Demystifying Solana Bots: From GitHub Blueprints to On-Chain Fingerprints*

Xiaoye Zheng
Zhejiang University
Hangzhou, China
xiaoyez@zju.edu.cn

Yujing Chen
Zhejiang University
Hangzhou, China
chenyujing@zju.edu.cn

Minghao Wu
Zhejiang University
Hangzhou, China
wuminghao@zju.edu.cn

David Lo
Singapore Management University
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

Daoyuan Wu
Lingnan University
Hong Kong, Hong Kong
daoyuanwu@ln.edu.hk

Xiaohu Yang
Zhejiang University
Hangzhou, China
yangxh@zju.edu.cn

Zhiyuan Wan†✉
Zhejiang University
Hangzhou, China
wanzhiyuan@zju.edu.cn

Abstract

Solana is an emerging blockchain platform designed for high throughput and low transaction fees, making it inexpensive to submit transactions at scale and, consequently, increasing exposure to bot spamming and related financial exploitation. Solana bots are typically off-chain software systems that operate in a competitive on-chain execution environment by constructing and submitting transactions, and the bot-related transactions on the decentralized exchanges exceed 250 million dollars in daily trading volume in January 2026. Prior studies on Solana have examined system performance, smart-contract security, and specific on-chain phenomena. However, we still lack a systematic understanding of what Solana bots implement in practice and how these implementations manifest as observable on-chain execution fingerprints. To address this gap, we performed a large-scale empirical study of Solana bots from two complementary views: (i) 586 bot repositories collected from GitHub, and (ii) 200 bot addresses on Solana, with over 44 million on-chain transactions. Our study derives an implementation-grounded taxonomy of Solana bots comprising 15 categories grouped into five domains (e.g., Trading Operations, MEV, and On-chain Analytics), identifies a largely shared five-stage operational pipeline manifested in bot implementations, and uncovers systematic variation in on-chain trading behaviors of Solana bots across diverse trading platforms and assets. Based on our findings, we highlight future research directions, and provide recommendations for building and operating bots on the Solana blockchain.

CCS Concepts

• **Software and its engineering** → **Software libraries and repositories**; • **Security and privacy** → **Economics of security and privacy**; • **General and reference** → **Empirical studies**.

Keywords

Solana, blockchain bots, MEV, bot taxonomy, operational pipeline

ACM Reference Format:
Xiaoye Zheng, Yujing Chen, Minghao Wu, David Lo, Difan Xie, Daoyuan Wu, Xiaohu Yang, and Zhiyuan Wan. 2026. Demystifying Solana Bots: From GitHub Blueprints to On-Chain Fingerprints. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3832783.3837522

1 Introduction

Solana is an emerging blockchain platform designed to offer high throughput with low transaction fees [73], making it an increasingly popular infrastructure for Web3 applications, including decentralized finance (DeFi) and non-fungible tokens (NFTs) [25, 27, 43]. Solana adopts a program-account design [5] that separates executable logic (programs, i.e., smart contracts on Solana) from mutable state (stored in on-chain accounts), enabling the runtime to execute non-conflicting transactions in parallel. Together with Proof of History and the Tower BFT consensus protocol [4], these design choices contribute to the high throughput and low transaction fees of Solana. As of May 2025, Solana had processed over 405 billion transactions, and supported more than 29.7 million fee-paying accounts [64], with total value locked in DeFi exceeding 9 billion USD [34].

*This research was supported by the National Science Foundation of China (No. 62472383), the Fundamental Research Funds for the Central Universities (No. 226-2025-00004), 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 the Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security.

ASE '26, Munich, Germany

ACM ISBN 979-8-4007-2882-2/2026/10
https://doi.org/10.1145/3832783.3837522

On the flip side, low transaction fees of Solana make it inexpensive to submit transactions at scale, exposing the network to bot spamming and related financial exploitation [3, 6]. In particular, the daily trading volume of decentralized exchange (DEX) attributed to trading bots has exceeded 250 million USD in January 2026 [14], and the revenue of these bots peaked at nearly 1.5 million USD in a single day in early 2024 [2]. Bots are typically off-chain software systems that interact with the blockchain by constructing and submitting transactions [58]. Accordingly, a bot rarely corresponds to a single on-chain entity; instead, it manifests through the set of accounts it controls and the on-chain records of the transactions it submits. Previous studies on Solana span system performance, smart contract security, and measurement of on-chain phenomena. Researchers have evaluated the scalability and transaction throughput [37, 61], and proposed vulnerability detection and testing techniques to improve smart contract security [31, 63]. More recently, large-scale empirical studies have measured diverse on-chain phenomena on Solana, including failed transactions [75], sandwiching attacks [40], meme coin markets [49], and rug pulls [19]. However, Solana bots remain underexplored as software artifacts from a software engineering perspective. We still lack a systematic understanding of what they implement in practice through their codebases, and the extent to which these implementations correspond to the on-chain fingerprints left by the transactions they submit. Such an understanding provides a complementary characterization of Solana bots from both implementation and execution perspectives, and provides practical insights into bot design choices and maintenance concerns in a competitive execution environment.

To address this gap, we conducted a large-scale empirical study of Solana bots to systematically characterize their functional categories, implementations in open-source codebases, and transaction-level fingerprints they leave on-blockchain. We curated a dataset comprising 586 Solana bot repositories on GitHub and 200 bot accounts from two widely used Solana bot services, with 44,118,825 on-chain transactions submitted by these accounts. Below, we present our research questions and highlight the key findings:

RQ1: What functionality taxonomy emerges from open-source Solana bot implementations? Prior studies often characterize blockchain bots (i.e., on Ethereum and Binance Smart Chain) through their on-chain behaviors or self-reported descriptions [22, 23, 47, 59], leaving a limited understanding of what functionalities are actually implemented by bots in open-source repositories. To establish an implementation-grounded view, we analyze 586 public Solana bot repositories on GitHub, and derive a functionality taxonomy spanning 5 domains and 15 categories, covering *Trading Operations*, *MEV* [13], *On-chain Analytics*, and *Tooling and Infrastructure*. The distribution of repositories across the resulting taxonomy indicates that open-source Solana bots are dominated by trading automation and MEV-oriented execution, with analytics and tooling/infrastructure categories constituting a smaller but diverse long tail.

RQ2: How are Solana bots implemented in open-source repositories? To complement the functionality taxonomy with an engineering perspective, we explore how Solana bot capabilities are realized in code across bot categories, by analyzing (i) code-level building blocks, (ii) recurring pipeline among these blocks, and (iii) the third-party dependencies underpinning such implementations. We observe a pronounced long tail in building-block coverage: 90% of the 2,044 building blocks appear in no more than 3.1% of repositories, and per-repository counts differ significantly across bot categories. Despite this heterogeneity, most repositories align with a shared backbone that can be abstracted into a five-stage canonical pipeline spanning *Setup*, *Observation and Acquisition*, *Analytics*, *Planning and Decision*, and *Execution and Reporting*, although the *Planning and Decision* stage is simplified or omitted in certain bot categories. Dependency analysis further reveals heavy reliance on core Solana SDKs (e.g., `@solana/web3.js` in 85.9% of JavaScript/-TypeScript repositories) and substantial technical lag, with over 30% of dependencies more than one year behind the latest versions.

RQ3: How do Solana bots manifest distinctive execution fingerprints on-chain? While the implementations and runtime of Solana bots reside off-chain, they are only observable on-chain through the accounts they control and the transactions they submit. Thus, we characterize the 200 bot-associated addresses in our dataset by extracting on-chain execution fingerprints from their transaction traces, capturing transaction-level features including submission intensity, execution effectiveness, cost, and trading assets. We observe four distinct execution clusters among the 200 bot-associated addresses, separated primarily along two axes: transaction submission intensity and execution effectiveness. Three clusters (56 addresses) exhibit execution characteristics consistent with *MEV* bots. Across the three clusters, WSOL [17] serves as the dominant pivot token, accounting for 70.9%–99.9% of their transactions; however, the three clusters diverge in the trading venues they engage with. Notably, the *MEV* bots whose transactions invoke proprietary AMMs [16] (e.g., HumidiFi) realize positive profit (62.3%) at three times the rate of those that do not, suggesting that interaction with proprietary AMMs may be associated with more favorable execution outcomes for MEV bots. The remaining cluster (102 addresses) exhibits characteristics consistent with the *Trading Operations* bots, with transaction activity concentrated on the Pump.fun platform (80.9%).

Based on our findings, we discuss the implications, such as recurring operational pipeline across bot implementations, substantial technical lag of third-party dependencies, and factors that affect the profitability of Solana bots. We also provide practical recommendations for bot design and maintenance, and highlight several research avenues. Our work makes the following contributions: (1) We compile two datasets relevant to Solana bots: 586 bot repositories on GitHub, and 200 bot-associated on-chain addresses, with 44,118,825 transactions. (2) We perform the first empirical study of Solana bots to characterize their functional categories, implementations in open-source codebases, and transaction-level fingerprints they leave on-blockchain. (3) We provide practical implications and outline future avenues for research.

2 Background

Solana bots submit transactions composed of instructions that invoke on-chain programs, exposing behavioral fingerprints across multiple dimensions, such as in program usage and fees.

Accounts and Programs. Solana adopts an account-based model, where *accounts*, identified by a 32-byte public key (address), store persistent state and executable logic [5]. A *program* in Solana is an executable account that contains program code. Solana programs

are stateless: they read and update state stored in (non-executable) data accounts passed in at execution time [1].

Transactions and Signers. A Solana transaction specifies a set of accounts and a sequence of instructions to be executed by on-chain programs [7]. To authorize state changes, a transaction must be signed by one or more *signer accounts* (i.e., accounts whose private keys approve the requested operations), and the corresponding signatures are included in the transaction message. A *finalized* transaction is ***successful*** if all instructions execute without error and ***failed*** otherwise. Both outcomes incur a base fee of 5,000 lamports per signature (0.000005 SOL) and may include an optional per-compute-unit priority fee for higher scheduling priority [42].

MEV. MEV refers to the maximum profit obtained by strategically including, excluding, or ordering transactions [32]. MEV opportunities often arise from order-dependent execution on decentralized exchanges (DEXs) and transient price discrepancies of digital assets such as cryptocurrencies across venues (e.g., between DEXs and centralized exchanges, CEXs). For instance, on mempool-based chains, traders can scan pending price-moving transactions and position their own transactions to capture the resulting price movement. Unlike Ethereum, Solana does not expose an publicly visible mempool. Instead, Solana adopts mempool-less transaction forwarding (i.e., Gulf Stream [65]), whereby transactions are proactively forwarded to upcoming slot leaders. Accordingly, MEV on Solana is shaped by latency advantages in reaching leaders and by specialized submission and ordering infrastructure. For instance, Jito provides low-latency transaction submission and bundle-based mechanisms [46].

3 Dataset

GitHub Repositories of Solana Bots. We collected 628 Solana bot via a two-round keyword search using the GitHub GraphQL API, by following a systematic dataset construction process [72]. Specifically, we queried repositories created between January 2021 and June 2025 using the GitHub GraphQL API by matching keywords in repository metadata (e.g., name, description, and topics) and README content. The initial query "*Solana bot*" returned 7,760 candidates. Following prior work [18], we applied automated repository-level filters to exclude non-code, link-curation/aggregator, low-engagement, and Solidity repositories, leaving 824 candidates. Two authors independently screened each candidate for inclusion using a shared guideline: we required evidence of (i) Solana-specific interaction from README, dependencies, or source code, and (ii) automation-oriented execution (e.g., monitoring loops or automated actions), yielding 536 Solana bot repositories. Moreover, to reduce keyword bias, we derived three additional query keywords from repository topics observed in the 536 repositories ("*Solana bundler*", "*Solana automation*", and "*Solana agent*"), which returned 3,046 candidates. After applying the same filtering and screening procedures, we obtained 92 additional Solana bot repositories. We then excluded 42 repositories that could not be processed by an open-source parser generator tool TREE-SITTER [68] for further investigation, due to missing source code or unsupported languages, resulting in a final dataset of 586 repositories. Among the 586 Solana bot repositories, TypeScript accounts for the largest share (48.2%), followed by Python (17.9%), JavaScript (15.5%), Rust (14.9%), and Go (1.9%). Repository popularity varies widely, with a median of 18 stars (mean: 62, min: 2, max: 2,220, std.: 159), and 7 forks (mean: 27, min: 2, max: 1,019, std.: 70).

On-chain Addresses of Solana Bots. ❶ ***Baseline Window.*** To collect on-chain Solana bot addresses, we started from two widely used bot services on Solana, *Trojan* [69] and *SolanaMevBot* [66]. Trojan is a Telegram-based bot service that supports automated trading, sniping, and copy trading, and is listed among the top services in the DEX trading-bot leaderboard on Dune Analytics [71]. SolanaMevBot is listed among the top services in the arbitrage leaderboard on circular.bot [28]. Some high-ranked services (e.g., *NotArb* on circular.bot) do not provide publicly accessible on-chain bot addresses, and are therefore excluded from our analysis. For each selected bot service, we collected the top-100 bot addresses on Solana reported by the corresponding leaderboard. For each address, we leveraged the Dune SQL engine [38] to retrieve its on-chain transaction records over a one-month window from October 1, 2025, to November 1, 2025, including transaction signatures, execution status, invoked program IDs, transaction fees, and token balance changes, as well as-level traces. The transaction volume per address during our one-month observation window varies substantially across bot services. Bot addresses from SolanaMevBot range from 5 to 6,742,370 transactions (median: 2,121, mean: 439,061). In contrast, bot addresses from Trojan range from 64 to 28,274 transactions (median: 1,366, mean: 2,127). ❷ ***Validation Window.*** We constructed an additional validation dataset over a one-month window from February 1, 2026, to March 1, 2026. Specifically, we collected the top-100 bot addresses reported for SolanaMevBot on circular.bot, as well as the top-100 addresses reported for Axiom on Dune Analytics [11] during this window. Axiom is a browser-based Solana trading-bot service that was listed among the top services in the DEX trading-bot leaderboard on Dune Analytics during this period. For each address, we applied the same Dune SQL-based data collection procedure to retrieve its on-chain transaction records. See Appendix A.2 for details.

4 RQ1: Taxonomy

To answer RQ1, we constructed a taxonomy of Solana bot repositories to systematically capture the functionalities implemented by these bots. We use Solana bot repositories on GitHub as inputs because they provide inspectable and versioned implementations of bot functionalities, allowing the taxonomy to be grounded in what is actually implemented rather than what is merely described.

4.1 Methodology

We applied an LLM-assisted open card sorting process to construct a taxonomy of functionalities implemented in Solana bot repositories on GitHub, because manually analyzing repository codebases is impractical given their scale and heterogeneity [35, 60]. Specifically, we decomposed the process into four steps and used an LLM (DeepSeek v3.2) as an auxiliary tool for code summarization and higher-level semantic abstraction. The full set of prompts is provided in the replication package.

❶ **Function-Level Card Preparation.** We used TREE-SITTER to extract all functions from each repository. For each extracted function, we provided its source code as input to the LLM, and obtained a concise natural language summary. These function-level summaries serve as cards in the subsequent open card sorting process.

❷ **Repository-Level Summarization.** For each repository, we aggregated its set of function-level cards, and prompted the LLM to generate descriptive functionality tags, along with the proportion of the function-level cards in the repository associated with each tag. These tags summarize salient functional aspects of the repository as a whole and facilitate higher-level grouping and taxonomy construction. As a result, we generated an average of 6.3 tags per repository (min = 5, median = 6, max = 9, std. = 1). ❸ **Taxonomy Construction.** To construct the taxonomy, we first consolidated synonymous tags across repositories (e.g., *solana-sniper-bot* and *sniping-bot*), yielding a canonicalized tag set (N = 372). Two of the authors then independently performed open card sorting on the canonicalized tag set. The two initial taxonomies converged at the domain level, differing in category granularity (e.g., one taxonomy merging semantically related MEV-oriented categories that the other kept separate) and category naming (e.g., differently worded labels covering the same underlying tag groups). The two authors reconciled these differences into a finalized taxonomy of 15 categories. Using finalized taxonomy, the two authors independently assigned the 372 canonicalized tags to one of the 15 categories; a tag was counted as agreed upon if both authors assigned it to the same category. These tag-level assignments achieved substantial inter-rater agreement (Cohen's $\kappa = 0.78$). Disagreements were resolved through discussion between the two authors; unresolved cases were independently reviewed by a third author, and the majority decision was adopted as the final assignment. ❹ **Repository Categorization.** We applied the taxonomy by mapping each repository to one or more taxonomy categories based on its repository-level tags. As a result, each repository is associated with 1 to 7 categories, while 16 repositories cannot be mapped to any category. As a sanity check, we randomly sampled 100 out of 586 repositories, allocating the sample across the 15 categories in proportion to the square root of each category's size. The strategy oversamples smaller categories compared to simple proportional allocation to balance estimation reliability across strata of varying sizes [29]. Two authors independently reviewed each sampled repository's README and source code, and judged whether the repository's dominant category accurately reflected a primary functionality of the repository, with substantial inter-rater agreement (Cohen's $\kappa = 0.81$). Note that we define the dominant category of a repository as the category of its highest-weight tag (i.e., the tag covering the largest proportion of function-level cards). Disagreements were resolved through discussion between the two authors until consensus was reached. 91 of the 100 sampled repositories were confirmed to have a dominant category that accurately reflected a primary functionality of the repository, with per-category accuracy ranging from 75% to 100%. See Table 11 in the Appendix for details. We further grouped repositories by dominant category for subsequent analysis.

4.2 Results

Table 1 presents the taxonomy of Solana bot functionalities derived from the open card sorting process based on our dataset of Solana bot repositories on GitHub. The taxonomy summarizes how Solana bots group around recurring functional themes, and organizes the observed repository functionalities of Solana bots into 5 higher-level domains and 15 categories, with each category accompanied by representative tags drawn from the repositories assigned to that category (the *Tags* column), and a representative repository for the category. For a detailed description of each category, please refer to Table 4in the Appendix [10].

We further conducted a qualitative analysis of the most-starred repository in each of the top three categories (ranked by repository count), focusing on their core functionalities. Below is a representative repository (additional examples are provided in the Appendix):

🤖 The `0xNineteen/solana-arbitrage-bot`, written in Rust and TypeScript, exemplifies the *Arbitrage* category, which combines (1) an off-chain client that searches for arbitrage opportunities and assembles transactions, and (2) an on-chain program that enforces atomic execution and *profit-or-revert* semantics of transactions. Specifically, the off-chain client searches for profitable swap paths across multiple DEX liquidity pools (e.g., Orca, Mercurial, and Saber), and assembles the corresponding swap instructions into a single transaction. The on-chain program validates the profitability of the transaction by tracking realized balances across hops, and reverts the entire transaction unless the final balance of the source token exceeds its initial balance. Moreover, we also observed multiple heterogeneous repositories whose functionalities span multiple taxonomy categories. Taking the `sendaifun/solana-agent-kit` repository as an example, it combines bot-like autonomous execution with tooling and infrastructure support, spanning the *trading operations* (🤖) and *tooling and infrastructure* (⚙) categories. On the one hand, it supports bot-like automation of on-chain trading-related actions (e.g., token swaps and other DeFi interactions). On the other hand, it provides a broad range of infrastructure and utility capabilities beyond trading, such as wallet management and NFT-related operations. Notably, it integrates with agent runtimes (e.g., LangChain and Vercel AI) by exposing on-chain actions through a unified interface, enabling agent planning modules to invoke both trading and infrastructure operations.

> **Finding 1:** We identify a taxonomy of Solana bot functionalities comprising 5 higher-level domains and 15 categories. Open-source Solana bot implementations emphasize automating order placement and reacting to time-sensitive signals.

5 RQ2: Implementation

To address RQ2, we explore GitHub repositories of Solana bots from an implementation perspective, focusing on the code-level building blocks of Solana bots and how these blocks are composed into typical bot pipelines. We also investigate the usage of third-party libraries, which reflect how bot pipeline components are implemented in practice.

5.1 Methodology

5.1.1 Code-Level Building Block Identification. We used the 78,300 functions extracted from the repositories in our dataset via TREE-SITTER, and grouped them into code-level building blocks based on lexical and semantic similarity, by following the steps below.

❶ **Function Embedding.** For each function, we preprocessed its code by removing comments and normalizing literals (e.g, numbers and strings) and identifiers (e.g., variable and parameter names). We then concatenated the function name and the normalized function body as the input, and used the Qwen3-Embedding-0.6B model [74] to obtain an embedding vector for the function, with the output

Table 1: Taxonomy of Solana Bot Repositories (Repos) on GitHub.

Category (# Repos)	Example Tags	Representative Repo
Trading Operations		
Order Execution and Management (137)	*trading-bot, automated-trading, transaction-execution, limit-orders, telegram-trading-bot*	`nmweaver/soltrade` (342 ★)
Liquidity Provision and Yield Farming (12)	*liquidity-management, bonding-curve, liquidity-bot, automated-market-maker, liquidity-pools*	`edwin-finance/meteora-liquidity-rebalancer` (4 ★)
Copy Trading (33)	*copy-trading, copy-trading-bot*	`cryptole0/Copy-Trading-Bot-Rust` (96 ★)
MEV		
Arbitrage (47)	*arbitrage-bot, flash-loan-arbitrage, cross-pool-arbitrage, cross-exchange-arbitrage, dex-cex-arbitrage*	`0xNineteen/solana-arbitrage-bot` (799 ★)
Sniping (134)	*sniper-bot, pumpfun-bot, pumpfun-trading-bot, memecoin-trading, token-sniper*	`warp-id/solana-trading-bot` (2,220 ★)
Transaction Ordering Exploitation (41)	*mev-bot, jito-bundle, transaction-bundling, pumpfun-bundler, sandwich-attack*	`jito-labs/mev-bot` (1,150 ★)
Market Manipulation		
Wash Trading (21)	*volume-bot, volume-simulation*	`web3batman/Raydium-Volume-Bot` (67 ★)
On-chain Analytics		
Monitoring and Alerting (26)	*real-time-alerts, token-monitoring, real-time-monitoring, wallet-tracking, transaction-monitoring*	`FriedDev/solana-rug-checker` (29 ★)
Portfolio and Risk Management (14)	*risk-management, risk-analysis, portfolio-management, simulation, data-analysis*	`henrytirla/Solana-PNL-Bot` (66 ★)
Tooling and Infrastructure		
Token Issuance and Administration (8)	*token-launchpad, token-creation, token-launch, token-management, spl-token*	`0xNevo/Solana_Token_Freezer` (14 ★)
NFT Minting and Marketplace Trading (14)	*nft-sales-bot, nft-tools, nft-minting-bot, nft, nft-bot*	`theskeletoncrew/air-support` (178 ★)
DEX Swap Routing and Integration (30)	*raydium-integration, jupiter-aggregator, dex-integration, jupiter-swap, dex-aggregator*	`YZYLAB/solana-swap` (130 ★)
Wallet Management (6)	*wallet-management, multi-wallet, token-transfer, account-management, keypair-management*	`LeaderMalang/Solana-Sweeper-Bot` (9 ★)
On-chain Data Pipeline (10)	*rpc-client, data-aggregation, birdeye-api, geyser-client, helius-webhook*	`weeaa/goyser` (66 ★)
Messaging Integration and Alerts (37)	*telegram-bot, twitter-integration, discord-bot, chat-interface, discord-notifications*	`KingJiongEN/DegentGroup` (65 ★)

dimension of 1,024. Qwen3-Embedding-0.6B is a lightweight embedding model that excels at code retrieval and text clustering on benchmarks, while offering a strong balance between effectiveness and efficiency for large-scale embedding scenarios [51, 57], making it well-suited for large-scale function embedding.

❷ **Function Clustering.** Based on the generated function embeddings, we clustered functions using a UMAP-HDBSCAN pipeline. UMAP [54] is a nonlinear dimensionality reduction method that projects high-dimensional embeddings into a lower-dimensional space, while largely preserving local neighborhood relationships. HDBSCAN [20] is a non-parametric clustering algorithm, which has demonstrated its efficacy for code clustering in recent studies (e.g., [26, 50, 52]). In the UMAP-HDBSCAN pipeline, HDBSCAN leverages the local neighborhood structure preserved in the UMAP-reduced space to identify coherent clusters without predefining the number of clusters, which makes the pipeline well-suited to function-level embeddings that often exhibit uneven distributions and outliers. In particular, we first applied UMAP to learn a lower-dimensional manifold representation of the function embeddings, and then ran HDBSCAN in this space to discover density-based clusters and label outliers as noise. Moreover, we tuned the hyperparameters of the UMAP-HDBSCAN pipeline via grid search to maximize the Silhouette score while constraining the fraction of functions labeled as noise. Specifically, we tuned *n_neighbors* and *n_components* of UMAP as well as *min_cluster_size* and *min_samples* of HDBSCAN, and set them to 10, 10, 5, and 5, respectively. As a result, we identified 5,007 function clusters, with a Silhouette score of 0.82, suggesting well-separated clusters. ❸ **Cluster Labeling and Consolidation.** For each function cluster, we aggregated the function-level summaries (generated in RQ1) of the functions in the cluster, and used an LLM (DeepSeek-v3.2) to generate one label in a constrained `[object]-[operation]` format (e.g., `token-swap`). To improve reproducibility, we fixed the model version and decoding configuration (e.g., *temperature* = 0) and recorded the prompts and generated labels. To validate labeling quality, we randomly sampled 95 clusters from the 5,007 function clusters, which provides a margin of error of approximately 10% at the 95% confidence level. Two of the authors independently reviewed the top-10 representative functions in each sampled cluster, ranked by HDBSCAN membership probability, and judged whether the generated label captured the functionality commonly observed across the 10 representative functions, based on manual inspection of each function's source code. Cohen's kappa over these independent judgments was $\kappa = 0.71$, indicating substantial agreement. Disagreements were resolved through discussion between the two authors until consensus was reached. After reconciliation, 84 of the 95 sampled clusters were confirmed to have labels that accurately described their functionality. Finally, we merged clusters with semantically overlapping labels into code-level building blocks (e.g., `transaction-construction` vs `transaction-creation`), yielding 2,003 building blocks in total.

5.1.2 Bot Pipeline Derivation. We derived bot pipelines that summarized how the identified code-level building blocks are composed in practice across bot categories. Following a bottom-up hierarchical abstraction approach inspired by prior work [53], we modeled a pipeline as an ordered sequence of stages, where each stage grouped building blocks that served a coherent architectural responsibility. Specifically, for each bot category, we selected multiple representative repositories, and inspected the building blocks in each repository. We then grouped these building blocks into pipeline stages primarily based on architectural responsibilities reflected by repository modules or components, and used their frequent co-occurrence within repositories of the same category as supporting evidence. We further consulted call or usage relations of building blocks when available to infer and validate plausible stage ordering.

5.1.3 Third-Party Library Usage Characterization. We first extracted the dependency entries from the language-specific dependency manifests of bot-related repositories in our dataset, i.e., `package.json`, `requirements.txt`, `Cargo.toml`, and `go.mod` for repositories written in JavaScript/TypeScript, Python, Rust, and Go, respectively. Each dependency entry specifies a library (by name) and, when provided, its version constraint. Next, to restrict our analysis to third-party dependencies, we excluded language-provided modules and packages (i.e., built-in modules or standard libraries) from the extracted library lists. Specifically, we obtained the built-in modules and standard libraries from official runtime APIs and toolchain commands, including `module.builtinModules` for JavaScript/TypeScript, `sys.stdlib_module_names` for Python, and `go list std` for Go. For Rust, we removed compiler-provided standard-library crates, e.g., `std`, `core`, `alloc`, and `proc_macro`. As a result, we obtained 15,524 dependency entries spanning 3,194 distinct third-party libraries. The average number of repositories using each third-party library is 4.86 (median = 1, min = 1, max = 328, and std = 15), indicating a highly right-skewed distribution.

Moreover, we characterized the up-to-dateness of third-party library usage in Solana bot repositories by measuring the *technical*

(a) Overall **(b) By bot category**

Figure 1: Repository coverage distribution per building block.

Table 2: Top-10 building blocks by repository coverage.

Building Block	Description	Repo Coverage (n, %)
`trade-execution`	Executes token swap or order operations.	333 (58.5%)
`token_data-fetching`	Retrieves on-chain token state, including price, supply, and identifiers.	267 (46.9%)
`transaction-construction`	Constructs transaction or instruction objects for on-chain operations.	257 (45.2%)
`transaction-submission`	Submits signed transactions to Solana network for on-chain execution.	215 (37.8%)
`pool_data-fetching`	Retrieves the configuration and state of a liquidity pool.	210 (36.9%)
`token_account_data-fetching`	Retrieves token account state, including balance, mint, and ownership.	157 (27.6%)
`market_data-fetching`	Retrieves aggregated market-level data, such as price feeds, trading volume, or cross-venue liquidity metrics.	150 (26.4%)
`client-creation`	Initializes RPC/API clients for interacting with Solana nodes or programs.	148 (26.0%)
`bundle-submission`	Bundles and submits multiple transactions.	146 (25.7%)
`wallet_data-fetching`	Retrieves wallet-level state (e.g., token balances).	124 (21.8%)

lag [30, 41] of each dependency entry. For each dependency entry, we resolved an *effective version* as the latest release that satisfies the declared version constraint, and computed *technical lag* as the time gap between the release date of this effective version and the release date of the latest available release of the same library.

5.2 Results

5.2.1 Code-Level Building Blocks in Bot Repositories. Figure 1a presents the cumulative distribution of repository coverage across building blocks. Overall, the repository coverage of building blocks follows a pronounced long-tail distribution, as shown in Figure 1a. Specifically, 90% of building blocks appear in no more than 3.1% of Solana bot repositories, indicating that most building blocks have very limited repository coverage. In contrast, a small minority of building blocks attains substantially higher coverage across bot repositories, forming a clear head–tail pattern. Table 2 makes the head of the distribution explicit by listing the ten most prevalent building blocks ranked by repository coverage. The most prevalent building block, `trade-execution`, appears in 58.5% of repositories, followed by `token_data_fetching` (46.9%) and `transaction-construction` (45.2%). Even the tenth-ranked building block is present in 21.8% of repositories, indicating that no building block approaches near-universal presence. The head concentrates on two capability families, (1) data acquisition (e.g., `token` / `pool` / `market` / `wallet_data-fetching`), and (2) the transaction pipeline (`transaction-construction` / `submission`, and `trade-execution`), revealing a common "observe–trade" backbone across bot repositories. The presence of `bundle-submission` among the most prevalent building blocks further indicates explicit support for batched/bundled submission beyond standard transaction submission.

As illustrated in Figure 1b, the average number of building blocks per repository varies across bot categories, ranging from 16 for the *Monitoring and Alerting* bots to 51 for the *Portfolio and Risk Management* bots. We applied the Kruskal–Wallis test and observed a statistically significant difference in building-block counts across bot categories ($H = 43.172$, $p = 8.152 \times 10^{-7}$). Moreover, the counts of building blocks per repository are decoupled from the degree of sharing in building blocks across repositories, as shown in Figure 1b. Specifically, building-block composition across bot categories exhibits three distinct types, including (1) ***low-count, core-shared composition***, e.g., *Wash Trading* bots contain an average of 17 building blocks, with 18% of the building blocks appearing in more than 30% of the corresponding repositories; (2) ***low-count, weakly shared composition***, e.g., *Monitoring and Alerting* and *Sniping* bots both remain small in size, with 16 and 26 building blocks per repository on average, yet less than 1% of their building blocks appear in more than 30% of the corresponding repositories; and (3) ***high-count, long-tail–dominated composition***, e.g., *Arbitrage bots* exhibit large block count (48 in average), while over 80% of their building blocks appear in fewer than 10% of repositories.

> **Finding 2:** The repository coverage of building blocks exhibits a long-tail distribution, suggesting an "observe-trade" backbone across bot repositories. The building-block counts per repository differ statistically significantly across bot categories.

5.2.2 Bot Pipelines. Figure 2 summarizes an abstracted, stage-based pipeline view across Solana bot categories, highlighting both shared components and category-level variations of Solana bots. At a high level, the pipeline of Solana bots is organized into five stages: (1) ***Setup***, which covers client initialization and configuration (e.g., establishing connections to Solana RPC services); (2) ***Observation and Acquisition***, which includes on-chain monitoring (e.g., liquidity pools, accounts, and events such as new token listings) and data fetching (e.g., token, pool, and market data); (3) ***Analytics***, which covers data processing and analysis (e.g., analyses of tokens, market and transaction behavior); (4) ***Planning and Decision***, which covers pre-execution logic such as strategy search and risk-reward evaluation; and (5) ***Execution and Reporting***, which includes capital allocation, trade execution, and post-trade reporting/notification.

Several components recur across bot categories, such as the ***Data Acquisition***, ***Data Analytics***, and ***Trade Execution*** modules, indicating a common execution backbone of Solana bots. We also observe that the ***Trade Execution*** module is decomposed into transaction construction (e.g., `array-creation` and `transaction-construction`) and transaction submission, where Solana bot repositories tend to adopt two different submission modes, including direct transaction submission (e.g., `transaction-submission`) and bundle-based submission (e.g., `bundle-submission`).

Differences across bot categories primarily arise from which functional components are traversed and how building blocks are composed within stages. Bots in some categories tend to compose additional components in the ***Planning and Execution*** stage, such as Strategy Search in *Arbitrage* bots, Risk-Reward Evaluation in *On-chain Analytics* bots. In contrast, bots in other categories reach *Trade Execution* with fewer intermediate components, often bypassing the ***Planning and Execution*** stage, and reach ***Trade Execution*** with fewer intermediate components, e.g., those in the *Order Execution and Management* and *Transaction Ordering Exploitation* categories.

> **Finding 3:** Solana bots follow a five-stage pipeline at a high level. Category-level variations arise from which functional components are traversed and how building blocks are composed within stages.

5.2.3 Third-Party Library Usage. Table 3 presents the five most frequently used third-party libraries in Solana bot repositories across programming languages. We made the following observations: (1)

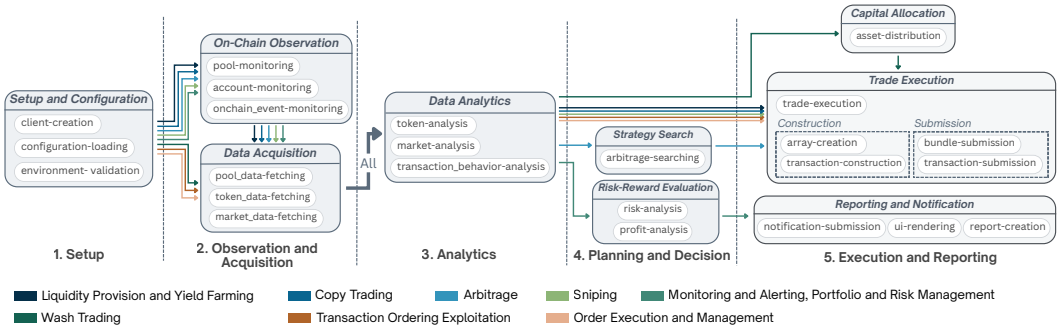

Figure 2: Bot pipeline across categories, organized into five stages.

Table 3: Top five third-party libraries frequently used by Solana bot repositories, grouped by programming language.

Programming Language	Library	Repo Coverage (n, %)	Technical Lag
JavaScript / TypeScript	`@solana/web3.js`	328 (85.9%)	125 days
	`dotenv`	292 (76.4%)	223 days
	`@solana/spl-token`	263 (68.8%)	0 days
	`bs58`	247 (64.7%)	857 days
	`axios`	231 (60.5%)	0 days
Rust	`solana-sdk`	66 (77.6%)	310 days
	`serde`	65 (76.5%)	0 days
	`serde_json`	62 (72.9%)	0 days
	`tokio`	61 (71.8%)	0 days
	`anyhow`	56 (65.9%)	0 days
Python	`requests`	55 (75.3%)	446 days
	`solana`	39 (53.4%)	572 days
	`python-dotenv`	37 (50.7%)	642 days
	`solders`	35 (47.9%)	611 days
	`base58`	34 (46.6%)	0 days
Go	`solana-go`	8 (80.0%)	386.5 days
	`binary`	6 (60.0%)	446 days
	`godotenv`	6 (60.0%)	140 days
	`grpc`	6 (60.0%)	679 days
	`testify`	5 (50.0%)	584 days

Dominance of the Solana client SDK. Solana client SDK libraries are widely adopted across languages, with coverage of 85.9% for `@solana/web3.js` in JavaScript/TypeScript, 77.6% for `solana-sdk` in Rust, 53.4% for `solana` in Python, and 80.0% for `solana-go` in Go repositories, respectively. (2) ***Frequent adoption of general-purpose libraries.*** For configuration loading, `dotenv` appears in 76.4% of JavaScript/TypeScript repositories, and `python-dotenv` appears in 50.7% of Python repositories. For HTTP communication, the coverage levels are 60.5% for `axios` in JavaScript/TypeScript repositories and 75.3% for `requests` in Python repositories. For Base58 encoding and decoding, `bs58` is adopted by 64.7% of JavaScript/TypeScript repositories, and `base58` is adopted by 46.6% of Python repositories. (3) ***Differences in coverage uniformity across languages.*** In Rust repositories, the top libraries exhibit relatively similar coverage levels (71%–78%): 77.6% for `solana-sdk`, 76.5% for `serde` and 71.8% for `tokio`. In contrast, in Python repositories, Solana-related libraries, i.e., `solana` (53.4%) and `solders` (47.9%), have substantially lower coverage as compared to the HTTP client library `requests` (75.3%).

Regarding dependency technical lag, the median lag across all third-party library dependencies in Solana bot repositories is 23 days; more than 30% of dependencies lag by over one year, indicating substantial dependency staleness for a non-trivial portion of dependencies. In terms of frequently used third-party libraries, technical lag varies substantially across their dependencies in Solana bot repositories of different programming languages, as shown in the *Technical Lag* column in Table 3. In particular, the dependencies of seven top-used libraries have a median technical lag of 0 days, whereas the dependencies of nine have a median technical lag exceeding one year. Across programming languages, Rust bot repositories show concentrated technical lag primarily in the core SDK (`solana-sdk`: 310 days), whereas Python bot repositories exhibit more systematic technical lag among their top-used library dependencies. In particular, four of the top five libraries used by Python bot repositories have a median technical lag exceeding one year, i.e., `python-dotenv` (642 days), `solders` (611 days), `solana` (572 days), and `requests` (446 days), reflecting notable dependency staleness in Python bot repositories.

In addition to programming language, dependency lag also differs significantly across bot categories. Repositories were grouped by their dominant taxonomy category, with repository-level lag summarized as the median across third-party dependencies. The differences are statistically significant (Kruskal–Wallis, $H = 48.029$, $p = 1.3 \times 10^{-5}$). *NFT minting and marketplace trading* bots exhibit the highest median repository-level lag (433.9 days), followed by *arbitrage* bots (391.2 days), whereas several categories, including *copy trading*, *sniping*, and *monitoring and alerting*, have a median lag of 0 days, suggesting that maintenance practices vary substantially across functional roles (see Table 5 in the Appendix).

We further examined trading-venue-specific package dependencies across bot repositories, such as `@raydium-io/raydium-sdk` for Raydium, `@jup-ag/core` for Jupiter, `@orca-so/common-sdk` for Orca, and `@meteora-ag/dlmm` for Meteora. We observe that Raydium-related dependencies are the most widely adopted, appearing in 195 bot repositories (33.3%), substantially exceeding Jupiter (40 repositories), Orca (45), Pump.fun (36), and Meteora (30). Beyond overall adoption, the breadth of venue integration also varies substantially across bot categories. In particular, *Arbitrage* bot repositories most frequently rely on multiple venues for trading, with 24% depending on two or more venues (average: 0.88), consistent with their need to identify and exploit price discrepancies across venues. In contrast, *Monitoring and Alerting* bot repositories exhibit much narrower venue integration, with only 5.5% depending on multiple venues (average: 0.41), reflecting a more specialized, single-venue trading strategy.

> **Finding 4:** Solana bot repositories show concentrated dependence on a small set of third-party libraries. More than 30% of dependencies lag by over one year. Among frequently used third-party libraries, technical lag varies substantially across programming languages and repositories.

6 RQ3: On-chain Fingerprints

Understanding the on-chain fingerprints of Solana bots requires examining how bot-associated addresses execute transactions under competitive on-chain conditions, e.g., contention for block inclusion and transaction execution. Per-address transaction traces are

often noisy, and the underlying intent of bots as operationalized by the proposed taxonomy in RQ1 could only be indirectly inferred from execution traces. To mitigate noise and avoid relying on explicit intent labels, we cluster bot addresses with similar execution patterns and characterize on-chain fingerprints at the cluster level.

6.1 Methodology

6.1.1 Address Clustering. We first represented each bot-associated address as a feature vector capturing its execution characteristics. In line with prior work [75], we adopted two intensity features (*transaction frequency* and *transaction volume*), and extended them with three additional features capturing execution effectiveness, cost, and breadth (*transaction success rate*, *transaction fee*, and *asset diversity*). Specifically, for each address, we measured *transaction frequency* as the mean inter-transaction interval between transactions initiated by the address; *transaction volume* by the total number of initiated transactions and the average number of initiated transactions per block over the analysis window; *transaction success rate* as the fraction of successful initiated transactions, where a transaction was marked as successful if its `error` field is null and failed otherwise; *transaction fee* as the total fee summed across all initiated transactions, extracted from the `fee` field; and *asset diversity* as the average number of distinct tokens involved per transaction initiated by the address. All features are standardized using z-scores prior to clustering, with missing or non-finite values imputed as 0. Next, we clustered the bot-associated addresses in our dataset using HDBSCAN [20], as it requires no predefined cluster count and handles noise, making it suitable for unknown behavioral bot diversity in our dataset. The model yielded four clusters of 33, 11, 12, and 102 addresses, and labeled 42 addresses as noise, achieving a silhouette score of 0.6588.

6.1.2 On-chain Behavior Characterization. We began with a cluster-wise profitability analysis, and restricted the analysis to transactions where Wrapped SOL (WSOL) serves as the base asset, to enable fair comparison across clusters. WSOL is the tokenized representation of native SOL, pegged 1:1, which enables SOL to participate in token-based programs and DEX swaps. Specifically, for each cluster of bot addresses, we examined the distribution of address-level realized profits in SOL-denominated units, which were derived from per-transaction balance deltas between the `pre_token_balances` and `post_token_balances` fields in transactions.

To interpret the resulting clusters, we further characterize each cluster using observable on-chain operational behaviors, including: (i) **Trading venues.** We extract the program ID(s) invoked by each transaction, from both top-level and inner instructions, and map program IDs to venues using Solscan annotations; (ii) **Traded tokens.** We identify the token accounts referenced in swap-related instructions (e.g., input and output token accounts) and extract their mint addresses to determine the traded tokens; (iii) **Transfer outflows.** We extract recipients from native `transfer` instructions in the System Program and from `transfer`/`transferChecked` instructions in the Token Program.

To captures economically meaningful transfer outflows that reflect external execution dependencies, rather than internal fund shuffling or protocol-mandated state transitions, we exclude: (i) intra-transaction transfers that form a closed value loop (i.e., outflows fully returned within the same transaction), (ii) transfers to bot-controlled addresses used for self-funding (i.e., identified based on publicly annotated funding relationships at the time of data collection), and (iii) transfers involving protocol state accounts, such as liquidity pool or bonding-curve accounts labeled by Solscan.

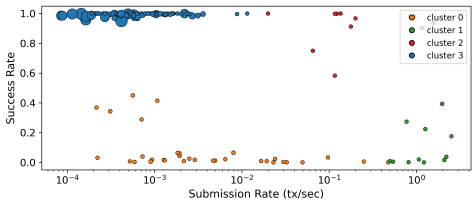

Figure 3: Transaction success rate vs. submission rate across clusters (log-scaled x-axis; bubble size: mean fee per address).

6.1.3 Validation across Time Windows. To assess whether the identified behavioral clusters generalize beyond the baseline window and the originally selected bot services, we projected the bot addresses collected during the validation window onto the baseline cluster space, rather than reclustering them. The design of projection enables us to evaluate whether the behavioral clusters learned from the baseline window remains applicable to unseen addresses collected in a later time window and from an additional bot-service source. Specifically, for each validation address, we computed the same address-level features described in Section 6.1.1. We then assigned each address to the cluster label of its five nearest neighbors in the baseline clustering using a 5-nearest-neighbor classifier with majority voting. Ties were broken by summing inverse distances separately for each tied label and choosing the label with the largest sum [36]. Next, to assess whether the behavioral characteristics of the identified clusters remain stable between the baseline and validation windows, we first characterized the validation addresses assigned to each cluster using the same behavioral dimensions that characterize the baseline clusters, as described in Section 6.1.2. We then compared the resulting characteristics of validation clusters with those of the corresponding baseline clusters in terms of DEX venues, traded tokens, and major outflow recipients.

6.2 Results

6.2.1 Cluster-level Execution Characteristics. Figure 3 visualizes the distribution of bot addresses in the success–frequency space of transaction execution. Rather than forming a single continuous trade-off, the clusters of bot addresses partition the space into four distinct execution regimes, which form a structured 2×2 landscape defined jointly by submission intensity (low vs. high) and execution effectiveness (low vs. high). At low submission rates (left side of Figure 3), two clusters exhibit sharply different outcomes. **Cluster 3** concentrates in the upper-left area, achieving near-perfect success (0.995), with 43.14% of addresses reaching 100% success; while submitting transactions rarely (9.82×10^{-4} tx/sec), reflecting highly selective execution. In contrast, **Cluster 0** also operates at very low submission rates (0.032 tx/sec) but remains concentrated in the lower-left region, with 84.85% of addresses below a 0.2 success rate, indicating sparse yet ineffective attempts. At high submission rates (right side of Figure 3), the other two clusters again separate

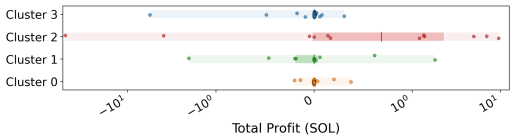

Figure 4: Total profits Distribution across clusters. Bands show the full (light) and interquartile (dark) ranges.

by execution effectiveness. **Cluster 1** occupies the lower-right region, characterized by frequent submissions (1.360 tx/sec) coupled with 72.73% of addresses below a 0.2 success rate, consistent with volume-driven speculative behavior. **Cluster 2**, however, lies in the upper-mid region, combining sustained activity (0.457 tx/sec) with high success (0.918 on average), representing a more balanced execution pattern. Table 6 in the Appendix provides quantitative summaries of each cluster of the bot addresses.

We further examined the realized per-address profits to characterize the economic outcomes associated with each execution regime, as shown in Figure 4. We observe that execution intensity and success rate do not reliably translate into stable profitability. Instead, they correspond to distinct risk exposures: near-zero outcomes (**Cluster 0**), frequent small losses (**Cluster 1**), heavy-tailed downside exposure (**Cluster 2**), and near-zero gains with occasional downside tail risk (**Cluster 3**).

> **Finding 5:** Execution characteristics of on-chain bot addresses in the success–frequency space fall into four clusters, characterized by success rate and submission rate. Realized per-address profit distributions show that neither execution intensity nor success rate translates into stable profitability, and instead they correspond to distinct risk exposures.

6.2.2 Trading Venues. To interpret the intent-level semantics of the execution regimes identified in Figure 3, we performed transaction-level inspection of representative addresses sampled from each cluster. We operationalized two execution patterns, *round-trip motif* and *venue-specific directional flow*, and manually annotated sampled addresses according to the protocol described below. ***Operational Definitions.*** Following prior studies [55, 70] on blockchain arbitrage, we define a *round-trip motif* as a transaction pattern satisfying two conditions: (i) the input and output assets are the same base asset, such as WSOL; (ii) the execution path traverses at least two distinct DEX venues. In contrast, we define a *venue-specific directional flow* as a transaction pattern confined primarily to a single venue or a DEX aggregator without a closed-loop return to the base asset, which is consistent with trading-operation behavior. ***Sampling and Annotation.*** For each of Clusters 0-3, we stratified addresses into high-, mid-, and low-probability tiers based on HDBSCAN membership probability using a tertile split within each cluster. We randomly sampled up to five addresses per tier; for tiers with fewer than five addresses, all available addresses were included. Two annotators independently labeled each address as round-trip, venue-specific directional, or ambiguous/other, achieving 96.23% inter-annotator agreement. See Appendix A.3 for details.

Across Clusters 0-2, 37 of the 38 sampled addresses exhibited the round-trip motif. One representative transaction [15] executes an aggregated swap across two DEX venues: it first swaps 3.549 WSOL for 23,899.845 PAYAI on Raydium, and then swaps the same amount of PAYAI back for 3.561 WSOL on Meteora DLMM. The resulting WSOL–PAYAI–WSOL path closes on the same base asset while crossing two venues, providing transaction-level evidence of the round-trip arbitrage motif. Meanwhile, all 15 sampled addresses in Cluster 3 exhibited venue-specific directional flows dominated by Pump.fun-related activity. Accordingly, we align Clusters 0–2 with MEV and Cluster 3 with Trading Operations when integrating the clusters into the taxonomy of RQ1.

We further cross-validated the semantic interpretation using known bot-service labels. Excluding noise, 90.32% of clustered SolanaMevBot addresses fall in Clusters 0-2 (33, 11, and 12 addresses; 38 noise), whereas all clustered Trojan addresses fall in Cluster 3 (96 addresses; 4 noise). This service-level separation corroborates the transaction-level evidence for interpreting Clusters 0–2 as MEV-oriented and Cluster 3 as trading-operation-oriented.

We examined the top 10 trading venues by transaction count across clusters of bot addresses (Table 8in the Appendix). The four clusters are separated along (i) venue concentration and (ii) aggregator mediation. The venue-level differences across clusters also tend to align with the distinct execution regimes observed in Section 6.2.1. Specifically, **Cluster 2** (Aggregator-Centric MEV), dominated by Jupiter Aggregator v6 (67.9%), corresponds to the high-success, sustained-activity regime, suggesting that aggregator-mediated routing could be associated with stable execution effectiveness. In contrast, **Cluster 3** (Pump.fun-Centric Trading), with over 80% of transactions confined to the Pump.fun ecosystem, coincides with the low-frequency but near-perfect execution regime, indicating highly selective activity in a single venue. **Cluster 1** (Venue-Specialized MEV), whose activity is concentrated on Raydium and Meteora pools (over 95% combined), aligns with the high-submission yet low-success regime, reflecting intensive but competitive execution in specialized venues. **Cluster 0** (Multi-Venue MEV), which distributes transactions across several venues without strong aggregator reliance, corresponds to the low-activity and low-success regime, indicating dispersed yet ineffective execution.

Beyond venue concentration and aggregator mediation, additional venue-level signals further differentiate the clusters of MEV bot addresses. In particular, the appearance of Kamino Lending within **Cluster 0** indicates that lending primitives are occasionally invoked alongside swap instructions, potentially enabling capital-efficient strategies such as flash-loan-assisted arbitrage. Moreover, **Cluster 2** interacts with several proprietary AMMs (e.g., HumidiFi [8, 9, 12]), suggesting routing paths that extend beyond standard public liquidity pools and may involve access to closed or specialized liquidity sources. We also observe that, within **Cluster 2**, transactions invoking the HumidiFi program are associated with more favorable profit outcomes. Specifically, 62.30% (152,502/244,733) of HumidiFi-invoking transactions are profitable, compared to only 21.01% (45,949/218,678) of non-HumidiFi-invoking transactions. In addition, HumidiFi-invoking transactions yield an average profit of 143, 661 lamports, whereas non-HumidiFi-invoking transactions incur an average loss of 8, 693 lamports.

> **Finding 6:** Trading venues differs across MEV bots. Trading Operations bots confined over 80% of transactions to the Pump.fun ecosystem.

6.2.3 Traded Tokens and Transfer Outflows. We examined the top-10 traded tokens in transactions initiated by each cluster and observed distinct token trading patterns across clusters (Table 9 in the Appendix). The main observations are as follows:

***MEV* bots (Clusters 0–2).** WSOL serves as a dominant pivot asset across the MEV-oriented clusters. WSOL appears in 99.94% of transactions initiated by addresses in Cluster 1 (Venue-Specialized MEV) and 98.25% in Cluster 2 (Aggregator-Centric MEV), and remains prevalent in Cluster 0 (Multi-Venue MEV) at 70.90%. The clusters also differ in token concentration and breadth. Cluster 1 exhibits the tightest co-anchor structure, where WSOL (99.94%) frequently co-occurs with a small set of recurring tokens such as USD1 (46.84%) and USELESS COIN (44.00%). Cluster 0 combines WSOL (70.90%) with substantial stablecoin exposure (USDC 50.97%) alongside additional long-tail tokens. Cluster 2 maintains a WSOL backbone (98.25%) while spanning a broader mid-frequency token mix (4.13–15.04% across tokens), indicating wider asset coverage.

***Trading Operations* Bots (Cluster 3)** show no dominant pivot tokens in their transactions, with WSOL appearing in only 10.93% of the transactions. Its top tokens are instead dominated by Pump.fun assets with low individual frequencies (all below 8%), consistent with a long-tail, platform-driven token distribution in transactions. Moreover, transfer-outflow recipients suggest different execution-side dependencies across clusters: *MEV* bots are infrastructure-centered, whereas *Trading Operations* bots are more platform- and service-distributed. Specifically, *MEV* bots (**Clusters 0–2**) exhibit extreme concentration toward inclusion infrastructure (i.e., Jito tip payment accounts), whereas *Trading Operations* bots (**Cluster 3**) distribute outflows across trading platforms (e.g., Trojan accounts) and inclusion services (e.g., Jito tip payment accounts). Detailed recipient distributions are provided in Table 10 in the Appendix.

> **Finding 7:** MEV bots mainly trade WSOL as pivot token, and transfer to infrastructure-related recipients. Trading Operations bots have no dominant pivot, and distribute transfers across platforms and services.

6.2.4 Cluster and Behavioral Consistency across Time Windows. The baseline clusters remain broadly applicable to validation-window addresses. All 100 Axiom addresses were mapped to Cluster 3. For 100 SolanaMevBot addresses from the validation window, 41, 4, and 2 were assigned to Clusters 0, 1, and 2, respectively; 20 were assigned to Cluster 3; and the remaining 33 were classified as noise. This corresponds to 0% and 33% noise ratios for Axiom and SolanaMevBot addresses in the validation window, respectively. The remaining SolanaMevBot noise addresses suggest that some later-window behaviors are less well captured by the clusters observed in the baseline window. Details see Appendix A.2.

Different behavioral dimensions exhibit different levels of temporal stability across the baseline and validation windows. Overall, trading-venue usage remains the most stable behavioral dimension, followed by transfer-outflow recipients, whereas traded tokens exhibit substantially greater temporal variation. ***Trading Venues.*** For the MEV-oriented clusters, C0 preserves its original multi-venue pattern, with 7 of its top-10 DEX venues overlapping with the original C0, including Jupiter, Pump.fun AMM, Meteora DLMM, and Whirlpools. C1 exhibits the strongest continuity, with 8 overlapping top-10 venues centered on Meteora, Raydium, Pump.fun AMM, and Whirlpools. C3 also retains a Pump.fun-oriented venue profile, with Pump.fun and Pump.fun AMM remaining among the dominant venues. Consistently, all validation-window Axiom addresses, which are assigned to C3, exhibit the same Pump.fun-oriented trading behavior. In contrast, C2 shows weaker stability, with only 2 overlapping top-10 venues. ***Transfer-outflow Recipients.*** C1 and C2 remain strongly Jitotip-centered, accounting for 80.64% and 100% of total transfers, respectively. C3 continues to direct transfers primarily to Jitotip (18.76%) and Pump.fun protocol-fee accounts (14.39%). Although C0 shows a different set of dominant recipient accounts, it still shares common service-level recipients such as Pump.fun protocol-fee accounts, indicating that the underlying execution infrastructure remains largely unchanged. ***Traded Tokens*** exhibit substantially greater temporal variation. Across clusters, only 3 of the top-10 traded tokens overlap with those in the baseline window. The stable overlap mainly consists of common base or pivot assets, including WSOL, USDC, and USD1, whereas long-tail traded tokens change considerably across the validation window.

7 Implications

Architectural Regularity and Engineering Implications for Solana Bots. Solana bots exhibit a stable high-level execution backbone despite substantial variation in low-level implementation (Finding 2). At the architectural level, repositories across bot categories repeatedly align with a common five-stage pipeline, especially around shared components such as *Data Acquisition*, *Data Analytics*, and *Trade Execution* (Finding 3). The observed architectural regularity of Solana bots at the code level is broadly consistent with prior studies of bots on Ethereum and BNB [23, 59], which identify recurring operational motifs such as sniping, front-running, and arbitrage from account- or transaction-level evidence. Meanwhile, the recurrence of a small set of building blocks suggests that reimplementation is pervasive in the Solana bot OSS ecosystem.

Takeaway: Instead of repeatedly developing end-to-end bots from scratch, practitioners should prioritize the shared maintenance of recurrent bot components, while preserving flexibility for long-tail, strategy-specific logic. Moreover, the five-stage pipeline identified in RQ2 can be viewed as an empirically derived architectural reference model for designing and auditing Solana bots.

Dependency Lag as an Ecosystem-Level Maintainability Concern of Solana Bots. More than 30% of the dependencies in the third-party libraries in Solana lag behind the latest available release by over one year (Finding 4). The observation suggests that dependency technical lag is not merely a repository-level issue, but an ecosystem-level maintainability concern for Solana bots. The maintainability concern, however, is not uniform across dependency types: for protocol-facing libraries such as core Solana SDKs, lag poses a genuine compatibility risk, since Solana's transaction formats, RPC interfaces, and fee-setting mechanisms evolve continuously, and staleness here risks silent incompatibility rather than merely missing new features. For general-purpose libraries not directly coupled to protocol evolution (e.g., dotenv, axios), lag instead reflects a stability-versus-currency tradeoff familiar from software maintenance more broadly, where a pinned, well-tested dependency environment can be preferable to frequent, disruptive upgrades. The maintainability concern is especially salient for Solana SDK dependencies, where the technical lag of widely used Solana SDKs ranges from 125 to 572 days. Blockchain bots tend to operate in latency-sensitive and adversarial transaction environments, where their execution success depends heavily on SDK APIs, RPC interfaces, transaction formats, and fee-setting mechanisms [39, 56].

Takeaway: Practitioners, therefore, should prioritize the management of SDK dependencies in Solana bots as part of ensuring transaction correctness and execution reliability, rather than as a background maintenance task, while general-purpose dependencies can be managed with more discretion, favoring stability where frequent upgrades offer limited benefit. Correspondingly, tool builders should support not only outdated-dependency detection, but also compatibility assessment for evolving transaction formats, RPC interfaces, and fee-related mechanisms, so that update decisions can be prioritized by dependency type rather than applied uniformly. Future work could investigate the underlying drivers of dependency lag in Solana bot repositories, including how the evolution of SDK libraries and transaction mechanisms affects dependency update decisions, maintenance effort, and compatibility management.

The Interaction of Infrastructure, Venue, and Execution Regime is Associated with Distinct Execution Outcomes of Solana Bots. Solana bots can be characterized through three interacting layers: (1) execution infrastructure, such as Jito, which shapes transaction submission and ordering; (2) DEX venue integration, such as proprietary AMM (e.g., HumidiFi), which is associated with how bots access liquidity and routing opportunities; and (3) execution regimes, which reflect how bots realize trading strategies within these constraints. Execution outcomes, therefore, are affected by the interaction of execution infrastructure, venue integration, and execution regime, rather than by trading logic alone (Findings 6 and 7), which is consistent with prior studies [39, 56].

Takeaway: Practitioners should design Solana bots through joint consideration of execution infrastructure, venue integration, and execution regimes, to secure execution edges in the highly competitive Solana market. Tool builders could support the design exploration across these layers. Future work could investigate the mechanisms underlying the observed coupling among execution infrastructure, venue integration, and execution regimes, including whether this coupling reflects deliberate design choices by practitioners, latency-driven optimization pressures [32, 39], or access constraints imposed by private liquidity ecosystems [8]. Cross-chain comparisons could further examine how ordering mechanisms, mempool visibility, and fee structures shape bot specialization.

8 Threats to Validity

Construct Validity. Our GitHub repository and on-chain address datasets of Solana bots were collected independently and cannot be linked at the address level. The two datasets therefore provide complementary views of the Solana bot ecosystem, covering implementation-level functionality and execution-level behavior. Accordingly, in Section 6.2.2, we establish correspondence between the two datasets at the behavioral level by linking on-chain clusters to the MEV and Trading Operations categories based on transaction traces from representative addresses in each cluster.

External Validity. Three repository domains (*Market Manipulation*, *On-chain Analytics*, and *Tooling and Infrastructure*) are absent from our on-chain cluster, because the two analyzed bot services (Trojan and SolanaMevBot) are trading/arbitrage-oriented and do not exercise these domains. The observation reflects the service-level scope of our on-chain sample rather than an artifact of the clustering procedure. Consequently, our on-chain findings (Section 6.2) generalize primarily to trading- and MEV-oriented bots.

9 Related Work

Studies on Solana. Early work investigated the performance and scalability of the Solana blockchain through its architecture [48], capability for large-scale IoT workloads [37], and transaction latency and fees [61]. Other prior work developed automated techniques to detect vulnerabilities in Solana smart contracts using static analysis [31] and coverage-guided fuzzing [63]. Recent studies have examined diverse on-chain phenomena on Solana through systematic measurement, including failed transactions [75], MEV sandwiching attacks [40], meme coin markets [49], and rug pulls [19], providing insights into transaction-level behaviors and economic dynamics on Solana. Despite extensive prior work on Solana, existing studies overlook Solana bots as software artifacts despite generating substantial on-chain activity. We addresses this gap through a systematic empirical study of Solana bots that jointly examines their code-level characteristics and observable on-chain behaviors.

Studies on Blockchain Bots. One line of prior work examines the behaviors of bots in AMM ecosystems on the Ethereum and BSC blockchains, including token sniping [21, 24], front-running [33, 47, 62], and sandwich attacks [62, 67, 76]. Another line of prior work explores the automatic detection of bot-controlled accounts across blockchain platforms, such as EOSIO [44] and Ethereum [45, 58]. A recent study characterizes the bot accounts on the Solana blockchain that initiate failed transactions [75]. Overall, existing studies primarily focus on account-level analyses of bot behaviors, with most efforts centered on the Ethereum and BSC blockchains. In contrast, our work explores bots on the Solana blockchain from a software-centric perspective. Closely related to our work, Niedermayer et al. [58] derive an Ethereum financial-bot taxonomy from literature, GitHub repositories, and on-chain data. The taxonomy organizes financial bot activity into 7 categories and 24 subcategories, which elevates *MEV* as a mechanism-level category rooted in transaction-ordering competition, distinguishes venue-based branches (i.e., *CEX* and *DEX*), and further differentiates application- or asset-oriented domains such as *NFT* and *Play-to-earn*.

10 Conclusion and Future Work

In this work, we present the first large-scale, implementation-grounded characterization of Solana bots by jointly analyzing 586 GitHub repositories and 200 bot-associated addresses with over 44 million on-chain transactions. We derived a 15-category taxonomy, identified a shared five-stage operational pipeline of bot implementations, and found distinct bot execution fingerprints across venues and assets. Our findings characterize Solana bots from both code-level implementation and observable on-chain execution perspectives, and highlight relevant software engineering concerns that arise in architecture, dependency management, and cross-layer execution design for bots operating in a competitive execution environment and a fast-evolving software stack. Future work could explore the differences in ordering mechanisms, mempool visibility, and fee structures across blockchain systems, and how they shape distinct forms of bot specialization across blockchain systems.

11 Data Availability

Our replication package is available online: https://doi.org/10.5281/zenodo.21359451.

References

[1] [n. d.]. Solana Programs. https://solana.com/docs/core/programs. Accessed: 2026-01-28.

[2] 2024. *Bots are Making Huge Profit on The Solana Ecosystem! Dominating Other Blockchains.* Accessed: 2025-05-13. https://support.bitrue.com/hc/en-001/articles/30134959225113-Bots-are-Making-Huge-Profit-on-The-Solana-Ecosystem-Dominating-Other-Blockchains

[3] 2024. *Bots suspected of pushing Solana over Ethereum — Research.* Accessed: 2024-10-28. https://cointelegraph.com/news/bots-pushing-solana-over-ethereum-research

[4] 2024. *Consensus on Solana.* Accessed: 2024-10-28. https://www.helius.dev/blog/consensus-on-solana

[5] 2024. Solana Account Model. https://solana.com/docs/core/accounts. Accessed: 2026-01-28.

[6] 2024. *Solana MEV Bots: A Detailed Explanation.* Accessed: 2024-10-28. https://www.coinfeeds.ai/crypto-blog/solana-mev-bots

[7] 2024. Solana Transactions. Accessed: 2026-01-28. https://github.com/solana-foundation/developer-content/blob/main/docs/core/transactions.md

[8] 2025. *Solana's Proprietary AMM Revolution.* Accessed: 2026-02-11. https://www.helius.dev/blog/solanas-proprietary-amm-revolution

[9] 2025. *Solana's Proprietary AMMs.* Accessed: 2026-02-11. https://blockworks.co/news/solanas-proprietary-amms

[10] 2026. *The Appendix for "Demystifying Solana Bots: From GitHub Blueprints to On-Chain Fingerprints".* Accessed: 2026-03-27. https://doi.org/10.5281/zenodo.19248227

[11] 2026. *Axiom Pro.* Accessed: 2026-07-13. https://dune.com/adam_tehc/axiom

[12] 2026. *How Dark Pools Are Changing DeFi and What It Means for the Industry.* Accessed: 2026-02-11. https://incrypted.com/en/how-dark-pools-are-changing-defi-and-what-it-means-industry/

[13] 2026. *Solana MEV: An Introduction.* Accessed: 2026-07-13. https://www.helius.dev/blog/solana-mev-an-introduction

[14] 2026. *Trading Bots on Solana.* Accessed: 2026-03-27. https://dune.com/adam_tehc/trading-bots-on-solana

[15] 2026. *Transaction 4MVwhwBg2VTgTEe58st6sPfP2tr2sX3fbGq55vzsrqpXap4L8hnPbb6vvquwNkg7BxdvSgBqEytqhi2STsUyz9zo.* Accessed: 2026-07-30. http://bit.ly/4hF537m

[16] 2026. *Understanding Proprietary AMMs.* Accessed: 2026-07-13. https://solana.com/news/understanding-proprietary-amms

[17] 2026. *What Is Wrapped SOL?* Accessed: 2026-07-13. https://solana.com/docs/tokens/basics/sync-native

[18] Roozbeh Aghili, Heng Li, and Foutse Khomh. 2023. Studying the characteristics of AIOps projects on GitHub. *Empirical Software Engineering* 28, 6 (2023), 143. doi:10.1007/s10664-023-10382-z

[19] Abdulrahman Alhaidari, Bhavani Kalal, Balaji Palanisamy, and Shamik Sural. 2025. SolRPDS: A Dataset for Analyzing Rug Pulls in Solana Decentralized Finance. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy* (Pittsburgh, PA, USA) *(CODASPY '25)*. Association for Computing Machinery, New York, NY, USA, 293–298. doi:10.1145/3714393.3726487

[20] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 160–172. doi:10.1007/978-3-642-37456-2_14

[21] Federico Cernera, Massimo La Morgia, Alessandro Mei, Alberto Mongardini, and Francesco Sassi. [n. d.]. The blockchain warfare: Investigating the ecosystem of sniper bots on Ethereum and BNB smart chain. *ACM Transactions on Internet Technology* ([n. d.]). doi:10.1145/3736763

[22] Federico Cernera, Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Francesco Sassi. 2023. Ready, aim, snipe! analysis of Sniper Bots and their impact on the DeFi ecosystem. In *Companion Proceedings of the ACM Web Conference 2023*. 1093–1102. doi:10.1145/3543873.3587612

[23] Federico Cernera, Massimo La Morgia, Alessandro Mei, and Francesco Sassi. 2023. Token spammers, rug pulls, and sniper bots: An analysis of the ecosystem of tokens in ethereum and in the binance smart chain (*BNB*). In *32nd USENIX security symposium (USENIX security 23)*. 3349–3366.

[24] Federico Cernera, Massimo La Morgia, Alessandro Mei, and Francesco Sassi. 2023. Token Spammers, Rug Pulls, and Sniper Bots: An Analysis of the Ecosystem of Tokens in Ethereum and in the Binance Smart Chain (BNB). In *32nd USENIX Security Symposium (USENIX Security 23)*. 3349–3366.

[25] 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

[26] Yujing Chen, Xuanming Liu, Zhiyuan Wan, Zuobin Wang, David Lo, Difan Xie, and Xiaohu Yang. 2025. Why Is My Transaction Risky? Understanding Smart Contract Semantics and Interactions in the NFT Ecosystem. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering*. doi:10.1109/ASE63991.2025.00172

[27] Usman W Chohan. 2021. Non-fungible tokens: Blockchains, scarcity, and value. In *Non-Fungible Tokens*. Routledge, 1–11. doi:10.4324/9781003435518-1

[28] Circular. 2026. *Circular Bots.* https://circular.fi/bots

[29] William G. Cochran. 1977. *Sampling Techniques* (3rd ed.). Wiley, New York.

[30] Filipe Roseiro Cogo, Gustavo A Oliva, and Ahmed E Hassan. 2019. An empirical study of dependency downgrades in the npm ecosystem. *IEEE Transactions on Software Engineering* 47, 11 (2019), 2457–2470. doi:10.1109/TSE.2019.2952130

[31] 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

[32] 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

[33] 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

[34] DefiLlama. [n. d.]. All Chains DeFi TVL. Accessed: 2025-05-13. https://defillama.com/chains

[35] Nilesh Dhulshette, Sapan Shah, and Vinay Kulkarni. 2025. Hierarchical repository-level code summarization for business applications using local llms. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*. IEEE, 145–152. doi:10.1109/LLM4Code66737.2025.00023

[36] Sahibsingh A Dudani. 1976. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics* 4 (1976), 325–327. doi:10.1109/TSMC.1976.5408784

[37] 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

[38] Dune. 2026. *Dune Analytics.* Accessed: 2026-03-24. https://dune.com/

[39] Nicole Gerzon, Ben Weintraub, Junbeom In, Alan Mislove, and Cristina Nita-Rotaru. 2025. Quantifying the Threat of Sandwiching MEV on Jito: A Measurement of Solana's Leading Validator Client. In *Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25)*. ACM. Also available as an author-hosted PDF.. doi:10.1145/3730567.3764493

[40] Nicole Gerzon, Ben Weintraub, Junbeom In, Alan Mislove, and Cristina Nita-Rotaru. 2025. Quantifying the Threat of Sandwiching MEV on Jito: A Measurement of Solana's Leading Validator Client. In *Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25)*. ACM, 937–943. doi:10.1145/3730567.3764493

[41] Runzhi He, Hao He, Yuxia Zhang, and Minghui Zhou. 2023. Automating dependency updates in practice: An exploratory study on github dependabot. *IEEE Transactions on Software Engineering* 49, 8 (2023), 4004–4022. doi:10.1109/TSE.2023.3278129

[42] Helius. 2023. Priority Fees: Understanding Solana's Transaction Fee Mechanics. https://www.helius.dev/blog/priority-fees-understanding-solanas-transaction-fee-mechanics. Accessed: 2026-01-28.

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

[44] 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

[45] 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

[46] Jito Labs. 2025. What is Jito? — Jito Labs Documentation. https://docs.jito.wtf/. Last updated: 2025-12-30; Accessed: 2026-01-28.

[47] 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

[48] 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

[49] Yueyao Li, Nanjun Yao, Yuhui Huo, and Wei Cai. 2025. Trust Dynamics and Bot-Driven Responses: An Approach to Rug Pulls in Solana Meme Coin Markets. In *Proceedings of the 17th ACM Web Science Conference 2025*. 106–116. doi:10.1145/3717867.3717922

[50] Wentao Liang, Xiang Ling, Jingzheng Wu, Tianyue Luo, and Yanjun Wu. 2023. A Needle is an Outlier in a Haystack: Hunting Malicious PyPI Packages with Code Clustering. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 307–318. doi:10.1109/ASE56229.2023.00085

[51] Yanyi Liu, Qingwen Yang, Tiezheng Guo, Feiyu Qu, Jun Liu, and Yingyou Wen. 2026. From Detection to Diagnosis: Advancing Hallucination Analysis with Automated Data Synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 32222–32230. doi:10.1609/aaai.v40i38.40495

[52] Zuchao Ma, Muhui Jiang, Feng Luo, Xiapu Luo, and Yajin Zhou. 2025. Surviving in Dark Forest: Towards Evading the Attacks from Front-Running Bots in Application Layer. In *34rd USENIX Security Symposium (USENIX Security 25)*.

[53] Onaiza Maqbool and Haroon Babri. 2007. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering* 33, 11 (2007), 759–780. doi:10.1109/TSE.2007.70732

[54] Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018). doi:10.48550/arXiv.1802.03426

[55] Robert McLaughlin, Christopher Kruegel, and Giovanni Vigna. 2023. A large scale study of the ethereum arbitrage ecosystem. In *32nd USENIX Security Symposium (USENIX Security 23)*. 3295–3312.

[56] Robert McLaughlin, Christopher Kruegel, and Giovanni Vigna. 2023. A large scale study of the ethereum arbitrage ecosystem. In *32nd USENIX Security Symposium (USENIX Security 23)*. 3295–3312.

[57] Bingshen Mu, Hexin Liu, Hongfei Xue, Kun Wei, and Lei Xie. 2026. Hearing More with Less: Multi-Modal Retrieval-and-Selection Augmented Conversational LLM-Based ASR. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 32519–32527. doi:10.1609/aaai.v40i38.40528

[58] 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

[59] Thomas Niedermayer, Pietro Saggese, and Bernhard Haslhofer. 2024. Detecting financial bots on the ethereum blockchain. In *Companion Proceedings of the ACM Web Conference 2024*. 1742–1751. doi:10.1145/3589335.3651959

[60] Amirkia Rafiei Oskooei, Selcan Yukcu, Mehmet Cevheri Bozoglan, and Mehmet S Aktas. 2025. Repository-Level Code Understanding by LLMs via Hierarchical Summarization: Improving Code Search and Bug Localization. In *International Conference on Computational Science and Its Applications*. Springer, 88–105. doi:10.1007/978-3-031-97576-9_6

[61] 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

[62] 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

[63] 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

[64] Solana. [n. d.]. Solana Official Website. Accessed: 2025-05-13. https://solana.com/

[65] Solana Foundation. 2019. Gulf Stream: Solana's Mempool-less Transaction Forwarding Protocol. https://solana.com/news/gulf-stream--solana-s-mempool-less-transaction-forwarding-protocol. Accessed: 2026-01-28.

[66] SolanaMevBot. 2026. *SolanaMevBot Documentation: Overview.* https://docs.solanamevbot.com/home/

[67] Christof Ferreira Torres, Ramiro Camino, et al. 2021. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*. 1343–1359.

[68] Tree-sitter. 2026. *Tree-sitter: An incremental parsing system.* Accessed: 2026-03-24. https://github.com/tree-sitter/tree-sitter

[69] Trojan On Solana. 2026. *Trojan On Solana: The Leading Telegram Trading Bot.* https://trojanonsolana.com/

[70] Ye Wang, Yan Chen, Haotian Wu, Liyi Zhou, Shuiguang Deng, and Roger Wattenhofer. 2022. Cyclic arbitrage in decentralized exchanges. In *Companion Proceedings of the Web Conference 2022*. 12–19. doi:10.1145/3487553.3524201

[71] whale_hunter. 2026. *DEX Trading Bot Wars.* Dune Analytics dashboard. https://dune.com/whale_hunter/dex-trading-bot-wars

[72] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in software engineering*. Vol. 236. Springer. doi:10.1007/978-3-642-29044-2

[73] Anatoly Yakovenko. 2018. Solana: A new architecture for a high performance blockchain v0. 8.13. Accessed: 2026-07-31. https://solana.com/solana-whitepaper.pdf

[74] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv preprint arXiv:2506.05176* (2025). doi:10.48550/arXiv.2506.05176

[75] 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. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 1489–1512. doi:10.1145/3728943

[76] 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 2026-03-26; accepted 2026-06-18