# Stake for Ranking:
# Towards the Tokenization of Recommender Systems
# (Early Draft)

Todd Chapman
todd@snickerdoodlelabs.io

November 26, 2023

# Contents

# 1 Abstract

This work proposes *stake for ranking*, a cryptoeconomic primitive that introduces a token mechanic to content-based recommender systems. Imparting an economic cost and high degree of content authorship traceability increases the robustness of recommender systems to sybil and spoofing attacks. Additionally, end user privacy is greatly enhanced due to data availability through decentralized content distribution networks.

# 2 Introduction

The design, implementation, and stability analysis of cryptoeconomic primitives [1] has been an intensive area of research since the emergence of smart contract networks like Ethereum [2]. Some prominent examples include Proof of Stake [3] (now the consensus mechanism securing Ethereum itself), tokenization (particularly stable coins backed by digital or traditional assets), curve bonding [4], token curated registries [5] (TCRs), yield staking, and distributed autonomous organizations [6] (DAOs).

In short, a cryptoeconomic primitive is a type of economic game in which the presence of a programmable token asset (which may or may not have a fixed supply) is a prerequisite for the proper functioning of the game. Proper functioning encompasses the initial and ongoing incentive alignment of the participants so that a stable (quasi)equilibrium of the game or market can be achieved without a centralized operator. The list of cryptoeconomic primitive examples given have sought to solve problems in finance, governance, and information asymmetry, as well as other areas.

This work proposes a new primitive, stake for ranking, which aims to serve as the core of a self-governing, content-based recommender system. This can be seen as an addition to the tool kit of curation market primitives which, among other things, seek to provide verifiable information to all parties on equal footing. As will be outlined, the token mechanics of stake for ranking also share similarities both with TCRs and yield staking.

# 3 Recommender Systems

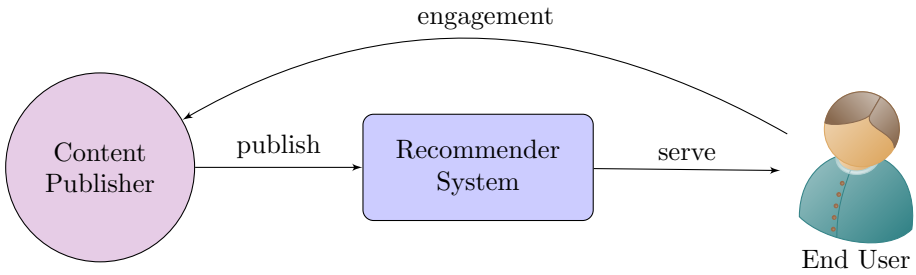


Figure 1: Recommender systems are pervasive in all aspects of modern digital commerce. They are the discoverability broker between publishers of content and end users searching for relevant information.

Recommender systems are pervasive in all aspects of modern digital commerce, social media, and digital advertising networks [7] [8] [9]. Google, Amazon, Meta, X.com, etc. all face the problem of how to suggest relevant content to their users. These systems face challenges in terms of scalability, transparency, and moderation [10].

Serving the recommendations and training the associated models is a significant engineering challenge with a great deal of subjectivity present in determining what makes a recommendation relevant, particularly for content-based recommender systems. The mere act of a client-side application selecting a recommended content entity reveals information about the end-user to the broker of the recommendation or even third-party services which would significantly compromise end user privacy after multiple sessions without end user consent [11]. The lack of economic repercussions for bad content authors and lack of incentive alignment between brokers, authors, and end users means recommender systems deployed in marketplace settings are highly susceptible to detrimental manipulation by all parties.

Stake for ranking specifically attempts to improve upon recommender systems where the content corpus and attribute space is potentially large and the content authors are entities external to the broker of the recommendation engine (i.e the organization that operates the recommender system is not the same entity that authors content). It is these systems which are highly susceptible to sybil attack by malicious content authors who post poor quality, or even false information and are able to boost the ranking of their content by exploiting ML/AI based algorithms that determine what content is served. This can be observed in the proliferation of fake/counterfeited product listings on Amazon, scam advertisements on Google, and bot reposts on Facebook and X.com.

There is essentially no economic cost or historic traceability to the authors that create bad content; if they are removed from the system or deranked, they can simply create a new identity at no cost and try again. End consumers have no means to independently verify the authenticity or history of the content author's identity [12].

Additionally, well-meaning content authors, even those that are well known, are often censored, deranked, or demonetized with no warning or explanation and often with no verifiable indication that they have been deranked (colloquially known as "shadow banning"). Furthermore, the brokers of recommendation systems are not directly incentivized to protect end users, they are instead directly incentivized to maximize engagement, which has a first-order effect on revenue generation.

## 3.1 Typical Recommender System Models

There are two primary strategies by which modern recommender systems approach the task of calculating relevance of content to an end user: content-based filtering and collaborative filtering. While stake for ranking is most closely associated with content-based filtering, for completeness both are defined here.

### 3.1.1 Content-based Filtering

Content-based filtering is a strategy where content is explicitly assigned keywords, features, or attributes in a database-like setting. These attributes are then compared against a user's profile state to determine relevance to that specific user. The user's profile is derived from their engagement with sites, shopping carts, and other pieces of content.

Some of the advantages of content-based filtering include:

- only the data from a user's personal profile is required to start producing recommendations. Hence, content-based filtering does not suffer from the "cold start" problem to the same degree that collaborative filtering systems do since they require many active users before recommendations can be optimized.
- recommendations tend to be highly relevant to the user due to the direct match of a user's profile attributes with content attributes.
- recommendations are highly interpretable and transparent to the end user.
- implementation simplicity.

Some typical disadvantages of traditional content-based recommender systems include:

- difficulty producing diverse or novel recommendations since only the user's own profile data and historical interactions are used for calculating relevance (unlike collaborative filtering where data from many user's are combined to produce recommendations).
- difficulty scaling the recommender database since each content item must maintain a curated list of features or attributes which accurately and consistently represents the content.

### 3.1.2 Collaborative Filtering

Recommender systems which leverage collaborative filtering methods employ algorithms which combine many user profiles together into a single predictive model which is able to propose recommendations to a user. The winning proposal of the Netflix Prize [9] is a famous example of a collaborative filtering system based on a matrix factorization approach [13].

Collaborative filtering excels at:

- recommending new relevant content outside of a user's historical interactions.
- high scalability for traditional implementations since there is no need to curate content features, only user profile data is required.

However, collaborative filtering has drawbacks related to:

- producing quality recommendations before there is a sufficient number of user profiles to train the recommender model (called the "cold start" problem).
- producing interpretable or common-sense recommendations (since predictions take into account many user profiles which may have little or no overlap in taste).

# 4 Stake for Ranking

## 4.1 Preliminaries

Stake for ranking attempts to address some of the shortcomings of closed-source, proprietary recommender systems by removing the need for a centralized party to curate content and its relative importance to an end user and allow for the natural and direct alignment of incentives between content authors and their audiences. It contributes:

- Transparent ranking mechanics via auditable on-chain logic
- Sybil resistance against spam listings due to economic costs of entering recommender system and curating attribute weights
- Enhanced end-user privacy due to decentralized content delivery and rankings hosted on a public blockchain
- Can be coupled with ERC-7529 [14] for enhanced content spoofing protection and DNS/domain-based search and moderation

## 4.2 Components

This section will describe the minimal components (and their roles) necessary for implementing the stake for ranking cryptoeconomic primitive. Depending on the usecase, there may likely be other required components for a proper implementation.

### 4.2.1 Content Objects

A *content object* is an on-chain asset (a smart contract) owned or operated by a *content author* and can be an ERC-721 compatible non-fungible asset, an ERC-20 token, or any other smart contract-based data structure. It is also assumed that the content object is some kind of asset that an end-user will be transacting with directly, like paying into a contract to mint an NFT or fungible token (thus the need to attract end users to your content object in the first place).

A content object will possess an interface defined in section 4.3 and a member variable mapping an attribute string (i.e. a human-readable tag) to a weight. For content object, $n$, this mapping represents an infinite dimensional sparse vector of weights, $w_n \in \mathbb{N}^{+\infty}$. Hence, we place no limits on the potential number of attributes that might be of use to the recommender system and for programming convenience in the context of the Ethereum Virtual Machine [15] (which works nicely with unsigned integers), weights are taken to be non-negative integer values.

We take the convention that the weight given to attribute $i$ is the value of the $i^{th}$ row of $w_n$:

$$attribute_i \rightarrow w_{n,i} \tag{1}$$

The weight vector, $w_n$, is limited to a finite number of non-zero entries (the exact number would depend on the usecase), and the magnitude of the non-zero entries is proportional to the amount of token the content author wishes to stake to that row's associated keyword as discussed later in section 4.4.

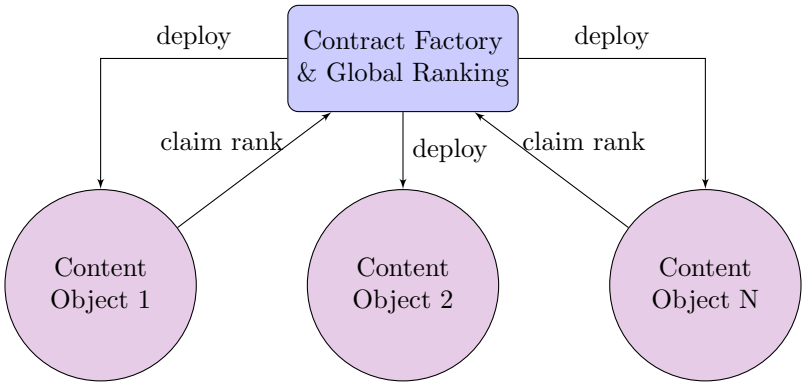


Figure 2: Stake for ranking works well with factory patterns since the contract factory serves as a natural location to store global ranking data structures. It also allows for programmatic exclusion of content not deployed through the factory itself which can be useful for certain applications.

#### 4.2.2 Global Ranking

The smart contract architecture is assumed to be based on a factory pattern (as depicted in figure 2), although it is not strictly necessary. Contract factories are a common pattern found in NFT marketplaces (like Rarible and Opensea) and decentralized finance protocols (like Uniswap and Aave). A contract factory is a natural location to host the data structures responsible for storing the global ranking of content objects with respect to each attribute. It also allows for the ranking structure to exclude the listing of content objects that were not deployed through standard template that would be defined by a contract factory.

Global ranking can be stored in a mapping variable that maps from an attribute string to a doubly linked list:

$$attribute_i \rightarrow \{(2^{256}-1) \leftrightarrow w_{...,i} \leftrightarrow w_{k_i+1,i} \leftrightarrow w_{k_i,i} \leftrightarrow w_{k_i-1,i} \leftrightarrow w_{...,i} \leftrightarrow 0\} \tag{2}$$

The weight, $w_{k_i,i}$, belongs to the $k^{th}$ highest ranked content object for the $i^{th}$ attribute (as described in equation 1) such that $w_{k_i,i} < w_{k_i+1,i}$ (as will be discussed in detail in section 4.4 a higher weight requires a exponentially increasing amount of staked value). Notice that the head of any attribute's linked list is the maximum value of a $uint256$ and the tail value is 0, thus $0 < w_{k_i,i} < (2^{256}-1); \forall i$.

This means that content objects that do not assign a non-zero weight to attribute $i$ do not appear in that attribute's corresponding rankings, so $0 \leq k_i \leq N$ where $N$ is the total number of content objects globally. Section 4.3 will outline the methods for content authors to call in order to enter this global ranking for a given attribute.

Doubly linked lists are an ideal data structure for tracking dynamically allocated ordered lists in a smart contract setting due to the constant complexity of insertion and deletion of a list entry (gas costs remain constant as the size of the list grows). Furthermore, entries can easily be traversed in a paginated fashion in either ascending or descending order.

#### 4.2.3 Likeness metric

A content-based recommender system must prescribe a metric by which a content object can be determined to be relevant to an end user. This is an off-chain function that takes as input a user's historical interactions and content object's weight vector and returns a score. We assume in this work that a perfect match would be normalized to 1 and a perfect miss would be 0. For completeness, a candidate process is proposed for determining relevance of a content object to an end user.

Consider a user who has shown interest in, or interacted with a set of content objects with weight vectors, $\{..., w_n, ...\}$. We can represent this unordered set of vectors by the matrix $W$ whose columns are the weight vectors themselves. Then, using the singular value decomposition, we can create a low-dimensional embedding space.

$$W = U\Sigma V^T \tag{3}$$

By definition, $U$, is an orthogonal matrix, (i.e. $U^T U = I$), and captures the span of set of weight vectors. Importantly, the columns of the matrix $U$ are optimally ordered, meaning that the first column of $U$ (corresponding to the largest diagonal entry of $\Sigma$) represents the best single-vector approximation of the set $\{..., w_n, ...\}$.

As a simple and computationally efficient model for vectorizing a user's interests, we take the first column of $U$, which will be referred to as $\tilde{u}$, as our user feature vector as it encapsulates the optimal (in the Euclidean norm) representation of all content they've previously interacted with in a single vector.

When a user encounters a new content object, $w_{new}$, its relevance score can be computed as

$$score = \frac{\tilde{u}^T w_{new}}{w_{new}^T w_{new}} \tag{4}$$

This score function has the desired properties; if it is a perfect match, then $score = 1$ whereas if the object has no component in common with the user's interest vector, then $score = 0$ (i.e. $\tilde{u} \perp w_{new}$).

It is important to remember that this is only one of an infinite number of possible likeness metrics which could be implemented by a user client for calculating relevant recommendations. This has the benefit that the SVD is a number of efficient algorithms for dense and sparse data sets and can be computed in near real time for very large collections of data. It also has a history of being used successfully in recommender system settings.

## 4.3 Interface Specifications

This section describes a minimal interface specification necessary for stake for ranking participants to manipulate content object weight vectors defined in section (4.2.1) and the global ranking listed list defined in section 4.2.2. Function specifications are written in Solidity format.

### 4.3.1 Content Objects

Content object contracts must have functions for staking and destaking economic value with respect to an attribute that is relevant to it. Smart contracts implementing the content object interface given here are responsible for calling the requisite functions of the smart contract hosting the global ranking storage variable (section 4.3.2).

```
/// @notice Retrieve all attributes and associated weights from the content object
/// @dev Its assumed the size of this array is small enough to return without pagination
function getTagArray() external view returns (Tag[] memory);

/// @notice Adds a new attribute to the global ranking namespace and pulls stake from the caller account
/// @dev Global ranking linked list: max(uint256) <-> _newWeight <-> 0
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _newWeight Weight to assign to new global attribute listing
function newGlobalAttribute(string memory attribute, uint256 _newWeight) external;

/// @notice Assigns weight to an attribute that is already tracked by the global ranking namespace but hasn't been weighted by this content object yet, pulls stake from the caller account
/// @dev  Global ranking linked list: _newWeight <-> _existingWeight
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _newWeight New linked list entry that will point to _existingWeight listing
/// @param _existingWeight upstream listing that will point to _newWeight
function newLocalAttributeUpstream(
        string memory attribute,
        uint256 _newWeight,
        uint256 _existingWeight
    ) external;

/// @notice Assigns weight to an attribute that is already tracked by the global ranking namespace but hasn't been weighted by this content object yet, pulls stake from the caller account
/// @dev  Global ranking linked list: _existingWeight <-> _newWeight
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _existingWeight upstream listing that will point to _newWeight
/// @param _newWeight New linked list entry that will point to _existingWeight listing
function newLocalAttributeDownstream(
```

```
        string memory attribute,
        uint256 _existingWeight,
        uint256 _newWeight
    ) external;

/// @notice Replaces an existing listing that has expired
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _weight The expired slot to replace with a new listing
function replaceExpiredListing(tring memory attribute, uint256 _weight) external;

/// @notice Removes this contract's listing under the specified tag
/// @param attribute Human readable string denoting the target attribute to destake
function removeListing(string memory attribute) external;
```

Listing 1: A minimal interface for a content object.

#### 4.3.2 Global Ranking Linked List

Contracts which host a global ranking storage variable must have functions for retrieving the current entries of the doubly-linked list for a given attribute. Functions which manipulate the global ranking storage variable are responsible for ensuring that only compatible smart contracts are able to call them (see section 4.2.2 regarding the factory pattern assumption).

```
/// @notice Initializes a doubly-linked list under the given attribute
/// @dev Should only be called by content objects
/// @dev Global ranking linked list: max(uint256) <-> _newWeight <-> 0
/// @param attribute Human readable string denoting the target attribute to insert into
    global ranking
/// @param _newWeight First list entry between max(uint256) and 0
function initializeAttribute(string memory attribute, uint256 _newWeight) external;

/// @notice Inserts a new listing into the doubly-linked Listings mapping
/// @dev Global ranking linked list: _newWeight <-> _existingWeight
/// @dev Should only be called by content objects
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _newWeight New linked list entry that will point to the Listing at
    _existingWeight
/// @param _existingWeight Listing slot that will be pointed to by the new Listing at
    _newWeight
function insertUpstream(
        string memory attribute,
        uint256 _newWeight,
        uint256 _existingWeight
    ) external;

/// @notice Inserts a new listing into the doubly-linked Listings mapping
/// @dev Global ranking linked list: _existingWeight <-> _newWeight
/// @dev Should only be called by content objects
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _existingWeight Listing slot that will be pointed to by the new Listing at
    _newWeight
/// @param _newWeight New linked list entry that will point to the Listing at
    _existingWeight
function insertDownstream(
        string memory attribute,
        uint256 _existingWeight,
        uint256 _newWeight
    ) external;

/// @notice Replaces an existing listing that has expired (works for head and tail listings)
/// @dev Should only be called by content objects
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _weight The expired weight to replace with a new listing
function replaceExpiredListing(tring memory attribute, uint256 _weight) external;

/// @notice removes a listing in the marketplace listing map
/// @dev Should only be called by content objects
/// @param tag Human readable string denoting the target attribute
/// @param _removedWeight the weight to remove from the Listing data structure
```

```
function removeListing(string memory attribute, uint256 _removedWeight) external;

/// @notice Returns an array of Listings from the marketplace linked list from highest to
    lowest ranked
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _startingWeight Weight to start at in the ranking linked list
/// @param numListings number of entries to return
/// @param filterActive boolean flag indicating if the results should include expired
    listings (false) or not (true)
function getListingsForward(
        string memory attribute,
        uint256 _startingWeight,
        uint256 numListings,
        bool filterActive
    )
    external
    view
    returns (string [] memory, Listing [] memory);

/// @notice Returns an array of Listings from the marketplace linked list from lowest to
    highest ranked
/// @param attribute Human readable string denoting the target attribute to stake
/// @param _startingWeight Weight to start at in the linked list
/// @param numListings number of entries to return
/// @param filterActive boolean flag indicating if the results should include expired
    listings (false) or not (true)
function getListingsBackward(
        string memory attribute,
        uint256 _startingWeight,
        uint256 numListings,
        bool filterActive
    )
    external
    view
    returns (string [] memory, Listing [] memory);

/// @notice Returns the total number of listings under a given attribute (including expired
    listings which have not been removed)
/// @param attribute Human readable string denoting the target attribute to stake
function getTagTotal(string memory attribute)
    external
    view
    returns (uint256);
```

Listing 2: A minimal interface for reading and writing to the global ranking namespace.

## 4.4 Mechanics

This section describes the usage of the components described in section 4.2. Specifically, this section describes the minimal token mechanics necessary to implement for stake for ranking as well as optional token mechanics which could be introduced depending on the usecase. This section also discusses aspects of content delivery and spoof resistance which are not necessarily tied to the use of stake for ranking, but which compliment it.

### 4.4.1 Economics

The weight vector of a content object (section 4.2.1) is constructed by assigning weights to relevant attributes (equation 1). In order for a network participant to assign a weight to an attribute for a given content object, the participant must stake economic value (i.e. lock up tokens) in an amount proportional to the magnitude of the weight. A candidate staking function (inspired by the concentrated liquidity interval spacing introduced in Uniswap V3 [16]) is:

$$stake_{n,i} = 1.0001^{w_{n,i}} \tag{5}$$

In this model, increasing a content object's attribute weighting by 1 unit requires 0.01% more staked token value. Hence, as the desired target weight increases linearly, the amount of stake required to achieve that target weight increases exponentially, disincentivizing frivolous weighting of genuinely unrelated tags and also normalizing the weighting power of large token holders to make it more difficult to grossly overweight

high-demand tags. A non-zero minimum weight to enter the global ranking namespace could optionally be imposed to further disincentivize applying small staking amounts to attributes that are not actually relevant to a content object.

As explained in section 4.2.2, content objects are discoverable through a global ranking namespace, where each attribute (that has at least one associated content object) maps to a double linked list data structure. The entries are linked in order of their weights, which requires stake proportional to equation 5 to obtain. Thus, the global ordering for an attribute effectively forms an open order book for ranking of content associated with a given tag. Content authors and end users both have direct insight into the relative ranking of content for a given tag/attribute.

Staking economic value does not result in rewards issued directly from the protocol itself to the depositor. Instead, it boosts the discoverability and visibility of a content object in a marketplace setting given a likeness metric that meets the criteria specified in section 4.2.3, leading to greater likelihood of end user engagement through transactions and thus revenue. Depositors can rearrange their stake among different attributes as they search the state space for the correct attribute allocation for their target audience and can reclaim all of their stake and exit the market if necessary. This is an important distinction from existing staking models. Popular tags (i.e. tags that are relevant to the largest number of users) will cost more for a higher number of impressions while tags associated with more specialized audiences will likely be cheaper.

Another optional feature, depending on the application, is to couple stake for ranking with a DAO. The DAO can be used to govern the global ranking namespace. For example, staked tokens could be made subject to forfeiture if the token holder community of the DAO deems the content object to be detrimental to the associated marketplace.

### 4.4.2 Content Delivery

Unlike existing recommender systems which serve content through traditional content distribution networks (CDNs), rankings and content in the stake for ranking setting are naturally delivered through decentralized systems. This makes it much more difficult to track what users preferences without their consent.

Global rankings are themselves hosted directly on an appropriated public blockchain network, thus rankings and weight vectors can be retrieved through any supporting RPC provider or in the extreme case, a user can always host their own node. Off-chain content pointed to by content object smart contracts can itself be hosted in content-addressable CDNs like IPFS or Arweave, meaning that a user client could fetch off-chain content through any public gateway or, again, embed the appropriate node in the user client itself.

### 4.4.3 Searchability and Spoof Resistance

While staking economic value helps to mitigate the effectiveness of sybil attacks against the content corpus of the recommender system, it alone cannot prevent a publisher for masquerading as another publisher. Leveraging the standard proposed in ERC-7529 (see reference [14] for details) would allow an end user client to independently verify that a content object is associated with a know DNS domain (e.g. snickerdoodle.com).

An additional benefit of being able to link a business' DNS domain with a smart contract-based content object is searchability. A user can easily find content posted by domains they are familiar with using the discoverability feature of ERC-7529, then leverage the global ranking namespace to find other content that is closely related that content based on the staked attributes.

Combining stake for ranking with ERC-7529 leads to a fully decentralized recommender system that is quite challenging to manipulate via spoofing or sybil attacks.

# 5 Conclusions

Stake for ranking represents a novel crytoeconomic primitive that brings tokenized economic markets to the field of recommender systems. Staking of economic value in order to enter the recommeder system mitigates the effectiveness of sybil attacks and incentivizes content authors to accurately represent their listings in order to connect with appropriate end users. End users can calculate recommendations based on only their local state and on-chain rankings; data-leakage of end user preferences can be prevented through the use of content-addressable networks like Arweave or IPFS. Content can be associated with known business domains via ERC-7529 leading to increased resilience against spoofing while also making smart contracts easier to search for.

# References


[1] Jacob Horne. The emergence of cryptoeconomic primitives. https://www.coinbase.com/blog/the-emergence-of-cryptoeconomic-primitives, 2018.

[2] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[3] QuantumMechanic. Proof of stake instead of proof of work. https://bitcointalk.org/index.php?topic=27787.0/, 2011.

[4] The Graph. An introduction to bonding curves. https://thegraph.academy/curators/introduction-to-bonding-curves/.

[5] Mike Goldin. Token-curated registries 1.0. https://docs.google.com/document/d/1BWWC__-Kmso9b7yCI_R7ysoGFIT9D_sfjH3axQsmB6E/edit, 2017.

[6] Ralph Merkle. Daos, democracy and governance. *Cryonics Magazine*, 37(4):28–40, 2016.

[7] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[8] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. *Recommender systems handbook*, pages 73–105, 2011.

[9] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

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

[11] Antoine Boutet, Florestant De Moor, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Antoine Rault. Collaborative filtering under a sybil attack: Similarity metrics do matter! In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 466–477. IEEE, 2018.

[12] Chen Lin, Si Chen, Meifang Zeng, Sheng Zhang, Min Gao, and Hui Li. Shilling black-box recommender systems by learning to generate fake user profiles. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[13] Mingrui Wu. Collaborative filtering via ensembles of matrix factorizations. In *KDD Cup and Workshop 2007*, pages 43–47, 2007.

[14] Todd Chapman, Charlie Sibbach, and Sean Sing. Erc-7529: Bridging dns and blockchain for automatic contract discovery and author verification. https://www.npmjs.com/package/@snickerdoodlelabs/erc7529, 2023.

[15] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[16] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. *Tech. rep., Uniswap, Tech. Rep.*, 2021.