes in X and y. The third step is defining the preprocessor for One-Hot Encoding. Then the last step is spliting the data into training and testing sets. We set it to 80-20. In line 50-55, we print some debug information and log information.

4.4.3.2 Logistic regression

```
1 # --- 5. Create the machine learning pipeline (Logistic Regression) ---
2 model_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
3                                   ('classifier', LogisticRegression(
                                        random_state=42, solver='liblinear'
                                        , class_weight='balanced'))])
4 # solver='liblinear' is a good default for small datasets and binary
      classification.
5 # class_weight='balanced' helps address potential class imbalance.
6
7 print("\n--- Training the Logistic Regression model with noisy data ---
      ")
8 model_pipeline.fit(X_train, y_train)
```

Listing 4.22: Logistic regression

After the preparation, we can start to train the machine learning model. Listing 4.22 shows the process of training logistic regression model. We use `Pipeline` to load the model and set the classifier to `LogisticRegression`, solver to `liblinear` and class weight to `balanced`. Because `liblinear` is a good default for small datasets and binary classification. `balanced` helps address potential class imbalance.

4.4.3.3 Random forest

```
# --- 5. Create the machine learning pipeline (Random Forest) ---
model_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                 ('classifier', RandomForestClassifier(
                                    random_state=42, n_estimators=200,
                                    class_weight='balanced'))])
# Increased n_estimators to 200 for potentially better learning with
    noise
# Added class_weight='balanced' to handle potential class imbalance if
    0s vastly outnumber 1s

print("\n--- Training the Random Forest model with noisy data ---")
model_pipeline.fit(X_train, y_train)
```

Listing 4.23: Random Forest

After the preparation, we can start to train the machine learning model. Listing 4.23 shows the process of training random forest model. We use `Pipeline` to load the model and set the classifier to `RandomForestClassifier`, estimators to 200 and class weight to `balanced`. Because increased `n_estimators` to 200 for potentially better learning with noise. `balanced` helps address potential class imbalance if 0s vastly outnumber 1s.

4.4.3.4 Decision tree

```
# --- 5. Create and train multiple Decision Tree models with different
    max_depth values ---
max_depths = [5, 10, 20, 100]
models = {}

print("\n--- Training Decision Tree models with different max_depth
    values ---")

for max_depth in max_depths:
    print(f"\nTraining Decision Tree with max_depth={max_depth}")

    # Create the machine learning pipeline (Decision Tree Classifier)
    model_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                     ('classifier',
                                        DecisionTreeClassifier(
                                        random_state=42, max_depth=
                                        max_depth, class_weight='
                                        balanced'))])

    # Train the model
    model_pipeline.fit(X_train, y_train)
```

```
16      models[max_depth] = model_pipeline
17      print(f"Model training complete for max_depth={max_depth}")
```

Listing 4.24: Decision tree

After the preparation, we can start to train the machine learning model. Listing 4.24 shows the process of training decision tree model. Different from logistic regression and random forest, decision tree has an extra parameter `max_depth`. The `max_depth` parameter in a decision tree is a hyperparameter that controls how deep the tree can be. It is used for controlling the complexity of the model. The higher `max_depth` is, more complex the model will be. We set it to 5, 10, 20 and 100 to observe the impact.

We use `Pipeline` to load the model and set the classifier to `DecisionTreeClassifier` and class weight to `balanced`. Because `balanced` helps address potential class imbalance.

4.4.3.5 Evaluation

```
1 # --- 6. Make predictions and evaluate the model ---
2 y_pred = model_pipeline.predict(X_test)
3 y_prob = model_pipeline.predict_proba(X_test)[:, 1] # Probabilities for
      the positive class (Is_Excessive=1)
```

Listing 4.25: Model prediction

In Listing 4.25, we can use the model for prediction and calculate the probabilities for the positive class which means `Is_Excessive =1`.

```
1 print("\n--- Model Evaluation on Noisy Test Set ---")
2 print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
3 print(f"Precision: {precision_score(y_test, y_pred):.4f}")
4 print(f"Recall: {recall_score(y_test, y_pred):.4f}")
5 print(f"F1-Score: {f1_score(y_test, y_pred):.4f}")
6 print(f"ROC AUC: {roc_auc_score(y_test, y_prob):.4f}")
```

Listing 4.26: numerical evaluation

Listing 4.26 shows the evaluation of the models from accuracy, precision score, recall score, F1 score and AUC score.

```
1  # Optional: Confusion Matrix for more detailed insights
2  cm = confusion_matrix(y_test, y_pred)
3  plt.figure(figsize=(6, 4))
4  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
5              xticklabels=['Predicted 0', 'Predicted 1'],
6              yticklabels=['Actual 0', 'Actual 1'])
7  plt.title('Confusion Matrix (Logistic Regression)')
8  plt.ylabel('Actual Label')
9  plt.xlabel('Predicted Label')
10 plt.savefig('confusion_matrix_lr.png', dpi=300, bbox_inches='tight')
11 plt.close()
```

Listing 4.27: Confusion matrix generation

Listing 4.27 shows how confusion matrices is generated. We use the function `confusion_matrix` to calculate confusion matrices. Then we use `seaborn.heatmap` to draw the scatter plots.

```
# --- Precision-Recall Curve ---
from sklearn.metrics import precision_recall_curve,
    average_precision_score

precision, recall, _ = precision_recall_curve(y_test, y_prob)
average_precision = average_precision_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2,
         label=f'Precision-Recall curve (AP = {average_precision:.3f})'
            )
plt.plot([0, 1], [1, 0], color='red', lw=1, linestyle='--', label='
    Random classifier')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (Logistic Regression)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('precision_recall_curve_lr.png', dpi=300, bbox_inches='
    tight')
plt.close()
```

Listing 4.28: Precision-Recall generation

We use `precision_recall_curve` from `sklearn.metrics` to draw the figures. We add the grid to make the plot more clear.

```
# --- ROC Curve ---
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2,
         label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], color='red', lw=1, linestyle='--', label='
    Random classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Logistic Regression)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('roc_curve_lr.png', dpi=300, bbox_inches='tight')
plt.close()
```

Listing 4.29: AUC-ROC curve generation

We use `ROC_curve` from `sklearn.metrics` to draw the figures.

```
# --- 6. Evaluate all models and generate plots ---
for max_depth in max_depths:
    model_pipeline = models[max_depth]

    # Make predictions
    y_pred = model_pipeline.predict(X_test)
    y_prob = model_pipeline.predict_proba(X_test)[:, 1] # Probabilities for the positive class (Is_Excessive=1)

    print(f"\n--- Model Evaluation for max_depth={max_depth} ---")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"Recall: {recall_score(y_test, y_pred):.4f}")
    print(f"F1-Score: {f1_score(y_test, y_pred):.4f}")
    print(f"ROC AUC: {roc_auc_score(y_test, y_prob):.4f}")

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Predicted 0', 'Predicted 1'],
                yticklabels=['Actual 0', 'Actual 1'])
    plt.title(f'Confusion Matrix (Decision Tree, max_depth={max_depth})')
    plt.ylabel('Actual Label')
    plt.xlabel('Predicted Label')
    plt.savefig(f'confusion_matrix_dt_depth{max_depth}.png', dpi=300, bbox_inches='tight')
    plt.close()

    # Precision-Recall Curve
    from sklearn.metrics import precision_recall_curve, average_precision_score

    precision, recall, _ = precision_recall_curve(y_test, y_prob)
    average_precision = average_precision_score(y_test, y_prob)

    plt.figure(figsize=(8, 6))
    plt.plot(recall, precision, color='blue', lw=2,
             label=f'Precision-Recall curve (AP = {average_precision:.3f})')
    plt.plot([0, 1], [1, 0], color='red', lw=1, linestyle='--', label='Random classifier')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve (Decision Tree, max_depth={max_depth})')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.savefig(f'precision_recall_curve_dt_depth{max_depth}.png', dpi=300, bbox_inches='tight')
    plt.close()

    # ROC Curve
```

```
47      from sklearn.metrics import roc_curve
48
49      fpr, tpr, _ = roc_curve(y_test, y_prob)
50      roc_auc = roc_auc_score(y_test, y_prob)
51
52      plt.figure(figsize=(8, 6))
53      plt.plot(fpr, tpr, color='blue', lw=2,
54               label=f'ROC curve (AUC = {roc_auc:.3f})')
55      plt.plot([0, 1], [0, 1], color='red', lw=1, linestyle='--', label='
            Random classifier')
56      plt.xlim([0.0, 1.0])
57      plt.ylim([0.0, 1.05])
58      plt.xlabel('False Positive Rate')
59      plt.ylabel('True Positive Rate')
60      plt.title(f'ROC Curve (Decision Tree, max_depth={max_depth})')
61      plt.legend()
62      plt.grid(True, alpha=0.3)
63      plt.savefig(f'roc_curve_dt_depth{max_depth}.png', dpi=300,
            bbox_inches='tight')
64      plt.close()
```

Listing 4.30: Evaluation for decision tree

In Listing 4.30, we evaluated decision tree model with different max depth, including accuracy, F1 score, confusion matrix, AUC-ROC curve and precision-recall curve.

```
1  # --- Optional: Visualize the Decision Tree (requires graphviz and
       pydotplus) ---
2  # This part is for visualization, you might need to install external
       software (Graphviz)
3  # and Python libraries (graphviz, pydotplus).
4  # pip install graphviz pydotplus
5  try:
6      from sklearn.tree import export_graphviz
7      import graphviz
8      from IPython.display import Image
9      import pydotplus
10
11     # Get feature names after one-hot encoding
12     feature_names = model_pipeline.named_steps['preprocessor'].
           named_transformers_['cat'].get_feature_names_out(
13          ['Requested_Data_Type', 'Purpose_of_Request']
14     )
15
16     dot_data = export_graphviz(model_pipeline.named_steps['classifier'
           ],
17                                feature_names=feature_names,
18                                class_names=['Not Excessive', 'Excessive
                                      '],
19                                filled=True, rounded=True,
20                                special_characters=True, out_file=None)
21     graph = pydotplus.graph_from_dot_data(dot_data)
22     graph.write_png('decision_tree.png')
23     print("\nDecision Tree visualization saved as 'decision_tree.png'")
24
25 except ImportError:
```

```
26      print("\nSkipping Decision Tree visualization. Install 'graphviz'
            and 'pydotplus' to enable it.")
27      print("You might also need to install Graphviz software: https://
            graphviz.org/download/")
28  except Exception as e:
29      print(f"\nAn error occurred during Decision Tree visualization: {e}
            ")
30
31  # Restore stdout and close log file
32  sys.stdout = original_stdout
33  log_file.close()
```

Listing 4.31: Visualization for decision tree

Listing 4.31 shows the visualization process of decision tree. Visualization helps us understand decision process more clearly. We need `graphviz` and `pydotplus` to run this code snippet. This step is optional and not mandatory for evaluation.

Chapter 5

Evaluation

This chapter discusses about the results from the experiments. We evaluate the results based on two metrics. One is capability test, which is used to measure the capability of the unsupervised-learning model in detecting excessive data requests. This is designed to test the whole pipeline. The other one is performance test. We measure the speed and accuracy of the machine learning models. We use speed, F1 score, AUC-ROC curve, precision-recall curve to compare the performance of different supervised-learning models.

5.0.1 Environment setup

This section discusses about the environment setup including both hardware and software.

5.0.1.1 Hardware

Device	Apple Macbook PRO 2020
Processor (CPU)	Apple M1 chip 8-core CPU
Graphics Processing Unit (GPU)	Apple M1 chip 8-core GPU
RAM	8GB
Operating System	macOS 12 Monterey

Table 5.1: Hardware setup

5.0.1.2 Software

Virtual Environment	conda 23.1.0
Python	3.10.16

Table 5.2: Software setup

Library	Version
pandas	2.3.1
numpy	2.2.5
skikit-learn	1.6.1
matplotlib	3.10
seaborn	0.13.2

Table 5.3: Libraries version

	name	email	passport number	birth date	phone number	height	blood type	shoe size	favorite color	eye color	pet name
1	1	0	1	1	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	1	0	0
3	1	0	1	1	0	0	0	0	0	0	0
4	1	0	1	1	0	0	0	0	0	0	0
5	1	0	1	1	0	0	0	0	0	0	0
6	1	0	1	1	0	0	0	0	0	0	0

Table 5.4: Sample dataset for use case: flight ticket purchase

5.0.2 Use case

In Table 5.4, 5.5 and 5.6, we can see the sample datasets for three use cases, including flight ticket purchase, student information management, and restaurant reservation. In these tables, 1 means this attribute is requested and 0 represents this attribute is not requested.

5.0.3 Impact on CredChain

This section discusses about the impact of restricted verifier on the Credchain. We measure the average decision time of different models and blockchain transaction and other overhead. We run the same test case for 10 times and take the average to represent the performance.

From Table 5.7, we can see that decision time is relatively short compared with normal blockchain transaction.

	name	email	pass num	birth date	phone num	student ID	gender	age	blood type	shoe size	color	eye color	pet name
1	1	0	1	1	0	0	0	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0	1	0	0	0	0
3	1	0	1	1	0	0	0	0	0	0	0	0	0
4	1	0	1	1	0	0	0	0	0	0	0	0	0
5	1	0	1	1	0	0	0	0	0	0	0	0	0
6	1	0	1	1	0	0	0	0	0	0	0	0	0

Table 5.5: Sample dataset for use case: student information management

	name	time	height	blood type	shoe size	favorite color	eye color	pet name
1	1	0	0	0	1	0	0	0
2	1	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0

Table 5.6: Sample dataset for use case: restaurant reservation

Decision time	0.0046 seconds	17.7%
Blockchain transaction	0.0212 seconds	82%
Other overhead	0.0001 seconds	0.3%

Table 5.7: Impact on Credchain

5.0.4 Capability test

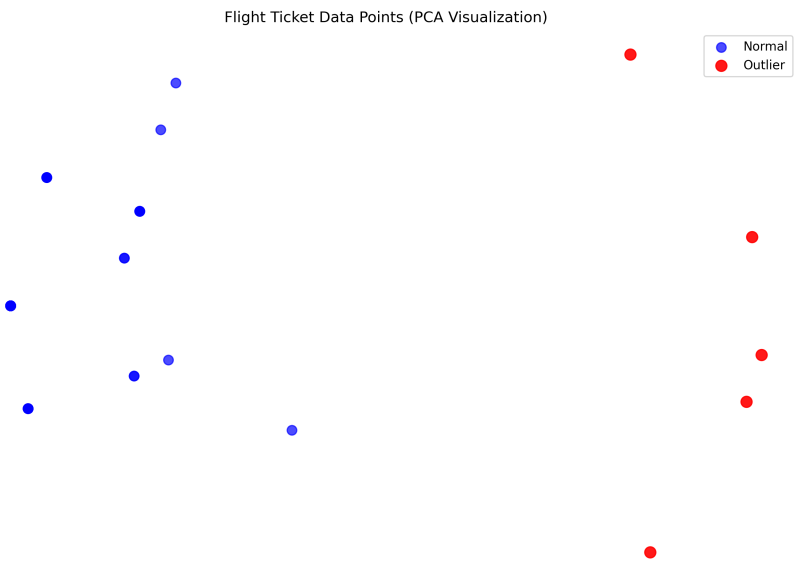

Figure 5.1: Point cloud for use case: flight ticket purchase

We design capability test to verify the functionality of our prototype. In these tests, we focus on the whole pipeline instead of the performance of specific models. We design three use cases and generate related datasets. We can see from Figure 5.1 that in the scenario of purchasing the flight ticket, the outliers are clearly separated from the cluster. On the contrary, Figure 5.2 and Figure 5.3 show that outliers are mixed with normal data points which increase the difficulty of telling them apart.

We train two unsupervised-learning models including Isolation Forest and Autoencoder. As the generated dataset is categorical, both models learn the rules very fast.

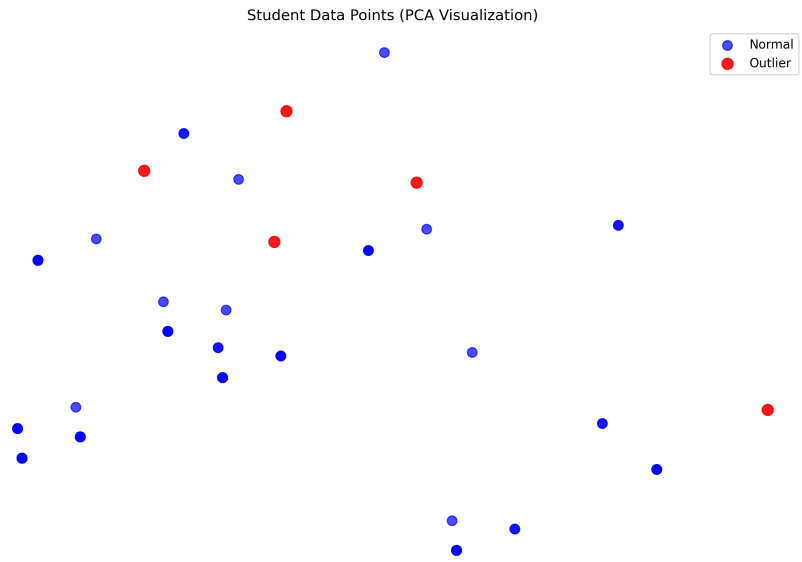

Figure 5.2: Point cloud for use case: student information management

5.0.5 Performance test

In this section, we discuss about the experiment results of supervised learning models. Unlike capability test, we generate synthetic data using LLM instead of SDV. LLM allows us to have more complex dataset. SDV is capable of generating categorical data which is not effective enough to simulate the real world. In this case, we employ Google Gemini [14] to generate synthetic data for training supervised-learning models.

We train three models with same training data including logistic regression, random forest and decision tree. Figure 5.4 to 5.9 shows the confusion matrices of different models. We can see that random forest has the highest true positive rate and lowest false negative rates. Logistic regression also performs well in reporting true positive. With different max depth, decision tree achieve higher in detecting true positive when max depth is higher.

From Figure 5.10 to 5.15, we can see the plots of precision-recall curve. From these figures, we can see the trade-off between precision and recall. In Figure 5.12, there is not much trade-off because the max depth is very low.

From 5.16 to 5.21, we can see the performance of models in detecting true positives against false positives. Random forest has the best performance and decision tree with max depth of 5 performs the worst. Figure 5.18 shows that decision tree with max depth of 5 almost has the same results as random guessing. With the max depth increasing, decision tree performs better. When max depth equals 100, it is possibly to be overfitting.

In Table 5.1, we can see that random forest has the highest accuracy and F1 score. The decision tree with a maximum depth of 5 has the lowest accuracy and F1 score. Logistic regression also performs well. The decision tree has higher accuracy and F1 score when the maximum depth is higher.

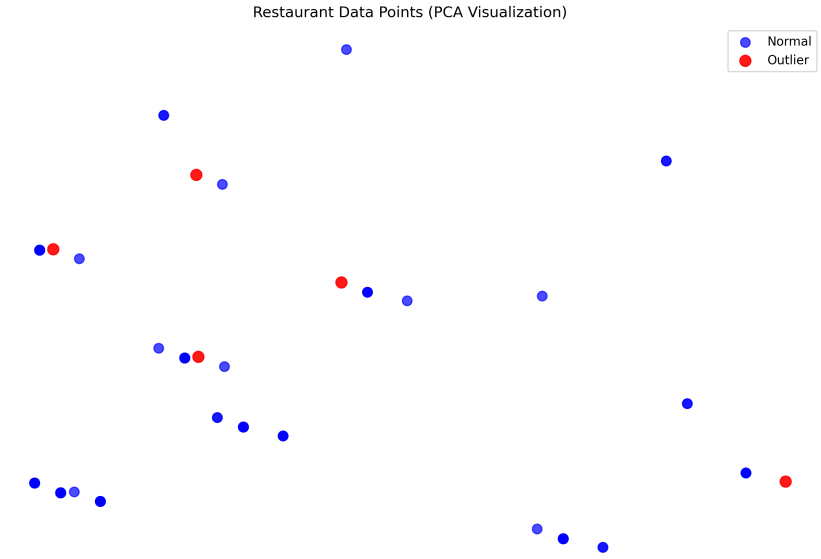

Figure 5.3: Point cloud for use case: restaurant reservation

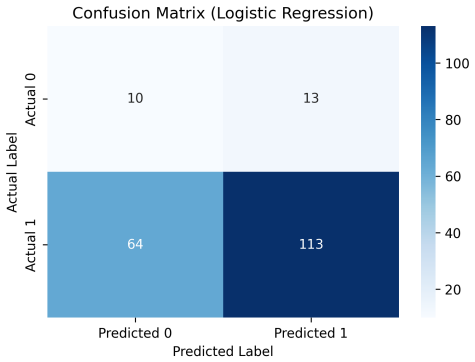

Figure 5.4: Confusion matrix: Logistic Regression

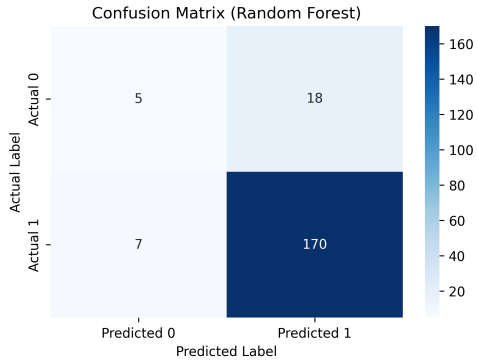

Figure 5.5: Confusion matrix: Random Forest

Figure 5.22 and 5.23 show the visualization of decision tree with different maximum depth.

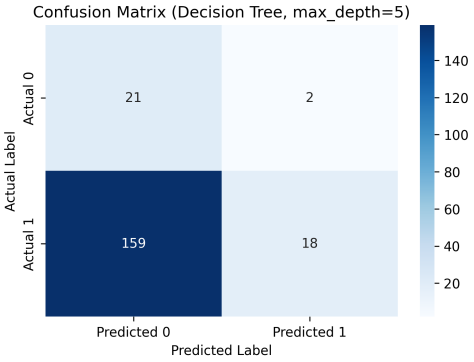

Figure 5.6: Confusion matrix: Decision Tree with Max_depth=5

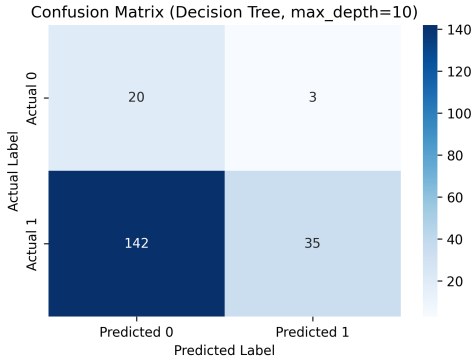

Figure 5.7: Confusion matrix: Decision Tree with Max_depth=10

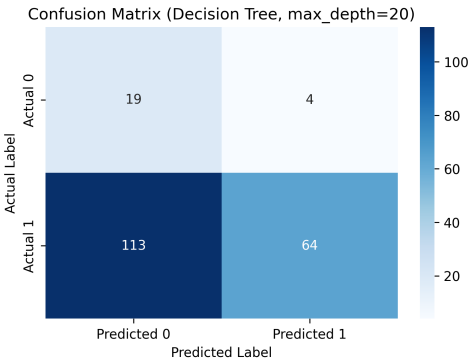

Figure 5.8: Confusion matrix: Decision Tree with Max_depth=20

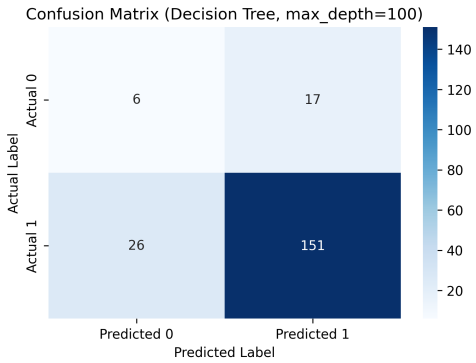

Figure 5.9: Confusion matrix: Decision Tree with Max_depth=100

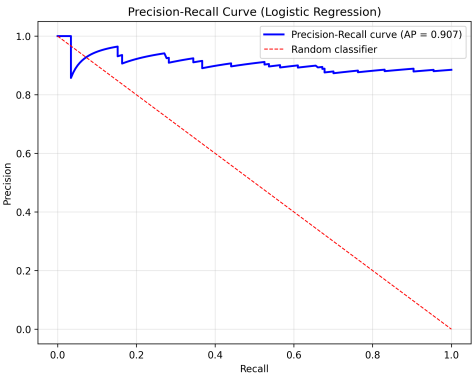

Figure 5.10: Precision-Recall curve: Logistic Regression

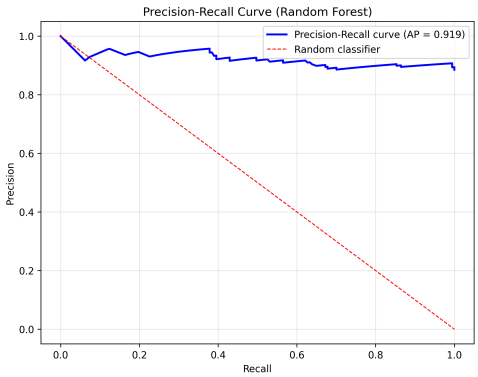

Figure 5.11: Precision-Recall curve: Random Forest

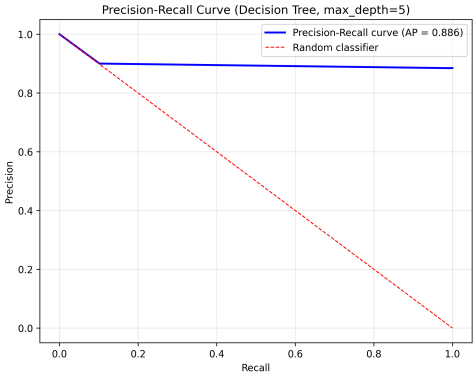

Figure 5.12: Precision-Recall curve: Decision Tree with Max_depth=5

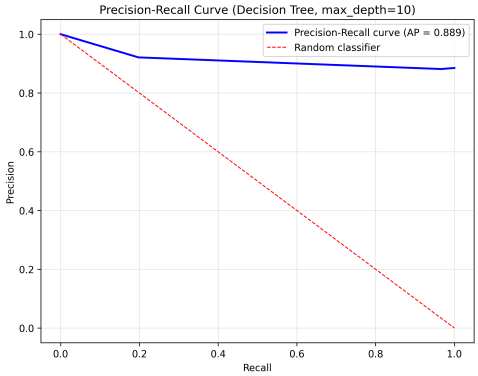

Figure 5.13: Precision-Recall curve: Decision Tree with Max_depth=10

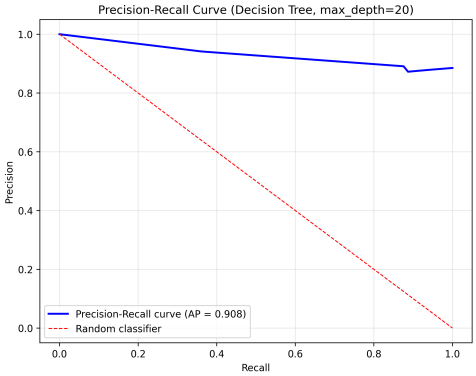

Figure 5.14: Precision-Recall curve: Decision Tree with Max_depth=20

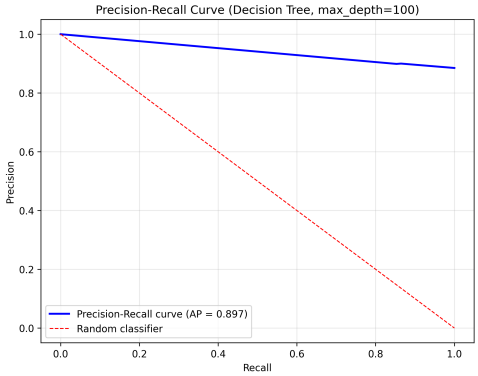

Figure 5.15: Precision-Recall curve: Decision Tree with Max_depth=100

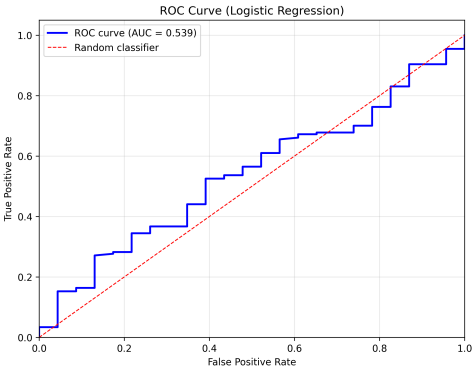

Figure 5.16: ROC curve: Logistic Regression

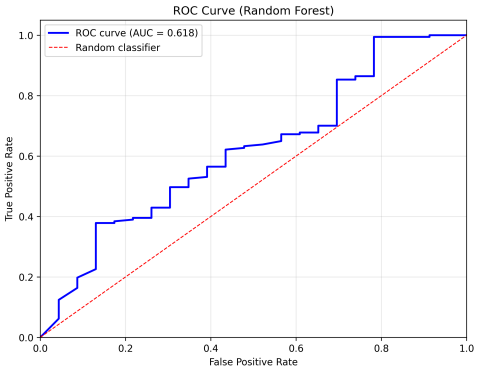

Figure 5.17: ROC curve: Random Forest

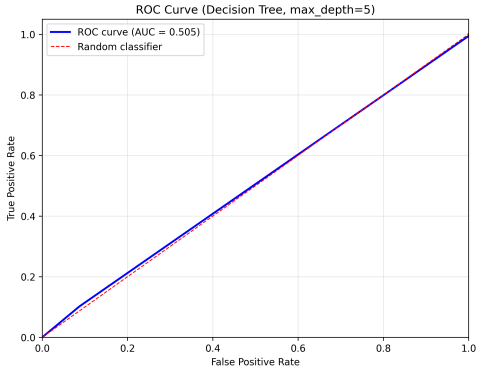

Figure 5.18: ROC curve: Decision Tree with Max_depth=5

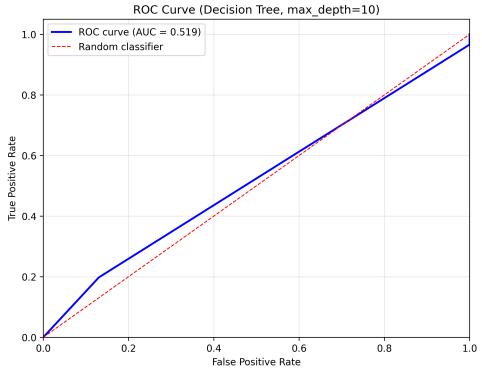

Figure 5.19: ROC curve: Decision Tree with Max_depth=10

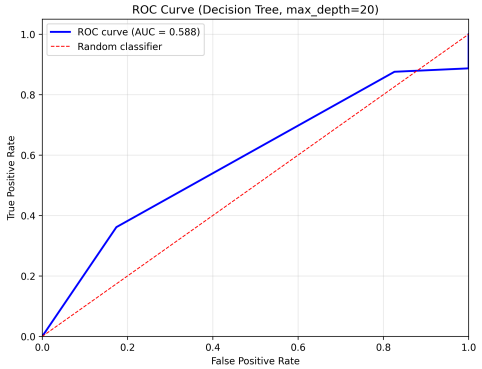

Figure 5.20: ROC curve: Decision Tree with Max_depth=20

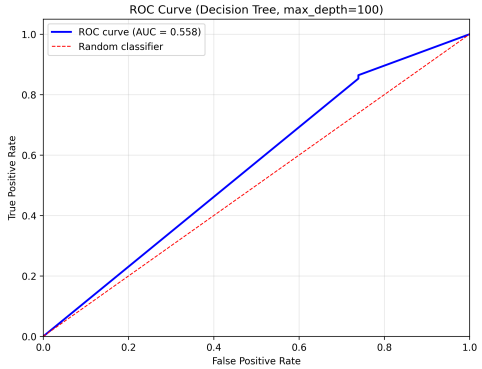

Figure 5.21: ROC curve: Decision Tree with Max_depth=100

Model	Accuracy	F1 score
Logistic Regression	0.6150	0.7459
Random Forest	0.8750	0.9315
decision tree(max_depth=5)	0.1950	0.1827
decision tree(max_depth=10)	0.2750	0.3256
decision tree(max_depth=20)	0.4150	0.5224
decision tree(max_depth=100)	0.7850	0.8754

Table 5.8: Accuracy and F1 score

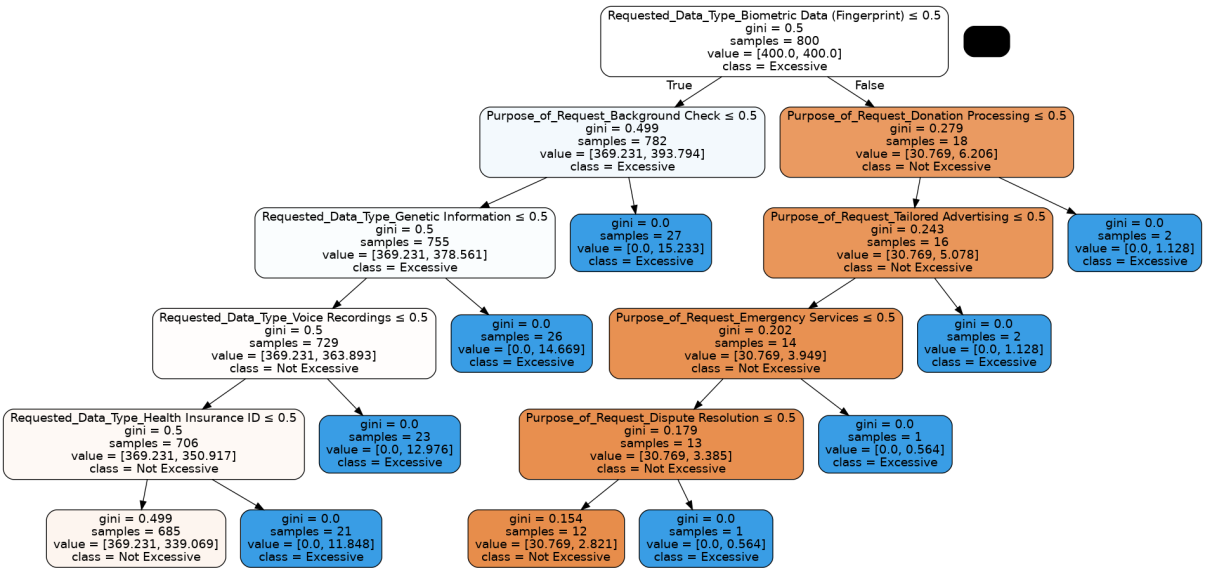

Figure 5.22: Visualization of Decision tree (max_depth=5)

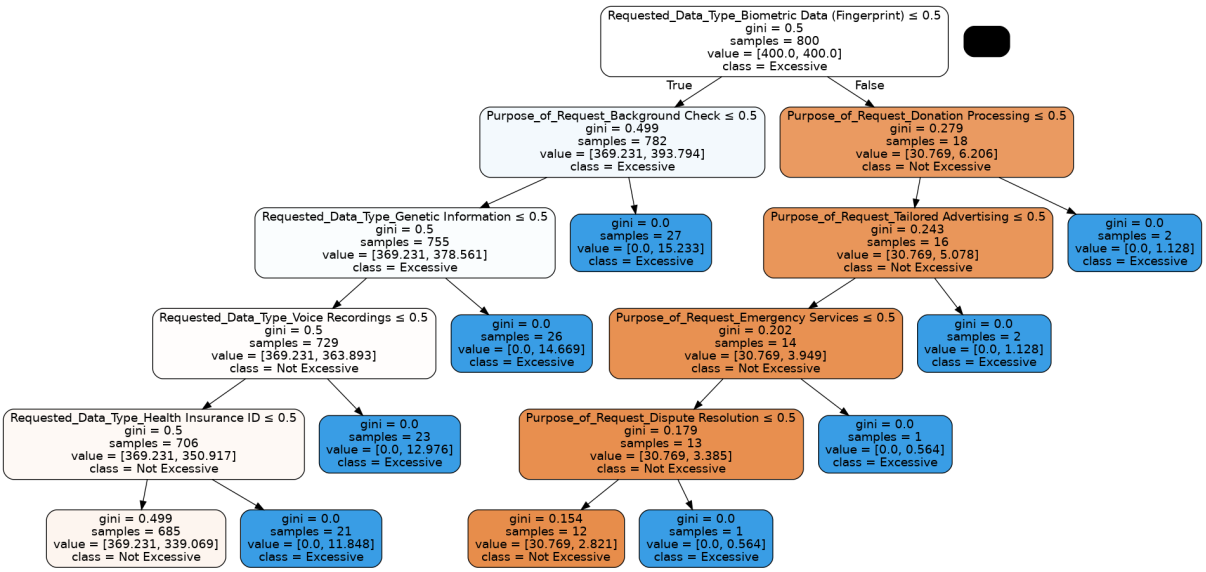

Figure 5.23: Visualization of Decision tree (max_depth=10)

Chapter 6

Final Considerations

6.1 Summary

This thesis successfully implemented and evaluated a new prototype for a user-centric, restricted verifier designed to address excessive data disclosure in SSI systems. The core of this work is developing a novel prototype that empowers information holders to verify the necessity of data requests.

A new smart contract, `RestrictedVerifier.sol`, is implemented and deployed on the blockchain to facilitate the process of event emission, response, and track. This smart contract, as the on-chain component, manage the communication flow between the verifier (service provider) and the holder (user) and ensuring the integrity and transparency of data requests.

For the off-chain component, a machine learning-based checker is developed to act as the automated decision-making tool. To build and train this checker, a comprehensive dataset is essential. Given the absence of suitable public dataset, we use a hybrid data generation approach using both the SDV library and a large language model Gemini to produce high-quality, synthetic datasets. The datasets are used to train and evaluate a variety of machine learning models.

Both unsupervised and supervised learning models are evaluated for their performance and suitability as the checker. Unsupervised models like Isolation Forest and Autoencoders are evaluated for their ability to detect anomalous data requests without prior labeling. For supervised learning, classifiers such as Random Forest, Logistic Regression, and Decision Tree are trained on a labeled dataset to predict whether a data request is excessive. The models are evaluated using various metrics to identify the most accurate and efficient solution for verifying data requests.

In conclusion, this work provides a functional and well-evaluated prototype that enables data minimization from the user's side. This approach fills the gap of user privacy within decentralized identity frameworks.

6.2 Conclusions

This work demonstrates that leveraging machine learning and artificial intelligence to enhance data privacy within DI and SSI systems is feasible and promising. This thesis successfully designs, implements, and evaluates a restricted verifier prototype that empowers users to verify data requests for compliance with data minimization principles.

The experimental results from our prototype show that supervised learning models significantly outperform unsupervised learning models in this specific case. This is because supervised models, trained on a labeled dataset, are able to learn the patterns that distinguish necessary from unnecessary data requests. In contrast, unsupervised models performs well in domain-specific scenarios but have difficulty in complex situations.

Furthermore, the research proves that model performance can be significantly improved through parameter fine-tuning. This process is crucial for achieving the higher accuracy and precision in order to build a reliable and trustworthy system. The fine-tuned model demonstrates the ability to accurately classify data requests.

The successful hybrid integration of the off-chain machine learning-based checker with an on-chain smart contract shows that it is possible to leverage AI to enhance user privacy without interfering the normal function of blockchain system. This approach addresses the latency challenges associated with running machine learning algorithms, which makes the solution practical for real-world practice.

6.3 Future Work

Future work can be done to further train more complex models to be the checker. Large language models also are promising for classifying data requests. Embedding LLM into checker will help it to handle more complex scenarios or corner cases.

This thesis provides a strong foundation for a machine learning-based restricted verifier, but there are many aspects to improve and explore in the future work.

First, although the machine learning models evaluated in this work, such as Random Forest, show good performance in detecting excessive data requests, more complex models could potentially achieve higher accuracy and better generalization ability. For example, deep learning models can discover more sophisticated patterns of data requests, allowing it to handle a wider range of cases that are difficult for traditional classifiers.

Furthermore, LLMs present a particularly promising solution for enhancing the functionality of the checker. LLMs excel at understanding context and intent, which are crucial for recognizing excessive data requests. Integrating an LLM into the checker would allow it to actually understand the scenario instead of doing simple rule-based or feature-based classification. This would allow the checker to handle complex scenarios and unforeseen corner cases.

Bibliography

[1] Matthias Babel and Johannes Sedlmeir. *Bringing data minimization to digital wallets at scale with general-purpose zero-knowledge proofs*. 2023. URL: https://arxiv.org/abs/2301.00823.

[2] Ilaria Battiston and Peter Boncz. "Improving Data Minimization through Decentralized Data Architectures". In: Aug. 2023.

[3] Asia J. Biega et al. "Operationalizing the Legal Principle of Data Minimization for Personalization". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. 2020, 399â408.

[4] Keith Bonawitz et al. "Practical secure aggregation for privacy-preserving machine learning". In: *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1175–1191.

[5] Lee Andrew Bygrave. *Data Privacy Law: An International Perspective*. Jan. 2014.

[6] *CredChain*. https://github.com/schummd/credchain.

[7] Josep Domingo-Ferrer and Alberto Blanco-Justicia. "Towards Machine Learning-Assisted Output Checking for Statistical Disclosure Control". In: *Modeling Decisions for Artificial Intelligence*. Ed. by Vicenç Torra and Yasuo Narukawa. 2021, pp. 335–345.

[8] European Commission. *Proposal for a Regulation amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity*. 2021.

[9] European Parliament and Council of the European Union. *General Data Protection Regulation*. May 4, 2016.

[10] Gemma Galdon Clavell et al. "Auditing Algorithms: On Lessons Learned and the Risks of Data Minimization". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. AIES '20. 2020, 265â271.

[11] Prakhar Ganesh et al. "The data minimization principle in machine learning". In: *arXiv preprint arXiv:2405.19471* (2024).

[12] Alexandra Giannopoulou. "Data protection compliance challenges for self-sovereign identity". In: *International Congress on Blockchain and Applications*. Springer. 2020, pp. 91–100.

[13] Komal Gilani et al. "A Survey on Blockchain-based Identity Management and Decentralized Privacy for Personal Data". In: *2020 2nd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*. 2020, pp. 97–101.

[14] Google. *Gemini 2.5 Flash*. Large language model. 2025. URL: https://gemini.google.com/.

[15] Harry Halpin. “NEXTLEAP: Decentralizing Identity with Privacy for Secure Messaging”. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ARES ’17. 2017.

[16] Thomas Hardjono. “Federated Authorization over Access to Personal Data for Decentralized Identity Management”. In: *IEEE Communications Standards Magazine* 3.4 (2019), pp. 32–38.

[17] Information Commissioner’s Office. *Guide to the UK General Data Protection Regulation (UK GDPR)*. 2021. URL: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/.

[18] Jim Isaak and Mina J. Hanna. “User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection”. In: *Computer* 51.8 (2018), pp. 56–59.

[19] Meg Leta Jones, Ellen Kaufman, and Elizabeth Edenberg. “AI and the Ethics of Automating Consent”. In: *IEEE Security Privacy* 16.3 (2018), pp. 64–72.

[20] Meenakshi Kandpal et al. “Towards Data Storage, Scalability, and Availability in Blockchain Systems: A Bibliometric Analysis”. In: *Data* 8.10 (2023).

[21] Meng Kang and Victoria Lemieux. “A Decentralized Identity-Based Blockchain Solution for Privacy-Preserving Licensing of Individual-Controlled Data to Prevent Unauthorized Secondary Data Usage”. In: *Ledger* 6 (2021).

[22] Li Li et al. “A review of applications in federated learning”. In: *Computers & Industrial Engineering* 149 (2020), p. 106854.

[23] Zhiji Li. “A verifiable credentials system with privacy-preserving based on blockchain”. In: *Journal of Information Security* 13.2 (2022), pp. 43–65.

[24] Alexander Mühle et al. “A survey on essential components of a self-sovereign identity”. In: *Computer Science Review* 30 (2018), pp. 80–86.

[25] Rahma Mukta et al. “A survey of data minimisation techniques in blockchain-based healthcare”. In: *Computer Networks* 205 (2022), p. 108766.

[26] Eman Naboush. “COVID-19 SMART AIR PASS CARD”. In: 15.4 (2022), 11â27.

[27] Michael Nofer et al. “Blockchain”. In: *Business & information systems engineering* 59 (2017), pp. 183–187.

[28] Matthew J Page et al. “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews”. In: *BMJ* 372 (2021), n71.

[29] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. “The Synthetic data vault”. In: *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2016, pp. 399–410.

[30] *Personal Information Protection Law of the People’s Republic of China*. 2021. URL: http://en.npc.gov.cn.cdurl.cn/2021-12/29/c_694559.htm.

[31] K. Samunnisa and Sunil Vijaya Kumar Gaddam. “Blockchain-Based Decentralized Identity Management for Secure Digital Transactions”. In: *Synthesis: A Multidisciplinary Research Journal* 1.2 (2023), 22â29.

[32] Frederico Schardong and Ricardo Custódio. “Self-sovereign identity: a systematic review, mapping and taxonomy”. In: *Sensors* 22.15 (2022), p. 5641.

[33] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. “A data minimization model for embedding privacy into software systems”. In: *Computers Security* 87 (2019), p. 101605.

[34] Robin Staab et al. “From Principle to Practice: Vertical Data Minimization for Machine Learning”. In: *2024 IEEE Symposium on Security and Privacy (SP)*. 2024, pp. 4733–4752.

[35] Quinten Stokkink et al. "A truly self-sovereign identity system". In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE. 2021, pp. 1–8.

[36] Victor Sucasas et al. "Attribute-Based Pseudonymity for Privacy-Preserving Authentication in Cloud Services". In: *IEEE Transactions on Cloud Computing* 11.1 (2023), pp. 168–184.

[37] *us privacy laws*. URL: https://www.commerce.gov/opog/privacy/privacy-laws-policies-and-guidance.

[38] Michael Veale and Reuben Binns. "Fairer machine learning in the real world: Mitigating discrimination without collecting sensitive data". In: *Big Data & Society* 4.2 (2017), p. 2053951717743530.

[39] Vinden Wylde et al. "Cybersecurity, data privacy and blockchain: A review". In: *SN computer science* 3.2 (2022), p. 127.

[40] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. "Decentralizing Privacy: Using Blockchain to Protect Personal Data". In: *2015 IEEE Security and Privacy Workshops*. 2015, pp. 180–184.

Abbreviations

ABI	Application Binary Interface
ACM	Association for Computing Machinery
AI	Artificial Intelligence
AUC	Area Under the Curve
BERT	Bidirectional Encoder Representations from Transformers
CCPA	California Consumer Privacy Act
DI	Decentralized Identity
DIDs	Decentralized Identifiers
ETH	Ether (Cryptocurrency)
EVM	Ethereum Virtual machine
EU	European Union
FL	Federated Learning
FPR	False Positive Rate
GDPR	General Data Protection Regulation
IEEE	Institute of Electrical and Electronics Engineers
LLM	Large Language Model
ML	Machine Learning
PIPL	Personal Information Protection Law of the People's Republic of China
ROC	Receiver Operating Characteristic
SDV	Synthetic Dateset Vault
SSI	Self-sovereign Identity
TCID	Technical Communication and Information Design
TPR	True Positive Rate
VC	Verifiable Claim

List of Figures

List of Tables

Listings

Appendix A

Contents of the Repository

The code repository contains the following content: The project consists of mainly three parts at the moment, including backend smart contracts, frontend React application and Flask application.

Installation

We need to install hardhat and Flask for Credchain. In order to use SDV, we need to configure the environment separately.

Operation

First, we need to compile and deploy smart contracts to the testnet. To do this, make sure Hardhat is installed on your environment or install using npm install –save-dev hardhat command. Once installed we can deploy contracts.

To deploy, start the hardhat node with npx hardhat node.

Open another terminal window and deploy the contracts using npx hardhat run –network localhost scripts/deploy.js command.

Once the contracts deployed you can see the contract address in the node terminal (e.g., 0x5fbdb2315678afecb367f032d93f642f64180aa3).

Since we are deploying few contracts, each one of them will have different address. Right now the deploy.js file contains all the deployment scripts that will be needed, but part of it is commented out so the only contract that is being deployed right now is DID Registry.

Now we can start the frontend app. To do this call the npm start from the frontend directory. The main logic is in /scr/App.js file and contains a simple app that interacts with deployed DID registry.

To make sure the frontend can access the deployed contracts, for each contract we need to compiled file to the /scr/utils directory. At the moment the only compiled contract is represented by DID.json.

For each deployed contract we also need to add contract ABI and its address in /src/utils/constants.js file. Follow the same pattern as for DID Registry.

Then we can start the Flask application. `python checker.py`