SoK: Taming Smart Contracts With Blockchain Transaction Primitives: A Possibility?

***Abstract*—Blockchain (decentralized) applications rely heavily on user-programmed transaction protocols called "smart contracts" as implementation building blocks. Such user-programmed transactions are needed because blockchain platforms typically only natively offer limited transaction types. Smart contracts, while providing significant flexibility and customizability of transaction behavior implementation, have a wide range of significant limitations that will likely impede the adoption of blockchains in specific application contexts. This paper explores the possibility of reducing many frequently occurring smart contract transaction behaviors into a smaller set of generalized abstractions. The smaller set of generalized transaction patterns may then be considered potential candidates for integrating blockchains as native transaction types out-of-the-box in blockchains. We extracted 93GB of transaction data from the ETHEREUM blockchain for analysis. We also studied smart contract source codes of applications described in research and industry publications from diverse domains. The results of our survey analysis validate the rationale for the survey.**

***Index Terms*—Smart Contracts, Blockchain Transactions, Analysis**

I. INTRODUCTION

Blockchains are being explored in an increasingly broad range of application contexts including Marketplaces [1]–[5], Supply Chain Management [6]–[10], Property Sale and Management [11]–[14], Insurance [15]–[19], Education [20]–[24], eGovernance [25]–[29], Internet of Things [30]–[34], and many others [35]. For these applications, the primary interest is to enable decentralized execution and management of business transactions and their associated processes to increase transparency and trustworthiness and/or improve multi-party processes' efficiency (e.g., supply chains) by eliminating bottlenecks that often bedevil centralized processing in such contexts.

However, because blockchains were initially proposed to support cryptocurrency and other financial applications, there is a gap in "blockchain-native" transactional support for applications outside these contexts. To appreciate this, observe that the primary transactional behavior needed for cryptocurrency applications is the ability to transfer currency from one account to another. Therefore, It is unsurprising that all blockchain platforms provide the TRANSFER transaction type out-of-the-box. In other words, blockchains provide *transfer* transactional behavior and automatic validity checking without the need for user programming, e.g., automatic checking for the doublespend error. On the other hand, where transactional behavior differs from the mere transferring of crypto-assets like cryptocurrencies, users must implement desired transaction behavior and validity checking themselves as programs popularly known as *smart contracts*. A smart contract is a self-executing program that implements terms of contracts within the program code. They automatically execute, verify, and/or enforce the terms of a contract when predefined conditions are met without central authorities governing the processes. They are implemented in high-level programming languages such as Solidity for ETHEREUM and executed in a virtual machine provided by a blockchain platform. They are immutable once deployed on the blockchain. In concrete terms, smart contracts are similar to classes in object-oriented programming. They have variables and methods to represent the structures of any non-native digital assets managed on the blockchain. Non-native digital assets can range from digital media like audio, video, and art to digital twins of physical assets like cars, houses, etc. Similarly, smart contracts may encode a wide range of non-native transactional behaviors such as buying, selling, leasing an asset, earning rewards through some activity, claiming rewards or benefits due, casting a vote in an election, and so on.

While smart contracts enable significant blockchain behavior extensibility through user programs, they have several limitations as a transaction mechanism. Besides the burden imposed on application developers to create a set of smart contracts for each use case, smart contracts take time to discover and reuse. They are also prone to programming errors and security vulnerabilities, which often have significant financial implications. For example, a seemingly minor programming bug led to a loss of over ∼$48 million in the DAO [36] attack. Their execution is associated with blockchain fees, which can be arbitrarily high. Finally, because blockchains are oblivious to the high-level application and transaction semantics encoded in smart contracts, they cannot reason accurately about concurrency conflicts. Consequently, platforms like ETHEREUM run smart contracts sequentially, which results in lower throughputs when compared to the execution of native transactions, which they can parallelize more effectively.

Many of these challenges are likely to slow down the adoption of blockchains in broader contexts. We can leverage lessons learned from developing and evolving relational database systems to mitigate these challenges. Indeed, before database systems, users had to implement programs that directly manipulate files to achieve their data processing needs. It was the observation of very similar challenges to those mentioned for smart contracts that motivated the creation of an appropriate middleware layer (databases) that provided a

We would like to acknowledge funding support from Cisco Research Inc

reasonable subset of useful, high-level abstractions (query operators) that would simplify task modeling accompanied by automatic query execution and optimization strategies that are native to databasesthis combination of features led to the enduring success that database systems have had.

A. Survey Contributions:

This paper seeks to leverage lessons learned from a very successful and related technology database systems. Before the existence of databases, users also had to write code to manipulate data in files. Such user code was also error-prone, hard to reuse, and difficult for the system to optimize execution automatically. To deal with these challenges, an effort was made to identify everyday file manipulation operations that can be made native to database systems. It further led to relational algebra and calculus, the foundations of the SQL query language, a fundamental component of data processing today. By providing a core set of primitives in the system, users can reuse them, and the system can reason about interactions/conflicts between primitive operations and possible optimization of workflows involving them. We posit that identifying some core set of transactional behavior primitives to include in blockchain engines as native transactions can yield very similar benefits. We do not suggest that all possible transactional behaviors can be captured or should be, just as SQL does not contain all possible data manipulation operations. However, by identifying and enabling a practically useful set of transaction primitives representing commonly demanded transaction behavior in blockchain applications, we may reduce the need for smart contracts by a significant percentage, making blockchains easier to use, optimize and analyze.

The above discussion raises interesting questions:

Q1 *Can techniques similar to those adopted to realize the concept of database systems be brought to bear on the design and implementation of the next generation of blockchain systems to improve their usability, robustness, and performance and their support of different application categories (just as databases do)?*

Q2 *What could the potential benefits of such primitives be? In other words, what limitations of smart contracts can be mitigated using such primitives?*

Q3 *Can analyzing real-world smart contracts provide useful insight into potential transaction primitives?*

In this paper, we offer some starting points for answering some of these questions based on surveying transactional behavior represented in present-day smart contracts. We begin by taking a closer look at the limitations of smart contracts. Then, we analyze transaction behavior patterns on a real-world blockchain, ETHEREUM. We apply some simple data reduction and abstraction techniques to the behavior patterns observed and discuss how this process might throw up transaction primitives that may be good candidates for behavior primitives to be integrated into blockchain kernels and offered to developers and users out-of-the-box.

The rest of the paper is organized as follows: Section 2 presents the background and related work, Section 3 presents the study of smart contract transactional behavior on blockchains, and Section 4 presents our conclusion and future work.

II. BACKGROUND

The methods of a smart contract generally represent transactional behavior necessary to support a blockchain or decentralized application. For example, Fig 1 shows an example smart contract for a decentralized voting application. Amongst the methods are methods to cast a vote, which is the primary transaction for this such an application. The methods must also include code to enforce all the checks and conditions necessary for validating that vote transaction. For example, the code might need to check whether the user can vote. It may also check that a user has not voted previously (i.e., preventing double voting). Real-world smart contracts also include a lot of non-application code necessary for ensuring the robustness of smart contracts, e.g., safe math code. Such non-application code is often a significantly more significant proportion of contract code.

In addition to methods, like typical programs, smart contracts have program variables. Very often, smart contract transactions manipulate assets (e.g., a vote) that are not native assets of the blockchain, like a cryptocurrency. Consequently, some program variables represent non-native assets, typically as *struct* variables. Additionally, since such assets are not native to the blockchain, the blockchain does not manage or maintain ownership relationships with such assets.

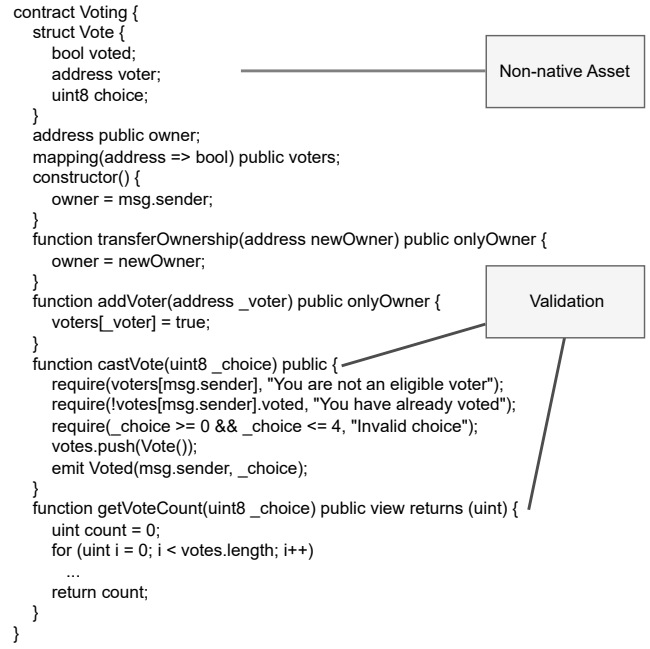

Fig. 1: Smart Contract Sample for Voting Implemented in Solidity

Therefore, *mapping* data structures (such as in Fig. 1) and associated methods are used to manage such associations.

Fig. 2 illustrates an ordered sequence of procedures delineates the lifecycle of a smart contract on a blockchain. Each discrete step signifies a distinct phase within the procedural framework, and directional indicators articulate the progression. The entities situated at the lowermost level symbolize the blocks constituting the blockchain. These blocks encapsulate transactions, spanning activities associated with the creation, deployment, and execution of the states of smart contract processes. The steps and procedures are highlighted as follows:

Creation: This phase consists of three steps. Multiple stakeholders initially discuss contractual obligations, rights, and prohibitions. The smart contract is conceptualized and designed in the first and second steps, specifying the logic, conditions, and requirements that the contract will execute. After finalizing these steps, the contractual specifications are transcribed into code during the programming phase, which is the third step. This critical stage entails the transformation of the contractual logic into executable code compatible with the blockchain. This process guarantees a precise and encoded representation of the agreement between parties within the contract framework.

Deployment: The fourth step begins with ensuring verification and integrity. Once verified, The contract is encoded and disseminated across the entire peer-to-peer (P2P) network. Each receiving node validates and temporarily stores the contract, anticipating the conclusion of the ongoing consensus round for further processing. Upon consensus, the designated node for block creation aggregates all stored contracts into a cohesive set, computes the hash value for this set and incorporates it into a new block. This newly formed block is then distributed throughout the network. Every node receiving the block must verify its data before acceptance. Through iterative exchanges, verifications, and consensus-building, all nodes in the network eventually harmonize on the disseminated contract set, which completes its distribution in the form of a block.

Execution: The operational framework of smart contracts within the blockchain undergoes periodic assessments of the state machine, transactions, and triggering conditions associated with each contract. Transactions initiated by users are the first step in activating smart contracts and triggering their programmed behaviors. Transactions aligned with predefined trigger conditions are queued for validation, awaiting consensus in step five. Conversely, those failing to meet trigger conditions persist within the blockchain. Upon receiving a transaction, a node initiates signature verification for contract validity, which is the final step. Successfully verified transactions enter a pending consensus set. Upon completion of all transactions within a contract, the state machine marks the contract as finished and then writes it to the blockchain; otherwise, it remains in progress, with new transactions stored in the most recent block. Throughout smart contract execution, a sequence of transactions, each corresponding to one or more statements in the contract, is executed and recorded in the blockchain.

A. Limitations of Smart Contracts as a Transaction Model

Skills Requirement Burden: One major limitation of the smart contract transaction model is the fact that application developers must possess skill sets beyond those needed for traditional applications, including knowledge about implementing security protocols and the burden of translating each application into correct, secure and efficient code before deploying it (smart contracts are immutable once deployed), application users bear a similar burden too!. Application users who interact with contracts must be aware. They must either completely trust the code's source or be able to check the code for themselves for errors or security vulnerabilities. They must do so to avoid putting their funds irreversibly at risk. For example, over 50% lines of code from [37] to be moved up. Due to the immutability of smart contracts once deployed, developers face significant pressure to ensure accuracy and correctness from the outset. In contrast, there is little to no burden associated with transaction types that are native because the system comes with the implementation out-of-the-box.

Related work:

Difficulty Of Discovery and Reuse: Blockchains are supposed to enable decentralized, peer-to-peer applications or *dApps*, which allow people to interact and transact on a global, permission-less, and self-executing platform without relying on a middleman that they need to trust. For example, a decentralized marketplace should be able to connect a seller of a product, e.g., a house, with a potential buyer directly without the need for some intermediaries such as aggregators like Redfin, Zillow, or even realtors. Intermediaries add fees to transactions and could act biasedly to put themselves or preferred partners at an advantage, e.g., Amazon ranking its products higher in search results [38]. Unfortunately, the current generation of blockchains does not make such peer-to-peer activity straightforward. It is not easy to search or query the blockchain to discover peer smart contracts relevant to a user's task, e.g., a relevant sales smart contract. Further, critical smart contract elements are represented as programming data structures and stored in local smart contract storage, which are not surfaced on the blockchain and cannot be easily queried. Consequently, current applications still have a reliance on the use of central entities like aggregators, e.g., NFT marketplaces like Rarible [39], OpenSea [40]. These centralized marketplaces use hybrid workflows in which most of the sales process is still executed in a proprietary off-chain process using their local databases. Application users also have to visit the websites of each aggregator separately to search their databases and learn their individual sales process, e.g., these include terms of sales, fees, and others, which is still tedious.

The issue of poor discoverability also negatively impacts contract developers because they may need to find sample smart contracts to reuse code. Limited reusability leads to repetitive effort and a proliferation of smart contract implementations. For example, we are exploring the ETHEREUM

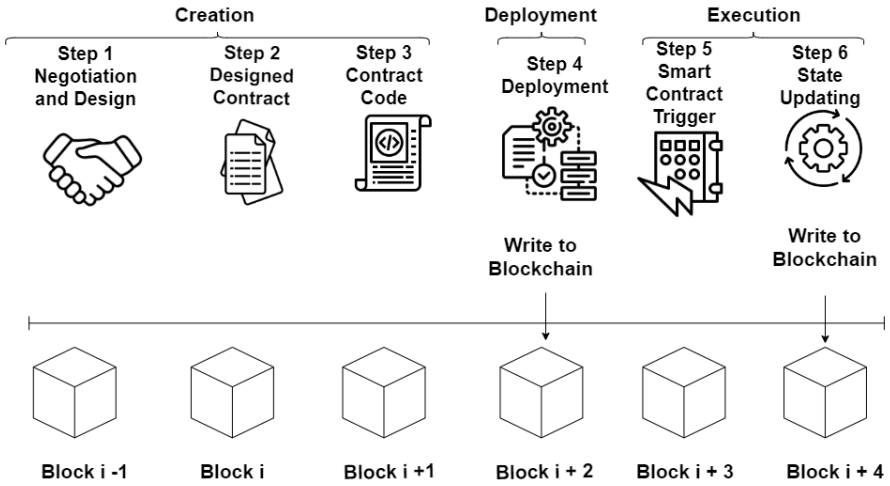

Fig. 2: Lifecycle of a Smart Contract

blocks mined between January 2021 and December 2021 and from January 2022 to December 2022. We find 16 different smart contracts with 5 slightly different implementations of a *deposit* function. Sometimes, the difference is merely due to the type of asset that is being deposited, e.g., "depositFund" [41] vs "SimpleDeposit" [42], but the side effect is the same. Similarly, a smart contract for leasing a car may have very similar functionality to one for leasing a house (perhaps with a few differences, e.g., the nature of the asset and terms) since the concept of leasing is generally similar. Even seemingly unrelated transaction behaviors may also have similar abstractions. For example, insurance claims vs. loyalty rewards claims can be viewed as *claim transactions*, claiming some benefits (either earned as a reward for the activity or by having paid for a policy). However, providing basic search functionality may not adequately support developers' needs because they may require even more advanced search capabilities when considering some problems identified concerning reusing smart contract code. [43] notes that sometimes, due to a lack of understanding of full code context, developers may lift and misuse incomplete solutions. For example, a smart contract with a *withdraw* function may be implemented with a dependency on a specific *deposit* function behavior. Consequently, copying only one of such functions may result in erroneous behavior. Another example is where a method imposes additional conditions beyond what the target context demands, perhaps implemented *function modifier* conditions, e.g., *onlyOwner* checks the permissions of the contract, whereas not having it leads to access control vulnerabilities. Such misalignment of conditions is often easy to miss and can result in problems. Even more problematic is that when reused code has errors, it makes matters worse by propagating those errors. In contrast, native transaction types are easily searchable even with existing search tools. In addition, native transactions make the issue of reuse moot.

Related work: There have been efforts to improve the overall usability of blockchains. *Blockchain search and explorer tools* such as Etherscan [44], DefiLlam [45] provides a graphical interface to aid the searching of blockchains. However, in general, the search filters provided by such tools focus more on supporting search for blocks or transactions, not necessarily deep searching for smart contracts. Some degree of reuse has been enabled by using some templates that have been published templates to simplify the development process [46]–[48]. [49] states that of the over 44 million smart contracts implemented on ETHEREUM, approx. 70% of the 15 million active contracts are duplicates of one of such templates. In another line of work, high-level declarative languages [50], [51] and domain-specific languages [52] for programming smart contracts have been proposed. Such languages may enable easier implementation of smart contracts and better-automated reasoning of smart contract behavior. However, they also introduce additional languages for developers to learn. They also tend to have limited application contexts.

Security Vulnerabilities : The issue of security vulnerabilities of smart contracts is quite topical, given the scale of financial losses that have resulted from them. Several vulnerabilities have emerged in practice, ranging from outright implementation errors to incorrect behavior (A seller can re-list an NFT without canceling the older listing). As a result, although the older one is not accessible through the UI, it is still valid. A malicious buyer can buy the NFT at a lower price, imagining the re-listed one has a higher price [53].) , to poor programming styles or patterns that largely implement correct behavior but leave room for forcing incorrect behavior (DAO attack). Other less obvious vulnerabilities do not occur explicitly inside the code of a specific method but rather allow circumvention of an intended application workflow that is typically achieved through a combination of methods. For example, *skipping royalties*: as the ERC-721 token contract does not enforce the royalty or marketplace fees, even though NFT royalty fees are incorporated into the NFT contract themselves, they are completely voluntary since there is no way to distinguish between a transfer and a sale, and only

TABLE I: Native vs. Non-native Transactions

Aspect	**Native**	**Non-native (Smart Contract)**
Creation	**N/A** (Out-of-the-box)	Contractor implements a set of **correct** and **efficient** contracts **for each** use case.
Deployment	**N/A** (Out-of-the-box)	Contractor deploys: **each** smart contracts must be deployed onto the blockchain, requiring additional steps and gas fees.
Specification	**N/A** (Out-of-the-box)	smart contracts require explicit specification of rules, functions, and data structures.
Validation	**N/A** (Out-of-the-box)	Validation depends on smart contract logic; custom rules can be applied.
Selection	**Minor** effort to select which blockchain platform to use	Contractee must evaluate source code ascertain the correctness and trustworthiness of **each** smart contract they which to engage with on **each platform**.
Transaction Speed	Typically **faster** due to lower complexity.	Potentially **slower** due to additional computations and smart contract processing.
User Experience	Simpler for basic transfers; familiar to users.	Require additional steps and an understanding of smart contract functionality.
Queryability	Transaction details can be queried directly from the blockchain.	Queries are often made through smart contract functions, providing specific information.
Migration	Transactions are native to the blockchain; migration may involve token swaps.	Migration involves deploying a new smart contract on the target blockchain and managing data/state transfer.
Dependencies	Minimal dependencies; rely on the native blockchain protocol.	Dependent on the smart contract language, libraries, and external data sources.
Gas Usage	Generally **lower** due to simplicity.	Can be **higher** due to additional computations and smart contract execution.
Security	Inherits blockchain's security features.	Relies on both blockchain and smart contract security.
Execution	Vetted system code execution on a physical machine. **fixed** financial costs for contractee	bytecode execution on a virtual machine Contractee financial costs **variable** and can spiral upwards; total **loss of funds** also possible
Storage	All state activity surfaced on the blockchain; easier to analyze	State and program variables stored within the smart contract; more complex to analyze

marketplace contracts/off-chain infrastructure pay creator royalties. Therefore, malicious sellers can transfer NFTs directly to the buyer to avoid this payment or settle the payment off-platform. In contrast, security vulnerabilities are less likely to occur in blockchain system code because such code is typically supported by a team of organizational developers and editorial processes, such as with the ETHEREUM Foundation. That rigor may be missing smart contracts implemented by arbitrary developers. Further, it is much easier to contain such vulnerabilities if they occur since they occur in a single location rather than having to deal with the large surface area of all possible smart contracts.

Related work. Techniques for smart contract code structure analytics in terms of their ast structure, or looking for known vulnerability patterns or patterns relevant to a particular kind of fraudulent activity, e.g., fraud analytics [54]–[56].

Performance : Smart contracts have an inherently more expensive execution model than native transactions. While transaction fees are fixed for native transactions, the nature of the statements determines smart contract fees, and they can have significant fees. This is in addition to the fact that fees are associated with the TRANSFER transaction that is used to trigger them. Be quite large, depending on the nature of programming statements. The smart contract execution model also leads to higher latencies for smart contracts than native transactions. A simple experiment we conducted that implemented TRANSFER behavior as a smart contract used 40% percent more resources than the native equivalent. In addition, sometimes, the virtual machine architectures used to execute smart contracts have limits that impede efficient execution. For example, the limited number of registers limits the number of smart contract function parameters; furthermore, in smart contract languages, e.g., Solidity, despite being Turing-complete, i.e., allowing the execution of arbitrary code, the arbitrary part is severely limited in practice. For example, contracts that perform heavy cryptographic operations or use machine learning models are infeasible. Thirdly, smart contracts are more challenging to parallelize because the blockchain is not privy to information about correct transaction semantics. Consequently, some platforms like ETHEREUM execute them sequentially, leading to low throughput. On the other hand, native transactions like TRANSFER are more easily parallelized since their explicit semantics within the system make concurrency conflict reasoning more effective.

Related work: The different transaction optimization techniques, such as layer 2 techniques like rollups, and off-chain, and techniques like sharding, may benefit not just

native transactions but also smart contracts. However, many techniques must still be mature enough to speak about them categorically. Furthermore, some techniques, like layer 2 use different tradeoffs with centralization and may need help to port across blockchains. Concerning parallel and concurrent execution of smart contracts, some techniques such as speculative concurrency control [57] have been proposed, offering some improvements but have yet to be significant. Results from [58] indicate that perhaps defining concurrency conflicts in terms of semantic conditions may produce better concurrency improvements than the current approach that defines conflicts merely in terms of R/W sets.

B. Other Transactional Study Efforts

There have been several research areas for transaction modeling in blockchain, one mainly focusing on sanctifying transactions. [59] proposes real-time semantic information extraction of the blockchain transactions. [51] proposes a declarative modeling framework for deploying and managing blockchain applications. The proposed framework supports a metamodel for blockchain application definition and processes to automate the deployment and management. Another track focuses on transaction processing from the tokenization perspective; [60] is a systematic study of the tokenization on the blockchain; it provides a general principle of tokenizing digital asset categorization. There have been surveys to characterize transactions to capture fraudulent activity on the blockchain, [61], [62] Another field of study focuses on improving transaction performance, [63], [64] characterize layer2 solutions and their privacy consideration.

[65] is a systematic survey on the decentralized applications ecosystem, its primitives, protocol types, and security.

For transaction modeling in blockchain, there have been several research directions. One track mainly focuses on semantifying transactions; [59] proposes real-time semantic information extraction of blockchain transactions, aiming to enhance the understanding of transaction semantics in real-time. [51] introduces a declarative modeling framework for deploying and managing blockchain applications, supporting a metamodel for blockchain application definition and processes to automate deployment and management. Another track concentrates on transaction processing from the tokenization perspective; [60] presents a systematic study of tokenization on the blockchain, providing a general principle for categorizing digital assets through tokenization. Surveys have been conducted to characterize transactions for capturing fraudulent activity on the blockchain [61], [62]. Another field of study focuses on improving transaction performance; [63], [64] characterize layer2 solutions and address privacy considerations in their works. Additionally, [65] offers a systematic survey on the decentralized applications ecosystem, covering primitives, protocol types, and security considerations.

III. A Study of Smart Contract Transactional Behavior on Real-World Blockchains

This section assesses transactional activity attributed to smart contracts deployed on real-world blockchains. The objective is to try and *identify transaction primitives that may be candidate transaction building blocks* that, if offered as native, out-of-the-box transaction types, can reduce the negative impact of smart contracts and improve the usability of blockchains. We focus on the Ethereum's blockchain as a representative use case, the largest blockchain supporting smart contracts. There are tens of millions of smart contracts deployed on Ethereum, each with a set of methods, some of which correspond to transactional activity. Consequently, an exhaustive and manual inspection of contract source codes is impractical. Therefore, we selected two data perspectives that would be more practical yet equally insightful. *First*, by analyzing smart contract method call data on the blockchain, we can identify not just what smart contract invocations exist but also the frequency distribution of such calls. *Second*, by aggregating information from research and industry publications about their smart contracts, we can learn not just the low-level transaction primitives but also *transaction workflows* that indicate groups or sequences of methods that work together. The publications also include discussions that provide a more complete application context.

A. Analyzing Ethereum Smart Contract Call Data

1) Data Gathering: The data collection phase was conducted on a cluster of four virtual machines running Ubuntu 22.04, each equipped with 16 cores, 100GB disk capacity, and 128GB RAM, situated at North Carolina State University. The transaction dataset was sourced from the Ethereum blockchain via Etherscan, encompassing transaction records spanning January 2021 to December 2021 and from January 2022 to December 2022. The CSV files were acquired and consolidated, incorporating various data fields, including *address_from*, *address_to*, *block_number*, *block_hash*, *gas*, *gas_price*, *input*, *is_create_contract*, *method*, *nonce*, *timestamp*, *transaction_hash*, *transaction_index*, *value*, *token_id*, *token_symbol*, and *transaction_hash*. Additionally, the Ethereum signature database [66] was crawled to obtain a mapping featuring over 1,331,539 signature mappings for *byte signature* and *text signature* in JSON format. The initial ten characters of the input code were programmatically employed to decode the method label of the smart contract. The total number of records in the gathered transaction dataset was 122M transactions with a size of 93.1 GB. Fig. 3 shows a word cloud of the text labels associated with smart contract calls based on their usage frequencies.

2) Data Preparation: We observe that the label of a large number of smart contract calls are *unknown* i.e., byte_code prefix is `0x`. Therefore, we filtered such tuples with the remaining 79M records left for analysis. Then, we loaded a JSON file containing a dictionary mapping smart contract call bytes_signatures to text labels. We leveraged the dictionary to assign labels to the smart contract call bytes_signatures

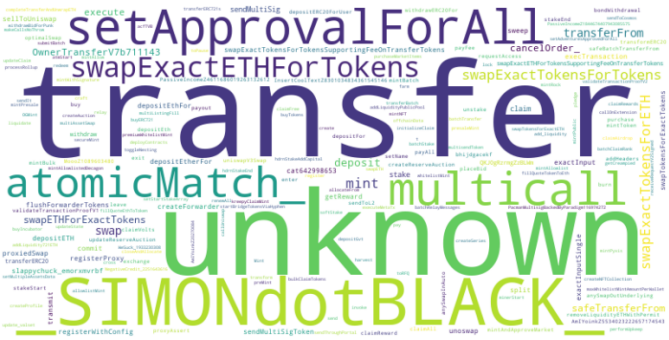

Fig. 3: Word Cloud for Smart Contract Call Labels (all)

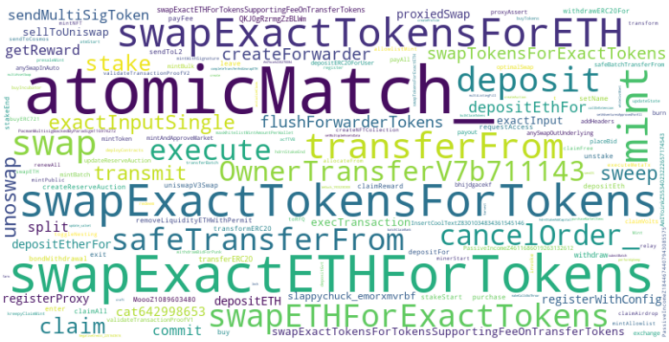

Fig. 4: Word Cloud for Reduced Set of Labels

in our dataset. This step was crucial for decoding method calls in ETHEREUM transactions. We employed a conditional approach: for each bytes_signature, we searched for a matching label in the JSON data. If found, we updated the dataset with this label. In this way, for example, the bytes_signature, `0xa9059cbb` was mapped to a TRANSFER label.

To gain a clearer picture of some of the dominant transaction call labels, we filter out the less frequent transaction call types using frequency count $\geq$1000 as a threshold. The result is the word cloud in Fig. 3. The next step in our analysis was to improve the differentiation of labels by filtering out some disproportionately frequent ones using some reasonable criteria. For example, the TRANSFER transaction is already a native transaction and so is not of interest here. A closer look at some of the smart contract transaction calls shows that there are at least two categories of transaction calls. First, there are transaction calls that are performing some application transactions. Then, there are "helper" transactions that are more *operational* to support some execution models not directly supported in ETHEREUM. An example is the method *sendMultiSig*, a special contract enabling multiple entities to sign a transaction since ETHEREUM transactions can only be signed by a single entity by default. A similar one is *setApprovalForAll*.

A third example is *multicall*, which is a library call that enables multiple smart contract constant function calls to be grouped into a single call, and the results aggregated into a single result, resulting in a reduced number of separate JSON RPC requests that need to be sent over the network. We manually identify a few such *operational* transaction types and drop them from the dataset. The remaining list is a set of 2451 unique labels. Fig. 4 shows a word cloud with this reduced set of labels.

Our observation shows similar labels; for example, *swapExactTokensForTokens* and *swapExactETHForTokens*. To understand if these 2451 labels represented distinct transaction behaviors or conducted an analysis based on label similarity. We exploited the observation that much of the label similarity present was in the label prefixes. We, therefore, clustered the labels into groups based on common prefixes, using prefixes of length = 4 as a similarity measure. That length was selected as a suitable threshold after sorting the labels alphabetically and manually observing patterns. This clustering produced 598 clusters from 2451 labels. The frequency distribution of the 20 largest clusters (based on the number of labels per cluster) is shown in Fig. 6. The frequency distribution of the 20 largest clusters (based on the total number of actual transactions, i.e., transactions of all labels per cluster) is shown in Fig. 5. Very often (not always), such similar transaction methods have similar labels, and this label similarity is reflected in the label prefix.

3) A Closer Look At Transaction Clusters: To understand how meaningful the transaction clusters might be, we analyzed one of the clusters in more depth. The analysis was done by manually looking at the source code of the smart contracts in the cluster, which is precisely the method with the same name as the cluster label. Since it was impractical to do this with all clusters manually, we selected one cluster (prefix label - "*depo*" to perform the analysis. The cluster was selected because the application context did not require special domain knowledge to understand its transactions. The cluster comprised of labels such as *deposit*, *depositEthFor*, *depositEthFor*, *depositEtherFor* and so on corresponding to transactional activity around making deposits.

Table II shows the list of labels in that cluster. Our manual analysis of a subset of the smart contracts in this cluster validated some similarities in transaction behavior across labels. We could abstract the analyzed subset into a generalized structure as shown in Fig. 7. This structure represents the cluster in terms of some core elements that are significantly expected: inputs - such as the type of asset being deposited and value; conditions - any conditions associated with deposits and, ultimately, side effects. (We ignore the important problem of developing a formal representation for a more precise representation of such a generalization.) We found a few categories of conditions: time-based conditions like deadlines, authorization-based, and threshold-based, e.g., minimum deposit requirements. Side-effects capture some anticipated state changes, e.g. the program variables representing balances will be updated and stored back in smart contract storage at the end of execution.

TABLE II: Prefix-based Aggregation of Deposit Transactions

Cluster Prefix	Transactions in the Cluster
depo	deposit, depositEthFor, depositEtherFor, depositETH, depositEth, depositFor, depositERC20ForUser, depositGvt, depositPwrd, depositETH2, depositSingle, depositToken, depositTempleFor, deposit_and_stake, depositFromCrv, depositNFTs, depositWithSignature, deposit2, depositAndBridgeOut, depositFromEth, depositWithPermit, depositTokens, depositAndBorrow, depositFromCvxCrv, depositGameKey, depositToCurveAndStake, depositWithoutLP, depositAlloyxDURATokens, depositToCurve, depositJobCredits, depositToTruefi, depositUsdt, depositToMaple, depositRequest, depositForUser, depositAtMaturity

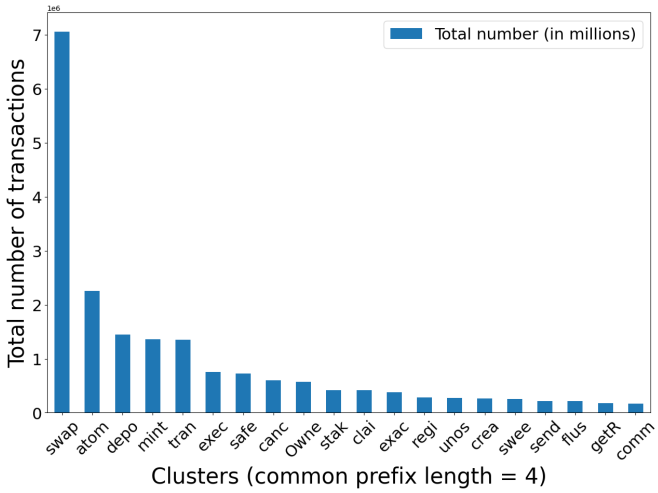

Fig. 5: Total Number of Transactions per Cluster

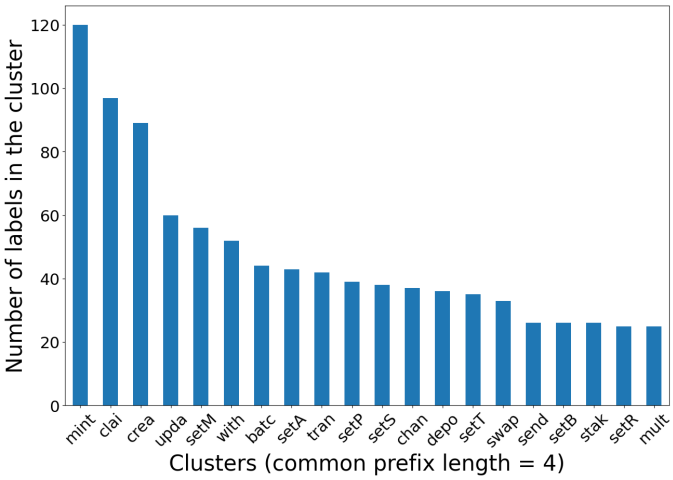

Fig. 6: Frequency Distribution of Top 20 Clusters

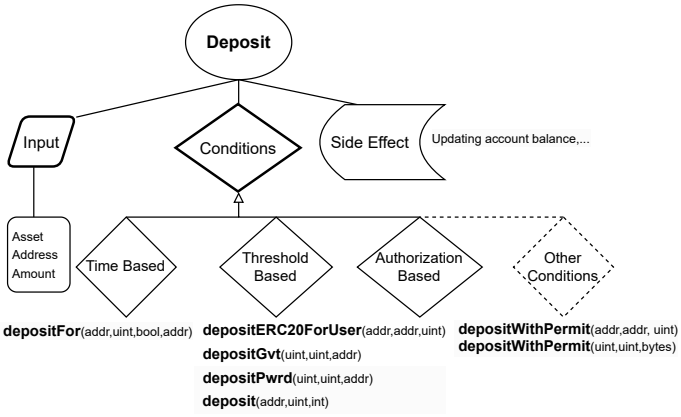

Fig. 7: Generalized Representation of Transactions in the "Depo" Cluster

B. Patterns of Smart Contract Transaction Usage in Applications

We also wanted to study smart contract transaction usage from the perspective of application contexts. Research and industry publications on blockchains often discuss their application implementation architecture. We focused on trying to find such papers. We searched in Google for blockchain application papers using search terms that involved the combination of the keywords *blockchain*, *smart contracts* and the name of some domain such as *agriculture*. From the search results, we selected for inclusion in our analysis 43 papers from across 12 domains ranging from energy trading, agriculture trading, digital asset trading, auction marketplaces, car-sharing and leasing, real estate, insurance, loyalty programs, product provenance tracking, gig-economy and so on. The goal was to see if we could gain additional insight into smart contract transaction behaviour in practice. Table I shows a mapping of application context to a set of smart contract transactions as reported by the corresponding publication.

Discussion: The benefit of the above analysis is that it suggests the possibility of reducing multiple smart contract transactions into generic primitives with extensibility elements such as the "conditions" component. Such primitives can then be implemented inside the blockchain and offered as transaction types out-of-box, eliminating the need to implement them as smart contracts. A good analogy is database query operators such as a JOIN operator, which can act differently based on join conditions (e.g., an equijoin vs a theta join). It is unnecessary to capture all possible transaction behavior types, just like SQL does not contain all possible data processing operators. However, identifying a reasonable subset that will have broad applicability could have a significant impact. A couple of other efforts have also suggested blockchain transaction modeling or abstractions. [107] proposes ontological definitions for some

TABLE III: Smart Contract Transactions and their Application Contexts

References	Area	Proposed Transactions
[67]–[69]	Marketplace	Advertise, Bid, Discovering Matching Bid
[70], [71]	Marketplace Auction	Create Auction, Bid, Reveal, Withdraw(Refund/Claim)
[72]–[74]	Freelance marketplace	Create, Bid, Create Project Contract, Project Owner Deposit Payment, Canceling Contract Agreement, Store/Sharing file, Withdraw
[75]	Crowdsourcing	Register, Post Task, Receive Task
[76], [77]	Energy Market	Create, Bid, Transfer, Redeem
[78], [79]	Agriculture Marketplace	Create, Purchase, Delivery, Rating
[80]	Proof of Delivery of Digital Asset	Request, Payment, Dispute, Revert
[81]	Energy Demand Marketplace	Bid, Return
[82], [83]	Real Estate	Property Ownership, Title Transfer
[11], [84], [85]	Real Estate	Create, Refund, PayRent, WithdrawRent
[86]	Agriculture Production Trading Marketplace	Bid, Return
[87]	Shared Bicycle Deposit Management System	Create, Transfer, Withdraw
[88], [89]	Car Registration System	Change Ownership, Create Lease, Cancel Lease, Confirm Lease
[90]–[96]	Loyalty Program	Create, Redeem, Transfer
[97]	Insurance	Registration, Issue of policy, Claiming of Policy
[16]	Insurance	Registration, Issue policy, Claim, Refund
[98]	Insurance	Insurance policy processing, Claim handling, Payment
[99]–[105]	Supply Chain	Trace
[99]	Egg Distribution	track products from farm to fork
[106]	Supply Chain	Register, Add Entity, Purchase

marketplace transactions on blockchains modeled using the Web Ontology Language (OWL). The authors also gave an insight into how these primitives may be integrated into a blockchain's modeling and processing architecture. [59] analyzed patterns of TRANSFER transaction workflows and abstracted different workflow shapes as categories.

The publication view gives a fuller context as they expose not just individual transaction types but groups (often sequences) of dependent transaction types. We also observe a similar trend of label similarity in this context as labels appear to reoccur across application contexts. Some labels from this portion of the study also overlap with some of the prefix-induced clusters in the earlier discussion e.g. the *create* label and the *crea* cluster, further validating the promise of such transactions as candidate native transactions for blockchains.

IV. CONCLUSION AND FUTURE WORK

The paper surveys and analyzes transactional behaviors encoded in smart contracts deployed on a popular blockchain platform. The analysis gives some insights that might be useful in determining possible candidates for adoption as native transaction types in blockchains.

REFERENCES

[1] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov, "A decentralized marketplace application on the ethereum blockchain," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2018, pp. 90–97.

[2] U. K. Shakila and S. Sultana, "A decentralized marketplace application based on ethereum smart contract," in *2021 24th International Conference on Computer and Information Technology (ICCIT)*. IEEE, 2021, pp. 1–5.

[3] D. Macrinici, C. Cartofeanu, and S. Gao, "Smart contract applications within blockchain technology: A systematic mapping study," *Telematics and Informatics*, vol. 35, no. 8, pp. 2337–2354, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736585318308013

[4] A. Avyukt, G. Ramachandran, and B. Krishnamachari, "A decentralized review system for data marketplaces," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2021, pp. 1–9.

[5] X. Zhou, M. Q. Lim, and M. Kraft, "A smart contract-based agent marketplace for the j-park simulator-a knowledge graph for the process industry," *Computers & Chemical Engineering*, vol. 139, p. 106896, 2020.

[6] C. F. Durach, T. Blesik, M. von Düring, and M. Bick, "Blockchain applications in supply chain transactions," *Journal of Business Logistics*, vol. 42, no. 1, pp. 7–24, 2021.

[7] P. De Giovanni, "Blockchain and smart contracts in supply chain management: A game theoretic model," *International Journal of Production Economics*, vol. 228, p. 107855, 2020.

[8] G. Prause, "Smart contracts for smart supply chains," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 2501–2506, 2019.

[9] S. Wang, D. Li, Y. Zhang, and J. Chen, "Smart contract-based product traceability system in the supply chain scenario," *IEEE Access*, vol. 7, pp. 115 122–115 133, 2019.

[10] W. Groschopf, M. Dobrovnik, and C. Herneth, "Smart contracts for sustainable supply chain management: Conceptual frameworks for supply chain maturity evaluation and smart contract sustainability assessment," *Frontiers in Blockchain*, vol. 4, p. 506436, 2021.

[11] I. Karamitsos, M. Papadaki, N. B. Al Barghuthi *et al.*, "Design of the blockchain smart contract: A use case for real estate," *Journal of Information Security*, vol. 9, no. 03, p. 177, 2018.

[12] B. K. Mohanta, S. S. Panda, and D. Jena, "An overview of smart contract and use cases in blockchain technology," in *2018 9th international conference on computing, communication and networking technologies (ICCCNT)*. IEEE, 2018, pp. 1–4.

[13] F. Ullah and F. Al-Turjman, "A conceptual framework for blockchain smart contract adoption to manage real estate deals in smart cities," *Neural Computing and Applications*, vol. 35, no. 7, pp. 5033–5054, 2023.

[14] S. R. Niya, F. Schüpfer, T. Bocek, and B. Stiller, "A peer-to-peer purchase and rental smart contract-based application (pursca)," *it-Information Technology*, vol. 60, no. 5-6, pp. 307–320, 2018.

[15] M. Raikwar, S. Mazumdar, S. Ruj, S. S. Gupta, A. Chattopadhyay, and K.-Y. Lam, "A blockchain framework for insurance processes," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2018, pp. 1–4.

[16] A. Hassan, M. I. Ali, R. Ahammed, M. M. Khan, N. Alsufyani, and A. Alsufyani, "Secured insurance framework using blockchain and smart contract," *Scientific Programming*, vol. 2021, pp. 1–11, 2021.

[17] L. Bader, J. C. Bürger, R. Matzutt, and K. Wehrle, "Smart contract-based car insurance policies," in *2018 IEEE Globecom workshops (GC wkshps)*. IEEE, 2018, pp. 1–7.

[18] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda, and V. Santamaría, "Blockchain and smart contracts for insurance: Is the technology mature enough?" *Future internet*, vol. 10, no. 2, p. 20, 2018.

[19] A. Sheth and H. Subramanian, "Blockchain and contract theory: modeling smart contracts using insurance markets," *Managerial finance*, vol. 46, no. 6, pp. 803–814, 2020.

[20] A. Alammary, S. Alhazmi, M. Almasri, and S. Gillani, "Blockchain-based applications in education: A systematic review," *Applied Sciences*, vol. 9, no. 12, p. 2400, 2019.

[21] H. Sun, X. Wang, and X. Wang, "Application of blockchain technology in online education." *International Journal of Emerging Technologies in Learning*, vol. 13, no. 10, 2018.

[22] D. Rustiana, D. Ramadhan, L. A. Wibowo, and A. W. Nugroho, "State of the art blockchain enabled smart contract applications in the university," *Blockchain Frontier Technology*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254308662

[23] D. eke and S. Kunosi, "Smart contracts as a diploma anti-forgery system in higher education - a pilot project," in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2020, pp. 1662–1667.

[24] A. Jain, A. Kumar Tripathi, N. Chandra, and P. Chinnasamy, "Smart contract enabled online examination system based in blockchain network," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, 2021, pp. 1–7.

[25] F. R. Batubara, J. Ubacht, and M. Janssen, "Challenges of blockchain technology adoption for e-government: a systematic literature review," in *Proceedings of the 19th annual international conference on digital government research: governance in the data age*, 2018, pp. 1–9.

[26] M. El Khatib, A. Al Mulla, and W. Al Ketbi, "The role of blockchain in e-governance and decision-making in project and program management," *Advances in Internet of Things*, vol. 12, no. 3, pp. 88–109, 2022.

[27] M. Stefanovi, . Prulj, S. Risti, D. Stefanovi, and D. Nikoli, "Smart contract application for managing land administration system transactions," *IEEE Access*, vol. 10, pp. 39 154–39 176, 2022.

[28] A. Khanna, A. Sah, V. Bolshev, M. Jasinski, A. Vinogradov, Z. Leonowicz, and M. Jasiński, "Blockchain: Future of e-governance in smart cities," *Sustainability*, vol. 13, no. 21, p. 11840, 2021.

[29] R. Adeodato and S. Pournouri, "Secure implementation of e-governance: A case study about estonia," *Cyber Defence in the Age of AI, Smart Societies and Augmented Humanity*, pp. 397–429, 2020.

[30] N. Fotiou, V. A. Siris, and G. C. Polyzos, "Interacting with the internet of things using smart contracts and blockchain technologies," in *Security, Privacy, and Anonymity in Computation, Communication, and Storage: 11th International Conference and Satellite Workshops, SpaCCS 2018, Melbourne, NSW, Australia, December 11-13, 2018, Proceedings 11*. Springer, 2018, pp. 443–452.

[31] A. Rashid and M. J. Siddique, "Smart contracts integration between blockchain and internet of things: Opportunities and challenges," in *2019 2nd International Conference on Advancements in Computational Sciences (ICACS)*. IEEE, 2019, pp. 1–9.

[32] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2018.

[33] L. Wei, J. Wu, and C. Long, "Blockchain-enabled trust management in service-oriented internet of things: Opportunities and challenges," in *2021 The 3rd International Conference on Blockchain Technology*, ser. ICBCT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 9095. [Online]. Available: https://doi.org/10.1145/3460537.3460544

[34] H. Baqa, N. B. Truong, N. Crespi, G. M. Lee, and F. Le Gall, "Semantic smart contracts for blockchain-based services in the internet of things," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2019, pp. 1–5.

[35] D. D. F. Maesa and P. Mori, "Blockchain 3.0 applications survey," *Journal of Parallel and Distributed Computing*, vol. 138, pp. 99–114, 2020.

[36] M. I. Mehar, C. L. Shier, A. Giambattista, E. Gong, G. Fletcher, R. Sanayhie, H. M. Kim, and M. Laskowski, "Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack," *Journal of Cases on Information Technology (JCIT)*, vol. 21, no. 1, pp. 19–32, 2019.

[37] "Wyvernexchangewithbulkcancellations," https://etherscan.io/address/0x7f268357a8c2552623316e2562d90e642bb538e5#code, 2023.

[38] "reuters," https://www.reuters.com/investigates/special-report/amazon-india-rigging/, 2023.

[39] "Rarible," https://rarible.com/, 2023.

[40] "Oepnsea," https://opensea.io/, 2023.

[41] "depositfund transaction," https://etherscan.io/address/0x24fc02900ad7ce5ff0a43fecff10f2a84d1f3876, 2023.

[42] "Simpledeposit transaction," https://github.com/azuredevil0818/solidity_patterns?tab=readme-ov-file, 2023.

[43] Q. Huang, D. Liao, Z. Xing, Z. Zuo, C. Wang, and X. Xia, "Semantic-enriched code knowledge graph to reveal unknowns in smart contract code reuse," *ACM Transactions on Software Engineering and Methodology*, 2023.

[44] "etherscan," https://etherscan.io/, 2023.

[45] "defillama," https://defillama.com/, 2023.

[46] J. Park, S. Jeong, and K. Yeom, "Smart contract broker: Improving smart contract reusability in a blockchain environment," *Sensors*, vol. 23, no. 13, p. 6149, 2023.

[47] X. Chen, P. Liao, Y. Zhang, Y. Huang, and Z. Zheng, "Understanding code reuse in smart contracts," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021, pp. 470–479.

[48] L. Guida and F. Daniel, "Supporting reuse of smart contracts through service orientation and assisted development," in *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*. IEEE, 2019, pp. 59–68.

[49] "bitkan:," https://bitkan.com/learn/how-many-smart-contracts-on-ethereum-how-do-ethereum-smart-contracts-work-3982, 2023.

[50] H. Chen, G. Whitters, M. J. Amiri, Y. Wang, and B. T. Loo, "Declarative smart contracts," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 281–293.

[51] L. Baresi, G. Quattrocchi, D. A. Tamburri, and L. Terracciano, "A declarative modelling framework for the deployment and management of blockchain applications," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, 2022, pp. 311–321.

[52] M. Wöhrer and U. Zdun, "Domain specific language for smart contract development," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–9.

[53] "Opensea fraud investigation," https://www.zdnet.com/article/opensea-reimbursing-people-affected-by-bug-used-to-purchase-nfts-below-market-value/, 2023.

[54] A. Ghaleb and K. Pattabiraman, "How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 415–427.

[55] J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2019, pp. 8–15.

[56] R. Tonelli, G. A. Pierro, M. Ortu, and G. Destefanis, "Smart contracts software metrics: A first study," *Plos one*, vol. 18, no. 4, p. e0281043, 2023.

[57] V. Saraph and M. Herlihy, "An empirical study of speculative concurrency in ethereum smart contracts," *arXiv preprint arXiv:1901.01376*, 2019.

[58] P. S. Anjana, H. Attiya, S. Kumari, S. Peri, and A. Somani, "Efficient concurrent execution of smart contracts in blockchains using object-based transactional memory," in *Networked Systems: 8th International Conference, NETYS 2020, Marrakech, Morocco, June 3–5, 2020, Proceedings 8*. Springer, 2021, pp. 77–93.

[59] Z. Wu, J. Liu, J. Wu, Z. Zheng, X. Luo, and T. Chen, "Know your transactions: Real-time and generic transaction semantic representation on blockchain & web3 ecosystem," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 1918–1927.

[60] G. Wang and M. Nixon, "Sok: Tokenization on blockchain," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, 2021, pp. 1–9.

[61] J. Nicholls, A. Kuppa, and N.-A. Le-Khac, "Sok: The next phase of identifying illicit activity in bitcoin," in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2023, pp. 1–10.

[62] K. Kolachala, E. Simsek, M. Ababneh, and R. Vishwanathan, "Sok: money laundering in cryptocurrencies," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021, pp. 1–10.

[63] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 2020, pp. 201–226.

[64] ——, "Sok: Off the chain transactions." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 360, 2019.

[65] S. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. Knottenbelt, "Sok: Decentralized finance (defi)," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 30–46.

[66] "4byte.directory," https://www.4byte.directory/, 2023.

[67] A. Merlina, R. Vitenberg, and V. Setty, "A general and configurable framework for blockchain-based marketplaces," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 216–225.

[68] S. Bajoudah, C. Dong, and P. Missier, "Toward a decentralized, trustless marketplace for brokered iot data trading using blockchain," in *2019 IEEE international conference on blockchain (Blockchain)*. IEEE, 2019, pp. 339–346.

[69] B. M. Yakubu, M. I. Khan, N. Javaid, and A. Khan, "Blockchain-based secure multi-resource trading model for smart marketplace," *Computing*, vol. 103, pp. 379–400, 2021.

[70] C. Pop, M. Prata, M. Antal, T. Cioara, I. Anghel, and I. Salomie, "An ethereum-based implementation of english, dutch and first-price sealed-bid auctions," in *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE, 2020, pp. 491–497.

[71] I. A. Omar, H. R. Hasan, R. Jayaraman, K. Salah, and M. Omar, "Implementing decentralized auctions using blockchain smart contracts," *Technological Forecasting and Social Change*, vol. 168, p. 120786, 2021.

[72] I. Afrianto, C. R. Moa, S. Atin, I. Rosyidin *et al.*, "Prototype blockchain based smart contract for freelance marketplace system," in *2021 Sixth International Conference on Informatics and Computing (ICIC)*. IEEE, 2021, pp. 1–8.

[73] P. Deshmukh, S. Kalwaghe, A. Appa, and A. Pawar, "Decentralised freelancing using ethereum blockchain," in *2020 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2020, pp. 881–883.

[74] M. Gandhi, P. Shah, D. Solanki, and M. Shah, "Decentralized freelancing system-trust and transparency," *International Research Journal of Engineering and Technology (IRJET)*, vol. 6, no. 09, 2019.

[75] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, 2018.

[76] N. Karandikar, A. Chakravorty, and C. Rong, "Blockchain based transaction system with fungible and non-fungible tokens for a community-based energy infrastructure," *Sensors*, vol. 21, no. 11, p. 3822, 2021.

[77] U. Damisa, N. I. Nwulu, and P. Siano, "Towards blockchain-based energy trading: A smart contract implementation of energy double auction and spinning reserve trading," *Energies*, vol. 15, no. 11, p. 4084, 2022.

[78] G. Leduc, S. Kubler, and J.-P. Georges, "Innovative blockchain-based farming marketplace and smart contract performance evaluation," *Journal of Cleaner Production*, vol. 306, p. 127055, 2021.

[79] M. Tarhini *et al.*, "Application of asset tokenization smart contracts and decentralized finance in agriculture [j]," *Journal of Financial Studies*, vol. 10, no. 6, pp. 152–163, 2021.

[80] H. R. Hasan and K. Salah, "Proof of delivery of digital assets using blockchain and smart contracts," *IEEE Access*, vol. 6, pp. 65 439–65 448, 2018.

[81] V. Deshpande, L. George, H. Badis, and A. A. Desta, "Blockchain based decentralized framework for energy demand response marketplace," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[82] "landtitle:," https://veridocglobal.medium.com/blockchain-and-land-title-registration-fighting-corruption-and-inefficiency-one-block-2023.

[83] A. Mizrahi, "A blockchain-based property ownership recording system," *A blockchain-based property ownership recording System*, 2015.

[84] S. Gatt and F. Inguanez, "Use of blockchain technology in automation of ad-hoc leasing agreements," in *2021 IEEE 11th International Conference on Consumer Electronics (ICCE-Berlin)*. IEEE, 2021, pp. 1–6.

[85] A. Gupta, J. Rathod, D. Patel, J. Bothra, S. Shanbhag, and T. Bhalerao, "Tokenization of real estate using blockchain technology," in *Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings 18*. Springer, 2020, pp. 77–90.

[86] S. Ahmed, M. K. Gurve, and S. Shukla, "Easy mandi-an agricultural production trading platform for indian markets," in *2022 8th International Conference on Signal Processing and Communication (ICSC)*. IEEE, 2022, pp. 112–116.

[87] D. Zhao, D. Wang, and B. Wang, "Research on a shared bicycle deposit management system based on blockchain technology," *Journal of advanced transportation*, vol. 2020, pp. 1–14, 2020.

[88] T. Rosado, A. Vasconcelos, and M. Correia, "A blockchain use case for car registration," in *Essentials of Blockchain Technology*. Chapman and Hall/CRC, 2019, pp. 205–234.

[89] S. Auer, S. Nagler, S. Mazumdar, and R. R. Mukkamala, "Towards blockchain-iot based shared mobility: Car-sharing and leasing as a case study," *Journal of Network and Computer Applications*, vol. 200, p. 103316, 2022.
[90] M. Agrawal, D. Amin, H. Dalvi, and R. Gala, "Blockchain-based universal loyalty platform," in *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*. IEEE, 2019, pp. 1–6.
[91] "loyyal:," https://loyyal.com/, 2023.
[92] "qiibee:," https://www.qiibee.com/, 2023.
[93] "adper:," https://advertise.permission.io/, 2023.
[94] "digitalbits:," https://digitalbits.io/, 2023.
[95] Ş. Bülbül and G. İnce, "Blockchain-based framework for customer loyalty program," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2018, pp. 342–346.
[96] O. Sönmeztürk, T. Ayav, and Y. M. Erten, "Loyalty program using blockchain," in *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 509–516.
[97] N. Jha, D. Prashar, O. I. Khalaf, Y. Alotaibi, A. Alsufyani, and S. Alghamdi, "Blockchain based crop insurance: a decentralized insurance system for modernization of indian farmers," *Sustainability*, vol. 13, no. 16, p. 8921, 2021.
[98] F. Loukil, K. Boukadi, R. Hussain, and M. Abed, "Ciosy: A collaborative blockchain-based insurance system," *Electronics*, vol. 10, no. 11, p. 1343, 2021.
[99] D. Bumblauskas, A. Mann, B. Dugan, and J. Rittmer, "A blockchain use case in food distribution: Do you know where your food has been?" *International Journal of Information Management*, vol. 52, p. 102008, 2020.
[100] P. Dutta, T.-M. Choi, S. Somani, and R. Butala, "Blockchain technology in supply chain operations: Applications, challenges and research opportunities," *Transportation research part e: Logistics and transportation review*, vol. 142, p. 102067, 2020.
[101] T. Sund, C. Lööf, S. Nadjm-Tehrani, and M. Asplund, "Blockchain-based event processing in supply chainsa case study at ikea," *Robotics and Computer-Integrated Manufacturing*, vol. 65, p. 101971, 2020.
[102] T. K. Dasaklis, T. G. Voutsinas, G. T. Tsoulfas, and F. Casino, "A systematic literature review of blockchain-enabled supply chain traceability implementations," *Sustainability*, vol. 14, no. 4, p. 2439, 2022.
[103] M. Westerkamp, F. Victor, and A. Küpper, "Tracing manufacturing processes using blockchain-based token compositions," *Digital Communications and Networks*, vol. 6, no. 2, pp. 167–176, 2020.
[104] I. Ehsan, M. Irfan Khalid, L. Ricci, J. Iqbal, A. Alabrah, S. Sajid Ullah, and T. M. Alfakih, "A conceptual model for blockchain-based agriculture food supply chain system," *Scientific Programming*, vol. 2022, pp. 1–15, 2022.
[105] G. Baralla, A. Pinna, R. Tonelli, M. Marchesi, and S. Ibba, "Ensuring transparency and traceability of food local products: A blockchain application to a smart tourism region," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 1, p. e5857, 2021.
[106] A. Shahid, A. Almogren, N. Javaid, F. A. Al-Zahrani, M. Zuair, and M. Alam, "Blockchain-based agri-food supply chain: A complete solution," *Ieee Access*, vol. 8, pp. 69 230–69 243, 2020.
[107] S. Mansouri, V. Samatova, N. Korchiev, and K. Anyanwu, "Demato: An ontology for modeling transactional behavior in decentralized marketplaces," 2023 (To Appear).