Pay Attention to Bitcoin: Utilizing Attention Mechanism with Lightning Network Information for Change Address Detection

by

Mikaeil MAYELI FERIDANI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN SOFTWARE ENGINEERING
M.A.Sc.

MONTREAL, NOVEMBER, 03, 2024

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Kaiwen Zhang, Thesis supervisor
Department of Software Engineering and IT, École de technologie supérieure

Mr. Abdelouahed Gherbi, Chair, Board of Examiners
Department of Software Engineering and IT, École de technologie supérieure

Mr. Roberto Lopez Herrejon, Member of the Jury
Department of Software Engineering and IT, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON OCTOBER, 28, 2024

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my deepest gratitude to my supervisor, Prof. Kaiwen Zhang, for his unwavering guidance and support throughout my M.Sc. program. His mentorship has been invaluable, providing me with the knowledge, experience, and inspiration necessary to navigate the challenges of academic research. His commitment to high-quality research and his insightful advice have been instrumental in shaping my work over the past two years. My sincere thanks go out to him for his immense support and patience during my master's studies.

I would also like to express my heartfelt appreciation to my wife, Masoumeh, who has been my pillar of strength throughout this journey. Her unwavering support, understanding, and encouragement have been invaluable to me. She has stood by me through every challenge, and I am deeply grateful for her love and sacrifices.

Finally, I would like to thank my friends, Ali, Amin, Hafez, Hossein, Mahdi, Michael, Shahin, and Sophia, who have been a source of encouragement and comfort during difficult times. Their companionship and support have helped me overcome the rough patches, and I am truly thankful for their friendship.

Faites attention à Bitcoin : Utilisation du mécanisme d'attention avec les informations du Lightning Network pour la détection des adresses de change

Mikaeil MAYELI FERIDANI

RÉSUMÉ

Alors que la popularité du Bitcoin a augmenté, sa scalabilité a été mise à l'épreuve par la capacité limitée des blocs, entraînant une augmentation des frais de transaction et diminuant son adéquation pour les microtransactions. Le Lightning Network (LN), introduit par Poon et Dryja comme solution de seconde couche, offre des transactions quasi instantanées avec des frais minimaux, ce qui en fait une alternative attrayante pour les utilisateurs de Bitcoin. Cependant, à mesure que l'adoption du LN augmente — sécurisant plus de 336 millions de dollars en Bitcoin — des préoccupations surgissent quant à la possible compromission de l'anonymat des utilisateurs en raison de la transparence des canaux publics du LN. Les recherches existantes se sont principalement concentrées sur la confidentialité au sein des réseaux LN ou Bitcoin indépendamment, laissant inexploitées les implications de la confidentialité inter-couches.

Cette étude examine l'impact de l'activité du LN sur l'anonymat des utilisateurs de Bitcoin, en se concentrant spécifiquement sur la détection des sorties de monnaie (change outputs) dans le réseau Bitcoin. Nous analysons la distribution des entrées et sorties de transactions, en comparant les transactions Bitcoin standard avec celles liées à la création de canaux LN. En utilisant un modèle modifié de Représentations Encodeur-Décodeur Bidirectionnelles de Transformateurs (BERT), nous améliorons notre capacité à identifier les sorties de monnaie, atteignant un taux de succès de 93,9% — une amélioration par rapport à la ligne de base. Nos résultats révèlent des risques significatifs pour la confidentialité posés par les interactions inter-couches entre les réseaux LN et Bitcoin, démontrant que les données du LN peuvent être utilisées efficacement pour désanonymiser les transactions Bitcoin.

Cette recherche fournit des perspectives critiques sur les implications en matière de confidentialité du Lightning Network et jette les bases pour une exploration plus approfondie des risques de désanonymisation inter-couches dans les systèmes de blockchain.

Mots-clés: Bitcoin, Détection des adresses de change, Mécanisme d'attention

Pay Attention to Bitcoin: Utilizing Attention Mechanism with Lightning Network Information for Change Address Detection

Mikaeil MAYELI FERIDANI

ABSTRACT

As Bitcoin's popularity has surged, its scalability has been challenged by limited block capacity, leading to increasing transaction fees and diminishing its suitability for microtransactions. The Lightning Network (LN), introduced by Poon and Dryja as a second-layer solution, offers near-instantaneous transactions with minimal fees, making it an attractive alternative for Bitcoin users. However, as LN adoption grows—securing over $336 million in Bitcoin—concerns arise about the potential compromise of user anonymity due to the transparency of public LN channels. Existing research has primarily focused on privacy within either the LN or Bitcoin networks independently, leaving the cross-layer privacy implications underexplored.

This study investigates the impact of LN activity on Bitcoin user anonymity, specifically focusing on the detection of change outputs in the Bitcoin network. We analyze the transactions and their inputs and outputs, comparing standard Bitcoin transactions with those linked to LN channel creation. Leveraging a modified Bidirectional Encoder Representations from Transformers (BERT) model, we enhance our ability to identify change outputs, achieving a 93.9% success rate—an improvement over the baseline. Our findings reveal significant privacy risks posed by cross-layer interactions between the LN and Bitcoin networks, demonstrating that data from the LN can be effectively used to deanonymize Bitcoin transactions.

This research provides critical insights into the privacy implications of the Lightning Network and lays the groundwork for further exploration of cross-layer deanonymization risks in blockchain systems.

Keywords: Bitcoin, Change address detection, Attention mechanism

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Page

LIST OF ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
LN	Lightning Network
PoW	Proof-of-Work
UTXO	Unspent Transaction Output
RFC	Random Forest Classifier
Multisig	Multi-signature

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

BTC	Bitcoin
sats	Satoshis
USD	United States dollar

INTRODUCTION

Since the invention of Bitcoin, numerous studies have sought to challenge the anonymity of Bitcoin users and understand the extent to which transactions and users are truly anonymous within the network. In the early stages, most research focused on analyzing Bitcoin user behavior and developing heuristics that allowed observers to cluster addresses controlled by the same entity, thereby identifying total balances associated with individual users. Notable works in this area include Ron & Shamir (2013), Harrigan & Fretter (2016), Meiklejohn *et al.* (2013), and Androulaki, Karame, Roeschlin, Scherer & Capkun (2013), among others.

As research in this field progressed, the limitations of heuristic-based clustering became apparent. New wallets and software were developed with these heuristics in mind. With the introduction of wallets like Samourai Wallet (Wallet (2022b)) and services such as Ricochet (Wallet (2022a)), StoneWall (Wallet (2022c)), and Whirlpool (Wallet (2022d)), the need for more complex models became evident. In response, researchers began utilizing machine learning models to detect address clusters, identify change addresses, and enhance deanonymization efforts. Significant contributions in this area include Ramos Tubino, Robardet & Cazabet (2022), Möser & Narayanan (2022), and Najjar, Naha, Feridani, Dekhil & Zhang (2024), among others.

Meanwhile, Bitcoin itself has evolved, with new features being added to its network. One such innovation is the Lightning Network (Poon & Dryja (2015)), a layer-2 solution that employs state channel technology to enable Bitcoin users to make (almost) instant payments with minimal to zero fees. Since its introduction, the Lightning Network has been widely adopted by Bitcoin users for micro-transactions. This adoption is reflected in the increasing amount of Bitcoin locked in the network, as shown in Figure 0.1.

The anonymity of the Lightning Network has also been the subject of multiple studies. Notable examples include Kappos *et al.* (2021), Herrera-Joancomartí, Navarro-Arribas, Ranchal-Pedrosa, Pérez-Solà & Garcia-Alfaro (2019), and Von Arx, Tran & Vanbever (2023).

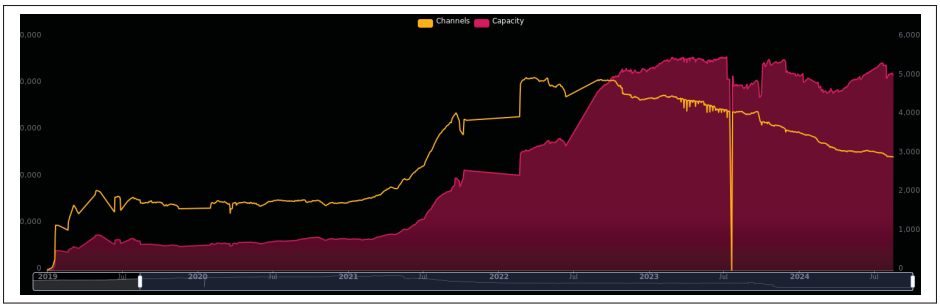

Figure 0.1 The capacity of the Lightning Network has been consistently rising, reaching nearly 350 million USD (exchange rate of 67,900 USD/BTC Taken from the CoinGecko, (2024))
Source: https://www.coingecko.com
Source: Mempool.space website (https://mempool.space/)

Despite these advances, there has been little research exploring the intersection of these two networks to examine the potential effects of the Lightning Network on Bitcoin user anonymity. A notable attempt was made by Romiti *et al.* (2021), who investigated the behavior of Lightning Network users on the Bitcoin network and introduced patterns that successfully clustered addresses on the Bitcoin network based on Lightning Network user behavior. Nonetheless, we believe that due to Lightning Network widespread usage there is still significant potential in this area, and our research aims to shed light on some of the potential effects of Lightning Network usage on Bitcoin user anonymity.

0.1 Research Motivation

Bitcoin transactions inherently possess a chronological order, which can be utilized to sequence these transactions effectively. The recent advancements in transformers, particularly their ability to manage complex, inherently sequential data, inspired us to explore their potential in detecting change outputs that could be used to cluster addresses controlled by a single entity. The most relevant research utilizing transformers for blockchain network transaction analysis

is Hu *et al.* (2023), where the authors employed a modified version of BERT (Devlin, Chang, Lee & Toutanova (2019)) to categorize and flag illicit transactions.

In our research, we aim to investigate the effects of Lightning Network usage on the detection of change output addresses and provide a proof-of-concept to assess the potential effectiveness of transformers in deanonymization techniques within the Bitcoin blockchain. In addition, we explore the potential application of transformers in Bitcoin deanonymization. We expect that a bidirectional encoder-only transformer is an effective machine learning model for recognizing and identifying patterns in transaction trees to detect change outputs, due to transformers' capability with time-series data and to pick up on complex patterns in transactions history.

0.2 Contributions

This research offers the following contributions:

- A comparative analysis of input and output characteristics between Lightning Network Funding transactions and standard Bitcoin transactions
- An assessment of the effectiveness of transformer models (particularly BERT) in identifying change outputs within Bitcoin transactions
- An exploration of the potential benefits of integrating data from Lightning Network nodes with Bitcoin transaction information to improve deanonymization methods

0.3 Thesis Organization

In the following sections, Chapter 1 provides the necessary background to understand the content of this thesis, along with a brief overview of related research in Bitcoin deanonymization, the Lightning Network, and cross-layer analysis. Chapter 2 then introduces our proposed solution, beginning with data gathering and analysis, followed by labeling, model implementation, and

parameter selection. In Chapter 3, we present our findings, compare them with baseline results, and conclude by summarizing our work and discussing potential directions for future research.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we start by reviewing the foundational knowledge necessary for understanding the material discussed in this thesis, followed by an exploration of the relevant literature.

1.1 Background

In this section, we begin by providing an overview of blockchain technology, introducing the fundamental concepts necessary for understanding this thesis. We then explain the properties of Bitcoin, detailing its transaction format, the concept of change outputs, and the multi-input heuristic. Following this, we explore the Lightning Network, a off-chain solution aimed at improving Bitcoin's scalability and privacy and analyze its architecture. Additionally, we examine transformers, with a focus on BERT, an encoder-only transformer that will be utilized in subsequent chapters.

1.1.1 Blockchain

Blockchain is an electronic cash system that is peer-to-peer and was introduced in 2009 by Satoshi Nakamoto (Nakamoto (2009)). In this system, each coin is defined as a chain of digital signatures (also known as transaction outputs or unspent transaction outputs), and a timestamp server records a hash of a block consisting of newly created transaction outputs. Each block contains a reference to its preceding block, confirming its existence and the sequence of transactions (hence the name "blockchain"). To prevent Sybil attacks on the network, Nakamoto employed the proof-of-work (PoW) mechanism, introduced in (Back (2002)). In Bitcoin, PoW requires participating nodes that wish to generate blocks to find a value called a nonce. This nonce combined with the block that is about to get generated must produce a hash that is in a determined subset. This subset is defined by the number of leading zeros at the beginning of the hash. Values that have an equal or smaller number of leading zeros, in the beginning, are in

the acceptable range, otherwise, they are rejected. This process is known as mining, and the participating systems are called miners.

1.1.2 Bitcoin

Bitcoin (Nakamoto (2009)) was the first use of blockchain technology and has remained the predominant cryptocurrency since its inception. With over 68,000 nodes participating in the network (Dashjr (2024)), Bitcoin is one of the most decentralized networks among cryptocurrencies. These nodes include full nodes, which maintain the entire transaction history in blocks dating back to 2009, and pruned nodes which only keep the header of the all blocks along with the transactions in a few latest generated blocks. In Bitcoin, each transaction (except coinbase transactions) consists of one or more inputs that were originally generated as outputs of previous transactions. These inputs are fully spent in each transaction, meaning there are no partial expenditures, and one or more new outputs are generated each time a coin needs to be partially spent. The newly created outputs are known as Unspent Transaction Outputs (UTXOs). For inputs to be valid in a transaction, they must be UTXOs at the time of spending. Due to the linkage between inputs and outputs, each UTXO can be traced back to coinbase transactions. Coinbase transactions are unique in that they generate new outputs without any inputs. These transactions are created in each block and contain the reward for the block's miner.

1.1.2.1 Change Output

As previously mentioned, each input in Bitcoin transactions is spent in its entirety every time it is used. However, there are transactions where the input value exceeds the required payment. For instance, imagine Alice wants to pay Bob 0.025 Bitcoin but has UTXOs worth 0.01 and 0.02, totaling 0.03 Bitcoin. If Alice combines her UTXOs and sends the entire amount to Bob, she would overpay him by 0.005 Bitcoin. To avoid this, Alice creates two UTXOs instead of one: one UTXO is sent to Bob with the required 0.025 Bitcoin, and the other UTXO, containing 0.005 Bitcoin, is sent back to Alice. This second UTXO is known as a Change Output. It functions similarly to receiving change after making a payment with cash. Figure 1.1 illustrates

this example. Since Change Outputs are controlled by the same entity, their detection can have serious implications for the transactor's anonymity. Identifying Change Outputs can enable an observer to link each UTXO to its owner, potentially revealing the owner's balance.

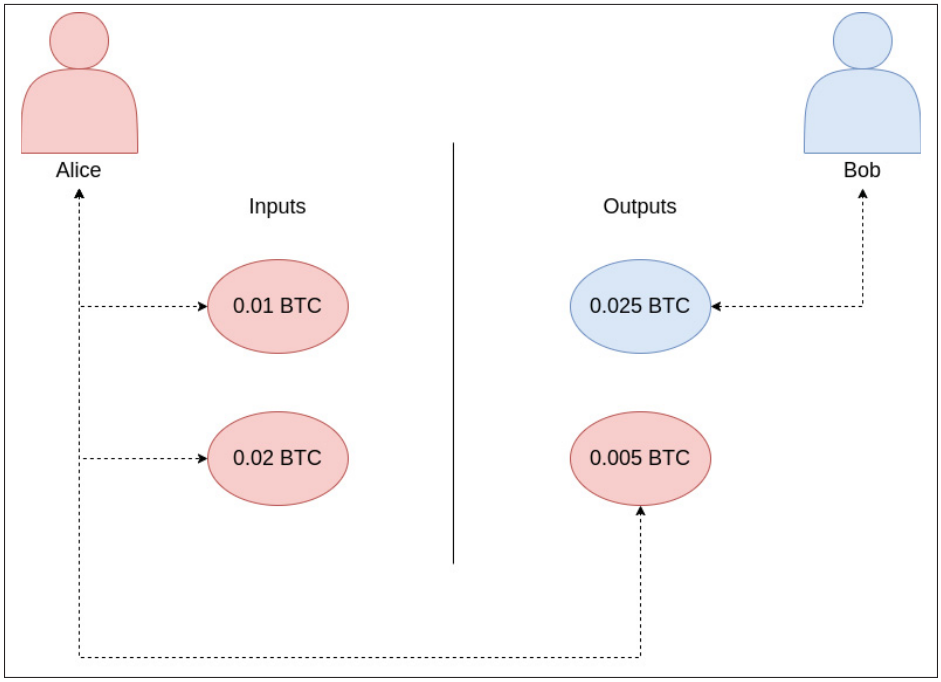

Figure 1.1 In this transaction, Alice uses her two UTXOs to create a new transaction, She sends 0.025 BTC to Bob and sends the remaining 0.005 BTC back to herself, The red output represents the Change Output

1.1.2.2 Multi-input Heuristic

The multi-input heuristic (Androulaki *et al.* (2013)) states that if a transaction has multiple inputs, they are likely controlled by the same entity. This heuristic has been extensively analyzed in various studies and has proven to be one of the most accurate heuristics for clustering Bitcoin addresses (Reid & Harrigan (2013), Androulaki *et al.* (2013), Harrigan & Fretter (2016), Meiklejohn *et al.* (2013), Ron & Shamir (2013)). In the example we explored in Figure 1.1, according to this heuristic, all of the inputs are controlled by the same entity, which, in this case, is indeed true. Later in Chapter 3, we will use this heuristic as one of the basis for our data labelling.

1.1.3 Lightning Network

The Lightning Network(Poon & Dryja (2015)) is a network of payment channels built on top of the Bitcoin network. The popularity of LN is due to its relatively low routing fees (almost zero) and instantaneous payments, all while maintaining Bitcoin as the currency. These features make the Lightning Network a suitable choice for users who wish to make small, instant transactions in BTC, especially when compared to the main Bitcoin network, where transaction fees can fluctuate up to 120 USD per transaction (Figure 1.2) and block generation times average 10 minutes. To understand how an LN channel is set up between two parties, let's imagine Alice

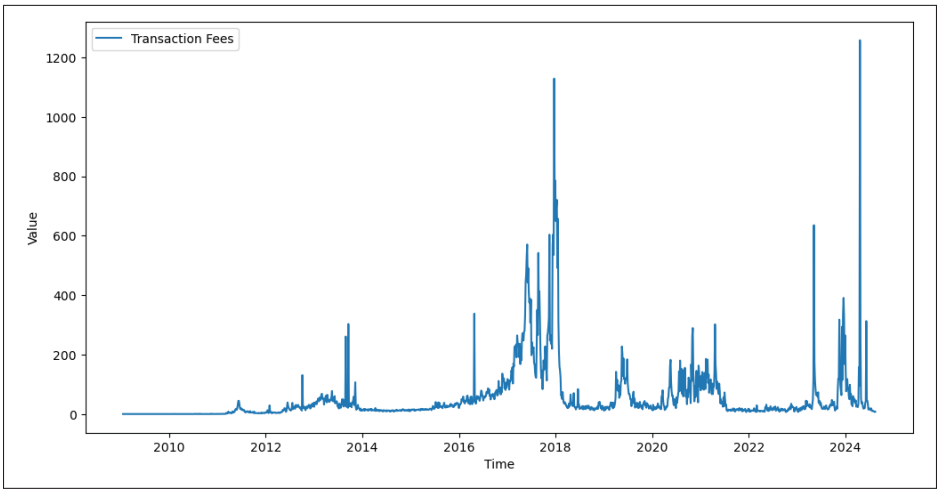

Figure 1.2 Transaction fees over time in Bitcoin—the value represented is in BTC

and Bob. Alice wants to create a Lightning Channel with Bob, with a capacity of 0.05 BTC. To initiate a Lightning Channel, Alice first needs to create a 2-of-2 multisignature address.

1.1.3.1 Multisignature Transactions

Bitcoin uses a stack-based scripting system for locking and unlocking transactions called Bitcoin Script. Bitcoin Script does not include loops and is not Turing-complete. Though limited, this scripting mechanism enables developers to devise various types of transactions, such as n-of-m multisignature transactions. In n-of-m multisignature transactions, n-signatures are required out

of m possible signatures to spend a transaction output. This allows Bitcoin users to lock their funds mutually. The Lightning Network uses 2-of-2 multisignature transactions as the basis for forming channels.

1.1.3.2 Channel Establishment

After Alice creates this 2-of-2 multisignature address with Bob, she generates a transaction called the Funding transaction, which is the first step in creating a channel between Alice and Bob. Before broadcasting the Funding transaction, however, Alice creates another transaction called the Refund transaction that spends the UTXO created by the Funding transaction. This ensures that if Bob becomes unresponsive after the announcement of the Funding transaction, Alice can still recover her funds. Before announcing the Funding transaction to the Bitcoin network, Alice obtains a signed version of the refund transaction from Bob and withholds it without publishing it. Once the refund transaction is secured, the Funding transaction is announced to the Bitcoin network, and the Lightning Channel between Alice and Bob is initiated (Figure 1.3). In the

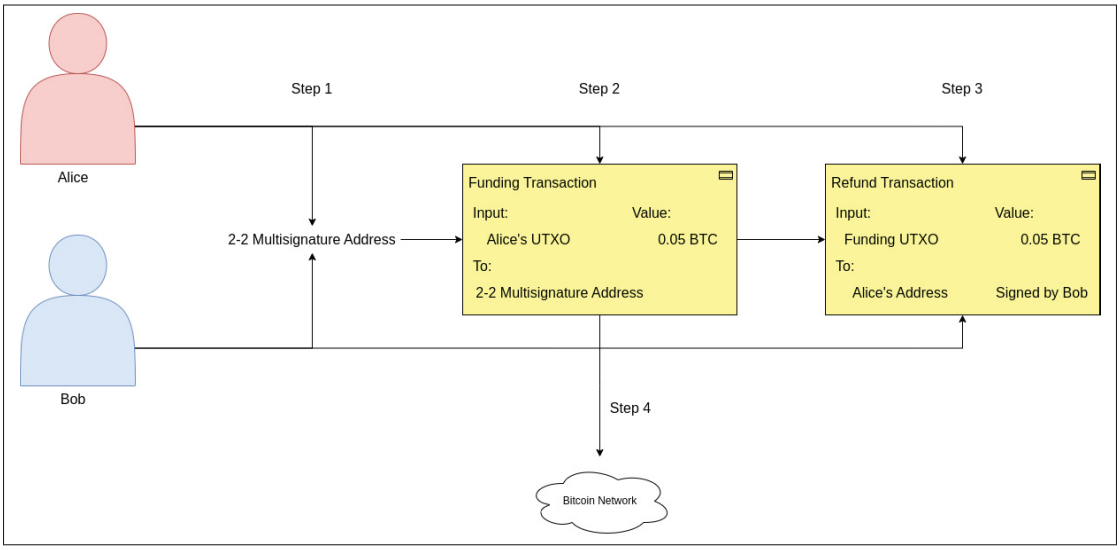

Figure 1.3 For creating a Lightning Network Channel: in Step 1, Alice and Bob create a 2-of-2 multisignature address; in Step 2, the Funding transaction is created; in Step 3, the Refund transaction is created; and in Step 4, the Funding transaction is announced to the Bitcoin network

Lightning Network, there are two types of channels:

- Public Channels: Public channels announce their existence to the network, including information such as channel capacity, routing fees, and methods to reach the node (using an IP address or the TOR network).
- Private Channels: Private channels do not advertise their existence to the Lightning Network. Although anyone can see that a 2-of-2 multisignature transaction has been created on the Bitcoin network, there is no apparent way to distinguish between a regular 2-of-2 multisignature transaction and one used to create a Lightning channel.

Once an active channel is established, Alice and Bob can begin sending and receiving transactions.

1.1.4 Transformers and Attention Mechanism

In 2017, Vaswani *et al.* (2017) introduced the Transformer architecture, revolutionizing the field of natural language processing. The Transformer model uses the attention mechanism to identify and utilize relationships between sequences in the input to generate output text. This section provides an overview of the main components of the Transformer architecture: the Embedding layer, Positional Encoding, Encoder block, Decoder block, Linear layer, and Softmax layer.

- Embedding Layer: The Transformer model includes two learned embedding layers—one for the input and one for the output—to convert text into numerical vectors of a fixed size. This transformation is essential for making the input suitable for processing within the Transformer architecture.
- Positional Encoding: Unlike recurrent neural networks (RNNs), Transformers do not inherently capture the sequential order of input data. by adding positional encoding to the input embeddings, the model will be able to recognize the relative positions of tokens in the sequence. The positional encoding in the Vaswani *et al.* (2017) is calculated using the following formula:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{1.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{1.2}$$

Here, pos represents the position, and i represents the dimension. This encoding ensures that each position in the sequence has a unique representation.

- Encoder block: The encoder block has a multi-head self-attention mechanism and a feed-forward network. The attention function takes three matrices as input: Query (Q), Key (K), and Value (V), and is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.3}$$

where d_k is the dimension of the key vectors and calculates the relevance of each word in the sequence to the others.

The Query (Q), Key (K), and Value (V) matrices are essential components of the attention mechanism. The Query matrix is obtained by multiplying the input embeddings by a learned weight matrix W_Q:

$$Q = XW_Q \tag{1.4}$$

where X represents the input embeddings.

Similarly, the Key and Value matrices are derived by multiplying the input embeddings by other learned weight matrices W_K and W_V, respectively.

During training, the Transformer model computes the loss, which measures the difference between the predicted output and the actual target output. The gradients of the loss with respect to each parameter in the model, including the weight matrices W_Q, W_K, and W_V, are calculated using backpropagation, and these weights are optimized using an algorithm such as Gradient Descent. As training progresses, the weight matrices W_Q, W_K, and W_V are continuously updated to improve the model's performance.

- Multi-head Attention: Instead of performing a single attention function, the multi-head attention mechanism runs several attention functions in parallel, referred to as heads and the results are concatenated and linearly transformed making the final output. This allows the

model to attend to different parts of the input sequence and capture the relationships between words from different angles.

- Decoder block: The decoder block is similar to the encoder block but includes an additional multi-head attention layer that performs attention over the encoder's output, allowing the model to attend to relevant parts of the input while generating the output.
- Linear and Softmax layer: The final output is passed through a linear transformation followed by a softmax function. The linear layer projects the output onto the vocabulary space, and the softmax function converts these projections into probabilities, effectively generating the predicted output sequence.

In this research, we use BERT (Bidirectional Encoder Representations from Transformers). A Bert is an encoder-only Transformer. BERT models leverage the bidirectional nature of the encoder blocks to capture context from both the left and right sides of a given token, making them highly effective at understanding the nuances and meanings in text. We utilized an encoder-only Transformer to identify relationships between various parts of a transaction in order to detect the change output.

1.2 Related Works

In this section, we review related works in Bitcoin privacy, Lightning Network privacy, cross-layer analysis, and the application of transformers in transaction analysis in blockchain, highlighting the challenges and opportunities within these fields.

1.2.1 Bitcoin Privacy

In Meiklejohn *et al.* (2013), the authors conducted a comprehensive analysis of Bitcoin transaction traceability using heuristic clustering techniques. They examined trends in Bitcoin user behavior over time, focusing on aspects such as the distribution of transaction values and the duration for which public keys retain their bitcoins. To enhance their data collection, the authors performed their own transactions across various services, which allowed them to bootstrap their clustering

efforts. They employed two key heuristics: the multi-input heuristic and the change output heuristic. Using these heuristics, the authors successfully clustered transactions associated with the same service providers. Their findings highlighted that, despite Bitcoin's pseudo-anonymity, there remains a significant gap between the perceived and actual levels of anonymity offered by Bitcoin. This research shows the importance of well-defined heuristics in the deanonymization process within the Bitcoin network. The multi-input heuristic examined in this paper sets the foundation for one of the heuristics that we use in data labeling procedure, which is discussed thoroughly in Chapter 2.

Harrigan & Fretter (2016) provided a thorough examination of why simple address clustering heuristics, particularly the multi-input heuristic, are effective in practice for Bitcoin transaction analysis. The authors quantified several key factors contributing to this effectiveness, including high levels of address reuse, avoidable merging of addresses, and the existence of "super-clusters" with high centrality in the transaction graph. By analyzing Bitcoin's first seven years of transactions, they demonstrated that address clustering typically results in incremental growth of clusters over time, with large clusters rarely merging. This stability benefits blockchain analysts by allowing for consistent results across time periods. The authors also highlighted how the presence of super-clusters, often associated with major Bitcoin services, enables the identification of a significant portion of on-chain activity. Though in recent years, many wallets have implemented features to implicitly or explicitly protect their users against the conditions that were effective for deanonymization in this research. Most standard wallets today generate a new address for the change output to prevent address reuse, and privacy-oriented wallets like Samourai Wallet (Wallet (2022b)) have features like batch spending or PayNyms to enhance user anonymity. Our research aims to overcome these challenges by utilizing an encoder-only transformer and integrating Lightning Network information with Bitcoin transactions to improve the information available to the model.

Möser & Narayanan (2022) aimed to enhance the accuracy and reliability of Bitcoin address clustering techniques. They developed a new ground truth dataset of transactions with known change outputs, allowing for the precise evaluation of various heuristics. By leveraging this

dataset, they introduced a Random Forest Classifier that significantly outperformed traditional heuristic methods, achieving higher accuracy with lower false positive rates. This study highlights the crucial role of advanced machine learning techniques and robust datasets in improving deanonymization processes within the Bitcoin network. As a baseline for our research, we utilize an RFC to compare our model's performance. Additionally, the dataset gathered by the authors is limited to what they call standard transactions, in which each transaction have exactly two outputs. In our research, we addressed this limitation by gathering a broader range of transactions, with one to three outputs, to train our model.

Najjar *et al.* (2024) proposed a novel approach to identifying change addresses in Bitcoin transactions, addressing a significant challenge in blockchain analysis. The authors utilized hierarchical clustering combined with multi-input heuristic to enhance the accuracy of change address detection. Their approach began with a data extraction method that traced transactions back to their origin, optimizing the clustering process by focusing on connected transactions rather than the entire blockchain. The hierarchical clustering algorithm was then applied at the transaction level, where similarities between inputs and outputs were analyzed to identify change addresses. The study demonstrated that this method significantly improved the precision and recall of change address detection, achieving better results than existing proprietary solutions. Although they achieved a 75% accuracy rate, the clustering technique was applied to a single transaction, preventing the model from utilizing potential patterns in prior transactions that generated the inputs. In our method, we gather parent transactions up to a depth of three to address this limitation.

In Ramos Tubino *et al.* (2022), authors presented a new approach to improve the identification of Bitcoin actors by leveraging supervised machine learning techniques. They tackled the challenge of accurately grouping Bitcoin addresses, which typically represent individual actors, into aggregates that can be more reliably associated with unique entities. They introduced two key methods: one for discovering change addresses using supervised learning and another for improving the aggregation of addresses associated with specific actors. Their approach involved creating a labeled training dataset using a combination of reliable heuristics and external

information sources, allowing them to train machine learning models that could more accurately identify change addresses and actor-specific address aggregates. The two models that produced the best results, RFC and XGBoost, are the ones we later use as baselines to compare our model with. Additionally, our dataset does not focus on a specific actor but can be applied to all Bitcoin transactions.

1.2.2 Lightning Network Privacy

In their study Kappos *et al.* (2021) conducted an examination of the privacy in LN. The authors identified several key privacy vulnerabilities in LN by devising various attacks that exploit publicly available information about the network. These attacks were capable of uncovering sensitive information such as channel balances and the identities of transaction participants, which were intended to remain private. Their research highlights significant privacy gaps in LN, showing that while the network promises enhanced privacy over Bitcoin, it still exhibits considerable vulnerabilities that can be exploited. In Von Arx *et al.* (2023), authors introduced a passive privacy attack on Lightning Network. Unlike previous attacks that required active participation in LN or collusion among many nodes, and were ineffective against single-hop payments, the Revelio attack presents a passive network-level attack that is effective against mutli-hop and single-hop payments. The study demonstrates that a small number of ASes can observe a significant portion of LN traffic, allowing them to infer the sender, receiver, and payment amounts with high accuracy. Through simulations and real-world LN data analysis, the authors show that up to 80% of LN traffic can be monitored by just five ASes, leading to the potential deanonymization of about one-third of the payments. These network-level attacks on the Lightning Network, combined with information available through cross-layer deanonymization techniques, like our research, have the potential to expose channels, balances, nodes, Bitcoin addresses, and other related information, such as IP addresses of Lightning Network and Bitcoin users, posing significant privacy concern.

1.2.3 Cross-Layer Analysis

Romiti *et al.* (2021) introduce an approach to understanding privacy vulnerabilities in the Lightning Network (LN) by linking on-chain Bitcoin transactions (Layer 1) with off-chain LN activities (Layer 2). They demonstrated how LN nodes can be linked to Bitcoin addresses, enabling cross-layer deanonymization. The study presents four clustering heuristics to group Bitcoin addresses and LN nodes based on their interaction patterns and shared characteristics such as IP addresses and aliases. Through these methods, the authors successfully link 45.97% of LN nodes to 29.61% of Bitcoin addresses involved in the LN, revealing that as few as five actors control over 33% of LN's total capacity, which poses severe risks to the network's security and privacy. The most prominent pattern in their on-chain clustering is the "Snake Pattern", in which an entity uses the change address to open another channel multiple times. This is one of the heuristics that we used in our data labelling procedure, discussed in Chapter 2.

1.2.4 Transformers for Transaction Analysis

Hu *et al.* (2023) introduced a new approach to enhancing fraud detection on the Ethereum blockchain using a pre-trained transformer model named BERT4ETH. The authors address the limitations of existing graph-based fraud detection methods, which struggle to capture the highly repetitive, skewed, and heterogeneous nature of Ethereum transactions. BERT4ETH is designed to model the sequential patterns inherent in Ethereum transactions and to generate robust and expressive account representations, making it versatile for various fraud detection tasks such as phishing account detection and deanonymization. This research shows the potential of transformers for deanonymization purposes on blockchains. In our research, we also utilize a modified BERT to detect change address.

CHAPTER 2

PROPOSED SOLUTION

In this chapter, we begin by analyzing LN Funding transactions and comparing them with regular Bitcoin transactions. We then explain how we prepared and labeled our dataset, and conclude with details on our model design and implementation.

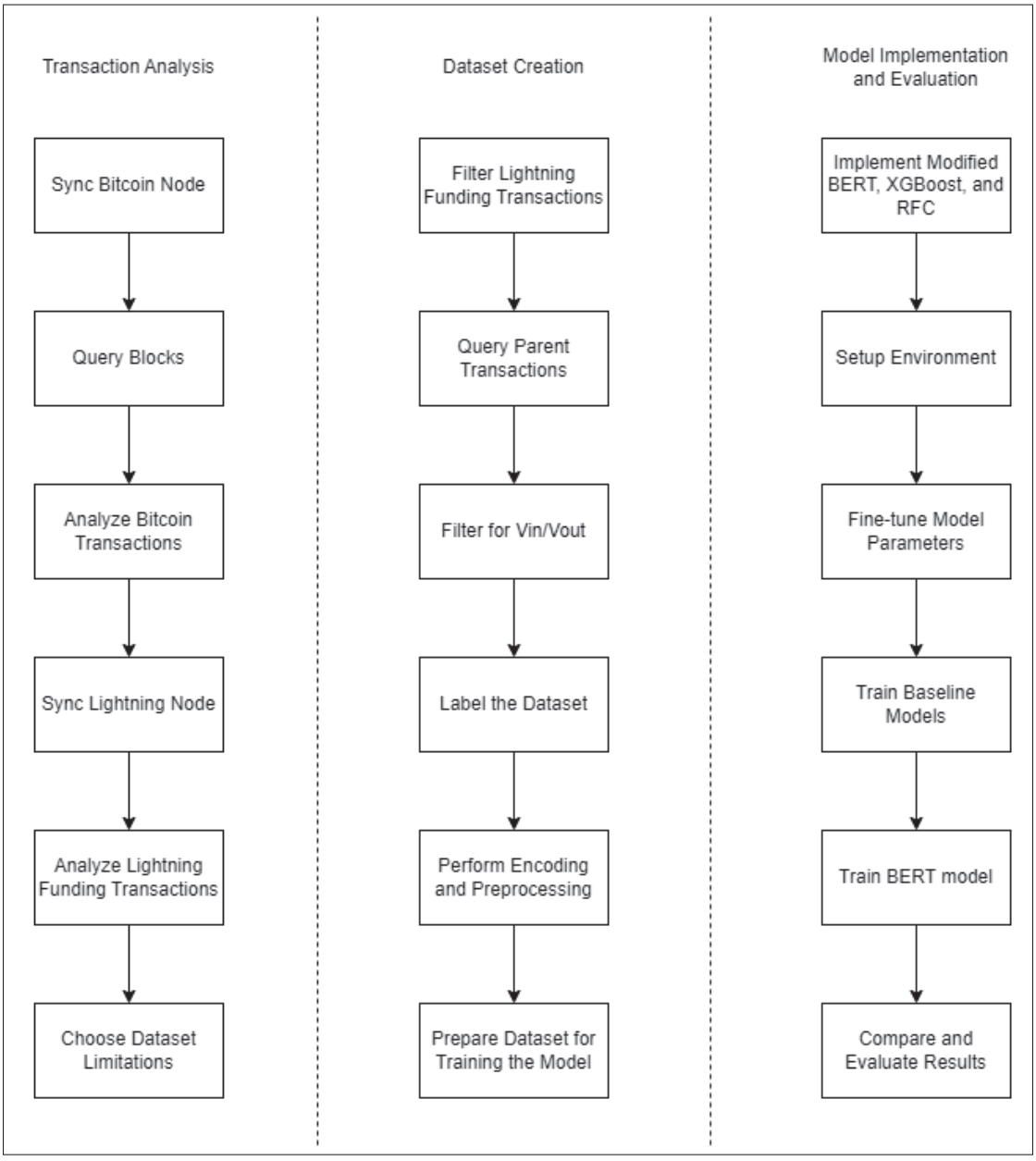

Figure 2.1 Flowchart of the Proposed Solution

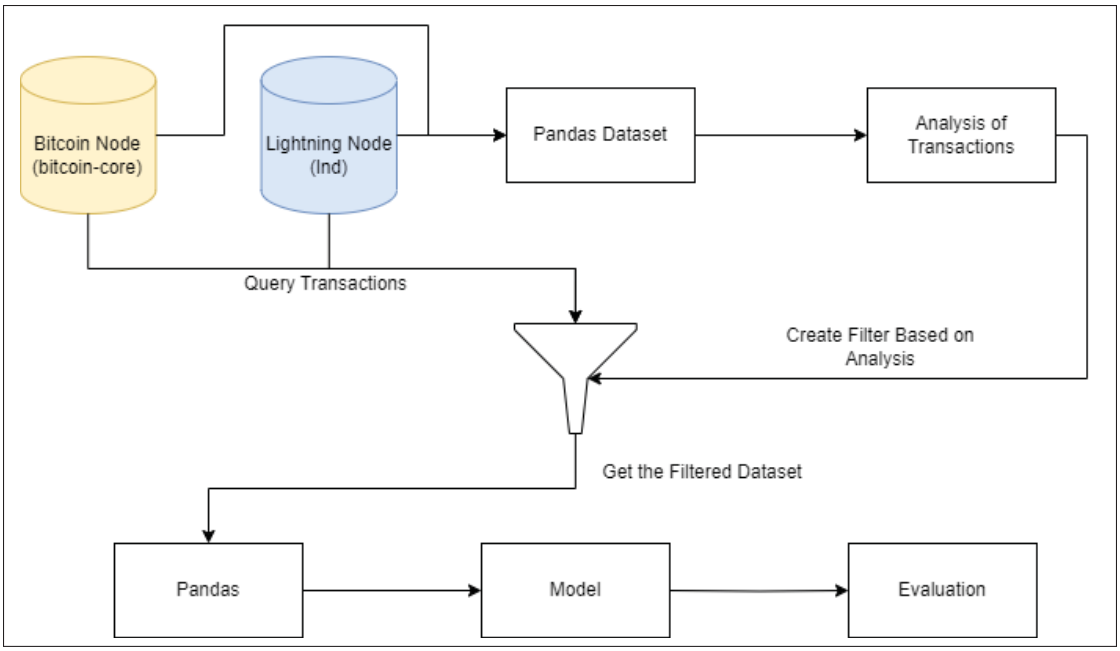

Figure 2.2 This figure shows main components of the proposed solution, such as data sources (Bitcoin and Lightning nodes), data containers and data flow to the models

2.1 Transaction Analysis

Our main resources for gathering our data were bitcoin-core (BitcoinProject (2024)) and LND (LightningLabs (2024)). Bitcoin-core is an implementation for Bitcoin nodes that connect to the Bitcoin network, downloads and validates the blocks since the beginning, and has an RPC API that can be used to make queries from the blocks. Using a fully synced Bitcoin node, we queried blocks 828,577 to 829,577. Block 829,577 was the latest block at the time we began our analysis, so we chose it as our endpoint and worked backward 1,000 blocks for our data range. We do not expect significant changes in results if the queried blocks are relatively recent. We also had to limit ourselves to 1,000 blocks due to computational constraints. First we developed a wrapper for $python - bitcoinrpc$. This enabled us to perform the following procedure for getting the transactions:

1. Get the latest block count using $getblockcount$
2. Get the blockhashes of the last 1000 blocks (828,577 to 829,577) using $getblockhash$

3. Get the transactions of a Bitcoin block by its hash using $getblock$ with the verbosity set to 3rd level ($verbosity = 2$)

The process can be seen in Figure 2.3. After querying each block, the transaction IDs, number

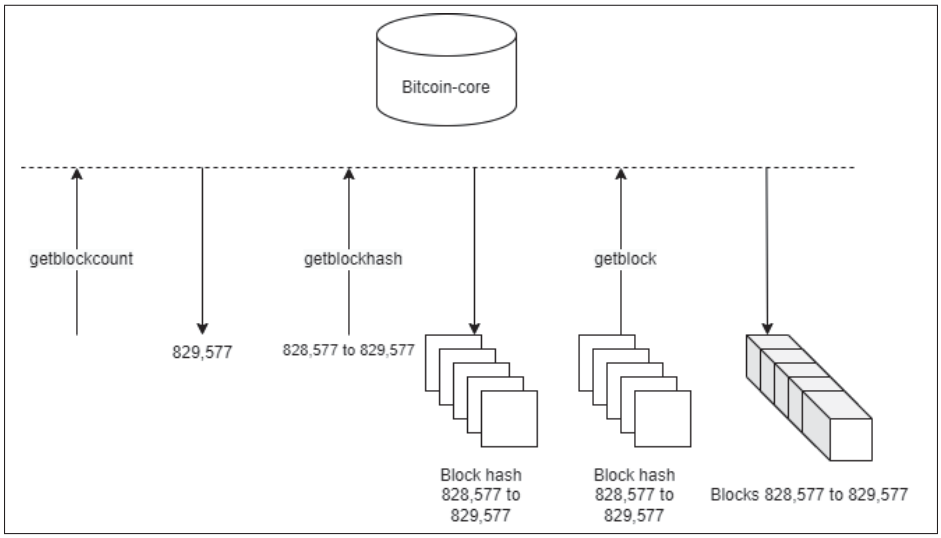

Figure 2.3 The initial transaction gathering procedure is illusterated here, each query to bitcoin-core software is displayed by an arrow towards bitcoin-core, and the result is the arrow that immediately follows the query from bitcoin-core to a value

of *vin*s and *vout*s were added to a Pandas DataFrame which were later on used for performing analysis which resulted in gathering and analyzing 2,257,416 transactions.

The initial analysis of the dataset was on the number of inputs and outputs of the transactions. This property was especially important, because we faced two important constraints:

1. The higher the number of inputs/outputs, a more complex model was required. Since in our algorithm (described in Section 2.2), we used an iterative approach to collect and label change transactions in a graph of transactions, setting a large number on the limit of input/output could result in very large datasets and require a complex model, and a very small limit on input/output could result in a very simple model that cannot handle most transactions and is incapable of comparison to real-world transactions,

2. Another issue that could arise was since we use heuristics to label our dataset, a higher number of inputs and outputs might lead to an increase in the heuristic's inaccuracy. Since we did not have access to any labeled dataset suitable for our needs, we utilized the multi-input heuristic.

Therefore to find a balance between the complexity and the applicability of our model, we decided to perform research on the number of transactions inputs/outputs.

2.1.1 Bitcoin Blockchain analysis

From all the 2,257,416 transactions that we gathered, 2,053,294 transactions had between 1 to 3 inputs and 2,046,083 transactions had between 1 to 3 outputs, and 1,927,338 transactions had between 1 to 3 inputs and outputs. These findings shows that over 85% of transactions in our blocks had between 1 to 3 inputs and outputs. Therefore we decided to check the Lightning channel Funding transactions as well to confirm whether this is a good constraint or not. The most common type of transactions were transactions with 1 input and 1 output or 1 input and 2 outputs (Figure 2.4).

2.1.2 Lightning Funding Transactions Analysis

To analyze the inputs and outputs of the Lightning channel Funding transactions, we fully synced a Lightning node using LND. Then we took a snapshot of all of the active channels of the network using $lnclidescribegraph$, and extracted the channel points using a python script. These channel points had the format of $transaction\_id : index$, in which $transaction\_id$ is the transaction that created the Lightning channel, and $index$ is the output that is locked in the 2-2 multisig contract. Then we used the same script that was used to analyze regular Bitcoin transactions. The result showed that between the 66,810 channels that we analyzed, 55,271 channels were created in transactions with 1 to 3 inputs, 58,974 with 1 to 3 outputs, and 50,563 with 1 to 3 inputs and outputs. For Lightning Funding transactions, the most common type of transactions had 1 input and 2 outputs and 2 inputs and 2 outputs (Figure 2.5).

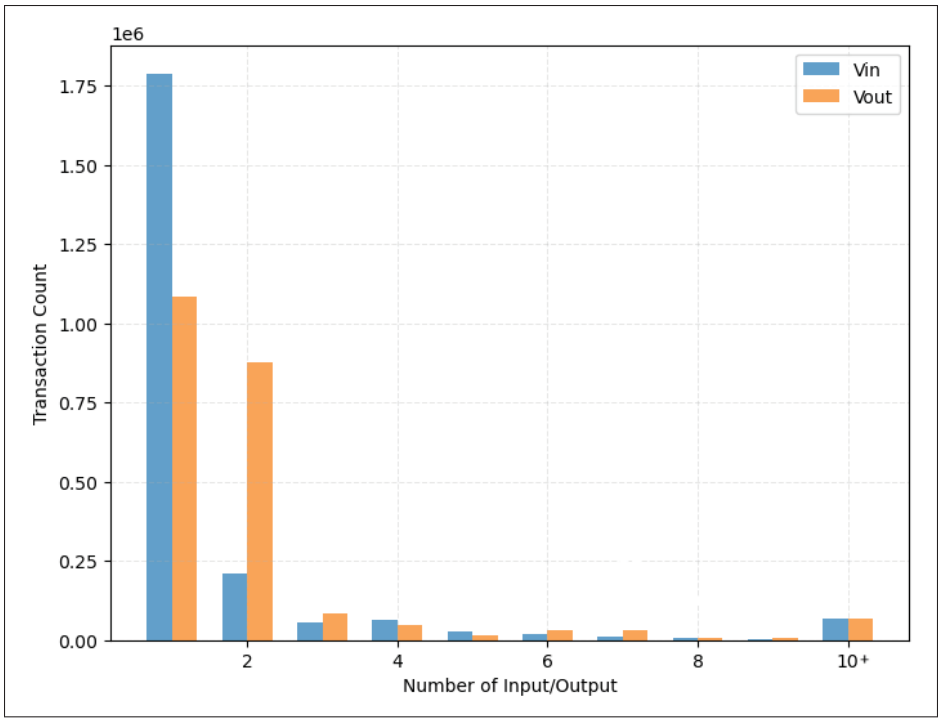

Figure 2.4 Histogram showing the distribution of the number of inputs (Vin) and outputs (Vout) in Bitcoin transactions, the x-axis shows the inputs/outputs count for the transactions, while the y-axis shows the transaction count

Based on this result, we decided to limit the number of inputs and outputs in our dataset to 3, to keep the model input size relatively reasonable, while covering 75% of our initial Funding transaction dataset. We do not expect the overall pattern we observed to change significantly if we query a larger block range, as our findings align with our initial intuitions. With further transaction gathering, we anticipate this pattern will remain consistent.

2.2 Dataset Creation

In this section, we first discuss the procedure we followed to gather data from the Bitcoin network and the Lightning Network. Next, we outline the labeling process used to build our ground truth dataset. Finally, we describe the preparation techniques employed to ready our dataset for modeling.

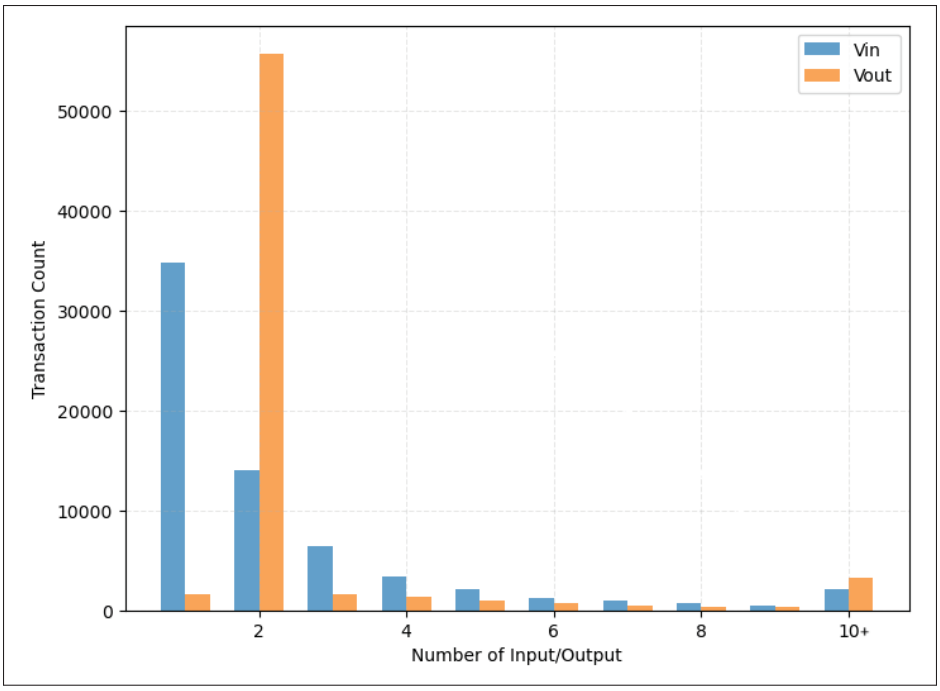

Figure 2.5 Histogram showing the distribution of the number of inputs (Vin) and outputs (Vout) in LN Funding transactions, the x-axis shows the inputs/outputs count for the transactions, while the y-axis shows the transaction count

2.2.1 Data Gathering

Our data gathering procedure began with the collection of LN Funding transactions. We created an array of all LN Funding transaction IDs and iteratively queried the parent transactions for each output spent to create those transactions, continuing this process until we reached a depth of 3 (Figure 2.6). Using this data, we first determined whether the transactions complied with our constraint of a maximum of 3 inputs and 3 outputs. If a transaction did not comply with this constraint, it was removed from our dataset. After gathering all transactions that met the criteria, we created a Pandas DataFrame for each LN Funding transaction and extracted the following fields, which were saved in the corresponding DataFrame:

- is_output: Indicates whether the field is related to an output (true) or an input (false),
- transaction ID: A unique identifier (hash) for the transaction,
- version: Indicates the version number of the transaction format,

- size: The total size of the transaction in bytes,
- vsize: The virtual size of the transaction, taking SegWit into account,
- weight: A metric used in the SegWit protocol that combines the transaction size and the witness data,
- locktime: Specifies the earliest time or block height at which the transaction can be added to the blockchain,
- index: The position of the input or output in the transaction (starting from 0),
- value: The amount of Bitcoin (in satoshis) being transacted,
- addr_type: The type of address involved in the transaction (e.g., P2PKH, P2SH, Bech32),
- is_lightning: Indicates whether the transaction is associated with the Lightning Network (true) or not (false),
- is_change: Indicates whether the output is a change address (true) or not (false) (explained in 2.2.2),
- depth: The depth in which this output/input was extracted

2.2.2 Data Labeling

Data labeling was one of the most challenging steps in our research. A well-labeled dataset is essential to ensure that our results are applicable in real-world scenarios. To maximize the accuracy of our labeling process, we used two of the most prominent heuristics applicable to our case:

2.2.2.1 Assumption No. 1 - Change Output in a Funding Transaction

Our first heuristic is designed to detect change outputs in Lightning Network Funding transactions. The heuristic states that if a transaction is used as a Funding transaction, the output associated with the Lightning channel's 2-of-2 multisignature is considered the change output. A similar pattern, described by Romiti *et al.* (2021), was identified and named the "snake pattern." It was found to be the most effective pattern among Lightning users. However, a key distinction between our heuristic and the Snake Pattern is that the Snake Pattern relies on multiple connected

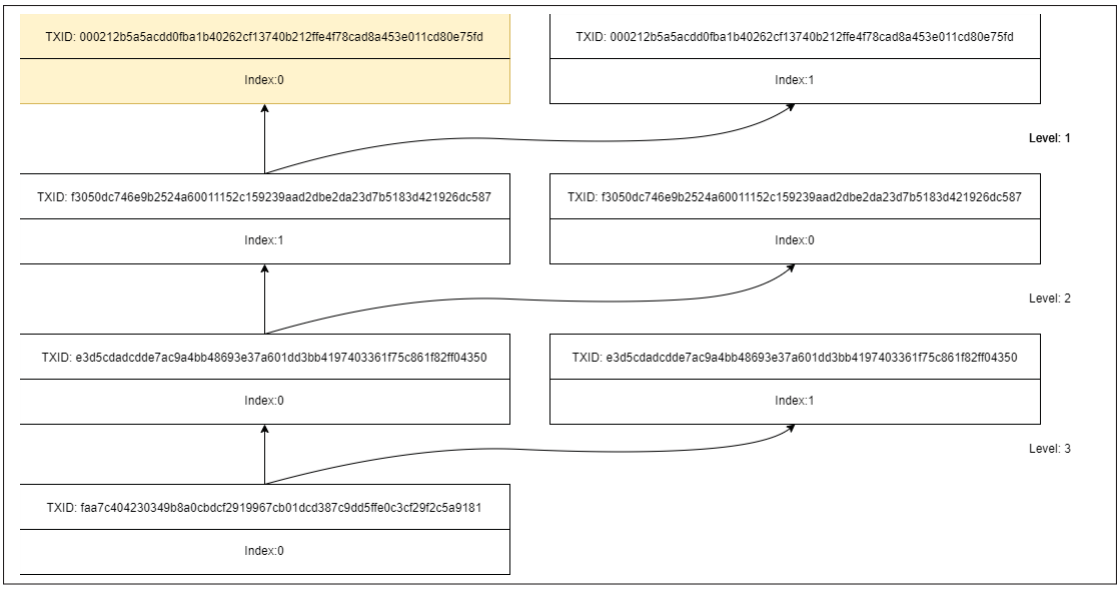

Figure 2.6 An example of a transaction tree generated from an LN Funding transaction, the highlighted transaction output at the top is the LN Funding transaction, serving as the root of the tree, each subsequent transaction is labeled with its transaction ID and index, showing the hierarchical relationship and the depth of each transaction output within the tree structure

transactions, whereas we focus on a single transaction. This allowed us to label more transactions, albeit with a potential increase in incorrect labeling. The formalized version of the heuristic is as follows:

Define F as follows:

$$F := \{f \mid f \text{ a Lightning Funding transaction output}\} \tag{2.1}$$

and define t as a Bitcoin transaction. Suppose O_t is defined as:

$$O_t := \{o_1, \ldots, o_n \mid 1 \leq n \leq 3, o_i \text{ is an output}\}. \tag{2.2}$$

Then, the change output is the output the minimum value in $O_t \setminus F$.

A false positive that might arise from our heuristic occurs if a transaction serves as both a payment and a Lightning Funding transaction. In this case, the selected change output may not actually be a change output and could belong to another entity. Another issue might involve false negatives, where in transactions with three outputs, either both outputs are change outputs, or the larger-valued output is the change output. To understand how these issues affect our dataset, we reviewed our transactions. Among the 29,537 Lightning Funding transactions remaining after applying the input/output constraint over the Funding transactions and its tree, only 163 transactions might have resulted in a false positive, which is less than 0.5% of our dataset.

2.2.2.2 Assumption No. 2 - Multi-Input Heuristic

As discussed in Chapter 1, the multi-input heuristic is one of the most accurate heuristics for clustering addresses in Bitcoin. Using this heuristic, we assumed that if the inputs of a transaction are controlled by the same entity, then the inputs of the previous transaction are also controlled by the same entity. We recursively applied this process to a certain depth (in this case, 3). The formalized version of the heuristic is as follows: Define t_i and t_j as two Bitcoin transactions such that t_j consumes an output of t_i. Let O_{t_i} as the set of outputs for t_i and I_{t_j} as the set of inputs for t_j be:

$$O_{t_i} := \{o_1, \ldots, o_n \mid 1 \leq n \leq 3\} \tag{2.3}$$

and

$$I_{t_j} := \{o_1, \ldots, o_n \mid 1 \leq n \leq 3\} \tag{2.4}$$

Then $o_j \in I_{t_j} \cap O_{t_i}$, if o_j is controlled by the same entity controlling both t_i and t_j, then o_j is t_i's change output.

Unlike the usual application of the multi-input heuristic, which is typically applied to a single transaction, we recursively apply it across a series of connected transactions. This approach may lead to false positives if the connected set of transactions is controlled by different entities, especially in transactions originating from exchanges or services. However, since these services typically involve a large number of inputs and outputs, and we have limited the number of inputs

and outputs, it is unlikely that such transactions are included in our analysis. Additionally, another potential issue arises if a previous input was owned by a different entity. The limit we set on the depth of our recursion helps mitigate some of these risks.

2.2.3 Data Preparation

After indexing the transaction trees and saving each one in a Pandas DataFrame, we applied two sets of transformations:

1. Transformations within individual transaction groups: The first set of transformations was applied to each transaction group separately. This transformation replaced the transaction IDs with unique numerical identifiers, enabling the model to correctly associate inputs and outputs in the proper order. This was done by assigning a unique number to each transaction ID within a transaction list, creating a mapping dictionary from transaction IDs to their unique identifiers, and then replacing the transaction IDs according to this mapping. The pseudocode is provided in Algorithm 2.1.
2. Transformations across all transactions: The second set of transformations was applied across all transactions in the database. First, all Pandas DataFrames were combined, and the following transformations were applied:
   a. The following features had the $\log(1+x)$ transformation applied:
      - $size$
      - $vsize$
      - $weight$
      - $locktime$
      - $value$

      This transformation was necessary because these values can range from very small to very large. Applying the $\log(1+x)$ transformation helps reduce skewness and compress the range of large values. This makes the features more comparable in scale, improving the model's ability to learn from the data.

b. The next step was to define and train transformations for each feature to prepare the data for input into the model. We used two sets of transformations: $OneHotEncoder$ for categorical and alphabetic features, and $MinMaxScaler$ for numerical features. The following categories were OneHotEncoded:

- $txid$
- $version$
- $addr\_type$
- $index$

The following categories were scaled using MinMaxScaler to limit the values between 0 and 1:

- $size$
- $vsize$
- $weight$
- $locktime$
- $value$

After fitting the transformation functions, each individual transaction DataFrame was processed with the necessary transformations.

The algorithm can be found in Algorithm 2.2.

2.3 Model Implementation

We utilized three models in total:

1. An Encoder-only Transformer
2. A Random Forest Classifier
3. An XGBoost Model

Our encoder-only transformer is based on a modified version of BERT, incorporating a positional encoding layer, an input layer, two attention blocks, a pooling layer, and a final linear layer with softmax.

Algorithm 2.1 Mapping Transaction IDs to Unique Identifiers

Input: Transaction trees in an array of DataFrames `dfs`
Output: Transaction trees in an array of DataFrames `dfs_with_mapped_txid` with mapped `txid`s

for *each DataFrame df in* `dfs` **do**
  /* Map and encode the 'txid' column */
  Initialize an empty list `txid_set`;
  for *each* `txid` *in df* **do**
    if `txid` *is not in* `txid_set` **then**
      Append `txid` to `txid_set`;
    end if
  end for
  Create `txid_dict`, mapping `txid` values to unique integers;
  Replace `txid` values in *df* with `txid_dict` values;
  if *the maximum value in the* `index` *column of df* > 8 **then**
    Skip to the next DataFrame;
  end if
  Append the processed *df* to `dfs_with_mapped_txid`;
end for

The key features of our model are as follows:

- **Input Layer:** The input layer processes 48 inputs/outputs with 48 features. If a set of transactions contains fewer inputs/outputs, they are automatically padded and passed to the positional encoding block.
- **Positional Encoding:** The positional encoding employs the algorithm proposed in Vaswani *et al.* (2017) with a minor modification. Instead of values ranging from [-1, 1], our model's positional encoding ranges from [-0.05, 0.05]. This adjustment was made because the feature values in our dataset are small, and applying a larger positional encoding relative to these values could significantly alter the feature's meaning. The positional encoding conveys the relative position of a transaction (e.g., which transaction is a child or parent) to the model.

Algorithm 2.2 Transforming Transaction Features

Input: Transaction trees in an array of DataFrames `dfs_with_mapped_txid`
Output: Transaction trees in an array of DataFrames `prepared_dfs` with mapped `txid`s

Concatenate all DataFrames in `dfs_with_mapped_txid` into `combined_df`;
`/* Log-transform and scale continuous columns */`
Log-transform the columns: `size`, `vsize`, `weight`, `locktime`, `value` in `combined_df`;
Fit MinMaxScalers on log-transformed columns to normalize values to [0, 1];
`/* Initialize and fit OneHotEncoders for categorical columns */`
Fit OneHotEncoders for the columns: `txid`, `version`, `addr_type`, `index`;
`/* Prepare encoded and transformed data */`
for *each DataFrame df in* `dfs_with_mapped_txid` **do**
    Log-transform the columns: `size`, `vsize`, `weight`, `locktime`, `value`;
    Apply OneHotEncoders to categorical columns, generating encoded features;
    Apply MinMaxScalers to continuous columns, generating scaled features;
    Concatenate all encoded and scaled features with `is_lightning` and `is_change` columns;
    Append the concatenated DataFrame to `prepared_dfs`;
end for

The necessary matrix is calculated using the following formulas:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \end{aligned} \tag{2.5}$$

We then apply the following transformation:

$$X_{\text{scaled}} = X_{\text{min}} + \left(\frac{X - X_{\text{orig\_min}}}{X_{\text{orig\_max}} - X_{\text{orig\_min}}}\right) \times (X_{\text{max}} - X_{\text{min}}) \tag{2.6}$$

where:

- X: The original data point.
- $X_{\text{orig\_min}}$: The minimum value of the original data range (e.g., $X_{\text{orig\_min}} = -1$).
- $X_{\text{orig\_max}}$: The maximum value of the original data range (e.g., $X_{\text{orig\_max}} = 1$).

- X_{min}: The minimum value of the desired scaled range (e.g., $X_{\text{min}} = -0.05$).
- X_{max}: The maximum value of the desired scaled range (e.g., $X_{\text{max}} = 0.05$).
- X_{scaled}: The transformed value within the new range $[X_{\text{min}}, X_{\text{max}}]$.

- **First Linear Layer:** This layer transforms the input from 2304 into a hidden representation of size 96. The transformation is performed by a linear layer, followed by dropout of 0.7 to prevent overfitting.
- **Transformer Block:** This block comprises a multi-head self-attention mechanism followed by a feed-forward neural network.
- **Pooling Layer:** After passing through the transformer blocks, the hidden representations are averaged across the sequence dimension using an adaptive average pooling layer. This reduces the sequence dimension to a fixed size of 1, summarizing the information from the entire sequence. The formula is as follows:

$$\text{Pooling}(x) = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.7}$$

- **Second Linear Layer:** This layer takes the pooled output and maps it to the final output size, which is reshaped to fit the desired output format (e.g., reshaped to `(-1, 12, 6)`). This transformation is achieved through a linear layer.
- **Softmax:** The softmax function is applied after the final linear layer, converting the output into probabilities, with the sum of probabilities equal to 1. The formula is given by:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.8}$$

Our model's architecture is illustrated in Figure 2.7. To assess the accuracy of our developed

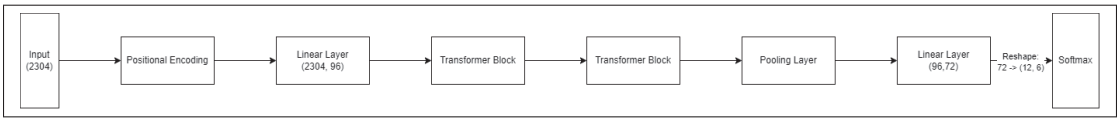

Figure 2.7 The internal architecture of the transformer, showcasing the positional encoding, transformer blocks, pooling layer, linear layer, and softmax function

model in comparison to a baseline, we used two of the best machine learning models from the literature. Ramos Tubino *et al.* (2022) demonstrated that the Random Forest Classifier(Breiman (2001)) and XGBoost(Chen & Guestrin (2016)) are among the best models for address clustering in Bitcoin. We employed these two models as baselines for comparison with our transformer architecture to evaluate its performance on the same data. To ensure optimal training of our baseline models, we used grid search. The grid search algorithm exhaustively explores all specified parameters to find the best result. Below are the parameters and selected values for each model:

- **Random Forest Classifier:** The Random Forest Classifier had the following parameters in the grid search:

  1. $n\_estimators$: Specifies the number of decision trees in the Random Forest.
  2. $max\_features$: Sets the maximum number of features considered when selecting the best split at each node.
  3. max_depth: Limits the maximum depth of each decision tree. Deeper trees can model more complex relationships but may lead to overfitting.
  4. $min\_samples\_split$: The number of samples required to split an internal node. Higher values prevent creating too many nodes with few samples, thereby reducing overfitting.
  5. $min\_samples\_leaf$: Defines the minimum number of samples required at a leaf node.

  Table 2.1 Random Forest Classifier Parameters: Potential and Selected Values

Parameter	Potential Values	Selected Value
n_estimators	[20, 50, 100]	50
max_features	['auto', 'sqrt', 'log2']	sqrt
max_depth	[None, 10, 20, 30]	10
min_samples_split	[2, 5, 10]	10
min_samples_leaf	[1, 2, 4]	2

  The results are summarized in Table 2.1.

- **XGBoost:** The XGBoost model had the following parameters in the grid search:

1. $n\_estimators$: Sets the number of gradient-boosted trees (i.e., the number of boosting rounds) to be created. Each tree corrects the errors of the previous trees. Increasing this value can improve performance but also increase computation time and risk overfitting. This is often one of the key parameters to tune.
2. max_depth: Determines the maximum depth of each tree. Deeper trees can capture more complex patterns but are more prone to overfitting. Shallow trees are generally preferred in XGBoost. Common values range from 3 to 10, but this can vary based on the specific problem.
3. $learning\_rate$: This parameter scales the contribution of each tree, helping prevent overfitting by making the boosting process more conservative. A smaller learning rate means the model learns more slowly but can achieve better performance. It is typically set between 0.01 and 0.3.
4. $colsample\_bytree$: Specifies the fraction of features (columns) to be randomly sampled for each tree. This feature subsampling introduces randomness, making the model more robust. A value of 1 means all features are considered for each tree, while lower values increase randomness. Typical values range from 0.5 to 1.

The results are summarized in Table 2.2.

Table 2.2 XGBoost Classifier Parameters:
Potential and Selected Values

Parameter	**Potential Values**	**Selected Value**
n_estimators	[10, 20, 50]	20
max_depth	[None, 10, 20, 30]	10
learning_rate	[0.01, 0.1, 0.2]	0.1
colsample_bytree	[0.5, 0.7, 1]	1

Each model was trained and tested twice: once with Lightning Network information included and once without, to evaluate the effectiveness of incorporating Lightning Network data for address clustering in Bitcoin.

CHAPTER 3

RESULTS

In this Chapter, we discuss the results that we got from each model.

3.1 Setup

The setup that we used was on a Windows 11 Operating System, Python 3.11, with an Nvidia RTX 2070 Super graphics card, 32 Gigabytes of DDR5 memory, and an Intel i5-13600K CPU. In total, there were 623,586 transactions outputs in 29,537 transactions. The training was performed on 20,676 (70%) and the test was performed on 8,861 (30%) from our dataset.

3.2 Evaluation Metrics

On each model, we performed the following evaluation metrics:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{3.1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

$$\text{F1 Score} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.4}$$

- Accuracy: Shows the overall correctness of the model. This value is calculated by dividing the number of correctly predicted instances to the total number of predictions.
- Precision: Show the true positive predictions over all positive predictions made by the model.
- Recall: Shows the model's ability to identify true positives out of the all positively labeled instances.
- F1 Score: Shows the harmonic mean of precision and recall.

3.3 Baseline - non transformers

The XGBoost and Random Forest Classifier almost performed similarly with or without the Lightning data. The result can be seen in table 3.1. Note that the numbers below are rounded and the values differ slightly.

Table 3.1 XGBoost and Random Forest classifier performance evaluation

Metric	RF	RF+LN data	XGBoost	XGBoost+LN data
Accuracy	0.87	0.87	0.87	0.87
Precision	0.99	0.99	0.99	0.99
Recall	0.25	0.25	0.25	0.25
F1 Score	0.40	0.40	0.40	0.40

3.4 Transformer

Similar to the XGBoost and Random Forest Classifier, we tested our modified BERT on with and without the Lightning data. The result can be seen in Table 3.2.

Table 3.2 BERT model performance evaluation

Metric	BERT	BERT+LN data
Accuracy	0.85	0.94
Precision	0.80	0.88
Recall	0.78	0.88
F1 Score	0.79	0.88

3.5 Comparison and Discussion

An interesting observation in our results was the significant impact of Lightning data on the performance of the BERT model, contrasted with the XGBoost and Random Forest Classifiers.

3.5.1 Baseline Models (XGBoost and Random Forest)

As shown in Table 3.1, both XGBoost and Random Forest classifiers performed almost identically across all metrics, regardless of the inclusion of Lightning data. The accuracy, precision, recall, and F1 scores remained constant at 0.87, 0.99, 0.25, and 0.40, respectively. This suggests that these models were not sensitive to the additional information provided by the Lightning data. The high precision but low recall across both models suggests that, while they were highly accurate in predicting positive instances, they failed to identify many true positives, resulting in a low F1 score. The extremely conservative labeling approach taken by these models in identifying change outputs may help prevent the formation of super clusters during address clustering; however, the very low recall means that many change outputs will not get flagged, ultimately reducing the reliability of the results. We were unable to identify a specific reason for this performance. Despite expanding the number of parameters in the grid search algorithm, we could not improve the models' high precision and low recall rates.

3.5.2 Transformer Model (BERT)

In contrast, the BERT model exhibited a substantial improvement in all evaluation metrics when the Lightning data was included, as depicted in Table 3.2. Without the Lightning data, BERT achieved an accuracy of 0.85, precision of 0.80, recall of 0.78, and F1 score of 0.79. However, with the inclusion of the Lightning data, all metrics saw a noticeable increase, with accuracy rising to 0.94, precision to 0.88, recall to 0.88, and F1 score to 0.88. This indicates that the BERT model was able to leverage the additional Lightning data effectively, resulting in a more balanced performance between precision and recall.

3.5.3 Overall Discussion

The results highlight the difference in how traditional ensemble models like Random Forest and XGBoost versus transformer-based models like BERT respond to additional data features. The BERT model's significant improvement with Lightning data suggests that it has a better capacity

to learn complex patterns from diverse data inputs compared to the other models. This could be due to the BERT model's ability to capture more intricate relationships within the data using its transformer architecture. On the other hand, the lack of performance gain in the XGBoost and Random Forest models could be attributed to their reliance on tree-based structures, which might not fully exploit the additional information provided by the Lightning data. Initially, we expected all models to improve by adding the lightning network data. Although all models showed improvement (though minimal in the case of the classical models), the gains for XGBoost and Random Forest were less than anticipated. This outcome might have differed with an alternative data preparation process or additional feature engineering techniques, which is interesting to explore in future research in our opinion. In addition, there are two reliable ways to undermine our model by generating erroneous results. First, if a user employs a coinjoin transaction, it can cause the multi-input heuristic to produce false positives. Second, if a user conducts Lightning Funding transactions with the other output as a payment, the Lightning Funding change output heuristic becomes unreliable, and the result cannot be used for deanonymization.

In conclusion, while XGBoost and Random Forest classifiers maintain strong precision, their low recall limits their overall effectiveness. The BERT model, however, demonstrates that integrating additional data features can lead to significant performance gains, making it a more suitable choice for tasks where capturing all relevant instances is crucial. In terms of our initial research problems, we believe that our results indicate that transformers are effective models for change output detection when provided with appropriate data. The high accuracy score, with maintained precision and recall, outperforming some of the best classical models, supports this claim. Furthermore, the addition of Lightning Network information can significantly enhance the accuracy of change output detection if the data is introduces to the proper model with the right embedding.

CONCLUSION AND RECOMMENDATIONS

In conclusion, we provide a summary of the thesis in Section 4.1 and discuss potential future work that could build on this research in Section 4.2.

4.1 Summary

Bitcoin privacy has been a subject of research since the cryptocurrency's inception, with efforts aimed at analyzing the network's privacy properties. Despite these efforts, we identified a gap in the literature, specifically the lack of research connecting the Lightning Network—a layer-2 scaling solution for Bitcoin—with its impact on the deanonymization of Bitcoin users. Additionally, recent advancements in transformer models motivated us to explore their potential effect on the deanonymization of the Bitcoin network. Consequently, this thesis aimed to develop a proof-of-concept to demonstrate the potential of integrating information from these two networks.

In Chapter 1, we provided the necessary background to understand the material presented in this thesis, followed by an overview of the current state of the literature on Bitcoin privacy, Lightning Network privacy, cross-layer analysis, and the application of transformers in blockchain deanonymization.

Next, in Chapter 2, we began by analyzing the features of the Bitcoin blockchain and Lightning Network Funding transactions. Specifically, we focused on the number of inputs and outputs, as well as user behavior patterns. We then used our findings to compile a dataset, applying the multi-input heuristic and identifying change addresses in Lightning Network Funding transactions to label our data. Subsequently, we discussed the data preparation techniques employed to ready the dataset for our machine learning models. Finally, we detailed the models we used and their specifications.

Lastly, in Chapter 3, we described our experimental setup, evaluation metrics, and compared the performance of the models. We observed that the BERT model effectively utilized the additional information provided by combining data from both the Lightning Network and Bitcoin.

4.2 Future Direction

There are several directions in which we believe our work can be improved. The first is to build a more robust labeled dataset. As explained, labeling our ground truth dataset was one of the most challenging aspects of our research, and with a more robust dataset, the model could better represent real-world scenarios. The second improvement is to reduce the constraints imposed on the dataset. We applied two constraints: one on the number of inputs and outputs in each transaction, and the other on the depth of the tree built for each transaction. By reducing these constraints, the model could learn more complex patterns. Another approach is to enhance the representation of data provided to the model, allowing it to better understand and process the information. Finally, we recommend applying transformers to other blockchains to evaluate whether these models can be effective for forensic purposes across different platforms. For UTXO-based blockchains, we hypothesize that similar results may be achieved. However, for account-based blockchains, we believe that a more intricate transaction gathering and embedding process will be necessary. Without further research, it is challenging to generalize our results to these blockchains as well.

APPENDIX I

SOURCE CODE IMPLEMENTATION

1. Wrapper for Python-BitcoinRPC:

```
from bitcoinrpc.authproxy import AuthServiceProxy
class BitcoinClient:
    def __init__(self, host, username, password):
        self.rpc = AuthServiceProxy(f"http://{username}:{
            password}@{host}:8332", timeout=120)
    def transaction_by_id(self, txid):
        return self.rpc.getrawtransaction(txid, True)
    def blockcount(self):
        return self.rpc.getblockcount()
    def getblock(self, blockhash):
        return self.rpc.getblock(blockhash, 2)
    def blockhash_by_height(self, height):
        return self.rpc.getblockhash(height)
    def blocks(self, rng):
        blockcount = self.blockcount()
        return [self.blockhash_by_height(blockcount - i)
            for i in range(rng)]
    def block_range(self, start, finish):
        return [self.blockhash_by_height(i) for i in range(
            start, finish + 1)]
```

2. Preparing Pandas DataFrames:

```
from typing import List
from bitcoin_client import BitcoinClient
import pandas as pd
def _address_type(address: str) -> int:
```

```
    if address.startswith("1"):
        return 1  # "P2PKH"
    if address.startswith("3"):
        return 2  # "P2SH"
    if address.startswith("bc1q"):
        return 3  # "Bech32"
    if address.startswith("bc1p"):
        return 4  # "P2TR"
    return 0  # "unkown"
def _is_lightning(txid: str, vout: int, lightning_txids:
   list) -> int:
    if f"{txid}:{vout}" in lightning_txids:
        return 1
    return 0
def _is_change(txid: str, vout: int, changes: list) -> int:
    if f"{txid}:{vout}" in changes:
        return 1
    return 0
def tx_to_df(
    client: BitcoinClient,
    txid: str,
    lightning_txids: List[str],
    changes: List[str],
    txo_limitation: int = 3,
) -> pd.DataFrame:
    tx = client.transaction_by_id(txid)
    if len(tx["vout"]) > txo_limitation or len(tx["vin"]) >
        txo_limitation:
```

```
        raise Exception(f"Transaction has more than {
            txo_limitation} inputs or outputs")
    df = pd.DataFrame(
        {
            "output": pd.Series(dtype="bool"),
            "txid": pd.Series(dtype="str"),
            "version": pd.Series(dtype="int"),
            "size": pd.Series(dtype="int"),
            "vsize": pd.Series(dtype="int"),
            "weight": pd.Series(dtype="int"),
            "locktime": pd.Series(dtype="int"),
            "index": pd.Series(dtype="int"),
            "value": pd.Series(dtype="float"),
            "addr_type": pd.Series(dtype="int"),
            "is_lightning": pd.Series(dtype="int"),
            "is_change": pd.Series(dtype="int"),
        }
    )
    for vout in tx["vout"]:
        txid = tx["txid"]
        version = tx["version"]
        size = tx["size"]
        vsize = tx["vsize"]
        weight = tx["weight"]
        locktime = tx["locktime"]
        n = vout["n"]
        value = vout["value"]
        addr_type = _address_type(vout["scriptPubKey"]["
            address"])
```

```
            lightning = _is_lightning(txid, n, lightning_txids)
            change = _is_change(txid, n, changes)
            df.loc[len(df.index)] = [
                True,
                txid,
                int(version),
                int(size),
                int(vsize),
                int(weight),
                int(locktime),
                int(n) + 1,
                float(value),
                addr_type,
                lightning,
                change,
            ]
        if len(tx["vout"]) < 3:
            for _ in range(3 - len(tx["vout"])):
                df.loc[len(df.index)] = [True, "", 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0]
        for vin in tx["vin"]:
            txid = vin["txid"]
            n = vin["vout"]
            vin_tx = client.transaction_by_id(txid)
            version = vin_tx["version"]
            size = vin_tx["size"]
            vsize = vin_tx["vsize"]
            weight = vin_tx["weight"]
            locktime = vin_tx["locktime"]
```

```
            value = vin_tx["vout"][n]["value"]
            addr_type = _address_type(vin_tx["vout"][n]["
                scriptPubKey"]["address"])
            df.loc[len(df.index)] = [
                False,
                txid,
                int(version),
                int(size),
                int(vsize),
                int(weight),
                int(locktime),
                int(n) + 1,
                float(value),
                addr_type,
                0,
                0,
            ]
        if len(tx["vin"]) < 3:
            for _ in range(3 - len(tx["vin"])):
                df.loc[len(df.index)] = [False, "", 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0]
        return df
```

3. Data Prepration for Training:

```
import pickle
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder,
    MinMaxScaler
```

```
with open('./dataframes.pkl', 'rb') as f:
    dfs = pickle.load(f)
def map_txid(df: pd.DataFrame) -> pd.DataFrame:
    txid_set = []
    for txid in df['txid']:
        if txid in txid_set:
            continue
        txid_set.append(txid)
    txid_dict = dict()
    txid_dict[""] = 0
    i = 1
    for j in range(len(txid_set)):
        if txid_set[j] == "":
            continue
        txid_dict[txid_set[j]] = i
        i+=1
    df['txid'] = df['txid'].replace(txid_dict)
    return df
dfs_with_mapped_txid = []
for txid, df in dfs.items():
    data = map_txid(df)
    if max(data['index']) > 8:
        continue
    dfs_with_mapped_txid.append(data)
dfs_with_mapped_txid[0]
len(dfs_with_mapped_txid)
combined_df = pd.concat(dfs_with_mapped_txid, axis=0,
   ignore_index=True)
combined_df.describe()
```

```
txid_encoder = OneHotEncoder()
txid_encoder.fit(np.array(combined_df['txid']).reshape
    (-1,1))
version_encoder = OneHotEncoder()
version_encoder.fit(np.array(combined_df['version']).
    reshape(-1,1))
address_type_encoder = OneHotEncoder()
address_type_encoder.fit(np.array(combined_df['addr_type'])
    .reshape(-1,1))
index_encoder = OneHotEncoder()
index_encoder.fit(np.array(combined_df['index']).reshape
    (-1,1))
columns_to_log_transform = ['size', 'vsize', 'weight', '
    locktime', 'value']
combined_df[columns_to_log_transform] = np.log1p(
    combined_df[columns_to_log_transform])
size_encoder = MinMaxScaler(feature_range=(0, 1))
size_encoder.fit(np.array(combined_df['size']).reshape
    (-1,1))
vsize_encoder = MinMaxScaler(feature_range=(0, 1))
vsize_encoder.fit(np.array(combined_df['vsize']).reshape
    (-1,1))
weight_encoder = MinMaxScaler(feature_range=(0, 1))
weight_encoder.fit(np.array(combined_df['weight']).reshape
    (-1,1))
locktime_encoder = MinMaxScaler(feature_range=(0, 1))
locktime_encoder.fit(np.array(combined_df['locktime']).
    reshape(-1,1))
value_encoder = MinMaxScaler(feature_range=(0, 1))
```

```
value_encoder.fit(np.array(combined_df['value']).reshape
    (-1,1))
txid_columns = []
for i in range (len(txid_encoder.transform(np.array(
    combined_df['txid']).reshape(-1,1)).toarray()[0])):
      txid_columns.append(f'txid_{i}')
version_columns = []
for i in range (len(version_encoder.transform(np.array(
    combined_df['version']).reshape(-1,1)).toarray()[0])):
      version_columns.append(f'version_{i}')
addr_type_columns = []
for i in range (len(address_type_encoder.transform(np.array
    (combined_df['addr_type']).reshape(-1,1)).toarray()[0])):
      addr_type_columns.append(f'addr_type_{i}')
index_columns = []
for i in range (len(index_encoder.transform(np.array(
    combined_df['index']).reshape(-1,1)).toarray()[0])):
      index_columns.append(f'index_{i}')
prepared_dfs = []
for df in dfs_with_mapped_txid:
      df[columns_to_log_transform] = np.log1p(df[
          columns_to_log_transform])
      txid_encoded = pd.DataFrame((txid_encoder.transform(np.
          array(df['txid']).reshape(-1,1)).toarray()), columns=
          txid_columns)
      version_encoded = pd.DataFrame((version_encoder.
          transform(np.array(df['version']).reshape(-1, 1)).
          toarray()), columns=version_columns)
```

```
addr_type_encoded = pd.DataFrame((address_type_encoder.
    transform(np.array(df['addr_type']).reshape(-1, 1)).
    toarray()), columns=addr_type_columns)
index_encoded = pd.DataFrame((index_encoder.transform(
    np.array(df['index']).reshape(-1,1))).toarray(),
    columns=index_columns)
size_encoded = pd.DataFrame((size_encoder.transform(np.
    array(df['size']).reshape(-1,1))), columns=['size'])
vsize_encoded = pd.DataFrame((vsize_encoder.transform(
    np.array(df['vsize']).reshape(-1,1))), columns=['
    vsize'])
weight_encoded = pd.DataFrame((weight_encoder.transform
    (np.array(df['weight']).reshape(-1,1))), columns=['
    weight'])
locktime_encoded = pd.DataFrame((locktime_encoder.
    transform(np.array(df['locktime']).reshape(-1,1))),
    columns=['locktime'])
value_encoded = pd.DataFrame((value_encoder.transform(
    np.array(df['value']).reshape(-1,1))), columns=['
    value'])
lightning = pd.DataFrame(df['is_lightning']).
    reset_index(drop=True)
change = pd.DataFrame(df['is_change']).reset_index(drop
    =True)
concated_df = pd.concat([txid_encoded, version_encoded,
     addr_type_encoded, index_encoded, size_encoded,
    vsize_encoded, weight_encoded, locktime_encoded,
    value_encoded, lightning, change], axis=1)
prepared_dfs.append(concated_df)
```

```
with open('./prepared_dataframes.pkl', 'wb') as f:
    dfs = pickle.dump(prepared_dfs, f, pickle.
        HIGHEST_PROTOCOL)
```

4. Querying the Parent of Transactions:

```
from bitcoin_client import BitcoinClient
def parent_finder(
    client: BitcoinClient, txid: str, depth: int = 3
) -> tuple[list[str], list[str]]:
    children = [txid]
    parent = []
    changes = []
    counter = 0
    while True:
        if counter == depth:
            break
        next_gen = []
        for child in children:
            child_json = client.transaction_by_id(child)
            for vin in child_json["vin"]:
                if "coinbase" in vin:
                    continue
                next_gen.append(vin["txid"])
                changes.append(f"{vin['txid']}:{vin['vout
                    ']}")
        parent.append(children.copy())
        children = next_gen.copy()
        counter += 1
    return (parent, changes)
```

5. Training BERT model and calculating its accuracy:

```
from sklearn.metrics import precision_score, recall_score,
    f1_score
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import math
from .bert import BERT
with open('./prepared_dataframes.pkl', 'rb') as f:
    dfs = pickle.load(f)
data = []
labels = []
for df in dfs:
    change = df[['is_change']]
    df['is_lightning'] = 0
    label = np.array(
        change.index[change['is_change'] == 1].tolist(),
            dtype=np.float64)
    labels.append(label)
    data.append(df.drop('is_change', axis=1, inplace=False)
        )
label_size_count = {}
for label in labels:
```

```
    if (len(label) in label_size_count):
        label_size_count[len(label)] += 1
    else:
        label_size_count[len(label)] = 1
for i in label_size_count:
    print(f'number of labels with {i} change label is {
      label_size_count[i]}')
max_label = 12
for i in range(len(labels)):
    diff = max_label - len(labels[i])
    labels[i] = labels[i] % 6
    labels[i] = np.concatenate([labels[i], ([-1]*diff)])
label_size_count = {}
for label in labels:
    if (len(label) in label_size_count):
        label_size_count[len(label)] += 1
    else:
        label_size_count[len(label)] = 1
for i in label_size_count:
    print(f'number of labels with {i} change label is {
      label_size_count[i]}')
data_size_count = {}
for df in data:
    if (df.shape[0] in data_size_count):
        data_size_count[df.shape[0]] += 1
    else:
        data_size_count[df.shape[0]] = 1
for i in data_size_count:
```

```
    print(f'number of data with {i} rows label is {
        data_size_count[i]}')
max_df_row = 48
lightning_positive = 0
for i in range(len(data)):
    lightning_positive += sum(data[i]['is_lightning'])
    padding = max_df_row - data[i].shape[0]
    new_rows = pd.DataFrame(
        np.zeros((padding, data[i].shape[1])), columns=data
            [i].columns)
    data[i] = pd.concat([data[i], new_rows], ignore_index=
        True)
data_size_count = {}
for df in data:
    if (df.shape[0] in data_size_count):
        data_size_count[df.shape[0]] += 1
    else:
        data_size_count[df.shape[0]] = 1
device = 'cuda' if torch.cuda.is_available() else 'cpu'
train_data, test_data, train_labels, test_labels =
    train_test_split(
    data, labels, test_size=0.3)
class TransactionDataset(Dataset):
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels
    def __len__(self):
        return len(self.data)
    def __getitem__(self, index):
```

```
        flattened = np.array(self.data[index]).flatten().
            astype(np.float64)
        x = torch.tensor(flattened, device=device, dtype=
            torch.float64)
        y = torch.tensor(self.labels[index],
                         device=device, dtype=torch.float64
                             )
        return x, y
train_dataset = TransactionDataset(train_data, train_labels
    )
test_dataset = TransactionDataset(test_data, test_labels)
train_dataloader = DataLoader(train_dataset, batch_size=16,
    shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=16,
    shuffle=False)
class GroupwiseCrossEntropyLoss(nn.Module):
    def __init__(self):
        super(GroupwiseCrossEntropyLoss, self).__init__()
    def forward(self, input, target):
        input = input.double()
        target = target.long()
        loss = 0
        batch_size, num_groups, _ = input.shape
        for i in range(num_groups):  # Iterate over the
            groups
            # Select the logits for the i-th group
            group_input = input[:, i, :]
            # Select the targets for the i-th group
            group_target = target[:, i]
```

```
                # Create a mask to ignore the padded values
                    (-1) in the targets
                valid_mask = group_target != -1
                # Apply the mask to select only the valid (non-
                    padded) elements
                valid_group_input = group_input[valid_mask]
                valid_group_target = group_target[valid_mask]
                # Calculate loss for the i-th group only if
                    there are valid elements
                if len(valid_group_target) > 0:
                    group_loss = F.cross_entropy(
                        valid_group_input, valid_group_target)
                    loss += group_loss
            # Return the average loss across the groups
            return loss / num_groups
def calculate_accuracy(labels, outputs):
    probabilities = nn.functional.softmax(outputs, dim=2)
    _, predicted_classes = torch.max(probabilities, dim=2)
    mask = labels != -1
    correct_predictions = (predicted_classes == labels) *
        mask
    total_valid_labels = mask.sum()
    accuracy = correct_predictions.sum() /
        total_valid_labels.float()
    return accuracy.item()
model = BERT(2304, 2, 2, 96, 12, 0.7).to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss = GroupwiseCrossEntropyLoss()
epochs = 30
```

```
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_dataloader:
        inputs, labels = inputs.to(device), labels.to(
            device)
        optimizer.zero_grad()
        output = model(inputs)
        l = loss(output, labels)
        l.backward()
        optimizer.step()
        running_loss += l.item()
    print(f"Epoch {epoch + 1}/{epochs}, Loss: {running_loss
        /
          len(train_dataloader)}")
    if (epoch + 1) % 2 == 0:
        model.eval()
        with torch.no_grad():
            eval_loss = 0.0
            total_accuracy = 0.0
            num_batches = 0
            for inputs, labels in test_dataloader:
                inputs, labels = inputs.to(device), labels.
                    to(device)
                output = model(inputs)
                l = loss(output, labels)
                eval_loss += l.item()
                total_accuracy += calculate_accuracy(labels
                    , output)
```

```
                num_batches += 1
            print(f'accuracy in epoch{
                  epoch+1} is {total_accuracy/num_batches}
                     percent.')
def calculate_metrics(labels, outputs):
    probabilities = nn.functional.softmax(outputs, dim=2)
    _, predicted_classes = torch.max(probabilities, dim=2)
    mask = labels != -1
    labels_flat = labels[mask].cpu().numpy()
    predicted_flat = predicted_classes[mask].cpu().numpy()
    accuracy = (predicted_flat == labels_flat).sum() / len(
       labels_flat)
    precision = precision_score(
        labels_flat, predicted_flat, average='weighted',
           zero_division=0)
    recall = recall_score(labels_flat, predicted_flat,
                          average='weighted', zero_division
                             =0)
    f1 = f1_score(labels_flat, predicted_flat,
                  average='weighted', zero_division=0)

    return accuracy, precision, recall, f1
model.eval()
with torch.no_grad():
    eval_loss = 0.0
    total_accuracy = 0.0
    total_precision = 0.0
    total_recall = 0.0
    total_f1 = 0.0
```

```
num_batches = 0
for inputs, labels in test_dataloader:
    inputs, labels = inputs.to(device), labels.to(
        device)
    output = model(inputs)
    l = loss(output, labels)
    eval_loss += l.item()
    accuracy, precision, recall, f1 = calculate_metrics
        (labels, output)
    total_accuracy += accuracy
    total_precision += precision
    total_recall += recall
    total_f1 += f1
    num_batches += 1
print(f'accuracy is {total_accuracy/num_batches}
    percent.')
print(f'precision is {total_precision/num_batches}
    percent.')
print(f'f1 is {total_f1/num_batches} percent.')
print(f'recall is {total_recall/num_batches} percent.')
```

6. Training the XGBoost and Random Forest Classifier:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split,
    GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
from sklearn.metrics import accuracy_score, precision_score
    , recall_score, f1_score
def calculate_metrics(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='
        weighted')
    recall = recall_score(y_true, y_pred, average='weighted
        ')
    f1 = f1_score(y_true, y_pred, average='weighted')
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    return accuracy, precision, recall, f1
dataset, label = load_prepared_dataset()
X_train, X_test, y_train, y_test = train_test_split(dataset
    , label, test_size=0.3)
rf_param_grid = {
    'n_estimators': [20, 50, 100],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
xgb_param_grid = {
    'n_estimators': [10, 20, 50],
    'max_depth': [10, 20, 30],
    'learning_rate': [0.01, 0.1, 0.2],
    'colsample_bytree': [0.5, 0.7, 1]
```

```
}
rf = RandomForestClassifier()
rf_grid_search = GridSearchCV(estimator=rf, param_grid=
    rf_param_grid, cv=5, scoring='accuracy')
rf_grid_search.fit(X_train, y_train)
best_rf = rf_grid_search.best_estimator_
xgb = XGBClassifier(use_label_encoder=False, eval_metric='
    mlogloss')
xgb_grid_search = GridSearchCV(estimator=xgb, param_grid=
    xgb_param_grid, cv=5, scoring='accuracy')
xgb_grid_search.fit(X_train, y_train)
best_xgb = xgb_grid_search.best_estimator_
print("Random Forest Classifier Metrics:")
rf_y_pred = best_rf.predict(X_test)
rf_metrics = calculate_metrics(y_test, rf_y_pred)
print("\nXGBoost Classifier Metrics:")
xgb_y_pred = best_xgb.predict(X_test)
xgb_metrics = calculate_metrics(y_test, xgb_y_pred)
```

APPENDIX II

ATTENTION TO LIGHTNING CHANGE: UTILIZING BERTS TO DETECT BITCOIN CHANGE ADDRESS WITH LIGHTNING NETWORK INFORMATION

Mikaeil Mayeli Feridani[1] , Rodrigue Tonga Naha[1] , Oumayma Dekhil[1] , Fatma Najjar[1] , Kaiwen Zhang[1]

[1] Department of Software and IT Engineering, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Paper submitted to the conference «BRAINS 2024» on June 2024.

1. Introduction

Bitcoin has encountered significant scalability challenges as its user base has expanded. Notably, transaction fees reached unprecedented levels during periods of intense activity, such as late 2017 and early 2018, and more recently in April 2024, reaching up to 120 USD during the halving of the Bitcoin mining rewardBlockchain.com (2024). To mitigate these scalability issues, Poon and Dryja introduced the Lightning Network in 2016Poon & Dryja (2015). Lightning Network is a Layer-2 state channel scalability solution that enables nearly instantaneous transactions at minimal costs. Given these benefits, Bitcoin users and major actors, including exchanges like BinanceBinance (2023), KrakenKraken (2022), and BitfinexBitfinex (2020), or mining pools such as BraiinsBraiins (2024) have increasingly adopted the LN in recent years, due to its low fees and reliability. As of the time of this writing, over $336 million worth of Bitcoin is locked in LN channels according to the Mempool.space website (https://www.mempool.space), a figure that has risen since the advent of this technology.

The inherent transparency of Bitcoin poses a risk of deanonymizing users behind pseudonymous addresses, compromising their privacy. Despite its increasing adoption and notable benefits, the impact of the Lightning Network on the anonymity and privacy of Bitcoin transactions remains insufficiently studied. Current research primarily focuses on the privacy features within the Lightning Network channels Kappos *et al.* (2021); Herrera-Joancomartí *et al.* (2019); Von Arx

et al. (2023) or the Bitcoin network itself Möser & Narayanan (2022); Harrigan & Fretter (2016); Zhang, Wang & Luo (2020); Meiklejohn *et al.* (2013); Androulaki *et al.* (2013). Research into how the Lightning Network influences the anonymity of entities on the Bitcoin network is underexplored. Although efforts such asRomiti *et al.* (2021) exists, there is still a significant gap in the literature given the potential of Lightning network and its increasing adaptation.

This study aims to explore how public Lightning Network activity impacts the detection of change outputs in the Bitcoin network. We employ a novel approach by integrating a modified Bidirectional Encoder Representations from Transformers (BERT)Devlin *et al.* (2019) model, enhancing our ability to handle sequential data and contextual relationships within the Bitcoin transaction network. We extract features from multiple transaction levels, alongside data from both the LN and the Bitcoin mainnet. This integrated approach facilitates the identification of relevant patterns and the extraction of features that significantly improve prediction accuracy, achieving a 93.9% success rate in identifying change transaction outputs.

In this paper, we provide the following contributions:

1. Analyzing the statistics of LN channel funding transactions on Bitcoin, and compare them to regular Bitcoin transactions.
2. Evaluating the effectiveness of encoder-only transformers in detecting change outputs on UTXO-based blockchains by providing a Proof-of-Concept.
3. Assessing the effects of combining the information from LN nodes with Bitcoin transactions to enhance change output detection.

The rest of the paper is structured as follows. In section 2, we discuss background information necessary for the work. In section 3 we look at the current state of literature for Bitcoin deanonymization and cross-layer privacy analysis. Then in section 4 we present our model, data-gathering techniques, followed by evaluation in section 5. Finally, we provide a conclusion in section 6 along with potential future works.

2. Background

In this section, we present the background information necessary to understand the paper. First, we provide a comprehensive overview of Bitcoin, information about its transactions and change outputs. Then we discuss multi-input heuristic, and its important role in deanonymizing addresses. Furthermore, we discuss the Lightning Network, a layer-2 solution designed to help with scaling Bitcoin network. Lastly, we explore the role of the Transformers architecture and its relevance to our research. Originally developed for natural language processing, Transformers use a self-attention mechanism that we adapt to analyze transaction patterns within the Bitcoin network.

2.1 Bitcoin

Bitcoin operates as a peer-to-peer (P2P) system consisting of nodes that relay transactions announced by users and incorporate them into "blocks" to maintain a unified history of all validated transactions. Each connected and updated full node has access to the entire transaction history recorded on the blockchain. Transactions typically involve a series of inputs—except for the Coinbase transaction, which is a reward granted to a miner who successfully mines a block—and generate new outputs known as Unspent Transaction Output (UTXO). In the process of creating a new transaction, a previously generated transaction output is utilized as an input, and a new output is created. Consequently, each new UTXO can be traced back to a previous transaction. There is significant research focused on clustering transactions related to the same entity, which can potentially lead to the deanonymization of Bitcoin users.

2.1.1 Change Output

In a typical Bitcoin transaction, if the total value of the inputs exceeds the amount a user intends to send to another entity, a separate output known as the change output is created. This change output returns the excess amount to the user (e.g., Fig. A II-1).

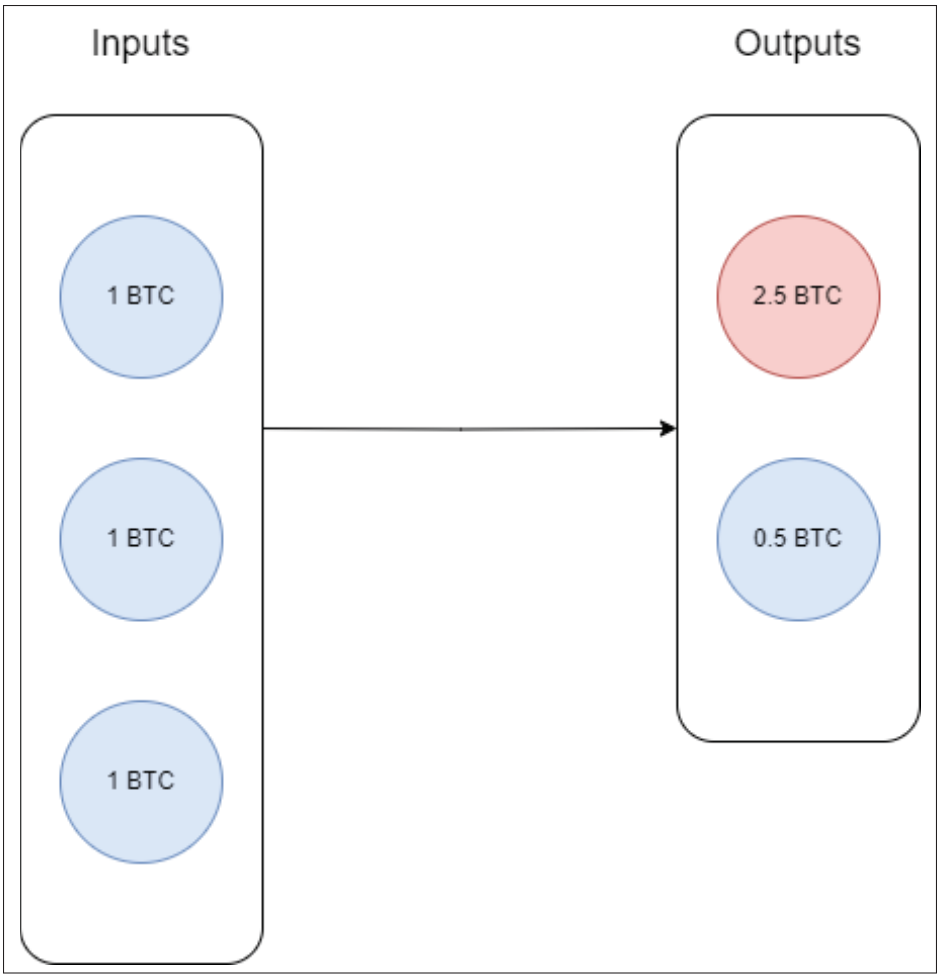

Figure A II-1 In this example, the transaction contains three inputs with a value of 1 BTC, a payment output with a value of 2.5 BTC, and a change output with a value of 0.5 BTC. The inputs and change output are controlled by the same entity

Change output transactions are generally controlled by the same entity that initiates the transaction. Detecting these change outputs facilitate the clustering of Unspent Transaction Outputs (UTXOs) and addresses, which directly impacts the anonymity of Bitcoin users.

2.1.2 Multi-Input Heuristic

The multi-input heuristicAndroulaki *et al.* (2013) assumes that if multiple inputs are spent in the same transaction, they are most likely controlled by the same entity. Therefore, by employing

the multi-input heuristic along with the detection of change output transactions, we can trace the chain of transactions to deanonymize public addresses. The multi-input heuristic has been thoroughly studied over the years and has been shown to be one of the most effective methods for clustering Bitcoin addressesMeiklejohn *et al.* (2013); Androulaki *et al.* (2013); Ron & Shamir (2013); Harrigan & Fretter (2016). As an example, in Fig A II-1, using this heuristic we can deduce that all of the three inputs are controlled by the same entity and the related addresses can get clustered together. We later on use this heuristic as one of our assumptions to label our dataset (discussed in Section 4).

2.2 Lightning Network

The Lightning NetworkPoon & Dryja (2015) is a network of channels that facilitates payments between two parties without the need to announce these transactions on the Bitcoin blockchain. The initial step in utilizing this network involves creating a payment channel through a 2-2 multisignature transaction on the Bitcoin blockchain, known as the "Funding" transaction. A 2-2 multisignature transaction is a special type of transaction that requires two signatures for the output to be spent. Once this funding transaction is recorded in a block, the channel becomes operational, and each party can send payments by creating a commitment transaction based on the funding transaction. When both participants agree to close the channel, they announce the final balance as the output to the multisignature transaction which is recorded on the blockchain. In the Lightning Network, node software (separate from a Bitcoin full node) is responsible for managing and handling these channels. These nodes can have either public or private channels; public channels disclose their IP address or Tor address, along with an alias, to enable other nodes to route transactions through them. We utilize the information available to the public nodes in the Lightning Network in our dataset for detecting change transaction outputs (discussed in section 4).

2.3 Transformers and Attention Mechanism

The Transformers architecture introduced by Vaswani et al.Vaswani *et al.* (2017) was initially developed for language translation tasks. This architecture incorporates a self-attention mechanism combined with positional encoding, presenting a novel approach to processing data that are sequential in nature without relying on a recurrent network. This increases efficiency by allowing the input to be processed in parallel rather than sequentially. The self-attention mechanism enables the model to assess the importance of different parts of the input sequence. While the original Transformer model included components for encoding the input and decoding the output (based on the input), BERT, developed by Devlin et al.Devlin *et al.* (2019), utilizes an encoder-only architecture to pre-train deep bidirectional representations from unlabeled text. In this paper, we utilize an encoder-only Transformer to analyze relationships between different transaction outputs and their properties, in order to detect the change outputs.

3. Related Works

In this section, we will review the works related to our paper. First, we explore various machine learning techniques employed in the field of Bitcoin transaction analysis, particularly focusing on deanonymizing users and clustering transaction activities. We examine a range of studies that have leveraged machine learning models to infer patterns and identities from transaction data on the blockchain, highlighting their methodologies, achievements, and limitations. We also examine cross-layer analyses, particularly those involving the Lightning Network and the Bitcoin blockchain to provide deeper insights into user behavior and cross-layer network dynamics.

3.1 Deanonymization using Machine Learning Models

Since the inception of the Bitcoin blockchain, its promise of anonymity has been tested. Various research efforts aimed at deanonymizing addresses and clustering identities behind pseudonymous public addresses using publicly available information. Machine Learning approaches have

shown promising results through multiple studies. Despite the extensive research, none of these approaches considered data from Lightning Network as a feature.

In Ramos Tubino *et al.* (2022), various supervised machine learning approaches were utilized to detect change transactions. Their research provided evidence that supervised learning is effective for change output detection. Wei et. al.Shao *et al.* (2018) suggested a novel approach for Bitcoin user identification leveraging deep learning techniques. They extracted comprehensive address features, including statistical metrics like average transaction intervals and frequent transaction types, along with behavioral patterns processed through a Gated Recurrent Unit (GRU) RNN. Inspired by natural language processing, the transactions were embedded using the Word2Vec algorithm to convert sparse data into dense vectors which achieved 91% accuracy. Despite its effectiveness, the method has limitations, such as reduced performance with insufficient transaction history and the need for comprehensive feature extraction, which can be resource-intensive. Nonetheless, this approach outperformed traditional heuristics by requiring fewer but more informative data points, offering a scalable and accurate solution for cryptocurrency transaction analysis.

In Nerurkar *et al.* (2020) authors suggested using a tree-based classifier to identify illicit entities within the Bitcoin network. Their classifier was trained on a dataset comprising 2,059 real-world Bitcoin entities, categorized into 28 licit and illicit user groups. Utilizing 9 features for training, the model achieved a 66% correct identification rate. Although they succeeded in detecting categories of entities, they need additional feature engineering to improve accuracy. Additionally, the features used were too general to capture malicious transaction patterns accurately, and the dataset size was relatively small.

In Lee, Maharjan, Ko & Hong (2020a), the authors proposed a machine-learning approach to detect illegal transactions on the Bitcoin blockchain. Their approach entails the utilization of machine learning techniques, specifically employing Random Forest (RF) and Artificial Neural Network (ANN) algorithms to analyze Bitcoin transaction data and classify transactions as illegal or legal. The performance of the models was evaluated using the F1 score, resulting

in ANN and RF achieving scores of 0.89 and 0.98, respectively. However, the test set may have been overfitted, and the number of features used to determine the positive samples were likely too small. Additionally, the suggested neural network structure lacks normalization and regularization methods, resulting in decreased reliability of detection outcomes.

A machine learning classifier was proposed by the authors in "Machine Learning Based Bitcoin Address Classification"Lee, Maharjan, Ko, Woo & Hong (2020b) based on Random Forest and ANN algorithms as classification models. They used 80 transaction-level features to train the models and classify Bitcoin addresses. They achieved the accuracy of 84% for Random Forest, which was relatively higher than that of ANN.

In Najjar *et al.* (2024), the authors used a transaction-level Hierarchical Clustering to detect the change output. Despite achieving 75% accuracy, their model was limited to only one transaction at a time, losing valuable information that could be extracted by considering the history of each transaction.

A recent study that used transformers for deanonymization is Hu *et al.* (2023). This research focused on detecting various fraudulent behaviors on Ethereum and demonstrated superior performance over existing graph-based models for detecting phishing accounts. In our approach, we also utilize an encoder-only transformer for detecting change outputs in UTXO-based blockchains, a unique challenge distinct from those in account-based systems. To the best of our knowledge, our study is the first to apply transformers to change output detection in UTXO-based blockchains.

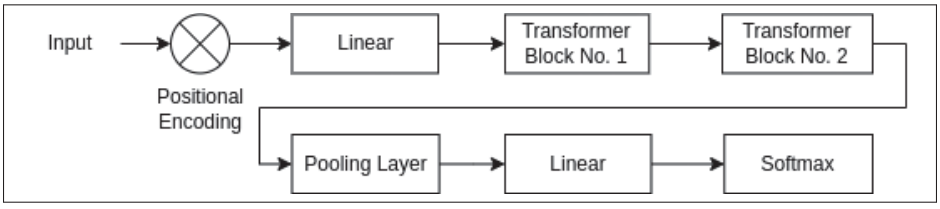

Figure A II-2 Model Architecture

3.2 Cross-Layer Analysis

In Romiti *et al.* (2021), Romiti et al. conducted a cross-layer analysis between the Lightning Network and Bitcoin for the first time. They introduced heuristics for clustering Bitcoin addresses and Lightning nodes, then analyzed the links between these clusters to connect the entities within each. In our research, we leverage the most promising heuristic they identified—the snake pattern. This pattern suggests that entities involved in sequential fund forwarding within a component are likely controlled by the same user. The authors were able to cluster 31% of funding components using this heuristic alone. We adopt this heuristic as a foundational assumption for labeling our dataset.

4. Proposed Solution

As mentioned before, our primary model is based on a modified BERTDevlin *et al.* (2019). The input size for this model is $48 * 48$. This dimension is derived from the fact that each transaction output has 48 encoded features, and there is a maximum of 48 inputs or outputs related to a single transaction in our dataset. For transactions with fewer inputs or outputs, padding with zero values is applied to achieve the required size. Each data point is derived from a tree of Bitcoin transactions. To help the model identify which transaction is the parent and which is the child, we employ positional encoding that assigns different values to different transactions but the same value to all the inputs and outputs related to the same transaction. The positional encoding formula used follows the one presented in Vaswani *et al.* (2017), but instead of values in range of [1, -1], our values are within the range of [-0.05, 0.05]. This adjustment is necessary because most of the features are relatively small in Bitcoin transactions, and a positional encoder that applies large values could drastically alter the meaning of the variables.

After applying the positional encodings, we use a fully connected layer to prepare our input for the attention layer. This layer is a matrix with the shape of (2304, 96). The processed results then pass through the attention blocks, and subsequently, adaptive average pooling is applied, followed by a fully connected layer with the shape of (96, 72), which is then transformed into a

(12,6) matrix. Finally, a Softmax function is applied to predict the transaction output for each entry, which is compared against the labels.

Each attention block in our model is composed of layers for the query, key, and value, followed by an output layer. The output layer is a linear layer with dimensions d_{model} to d_{model}, and the attention mechanism is governed by the following formula:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{A II-1}$$

Taken from (Vaswani et al., 2017, page 4). Where Q is Query matrix, K is Key matrix, V is Value matrix, and d_k is dimension of the query and key vectors. This attention mechanism is identical to that defined in Vaswani *et al.* (2017).

For loss calculation, we utilized a modified version of CrossEntropyLoss, where no loss is calculated for the padded transaction outputs. The total loss is then the average of the sum of losses across all real transaction outputs.

Additionally, we trained two other models for baseline comparison. The first model employed was a random forest (Breiman (2001)), and the second was XGBoost (Chen & Guestrin (2016)). We selected these models because Möser & Narayanan (2022) and Ramos Tubino *et al.* (2022) demonstrated promising results with random forest and XGBoost, respectively. For each model, we conducted training and testing using data combined from the Lightning Network and the Bitcoin network, as well as using only data from the Bitcoin network. We will present these results in Section 5.

4.1 Data Gathering

Our initial step in developing our solution was to conduct a comprehensive analysis of transactions on the Bitcoin blockchain, including those associated with funding public channels on the Lightning Network. For the first stage, we queried all transactions within blocks 828,577 to 829,577. This was achieved using a fully synced Bitcoin node (via bitcoin-core), accessing it over

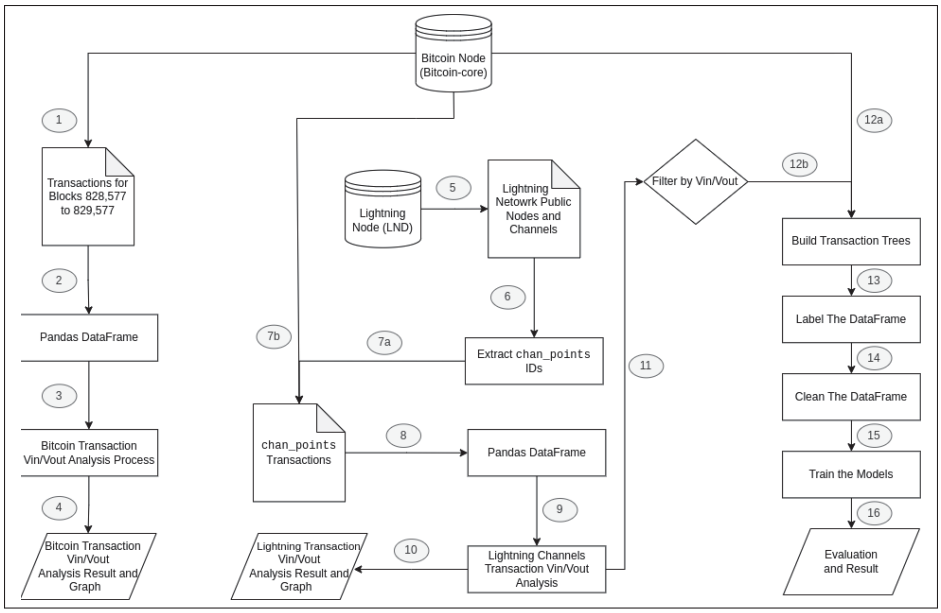

Figure A II-3 Data Gathering Procedure

RPC, and retrieving all block hashes using `blockhash_by_height`, followed by `getblock` with `verbosity` set to level 3. We then collected all transactions (2,257,416) within these blocks, examining the number of inputs used and outputs generated in each transaction (Fig. A II-4).

Our findings revealed that 90% of transactions generated 1 to 3 UTXOs. Similarly, over 90% of transactions consumed 1 to 3 inputs. We further investigated how many transactions had both 1 to 3 inputs and 1 to 3 outputs at the same time. The results, displayed in Table A II-1, indicate that these sets of transactions account for over 85% of all transactions within the examined block range. So, we determined it was appropriate to set our model's input and output limit size to 3.

Table A II-1 Percentage of Transactions by Inputs and Outputs

Inputs Vs Outputs	**1 Output**	**2 Output**	**3 Output**
1 Input	0.42%	0.30%	0.03%
2 Inputs	0.02%	0.06%	0.01%
3 Inputs	0.01%	0.01%	0.001%

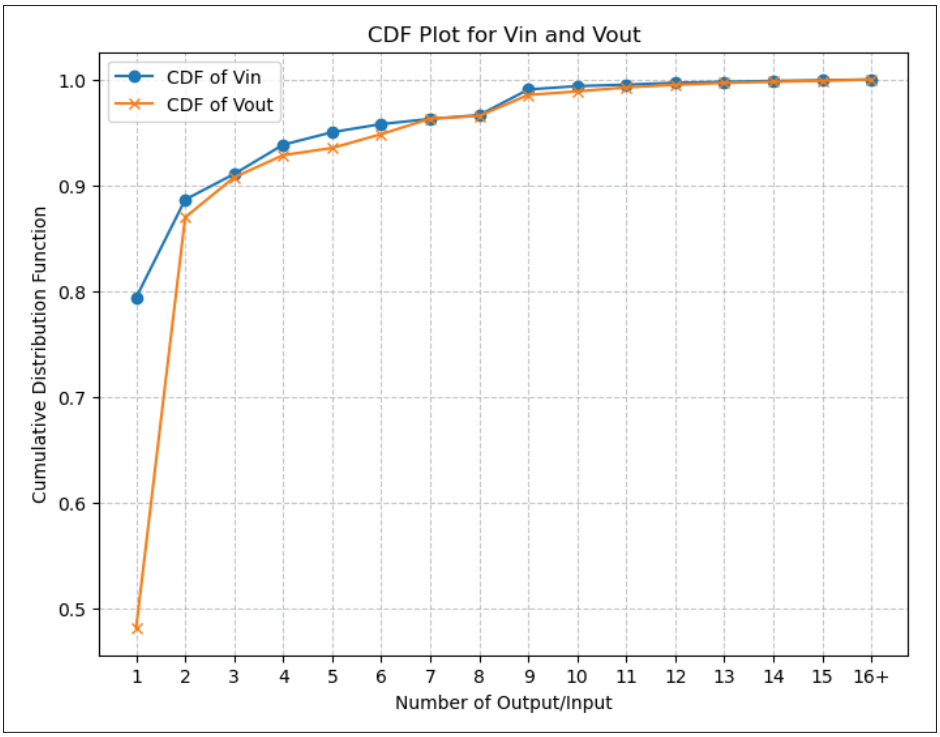

Figure A II-4 Bitcoin Transactions Input/Output CDF

In the next step, we analyzed publicly announced Lightning Network funding transactions. First, we captured a snapshot using a fully-synced Lightning node (employing LND software) during the first week of February 2024. We then queried the funding transaction IDs using our Bitcoin node and analyzed the results, gathering a total of 66,810 `chan_points`. Similar to regular Bitcoin transactions, near 90% of transactions had 1 to 3 inputs or 1 to 3 outputs (Fig. A II-5). Transactions with 1 to 3 inputs and 1 to 3 outputs covered over 75% of our gathered dataset.

Table A II-2 Percentage of Lightning Funding Transactions by Inputs and Outputs

Inputs Vs Outputs	**1 Output**	**2 Output**	**3 Output**
1 Input	0.01%	0.46%	0.01%
2 Inputs	0.01%	0.18%	0.01%
3 Inputs	0.002%	0.078%	0.004%

At the next stage, to label our dataset, we developed a program that iteratively constructs a tree of previous transactions, starting from the Lightning channel opening transaction as the root,

and creates a tree with a depth of three, encompassing all parent transactions. Starting from the top transaction, the transaction output unrelated to the Lightning channel was identified as the change transaction. Additionally, each input used to create the current transaction was designated as a change output for the transaction that generated this input. Afterwards, we cleaned this dataset for each model, trained them once with Lightning data, and once without it. Finally, we performed our evaluation. A general schema of the whole process can be seen in Fig. A II-3.

4.2 Data Labelling

One of our most significant challenges was defining a labeled dataset for training our model. For this purpose, we leveraged a snapshot of channels in conjunction with the multi-input heuristic to identify potential change outputs. We established the following assumptions to label our dataset:

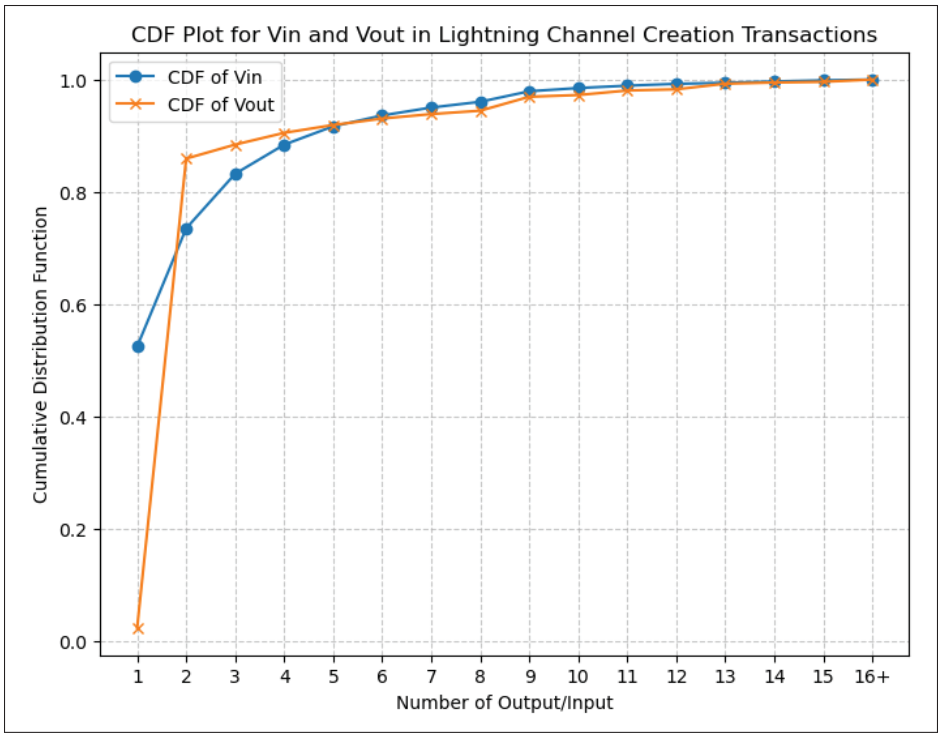

Figure A II-5 Lightning Channels Funding Transactions Input/Output CDF

1. In a Lightning Network funding transaction, the UTXO created alongside the funding transaction can be considered a change output. This aligns with the "snake pattern" described by Romiti et al.Romiti *et al.* (2021), which they found to be the most effective pattern for cross-layer deanonymization. Our labeling for funding transactions can be formalized as follows:
   Let F be the set of all lightning funding transactions such that

$$F := \{f \mid f \text{ is a lightning funding transaction}\} \tag{A II-2}$$

   and let t be a Bitcoin transaction. Suppose O_{t} is defined as the set of outputs of t:

$$O_{\mathrm{t}} := \{o_1, \ldots, o_n \mid 1 \leq n \leq 3, o_i \text{ is an output}\} \tag{A II-3}$$

   Then, the output with minimum value in $O_{\mathrm{t}} \setminus F$ is the change output.
2. Assuming that the inputs are controlled by the same entity (according to the multi-input heuristic), these inputs can be considered the change outputs of their originating transactions. Since the inputs are controlled by the same entity that created the Lightning Network funding transaction, using the multi-input heuristic implies that these inputs were also controlled by the same entity in their originating transaction, thus qualifying them as change outputs for those transactions. This concept can be formalized as follows:
   Let t_i and t_j be two Bitcoin transactions such that t_j consumes an output of t_i. Define:

$$O_{\mathrm{t}_i} = \{o_1, \ldots, o_n \mid 1 \leq i \leq 3\} \tag{A II-4}$$

   and

$$I_{\mathrm{t}_j} = \{o_1, \ldots, o_n \mid 1 \leq i \leq 3\} \tag{A II-5}$$

   as the set of outputs for t_i, and set of inputs for t_j respectively. Considering $o_j \in I_{\mathrm{t}_j}$, and $o_j \in O_{\mathrm{t}_i}$. If o_j is controlled by the same entity controlling both t_i and t_j, then o_j is the change output for t_i.

Despite our simple assumptions, there might be potential issues that could arise from them:

1. The multi-input heuristic may yield false positives if the preceding transaction is controlled by a different entity. This is particularly prevalent in transactions originating from exchanges or similar services, which typically feature higher numbers of inputs or outputs. Additionally, the likelihood of generating false positives increases as we trace further back in the transaction chain. By restricting the number of inputs and outputs used in our final dataset and limiting the depth of iterations to three, we have attempted to mitigate these risks.
2. Choosing the smallest output as the change output in a Lightning funding transaction may lead to false positives and false negatives. For instance, consider a transaction with three outputs: one for Lightning funding and two regular outputs, where the smallest output is a payment to another user. To assess this risk, we filtered for transactions in our dataset that meet this criterion. We discovered that only 163 transactions out of 29,537 Lightning funding transactions matched this criterion, representing less than 0.5% of our dataset. Therefore, we believe this will not significantly impact the overall accuracy of the model. Another scenario could occur if an entity opens a Lightning channel and sends the change to a different entity, potentially leading to inaccurate results.

In conclusion, while these assumptions may not apply to more complex transactions, we believe that the methodological framework we have presented is sufficiently robust for the scope of this study.

4.3 Evaluation Metrics

To evaluate each model correctly, we conducted our tests under comparable conditions and calculated the Accuracy, Precision, Recall, and F1 Score for each. Here is how we computed

these metrics:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{A II-6}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{A II-7}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{A II-8}$$

$$\text{F1 Score} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{A II-9}$$

Accuracy indicates the proportion of predictions that were correct across all predictions made. Precision identifies the proportion of identified positives that were actual positives. Recall quantifies the model's ability to identify all relevant instances accurately. Lastly, the F1 Score, which is the harmonic mean of precision and recall, serves as a single metric that balances both precision and recall.

Table A II-3 Comparison of Model Performance Metrics

Metric	BERT	BERT+LN data	RF	RF+LN data	XGBoost	XGBoost+LN data
Accuracy	0.85	0.94	0.87	0.87	0.87	0.87
Precision	0.80	0.88	0.99	0.99	0.99	0.99
Recall	0.78	0.88	0.25	0.25	0.25	0.25
F1 Score	0.79	0.88	0.40	0.40	0.40	0.40

5. Performance Evaluation

In this section, we will present our evaluation results. First, we present the parameters of the experiment, then we provide detailed performance metrics of the various models tested, and finally, we discuss the implications of our findings and potential avenues for future research.

5.1 Experimental Setup

Our training setup consisted of an Nvidia RTX 2070 Super graphics card, 32 GB of DDR5 memory, and an Intel i5-13600K CPU. Our dataset comprised 623,586 of individual transaction outputs, which formed 27,209 transactions (including the initial Lightning Network funding

transaction and subsequent transactions resulting from recursion). We divided these transactions into 19,046 (70%) for training and 8,163 (30%) for testing.

In our modified BERT, the number of transformer blocks is 2, in comparison to the original BERT which suggested 12, to prevent over-fitting because our database was relatively small. Each transformer block has a hidden layer size of 96, two attention heads, and a dropout rate of 60%.

For tuning the random forest classifier, we employed a grid search algorithm. The optimal parameters based on the Grid search for achieving the highest accuracy were:

- `n_estimators`: 50
- `max_features`: `sqrt`
- `max_depth`: 10
- `min_samples_split`: 10
- `min_samples_leaf`: 2

Similarly, we utilized a grid search algorithm to identify the best parameters for the XGBoost model. The best parameters that we found for the highest accuracy were:

- `n_estimators`: 20
- `max_depth`: 10
- `learning_rate`: 0.1
- `colsample_bytree`: 1

For each model, we trained and tested both with and without the inclusion of Lightning Network data, keeping other parameters constant. This approach allowed us to focus specifically on the effect of adding Lightning Network information to our model's performance.

5.2 Results

As shown in the Table A II-3 , the modified BERT model with Lightning data achieved the best overall results, reaching 94% accuracy, except for precision. The closest competitors, the RF

and XGBoost models, achieved 87% accuracy, marking a 7% gap. Although RF and XGBoost models with and without Lightning data exhibited higher precision, their recall scores were significantly lower than other models, indicating a tendency to miss many positive samples. An interesting observation is the lack of ability of XGBoost and Random Forest models to utilize the Lightning Network information.

5.3 Discussion

By analyzing the important features for the Random Forest classifier, we see that the most important features to detect a change output was the number of unique txids in a transaction, indexes, address type, and locktime. Our hypothesis is that the ability of the modified BERT model to improve its statistics by adding the Lightning Network information can be due to the ability of transformer encoder's to find complex patterns between different inputs. For further research, we would like to use more complex data to test the BERTs ability in figuring out the change output in more challenging tasks.

6. Conclusion

This study utilized a modified BERT model to investigate change output detection in Bitcoin transactions, integrating Lightning Network data to enhance model accuracy. The model demonstrated a significant improvement, achieving 94% accuracy with Lightning Network data, compared to 85% without it, highlighting the benefits of cross-chain data analysis. Our Proof-of-Concept underscores the potential of incorporating Lightning Network information for change address detection and highlights the applicability of transformers for detecting change outputs and deanonymizing addresses in UTXO-based blockchains.

However, the research faced limitations such as potential biases from heuristic-based labeling and the computational constraints of analyzing deep transaction histories. For future work, we aim to test our model on a larger dataset to verify and enhance the effectiveness of our approach. As the data complexity increases, we believe it would be beneficial to implement the model with relational positional encoding Shaw, Uszkoreit & Vaswani (2018) as well. Additionally,

exploring the adaptation of this model to account-based models like Ethereum could yield further insights and broaden the applicability of our findings.

BIBLIOGRAPHY

Androulaki, E., Karame, G. O., Roeschlin, M., Scherer, T. & Capkun, S. (2013). Evaluating user privacy in bitcoin. *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, pp. 34-51.

Back, A. (2002). Hashcash - A Denial of Service Counter-Measure. Retrieved from: http://www.hashcash.org/papers/hashcash.pdf.

Binance. [binance]. (2023, June, 20). #Binance is working to integrate the #Bitcoin Lightning Network for deposits and withdrawals. Some eagle-eyed users spotted our new lightning nodes recently. Yes - that's us! However, there's still more tech work to be done. We'll update once Lightning is fully integrated. [Tweet]. Retrieved from: https://twitter.com/binance/status/1671042638592589826.

BitcoinProject. (2024). Bitcoin Core [Website]. Retrieved from: https://bitcoin.org/en/bitcoin-core/.

Bitfinex. (2020, March, 20). Struck by the Lightning [Blog post]. Retrieved from: https://blog.bitfinex.com/announcements/struck-by-the-lightning.

Blockchain.com. (2024). Fees Per Transaction (USD). Retrieved from: https://www.blockchain.com/explorer/charts/fees-usd-per-transaction.

Braiins. [BraiinsMining]. (2024, Feburary, 21). Introducing Lightning payouts We are excited to be the first mining pool using the Lightning Network. Our miners can now instantly receive rewards with no minimums or fees. [Tweet]. Retrieved from: https://x.com/BraiinsMining/status/1760319741560856983.

Breiman, L. (2001). Random forests. *Machine learning*, 45, 5-32.

Chen, T. & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (KDD '16), 785–794. doi: 10.1145/2939672.2939785.

Dashjr, L. (2024, July, 30). Bitcoin Nodes. Retrieved from: https://luke.dashjr.org/programs/bitcoin/files/charts/software.html.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved from: https://arxiv.org/abs/1810.04805.

Harrigan, M. & Fretter, C. (2016). The unreasonable effectiveness of address clustering. *2016 intl ieee conferences on ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress (uic/atc/scalcom/cbdcom/iop/smartworld)*, pp. 368-373.

Herrera-Joancomartí, J., Navarro-Arribas, G., Ranchal-Pedrosa, A., Pérez-Solà, C. & Garcia-Alfaro, J. (2019). On the difficulty of hiding the balance of lightning network channels. *Proceedings of the 2019 ACM asia conference on computer and communications security*, pp. 602-612.

Hu, S., Zhang, Z., Luo, B., Lu, S., He, B. & Liu, L. (2023). BERT4ETH: A Pre-trained Transformer for Ethereum Fraud Detection. *Proceedings of the ACM Web Conference 2023*, pp. 2189–2197.

Kappos, G., Yousaf, H., Piotrowska, A., Kanjalkar, S., Delgado-Segura, S., Miller, A. & Meiklejohn, S. (2021). An empirical analysis of privacy in the lightning network. *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25*, pp. 167-186.

Kraken. (2022, April, 1). Kraken Now Supports Instant Lightning Network BTC Transactions [Blog post]. Retrieved from: https://blog.kraken.com/product/kraken-now-supports-instant-lightning-network-btc-transactions.

Lee, C., Maharjan, S., Ko, K. & Hong, J. W.-K. (2020a). Toward Detecting Illegal Transactions on Bitcoin Using Machine-Learning Methods. *Blockchain and Trustworthy Systems*, pp. 520-533.

Lee, C., Maharjan, S., Ko, K., Woo, J. & Hong, J. W.-K. (2020b). Machine Learning Based Bitcoin Address Classification. *Blockchain and Trustworthy Systems*, pp. 517-531.

LightningLabs. (2024). Lightning Network Daemon. Retrieved from: https://github.com/lightningnetwork/lnd.

Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M. & Savage, S. (2013). A fistful of bitcoins: characterizing payments among men with no names. *Proceedings of the 2013 conference on Internet measurement conference*, pp. 127-140.

Möser, M. & Narayanan, A. (2022). Resurrecting address clustering in bitcoin. *International Conference on Financial Cryptography and Data Security*, pp. 386-403.

Najjar, F., Naha, R. T., Feridani, M. M., Dekhil, O. & Zhang, K. (2024). Change Address Detection in Bitcoin using Hierarchical Clustering. *2024 IEEE World Forum on Public Safety Technology (WFPST)*, pp. 42-48. doi: 10.1109/WFPST58552.2024.00015.

Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from: https://bitcoin.org/bitcoin.pdf.

Nerurkar, P., Busnel, Y., Ludinard, R., Shah, K., Bhirud, S. & Patel, D. (2020). Detecting Illicit Entities in Bitcoin using Supervised Learning of Ensemble Decision Trees. *Proceedings of the 10th International Conference on Information Communication and Management*, (ICICM '20), 25–30.

Poon, J. & Dryja, T. (2015). The bitcoin lightning network. *Scalable on-chain instant payments*, 20–46.

Ramos Tubino, R., Robardet, C. & Cazabet, R. (2022). Towards a better identification of Bitcoin actors by supervised learning. *Data & Knowledge Engineering*, 142, 102094. doi: https://doi.org/10.1016/j.datak.2022.102094.

Reid, F. & Harrigan, M. (2013). An Analysis of Anonymity in the Bitcoin System. In Altshuler, Y., Elovici, Y., Cremers, A. B., Aharony, N. & Pentland, A. (Eds.), *Security and Privacy in Social Networks* (pp. 197-223). New York, NY: Springer New York.

Romiti, M., Victor, F., Moreno-Sanchez, P., Nordholt, P. S., Haslhofer, B. & Maffei, M. (2021). Cross-layer deanonymization methods in the lightning protocol. *International Conference on Financial Cryptography and Data Security*, pp. 180-204.

Ron, D. & Shamir, A. (2013). Quantitative analysis of the full bitcoin transaction graph. *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, pp. 6-24.

Shao, W., Li, H., Chen, M., Jia, C., Liu, C. & Wang, Z. (2018). Identifying Bitcoin Users Using Deep Neural Network. *Algorithms and Architectures for Parallel Processing*, pp. 178-192.

Shaw, P., Uszkoreit, J. & Vaswani, A. (2018). Self-Attention with Relative Position Representations. Retrieved from: https://arxiv.org/abs/1803.02155.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30, 5998–6008.

Von Arx, T., Tran, M. & Vanbever, L. (2023). Revelio: A Network-Level Privacy Attack in the Lightning Network. *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pp. 942-957.

Wallet, S. (2022a). Ricochet. Retrieved from: https://web.archive.org/web/20220317191404/https://samouraiwallet.com/ricochet.

Wallet, S. (2022b). Samourai Wallet. Retrieved from: https://web.archive.org/web/20220201061906/https://samouraiwallet.com/.

Wallet, S. (2022c). STONEWALL. Retrieved from: https://web.archive.org/web/20210913214623/https://samouraiwallet.com/stonewall.

Wallet, S. (2022d). Whirlpool. Retrieved from: https://web.archive.org/web/20220119122808/https://samouraiwallet.com/whirlpool.

Zhang, Y., Wang, J. & Luo, J. (2020). Heuristic-Based Address Clustering in Bitcoin. *IEEE Access*, 8, 210582-210591.