Blockchain Recommender Systems using Blockchain Data

by

Sean Khatiri

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Management Sciences

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Blockchain systems allow a network of pseudo-anonymous users (identified only by their public key) to maintain a secure transaction ledger in a decentralized manner. Transactions are executed and recorded on the ledger by programs called *smart contracts*. Decentralized applications (dApps) can be built on top of blockchains, for tasks such as exchanging cryptocurrencies and other digital assets, without the need for trusted third parties such as banks. As is the case with traditional Web applications, personalization is key to user acquisition and retention in decentralized systems. We therefore ask the following question in this thesis: how can we build effective blockchain recommender systems?

To answer this question, we turn to collaborative filtering, a popular recommendation approach that captures similarities among users in terms of their transaction histories. For example, if two users liked movies a, b, c, and d, and the first user additionally liked movie e, then collaborative filtering may suggest movie e to the second user. The main technical challenge we address is how to map smart contract code to the underlying items or concepts that may be recommended, e.g., a smart contract that facilitates an in-game purchase using Bitcoin may map to the "gaming" concept. Using this mapping and real-world data from the Ethereum network, which is the largest smart-contract-enabled platform, we test two collaborative filtering systems: a simple and fast Matrix Factorization (MF) algorithm and a more complex one based on Graph Neural Networks (GNN). Our empirical results show that GNN outputs more effective recommendations, at the expense of latency. We conclude with an overview of a blockchain-native implementation of our framework as a decentralized recommendation service, and we discuss the corresponding practical challenges such as incentive mechanisms (tokenomics).

Acknowledgements

I wish to extend my gratitude to Professor Lukasz Golab, my supervisor, for his unwavering support, patient mentorship, and invaluable guidance during the course of my research endeavors.

My heartfelt appreciation also goes to Professor Morteza Zihayat, Professor Mehdi Kargar, and Professor Jaroslaw Szlichta for their invaluable guidance and patient counsel that have profoundly contributed to the advancement of my research.

Dedication

This is dedicated to my lovely wife and family, the ones I love.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CF Collaborative Filtering 3

$\mathbf{CF_{GNN}}$ Collaborative Filtering Graph Neural Network 6

$\mathbf{CF_{MF}}$ Collaborative Filtering Matrix Factorization 5

dApps Decentralized Applications 1

GNN Graph Neural Network 5

$\mathbf{Hybrid_{GNN\text{-}CLUSTERING}}$ Hybrid Graph Neural Network with clustering method 6

$\mathbf{Hybrid_{GNN\text{-}SBERT}}$ Hybrid Graph Neural Network with Sentence-BERT method 6

$\mathbf{Hybrid_{MF\text{-}CLUSTERING}}$ Hybrid Matrix Factorization with clustering method 5

$\mathbf{Hybrid_{MF\text{-}SBERT}}$ Hybrid Matrix Factorization with Sentence-BERT method 5

MF Matrix Factorization 5

PoS Proof of Stake 12

PoW Proof of Work 12

SBERT Sentence-BERT 5

Chapter 1

Introduction

1.1 Background

The modern online world, referred to as Web3, is becoming decentralized due to blockchain platforms. Here, interactions among pseudo-anonymous users (identified only by their public key) who may not necessarily trust each other are facilitated by consensus algorithms and recorded in a secure ledger. The ledger is replicated across a distributed network of *validator* nodes that earn rewards for maintaining the ledger. The Bitcoin network [56] was the first popular example, allowing users to trade the Bitcoin cryptocurrency without the use of a bank or any other trusted third party.

The Bitcoin platform supports transactions that transfer Bitcoins from one user (public key) to another. A more general blockchain platform then emerged – Ethereum [13] – with the addition of smart contracts, which are pieces of code that execute arbitrary transaction logic. Smart contracts enable Decentralized Applications (dApps) to be built on top of blockchains, for tasks such as exchanging cryptocurrencies or other assets, or decentralized auctions.

Figure 1.1 shows an example of two dApps: one for cryptocurrency swap and one for a digital asset auction. To deploy a dApp on a blockchain, the dApp developer writes smart contracts, shown in the figure as written in the Solidity programming language, to implement the desired dApp features. For example, the *transfer* function of the *SimpleTokenSwap* contract first checks the sender's account balance and then subtracts the amount transferred from the sender's balance and adds it to the recipient's balance. End users can then connect to the dApp front end, and user interactions trigger the corresponding

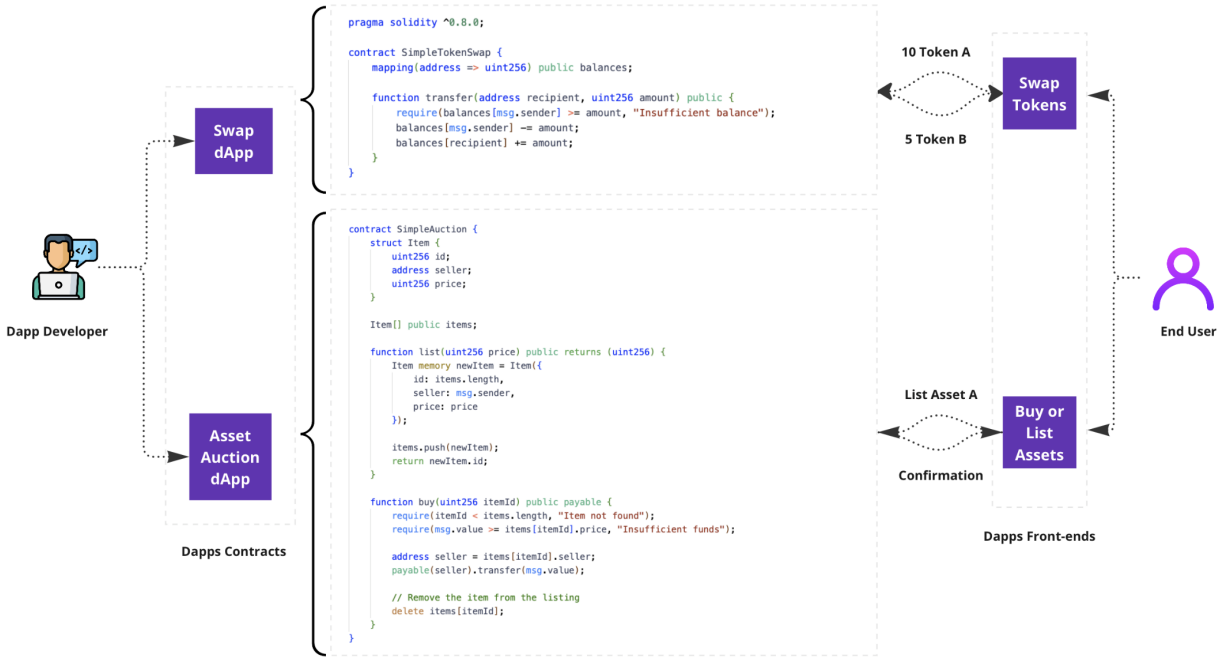


Figure 1.1: An example of user-contract interactions in decentralized auction and token transfer dApps

smart contracts to execute the transaction logic and record the result on the ledger. In the figure, we show a user interacting with the Swap dApp to exchange ten token As for five token Bs (using the *transfer* function of the *SimpleTokenSwap* contract), and listing and buying assets on the auction dApp (using the *list* and *buy* functions of the *SimpleAuction* contract). Notably, these transactions are executed and maintained on the blockchain platform, without the need for intermediaries such as banks or lawyers. Instead, each execution of smart contract requires a transaction fee that the user pays to the validator nodes in the blockchain network.

A number of further innovations since the introduction of Ethereum have contributed to the growth of Web3. Oracles, i.e., trusted data sources, were introduced to allow smart contracts to access external (off-chain) data such as weather forecasts. Oracles were initially implemented as services provided by centralized entities, but decentralized oracle protocols such as ChainLink [10] have recently emerged to fetch external data from a set of nodes participating in the blockchain network itself. Likewise, querying and indexing blockchain data can now be done in a decentralized manner through the Graph protocol [78], eliminating the need to trust third parties to provide these services.

We view personalization as the next big milestone in the growth of Web3. As in

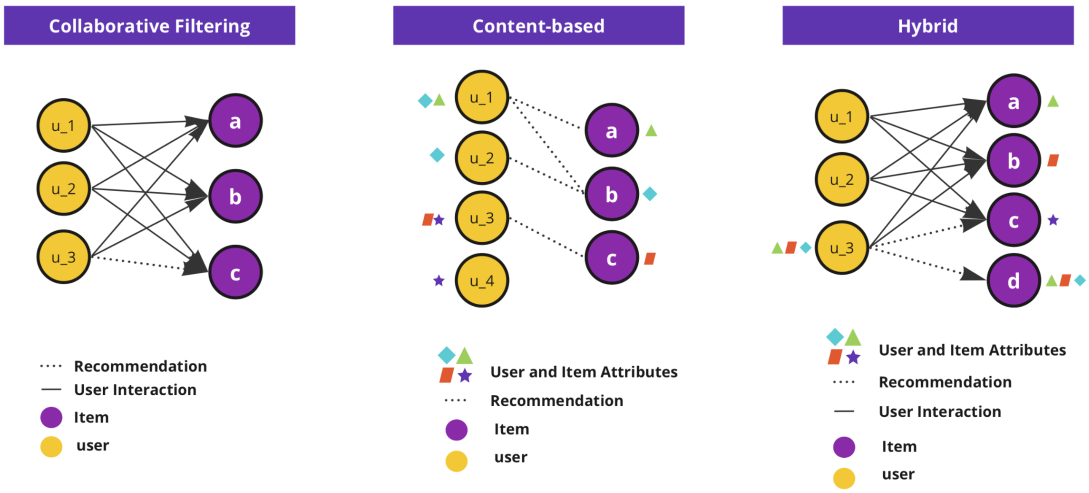


Figure 1.2: Collaborative Filtering (CF), Content-based, and Hybrid recommender systems

the current Web2 (with centralized social networks and centralized e-commerce entities), personalization is critical for user acquisition, satisfaction, and retention. Research has shown that one of the most effective enablers of personalization is the use of recommender systems [25]. While there has been extensive research in designing recommender systems for centralized applications, the area of dApp recommender systems remains under-explored [4]. This is the gap we intend to fill in this thesis.

We begin with an overview of recommender systems. Usually, users are the entities that receive recommendations, and items are the entities that are recommended to users. Recommender systems can be categorized by the data they use to make recommendations: user interactions with the available items or services, user attributes, or item attributes. Based on this categorization, there are three approaches, illustrated in Figure 1.2: Collaborative Filtering (CF), content-based, and Hybrid [38]. CF methods focus solely on interactions. For example, if user u_1 interacted with items a, b, and c, and user u_2 interacted with items a and b, then item c could be recommended to user u_2. This approach considers only the similarity in interactions without accounting for user or item attributes. Conversely, pure content-based methods consider only the similarities among user and item attributes. For instance, if user u_1 frequently watches video game streams, gaming items might be recommended to them based on content similarity. Finally, Hybrid methods take into account both interactions and the attributes of users and items. For example, suppose user u_1 and user u_2 interacted with items a, b, and c, and user u_3 interacted with items a and b. If the profile of item d is similar to the preferences of the user u_3, then item c could

be recommended due to the similarity in interactions, while item d could be recommended due to the similarity in attributes.

Translating the recommendation problem to the blockchain context, users are represented by public keys (i.e., user addresses on the blockchain) and items by smart contracts, which also have addresses. Each transaction is thus an interaction between a user address and a contract address, and this pair of addresses (plus the output of the contract) is recorded on the ledger. As a result, simple CF methods can be used for dApp recommender systems using only the transaction records available on the ledger[1]. On the other hand, pure content-based methods may not apply, as the pseudo-anonymous users on the blockchain are known only by their public keys and lack additional descriptive attributes. In principle, linking blockchain addresses to Web profiles (i.e., those available on centralized applications such as social networks) could create more detailed user profiles; however, this raises privacy concerns and is not straightforward since some users may not disclose their addresses on these platforms. Finally, Hybrid approaches can be considered, based on the similarity of items in terms of their attributes or descriptions. Hybrid approaches may produce recommendations that would go unnoticed with pure CF methods that only consider interactions. For example, Figure 1.3 illustrates three SimpleLending dApps where lenders can deposit (using the *deposit* method) cryptocurrency (i.e., tokens) to earn interest and borrowers can take out loans using those deposited tokens (using the *borrow* method). After some time, lenders can withdraw their deposited amount along with the interest earned (using the *withdraw* method). Each lending contract specifies the interest rate and the type of tokens it accepts (i.e., deposit type). Suppose users u_1 and u_2 interact with the SimpleLendingA and SimpleLendingB contracts to deposit tokens. When user u_3 interacts with SimpleLendingA, a CF recommender method may suggest SimpleLendingB. However, with only user interactions, CF cannot suggest SimpleLendingC. A Hybrid method can find similarities between SimpleLendingA and SimpleLendingC, such as the same type of accepted deposits, and thus can recommend SimpleLendingC to user u_3.

However, the technical challenge is how to identify similar smart contracts. As illustrated in Figure 1.1, smart contracts are pieces of code written in a programming language such as Solidity, which differentiates them from product descriptions or movie genres found in e-commerce or movie recommender systems. This raises a crucial question for dApp recommender systems: ***How can we effectively provide recommendations based on user interactions with smart contracts?***

[1]Note that when a new user arrives, since there are no previous transactions (i.e., interactions) associated with them, collaborative filtering methods may not work effectively. This issue is known as the Cold Start Problem (CSP) in the literature on recommender systems.

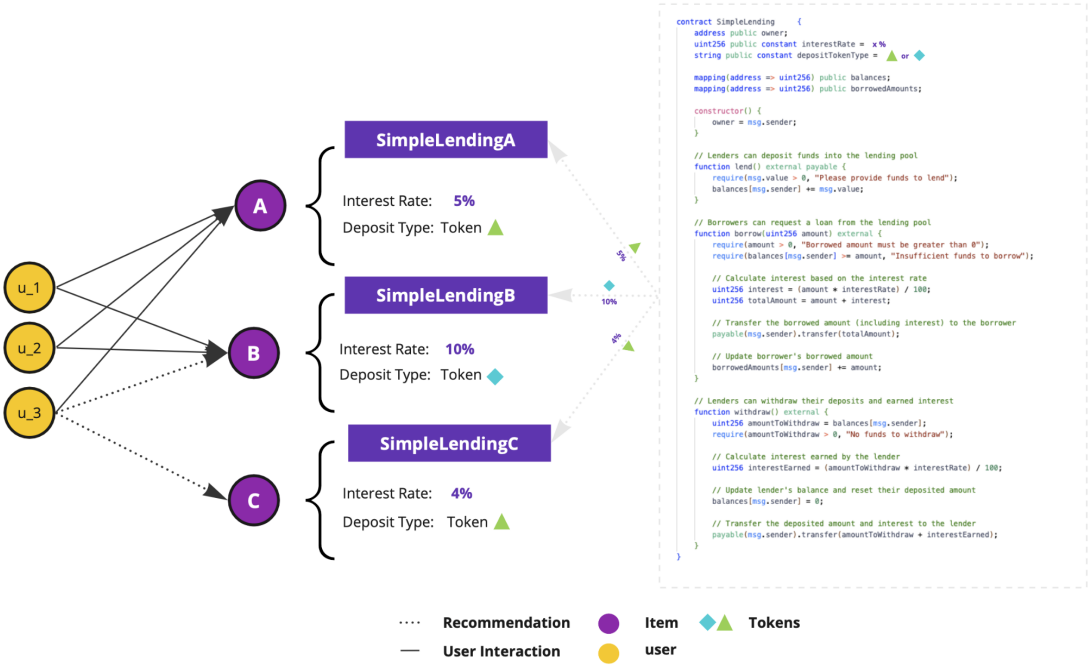


Figure 1.3: An example of Hybrid dApp recommender system in a decentralized lending dApp

We propose two methods to identify similar contracts:

1. A clustering method that segments the contracts according to their comments.
2. An embedding method that builds a dense numeric representation (i.e., embedding) for each contract based on the comments within the smart contract code. Here, contracts with semantically similar comments have similar embedding vectors (i.e., the distance between their embedding vectors is small). We utilize Sentence-BERT (SBERT [63]) pre-trained model to calculate contract embeddings.

In this thesis, we consider six recommender systems, illustrated in Figure 1.4. We position our choices in a two-dimensional space according to data needs (pure CF, $\text{Hybrid}_{\text{CLUSTERING}}$, and $\text{Hybrid}_{\text{SBERT}}$) and model complexity (simple Matrix Factorization (MF) and Graph Neural Network (GNN)). We start with a simple and fast MF-based CF as one of the popular implementations of CF recommender systems [43] (CF_{MF} shown at the bottom left in Figure 1.4). Transitioning to $\text{Hybrid}_{\text{MF}}$ methods, we add contract cluster IDs and contract embeddings (i.e., SBERT embeddings) to CF_{MF} method ($\text{Hybrid}_{\text{MF-CLUSTERING}}$ and $\text{Hybrid}_{\text{MF-SBERT}}$ displayed at the middle and top left in Figure 1.4 respectively). These

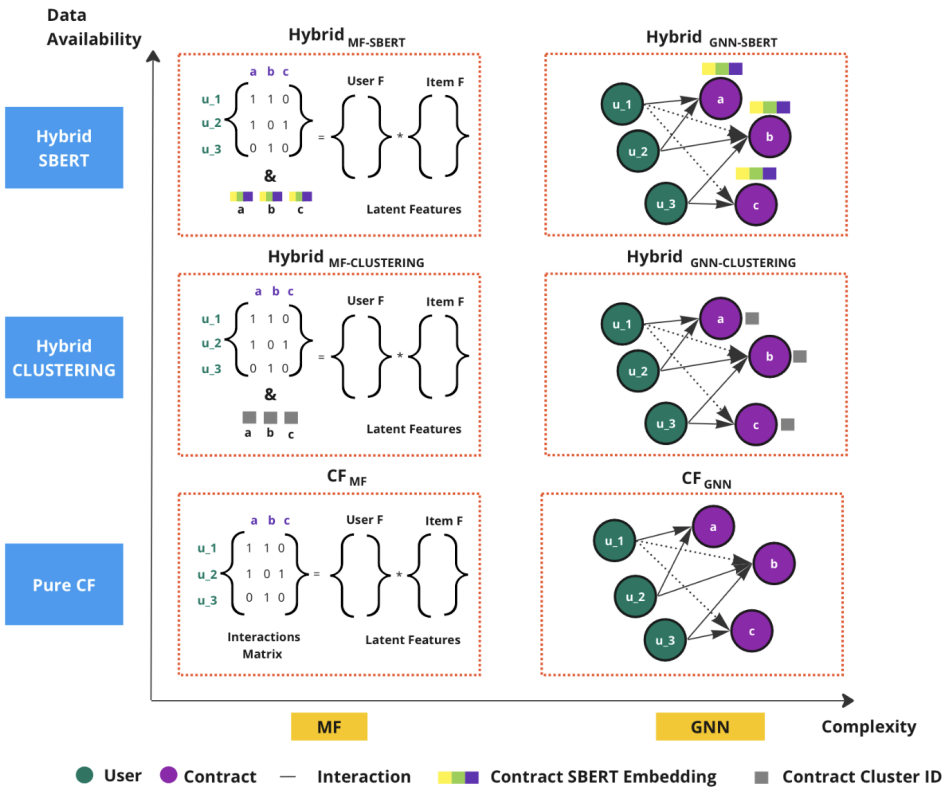


Figure 1.4: Matrix Factorization and Graph Neural Network implementation of pure Collaborative Filtering and Hybrid recommender systems.

Hybrid methods account for the similarity between contracts, enabling the dApp recommender systems to expand their recommendation capabilities by considering similarities within interactions data as well as contract attributes.

Transitioning to neural methods, various architectures can be adopted, such as Neural Matrix Factorization (Neural MF) [64] and GNNs [91]. Given the bipartite graph structure of user-contract interactions, GNN architectures are appropriate. This is mainly because GNNs can capture the complex and nonlinear structure of data, where relationships extend beyond immediate neighbors, and interactions are not limited to being pairwise [91] (as is typically considered in Neural MF methods). Figure 1.4 illustrates the CF_{GNN}, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$ recommender systems. Similar to MF methods, we add contract cluster IDs and contract embeddings (i.e., SBERT embeddings) to CF_{GNN} method ($\text{Hybrid}_{\text{GNN-CLUSTERING}}$ and $\text{Hybrid}_{\text{GNN-SBERT}}$ displayed at the

middle and top right in Figure 1.4 respectively). For example, if user u_1 interacts with item a, and user u_2 interacts with items a and b, and considering that user u_3 interacts with items b and c, GNN methods can enable the higher-order recommendation of item c to user u_1. These models achieve this by learning the embeddings of each user and contract to perform recommendations by predicting whether a link should exist between the user and contract nodes. The pure CF_{GNN} only considers user-contract interactions, while the $\text{Hybrid}_{\text{GNN-CLUSTERING}}$ and $\text{Hybrid}_{\text{GNN-SBERT}}$ also incorporate contract cluster IDs and SBERT embeddings respectively alongside the interactions data.

1.2 Our Contributions

Our contributions are summarized below:

- Conceptually, we formulate a dApp recommender system problem that suggests smart contracts to users. To the best of our knowledge, the only prior attempt to formalize a dApp recommender system was restricted to digital tokens as items [94]. In contrast, we propose a general blockchain recommendation framework, treating smart contracts as items to recommend and considering smart contract similarity in addition to user-item interactions when making recommendations. Designing a recommender system that can handle any smart contract, beyond digital token transfer, is critically important because future blockchains are expected to extend beyond cryptocurrency and digital tokens.

- As a technical contribution, we propose two methods to enhance CF dApp recommender system by incorporating additional contract information: (1) a clustering approach that groups similar contracts together (2) contract embeddings. Using these methods, illustrated in Figure 1.4, we integrate MF and GNN recommender architectures into both CF and Hybrid dApp recommender systems.

- Empirically, we compare the effectiveness and efficiency of Pure CF and Hybrid dApp recommender systems. To this end, we create a dataset comprising 400,000 real Ethereum transactions that took place between 2018 and 2023. These transactions include source and target addresses (i.e., user and smart contract public keys), thus capturing user-contract interactions. We compare these approaches with a baseline that recommends popular contracts to all users. Furthermore, to enhance our understanding of blockchain recommender systems' performance, we examine the results of the $\text{Hybrid}_{\text{GNN-SBERT}}$ approach on both our Ethereum dataset and the MovieLens dataset [27] in Subsection 4.1.2.

1.3 Thesis Overview

The structure of this thesis is as follows:

In Chapter 2, we discuss the fundamentals of blockchains and smart contracts. Following this, we describe CF and Hybrid recommender methods, including Matrix Factorization and Graph Neural Network methods.

In Chapter 3, we begin by explaining how our transaction dataset is constructed. We also conduct an exploratory data analysis (EDA). Then, we describe the MF and GNN dApp recommender systems architectures both for pure CF and Hybrid recommender methods.

In Chapter 4, we begin with the results of our popular contract recommender system (POP). This system operates by recommending the same popular contracts to all users. Subsequently, we discuss the performance of the advanced recommender systems we have examined.

In Chapter 5, we summarize our findings, outline directions for future work, and consider the practical aspects of a decentralized implementation of dApp recommender systems.

Chapter 2

Preliminaries and Related Work

In this chapter, we start with an overview of blockchains (Section 2.1). We then proceed to summarize related work on blockchain data analytics (Section 2.2), recommender systems (Section 2.3), and blockchain recommender systems (Section 2.4).

2.1 Blockchains

A blockchain system is an example of state machine replication [24], described as follows. Consider a state machine $\mathcal{M}$ represented by the tuple:

$$\mathcal{M} = (S, s_0, \Sigma, \Lambda, \delta)$$

where:

- S denotes a particular status of the system at a specific point in time, representing the "world state" in a blockchain context. The world state is application-dependent: e.g., the balance of each account, the current owner of each asset, etc.
- s_0 is the initial state, known as *Genesis Block* in blockchain context.
- Σ signifies the set of possible transactions that users can perform to modify the state.
- $\delta : S \times \Sigma \to S$ is the state transition function. Given a state s and a transaction t, $\delta(s, t)$ produces a new state s'.

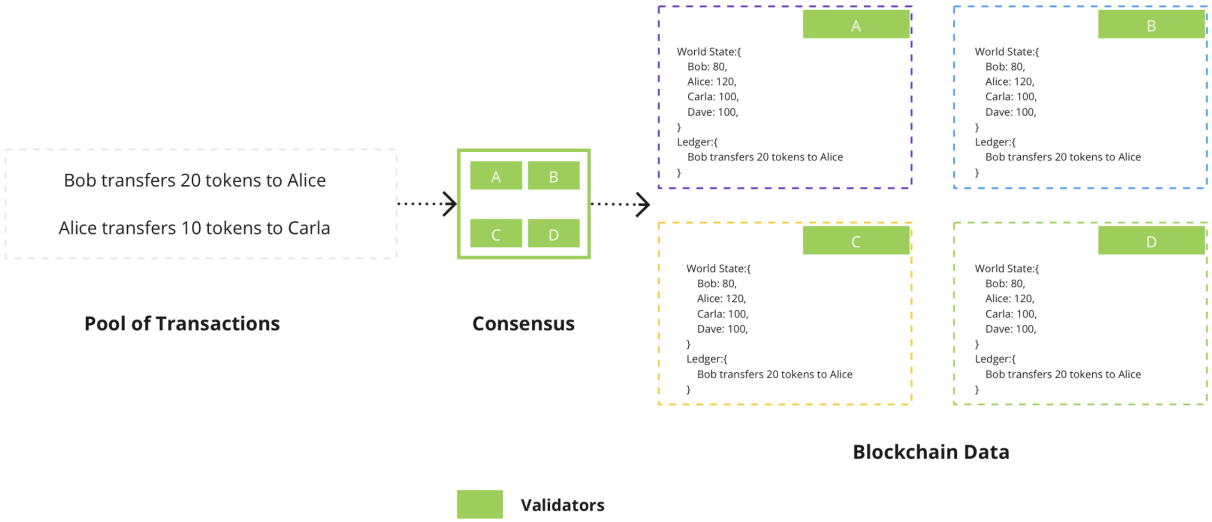


Figure 2.1: A comprehensive illustration of primary components of blockchains.

The state transition function δ takes a state s and a transaction t from Σ as inputs and produces a new state s', effectively reflecting how t modifies s. Figure 2.1 demonstrates the primary components of blockchains. First, users submit their transactions. Validators then select a batch of transactions (i.e., a block) based on the combined transaction fees of the block's transactions. Consequently, users can offer higher fees for their transactions to be selected by validators sooner. The first transaction (e.g., 'Bob transfers 20 tokens to Alice') is selected. Additionally, users can submit a transaction by invoking the *sayHello()* method of the *HelloWorld* contract in blockchains that support smart contracts (such as Ethereum) as discussed in Section 1.1. In Subsection 2.1.3, we provide further details on smart contracts. Validators, as a replicated set of state machines, then come to an agreement (i.e., consensus) on the world state. We first discuss blockchains as replicated state machines and the need for them to tolerate faults in Subsection 2.1.1. We then elaborate on consensus algorithms in Subsection 2.1.2. After the transaction 'Bob transfers 20 tokens to Alice' is processed and agreed upon by validators, they update their world state and keep a record of the transactions (i.e., ledger). In the initial world state (i.e., known as the Genesis block in blockchains), both Bob and Alice have 100 tokens. After transferring 20 tokens from Bob to Alice, the world state reflects their new balance (i.e., Bob:80, Alice:120). The final world state can be derived by sequentially replaying the transactions recorded in the ledger. However, in practice, this approach is inefficient due

to the computation-intensive nature of the task[1]. Consequently, validators maintain both the ledger and the world state as distinct data structures. In Subsection 2.1.4, we discuss the ledger and world state data structure in detail.

2.1.1 Fault Tolerance

In a blockchain, users should be able to interact with each other without requiring mutual trust or trust in third parties. If only one state machine exists, users would need to trust it, which contradicts the principle of eliminating the need for trust in intermediaries. Specifically, considering a state machine can be faulty, such as being unavailable to provide the world state due to system errors. One intuitive approach is to have a set of replicated state machines to compare their world states when a conflict arises. Assuming the majority of nodes are well-behaved, a majority vote can determine the correct world state. For blockchains to work correctly, there needs to be a certain number of nodes to tolerate certain faulty nodes such as fail-stop (i.e., unavailability due to system error), malicious nodes (i.e., nodes capable of providing contradictory world states), and Byzantine faulty nodes (i.e., nodes capable of lying, refusing to provide the world state, and so on). In this section, we examine this threshold for blockchains to be fail-stop and Byzantine fault-tolerant.

In an ideal scenario where all nodes are well-behaved, if all nodes except one become unavailable due to system errors (i.e., fail-stop), users need to receive the world state from the last remaining node to ensure at least one correct response [24]. However, considering malicious nodes that can provide contradictory world states, a majority vote is required to determine the correct world state. Assume there are f votes from malicious nodes. Therefore, at least $2f + 1$ votes are needed, considering $f + 1$ votes from well-behaved nodes. This leads to a requirement of $2f + 1$ nodes for the system to be crash fault-tolerant [70]. For instance, in a network comprising Alice, Bob, and Dave, if Alice becomes unavailable, users can receive the world state from Bob or Dave, which is sufficient to find at least one correct world state. However, if Alice provides a false world state, users receiving the world state from both Alice and Bob will encounter two contradictory world states. Consequently, a majority vote is necessary. When Dave provides the world state, users can easily select the correct world state by a 2-to-1 vote, with Alice's world state being the incorrect one. In this scenario, a system with three nodes can tolerate one malicious node, fulfilling the $2f + 1$ criterion.

[1]For the Ethereum blockchain, as of the date of this thesis, approximately 1 million transactions are added to the Ethereum ledger daily

Conversely, let's consider f Byzantine faulty nodes capable of providing contradictory world states, refusing to provide any response, or even returning the correct world state. In an ideal scenario, users can query $f + 1$ nodes until they receive at least one correct response from a well-behaved node. However, the malicious nodes can lie. Now, users have f lies and just 1 honest response. Therefore, there needs to be a majority voting with $2f+1$ votes ($f + 1$ honest responses and f lies). However, upon a voting request, the malicious nodes can refuse to provide any responses. Since we needed $2f + 1$ votes, there should be $2f+1$ well-behaved nodes to finalize the voting. Therefore, with $2f+1$ well-behaved nodes and f malicious nodes, we need $3f+1$ nodes for the system to be Byzantine Fault Tolerant (BFT) [46, 24]. Extending the previous example, introduce Carla into the network, creating a four-node system. Carla is a Byzantine faulty node. Users receive contradictory world states from Alice and Carla, therefore necessitating majority voting. Users need at least 3 votes (i.e., 1 faulty vote and 2 correct votes) to determine the correct world state. However, upon actual voting, Carla refuses to provide any responses. Therefore, all 3 well-behaved nodes (i.e., Alice, Bob, and Dave) can provide their voting. Consequently, this four-node network meets the requirements for Byzantine fault tolerance, as it would necessitate at least $3f + 1 = 4$ reliable nodes to tolerate $f = 1$ Byzantine faulty node.

2.1.2 Consensus

Similar to the challenges users face when trying to determine the correct world state, nodes may need to receive the world state from other nodes (in scenarios such as being unavailable). Therefore, there needs to be a mechanism for nodes to agree on the world state and the subsequent steps (i.e., consensus). Essentially, consensus is a mechanism that determines who has the right to propose a block by utilizing a Proof-of-X strategy [24]. In Proof-of-Work (PoW) consensus algorithms, this right is determined by solving a mathematical puzzle; the one who finds the solution sooner can propose the valid block. This puzzle is hard to solve but easy to verify, allowing others to verify both the proof and the block. On the other hand, in Proof of Stake (PoS) consensus algorithms, nodes that have locked a certain amount of tokens (i.e., staking) are selected based on the amount of their staked tokens, randomness, and so on. Therefore, they propose the block by showing proof of their stake. These consensus algorithms must inherently offer incentives to nodes that accurately process new transactions, while concurrently penalizing malicious nodes. Distinct blockchain architectures have introduced unique solutions to the consensus challenge; notably, Bitcoin's architecture introduced the PoW method, followed by Ethereum's adoption of the PoS consensus algorithms.

In PoW consensus algorithms, validators validate new blocks by executing a state tran-

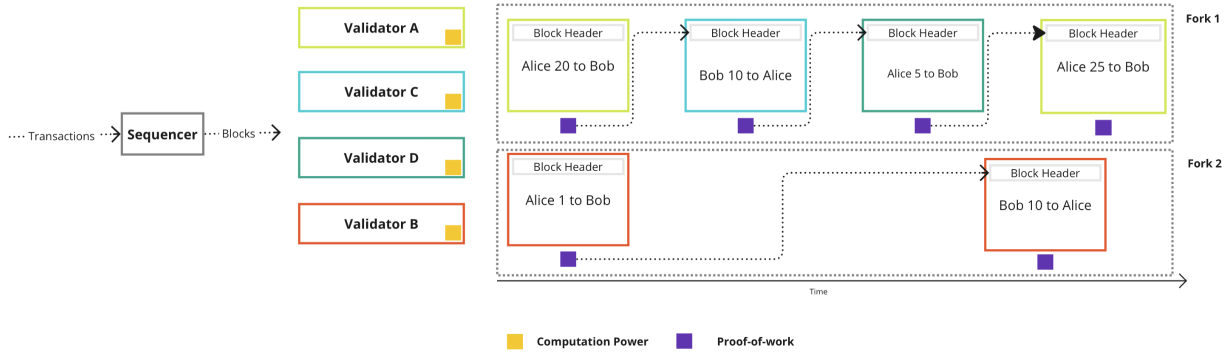


Figure 2.2: An illustration of Proof of Work consensus mechanism

sition function on transactions sequentially. Figure 2.2 demonstrates a PoW consensus framework among three reliable validators (i.e., validators A, C, and D) and one malicious validator (i.e., validator B)[2]. A sequencer initially aggregates and orders user transactions into a block. To process a block (i.e., replaying transactions sequentially within the block), validators are required to solve a computational puzzle that is difficult to solve but easy to validate and must provide evidence of their solution (i.e., proof-of-work). The solving probability depends on their computational power. Additionally, validators ignore blocks that lack a proof, thereby preventing denial-of-service attacks. Each block's proof-of-work depends on its predecessor, preventing easy replication of work by malicious nodes. This creates a unique block header for each block, forming a sequential chain of blocks. The block header is derived from the hash (i.e., a deterministic mapping of input) of the previous block's proof-of-work and transactions. Considering a transaction where 'Alice transfers 20 tokens to Bob', validators A and B concurrently solve the puzzle, but validator B presents a malicious transaction (i.e., Alice transfers 1 token to Bob). Upon these two distinct transactions (i.e., 'Alice transfers 20 token to Bob' and Alice transfers 1 token to Bob) divergent blockchains, or *forks*, emerge. Since honest validators (i.e., A, C, and D) have greater combined computational power they process more blocks than validator B leading to a longer chain of blocks. Therefore, nodes may wait for one fork to become longer to determine the correct world state. However, a set of validators with over 50% computational power can maintain an incorrect state, a vulnerability known as the 51% attack in PoW blockchains like Bitcoin.

In PoS consensus algorithms, validators are chosen for transaction processing based on

[2]For simplification, this illustration assumes each block contains a single transaction, although in practical applications, the number of transactions per block varies and is typically larger.

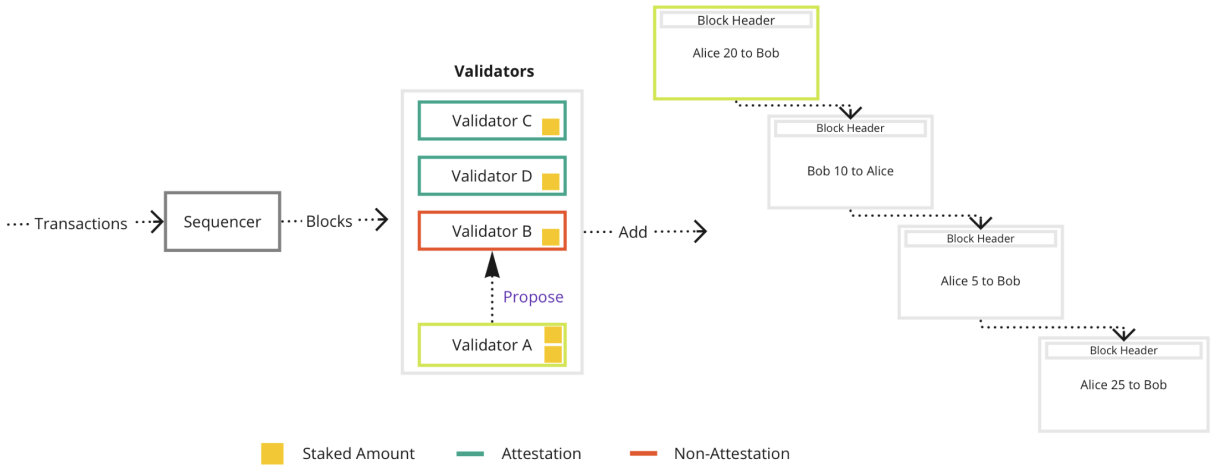


Figure 2.3: An illustration of Proof of Stake consensus mechanism

staking. Validators must lock a certain amount of tokens as their stake. The design of PoS, unlike PoW, does not rely on computational power. Instead, validator selection is based on randomness and the size of the stake. Figure 2.3 shows a set of honest validators (i.e., validators A, C, and D) and one malicious validator (i.e., validator B). Similar to PoW, a sequencer initially aggregates and orders user transactions into a block. The chosen validator, or proposer (i.e., validator A), adds a block to the blockchain after executing the block's transactions. Subsequently, other validators engage by verifying the proposed block. They execute the identical state transition function on the same transactions contained within the block. If their results match the block proposed by the proposer, they attest to it. Validators C and D verify the proposed block, whereas validator B does not. A block is accepted if enough validators attest to it (i.e., accounting for one malicious validator, here four validators suffice to achieve BFT). The proposer receives rewards based on the number of attestations received. The system is designed so that if the proposer acts maliciously, a portion of the proposer's staked amount is forfeited (i.e., slashed). Moreover, other validators (i.e., B, C, D) receive incentives or face penalties based on their attestations.

In a blockchain, the record of transactions (i.e., blocks) should be immutable to always derive the same world state by applying the deterministic state transition function. Otherwise, any malicious party could change the record of transactions for their own profit. For instance, consider *Alice:100* and *Bob:100* as the initial state, and Alice alters the transaction from 'Bob transfers 20 tokens to Alice' to 'Bob transfers 80 tokens to Alice'. Therefore, the world state changes from *Alice:120* and *Bob:80* to *Alice:180* and *Bob:20*. Blockchains maintain immutability by chaining blocks, making any alteration in a block

significantly change the subsequent world state. This is done by summarizing previous blocks in new block headers, typically using cryptographic hash functions. These functions produce a fixed-size byte string from input, with any input change leading to a vastly different hash [3].

2.1.3 Smart Contracts

Smart contracts are pieces of code that execute arbitrary transaction logic. Upon receiving a transaction, smart contracts determine the new world state by executing their pre-defined logic with transaction inputs. Therefore, they can be considered as state transition functions. Ethereum introduced smart contracts so that users could design their own state transition functions to implement their business logic in a decentralized environment. As a result, it enabled the development of dApps independent of central oversight. In other words, instead of pre-defined state transition functions by Ethereum, any dApp developer can program their own business logic as a smart contract with unique inputs and outputs.

To begin with, we provide a brief overview of the interaction of users with smart contracts within the Ethereum blockchain. Users and contracts are each assigned a unique identifier (i.e., a public key) and an internal state upon creation. The state of a user is comparatively simpler, primarily comprising their balance of Ether (i.e., native Ethereum cryptocurrency). Conversely, a contract's state is defined based on its programmed logic. The programmed logic primarily consists of a set of methods callable by users. To call a contract's method, it is necessary for users to specify the contract address and the method they wish to interact with (i.e., call) along with the required transaction fee (i.e., the transaction processing fee, paid in Ether to validators). Furthermore, users should pass the required inputs defined in the contract's method as a transaction payload. Subsequently, the consensus layer is responsible for the execution of the contract's method with the passed inputs to derive the world state. For illustration, consider a more complex transaction involving users Alice and Bob. Bob intends to lend 20 A tokens to Alice. Merely transferring 20 A tokens to Alice does not address the lending challenge, as there is no assurance that Alice will repay the debt. Consequently, a contract can be established between a lender and a borrower, whereby the borrower must deposit a specified quantity of B tokens to borrow 20 A tokens. Additionally, the contract may include conditions such as, if Alice fails to return the 20 A tokens by a predetermined deadline, the collateral is automatically

[3] However, as data volume grows, hash functions become inefficient, leading to the use of Merkle trees [54] for optimized data validation. In Merkle trees, each leaf node is a hash of a data block, and non-leaf nodes are hashes of their child nodes. The Merkle Root at the tree's top represents the entire data set,

transferred to Bob. This represents a lending application facilitating transactions between lenders and borrowers without reliance on mutual trust or intermediaries like banks. In this scenario, the world state encompasses the lending contract state, tracking borrowers and lenders, collaterals, and loan deadlines, among others. The state transition function is defined with custom inputs and outputs to process various loan requests.

2.1.4 Blockchain Data

Blockchain data essentially comprises the world state and the transactions recorded in the ledger. Figure 2.4 illustrates the interaction of user A with the `HelloWorld` contract and, upon processing the transaction, how the blockchain data would change. `HelloWorld` contract comprises two methods: `sayHello()` to view the most recent `greeting` message and `updateGreeting()` to modify the `greeting` message (i.e., contract state). Considering user A invokes the `updateGreeting` method by submitting 'Hello World Again!' (i.e., the new `greeting` message) along with the necessary transaction fee. Following the transaction validation by validators, the `updateGreeting` method is executed.

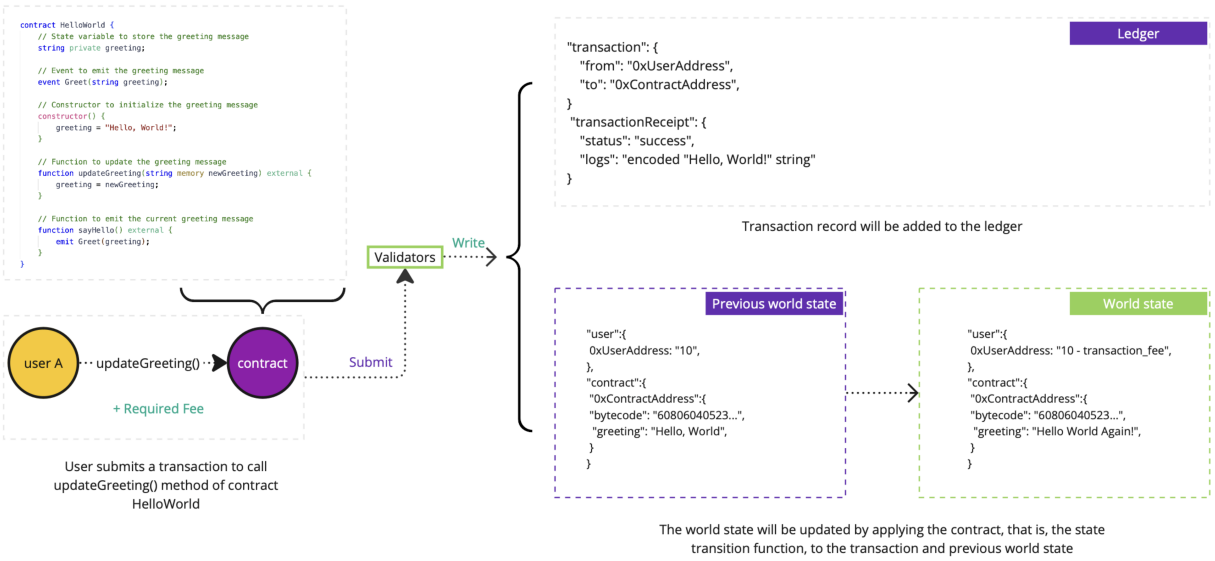


Figure 2.4: A comprehensive illustration of a user interacting with a smart contract, detailing the blockchain data (i.e., the ledger and world state).

where any data alteration changes the Merkle Root.

Subsequent to execution, updates are applied to both the ledger and the world state. The ledger maintains transaction information, encompassing details such as the user A public key as 'from' and the `HelloWorld` contract public key as 'to', and the transaction receipt (i.e., status and log). Regarding the transaction log, if user A executes the `sayHello` method, the log will reflect the encoded string of the latest `greeting` message stored in the contract state. Additionally, both world states (i.e., user and contract state) will be updated. User A paid the transaction fee in Ether, hence the user account balance (i.e., user state) will be updated. The contract's state containing the `greeting` message will be altered to mirror the most recent `greeting` message included in the user A's transaction. Furthermore, Ethereum does not store the raw contract code (i.e., Solidity programs) in the contract state. Instead, it utilizes a blockchain virtual machine (i.e., Ethereum Virtual Machine (EVM)) to derive the bytecode of contracts and store the bytecode.

2.2 Blockchain Data Analysis

Given the vast amount of blockchain data available from various permissionless blockchain platforms, there has been extensive research in the field of blockchain data analytics. The authors of [51], proposed a comprehensive taxonomy for these analytics and categorized them into three categories: 1) Address Identity Inference, 2) Anomaly Detection, and 3) Price Prediction.

The task of address identity inference in blockchain, referring to the process of determining the real-world identity or attributes associated with a particular blockchain address, mainly uses classification techniques. These techniques are split into two types: binary and multi-class classifications. Binary classification techniques aim at identifying potentially illicit user accounts based on their transaction patterns. Multi-class classification, on the other hand, categorizes user addresses into a broader taxonomy. The authors of [92] demonstrated that 42% of Bitcoin addresses belong to exchanges, while 23% are personal wallets. Features used for this classification include network-related features, such as centralities, neighbor identities, and motifs [92, 79, 39, 62], as well as transactional features like volume, smart contracts, and timestamps. Regarding learning algorithms for address identity inference, various methods have been employed. These methods range from tree-based algorithms like Random Forest to more advanced GNN-based approaches [51].

The task of anomaly detection in blockchain refers to the process of identifying and flagging unusual or suspicious activities, transactions, or behaviors within a blockchain network. Given the anomaly detection task, traditional supervised learning algorithms often

fall short due to the class imbalance between legitimate and illicit transactions. This led to the adoption of rule-based or unsupervised learning methods to identify outlier addresses [51]. Various algorithms such as k-means clustering and Gaussian Mixture Models (GMM) have been applied to decompose network features like volume and temporality into factor matrices for identifying anomalous accounts [51]. Many blockchain analytics studies work on preventing Sybil attacks (i.e., the malicious act of a single entity creating multiple fake identities (Sybil nodes) to gain control or manipulate the network's consensus mechanism) through address feature prediction or spotting unusual activities using network analytics approaches.

Price prediction aims to forecast the future values of cryptocurrencies or tokens based on historical data and relevant features. This task is of paramount importance to investors, traders, and stakeholders in the blockchain ecosystem, as it assists in making informed decisions regarding asset management and trading strategies. Price prediction models typically employ various techniques from machine learning and time series analysis, such as regression, deep learning, and statistical methods, to capture complex patterns and trends in cryptocurrency prices.

We have not seen extensive studies on dApp recommender systems through the lens of blockchain data analytics. Therefore, in this study, we aim to provide a detailed analysis of blockchain data to offer recommendations based on interactions (using CF recommender methods) and contract attributes (using Hybrid recommender methods).

2.3 Recommender Systems

Recommender systems can be modeled as a ranking function R [4]. Given a user u_i and a list of interactions $\mathcal{U}$, the ranking function takes as input the list of all items $I = \{i_0, i_1, \ldots, i_n\}$. It produces a ranked list I' as output:

$$R : \mathcal{U} \times I \rightarrow I'$$

Given a user u and a set of previous interactions I_u associated with that user, the recommendation task aims to produce a top-k list of items based on the output of a ranking function R. Numerous methodologies have been developed to refine and sophisticate the nature of this ranking function [9, 32, 33]. Generally, ranking functions R can be categorized into three main types, each providing a unique perspective on the interaction data:

1. **Collaborative Filtering (CF) Recommender Systems:** CF recommender systems exclusively utilize interaction data [3]. The fundamental assumption is that similarities exist in user interactions [30, 16]. These methods are primarily utilized in contexts where interaction data is available, and user or item attributes are less accessible.

2. **Content-Based Recommender Systems:** Content-based recommender systems focus on the attributes of both items and users. Specifically, recommendations are produced by aligning the attributes of users with those of items [47, 55, 7].

3. **Hybrid Recommender Systems:** Hybrid recommender systems integrate the methodologies of both CF and content-based approaches by leveraging user and item interactions as well as attributes to enhance recommendation quality [11, 60]. These systems may be executed in several manners, such as by synthesizing predictions from both methods or by integrating collaborative (interaction-based) and content-based (attribute-oriented) features into a unified model.

One straightforward implementation of CF methods adopts rule-based architectures, such as employing Association Rule Mining, a technique that identifies frequent itemsets in interactions and then generates association rules that describe how these items often appear together. For instance, if customers who buy book A also often buy book B, then the rule $[A \Rightarrow B]$ could be generated. Given a set of items $I = \{i_1, i_2, \ldots, i_n\}$ and a user u who has interacted with a subset of these items $I_u \subseteq I$, the goal is to find a set of rules $R = \{r_1, r_2, \ldots, r_m\}$ such that applying these rules to I_u will yield a set of recommended items $R(u)$. Multiple implementations of rule-based architectures exist within CF methods, each with its own set of advantages and limitations. Early implementations often relied on straightforward rule sets, typically based on if-then heuristics [2]. More recent advancements incorporate machine learning techniques to dynamically update and refine the rule sets based on interactions [52]. Some even integrate time-sensitive rules that adapt recommendations based on seasonal or time-dependent patterns [98].

Transitioning to content-based methods, the primary distinction from CF methods is that content-based methods focus on the attributes of users and items, whereas CF methods emphasize the interactions [67]. For instance, in the context of the Amazon product review dataset [45], each product possesses a multitude of attributes, such as descriptions, tags, and reviews. Content-based methods tend towards leveraging these item attributes to make recommendations. Additionally, hybrid methods provide recommendations based on both interaction similarities and user and item attributes.

Memory-Based Collaborative Filtering: We begin with a brief summary of CF methods: Memory-Based and Model-Based. Memory-Based methods are further classified into Item-Based and User-Based collaborative filtering:

- **Item-Based:** This approach calculates the similarity between items based on user interactions. If two items i and j are often interacted with by the same users, they are considered similar. Mathematically, this can be expressed as:

$$\text{Similarity}(i, j) = \cos(\text{Vec}(i), \text{Vec}(j))$$

  where $\text{Vec}(i)$ and $\text{Vec}(j)$ are vectors representing user interactions with items i and j respectively.

- **User-Based:** Conversely, User-Based methods focus on finding users that are similar to the target user, based on their interactions with items. The mathematical expression for user similarity could be given as:

$$\text{Similarity}(u, v) = \cos(\text{Vec}(u), \text{Vec}(v))$$

  where $\text{Vec}(u)$ and $\text{Vec}(v)$ are vectors encapsulating the items interacted with by users u and v respectively.

The concept of Memory-Based Collaborative Filtering has a rich history in the literature, especially in the domains of e-commerce and content recommendation. Early work by Sarwar et al. [69] provided a comprehensive study on item-based collaborative filtering algorithms, focusing on scalability and effectiveness. Their work set the foundation for utilizing item-item similarity metrics in large-scale e-commerce settings. In addition, user-based approaches have also received substantial attention. The authors of [65] introduced the GroupLens system, one of the first user-based collaborative filtering systems for Usenet news. Furthermore, Herlocker et al. [31] later provided a detailed evaluation framework for user-based methods, examining the impact of different similarity metrics and recommendation algorithms. Some studies have combined both item-based and user-based methods to overcome their individual limitations. For instance, Burke [12] discussed such recommender systems that could integrate both user and item-based approaches to improve recommendation quality.

Model-Based Collaborative Filtering: Model-based collaborative filtering encompasses various approaches, such as neighborhood-based [75], clustering-based [23], and

factorization-based methods [43]. Among these, Matrix Factorization [43] and Graph Neural Networks (GNNs) [88] are particularly prominent in the literature. Matrix Factorization techniques, which include Singular Value Decomposition (SVD) [97] and Alternating Least Squares (ALS) [77], are favored for their simplicity and effectiveness. Model-based CF methods receive an interaction matrix containing user IDs as rows and item IDs as columns. The interaction matrix can be constructed by putting 1 for every cell_{ij} if there is an interaction between user i and item j, and 0 otherwise. They seek to decompose the interaction matrix into latent factors, revealing hidden features that influence interactions and improve recommendation accuracy. We focus on analyzing two methods: Collaborative Filtering Matrix Factorization (CF_{MF}) and Hybrid Matrix Factorization ($\text{Hybrid}_{\text{MF}}$) recommender systems.

2.3.1 Matrix Factorization

CF_{MF} Recommender System: Matrix Factorization involves decomposing an interaction matrix into multiple lower-dimensional matrices, usually termed as the 'user matrix' and 'item matrix'. These lower-dimensional matrices capture latent factors or hidden features that help in making accurate recommendations. Suppose we have an interaction matrix A of dimensions $m \times n$, where m represents the number of users and n represents the number of items. Matrix Factorization aims to approximate A as the product of two lower-dimensional matrices U and V, each of dimensions $m \times k$ and $k \times n$ respectively. Mathematically, this can be expressed as:

$$A \approx U \times V$$

The objective is to minimize the difference between A and $U \times V$, typically measured using the Frobenius norm, given by:

$$\min_{U,V} ||A - U \times V||_F^2$$

Various optimization algorithms like Stochastic Gradient Descent can be used to solve this problem. In the context of CF methods, the ranking function R_{CF} can be expressed as:

$$R_{\text{CF}}(u_i, I) = U_i \times V$$

where U_i is the i^{th} row of the user matrix U, and V is the item matrix [43].

A frequent implementation of CF MF methods is Singular Value Decomposition (SVD) [43]. In SVD, the original interaction matrix is decomposed into three other matrices: U, Σ, and V^T, where U and V^T are orthogonal matrices and Σ is a diagonal matrix containing

singular values. This decomposition helps to identify latent factors effectively and serves to fill in the missing values for interactions, thereby facilitating the recommendation process. Given an $m \times n$ matrix A representing interactions, SVD decomposes A into three matrices U, Σ, and V^T such that:

$$A = U \times \Sigma \times V^T$$

Here, U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix containing the singular values in descending order, and V^T is an $n \times n$ orthogonal matrix. The idea is to approximate A by considering only the top k singular values in Σ, and corresponding vectors in U and V^T. This approximation is given by:

$$A_k = U_k \times \Sigma_k \times (V_k)^T$$

where U_k, Σ_k, and V_k contain only the first k columns of U, Σ, and V respectively. In CF MF methods, the SVD-based ranking function R_{SVD} can be defined as:

$$R_{\text{SVD}}(u_i, I) = (U_k)_i \times \Sigma_k \times V_k$$

where $(U_k)_i$ is the i^{th} row of the matrix U_k [68].

Hybrid$_{\text{MF}}$ Recommender System: Hybrid Recommender Systems aim to combine the strengths of different recommendation strategies to enhance overall recommendation performance. In the context of Matrix Factorization (MF), a Hybrid approach often involves integrating user and item attributes into the collaborative filtering framework. This integration allows the system to not only exploit the interaction patterns in the interactions matrix but also leverage side information such as user and item attributes.

The Hybrid Matrix Factorization model can be formalized by extending the original factorization concept. Let C_u and C_i denote the content-based embeddings for users and items respectively. The hybrid model then seeks to factorize the interaction matrix A while simultaneously considering these content attributes. The factorization can be represented as:

$$A \approx (U + C_u) \times (V + C_i)^T$$

where U and V are the user and item latent feature matrices as defined previously, and C_u and C_i are matrices constructed from the content attributes. The matrices C_u and C_i are usually of dimensions $m \times d$ and $n \times d$ respectively, where d is the number of content attributes.

The optimization objective in a hybrid MF model is to minimize the error in predicting the interactions while also minimizing the error in representing the content attributes. This can be expressed as:

$$\min_{U,V} ||A - (U + C_u) \times (V + C_i)^T||_F^2 + \lambda(||U||_F^2 + ||V||_F^2)$$

where λ is a regularization parameter that controls the extent of regularization to prevent overfitting.

The recommendation function R_{Hybrid} in a Hybrid MF system can be formulated as:

$$R_{\text{Hybrid}}(u_i, I) = (U_i + C_{u_i}) \times (V + C_i)^T$$

where U_i is the i^{th} row of the user attribute matrix and C_{u_i} is the content attribute vector for the i^{th} user.

2.3.2 Graph Neural Network

Recently, GNN methods have gained attention for different data mining tasks, particularly when interaction datasets naturally form graph structures (e.g. bipartite interaction graphs). Social media platforms offer classic examples, where interactions between users and items create such graphs. In recent years, GNNs have emerged as the go-to architecture for handling graph-structured data, including in the domain of recommender systems. These networks leverage the topological properties of graphs to capture higher-order interactions between nodes (i.e., users and items). This enhances the quality of recommendations in the bipartite interaction graph. We first formulate a graph and then discuss GNN network architectures to deliver downstream recommender systems.

A graph G can be formally defined as:

$$G = (V, E)$$

where V represents the set of vertices (or nodes) and E represents the set of edges connecting these vertices. The neighborhood of a node v_i can be formally defined as a set $N(v_i)$ containing all nodes that are directly connected to v_i by an edge:

$$N(v_i) = \{v_j : (v_i, v_j) \in E\} \quad \text{or} \quad N(v_i) = \{v_j : (v_j, v_i) \in E\}$$

The fundamental mechanism of GNNs operates on two central components: Aggregation and Update. The aggregation process accumulates the embeddings of a node's immediate neighborhood to generate an aggregated embedding a_v:

$$a_v = \text{AGGREGATE}(\{x_u : u \in N(v)\})$$

Several types of aggregation functions are commonly used in GNNs [1] such as:

- **Mean Aggregator**: Takes the mean of neighbor embeddings.

$$a_v = \frac{1}{|N(v)|} \sum_{u \in N(v)} x_u$$

- **Sum Aggregator**: Sums up all neighbor embeddings.

$$a_v = \sum_{u \in N(v)} x_u$$

- **Max Aggregator**: Takes the maximum value among neighbor embeddings.

$$a_v = \max_{u \in N(v)} x_u$$

After the aggregation phase, the aggregated embedding a_v is used to update the embedding x_v of the node v using an update function UPDATE:

$$x'_v = \text{UPDATE}(x_v, a_v)$$

The aggregation process is not limited to a single layer or immediate neighbors. A hierarchical or multi-layer approach is often applied to incorporate embeddings from nodes that are further away.

CF_{GNN} Recommender System: A CF_{GNN} recommender system can be formulated as a link prediction problem [37]:

$$\text{Predict}(u, i) = \text{DECODER}(GNN(G, x_u), GNN(G, x_i))$$

Here, DECODER is a function that uses the embeddings from the GNN architecture to predict the likelihood of a link (i.e., interaction) between user u and item i. Many architectures have become popular in the field of GNNs. Each of these architectures has its own advantages and disadvantages. Their underlying mechanisms are suited for different needs and scenarios. Below, a comparison of two popular GNN architectures, namely Graph Convolutional Network and Graph Sample and Aggregation, is provided.

- **Graph Convolutional Networks (GCNs)** [90]: Suited for homogeneous graphs; assumes that closer nodes are more similar. Aggregation function:

$$a_v = \text{ReLU}\left(\sum_{u \in N(v)} W x_u\right)$$

  where ReLU (Rectified Linear Unit) is defined as: $\text{ReLU}(x) = \max(0, x)$

- **Graph Sample and Aggregation (GraphSAGE)** [26]: Allows for inductive learning and can generalize to unseen nodes. Aggregation function:

$$a_v = \text{CONCAT}\left(x_v, \text{MEAN}\left(\{x_u : u \in N(v)\}\right)\right)$$

  where CONCAT refers to the concatenation of vectors.

To complement the GNN layer for a binary link prediction task, a binary classifier component (i.e., DECODER) is essential. This component takes the node embeddings from the GNN and predicts whether a link should exist. One common approach is to use a sigmoid function, which outputs a probability score representing the likelihood of a link between two nodes. The general form of the classifier component can be expressed as:

$$\text{Binary Classifier}(z_u, z_i) = \sigma\left(W_c[z_u \| z_i] + b_c\right)$$

Here, z_u and z_i are the learned embeddings for user u and item i respectively, σ denotes the sigmoid activation function, W_c represents the learnable weight matrix, b_c is the bias term, and $\|$ denotes the concatenation of the embeddings. The sigmoid function maps the result into a range between 0 and 1, which can be interpreted as the probability of the existence of a link.

Incorporating this classifier with the GNN layer, CF_{GNN} method can effectively learn to predict links that represent interactions, which is the fundamental goal in CF recommender systems. This binary classification task is critical as it directly correlates with the recommendation system's performance in suggesting relevant items to users.

$\text{Hybrid}_{\text{GNN}}$ Recommender System: To build on the concept of CF_{GNN} recommender systems, $\text{Hybrid}_{\text{GNN}}$ methods aim to integrate additional sources of information to enhance the recommendation power. These Hybrid approaches leverage both content-based and CF techniques, utilizing node and edge attributes alongside the graph structure (i.e., bi-partite graph structure of interactions). The GNN serves as the backbone, processing the interactions, while additional layers incorporate the additional information (i.e., user and item attributes).

One common method is to integrate user and item attributes into the node embeddings. This is achieved by concatenating or combining the GNN-generated embeddings with attributes (i.e., semantic content) from users or items. The enriched embeddings capture both the graph's topological information and the semantic content of the nodes, leading to more accurate recommendations.

The overall architecture of a $\text{Hybrid}_{\text{GNN}}$ recommender system could be formalized as follows:

$$\begin{aligned}\text{HybridPredict}(u,i) = \text{HybridDECODER}\Big(&\text{COMBINE}\big(GNN(G,x_u),F_u\big),\\ &\text{COMBINE}\big(GNN(G,x_i),F_i\big)\Big)\end{aligned} \tag{2.1}$$

Here, F_u and F_i represent embeddings containing the attributes of user u and item i, respectively. The COMBINE function merges these embeddings with the GNN embeddings (i.e., constructed solely based on interactions), and the HybridDECODER function then uses the combined embeddings to predict the likelihood of interaction. Considering CF_{GNN} methods, the same classifier (i.e., HybridDECODER) as described previously can be utilized as the final layer to predict binary links (i.e., whether they exist or not).

2.4 Blockchain Recommender Systems

Research on recommender systems using only blockchain data (i.e., blockchain recommender systems) is limited. Some studies consider Web2 data to provide recommendations to Web2 users using blockchain as a ledger to store part of the recommendation data instead of centralized databases [34]. Since they do not consider blockchain data, they are orthogonal to this study, where we focus on blockchain recommender systems. To the best of our knowledge, there is only one study on blockchain recommender systems [94]. This study focuses on a specific type of blockchain asset (NFTs) and leverages interactions with these NFTs to provide new NFT recommendations. However, it does not consider recommending smart contracts. Therefore, our contribution is more general in that it considers any interactions with contracts on a blockchain to provide further contract recommendations.

Chapter 3

Methodology

In this chapter, we outline the steps to create a blockchain interactions dataset focused on Ethereum (Section 3.1). We then examine this dataset through Exploratory Data Analysis (EDA) (Section 3.2). Following this, we introduce a blockchain data pipeline for retrieving and processing data for recommender systems, as well as the architectures of these recommender systems applied to the blockchain recommendation task (Section 3.3).

3.1 Dataset Creation

As of the time of writing this study, there are over 40 permissionless blockchains, each with unique characteristics and capabilities. Among these, Ethereum stands out as a particularly compelling network for the exploration of blockchain recommender systems for several key reasons:

- **Transaction History:** Ethereum is the second most popular blockchain network, second only to Bitcoin. Its widespread adoption has led to a long and detailed history of user transactions, leading to more interactions for each user. Thus, with increased user interactions, both Collaborative Filtering (CF) and Hybrid blockchain recommender systems can bring higher recommendation power.
- **Smart Contract Capabilities:** Ethereum is different from other blockchains such as Bitcoin by supporting the execution of contract code. This capability enables the development of dApps. Thus, this research explores the feasibility of blockchain recommender systems by considering contracts as items.

- **Data Availability:** Ethereum's well-documented APIs and readily available chain-scans facilitate the process of constructing a transactions (i.e., interactions) dataset.

Several methods are available for creating the dataset required for this study. One approach is to operate an Ethereum node and synchronize it with the world state. However, this approach requires substantial resources. An alternative is to utilize libraries such as `geth` and `web3.js`, which facilitate data synchronization and retrieval by connecting to publicly hosted nodes that contain the Ethereum ledger, such as Infura[1]. Nevertheless, these approaches still require additional resources. Chain-scans are entities responsible for reading the blockchain data and, by indexing them, providing better accessibility to blockchain data for everyone. Consequently, Etherscan[2] was employed as a popular chain-scan for the Ethereum network to obtain interaction data.

To test the feasibility of blockchain recommender systems, as in other recommender system domains, interaction data is needed. Interactions stored on the Ethereum ledger can be considered. However, due to the vast number of these interactions, it may be computationally intensive to run downstream recommender systems on such a massive interactions dataset. A moving window can be adopted to consider all blockchain data in a time frame (e.g., the past 2 months). Considering nearly $1M$ transactions each day on Ethereum, this yields around $60M$ transactions. Although this amount of data is larger compared to other recommender systems datasets, such as MovieLens-100k and MovieLens-$1M$, it may yield very sparse interactions. This is mainly because users may interact with contracts over a wider time frame. Therefore, one approach can be to consider a set of randomly selected users and fetch all their past transactions. We selected a random day, December 18, 2022, and, out of the 917,000 user addresses who called at least one contract that day, we randomly selected 42,000 user addresses. Then, we downloaded these users' most recent transactions utilizing the Etherscan API, up to the API limit of 50. The final 400,000 transactions are with 31,527 unique contracts, with an average of ten contracts per user.

The transactions in the dataset have three main attributes:

- **From and To Addresses:** These indicate the origin (i.e., source) of a transaction and its destination. Often, the *from* attribute is a user address and the *to* attribute is a contract address.

[1] https://www.infura.io
[2] https://etherscan.io/

- **Contract Code:** Ethereum's smart contracts are written in Solidity programming language. The contract's Solidity code is retrieved from Etherscan's endpoints. The *solidity-parser*[3] library is utilized to prepare the contract data for the Hybrid blockchain recommender system.

- **Payload:** Upon transaction submission by users, specifying which method of the destination contract they wish to call, the required arguments are included as inputs, as defined by dApp developers within the contract methods' signatures (i.e., contract's method definition in Solidity code)

We obtain the raw Solidity code for each contract address using Etherscan's public API [4]. Hybrid blockchain recommender systems require a contract's code to find similarities among different contracts. Moreover, the Solidity code in smart contracts can be challenging to use directly by Hybrid recommender systems due to its complex structure and special syntax. Preprocessing is required to make the data useful. Solidity has a special comment structure that enables the easy separation of code and comments. We employ the `Solidity-parser` Python package to examine various components of the contract, such as classes, functions, and comments (see the contract parser module in Figure 3.3).

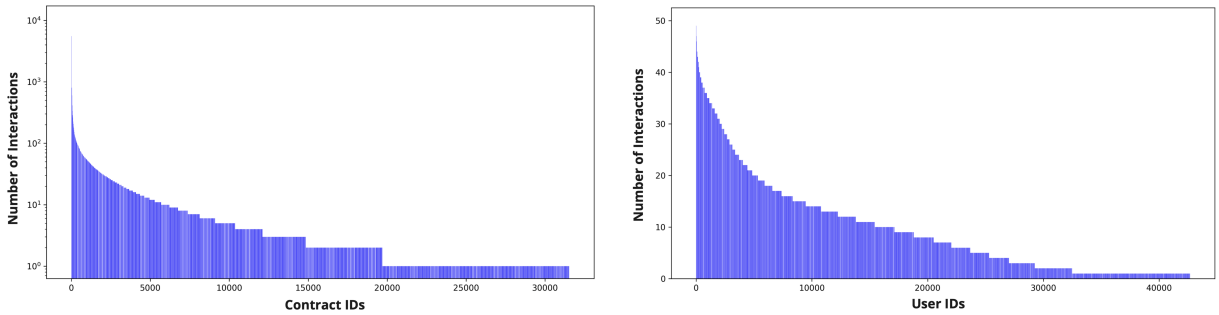
3.2 Dataset Exploration

To better understand the interactions dataset, we conduct an Exploratory Data Analysis (EDA). This analysis offers insights such as the distribution of transactions (that is, interactions) across contracts and users. As shown in Figure 3.1, the distribution of the number of interactions per unique user reveals that most users have engaged in only a limited number of interactions. This characteristic indicates that interactions in the blockchain context are sparse, similar to those observed in other domains of recommender systems. The sparsity of interactions limits the effectiveness of CF methods that depend exclusively on interactions to make recommendations. Consequently, the significance of Hybrid recommender systems intensifies. Such methods can utilize additional information from contracts, leading to enhanced recommendation capabilities by capturing similarities

[3]https://www.github.com/consensys/python-solidity-parser/

[4]Ethereum runs the bytecode of smart contracts, not the Solidity source code, which is readily interpretable by humans. Thus, for the source code to be public, the contract developer must upload it to a chain-scan platform like Etherscan. When uploaded, Etherscan compiles the source code and checks the generated bytecode against the contract's bytecode stored on the ledger. Upon matching, the source code is marked as *verified*.

within contracts. Figure 3.1 also shows the distribution of interactions associated with each contract, with some contracts—known as popular contracts—having a higher number of interactions. This pattern is well-documented in other domains of recommender systems. Based on this observation, we provide a baseline popular contract recommender—namely, it produces popular contracts as recommendations—to compare with the effectiveness of blockchain recommender systems in Chapter 4.



(a) Distribution of Contract IDs (Log Scale) (b) Distribution of User IDs (Normal Scale)

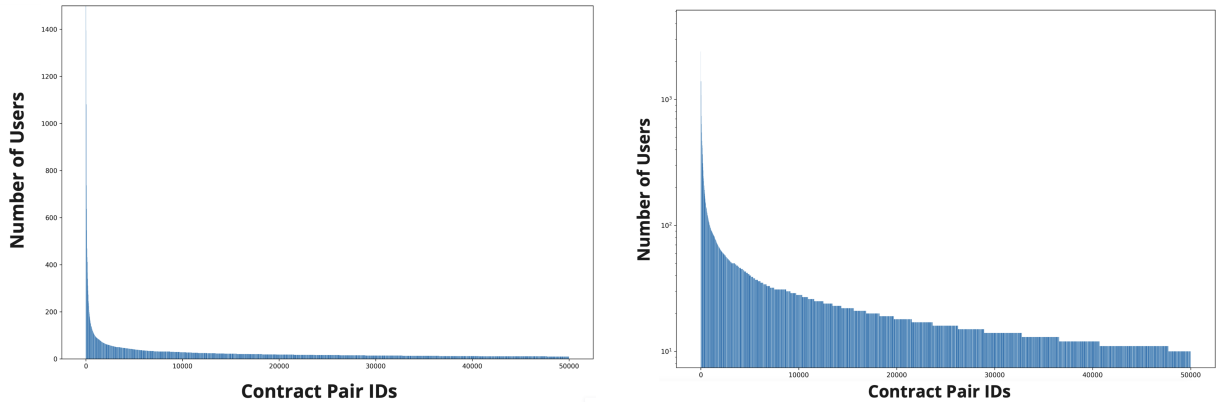
Figure 3.1: Distribution of user and contract IDs over interactions dataset.

Prior to conducting CF and Hybrid recommender systems analyses, a basic recommender system may be implemented as a rule-based method. Analogous to CF methods, these encapsulate rules in interactions. Therefore, to examine whether blockchain data contains insightful information for recommendations, initially, the identification of contract pairs with the highest occurrence in the interaction data is conducted. Subsequently, following interaction with the first contract in these pairs, the second contract is recommended. Let C be the set of all contracts, i.e., $C = \{c_1, c_2, \ldots, c_n\}$. For any two contracts c_i and c_j in C, the co-occurrence count is calculated, defined as the number of user addresses that have interacted with both c_i and c_j.

$$\text{Co-occurrence}(c_i, c_j) = |\{u \in U : (u, c_i) \in T \wedge (u, c_j) \in T\}| \tag{3.1}$$

where U is the set of user addresses and T represents the set of all interactions.

Figure 3.2 demonstrates that some pairs of contracts tend to be more popular among users (i.e., contracts with smaller IDs), while others may be less popular (i.e., contracts with higher IDs). This finding suggests that interaction data in the context of blockchain do in fact convey valuable information to base recommendations on. In the next section, we discuss more advanced recommender methods concerning the extent to which blockchain-based recommendations can be enhanced.



(a) Distribution of Contract Pair IDs(Normal Scale) (b) Distribution of Contract Pair IDs(Log Scale)

Figure 3.2: Distribution of the top 50,000 most frequently interacting pairs of contracts. The x-axis represents each unique pair of contract IDs, and the y-axis represents the number of unique users who interacted with both contracts within a contract pair ID.

3.3 Recommender Methods

This study considers two recommender methods (i.e., CF and Hybrid recommender systems) each with two architectures (i.e., MF and GNN) named as CF_{MF}, $\text{Hybrid}_{\text{MF}}$, CF_{GNN}, and $\text{Hybrid}_{\text{GNN}}$. Additionally, to hybridize both CF_{MF} and CF_{GNN}, we utilize clustering and embedding approaches, leading to six recommender methods: CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, CF_{GNN}, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$. Figure 3.3 depicts a blockchain data pipeline. It connects to various chain-scans, fetches blockchain data, and performs unique processing steps (i.e., Graph Inserter, Contract Embedder, Matrix Creator, and Clustering) to provide compatible data to each recommender method.

The MF blockchain recommender systems (i.e., CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, and $\text{Hybrid}_{\text{MF-SBERT}}$) require an adjacency matrix of interactions to provide recommendations. Therefore, the *Matrix Creator* module creates the adjacency matrix as illustrated in Figure 3.3. The adjacency matrix, also known as the interaction matrix, can be input into the MF methods to produce the top-k contract recommendations. The CF_{MF} method requires only the interaction matrix to generate recommendations. However, $\text{Hybrid}_{\text{MF-CLUSTERING}}$ and $\text{Hybrid}_{\text{MF-SBERT}}$ require additional information (i.e., contract cluster IDs and contract embeddings respectively). This additional information (i.e., contract features) conveys the notion of similarity among contracts to aid in the functionality of the MF methods. This process is depicted by the *clustering* and *contract embedder* module in Figure 3.3. The *clus-*

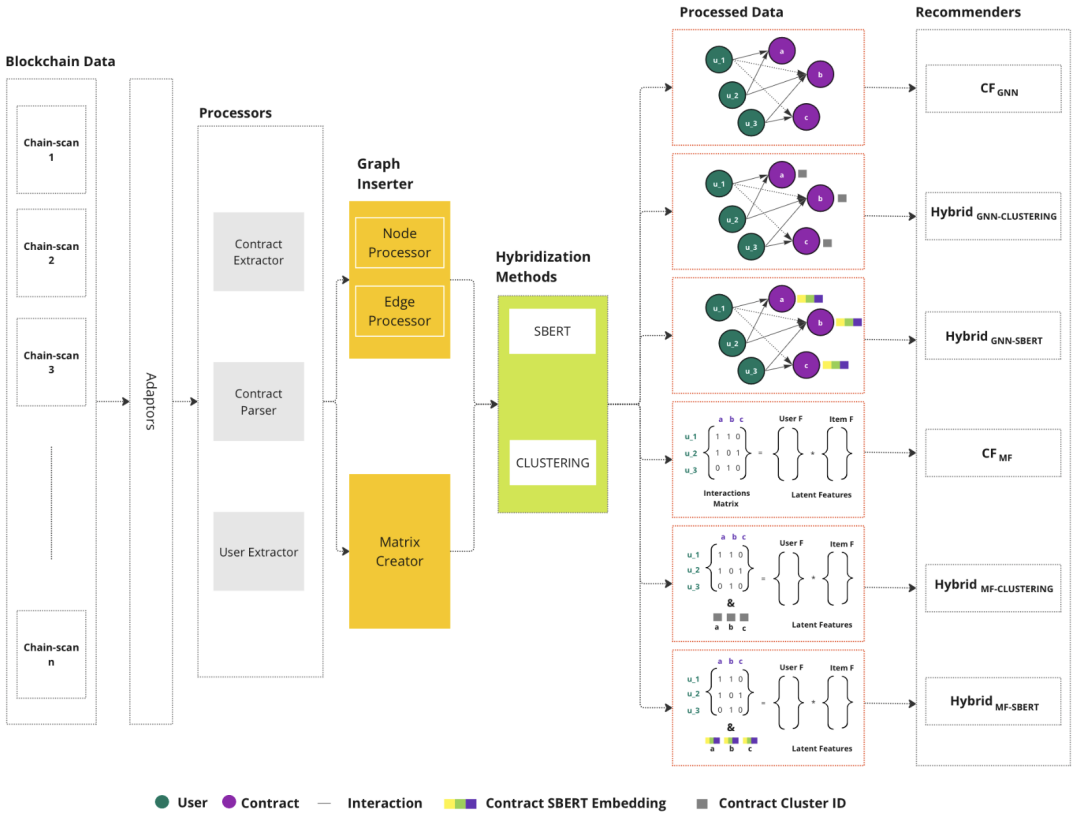


Figure 3.3: Transaction Processors Pipeline. The output of the pipeline will be utilized by blockchain recommender systems.

tering module assigns a cluster ID to each contract as the output of the clustering method performed over the corpus of contract comments. Moreover, the *contract embedder* module derives the necessary contract embeddings for the $\text{Hybrid}_{\text{GNN-SBERT}}$ method.

Conversely, GNN recommender systems (i.e., CF_{GNN}, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$) require the interaction graph (i.e., users and contracts as nodes, and interactions as edges). Therefore, additional modules are required to create such a graph, as shown by the *Graph Inserter* module in Figure 3.3. The CF_{GNN} method requires only the interaction graph to provide recommendations. However, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$ and $\text{Hybrid}_{\text{GNN-SBERT}}$ require contract cluster IDs and contract embeddings respectively to encapsulate similarities within contracts and offer enhanced recommendation accuracy.

In this study, we develop a blockchain data pipeline and adopt recommendation meth-

ods using Python version 3.9. To implement the Matrix Factorization (MF) methods, the *LightFM* library is utilized [44]. Additionally, to implement and train Graph Neural Network (GNN)-based recommender methods, the PyTorch version 1.13 [5] library is utilized.

3.3.1 Contract Embedding

In this study, we incorporate contract similarities as additional features to enhance the pure CF recommender methods. To identify similar contracts, we consider two methods: 1) clustering and 2) embedding of similar contracts. Figure 3.4 demonstrates lending contract comments (i.e., Aave lending protocol) and staking contract comments (i.e., Lido staking protocol). A staking contract allows users to lock up their cryptocurrency tokens to participate in network operations and earn rewards. Therefore, we consider these contract comments for both clustering and embedding methods.

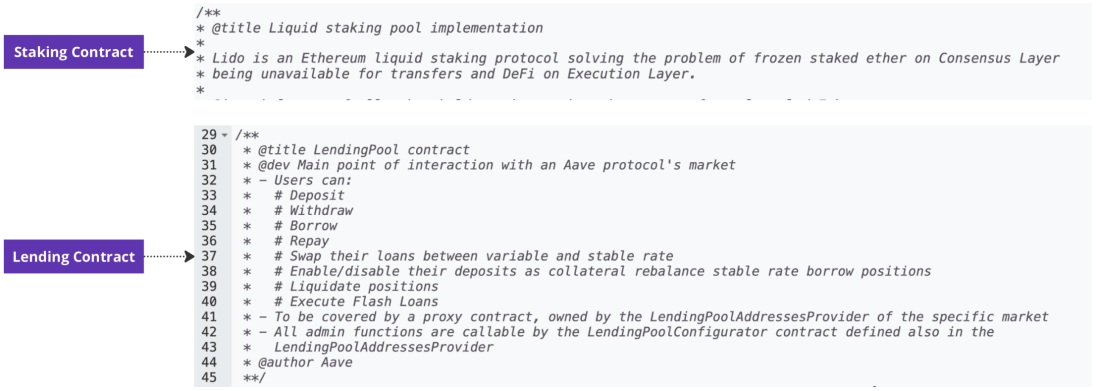


Figure 3.4: Lending and staking contract comments.

Considering the clustering method, various clustering approaches exist, but one notably efficient and effective method is Latent Dirichlet Allocation (LDA). LDA can be formulated as a generative probabilistic model where each document (contract comments) is represented as a mixture of a small number of topics, with the generation of each word being attributable to one of the document's topics.

[5] https://pytorch.org/docs/stable/torch.html

$$\text{LDA}(D, K) = \sum_{i=1}^{D} \sum_{j=1}^{K} P(Topic_j|Document_i) \times P(Word|Topic_j)$$

Here, D denotes the set of contracts, K denotes the number of topics (i.e, clusters), $P(Topic_j|Document_i)$ is the probability of $Topic_j$ occurring in $Document_i$, and $P(Word|Topic_j)$ is the probability of a word occurring given $Topic_j$. Fundamentally, LDA enables us to map each contract to a particular cluster. To run the LDA model effectively on contract codes, it is necessary to preprocess them first. Given that LDA is most effective with textual data, the process begins by separating the comments from the actual code. The Solidity-parser assists in this process, extracting different classes and functions from each contract. Subsequently, the extracted developers' comments linked to these parts are provided to the LDA model. Selecting an appropriate number of topics, k, is imperative for effective clustering. To determine the best k, the Elbow Method is used. The Elbow Method looks at how adding more topics affects the model's performance. The search is conducted for the point where increasing topics doesn't make much difference anymore—this point is the "elbow." Upon considering this method, the conclusion was reached that $k = 15$.

Transitioning to the contract embedding method, we utilize Sentence-BERT [63] (i.e., SBERT) embedding approach to encode sentences (i.e., contracts) into embedding vectors in a manner that places contracts with similar meanings close together in the embedding space. This is accomplished through pre-training on a substantial text corpus and leveraging a transformer architecture [81], which enables it to understand the context and semantic meaning of sentences.

3.3.2 Matrix Factorization Recommender Systems

Consider $T = [t_1, t_2, \ldots, t_n]$ to denote the set of all transactions, $U = [u_1, u_2, \ldots, u_m]$ be the set of user addresses, and $C = [c_1, c_2, \ldots, c_t]$ to denote the set of contract addresses. The objective is to derive a structured representation that can capture the relationships between these sets. One approach to accomplish this is by constructing an adjacency matrix $\mathbf{A}$ of dimensions $m \times t$, where each cell A_{ij} signifies the existence of at least one transaction between user u_i and contract c_j.

The goal of MF recommender methods is to derive user and item latent feature matrices from the interaction matrix. As previously discussed, Singular Value Decomposition (SVD) is employed for both CF_{MF} and $\text{Hybrid}_{\text{MF}}$ recommender methods. Given a user

u_i, and contract c_j, the multiplication of the corresponding values in user and contract latent feature matrices yields the relevance score of contract c_j to the user u_i. Given the calculation of this score for all pairs of users and contracts, we sort to derive the top-k contracts with the highest relevance score for each user. This study utilizes the LightFM [44] Python library, a well-regarded and efficient implementation of SVD for MF.

Transitioning to $\text{Hybrid}_{\text{MF-SBERT}}$, similar to $\text{Hybrid}_{\text{MF-CLUSTERING}}$, we utilize the LightFM implementation of MF recommender methods by passing the SBERT embeddings of contracts to the model. The remainder of the process is identical to the $\text{Hybrid}_{\text{MF-CLUSTERING}}$ method.

3.3.3 Graph Neural Network Recommender Systems

Matrix Factorization (MF)-based blockchain recommender systems, which include both CF and Hybrid methods, do not adequately capture higher-order interactions between users and contracts. Graph Neural Networks (GNNs) have recently demonstrated significant capabilities in capturing higher-order relationships in interactions data. Utilizing GNN methods results in enhanced recommendation accuracy for blockchain recommender systems.

Considering the vast number of daily transactions on the Ethereum blockchain, it is crucial to have a robust methodology for constructing the transaction graph. Therefore, we establish a modular transaction graph processing pipeline capable of integrating with any chain-scans to provide transactions to the blockchain recommender systems. For this study, we integrate the pipeline with the Ethereum blockchain, focusing exclusively on a selected group of user addresses. Our comprehensive pipeline is composed of two key components:

- **Graph Inserter:** Has two main processors: 1) Node processor and 2) Edge processor. The Node processor manages the preliminary processing of nodes, preparing them for insertion into the graph. The Edge processor establishes vertices between each user and contract nodes appear within an interaction.
- **Contract Embedder:** This component receives contract nodes from the Graph Inserter module and computes embeddings from contract comments.

The Node Processor's task begins by examining a transaction t, distinguishing between user and contract addresses, and thereby categorizing nodes into two types: users and

contracts. It then employs the contract-parser to append the parsed contract code to each contract node, utilizing the Etherscan contract API endpoint. Consequently, a collection of nodes, including contracts and user addresses, is forwarded to the Edge Processor. The Edge Processor processes batches of transactions t along with an array of predefined nodes. It examines the "from" and "to" attributes in a transaction t, establishing an edge between the corresponding nodes.

CF_{GNN} is capable of providing recommendations exclusively by considering the transaction graph. However, when transitioning to the $\text{Hybrid}_{\text{GNN-CLUSTERING}}$ and $\text{Hybrid}_{\text{GNN-SBERT}}$ methods, the latter requires additional contract information. This supplementary information enables the Hybrid method to account for the notion of similarity among contracts and to offer better recommendations. One effective strategy for this is the construction of contract embeddings that map similar contracts closely together in a numerical space. Thus, in the *contract embedder* module (illustrated in Figure 3.3), *SBERT* and *LDA* Clustering methods are employed to generate the contract embeddings and contract cluster IDs required for the $\text{Hybrid}_{\text{GNN-SBERT}}$ and $\text{Hybrid}_{\text{GNN-CLUSTERING}}$ methods respectively.

GNN recommender systems may be conceptualized as binary link prediction tasks [20]. Specifically, the edges connecting user and contract nodes in the transaction graph can be predicted via a binary classification approach. Figure 3.5 illustrates the construction of CF_{GNN} and $\text{Hybrid}_{\text{GNN}}$ (i.e., $\text{Hybrid}_{\text{GNN-CLUSTERING}}$ and $\text{Hybrid}_{\text{GNN-SBERT}}$) with two main layers: 1) Blockchain data layer, and 2) Recommendation layer. The Blockchain Data Layer fundamentally constitutes a series of steps essential for generating the transaction graph as demonstrated in Figure 3.3. The recommendation layer is principally derived from the Deep GraphSage recommender design [20], utilizing the GraphSage GNN architecture to learn user and contract embeddings, given the premise that nodes representing users and contracts with an edge (i.e., interaction) between them should be close to each other in the numerical space.

The CF_{GNN} recommender method solely requires a transaction graph to generate recommendations. Conversely, $\text{HYBRID}_{\text{GNN}}$ methods additionally necessitate contract features (i.e., contract embedding or contract cluster ID) to consider contract similarity as well as interaction data. Hence, the architectures of both CF_{GNN} and $\text{HYBRID}_{\text{GNN}}$ recommender methods encompass two primary components: 1) two GraphSage layers, and 2) a classifier layer. $\text{HYBRID}_{\text{GNN}}$ methods additionally incorporate contract features. The *GraphSage layers* facilitate the integration of higher-order features by aggregating information from adjacent nodes, employing GraphSAGE [26]. The *classifier*, using the acquired embeddings, executes a dot-product operation between source and destination node embeddings to generate edge-level predictions, i.e., the likelihood of interactions existing between user and contract nodes. The sole distinction between CF_{GNN} and $\text{HYBRID}_{\text{GNN}}$ methods

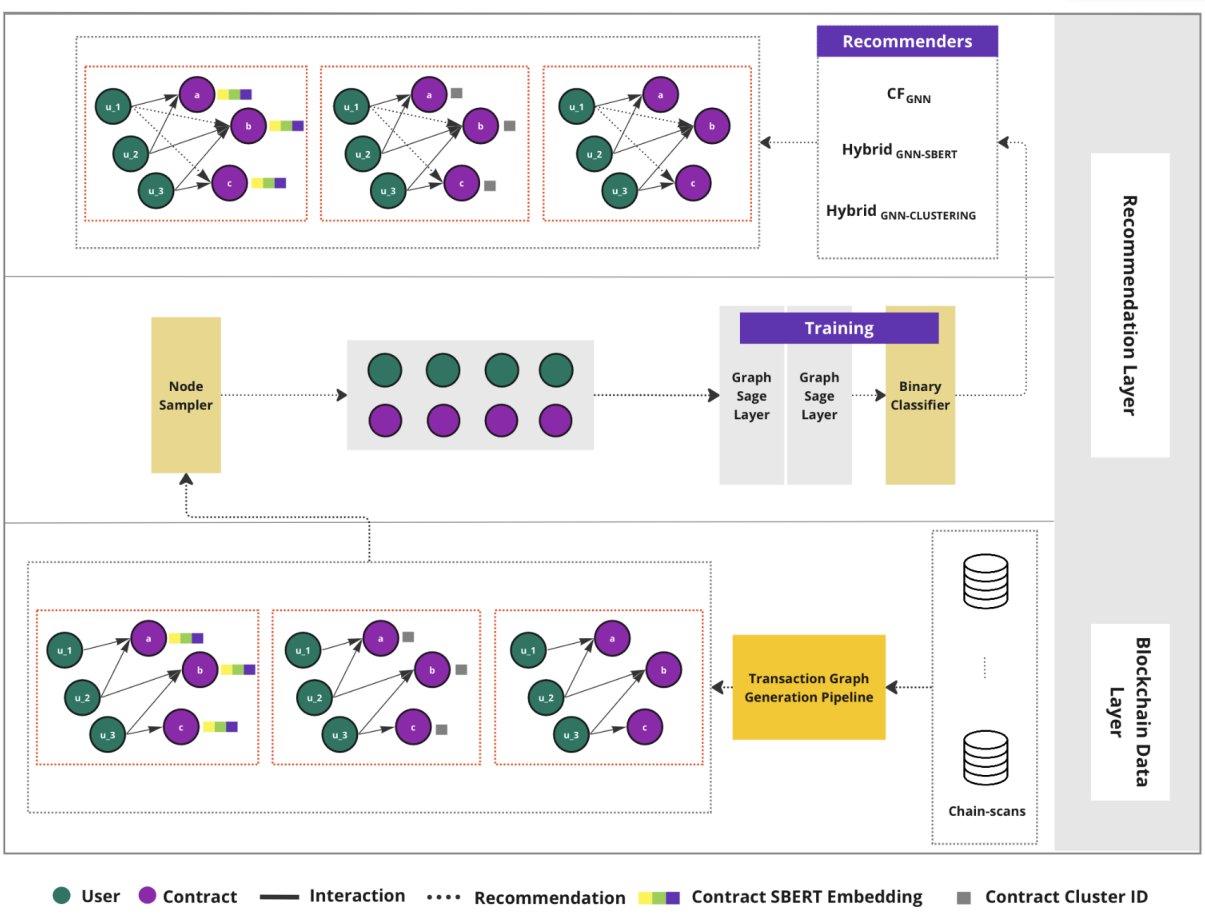


Figure 3.5: CF_{GNN} and $Hybrid_{GNN}$ blockchain recommender systems

is that the latter accounts for additional contract features, thus learning from both the similarity of contracts and interactions in the GraphSage layers. The node sampler module samples positive and negative nodes; positive nodes are pairs of nodes with an actual edge (i.e., interaction) between them, while negative nodes are randomly selected pairs of nodes where there is no interaction. This sampling strategy assists the model in effectively learning user and contract embeddings by bringing positive nodes closer together and distancing negative nodes. It should be noted that this sampling is different from the sampling strategy utilized during test time (Subsection 4.1.2).

Chapter 4

Evaluation

In this chapter, we explore the feasibility of providing recommendations based on blockchain data. We first elaborate on the evaluation metrics utilized, such as Hit Rate at k, Mean Average Precision at k, and Normalized Discounted Cumulative Gain at k (Subsection 4.1.1). Then, we evaluate the effectiveness of different blockchain recommender systems, including CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, CF_{GNN}, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$ (Subsection 4.1.2). Finally, we provide a comparison of the efficiency of these recommender systems in terms of inference latency and memory usage (Subsection 4.1.3).

Our experiments were run on a Linux-based server operating Ubuntu 21.10, equipped with 24 CPU cores and an NVIDIA 2080 Ti GPU. While the GPU notably enhances the training phases of GNN recommender methods, it is important to highlight that the entire data processing pipeline is designed to function independently of GPU resources.

4.1 Recommendation Performance

This section presents the experimental results on the performance of the Popular Contract Recommender (POP), CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, CF_{GNN}, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$. The POP recommender ranks contracts by their popularity, defined by their frequency in user-contract interactions. Users often exhibit satisfaction with popular recommendations [99], which is why the POP recommender is used as a baseline.

4.1.1 Evaluation Metrics

To evaluate the performance of different methods, we use the following metrics which are common metrics in the context of recommendation systems:

Hit at k (HR@k): The HR@K evaluates the ability of the model to recommend a relevant item within the top k positions of a ranked list. Mathematically, it is defined as:

$$\text{HIT@}k = \frac{1}{|U|} \sum_{u=1}^{|U|} \mathbb{1}\left(\bigcup_{i=1}^{K} \{\text{rel}_{ui} > 0\} \right)$$

where U is the set of user addresses (i.e., public keys), $\mathbb{1}(\cdot)$ is the indicator function which equals 1 if the user u has at least one relevant contract in the top-k contracts and 0 otherwise, and rel_{ui} indicates the relevance of contract i to user u.

Mean Average Precision at k (MAP@k): MAP@k assesses the average precision of the model across all users, up to the position K in the ranked list. Mathematically, MAP@k is defined as:

$$\text{MAP@}k = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{\min(m, K)} \sum_{k=1}^{K} \text{Precision@}k \cdot \Delta\text{rel}_{uk}$$

where U is the set of user addresses on chain, m is the number of relevant contracts for user u, and Δrel_{uk} is a binary indicator variable that takes a value of 1 when the condition of the contract at rank k being relevant is met, and 0 otherwise. Precision@k is the precision at cutoff k in the ranked list. It is defined as the proportion of relevant items found in the top k positions of the ranked list and is expressed as:

$$\text{Precision@}k = \frac{\text{Number of relevant items among the top } k}{k}$$

Normalized Discounted Cumulative Gain at k (NDCG@k): NDCG accounts for the position of the relevant items in the ranked list, giving higher importance to hits at the top of the list. For a list of k ranked items, NDCG is defined as:

$$\text{NDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k}$$

where $\text{DCG}_k = \sum_{i=1}^{k} \frac{\text{rel}_i}{\log_2(i+1)}$, and IDCG_k is the DCG value of the ideal best-possible ranking. The rel_i is the relevance score of each item in the ranked list. Simply put if the recommended item exists in the ground truth (i.e., list of actual items user interacted with) the rel_i will be 1 and otherwise 0.

4.1.2 Effectiveness

We compute evaluation metrics utilizing 5-fold cross-validation. We also adopt a negative sampling strategy in the evaluation step, for each user u_i, incorporates n negative contracts (i.e., contracts that user u_i did not interact with) alongside positive contracts (i.e., contracts that user u_i did interact with). Each recommender method aims to produce a ranked list of negative and positive contracts, placing the positive contracts higher on the list. This negative sampling approach accelerates the evaluation process and is a common practice in the literature [29, 64, 87]. A larger number of negative samples (i.e., n) leads to longer testing time while smaller n values make it trivial for all recommender methods to differentiate positive and negative samples. Thus, from an empirical standpoint, we consider $n = 100$.

Table 4.1: Effectiveness comparison of recommender systems—POP, CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, CF_{GNN}, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$—across multiple rank thresholds (i.e., $k = 1, 5, 10, 15, 20$), with bold numbers highlighting the best-performing model.

	k	POP	CF_{MF}	$\text{Hybrid}_{\text{MF-CLUSTERING}}$	$\text{Hybrid}_{\text{MF-SBERT}}$	CF_{GNN}	$\text{Hybrid}_{\text{GNN-CLUSTERING}}$	$\text{Hybrid}_{\text{GNN-SBERT}}$
HIT@k	1	0.5251	0.6285	0.5943	0.5897	0.6261	0.6206	**0.6521**
	5	0.5316	0.7775	0.7391	0.7750	0.8226	0.8209	**0.8449**
	10	0.5316	0.8261	0.7879	0.8262	0.8811	0.8799	**0.9049**
	15	0.5316	0.8539	0.8198	0.8556	0.9115	0.9077	**0.9310**
	20	0.5316	0.8740	0.8398	0.8754	0.9335	0.9264	**0.9486**
MAP@k	1	0.5251	0.6285	0.5943	0.5897	0.6261	0.6206	**0.6521**
	5	0.5282	0.6699	0.6383	0.6469	0.6802	0.6763	**0.7033**
	10	0.5282	0.6514	0.6218	0.6287	0.6584	0.6553	**0.6789**
	15	0.5282	0.6361	0.6059	0.6137	0.6430	0.6380	**0.6604**
	20	0.5281	0.6238	0.5943	0.6012	0.6292	0.6247	**0.6462**
NDCG@k	1	0.5251	0.6285	0.5943	0.5897	0.6261	0.6206	**0.6521**
	5	0.3770	0.5523	0.5051	0.5328	0.5806	0.5778	**0.6070**
	10	0.3726	0.5730	0.5220	0.5559	0.6131	0.6111	**0.6429**
	15	0.3726	0.5893	0.5393	0.5728	0.6346	0.6331	**0.6657**
	20	0.3726	0.6007	0.5502	0.5846	0.6496	0.6481	**0.6811**

The effectiveness of the recommender methods is demonstrated in Table 4.1. The $\text{Hybrid}_{\text{GNN-SBERT}}$ method outperforms the POP method and works better compared to other recommender methods (i.e., CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, CF_{GNN},

and $\text{Hybrid}_{\text{GNN-CLUSTERING}}$). This is mainly due to $\text{Hybrid}_{\text{GNN-SBERT}}$'s capability of capturing higher-order interactions and utilizing SBERT embeddings to find contract similarities. Both of these help the method to construct representations of users and contracts and provide greater recommendation effectiveness. Furthermore, the POP recommender method demonstrates comparable effectiveness for $k = 1$, indicating that a majority of users have engaged with the most popular contract. Nonetheless, as k increases, the effectiveness of the POP recommender fails to improve, suggesting its inability to capture additional interactions from users as effectively as other models. For all methods except POP, the HIT@k results are generally better compared to MAP@k and NDCG@k results. Unlike MAP@k and NDCG@k, which consider both the presence and the position of relevant contracts, HIT@k simply checks if at least one relevant contract appears in the top-k recommendations list. Therefore, HIT@k can be high even if relevant contracts are not optimally ranked, as long as they are within the top-k recommendations.

Considering CF methods, especially when k grows, CF_{GNN} provides more effective recommendations compared to CF_{MF}. This is mainly due to considering higher-order relationships among users and contracts within interaction data. However, this comes at the expense of increased latency and memory usage, as discussed in Subsection 4.1.3. Transitioning to Hybrid methods, incorporating smart contract SBERT embeddings to create hybrid methods is effective for GNN but not for MF. This could be because GNNs are already designed to learn embeddings to represent the graph structure, and SBERT can enrich these embeddings with additional contract information. On the other hand, utilizing cluster IDs of contracts to create hybrid methods is not effective for both GNN and MF. It shows the current method of clustering (i.e., LDA) is incapable of providing useful additional information to recommender methods compared to SBERT embeddings of contracts.

To adequately address the primary research question of this study, "Is it feasible to provide recommendations solely based on blockchain data?", we compare the best-performing recommender method, $\text{Hybrid}_{\text{GNN-SBERT}}$, in two scenarios: first, when fed with Ethereum dataset, and second, when fed with a popular interactions dataset like user-movie interactions in MovieLens-100k [27]. In the case of MovieLens-100k, movie genres serve as inputs to construct movie embeddings. Given the disparity in the average number of interactions per user (182 in the MovieLens-100k vs. 10 in Ethereum), we consider two variants of MovieLens recommenders: $\text{MovieLens}_{\text{FULL}}$, containing all interactions, and $\text{MovieLens}_{\text{SAMPLED}}$ maintaining the same users-to-items ratio as the Ethereum dataset (approximately 1.3) while randomly retaining 10% of each user's interactions, resulting in an average of 16 interactions per user.

Table 4.2 shows the results at various rank thresholds. $\text{Hybrid}_{\text{GNN-SBERT}}$ performs 10-

Table 4.2: Effectiveness of $\text{Hybrid}_{\text{GNN-SBERT}}$ on Ethereum and MovieLens across multiple rank thresholds (i.e., $k = 1, 5, 10, 15, 20$), with bold numbers highlighting the best-performing model.

		$\text{Hybrid}_{\text{GNN-SBERT}}$		
	k	Ethereum	$\text{MovieLens}_{\text{SAMPLED}}$	$\text{MovieLens}_{\text{FULL}}$
HIT@k	1	0.6521	0.3648	**0.7763**
	5	0.8449	0.7403	**0.9671**
	10	0.9049	0.8390	**0.9917**
	15	0.9310	0.8927	**0.9967**
	20	0.9486	0.9206	**0.9983**
MAP@k	1	0.6521	0.3648	**0.7763**
	5	0.7033	0.4890	**0.8308**
	10	0.6789	0.4623	**0.7976**
	15	0.6604	0.4412	**0.7743**
	20	0.6462	0.4236	**0.7587**
NDCG@k	1	0.6521	0.3648	**0.7763**
	5	0.6070	0.3936	**0.7486**
	10	0.6429	0.4335	**0.7476**
	15	0.6657	0.4664	**0.7578**
	20	0.6811	0.4914	**0.7682**

15% worse on Ethereum compared to $\text{MovieLens}_{\text{FULL}}$. The primary difference in their effectiveness stems from the higher number of interactions per user that $\text{MovieLens}_{\text{FULL}}$ has access to. Generally, the more interactions (for each user) a recommender method has access to, the better the recommendations it can provide. Additionally, $\text{Hybrid}_{\text{GNN-SBERT}}$ on Ethereum outperforms $\text{MovieLens}_{\text{SAMPLED}}$ demonstrating the usefulness of Ethereum data to provide recommendations. However, a user study should be conducted to verify the effectiveness of these recommendations. This can be considered an interesting direction for future work.

4.1.3 Efficiency

In this section, we compare the efficiency of recommender methods in terms of latency and memory usage during inference time. The POP recommender is excluded from this comparison because both its memory usage and latency are $O(1)$ (it always recommends the same contracts for each user, resulting in constant time complexity). All recommender methods perform the recommendation task on the CPU to allow a fair comparison of their latency and memory usage. Latency refers to the time required to calculate all necessary recommendation scores in the test set (with $n = 10$, yielding nearly 100k predictions). Memory usage denotes the amount of memory required to perform recommendations (i.e., for 100k pairs of users and contracts). The results are presented in Table 4.3. We assume that the contract cluster IDs and SBERT embeddings have been computed beforehand and are ready to be used. Therefore, we do not consider their memory footprint and the additional time required to calculate them in the latency measurement.

Table 4.3: Latency and Memory usage comparison of recommender systems—CF_{MF}, $Hybrid_{MF\text{-}CLUSTERING}$, $Hybrid_{MF\text{-}SBERT}$, CF_{GNN}, $Hybrid_{GNN\text{-}CLUSTERING}$, and $Hybrid_{GNN\text{-}SBERT}$- with bold numbers highlighting the best-performing model.

	Latency (s)	**Memory (MB)**
CF_{MF}	**6.66**	**608**
$Hybrid_{MF\text{-}CLUSTERING}$	6.76	611
$Hybrid_{MF\text{-}SBERT}$	6.74	1216
CF_{GNN}	27.73	2518
$Hybrid_{GNN\text{-}CLUSTERING}$	28.05	2535
$Hybrid_{GNN\text{-}SBERT}$	30.34	2621

As observed in Table 4.3, CF_{MF} surpasses other recommender systems in terms of memory usage and latency during inference time. This is mainly because of the simpler MF architecture utilized in CF_{MF} compared to GNN-based methods and the fact that CF_{MF} does not utilize any additional contract features. Consequently, the best performing method (i.e., CF_{MF}) produces each recommendation in 0.05 milliseconds, which is an acceptable latency in real-world applications [73]. In terms of memory usage, hybridizing both GNN and MF-based CF methods with contract cluster IDs does not significantly impact memory usage, as each contract feature in this case is a one-hot vector of size 15. However, hybridizing these methods with SBERT embeddings of contracts affects memory usage, since each contract in this case has a vector of size 768. In terms of latency, the

choice of architecture (i.e., GNN or MF) tends to have a greater impact as opposed to the choice of additional contract features (i.e., SBERT embeddings or clusters of contracts). Therefore, the GNN-based methods demonstrate nearly 4.5 times more latency during inference time.

4.2 Discussion

In this chapter, we investigated how recommender systems can use blockchain data to provide contract recommendations. We compared seven recommender methods: POP, CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, CF_{GNN}, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$. We considered the effectiveness and efficiency of these methods on the Ethereum dataset. From these, we conclude the following points:

- The $\text{Hybrid}_{\text{GNN-SBERT}}$ method outperforms the POP method and demonstrates better performance compared to other recommender methods (Table 4.1), such as CF_{MF}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, CF_{GNN}, and $\text{Hybrid}_{\text{GNN-CLUSTERING}}$. The better performance of $\text{Hybrid}_{\text{GNN-SBERT}}$ can be attributed to its ability to capture higher-order interactions and to effectively use SBERT embeddings of contracts in identifying similarities between contracts. Additionally, POP performs well for $k = \{1\}$. This indicates that the most popular item is relevant to most users. However, as k increases, the performance of this method remains constant, failing to improve, unlike the other recommender methods.

- Considering CF methods, CF_{GNN} demonstrates greater recommendation power compared to CF_{MF}, especially for higher k values (i.e., k = [5, 10, 15, 20]). The better performance of CF_{GNN} is mainly due to the consideration of higher-order interactions to construct representations of users and contracts. However, as demonstrated in Table 4.3, the greater recommendation power of CF_{GNN} comes at the expense of increased latency and memory usage on inference time.

- Considering Hybrid methods, utilizing SBERT embeddings tends to improve the GNN effectiveness. Conversely, doing the same for MF methods fails to improve the effectiveness. This could be because GNNs are already designed to learn embeddings to represent the graph structure, and SBERT can enrich these embeddings with additional contract information. On the other hand, neither MF nor GNN seem to benefit from the addition of contract cluster IDs, suggesting that clustering cannot reflect code similarity well.

- $\text{Hybrid}_{\text{GNN-SBERT}}$ performs 10-15% worse on Ethereum compared to $\text{MovieLens}_{\text{FULL}}$. The difference in their performance is primarily due to the higher average number of interactions per user in the $\text{MovieLens}_{\text{FULL}}$ dataset. Additionally, $\text{Hybrid}_{\text{GNN-SBERT}}$ on Ethereum outperforms $\text{MovieLens}_{\text{SAMPLED}}$ demonstrating the usefulness of Ethereum data to provide recommendations. However, a user study is necessary to confirm the usefulness of our blockchain recommendations.

Chapter 5

Conclusion and Future Directions

In this chapter, we begin with a summary of major contributions and insights into the feasibility of blockchain recommender systems using blockchain data (Section 5.1). We then explore future directions, focusing on advanced architectures for improved recommendation performance (Subsection 5.2.1). Finally, we discuss potential blockchain implementations of recommender methods using exclusively blockchain data (Subsection 5.2.2).

5.1 Summary

In this study, we investigated the potential of providing recommendations using only the blockchain data. Through empirical analysis of CF_{MF}, CF_{GNN}, $\text{Hybrid}_{\text{MF-CLUSTERING}}$, $\text{Hybrid}_{\text{MF-SBERT}}$, $\text{Hybrid}_{\text{GNN-CLUSTERING}}$, and $\text{Hybrid}_{\text{GNN-SBERT}}$, each recommender method's effectiveness and efficiency on interactions data were evaluated. In summary, we present the following contributions and insights:

- We constructed a dataset of 400,000 real interactions on the Ethereum network. These addresses were randomly selected from a group of addresses that had at least one transaction during a certain time frame [1]. This dataset is provided as part of this study to encourage further research into blockchain recommender systems.
- We demonstrated that the $\text{Hybrid}_{\text{GNN-SBERT}}$ outperforms other recommender methods. Additionally, the $\text{Hybrid}_{\text{GNN-SBERT}}$'s performance on Ethereum was compared

[1] We selected a random day, December 18, 2022, and, out of the 0.917 million public keys who ran at

with its performance on the $\text{MovieLens}_{\text{SAMPLED}}$ and $\text{MovieLens}_{\text{FULL}}$. $\text{Hybrid}_{\text{GNN-SBERT}}$ on Ethereum outperforms $\text{MovieLens}_{\text{SAMPLED}}$ underscoring the feasibility of designing blockchain recommender systems based solely on blockchain data.

- We demonstrated that incorporating SBERT embeddings tends to improve the GNN effectiveness. However, applying the same approach to MF does not improve its effectiveness. On the other hand, neither MF nor GNN seem to benefit from the addition of contract cluster IDs, suggesting that clustering cannot reflect code similarity well.
- We demonstrated that the CF_{MF} method, which considers only interactions, outperforms other recommender methods in terms of efficiency, including memory usage and latency at inference time. Additionally, the choice of architecture (i.e., MF or GNN) has a greater impact on latency and memory usage compared to the choice of hybridization method (i.e., contract cluster IDs or SBERT embeddings). Lastly, SBERT embeddings of contracts require more memory as they provide higher-dimensionality embeddings of contracts (i.e., 768) as opposed to contract cluster IDs (i.e., 15).

5.2 Future Work

As future work, several approaches are considered to enhance the efforts in designing blockchain recommender systems. In the subsequent sections, various methods to improve the overall performance of recommender systems are discussed. Furthermore, an exploration is conducted on the potential implementation of blockchain recommender systems on a blockchain as a practical consideration.

5.2.1 Recommendation Performance

Transformer-based GNN

In this study, a GraphSAGE [20] GNN architecture was employed to derive the construction of both user and contract embeddings based on interactions data and the initial contract embedding (i.e., the output of a pre-trained model). Recent developments indicate that transformer-based architectures, especially Graphormer [72, 93], have utilized the attention mechanism to construct more nuanced user and item embeddings. The attention mechanism may enhance the embedding of users and items in a GNN-based recommendation

least one smart contract that day, we selected 42,000 at random.

system, particularly when formulated as binary link prediction. By focusing on specific parts of the graph that are more relevant for predicting a link between a user and an item, the attention mechanism can learn the importance of different neighbors or paths in the graph. This approach enables the model to capture complex dependencies and interactions within the graph, resulting in more accurate predictions of whether a link should exist between a user and an item.

Beyond Smart Contracts

In this study, we considered all smart contracts as items, each accompanied by its corresponding Solidity code. A significant proportion of smart contracts constitute token contracts. These token contracts may be classified into fungible tokens and Non-Fungible Tokens (NFTs). Fungible tokens are recognized as cryptocurrencies, such as the Tether token, for example, implemented on the Ethereum chain, which is an abstraction from the ERC20 token contract. This abstraction enables the expedited design of token contracts through the reuse of their principal components and predefined methods, such as transfer, burn (i.e., destroying a certain amount of tokens), and mint (i.e., creating new tokens). Conversely, NFTs exhibit uniqueness, in contrast to fungible tokens, with each NFT possessing distinct characteristics. Commonly, NFTs encompass visual content in addition to textual content. The textual content may be stored on the blockchain ledger, while the visual part is often stored in decentralized storage systems such as IPFS [61]. Given the unique visual components, these non-textual features may be considered as supplementary data to construct more nuanced contract embeddings. Clearly, such item embeddings are not limited to just the Solidity code of contracts, which often exhibit substantial similarity. Rather, they can capture the distinctive characteristics of each NFT that a user has interacted with, thereby facilitating the recommendation of novel content.

5.2.2 Practical Consideration

An important direction for future work is to build a real-life implementation of a blockchain recommendation service on an actual blockchain platform. One way to do this in a centralized manner is to build it as a centralized Oracle. In that case, this would be similar to Oracles for weather data as discussed in Section 1.1. Specifically, dApps may contact external Oracles with a user's public key, and the Oracle returns the recommendation results (i.e., top-k relevant contracts based on the specified public key's previous transactions). To do this, the external Oracle needs to download the transaction (i.e., interactions) blockchain data as we did in this thesis. Then, by training recommender algorithms on

such data, it can serve the recommendation requests. For this to work, Oracle would expect some service fee to be paid by the users who request recommendations. However, this is equivalent to a Web2 solution where users pay a centralized entity to receive services. For instance, when advertisers pay Google to advertise their business, they are at the mercy of Google regarding what algorithms it uses to show their advertisements to users, and they can raise prices for their services.

On the other hand, to align with the principles of blockchains, another implementation of a blockchain recommender system is possible in a decentralized manner. In this decentralized solution, there are two approaches: 1) Everything to be decentralized: This means that a subset of nodes in the blockchain should agree to collect the data, train the recommendation algorithms, and respond to dApp requests for recommendations. This could be implemented through smart contracts. Such decentralized solutions have multiple advantages, such as Users have a choice of algorithms to use for their recommendation requests, and they can pay different transaction fees determined in a transparent manner, eliminating the single point of control as seen in the centralized Oracle solution. 2) In another approach in this design space, the recommender models can be trained off-chain (i.e., on a centralized server), and a hash of the model is stored on the ledger. Such approaches can reduce the service fee cost since running computation-intensive smart contracts on a blockchain is expensive[2].

[2]For comparison, a simple multiplication of two numbers costs approximately 300 gas units on Ethereum. With a gas price of 30 GWEI (i.e., 30×10^{-9} Ether) and an Ether price of \$2,000, the transaction fee for execution on the Ethereum blockchain is around \$0.00003. Considering a dataset of 100,000 interactions to rank contracts, yielding top-k relevant items for a user, involves numerous such multiplications. This makes such implementations challenging in real-world scenarios with billions of interactions and high recommendation requests per user per day.

References

[1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys*, 2021.

[2] Fabian Abel, Ig Ibert Bittencourt, Nicola Henze, Daniel Krause, and Julita Vassileva. A rule-based recommender system for online discussion forums. In *International Conference on Adaptive Hypermedia and Adaptive Web*, 2008.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 2005.

[4] Gediminas Adomavicius, Nikos Manouselis, and YoungOk Kwon. Multi-criteria recommender systems. In *Recommender Systems Handbook*. 2011.

[5] C.C. Aggarwal. *Attack-resistant recommender systems*. 2016.

[6] Cuneyt Gurcan Akcora, Yulia R. Gel, and Murat Kantarcioglu. Blockchain: A graph primer. *arXiv*, 2017.

[7] N. Antonopoulus and J. Salter. Cinema screen recommender agent: combining collaborative and content-based filtering. *IEEE Intelligent Systems*, 2006.

[8] Y.M. Arif, H. Nurhayati, F. Kurniawan, S.M.S. Nugroho, and M. Hariadi. Blockchain-based data sharing for decentralized tourism destinations recommendation system. *International Journal of Intelligent Engineering and Systems*, 2020.

[9] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998.

[10] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tramér, and Fan Zhang. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. 2021.

[11] R. Burke. Hybrid recommender systems: survey and experiments. *User Modeling and User-Adapted Interaction*, 2002.

[12] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 2002.

[13] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. *white paper*, 2013.

[14] Vitalik Buterin. What proof of stake is and why it matters. *Bitcoin Magazine*, 2013.

[15] H. Cai and F. Zhang. An unsupervised method for detecting shilling attacks in recommender systems by mining item relationship and identifying target items. *The Computer Journal*, 2019.

[16] L. Candillier, F. Meyer, and M. Boullé. Comparing state-of-the-art collaborative filtering systems. In *Lecture Notes in Computer Science*, 2007.

[17] Y. Cao, X. Chen, L. Yao, X. Wang, and W.E. Zhang. Adversarial attacks and detection on reinforcement learning-based interactive recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.

[18] S. Chowdhury. *Evaluating Cold-Start in Recommendation Systems Using a Hybrid Model Based on Factorization Machines and SBERT Embeddings*. PhD thesis, 2022.

[19] Y. Deldjoo, T. DI NOIA, and F.A. MERRA. A survey on adversarial recommender systems: From attack/defense strategies to generative adversarial networks. *ACM Computing Surveys*, 2020.

[20] D. El Alaoui, J. Riffi, A. Sabri, et al. Deep graphsage-based recommendation system: jumping knowledge connections with ordinal aggregation network. *Neural Computing and Applications*, 2022.

[21] M. Fang, G. Yang, N.Z. Gong, and J. Liu. Poisoning attacks to graph-based recommender systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018.

[22] Ethereum Foundation. Logging data from smart contracts with events, 2021.

[23] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005.

[24] Christian Gorenflo. *Towards a New Generation of Permissioned Blockchain Systems*. PhD thesis, University of Waterloo, 2020.

[25] Michele Gorgoglione, Umberto Panniello, and Alexander Tuzhilin. Recommendation strategies in personalization applications. *Information & Management*, 2019.

[26] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Neural Information Processing Systems (NeurIPS)*, 2017.

[27] F. Maxwell Harper and Joseph A. Konstan. Movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2016.

[28] J.D. Harris and B. Waggoner. Decentralized and collaborative ai on blockchain. In *IEEE International Conference on Blockchain*, 2019.

[29] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, 2017.

[30] J.L. Herlocker, J.A. Konstan, A.L. Borchers, and J.T. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1999.

[31] Jonathan L Herlocker et al. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 2004.

[32] Jonathan L. Herlocker, Joseph A. Konstan, and John T. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information Retrieval*, 2002.

[33] Jonathan L. Herlocker, Joseph A. Konstan, John T. Riedl, and Loren G. Terveen. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 2004.

[34] Yassine Himeur, Aya Sayed, Abdullah Alsalemi, Faycal Bensaali, Abbes Amira, Iraklis Varlamis, Magdalini Eirinaki, Christos Sardianos, and George Dimitrakopoulos. Blockchain-based recommender systems: Applications, challenges and future opportunities. *Computer Science Review*, 2022.

[35] R. Hu, Y. Guo, M. Pan, and Y. Gong. Targeted poisoning attacks on social recommender systems. In *2019 IEEE Global Communications Conference*, 2019.

[36] Yao-Chieh Hu, Ting-Ting Lee, Dimitris Chatzopoulos, and Pan Hui. Analyzing smart contract interactions and contract level state consensus. *Concurrency and Computation: Practice and Experience*, 2019.

[37] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, 2021.

[38] D. Jannach, M. Zanker, M. Ge, and M. Gröning. Recommender systems in computer science and information systems - a landscape of research. In *Proceedings of the 13th International Conference on E-Commerce and Web Technologies*, 2012.

[39] M. Jourdan, S. Blandin, L. Wynter, and P. Deshpande. Characterizing entities in the bitcoin blockchain. In *IEEE International Conference on Data Mining Workshops*, 2018.

[40] H.N. Kim, A.T. Ji, I. Ha, and G.S. Jo. Collaborative filtering based on collaborative tagging for enhancing the quality of recommendations. *Electronic Commerce Research and Applications*, 2010.

[41] F. Kong, X. Sun, and S. Ye. A comparison of several algorithms for collaborative filtering in startup stage. *IEEE Transactions on Networks, Sensing and Control*, 2005.

[42] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. 2008.

[43] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[44] Maciej Kula. Metadata embeddings for user and item cold-start recommendations. In Toine Bogers and Marijn Koolen, editors, *Proceedings of 9th ACM Conference on Recommender Systems*, 2015.

[45] Himabindu Lakkaraju, Julian McAuley, and Jure Leskovec. Understanding the interplay between titles, content, and communities in social media. In *Proceedings of the Seventh International AAAI Conference on Weblogs and Social Media*, 2013.

[46] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1982.

[47] Ken Lang. Newsweeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[48] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 2003.

[49] A. Lisi, A. De Salve, P. Mori, and L. Ricci. A smart contract based recommender system. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, 2019.

[50] Siwei Liu. *Effective graph representation learning for ranking-based recommendation*. PhD thesis, 2023.

[51] Xiao Fan Liu, Xin-Jian Jiang, Si-Hao Liu, and Chi Kong Tse. Knowledge discovery in cryptocurrency transactions: A survey. *arXiv*, 2021.

[52] M Loukili, F Messaoudi, and M El Ghazi. Machine learning based recommender system for e-commerce. *International Journal of Artificial Intelligence*, 2023.

[53] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Stanford University, 1979.

[54] Ralph Charles Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology*, 1988.

[55] R. Meteren and M. Someren. Using content-based filtering for recommendation. In *Proceedings of ECML 2000 Workshop: Machine Learning in the Information Age*, 2000.

[56] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[57] M. Niranjanamurthy, B. Nithya, and S. Jagannatha. Analysis of blockchain technology: pros, cons and swot. *Cluster Computing*, 2019.

[58] Harris Papadakis, Antonis Papagrigoriou, Costas Panagiotakis, Eleftherios Kosmas,

and Paraskevi Fragopoulou. Collaborative filtering recommender systems taxonomy. *Knowledge and Information Systems*, 2022.

[59] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 1980.

[60] C. Porcel, A. Tejeda-Lorente, M.A. Martínez, and E. Herrera-Viedma. A hybrid recommender system for the selective dissemination of research resources in a technology transfer office. *Information Sciences*, 2012.

[61] Protocol Labs. InterPlanetary File System (IPFS), 2015.

[62] S. Ranshous, C. A. Joslyn, S. Kreyling, K. Nowak, N. F. Samatova, C. L. West, and S. Winters. Exchange pattern mining in the bitcoin transaction directed hypergraph. In *in Proceedings of the 21th International Conference on Financial Cryptography and Data Security*, 2017.

[63] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[64] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the 14th ACM Conference on Recommender Systems*, 2020.

[65] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, 1994.

[66] F. Rezaimehr and C. Dadkhah. A survey of attack detection approaches in collaborative filtering recommender systems. *Artificial Intelligence Review*, 2021.

[67] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems: Introduction and Challenges*. 2015.

[68] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system–a case study. *ACM Web mining and social network analysis workshop*, 2000.

[69] Badrul Sarwar, George Karypis, Joseph A Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, 2001.

[70] Fred B. Schneider. Byzantine generals in action: Implementing fail-stop processors. *ACM Transactions on Computer Systems (TOCS)*, 1984.

[71] Ilya Sergey and Aquinas Hobor. A concurrent perspective on smart contracts. In *Financial Cryptography and Data Security*, 2017.

[72] Yu Shi, Shuxin Zheng, Guolin Ke, Yifei Shen, Jiacheng You, Jiyan He, Shengjie Luo, Chang Liu, Di He, and Tie-Yan Liu. Benchmarking graphormer on large-scale molecular modeling datasets. *arXiv*, 2022.

[73] M. Singh. Scalability and sparsity issues in recommender datasets: a survey. *Knowledge and Information Systems*, 2020.

[74] Douglas R. Stinson. Some observations on the theory of cryptographic hash functions. *Designs, Codes, and Cryptography*, 2006.

[75] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. In *Advances in Artificial Intelligence*, 2009.

[76] P. Symeonidis, A. Nanopoulus, and Y. Manolopoulos. Movieexplain: a recommender system with explanations. In *Proceedings of the 2009 ACM Conference on Recommender Systems*, 2009.

[77] Gábor Takács and Domonkos Tikk. Alternating least squares for personalized ranking. In *Proceedings of the 6th ACM Conference on Recommender Systems*, 2012.

[78] Yaniv Tal, Brandon Ramirez, and Jannis Pohlmann. The graph: A decentralized query protocol for blockchains. 2018.

[79] K. Toyoda, P. T. Mathiopoulos, and T. Ohtsuki. A novel methodology for hyip operators' bitcoin addresses identification. *IEEE Access*, 2019.

[80] Muneeb Ul Hassan, Mubashir Husain Rehmani, and Jinjun Chen. Anomaly detection in blockchain networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 2023.

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, 2017.

[82] Haodi Wang and Thang Hoang. ezdps: An efficient and zero-knowledge machine learning inference pipeline, 2023.

[83] Q. Wang, R. Li, Q. Wang, and S. Chen. Non-fungible token (nft): Overview, evaluation, opportunities and challenges. *arXiv*, 2021.

[84] Qin Wang, Guangsheng Yu, Shange Fu, Shiping Chen, Jiangshan Yu, and Sherry Xu. A referable nft scheme. *arXiv*, 2022.

[85] S. Wang, C. Huang, J. Li, Y. Yuan, and F.-Y. Wang. Decentralized construction of knowledge graphs for deep recommender systems based on blockchain-powered smart contracts. *IEEE Access*, 2019.

[86] Shoujin Wang, Xiuzhen Zhang, Yan Wang, and Francesco Ricci. Trustworthy recommender systems. *ACM Transactions on Intelligent Systems and Technology*, 2022.

[87] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019.

[88] Xiang Wang, Xiangnan Yu, Ziqi Gu, Yuanchao Liu, Mengqi Zhang, Shuxian Zhang, and Yujing Gao. Graph neural networks for social recommendation. *The World Wide Web Conference*, 2019.

[89] C.A. Williams, B. Mobasher, and R. Burke. Defending recommender systems: Detection of profile injection attacks. *Service Oriented Computing and Applications*, 2007.

[90] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[91] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Computing Surveys*, 2022.

[92] H. H. Sun Yin, K. Langenheldt, M. Harlev, R. R. Mukkamala, and R. Vatrapu. Regulating cryptocurrencies: A supervised machine learning approach to de-anonymizing the bitcoin blockchain. *Journal of Management Information Systems*, 2019.

[93] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *35th Conference on Neural Information Processing Systems*, 2021.

[94] G Yu, Q Wang, T Altaf, X Wang, X Xu, and S Chen. Predicting nft classification with

gnn: A recommender system for web3 assets. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2023.

[95] F. Zhang and S. Wang. Detecting group shilling attacks in online recommender systems based on bisecting k-means clustering. *IEEE Transactions on Computational Social Systems*, 2020.

[96] F. Zhang and Q. Zhou. Hht–svm: An online method for detecting profile injection attacks in collaborative recommender systems. *Knowledge-Based Systems*, 2014.

[97] Shuai Zheng, Chris Ding, and Feiping Nie. Regularized singular value decomposition and application to recommender system. *arXiv*, 2018.

[98] Morteza Zihayat, Anteneh Ayanso, Xing Zhao, Heidar Davoudi, and Aijun An. A utility-based news recommendation system. *Decision Support Systems*, 2019.

[99] Òscar Celma and Paul Lamere. A music similarity function based on signal analysis. In *Proceedings of the International Conference on Multimedia*, 2008.