Why Does My Transaction Fail? A First Look at Failed Transactions on the Solana Blockchain*

XIAOYE ZHENG, Zhejiang University, China
ZHIYUAN WAN†‡, Zhejiang University, China
DAVID LO, Singapore Management University, Singapore
DIFAN XIE, Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, China
XIAOHU YANG, Zhejiang University, China

Solana is an emerging blockchain platform, recognized for its high throughput and low transaction costs, positioning it as a preferred infrastructure for Decentralized Finance (DeFi), Non-Fungible Tokens (NFTs), and other Web 3.0 applications. In the Solana ecosystem, transaction initiators submit various instructions to interact with a diverse range of Solana smart contracts, among which are decentralized exchanges (DEXs) that utilize automated market makers (AMMs), allowing users to trade cryptocurrencies directly on the blockchain without the need for intermediaries. Despite the high throughput and low transaction costs of Solana, the advantages have exposed Solana to bot spamming for financial exploitation, resulting in the prevalence of failed transactions and network congestion.

Prior work on Solana has mainly focused on the evaluation of the performance of the Solana blockchain, particularly scalability and transaction throughput, as well as on the improvement of smart contract security, leaving a gap in understanding the characteristics and implications of failed transactions on Solana. To address this gap, we conducted a large-scale empirical study of failed transactions on Solana, using a curated dataset of over 1.5 billion failed transactions across more than 72 million blocks. Specifically, we first characterized the failed transactions in terms of their initiators, failure-triggering programs, and temporal patterns, and compared their block positions and transaction costs with those of successful transactions. We then categorized the failed transactions by the error messages in their error logs, and investigated how specific programs and transaction initiators are associated with these errors.

We find that transaction failure rates on Solana exhibit recurring daily patterns, and demonstrate a strong positive correlation with the volume of failed transactions, with bots on Solana experiencing a high transaction failure rate of 58.43%. We identify ten distinct error types in the error logs of failed transactions, with *price or profit not met* and *invalid status* errors accounting for 67.18% of all failed transactions. AMMs primarily experience *invalid status* errors among failed transactions, while DEX aggregators are more commonly affected by *price or profit not met* errors. Among transaction initiators, bots encounter a broader range of errors due to their high-frequency trading and complex interactions with smart contracts. In contrast, human users

*This research was supported by the National Science Foundation of China (No. 62472383 and No. 62102358), and the Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University.
†Zhiyuan Wan is the corresponding author.
‡Also with Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security.

Authors' Contact Information: Xiaoye Zheng, Zhejiang University, The State Key Laboratory of Blockchain and Data Security, Hangzhou, China, xiaoyez@zju.edu.cn; Zhiyuan Wan, Zhejiang University, The State Key Laboratory of Blockchain and Data Security, Hangzhou, China, wanzhiyuan@zju.edu.cn; David Lo, Singapore Management University, School of Computing and Information Systems, Singapore, Singapore, davidlo@smu.edu.sg; Difan Xie, Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, Hangzhou, China, xiedifan@bcds.org.cn; Xiaohu Yang, Zhejiang University, The State Key Laboratory of Blockchain and Data Security, Hangzhou, China, yangxh@zju.edu.cn.

ACM 2994-970X/2025/7-ARTISSTA066
https://doi.org/10.1145/3728943

experience a more limited range of errors. Based on our findings, we provide recommendations to mitigate transaction failures on Solana and outline future research directions.

CCS Concepts: • **Software and its engineering → Software post-development issues**; • **General and reference → Empirical studies**.

Additional Key Words and Phrases: Solana, failed transaction, blockchain ecosystem, DeFi, bot

ACM Reference Format:
Xiaoye Zheng, Zhiyuan Wan, David Lo, Difan Xie, and Xiaohu Yang. 2025. Why Does My Transaction Fail? A First Look at Failed Transactions on the Solana Blockchain. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA066 (July 2025), 24 pages. https://doi.org/10.1145/3728943

1 Introduction

Solana is an emerging blockchain platform recognized for the high throughput and low transaction costs that it offers, making it a preferred infrastructure for DeFi [49], NFTs [51], and other applications of Web 3.0 [61]. At the core of the design of Solana is a stateless smart contract model, where smart contracts (i.e., Solana programs) do not retain internal states. Instead, all necessary state data resides in external accounts, which are passed as inputs when transactions invoke these programs. This stateless model not only streamlines the logic of programs and minimizes storage on-chain, but also facilitates the parallel execution of transactions. Together with the Proof of History mechanism and the Tower BFT consensus protocol of Solana [19], the design achieves a substantial increase in throughput while maintaining low transaction fees.

Figure 1 presents a high-level overview of the Solana ecosystem, illustrating the typical interaction flow from transaction initiators to the various Solana programs they interact with. On the left, transaction initiators submit various instructions, such as minting, swapping, and depositing. A transaction is considered *successful* if all instructions execute without errors; otherwise, it is classified as *failed* if one or more instructions encounter errors during execution. On the right, the figure illustrates the range of programs invoked by Solana transactions, including DeFi programs such as DEXs and DEX aggregators. DEXs enable users to trade assets directly with one another on a blockchain, without the need for centralized intermediaries. DEXs frequently rely on AMM mechanisms, which allow users to trade assets through liquidity pools rather than traditional order books.

While Solana aims to maintain low transaction fees and high throughput, the low cost of transactions has inadvertently exposed the Solana network to bot spamming for financial exploitation [17, 27], where transactions are automatically generated by machines. The bot activity has led to increased failure rates of transactions and network congestion, with reports indicating that up to 75% of non-vote transactions failed during periods of "memecoin mania" [22] and bot-induced spam. Non-vote transactions, unlike vote transactions required for network consensus, encompass general-purpose activities such as token transfers and smart contract executions, making them more vulnerable to automated abuse. Mert Mumtaz, CEO of Helius, notes that a significant portion of these transaction failures originates from bot activity rather than normal user interactions, resulting in a degraded user experience on the network [29]. Our experimental results, depicted in Fig. 2, present the hourly trends of failed non-vote transactions on Solana from March 6 to March 21, 2024, highlighting the prevalence of transaction failures on Solana.

Previous studies on the Solana ecosystem have primarily explored two aspects of the ecosystem, either evaluating the performance of the Solana blockchain, with a particular focus on its scalability [77] and transaction throughput [57, 77], or enhancing smart contract security through vulnerability detection techniques [54, 79]. Nonetheless, these efforts have overlooked the transaction-level

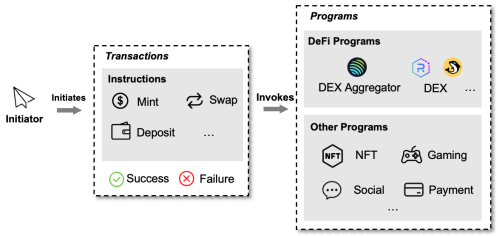


Fig. 1. Overview of Solana Ecosystem.

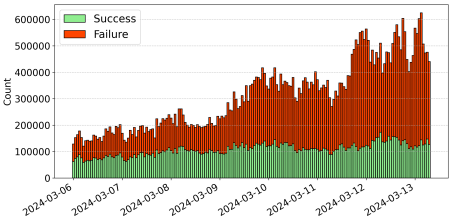


Fig. 2. Hourly Trends of Successful and Failed Non-Vote Transactions on Solana (March 6-13, 2024).

analysis in the Solana ecosystem, leaving a critical gap in understanding the characteristics and implications of failed transactions on Solana. As reported in previous studies [28, 32], Solana exhibits a significantly higher transaction failure rate, which can exceed 75% during periods of network congestion, in contrast to the transaction failure rate from 1% to 3% on Ethereum [44, 73]. Such high transaction failure rate of Solana can introduce delays in transaction finality and contribute to an increased network load, potentially disrupting the seamless operation expected by users and developers. If left unaddressed, high transaction failure rates could undermine confidence of users and developers in the Solana ecosystem, particularly if users and developers come to perceive the ecosystem as unreliable. Thus, a comprehensive understanding of the characteristics and implications of failed transactions would provide valuable insights into the Solana ecosystem, enabling both regular users and bot developers to optimize transaction design and execution, ultimately improving the efficiency of the transactions and the reliability of the network.

To address this gap, we conducted a large-scale empirical investigation into failed transactions on the Solana blockchain. Specifically, we curated a dataset comprising 2,898,175,006 non-vote transactions, comprised of 1,387,340,838 *successful* and 1,510,834,168 *failed* transactions, spanning from August 1, 2023 to July 31, 2024, enriched with both relevant on-chain and off-chain data. Below, we present our research questions and highlight the key findings:

RQ1: What are the characteristics of the failed transactions?

Given the high transaction volume and substantial failure rates on Solana, it is essential to systematically capture the characteristics of these failed transactions. The understanding of the failed transactions can help identify issues that contribute to the failures, and inform the development of targeted strategies to address these issues in the Solana ecosystem. Therefore, we conducted a two-level analysis of the failed transactions on the Solana blockchain in our dataset, macro- and micro-level analysis. At the macro level, we captured the initiators, failure-triggering programs, and temporal patterns of the failed transactions. At the micro level, we focused on attributes of the transactions in our dataset, comparing transaction fees and block positions between successful and failed transactions. Our analysis revealed distinct daily patterns in transaction failures, with bots showing a high transaction failure rate of 58.43%. The top ten programs with the highest volume of failed transactions account for 77.95% of all failed transactions. In addition, failed transactions tend to be positioned deeper in blocks[1] as compared to successful transactions (592 vs. 529 in median).

RQ2: What are the errors that cause transaction failures?

Failed transactions on Solana are accompanied by a range of errors, as outlined in their log messages. These errors can provide insights into the underlying causes of transaction failures, and uncover programming challenges faced by developers and users of Solana. Thus, in this RQ, we aim to establish a systematic classification of the errors responsible for transaction failures in our

[1]A Solana block is a collection of transactions processed and confirmed by validators on the Solana blockchain, representing a specific, sequential segment of the transaction history of the blockchain.

dataset. Specifically, we extracted error messages from the log messages associated with failed transactions in our dataset, and applied thematic analysis to categorize the identified error messages into distinct error types. We identified 10 distinct error categories, with the vast majority of failed transactions (67.18%) attributed to either *price or profit not met* or *invalid status* errors. A smaller portion of failures stems from issues arising in the improper interactions between transaction initiators and the corresponding programs, including *out of funds* (2.16%), *invalid input parameters* (2.55%), and *invalid input account* (3.27%). Furthermore, a limited number of failures are linked to runtime environment issues (0.72%) and resource management constraints, with *out of resource* errors accounting for 0.49%, indicating potential gaps in program testing and validation for initiating Solana transactions.

RQ3: How do programs and transaction initiators contribute to different error types in transaction failures?

Transactions on Solana involve interactions between programs and transaction initiators, with failures arising from both program design and the actions of transaction initiators. Characterizing the programs and transaction initiators contribute to transaction failures can provide insights into improving program design and transaction initiation practices, with the goal of reducing failures by addressing specific error types. Thus, we investigated how programs and transaction initiators shape the distribution of error types in failed transactions. Our investigation focused on two dimensions, the error type distributions across the programs with high failed transaction volumes, as well as the difference in distributions of error types between bot-initiated and human-initiated transactions. We observed that the programs of DEX aggregators predominantly fail due to *price or profit not met* errors. The programs of AMM frequently encounter *invalid status* errors, often caused by bots attempting to front-run or manipulate transactions. Bot-initiated transactions exhibit various error types, with *price or profit not met* errors being the most common. Human-initiated transactions are more prone to *out of funds* errors as compared to bot-initiated transactions.

Based on our findings, we discuss implications and provide recommendations for Solana ecosystem core developers, protocol designers, and end users, including mechanisms and tools to mitigate transaction failures, and outline future avenues of research. We make the following contributions:

- We present the first large-scale empirical study on failed transactions on Solana, identifying 10 distinct error types associated with failed transactions and providing insights into the Solana ecosystem.
- We curate a dataset that comprises 1,510,834,168 failed transactions on Solana along with corresponding error messages for future investigation by others.
- We provide practical recommendations for practitioners in the Solana ecosystem, and outline future avenues of research.

2 Background

The architecture of the Solana blockchain is built around four essential concepts. To interact with the Solana blockchain, an entity must first create an *account*, which serves not only as a unique identifier but also as a store of relevant data and states. This account can initiate the execution of a smart contract, referred to as a Solana *program*, by submitting a *transaction*. A transaction consists of one or more *instructions*, each specifying an operation to be performed by a program. The following provides a detailed exploration of these concepts.

2.1 Solana Accounts and Programs

The Solana blockchain relies on a flexible account-based model to manage both data and executable logic. At the core of the model are two distinct types of accounts: (1) *data accounts*, which store non-executable data, and (2) *program accounts*, which hold executable code that defines the behavior

of programs. Programs in Solana are stateless, they do not maintain persistent data between transactions. Instead, they interact with data accounts to store and modify state(s) as needed.

Each account in Solana is uniquely identified by a 32-byte public key and consists of four components [24]: (1) Public key of the owner program, which specifies the program that controls the operations of the account. (2) Executable flag, which indicates whether the account contains a program (executable code). (3) Data content, which stores relevant information, such as token balances or configuration parameters. and (4) Balance, which holds the balance of the account, measured in Solana's smallest currency unit, lamports.

In the stateless programming model of Solana, programs manage states by creating or updating data accounts rather than maintaining states internally. Consequently, for each instruction of a transaction to execute, the initiator of the transaction must explicitly provide all relevant accounts as instruction arguments, including both the data accounts and program accounts involved in the operation.

2.2 Solana Transactions

In Solana, transactions are divided into two categories, *vote* transactions and *non-vote* transactions, each serving a distinct role in the Solana network. Vote transactions are system-level operations initiated by validator nodes to maintain network consensus by confirming new blocks and ensuring synchronization of the blockchain's state. Vote transactions are essential for preserving network stability and are integral to the consensus mechanism of Solana [12]. In contrast, non-vote transactions encompass all user-initiated activities, such as token transfers, account creation, decentralized exchange (DEX) operations, and NFT minting. Non-vote transactions represent the primary interactions of users and applications with Solana [14]. The separation between vote and non-vote transactions aims to ensure that the network can efficiently handle both consensus-related operations and user-driven activities, maintaining high throughput and reliability of Solana.

Solana transactions are composed of *instructions* that interact with various on-chain programs, where each instruction specifies a distinct operation to be performed [31]. When an entity submits a transaction to the Solana network, it includes one or more instructions, each containing the program to be invoked, relevant account addresses, and a data byte array. The designated program processes the data and executes the specified operations on the given accounts. Transactions on Solana are executed sequentially and atomically–if any instruction fails, all state changes made by the transaction are discarded, ensuring system consistency [18].

The lifecycle of a transaction begins when it is ***submitted*** to a Remote Procedure Call (RPC) server–an endpoint that allows entities to interact with the blockchain network–which forwards it to both the current leader and the next scheduled leader. Note that a leader is a validator temporarily assigned to produce blocks during a specific time slot. Unlike Ethereum, where transactions are propagated through a public mempool, Solana broadcasts transactions directly to validators responsible for block production. The leader processes the transaction, appends it to the current block, and then propagates it across the network using Solana's Turbine broadcast mechanism, which efficiently fragments and distributes data to validators [41]. For a transaction to be ***accepted*** and ***processed***, it must contain one or more ***signers*** who authorize the operation by providing digital signatures generated with their private keys. Each transaction must include at least one writable signer account, typically the fee payer, who bears the transaction cost. The signer account is serialized first in the account input list of the transaction to ensure the fee deduction happens before any further processing. The signer accounts can be categorized into two distinct types [71]: (1) bot accounts (🤖), which are operated programmatically by automated systems, and (2) human accounts (👤), which are managed manually by users.

A block is considered ***confirmed*** once it receives votes from at least two-thirds of the active validators. For a transaction to reach ***finality***, 31 additional blocks must be built upon the corresponding block containing the transaction, ensuring its immutability [30]. A finalized transaction can be classified as either ***successful*** if all its instructions are executed without any errors, or ***failed*** if one or more instructions in it encounter an error during its execution. Failed transactions produce detailed log messages, enabling developers to diagnose the underlying issues. For instance, Listing 1 presents a log message fragment from the failed transaction `36DLJ5vufG`, in which, the error log entry at line 6 indicates a slippage-related error. We attribute responsibility for the error to the outermost program in the call stack of the log message, which in this example is the program `6Q4Xu2`, as identified at line 1 of Listing 1.

Both successful and failed transactions incur fees to cover the operational costs of processing and prevent spam in the Solana network. Each transaction requires a base fee of 5,000 lamports per signature, regardless of the computational resources required, ensuring that even small transactions contribute to the sustainability of the network [20]. Lamports are the smallest denomination of Solana's native cryptocurrency, SOL, where 1 SOL equals 1 billion lamports. Solana provides further flexibility through two parameters, *compute unit limit* and *compute unit prices*, allowing users to optimize the performance and priority of the transactions based on their complexity and urgency. The compute unit limit specifies the maximum computational resources, measured in compute units (CU), that a transaction can consume. The limit is configured by the sender of a transaction based on the complexity of the transaction. The compute unit price, measured in micro-lamports per CU, reflects the extra amount of SOL that the sender is willing to pay for a transaction. Validators are incentivized to process transactions with higher CU prices first, ensuring optimal resource utilization in the network and faster execution of transactions.

Solana employs a fee-based prioritization system [6] to optimize transaction processing to maintain the high performance of Solana. While users can increase transaction fees to compete for higher priority in transaction processing, the transaction scheduling mechanism of Solana does not always guarantee that transactions with higher fees be processed in prior to those with lower fees [50]. Other factors, such as the allocation of compute units by transaction initiators [81] and the mechanisms applied by validators [50], can affect the order of transactions in blocks on Solana. For instance, the Jito Tips mechanism allows searchers and users to offer tips to validators in return for prioritizing their transactions [74].

Listing 1. Log Message Fragment of an Example Failed Transaction on the Solana Blockchain.

```
1  Program 6Q4Xu2 invoke [1]
2  Program log: Instruction: Swap
3  Program log: x.com/TechnoBotSolana
4  Program 675kPX9 invoke [2]
5  Program log: ray_log: ...
6  Program log: Error:exceeds desired slippage limit
7  Program 675kPX9 consumed 16323 of 62684 compute units
8  Program 675kPX9 failed: custom program error: 0x1e
9  Program 6Q4Xu2 consumed 53339 of 99700 compute units
10 Program 6Q4Xu2 failed: custom program error: 0x1e
```

3 Dataset

3.1 Data Collection and Preprocessing

To conduct a comprehensive investigation of failed transactions on Solana, we selected a one-year period from August 1, 2023 to July 31, 2024, spanning 72,123,900 blocks on the Solana blockchain

with the block range from 208,703,000 to 280,826,900. Given the large transaction volume over the one-year period, we designed a stratified sampling strategy for block selection on a weekly basis, where we randomly selected one day per week throughout the year, while ensuring a balanced representation of both weekdays and weekends. The sampling strategy achieves a 99.999% confidence level with a margin of error of ±0.06%, ensuring that the selected sample represents the broader transaction patterns on Solana. Consequently, we selected 10,458,452 blocks from 53 days between August 1, 2023 and July 31, 2024.

On-chain Data. We first retrieved the selected blocks from the Solana blockchain through an RPC node, provided by Alchemy [1], utilizing the Solana JSON RPC API. For each block, we iterated over its transactions, filtered out vote transactions (those invoking the vote program) and extracted multiple entries from each remaining non-vote transaction, including (1) the block number, (2) the block timestamp, (3) transaction hash, (4) account inputs, (5) signer(s), (6) transaction fees, (7) *compute units* the transaction costs, (8) transaction log messages, and (9) program address and its corresponding instruction that raises errors if the transaction fails. Note that we consider a transaction as a *failed* transaction if the error fields in its metadata are not empty. For each failed transaction, we further analyzed the log messages to identify the specific error log messages the triggered during execution.

Off-chain data of Solana. We also collected off-chain data related to Solana programs. We obtained program public names from Solscan [82], a comprehensive Solana blockchain explorer and analytics platform. Additionally, we retrieved and cloned program source code from GitHub repositories when available through URLs listed in Solscan's security section.

Consequently, we curated a dataset of 2,898,175,006 non-vote transactions on the Solana blockchain, comprising both on-chain data (including block information, transaction details, compute units, and error messages if any) and off-chain data (including program names and their source code if any).

3.2 Account Classification

The transactions on Solana may originate from accounts managed by machines, bot accounts, or human users. To classify the accounts that initiate the transactions in our dataset, we applied a classification algorithm that takes into account the on-chain behavior of the accounts. Specifically, we characterized the on-chain behavior of a Solana account by considering multiple features from its transaction histories, which aligned with prior work [62, 71]:

❶ ***Transaction Frequency:*** We measured the transaction frequency of each account by computing the average and variance of the intervals between the transactions the account has initiated. Specifically, for each account, we extracted a time series $[t_1, t_2, ..., t_n]$, representing the timestamps of n transactions initiated by the account within our dataset. t_i denotes the timestamp when the account initiated its ith transaction, as recorded in the corresponding block. The interval between consecutive transactions is calculated as $t_{i+1} - t_i$, where $i < n$.

❷ ***Transaction Volume:*** We quantified the transaction volume of each account by calculating both the total number of transactions and the average number of transactions per block initiated by the account within our dataset. In calculating the average transactions per block, we excluded blocks that do not contain any transactions initiated by the respective account.

We utilized the scikit-learn implementation[2] of the random forest (RF) algorithm [76] to classify 12,712,516 accounts in our dataset. Specifically, we first constructed the ground truth by manually labeling 200 accounts randomly selected from our dataset. Specifically, for each account, we analyzed the frequency, volumes and intervals of transactions the account initiated, programs the account

[2]https://scikit-learn.org/

interacted with, and its involvement in casino games. The manual labeling resulted in the ground truth consisting of 96 human accounts and 104 bot accounts. Using the ground truth as the training set, we built a RF classification model to categorize the remaining accounts. For each account, the classification model outputs a probability for the human or bot category. Accounts with a probability above 0.9 were classified with high confidence, while those below 0.9 were labeled as "unknown". As a result, we classified the accounts in our dataset into 803,136 bot accounts and 1,359,772 human accounts. To evaluate the reliability of the classification model, we randomly sampled 97 bot accounts and 97 human accounts from the classifications for validation, using a 95% confidence interval and a 10% sampling error. Among the 194 randomly sampled accounts, we identified six misclassified cases, including three human accounts exhibiting high-frequency trading within a single day, and three bot accounts performing low-frequency but consistent transactions, suggesting that our classification model achieved an accuracy of 96.91%.

4 Characteristics of Failed Transactions (RQ1)

4.1 Methodology

To answer RQ1, we first aggregated the failed transactions based on the accounts that initiated the transactions, distinguishing between bots and human users. Next, we grouped the failed transactions by the programs responsible for the corresponding failures and characterized the functionality of these programs. To account for variations in overall transaction volume, we also normalized the number of failed transactions against the total non-vote transactions per hour, yielding the hourly transaction failure rate. Lastly, we explored the relationship between hourly transaction volume and failure rate, and employed autocorrelation analysis [45] to uncover potential periodic patterns in the time series of transaction failure rates.

We followed up with a micro-level analysis of the failed transactions in our dataset. Specifically, we first compare the positions within blocks, as well as the transaction fees, compute units, and cost efficiency between successful and failed transactions on Solana. We then employed Wilcoxon rank-sum tests to determine the statistical significance of these differences.

4.2 Results

4.2.1 Macro-Level Analysis. We first provide an overview of the failed transactions on Solana from three dimensions: the types of accounts initiating these transactions, the programs responsible for the failures, and the temporal trends in transaction volume and failure rate.

Account Types. Table 1 compares identified bot and human accounts on the Solana blockchain, revealing notable differences in the numbers of accounts and failure rates of transactions between bot and human accounts. Bot accounts, comprising over 0.8 million accounts in our dataset, exhibit a high transaction failure rate of 58.43%, with approximately 453.5 million failed transactions compared to 322.6 million successful ones. The high failure rate suggests that bot-initiated transactions may rely heavily on high-frequency strategies, which may contribute to network congestion. In contrast, the 1,359,772 human accounts show a considerably lower transaction failure rate of 6.22%, achieving 2,978,711 successful transactions against only 197,535 failures. The disparity in transaction failure rates between bot and human accounts indicates that human-initiated transactions are more deliberate, potentially benefiting from strategic resource management and reduced exposure to network contention, as compared to bot-initiated transactions.

> **Finding 1:** Bot accounts on Solana exhibit a high transaction failure rate of 58.43%, in contrast to the substantially lower rate of 6.22% observed among human accounts. The disparity indicates that human-initiated transactions tend to be more deliberate than automated high-frequency bot operations.

Table 1. Comparison of Failed and Successful Transactions Initiated by Bot and Human Accounts on Solana.

Account Type	# Identified Accounts	# Failed Transactions	# Successful Transactions
Bot	803,136	453,547,307	322,618,527
Human	1,359,772	197,535	2,978,711

Table 2. Top 10 Programs Responsible for the Highest Volume of Failed Transactions (Txs) on Solana.

Address	Public Name	Description	Failed Txs (# | %)	Failure Rate (%)	Affected Accounts (#)
675kPX9MHT	Raydium Liquidity Pool V4 [7]	AMM	327,650,466 | 21.69%	74.21%	1,565,324
JUP6LkbZbj	Jupiter Aggregator V6 [5]	DEX aggregator	252,017,540 | 16.68%	79.74%	1,115,631
cjg3oHmg9uu	Chainlink Data Store [4]	Decentralized Data Storage	226,952,883 | 15.02%	94.18%	23
6Q4Xu2sXxM	-	-	87,172,461 | 5.77%	99.12%	105
JUP4Fb2cqi	Jupiter Aggregator V4 [64]	DEX aggregator	84,279,822 | 5.58%	96.14%	5,816
GDDMwNyyx8	Sequence Enforcer [8]	Transaction ordering mechanism	71,616,610 | 4.74%	37.37%	105
ATokenGPvb	Associated Token Account [3]	Account management utility	44,956,578 | 2.98%	32.90%	544,818
9uW2TqLyfY	-	-	40,817,555 | 2.70%	97.67%	924
YmirFH6wUr	-	-	24,635,690 | 1.63%	95.21%	4
3J3HFc8jXx	-	-	17,509,217 | 1.16%	91.75%	7

Programs. A total of 2,324 distinct programs are identified as being responsible for at least one failed transaction in our dataset, among which, each program is responsible for a median of 30 failed transactions (min: 1, average: 649,411, max: 327,650,466). Figure 3 presents the logarithmic scale Cumulative Distribution Function (CDF) of the number of failed transactions across these programs. The CDF curve exhibits a steep increase from 10^1 to approximately 10^3 failed transactions, indicating that a large number of programs are responsible for relatively few failed transactions. After the initial steep increase, the slope of the curve becomes more gradual, indicating that fewer programs are responsible for higher numbers of failed transactions. The curve extends towards the rightmost end, suggesting that a very small fraction of programs are responsible for hundreds of millions of failed transactions, which tend to play critical roles in the Solana ecosystem.

We further performed a detailed analysis of the top 10 programs responsible for the highest volume of failed transactions in our dataset, situated at the rightmost end of the CDF curve. An overview of the ten programs is presented in Table 2, among which, six are public services, while four are privately deployed with no name disclosed. We made the following observations:

- Operating as an automated market maker (AMM)[3], Raydium Liquidity Pool V4 alone accounts for 21.69% of all failed transactions, totaling over 327 million failed attempts, affecting 1,565,324 accounts. The high failure rate of 74.21% indicates that a vast number of transactions interacting with the program encounter issues, potentially due to liquidity constraints, programmatic bugs, or excessive competition for resources.
- Jupiter Aggregator V6 follows closely, designed as a decentralized exchange (DEX) aggregator, contributing 16.68% of the total failed transactions with a failure rate of 79.74%, affecting 1,114,631 accounts.
- The Chainlink Data Store program contributes to 15.02% of all failed transactions, with the high failure rate of 94.18% being attributed to only 23 accounts.
- The Associated Token Account and Sequence Enforcer programs, both integral to the blockchain operations of Solana, exhibit a comparable moderate failure rate, each approaching 40%. This indicates potential bottlenecks at the infrastructure level of Solana, such as network congestion or insufficient computational resources. Meanwhile, a significant difference exists in the number of accounts they affect (544,818 vs. 105), reflecting the fundamental difference

[3]An automated market maker is a decentralized trading mechanism that allows users to trade assets through liquidity pools rather than traditional order books. In an AMM, liquidity providers deposit pairs of tokens into these pools, and trades are facilitated by an algorithm that adjusts prices based on the pool's token ratios. This model is widely used in DEXs, providing continuous liquidity and enabling permissionless trading.

in their usage patterns and operational scope in the Solana ecosystem. In particular, the Associated Token Account program facilitates everyday user interactions and token management, while the Sequence Enforcer program focuses on transaction order enforcement in specialized contexts.

- Several programs, such as the one associated with the address 6Q4Xu2sXxM and Jupiter Aggregator V4, display extremely high failure rates, exceeding 95%. This suggests they might be highly congested or not adequately optimized for the volume of incoming transactions.

Finding 2: 50% of the programs were involved in fewer than 30 failed transactions on Solana. The top ten programs with the highest volume of failed transactions account for 77.95% of all failed transactions, with three of them being public DeFi programs in the Solana ecosystem.

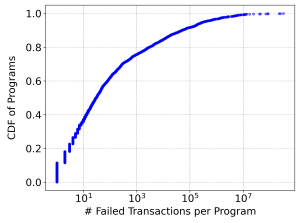


Fig. 3. Cumulative Distribution of Programs Responsible for Failed Transactions on Solana.

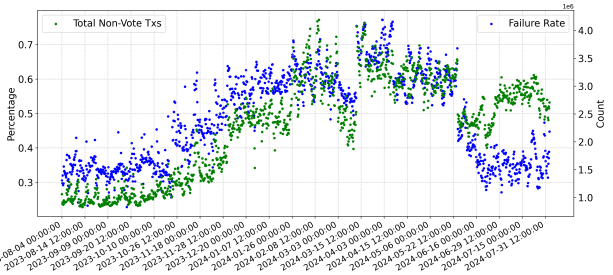


Fig. 4. Hourly Number of Non-Vote Transactions and Failure Rate on Solana.

Temporal Trends. Figure 4 provides a detailed temporal analysis of non-vote transactions on the Solana blockchain, depicting the sampled transaction volume (green dots) and transaction failure rate (blue dots) on an hourly basis between August 1, 2023 and July 31, 2024. Overall, the fluctuations in non-vote transaction volume appear to coincide with the variations in transaction failure rates over time. On the one hand, the hourly transaction volume (green dots) demonstrates noticeable variability over time, with frequent peaks and troughs. The variability may suggest periodic surges in network activity, possibly driven by specific applications, market events, or high-demand use cases like token transfers or DeFi operations. On the other hand, the hourly transaction failure rate (blue dots) similarly fluctuates, indicating that higher failure rates coincide with increased transaction volume. It is noteworthy that the transaction failure rates experienced a significant reduction after June 16, 2024, accompanied by a modest decrease in transaction volume. The reduction in transaction failure rates may be attributed to the validator update and consensus optimization that were introduced to the Solana blockchain on June 10, 2024 [21]. A Wilcoxon rank-sum test confirms the statistical significance of the difference in block positions between failed and successful transactions (p-value < 0.001), suggesting that failed transactions are outcompeted, leading to their deeper placement within the blocks.

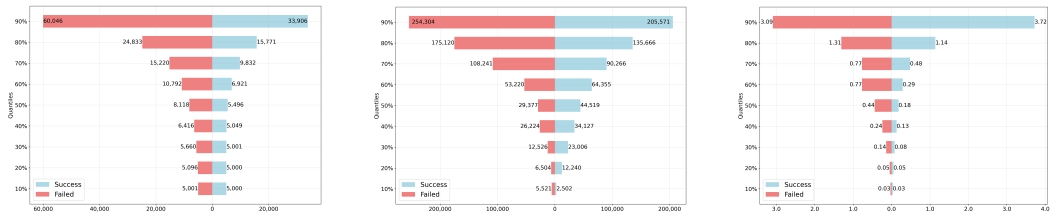
Furthermore, the autocorrelation analysis of transaction failure rates demonstrates statistically significant temporal dependencies, indicating that recent transaction failure rates are influenced by preceding ones. The analysis also identifies recurring patterns at regular intervals, most prominently at 24 hours, suggesting the consistent reoccurrence of specific operational processes or user behaviors within a structured daily cycle.

Finding 3: Failure rates exhibit a strong positive correlation with the volume of failed transactions on an hourly basis, and demonstrate cyclical patterns at 24-hour interval, indicating structured, recurring patterns in operational processes and user behavior over time.

4.2.2 Micro-Level Analysis. We then performed a detailed analysis of failed transactions on Solana across four dimensions: their positions in blocks, the transaction fees incurred, the compute units (CU) utilized, and cost efficiency, defined as the transaction fees per CU.

Positions in Blocks. Failed transactions occupy a median position of 592 (mean: 754, min: 0, max: 8,134) in blocks, whereas successful transactions are positioned closer to the top of blocks, with a median rank of 529 (mean: 764, min: 0, max: 8,264). A Wilcoxon rank-sum test confirms the statistical significance of the difference in block positions between failed and successful transactions (p-value < 0.001), suggesting that failed transactions are outcompeted, leading to their deeper placement within the blocks.

Transaction Fees, Compute Units, and Cost Efficiency. Figure 5 compares successful and failed transactions on Solana across three dimensions: transaction fees, compute units (CUs) consumed, and cost efficiency (fees per CU). As shown in Figure 5a, failed transactions incur higher transaction fees than successful transactions across all quantiles. Figure 5b shows that failed transactions consume fewer compute units than successful transactions, particularly below the 70% quantile, which may indicate an under-allocation of computing resources [80]. Meanwhile, as depicted in Figure 5c, failed transactions exhibit higher fees per CU compared to successful transactions, especially below the 80% quantile, suggesting potential resource misallocation or suboptimal execution of failed transactions. The Wilcoxon rank-sum test yielded p-values less than 0.001 for all three dimensions, confirming statistically significant differences between failed and successful transactions in terms of transaction fees, compute units, and cost efficiency.



(a) Transaction Fee (b) Compute Units (CU) (c) Fee per CU

Fig. 5. Fees, Compute Units, and Cost Efficiency of Successful vs. Failed Transactions on Solana.

In summary, the failed transactions incur higher transaction fees, indicating their attempt to prioritize their processing in line with the fee-based prioritization system of Solana. However, the failed transactions have be outcompeted by the successful ones, as indicated by their later positions in blocks. The significantly lower CU and higher fees per CU of failed transactions further suggests the inefficiency in the execution of failed transactions, which may explain why failed transactions have been outcompeted by successful ones. The inefficiency highlights the need for Solana program developers to implement mechanisms that provide users initiating failed transactions with timely feedback, include program-specific guidance on optimizing transaction submission timing, fee estimation, and compute unit allocation, rather than relying on indiscriminate fee increases to improve transaction success rates.

Finding 4: Failed transactions tend to appear in later positions in blocks compared to successful ones (median position 592 vs. 529) and exhibit higher transaction fees, as well as higher fees per CU, despite using lower compute units, suggesting a relative inefficiency in the execution of failed transactions. The observed inefficiency highlights the need for Solana program developers to implement mechanisms that provide users initiating failed transactions with timely feedback, include program-specific guidance on optimizing transaction submission timing, fee estimation, and compute unit allocation, rather than relying on indiscriminate fee increases to improve transaction success rates.

5 Errors in Failed Transactions (RQ2)

5.1 Methodology

To answer RQ2, we first extracted the error message from each error log entry in the log messages of the failed transactions in our dataset by taking into account their development frameworks through the following steps. ❶ ***Anchor framework development:*** For the failed transactions originating from the programs that are developed based on the Anchor framework[4], the error log entries follow a specific macro defined in the Anchor framework, i.e., `#[error(error_code)]`. These entries are formatted as `Program log: AnchorError occurred. Error Code: <Error_Code>. Error Number: <Error_Number>. Error Message: <Error_Message>`. We extracted such error log entries based on the `Error_Message` field. ❷ ***Non-Anchor framework development:*** For the failed transactions originating from the programs that are built with no support from the Anchor framework, we located the corresponding error log entries by searching for the keywords in the log messages, such as *error*, *failed*, *invalid*, *panicked*, and *mismatched*. For example, in Listing 1, we identified line 6 as the error log entry and extracted "exceeds desired slippage limit" as the error message. Note that we excluded error logs that contained incomplete, truncated, or missing error messages from failed transactions. Consequently, we extracted 2,311 unique error messages from 1,145,479,366 failed transactions in our dataset.

Next, we applied thematic analysis [53] to the error messages for classification, through the following steps.

❶ ***Coding:*** We began by coding the top 100 error messages that are most frequently occurring in the log messages of the failed transactions in our dataset. For each error message, we randomly selected multiple failed transactions from our dataset that were associated with the error message, until adequate information for determining its code. Two labelers reviewed the complete log messages of the selected failed transactions, as well as the source code of the corresponding programs if available. To ensure the quality of the codes, the two labelers discussed the initial codes to reach an agreement. As a result, 34 codes emerged in the initial round of coding. We then continued to expand the scope of coding by considering 73 additional error messages with more than 50,000 occurrences, until no new code emerged. Consequently, we identified 11 additional codes from the error messages, resulting in 45 codes in total, accounting for 99.71% of the error messages associated with the failed transactions with explicit error messages of our dataset. Among the 173 error messages we coded, 61 were associated with at least one open-source programs, for which the source code was available for reference during the coding process, while 112 were triggered exclusively by closed-source programs. For the 91 error messages that lacked source code, we derived codes based on the literal content of the corresponding log messages. For example, the error message *Slippage tolerance exceeded*, which is associated with 150,672,490 failed transactions originating from the Jupiter Aggregator V6 in our dataset, was coded as "Slippage". The remaining 21 error messages, for which no source code was available and whose literal content was ambiguous, were coded as "Unknown".

❷ ***Card Sorting:*** We used the open card sorting technique [84], a widely adopted technique for organizing items without predefined categories. Specifically, the two labelers separately sorted the codes into potential themes for thematic similarity, following a similar process as prior studies, e.g., LaToza et al.'s study [65].

Any discrepancies between the two labelers were resolved through discussions with another author of the paper, an expert in Solana, to reach a consensus. For further validation, the third author reviewed the derived codes during the coding phase and classified the codes according to the proposed themes. No misclassification cases were identified in the validation process.

[4] https://www.anchor-lang.com/

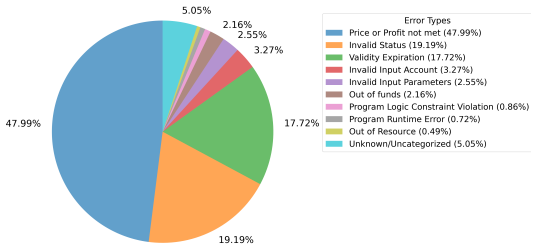


Fig. 6. Distribution of Error Types in Failed Transactions.

5.2 Results

Figure 6 illustrates the distribution of the ten error types we identified in failed transactions, revealing a long-tail distribution. The three most prominent error types, *price or profit not met* (47.99%), *invalid status* (19.19%), and *validity expiration* (17.72%), together account for 84.90% of all failures, indicating the critical role of business logic validation, state management, and infrastructure reliability in ensuring transaction success on Solana. Secondary error types, such as *invalid input account* (3.27%), *invalid input parameters* (2.55%), and *out of funds* (2.16%), indicate operational challenges related to resource availability and input validation. Less frequent error types, including *program logic constraint violation* (0.86%), *program runtime error* (0.72%), and *out of resource* (0.49%), though rare, highlight the presence of potential corner cases and programming defects that could impact system robustness under specific conditions. The following provides a detailed description of each error type, accompanied by the corresponding error logs and representative transaction examples.

Price or Profit not Met serves two essential purposes: (1) protecting users from potential financial losses due to price slippage, and (2) mitigating the risk of front-running attacks, which exploit latency and market inefficiencies to gain unfair advantages [26]. The relevant error logs of this type are presented below:

- *Slippage tolerance exceeded:* The error occurs when token prices fluctuate between the submission and processing of a transaction, and a low slippage tolerance causes the transaction to fail if the fluctuation of token prices exceeds the specified threshold. A representative example of this error arises from the transaction `5BB13Yde` [37], which involves an attempt at cyclic arbitrage to acquire USDT tokens. The transaction enforces a zero-slippage tolerance, requiring an exact match in exchange rates to proceed. Due to this strict condition, the transaction fails when the final balance of USDT tokens proves insufficient, resulting in the loss of only the transaction fees.
- *IOC order failed to meet minimum fill requirements:* The error log reflects the failure of an Immediate-Or-Cancel (IOC) order to meet the required minimum quantity or price conditions, leading to its automatic cancellation. On the Solana blockchain, IOC orders are often employed in DEXs to ensure efficient execution by either fulfilling the specified conditions immediately or canceling the order to prevent unfavorable outcomes [25]. If market conditions shift, such as slippage exceeding the set tolerance or liquidity being insufficient, the order fails to execute within the required parameters.

Invalid Status occurs when transaction initiators attempt to execute operations on program-owned data or off-chain data that is not in a valid state to permit the requested operations. The relevant error logs of this type are presented below:

- *The amm account owner is not match with this program:* The error occurs when the on-chain data account of a liquidity pool, which stores essential information such as the state of the pool and its fee rates, is not yet finalized at the time of creation. If a swap transaction is

initiated before the completion of the initialization of the pool, the operation will fail with this error. The error is often caused by sniper bots that detect pool creation transactions and attempt to execute trades prematurely, aiming to be among the first transactions processed in the newly created pool. For instance, the scheduled opening time of the BLAST-SOL pool was March 6, 2024, at 08:05:13 UTC. However, the transaction `5AnmV4S` [36] attempted to perform a swap 49 seconds prior to the initialization of the pool, leading to the error.

- *Account is frozen:* The error log indicates that a transaction attempts to interact with a token account that has been frozen by the mint authority, rendering all token operations unavailable. Such freezing may be associated with honeypot attacks, where malicious actors issue fraudulent tokens to attract investments and subsequently freeze the funds of investors, preventing them from making withdrawals [33].

Validity Expiration occurs when a transaction fails to execute within its designated validity window. On Solana, each transaction incorporates a `recent_blockhash` field to define the time window during which it remains valid. If the transaction reaches a validator after the expiration of its validity window, typically due to network congestion or connectivity disruptions, the validator will reject the transaction, thereby mitigating the risk of replay attacks [38]. Examples of error logs for this error type include include *slot expired*, *stale report*, and *time expired*.

Invalid Input Account occurs when a user provides unauthorized or incorrect accounts for an instruction or omits the required accounts. In the stateless programming model of Solana, instructions depend on the transaction initiators to specify the accounts on which the operations will act. If the provided accounts are invalid or do not align with the requirements of the instructions, such error type will be triggered. The relevant error logs of this type are presented below:

- *Not enough account keys given to the instruction:* The error log is generated when the required account addresses are not adequately supplied for the execution of an instruction. For example, the transaction `ebX6rwkn` [39] involves a complex operation consuming 267,105 compute units and interacting with 16 distinct programs. Although 66 input accounts are provided for the `3J3HFc8` program instruction, the transaction encounters such an error. For operations of this scale, it can be particularly challenging for transaction initiators to identify all required account addresses and arrange them in the correct sequence to ensure successful execution.
- *InvalidSplTokenProgram:* The error log occurs when the owner of the provided input account does not align with the expected address of the Solana native token program. For instance, when the account for a specific token has not been properly initialized, the use of such an invalid address in a token swap operation will trigger such error log [34].

Invalid Input Parameters arises when a user provides an argument for an instruction that falls outside the acceptable range, either too small, too large, or in an incorrect format. Such errors often occur because programs impose specific requirements on input data, such as predefined ranges or structures, which transaction initiators may overlook, particularly if the program documentation does not clearly specify these requirements. An example of error logs for this error type is *InvalidInput*, which indicates that the provided instruction input is invalid. For instance, the Raydium program enforces that the minimum output swap amount must be greater than zero. However, in the transaction `zxFnAz` [40], the input parameter for the minimum swap output is incorrectly set to 0, triggering this error log.

Out of Funds indicates that Solana users lack sufficient tokens in their wallets to cover the transaction amount along with the associated fees. The Raydium program enforces a minimum balance of SOL in the wallet to facilitate trades or swaps [23]. Transaction initiators may fail to account for these fees, leading to this error type. Examples of error logs for this error type include *insufficient funds*, *insufficient collateral*, and *insufficient funds for instruction*.

Program Logic Constraint Violation occurs when a transaction violates program logic constraints and cannot be executed. Examples of error logs for this error type include *Player has not requested a result* and *A seeds constraint was violated*. It is noteworthy that infrequent failures (fewer than 0.05 million failed transactions) are associated with the error message *The provided victim signature was already used*, as exemplified by a failed transaction on December 14, 2023 [15]. Such failures may indicate the potential occurrence of security attacks, which could serve as a critical signal for program developers to investigate the integrity of the associated programs.
Program Runtime Error arises when a transaction triggers an unhandled corner case that causes the program to abort or panic. Examples of error logs for this error type include *panicked at range end index 72 out of range for slice of length 0*, *panicked at called* `Option::unwrap()` *on a* `None` *value*, and *panicked at attempt to subtract with overflow*.
Out of Resource arises when a transaction exhausts its allocated resources, such as compute units, or exceeds the runtime constraints imposed by Solana, including constraints on heap memory and cross-program invocation data size [13]. Examples of error logs for this error type include *memory allocation failed, out of memory*, *computational budget exceeded*, and *exceeded CUs meter at BPF instruction*. For instance, the transaction `26hZfe` [35] failed due to memory exhaustion while interacting with 93 accounts via Jupiter Aggregator V6.

Finding 5: We identify ten distinct error types from the error messages of the failed transactions on Solana. The distribution of these error types across the failed transactions follows a long-tail pattern, with the top three types, *price or profit not met*, *validity expiration*, and *invalid status*, accounting for 47.99%, 19.19%, 17.72% of transaction failures, respectively, which indicate challenges in business logic validation, state management, and infrastructure reliability on Solana. Other error types, such as *out of funds* and *invalid input parameters*, suggest potential operational challenges related to resource availability, user input accuracy in the Solana ecosystem.

6 Contribution of Programs and Accounts to Errors (RQ3)

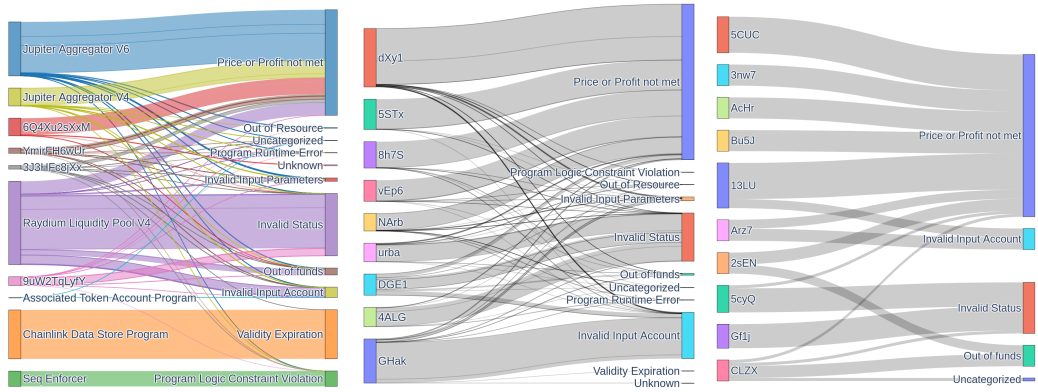
6.1 Methodology

To answer RQ3, we first identified the programs with the highest volume of failed transactions and categorized their failures according to the error types established in RQ2, revealing the specific error types associated with each program. Next, we analyzed the distributions of error types associated with two account types, and performed a comparative analysis of the error distributions between bot-driven and human-driven transactions. Moreover, we investigated how program functionalities and the distinct behaviors of these initiators contribute to the emergence of specific error types, aiming to provide insights into the root causes underlying transaction failures.

6.2 Results

Fig. 7a illustrates the distribution of various error types across different programs with top 10 failed transactions on the Solana blockchain. Each program is listed on the left side, with connections flowing to specific error types on the right, highlighting how different programs contribute to various error types of transaction failures. We made the following observations:

- Jupiter Aggregator V6 and Jupiter Aggregator V4 contribute significantly to the *price or profit not met* errors. The two programs aggregate DEXs and facilitate financial swaps, where pricing mismatches or arbitrage conditions not being met are common causes of transaction failure.
- Raydium Liquidity Pool V4 primarily causes the *invalid status* error. As suggested by a representative error message “The amm account owner does not match with this program”,



(a) Top 10 Programs (b) Top 10 Bots (c) Top 10 Human Accounts

Fig. 7. Mappings of Error Types Across Top 10 Programs, Bots, and Human Accounts with the Highest Volumes of Failed Transactions.

the program tends to be frequently targeted by sniping bots or misused, leading to failed status checks.

- The Chainlink Data Store program is identified as a significant contributor to the *validity expiration* error. Specifically, as indicated by the frequently occurred error message "Stale report", Chainlink seeks to notify the initiators of the failed transactions that the data provided by Chainlink is subject to expiration [11].
- The two private programs, 6Q4Xu2sXxM and 9uW2TqLyfY, primarily trigger the *price or profit not met* and *invalid status* errors, aligning with behaviors seen in arbitrage bots and snipping bots, respectively, and consistent with the observations in Reddit discussions [16, 42, 43].

Finding 6: DEX aggregators frequently encounter failures attributed to *price or profit not met* errors, while AMMs are primarily impacted by *invalid status* errors and tend to be targeted by sniper bots. Additionally, private programs demonstrate error patterns indicative of bot-like activity.

Fig. 7b and Fig. 7c present the distributions of error types across the accounts of two distinct categories, bot accounts (b) and human accounts (c), with top 10 failed transactions. In each figure, each account is listed on the left side, with connections flowing to specific error types on the right, highlighting how different accounts contribute to various error types of transaction failures. We made the following observations:

- The *price or profit not met* error accounts for 60.65% and 63.95% of the top 10 bot-initiated and top 10 human-initiated failed transactions, respectively, suggesting the frequent potential interactions of trading bots and human users with AMMs.
- A significant number (18.78%) of the top 10 bot-initiated failed transactions result in the *invalid status* error, which may be due to the fact that bots frequently engage with liquidity pools or other stateful programs, thus leading to status mismatches. For instance, as the frequent occurrence of the error log "The amm account owner does not match with this program" suggests, bots tend to interact with outdated or unapproved accounts, indicating misuse or aggressive strategies of sniping bots.

- The *invalid input account* error accounts for a large number of the failed transactions (17.80%) initiated by the top-10 bot-initiated failed transactions, which dominates the failed transactions of the GHak bot, suggesting the tendency of the bot in providing incorrect account addresses as parameters when initiating transactions.
- The *invalid status* error accounts for 19.77% of the top 10 human-initiated failed transactions. For instance, the error log "Account is frozen" suggests that users may have invested in potentially fraudulent tokens, leading to the freezing of the associated token accounts in order to prevent further user losses.
- The *out of funds* error accounts for 8.14% of the top 10 human-initiated failed transactions, suggesting that human users failed to provide adequate fees when initiating transactions.

In comparison, bot accounts trigger a broader range of error types as compared to human accounts. The difference in the distributions of error types between human and bot accounts suggests that bots tend to face more complicated challenges due to their high transaction volume and interaction with more advanced financial operations in the Solana ecosystem, as compared to human users.

> **Finding 7:** Bot accounts exhibit a broader range of error types in their failed transactions as compared to human accounts. The *price or profit not met* error accounts for the majority of the failed transactions for both bot and human accounts. Human-initiated transactions are more prone to *out of funds* errors as compared to bot-initiated transactions. The difference in the distributions of error types between bot and human accounts indicates that bots encounter more complex challenges arising from their high transaction volumes and interactions with advanced operations in the Solana ecosystem.

7 Discussion

7.1 Comparison across Blockchain Ecosystems

Failure Rate. We observed a transaction failure rate of 6.22% for human accounts on Solana (Finding 1), which is comparable to the transaction failure rate ranging from 1% to 3% on Ethereum, as reported in previous studies [44, 73]. In contrast, bot accounts on Solana exhibited a significantly higher transaction failure rate of 58.43% (Finding 1). The observed discrepancy in transaction failure rates for bot accounts between Solana and Ethereum can be ascribed to the fundamental architectural differences between the two platforms. Specifically, Solana's parallel processing model of transactions and low transaction fees facilitate high throughput, but may unintentionally encourage bot activity. In contrast, Ethereum's sequential processing model of transactions and gas auction mechanism inherently constrain the volume of bot activity. As prominent Layer 2 scaling solutions for Ethereum, Base and Arbitrum employ rollups to bundle multiple transactions and execute them off-chain, thereby alleviating network congestion and reducing transaction fees on Ethereum [2, 52]. The reduction in transaction fees on Base and Arbitrum incentivizes an increase in transaction volume, resulting in observed transaction failure rates of 21% and 15.4% on Base and Arbitrum, respectively [60]. Nonetheless, the failure rates on Base and Arbitrum remain significantly lower than the 58.43% observed for bot accounts on Solana (Finding 1).

Failure Patterns. We identified ten distinct error types from the error messages of failed transactions on Solana (Finding 5), three of which are also reported in the failed transactions of Ethereum [10], namely *Out of Funds (Gas)*, *Program (Contract) Runtime Error*, and *Invalid Status*. The most frequently encountered error type in failed transactions varies between Solana and Ethereum. Specifically, *Price or Profit Not Met* accounts for the majority of failed transactions on Solana (47.99%) (Finding 5), while *Out of Funds (Gas)* is the leading error type of failed transactions

on Ethereum [66, 70]. The difference in the dominant error type of failed transactions indicates that Solana's failures are predominantly attributed to non-profitable transactions in a high-volume, low-cost platform, while Ethereum's failures primarily result from insufficient gas for transaction execution in a high-cost, congested platform. The difference also reflects the differing trade-offs between transaction throughput and cost predictability inherent in the design of the two platforms.

Due to the limited availability of public transaction failure data for Bitcoin and Binance Smart Chain (BSC), we suggest that future work could put efforts into collecting failure data from the Bitcoin and BSC ecosystems, to enable comprehensive comparisons of transaction failures across different blockchain ecosystems.

7.2 Implications

Ecosystem-wide strategies could be beneficial in reducing the elevated rate of failed transactions on Solana. We observe a substantial incidence of failed transactions on Solana, possibly exacerbated by bots repeatedly submitting transactions, which further contributes to potential network congestion on Solana (Finding 1). We also note a correlation between increased transaction surges and elevated failure rates on the network (Finding 3). To prevent bots from overwhelming the network and to enhance user experience, Solana could benefit from implementing comprehensive, ecosystem-level strategies. Ethereum, by contrast, leverages a gas price auction mechanism that effectively curtails bot activity through elevated transaction fees during periods of congestion [58, 83]. However, the gas price auction mechanism on Ethereum can introduce significant volatility in gas prices, particularly during periods of network congestion, leading to unpredictable transaction costs and a poor experience for users with limited resources. In light of the trade-offs associated with the gas auction mechanism on Ethereum, the Solana ecosystem could consider incorporating a novel dynamic fee structure that automatically adjusts the costs for rapid, repeated transaction submissions from the same account. Such a dynamic fee structure could help moderate bot activity while preserving the low transaction fees of Solana, potentially balancing network accessibility with improved resilience by mitigating transaction failures. Future work could involve a comparative analysis of transaction fee designs across blockchain ecosystems, with a focus on evaluating the potential impact of varying transaction fee designs on both the network accessibility and overall resilience of the Solana ecosystem.

DeFi protocol mechanisms are necessary to mitigate bot-induced transaction failures on Solana. The high frequency of *invalid status* errors observed in Raydium suggests that this AMM has become a primary target for sniper bots on Solana (Finding 6). Engaging in speculative front-running within newly launched liquidity pools, these sniper bots contribute to a substantial volume of failed transactions, potentially gaining disproportionate market advantages and introducing risks to market stability [9]. To address this challenge, developers of AMMs on Solana might consider adopting mechanisms similar to those on Ethereum, such as imposing elevated transaction fees or restricting token purchases per transaction and/or per address during the initial phase of a liquidity pool [9]. On the other hand, another prevalent error type, *price or profit not met*, is commonly associated with arbitrage bot activity directed at Jupiter Aggregator V6, a DEX aggregator (Finding 5 and Finding 6). The error type indicates that arbitrage bots cause transaction failures when anticipated profits do not materialize or market conditions change during execution, likely due to the aggressive profit-seeking strategies they employ. To mitigate such transaction failures, DEX aggregators could implement differentiated aggregator service fee structures, where transactions initiated by arbitrage bots are subject to higher fees than those initiated by human users. The differentiated aggregator service fee structures could help disincentivize arbitrage bots from engaging in excessive profit-driven activities, thereby reducing the occurrence of transaction failures.

The need for automated tools to address development challenges of human accounts on Solana. Human accounts frequently encounter *out of funds* errors, indicating the necessity for accurate token cost calculation to prevent wallet depletion (Finding 7). Automated token cost calculation tools would assist users in managing balances more effectively, thereby reducing such errors. Besides, failed transactions appear in deeper block positions with higher per-unit fees (Finding 4). The development of precise computation unit estimation tools and gas fee calculators would help users optimize transaction placement and increase success rates. The frequent occurrence of *invalid input account* and *invalid input parameters* errors indicates a susceptibility among human accounts to input invalidity, particularly in complex transactions (Finding 5). Tools for online account query and account data structure visualization could help Solana developers verify input accounts and evaluate account states, thus reducing input-related failures. For issues such as *program runtime error*, introducing automatic testing and debugging tools for Solana programs could improve program reliability and help prevent runtime exceptions in transactions. In addition, the error type of *account is frozen*, which may indicate potential attackers on Solana, could be leveraged in the future development of intrusion detection systems. For instance, transactions associated with this error type could be integrated with other behavioral data, serving as a multi-dimensional training dataset, thereby enhancing the ability of intrusion detection systems to detect and mitigate sophisticated attacks on Solana.

7.3 Threats to Validity

Internal Validity. The classification of error messages for failed transactions involves manual inspection, which may introduce potential threats to internal validity due to subjective interpretations by labelers. To mitigate such threats, two labelers independently coded the error messages, engaging in regular comparisons and discussions to ensure coding consistency. For instances of disagreement, we facilitated in-depth discussions involving a moderator and consulted literature and contextual data to achieve accurate and reasonable categorization. For error messages triggered solely by closed-source programs, we assigned codes based on the literal content of the error messages during thematic analysis. To minimize misalignment between the actual error logic of closed-source programs and the literal content of their error messages, we consulted relevant documentation and official websites of the closed-source programs for further validation. To reduce bias from subjectivity, the results of thematic analysis were cross-validated by three of the authors.
Construct Validity. We used a classification algorithm to distinguish between bot and human accounts, which may introduce potential threats to construct validity due to potential account misclassifications. To mitigate such threats, we rigorously defined the behavioral characteristics that differentiate bots from human accounts, based on established literature [62, 71], ensuring clear and well-grounded criteria for classification. Moreover, we validated a random sample from the classification results, confirming the reliability of our classifications.

8 Related Work

8.1 Studies on Solana

Several researchers have explored the performance and scalability of the Solana blockchain platform. Li et al. [68] examined the evolution of blockchain technologies from Bitcoin to Solana, identifying key scalability and performance challenges in the enterprise adoption of blockchain solutions. Duffy et al. [57] investigated the high transaction throughput of Solana, emphasizing its potential to support large-scale IoT business applications. Pierro et al. [77] evaluated the transaction throughput on Solana and user fees for transaction confirmation, highlighting the significantly lower user fees on Solana compared to those on other blockchain platforms. Other researchers have developed

tools to detect vulnerabilities in Solana smart contracts. VRust [54] employs static analysis to detect vulnerabilities in the source code of Solana smart contracts. FuzzDelSol [79] utilizes coverage-guided fuzzing to detect vulnerabilities in closed-source, binary smart contracts on Solana.

Previous studies have predominantly explored the performance and scalability of the Solana blockchain, as well as vulnerabilities of Solana programs, yet lack empirical, transaction-based analysis of the ecosystem. Our study addresses that gap by providing the first comprehensive, large-scale exploration of the Solana ecosystem, with a particular emphasis on failed transactions.

8.2 Studies on Blockchain Ecosystems

A breadth of prior work has taken blockchain transactions as the primary subject of research. Some studies have applied a range of graph analysis techniques to explore transactions on various blockchain platforms, addressing a variety of aspects, such as anonymity, transaction network and unusual behaviors of Bitcoin [56, 59, 85], money transfer patterns and smart contracts of Ethereum [47, 48, 72], and the security issues with EOSIO [62]. Meanwhile, other studies have focused on transactions with specific attributes, such as the empirical investigation of private transactions on Ethereum [69], the examination of the relationship between transaction processing times and gas prices on Ethereum [75], the prediction of transaction statuses (confirmation or failure) using machine learning models on Ethereum [73], and the investigation of concurrency-related transaction failures on Hyperledger Fabric [46]. In contrast to prior work, our study focuses on the Solana ecosystem, providing a multifaceted characterization of failed transactions on Solana, and evaluating their financial impacts in the Solana ecosystem.

Another spectrum of prior work focuses on the analysis and detection of bot-controlled accounts on various blockchain platforms. Some studies examine the behaviors of bots on Ethereum and Binance Smart Chain, including token sniping [9], front-running [55, 67, 78], and sandwich attacks [78, 86]. Other research efforts propose techniques for identifying bot accounts on various blockchain platforms, such as EOSIO [62] and Ethereum [63, 71]. Building upon existing bot detection techniques for other blockchain platforms, our study proposes an approach to identify bots on Solana, and provide the first comprehensive characterization of their behaviors.

9 Conclusion and Future Work

In this paper, we present the first large-scale empirical study of failed transactions on the Solana blockchain, using a curated dataset of over 1.5 billion failed transactions across more than 72 million blocks. Through the dataset we curated, we systematically analyze the characteristics of failed transactions, identify ten distinct error types arising in the failed transactions, and examine the associations of error types with specific programs and transaction initiators, thereby providing insights into the Solana ecosystem. Future work could put efforts into improving the reliability and usability of the Solana blockchain through advanced error handling and recovery mechanisms, optimization of protocol and program design patterns, and the development of automated testing tools for programs and real-time monitoring for transaction failures. Moreover, cross-platform empirical studies could provide insights that improve system resilience and user experience, both on Solana as well as other blockchain ecosystems.

10 Data Availability

Our replication package is available online: https://github.com/ZXXYy/Solana_Failed_Tx.

References

[1] [n. d.]. *Alchemy Official Website.* https://www.alchemy.com/solana Accessed: 2025-04-08.
[2] [n. d.]. *The Arbitrum Network Explained.* https://www.hord.fi/blog/arbitrum-network-explained Accessed: 2025-01-02.

[3] [n. d.]. *Associated Token Account.* https://spl.solana.com/associated-token-account Accessed: 2024-10-28.
[4] [n. d.]. *Chainlink Data Store Program.* https://solscan.io/account/cjg3oHmg9uuPsP8D6g29NWvhySJkdYdAo9D25PRbKXJ Accessed: 2025-01-21.
[5] [n. d.]. *Jupiter Swap API V6.* https://station.jup.ag/docs/apis/swap-api Accessed: 2024-10-28.
[6] [n. d.]. *Priority Fees: Understanding Solana's Transaction Fee Mechanics.* https://www.helius.dev/blog/priority-fees-understanding-solanas-transaction-fee-mechanics Accessed: 2025-01-02.
[7] [n. d.]. *Raydium.* https://docs.raydium.io/raydium Accessed: 2024-10-28.
[8] [n. d.]. *Sequence Enforcer.* https://solscan.io/account/GDDMwNyyx8uB6zrqwBFHjLLG3TBYk2F8Az4yrQC5RzMp Accessed: 2024-10-28.
[9] [n. d.]. Token Spammers, Rug Pulls, and Sniper Bots: An Analysis of the Ecosystem of Tokens in Ethereum and in the Binance Smart Chain (BNB), author=Cernera, Federico and La Morgia, Massimo and Mei, Alessandro and Sassi, Francesco, booktitle=32nd USENIX Security Symposium (USENIX Security 23), pages=3349–3366, year=2023.
[10] [n. d.]. *What are the Reasons for Failed Transactions?* https://info.etherscan.com/reason-for-failed-transaction/ Accessed: 2025-01-02.
[11] 2023. *Chainlink Data Feeds, Security Researcher's Perspective.* https://ackee.xyz/blog/chainlink-data-feeds/ Accessed: 2025-01-28.
[12] 2023. *Network Performance Report: July 2023.* https://solana.com/news/network-performance-report-july-2023 Accessed: 2024-10-28.
[13] 2023. *Program logged: "Error: memory allocation failed, out of memory".* https://solana.stackexchange.com/questions/6876/program-logged-error-memory-allocation-failed-out-of-memory Accessed: 2024-10-28.
[14] 2023. *Solana: Past, Present, and Future.* https://research.nansen.ai/articles/solana-past-present-and-future Accessed: 2024-10-28.
[15] 2023. *Transaction 3HdLQWvpANEHTzhjUdQ71m81DegUCckpqXqgU2oWKBiCSfrdGuDqXFWg41Qidch26KwtjpRh-Vj7GqNvXHmGQ2njv.* https://bit.ly/4irDawA Accessed: 2024-10-28.
[16] 2024. *9uW2TqLyfYyrcNVrgCy4jPpqDKQoBZhXWypzzFxbixQE.* https://www.reddit.com/r/solana/comments/1cc9g05/9uw2tqlyfyyrcnvrgcy4jppqdkqobzhxwypzzfxbixqe/ Accessed: 2024-10-28.
[17] 2024. *Bots suspected of pushing Solana over Ethereum — Research.* https://cointelegraph.com/news/bots-pushing-solana-over-ethereum-research Accessed: 2024-10-28.
[18] 2024. *Common JSON Data Structures for Solana RPC Methods.* https://solana.com/docs/rpc/json-structures Accessed: 2024-10-28.
[19] 2024. *Consensus on Solana.* https://www.helius.dev/blog/consensus-on-solana Accessed: 2024-10-28.
[20] 2024. Fees on Solana. https://solana.com/docs/core/fees. Accessed: 2024-10-28.
[21] 2024. *Mainnet Beta Validators: Please upgrade to.* https://x.com/SolanaStatus/status/1800027464896328153 Accessed: 2025-01-28.
[22] 2024. *Memecoin Mania Drives Solana Toward All-Time Highs.* https://www.bloomberg.com/news/articles/2024-03-18/slerf-snap-memecoin-mania-drives-solana-toward-all-time-highs Accessed: 2024-10-28.
[23] 2024. *Raydium says Insufficient SOL balance if I try to buy anything more than 90% of my balance - any advice?* https://www.reddit.com/r/solana/comments/qo15jc/raydium_says_insufficient_sol_balance_if_i_try_to Accessed: 2024-10-28.
[24] 2024. Solana Account Model. https://solana.com/docs/core/accounts. Accessed: 2024-10-28.
[25] 2024. *Solana: Efficiently Minimizing Transaction Duration.* https://web3engineering.co.uk/solana-ioc-orders Accessed: 2024-10-28.
[26] 2024. *Solana MEV: An Introduction.* https://www.helius.dev/blog/solana-mev-an-introduction Accessed: 2024-10-28.
[27] 2024. *Solana MEV Bots: A Detailed Explanation.* https://www.coinfeeds.ai/crypto-blog/solana-mev-bots Accessed: 2024-10-28.
[28] 2024. *Solana Network Faces High Failure Rate in Transactions Amid Memecoin Mania.* https://coinmarketcap.com/academy/article/solana-network-faces-high-failure-rate-in-transactions-amid-memecoin-mania Accessed: 2025-01-28.
[29] 2024. *Solana struggles: Record 75% of user txs are failing... or are they?* https://cointelegraph.com/news/solana-struggling-record-seventy-five-percent-trasnactions-fail-memecoin-mania Accessed: 2024-10-28.
[30] 2024. *Solana Terminology.* https://solana.com/docs/terminology Accessed: 2024-10-28.
[31] 2024. *Solana Transactions.* https://github.com/solana-foundation/developer-content/blob/main/docs/core/transactions.md Accessed: 2024-10-28.
[32] 2024. *Solana Tx Fail Rate.* https://dune.com/scarn_eth/solana-tx-fail-rate Accessed: 2025-01-28.
[33] 2024. *Token Account Frozen! Anyone Know how to fix?* https://www.reddit.com/r/solana/comments/1ajnurc/token_account_frozen_anyone_know_how_to_fix/ Accessed: 2024-10-28.

[34] 2024. *Token swap error 0x26 InvalidSplTokenProgram on raydium rust.* https://solana.stackexchange.com/questions/13208/token-swap-error-0x26-invalidspltokenprogram-on-raydium-rust Accessed: 2024-10-28.
[35] 2024. *Transaction 26hZfeUESwWx8qykY9BbKEbadnhQSHmy1cH9DhrDBVJBh7d6Jh3LuqomqBoCetPuQ5xGN1j2eMWMs9fv8YgXKd8L.* https://bit.ly/4lEmHI3 Accessed: 2024-10-28.
[36] 2024. *Transaction 5AnmV4Sx6HiXWXzCzBkzu8NMas4HmAiV3ZxZ9H81Q8JegM7NJ8YVi9c1KVUT452unMXTfULGFkjZ2En6DYvd4Jv8.* https://bit.ly/3RqJWaI Accessed: 2024-10-28.
[37] 2024. *Transaction 5BB13YdeDP8ke37ZcU3yonEGsnVUKa24t5uctSRLQQupikejFtj2fcEwKVVSBUUfbbVxcJxKi9zVyyVB6KtQXCiv.* https://surl.li/goodup Accessed: 2024-10-28.
[38] 2024. *Transaction Confirmation and Expiration.* https://solana.com/docs/advanced/confirmation Accessed: 2024-10-28.
[39] 2024. *Transaction ebX6rwknxsXPvo1DDGq3far8uUtGywz2RBgLNZLBRF8Ws1pgmnstYVhsi2KQdTsXbvtWWt27PoittqgiUaEP8Dr.* https://bit.ly/4itS6Kl Accessed: 2024-10-28.
[40] 2024. *Transaction zxFnAz3WEYRJeKPtbFppM4SoEiRxvkJ9KqiZzdSbbQFGQf7sHJMEJJfgWYy7G6PMoj7K3VwV4QBu4CUr1miEB16.* https://bit.ly/42zBFGE Accessed: 2024-10-28.
[41] 2024. *Turbine: Block Propagation on Solana.* https://www.helius.dev/blog/turbine-block-propagation-on-solana Accessed: 2024-10-28.
[42] 2024. *What is PepperMints?* https://solanabox.tools/tools/peppermints Accessed: 2024-10-28.
[43] 2024. *What is this contract doing?...* https://www.reddit.com/r/solana/comments/1adoj2n/what_is_this_contract_doing Accessed: 2024-10-28.
[44] Dune Analytics. [n. d.]. Daily Transaction Failure Rate across Ethereum, Optimism and Arbitrum. https://dune.com/queries/2839305/4741938 Accessed: 2025-01-02.
[45] Maurice S Bartlett. 1946. On the theoretical specification and sampling properties of autocorrelated time-series. *Supplement to the Journal of the Royal Statistical Society* 8, 1 (1946), 27–41. doi:10.2307/2983611
[46] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why do my blockchain transactions fail? a study of hyperledger fabric. In *Proceedings of the 2021 international conference on management of data*. 221–234. doi:10.1145/3448016.3452823
[47] Wren Chan and Aspen Olmsted. 2017. Ethereum transaction graph analysis. In *2017 12th international conference for internet technology and secured transactions (ICITST)*. IEEE, 498–500. doi:10.23919/ICITST.2017.8356459
[48] Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. 2020. Understanding ethereum via graph analysis. *ACM Transactions on Internet Technology (TOIT)* 20, 2 (2020), 1–32. doi:10.1109/INFOCOM.2018.8486401
[49] Yan Chen and Cristiano Bellavitis. 2020. Blockchain disruption and decentralized finance: The rise of decentralized business models. *Journal of Business Venturing Insights* 13 (2020), e00151. doi:10.1016/j.jbvi.2019.e00151
[50] Ryan Chern. [n. d.]. *Solana Fees in Theory and Practice.* https://www.helius.dev/blog/solana-fees-in-theory-and-practice Accessed: 2025-01-02.
[51] Usman W Chohan. 2021. Non-fungible tokens: Blockchains, scarcity, and value. In *Non-Fungible Tokens*. Routledge, 1–11. doi:10.2139/ssrn.3822743
[52] Coinbase. [n. d.]. *What are Ethereum Layer-2 blockchains and how do they work?* https://www.coinbase.com/learn/crypto-basics/what-are-ethereum-layer-2-blockchains-and-how-do-they-work Accessed: 2025-01-02.
[53] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284. doi:10.1109/ESEM.2011.36
[54] Siwei Cui, Gang Zhao, Yifei Gao, Tien Tavu, and Jeff Huang. 2022. VRust: Automated vulnerability detection for solana smart contracts. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 639–652. doi:10.1145/3548606.3560552
[55] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*. IEEE, 910–927. doi:10.1109/SP40000.2020.00040
[56] Damiano Di Francesco Maesa, Andrea Marino, and Laura Ricci. 2017. An analysis of the bitcoin users graph: inferring unusual behaviours. In *Complex Networks & Their Applications V: Proceedings of the 5th International Workshop on Complex Networks and their Applications (COMPLEX NETWORKS 2016)*. Springer, 749–760. doi:10.1007/978-3-319-50901-3_59
[57] Fintan Duffy, Malika Bendechache, and Irina Tal. 2021. Can Solana's high throughput be an enabler for IoT?. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 615–621. doi:10.1109/QRS-C55045.2021.00094
[58] Youssef Faqir-Rhazoui, Miller-Janny Ariza-Garzón, Javier Arroyo, and Samer Hassan. 2021. Effect of the gas price surges on user activity in the daos of the ethereum blockchain. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–7. doi:10.1145/3411763.3451755

[59] Michael Fleder, Michael S Kester, and Sudeep Pillai. 2015. Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657* (2015). doi:10.48550/arXiv.1502.01657

[60] Liezl Gambe. [n. d.]. *Ethereum layer 2 networks struggle with increased transaction failures.* https://www.bitget.com/news/detail/12560604172286 Accessed: 2025-01-02.

[61] Jim Hendler. 2009. Web 3.0 Emerging. *Computer* 42, 1 (2009), 111–113. doi:10.1109/MC.2009.30

[62] Yuheng Huang, Haoyu Wang, Lei Wu, Gareth Tyson, Xiapu Luo, Run Zhang, Xuanzhe Liu, Gang Huang, and Xuxian Jiang. 2020. Understanding (mis) behavior on the eosio blockchain. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–28. doi:10.1145/3392155

[63] Hai Jin, Chenchen Li, Jiang Xiao, Teng Zhang, Xiaohai Dai, and Bo Li. 2022. Detecting arbitrage on ethereum through feature fusion and positive-unlabeled learning. *IEEE Journal on Selected Areas in Communications* 40, 12 (2022), 3660–3671. doi:10.1109/JSAC.2022.3213335

[64] Jupiter. [n. d.]. *Jupiter Swap API V4.* https://station.jup.ag/docs/legacy/apis/swap-api Accessed: 2024-10-28.

[65] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China) *(ICSE '06)*. ACM, New York, NY, USA, 492–501. doi:10.1145/1134285.1134355

[66] Ledger. [n. d.]. *Transaction failed - Out of Gas.* https://support.ledger.com/article/4406279901969-zd Accessed: 2025-01-02.

[67] Kai Li, Shixuan Guan, and Darren Lee. 2023. Towards understanding and characterizing the arbitrage bot scam in the wild. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 7, 3 (2023), 1–29. doi:10.1145/3626783

[68] Xiangyu Li, Xinyu Wang, Tingli Kong, Junhao Zheng, and Min Luo. 2021. From bitcoin to solana–innovating blockchain towards enterprise applications. In *International Conference on Blockchain*. Springer, 74–100. doi:10.1007/978-3-030-96527-3_6

[69] Xingyu Lyu, Mengya Zhang, Xiaokuan Zhang, Jianyu Niu, Yinqian Zhang, and Zhiqiang Lin. 2022. An empirical study on ethereum private transactions and the security implications. *arXiv preprint arXiv:2208.02858* (2022). doi:10.48550/arXiv.2208.02858

[70] Metamask. [n. d.]. *User Guide: Transactions and Failed Transactions.* https://support.metamask.io/manage-crypto/tokens/user-guide-transactions-and-failed-transactions/ Accessed: 2025-01-02.

[71] Thomas Niedermayer, Pietro Saggese, and Bernhard Haslhofer. 2024. Detecting Financial Bots on the Ethereum Blockchain. In *Companion Proceedings of the ACM on Web Conference 2024*. 1742–1751. doi:10.1145/3589335.3651959

[72] Gustavo A Oliva, Ahmed E Hassan, and Zhen Ming Jiang. 2020. An exploratory study of smart contracts in the Ethereum blockchain platform. *Empirical Software Engineering* 25 (2020), 1864–1904. doi:10.1007/s10664-019-09796-5

[73] Vinicius C Oliveira, Julia Almeida Valadares, Jose Eduardo A. Sousa, Alex Borges Vieira, Heder Soares Bernardino, Saulo Moraes Villela, and Glauber Dias Goncalves. 2021. Analyzing transaction confirmation in ethereum using machine learning techniques. *ACM SIGMETRICS Performance Evaluation Review* 48, 4 (2021), 12–15. doi:10.1145/3466826.3466832

[74] Chorus One. [n. d.]. *Transction Latency on Solana.* https://chorus.one/articles/transaction-latency-on-solana-do-swqos-priority-fees-and-jito-tips-make-your-transactions-land-faster Accessed: 2025-01-02.

[75] Michael Pacheco, Gustavo Oliva, Gopi Krishnan Rajbahadur, and Ahmed Hassan. 2023. Is my transaction done yet? an empirical study of transaction processing times in the ethereum blockchain platform. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–46. doi:10.1145/3549542

[76] Mahesh Pal. 2005. Random forest classifier for remote sensing classification. *International journal of remote sensing* 26, 1 (2005), 217–222. doi:10.1080/01431160412331269698

[77] Giuseppe Antonio Pierro and Roberto Tonelli. 2022. Can solana be the solution to the blockchain scalability problem?. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 1219–1226. doi:10.1109/SANER53432.2022.00144

[78] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest?. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 198–214. doi:10.1109/SP46214.2022.9833734

[79] Sven Smolka, Jens-Rene Giesen, Pascal Winkler, Oussama Draissi, Lucas Davi, Ghassan Karame, and Klaus Pohl. 2023. Fuzz on the beach: Fuzzing solana smart contracts. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1197–1211. doi:10.1145/3576915.3623178

[80] Solana. [n. d.]. *How to use Priority Fees on Solana.* https://solana.com/developers/guides/advanced/how-to-use-priority-fees Accessed: 2025-01-02.

[81] Solana Foundation. [n. d.]. *How to Optimize Compute Usage on Solana.* https://solana.com/developers/guides/advanced/how-to-optimize-compute Accessed: 2025-01-02.

[82] Solscan. [n. d.]. *Solscan Official Website.* https://solscan.io/ Accessed: 2025-04-08.

[83] Michael Spain, Sean Foley, and Vincent Gramoli. 2020. The impact of ethereum throughput and fees on transaction latency during icos. In *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. doi:10.4230/OASICS.TOKENOMICS.2019.9

[84] Donna Spencer. 2009. *Card sorting: Designing usable categories.* Rosenfeld Media.
[85] Bishenghui Tao, Hong-Ning Dai, Jiajing Wu, Ivan Wang-Hei Ho, Zibin Zheng, and Chak Fong Cheang. 2021. Complex network analysis of the bitcoin transaction network. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 3 (2021), 1009–1013. doi:10.1109/TCSII.2021.3127952
[86] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2021. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 428–445. doi:10.1109/SP40001.2021.00027

Received 2024-10-31; accepted 2025-03-31