Beyond the Stars: Multimodal Detection of Scams on GitHub

Tillson Galloway, Kevin Valakuzhy, Manos Antonakakis, Fabian Monrose
School of Electrical and Computer Engineering
Georgia Institute of Technology

Abstract

The open-source software development ecosystem has become a prime target for phishing and supply-chain attacks, precisely because its openness encourages trust. Defenders fighting these attacks operate in an environment where adversaries generate malicious artifacts at industrial scale, evade signature-based detection through trivial mutations, and exploit delays inherent in community-driven labeling. This combination produces sparse, biased ground truth and extreme class imbalance—conditions under which conventional evaluation pipelines break down. Unsurprisingly, many detection systems that excel offline degrade sharply in real-world deployments, limiting their impact against sustained campaigns.

We present *OctoWatch*, a multimodal machine learning system for expanding detection of large-scale malicious source code repositories under real-world operational constraints while enabling rapid analyst pivoting through learned representations. The system fuses natural-language representations of repository metadata and file contents with structural embeddings of user–repository interactions and temporal activity statistics. Evaluated on real-world datasets—including large-scale secret-exposure attacks, prior threat-intelligence reporting, and a previously unreported phishing campaign—the approach consistently outperforms prior baselines in both precision and Matthews Correlation Coefficient (MCC). To assess operational viability, we conduct an online deployment that processes over ten million newly active repositories and identifies 14,667 total malicious repositories absent from major OSINT feeds. The system further enables discovery of coordinated campaigns, including one activity cluster associated with $5.9 million USD in cryptocurrency transactions on the Ethereum blockchain. Responsible disclosure of these findings led to the takedown of thousands of malicious repositories, demonstrating that multimodal fusion enables scalable detection of software-ecosystem abuse that systematically evades existing community-driven defenses.

1 Introduction

Centralized open-source platforms (OSPs) that distribute software artifacts and source code form a critical pillar of the modern software development ecosystem. Recent estimates indicate nearly 6.6 trillion package downloads per year, with content volume growing rapidly—driven in part by the proliferation of AI-generated code [38]. At this scale, manual review and community reporting are no longer viable, forcing platforms to rely on automated mechanisms to identify abuse.

This combination of massive scale, limited human oversight, and automated trust signals has made open-source platforms such as GitHub and npm high-value targets for phishing and supply-chain attacks. Adversaries exploit repositories and packages to impersonate trusted projects, distribute malicious code, and conduct large-scale scams, attracted by free shared hosting and the implicit legitimacy conferred by platform scale and visibility [40, 41].

Empirical studies show that users systematically overestimate the trustworthiness of open-source platforms, relying on surface-level signals such as star counts and apparent community adoption, and consequently applying weaker scrutiny to hosted artifacts. Longitudinal studies confirm that attackers deliberately exploit these ecosystems by mimicking popular projects and manipulating visible metadata, capitalizing on users' routine execution of code and automated dependency resolution [12, 18, 29, 32, 36]. This trust misalignment enables malicious repositories and packages to spread widely before detection [8]. Further, targeting software development pipelines has been shown to be substantially more effective than conventional web-based attacks when targeting software developers [37]. Reducing user compromise requires shortening the timeframe in which malicious repositories and packages are available—at scale, this requires automated detection.

Despite years of effort by both the security community and platform operators to address malware on platforms like GitHub [16, 18, 32], a substantial gap remains between research prototypes and deployable abuse detection. Existing approaches often evaluate on small labeled datasets that are

not representative of GitHub at large, or exclude these threats due to the high false-positive rates of network-based detection mechanisms against malicious samples that abuse shared hosting providers [10]. Closing this gap requires that we design systems that work well under real-world deployment constraints, including dealing with incomplete and ephemeral ground truth, limited observability due to platform-imposed rate limits, and aggressive deletion of content, all while preserving high model quality under extreme class imbalance.

To this end, we develop an expressive multimodal machine learning framework that integrates three complementary signal classes: natural language, graph structure, and statistical behavior. Natural-language features, implemented using Transformer models [45], capture lexical patterns and semantic intent in repository names and file contents. Graph-based features, learned using a graph neural network (GNN), model social relationships such as coordinated starring by compromised or automated accounts. Statistical features capture temporal and behavioral dynamics, including repository age, activity bursts, and reputation growth over time. Together, these modalities provide overlapping and mutually reinforcing signals that remain effective even when individual feature classes are sparse, missing, or manipulated.

Using this framework, we explore the following questions:

- **RQ1 (Real-World Effectiveness):** *How do multimodal methods perform in malicious-campaign expansion and analyst-driven pivoting tasks under real-world conditions?*
- **RQ2 (Threat Coverage):** *What is the scope of the threat activity that can be disrupted using these methods in real-world scenarios?*
- **RQ3 (Engagement & Impact):** *How frequently do potential victims engage with scams? What is the financial impact of detected abuse on GitHub?*

Our evaluation demonstrates that the system achieves strong performance across multiple real-world scam and supply-chain attack datasets, substantially outperforming prior content-only and graph-only baselines under severe class imbalance. In an online study considering 13.5 million repositories, OctoWatch identified 14,667 previously unreported malicious repositories at analyst-operable false-positive rates, enabling earlier disruption of repositories involved in large-scale phishing and malware campaigns. Beyond detection performance, our analysis shows that these campaigns impose substantial real-world harm, including widespread user engagement with scam repositories, systematic reputation manipulation, and significant losses for victims.

In summary, this work makes the following contributions:

- We design and implement a multimodal framework to expand coverage of malicious GitHub campaigns by identifying related repositories at platform scale. Our approach integrates statistical activity signals, graph-based interaction structure, and NLP-derived representations of repository metadata and content.
- We conduct the first large-scale, operationally grounded evaluation of multimodal abuse detection on GitHub, combining offline benchmarks with a 45-day online deployment spanning millions of newly active repositories.
- Our system uncovers 14,667 previously active scam repositories, including a cryptocurrency scam campaign associated with nearly $6 million USD in victim losses. We disclose all of our findings to GitHub and the broader security community.
- We release the first large-scale GitHub phishing scam dataset that preserves historic file contents, enabling reproducible research and retrospective evaluation of scam and malware distribution in open-source platforms.

2 Background

GitHub and the Open-Source Developer Ecosystem. Open-source software is a foundational component of modern software development workflows. The emergence of software development repositories and package management systems has fundamentally reshaped how developers build software, enabling large-scale reuse of code, rapid experimentation, and the discovery and dissemination of implementation patterns and ideas [11]. These platforms allow developers not only to forage online for reusable components [47], but also to advertise and distribute high-quality code to a global audience.

GitHub is the most prominent example of such a platform. It serves as a centralized hub for code collaboration and storage, where developers host public and private code repositories. With millions of active projects, GitHub plays a central role in enabling collaboration, version control, and code discovery [11]—supported by advanced search and ecosystem integrations. This centrality also places GitHub in a uniquely sensitive position within the software supply chain: content moderation decisions must balance abuse mitigation against the risk of disrupting legitimate development workflows and downstream users [44]. In particular, because GitHub-hosted code is frequently consumed directly or indirectly as a dependency in production systems, repository takedowns can propagate availability failures across dependent software deployments [34].

Multimodal Learning. Multimodal learning refers to machine learning systems that integrate heterogeneous sources of information. Originally popularized in world-modeling applications like vision–language and speech–text systems,

multimodal architectures have recently gained traction in network security. Prior work has applied these techniques to netflow analysis [49], malware classification [20], and intrusion detection [22], demonstrating their ability to capture complementary signals that are difficult to model with a single feature modality.

In this work, we consider three distinct modalities (i.e., natural language, graph-structured data, and statistical features), which must be combined through a *fusion* step. Contemporary multimodal systems typically rely on one of two fusion strategies: (1) *affine* fusion layers, which learn linear interactions among modality-specific feature embeddings, and (2) *bilinear* fusion layers, which introduce cross-terms to explicitly model pairwise interactions between features from different modalities. Although bilinear fusion is more expressive, it incurs a quadratic increase in computational cost versus affine alternatives, making its use intractable at scale.

3 Related Work

A substantial body of work has examined the social and structural dynamics of open-source ecosystems. This includes network-level analyses of developer interactions and trust relationships [27], as well as studies of underground or otherwise abusive communities operating within and around open-source platforms [19, 41, 52]. Collectively, these efforts motivate both the pervasiveness of abuse on platforms such as GitHub and the suitability of graph-based techniques for analyzing it. Our work builds on these insights but differs in scope: rather than focusing solely on measurement or characterization, we design and deploy an operational system for large-scale abuse detection.

Table 1 summarizes prior work on malicious entity detection on GitHub, spanning commits [16], repositories [30, 31], forks [4], and users [18]. As shown, a key limitation of existing evaluations is scale. Most studies rely on ground-truth datasets containing on the order of 10^3 to 10^5 benign and malicious repositories. In contrast, our analysis of GitHub event activity logs shows that approximately 10^6 repositories are modified *per day*, raising questions about the applicability of these approaches to Internet-scale settings.

Several works have demonstrated strong performance using specialized architectures for detecting specific abuse classes, such as GitHub user bots [18]. While effective in their target domains, such approaches are often tightly coupled to a single attack type and do not generalize well across heterogeneous abuse behaviors. Further, models vary in their consideration of features derived from social network connections, file contents, and temporal patterns. In contrast, we focus on architectures and feature representations that are compatible with classical machine learning pipelines while still leveraging deep representation learning. By combining social, content, and temporal signals into natural language, graph-structured, and statistical features, our model is more expressive than prior approaches and better suited to expanding detection of the campaign families evaluated in this work.

Finally, prior work has highlighted the important distinction between repositories that are malicious and those created for educational or demonstrative purposes [41, 44]. We build on this distinction by explicitly separating *detection* from *takedown*. Educational repositories—such as proof-of-concept exploit code—are frequently repurposed for real-world attacks [6] and therefore merit detection, even if they are not themselves policy violations. In contrast, repositories hosting scams should be removed promptly upon discovery. In this work, we focus on the latter category. Educational repositories are a major source of false positives for purely content-based detection approaches; incorporating statistical and behavioral features mitigates this issue to some extent.

4 Methodology

As noted earlier, detecting scams and other forms of malicious content hosting on open-source software platforms requires operating under a set of real-world constraints that fundamentally shape system design. Unlike offline or retrospective analyses, production-scale detection systems must contend with incomplete and ephemeral ground truth, limited observability due to platform-imposed rate limits, and extreme class imbalance. Systems that ignore these constraints may achieve strong offline performance but fail when deployed in practice.

Below, we describe the primary operational challenges that motivate the design of our system.

Ground Truth Data. Effective modeling of heterogeneous abuse behaviors requires representative and high-quality ground truth. However, obtaining such data on GitHub is inherently challenging. Many malicious repositories are deleted shortly after discovery, erasing both their contents and metadata. Moreover, GitHub hosts a wide range of abuse types, each with distinct characteristics; unrepresentative or narrowly scoped ground truth datasets can yield optimistic evaluation results while performing poorly on previously unseen threats. *We address this challenge by modeling multiple classes of content-based abuse using datasets derived from crowdsourced reports and human-led discovery of novel threats. To support reproducibility and future research, we release our original datasets to the community.*

Rate Limiting and Observability. Even when ground truth exists, the ability to observe relevant signals at scale is constrained by platform-imposed rate limits. Most prior work on malicious entity detection on GitHub relies on patch-level features—such as full file contents, commit diffs, or line-level code changes. *To remain deployable at scale, our system is built around passive data and, when necessary, uses*

Table 1: **Comparison of ML-related research in GitHub safety.** ✓ indicates usage of feature class; ✗ otherwise.

Paper	Model	Scope	Type of Abuse	Evaluation set	Malicious set	Social Graph	File Contents	Temporal Stats
GitCyber2020 [51]	DNN	Hosting	Crypto scams	3,700 repos	1,500 repos	✓	✓	✗
Anomalicious2021 [16]	Rules	Supply chain	Malicious commits	115 repos	15 repos	✓	✓	✓
Meta-AHIN2021 [31]	GNN	Hosting	Covid scams	21,000 repos	7,000 repos	✓	✓	✗
SourceFinder2020 [32]	NLP	Hosting	Scams/hacking tools	2,000 repos	820 repos	✗	✓	✗
CLA-HG2022 [30]	GNN	Hosting	Covid/crypto scams	29,000 repos	11,000 repos	✓	✓	✗
Repo2Vec [33]	Embeddings	Hosting	Hacking tools	1,000 repos	430 repos	✗	✓	✗
StarScout [18]	Graph metrics	Reputation	Star bombing	330,000,000 repos	19,000 repos	✓	✗	✓
Our Work	DNN	Hosting	Scams/exfiltration	**14,000,000 repos**	**24,000 repos**	✓	✓	✓

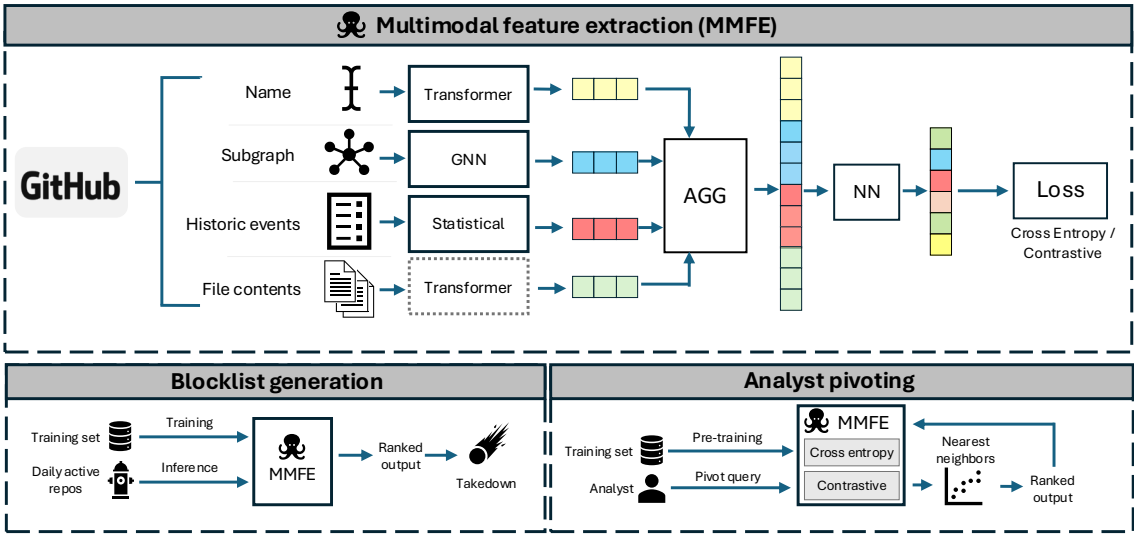

Figure 1: **Multimodal feature extraction and evaluation setup.**

lightweight Git client operations that retrieve a small subset of files (e.g., README.md). This design enables the use of content-based features when available without sacrificing scalability or violating platform rate constraints.

Model Quality Under Extreme Class Imbalance. Finally, even with adequate data, observability, and resources, detection quality remains constrained by base-rate effects [3]. As the number of inferences performed by a system increases, even low false-positive rates can overwhelm operators [1]. For example, a model identifying malicious commits within a corpus of 2.3 million commits per day at a 0.01 false-positive rate would generate approximately 23,000 false alerts daily—often far exceeding the number of true positives observed in practice. In such environments, precision, stability, and threshold calibration dominate traditional accuracy-based metrics. *We address this challenge through fine-grained evaluation and model selection strategies that explicitly account for class imbalance and operational false-positive budgets.*

Together, these constraints define the feasible design space for Internet-scale scam discovery systems and directly shape the architectural and modeling decisions underlying our approach. The resulting platform continuously ingests user, repository, and package activity; extracts multimodal features from heterogeneous data sources; and produces both a scalar maliciousness score and a vector embedding for each repository. These outputs support multiple operational workflows, including automated blocklist generation as well as analyst-driven threat hunting and investigation.

The system comprises four stages, illustrated in Figure 1, that include event ingestion, feature extraction, multimodal fusion, and prediction. The ingestion layer collects commit activity, repository creation events, starring behavior, and other social interactions from GitHub. For each repository, the system derives natural-language, graph-based, and statistical features, which are fused using a shallow neural network. The resulting unified representation is then passed to a task-specific model head, enabling either supervised classification or similarity-based retrieval via nearest-neighbor search. We discuss those specific steps in the following sections.

4.1 Multimodal Feature Extraction

Our multimodal design is a direct response to the operational constraints of Internet-scale scam discovery. Reliance on any single feature class is brittle in adversarial settings: attackers routinely adapt to evade detectors by manipulating the most salient signals. For example, rather than algorithmically generating repository names that are easily flagged, attackers increasingly mimic the names of popular benign repositories, undermining purely lexical or content-based approaches. Moreover, some abuse behaviors are inherently observable only through specific modalities. Reputation manipulation, such as inflating a repository's visibility via starring botnets, produces little semantic signal and can be detected only through graph-structured or statistical features.

These challenges are exacerbated by the practical constraints on observability, storage, and compute noted earlier. Platform rate limits restrict access to fine-grained patch-level data, malicious repositories are frequently deleted before comprehensive analysis is possible, and retaining full repository histories is prohibitively expensive at scale. Under these conditions, effective detection requires extracting robust signals from sparse, heterogeneous, and partially missing data.

Accordingly, we employ three complementary feature modalities that are informed by prior studies (see Table 1): (i) natural-language features that capture semantic intent in artifact names, documentation, and code when content is available; (ii) graph-based features that encode social and infrastructural relationships among users, repositories, and packages and remain observable even after content deletion; and (iii) statistical features that summarize temporal and behavioral patterns with minimal storage and compute overhead. Each modality is designed to remain informative when others are noisy, missing, or adversarially manipulated, enabling robust detection under realistic deployment conditions.

4.1.1 Transformers

Our multimodal feature-extraction pipeline incorporates two complementary natural-language inputs: (i) artifact identifiers (repository names, along with creator usernames) and (ii) the contents of selected repository files. Both inputs are encoded using bidirectional Transformer models [45], chosen for their ability to capture semantic structure in short, noisy text typical of open-source artifacts.

Artifact Identifiers. The embeddings that encode repository names and creator usernames capture lexical and semantic regularities that enable OctoWatch to identify algorithmically generated identifiers as well as coordinated, theme-based scam campaigns. Recent large-scale supply-chain attacks leveraged algorithmically generated repository names to exfiltrate stolen secrets to public GitHub repositories [40]. Similarly, scam campaigns often reuse distinctive themes—such as gaming cheats or cryptocurrency trading tools—in names of repositories and users.

File Contents. When available, we encode the contents of selected repository files, as these features provide strong semantic signal. However, reliance on file contents alone is impractical in operational settings: security takedowns frequently remove malicious repositories shortly after discovery, making repository content inherently ephemeral. As shown in Table 2, between 50–100% of repositories in our malicious datasets have been deleted. To account for this ephemerality during training, we randomly replace file contents with a `[MASK]` token in a subset of samples during each epoch. This augmentation discourages over-reliance on content features and encourages the model to fall back on other modalities when content is absent. At inference time, unavailable file contents are handled by supplying the same `[MASK]` token, ensuring consistent behavior between training and deployment. We found that this technique had a negligible effect on model performance.

4.1.2 Graph Neural Network

We model users and repositories in GitHub's social ecosystem as a graph, capturing interaction patterns that are difficult for adversaries to manipulate at scale. Prior work has shown that social and relational structures exhibit predictable regularities, making them more robust to evasion than intrinsic features such as names or content [48]. In the software development ecosystem, these graph structures naturally encode abuse behaviors such as bot-driven repository starring, where coordinated groups of automated accounts inflate the apparent popularity of malicious repositories.

To capture these signals, we construct a heterogeneous graph neural network (GNN) whose nodes represent users and repositories, and whose edges correspond to 15 distinct event types, including commit pushes and starring actions. We adopt an inductive learning approach based on *GraphSAGE*, motivated by both deployment and compute constraints. GraphSAGE generates node representations using localized neighborhood aggregation, allowing the model to operate efficiently on large, dynamic graphs without requiring full-graph traversal or retraining. To further isolate structural signal, we initialize all node features to a constant vector of size $[1] \times 10$. This design forces the GNN's non-linear message-passing layers to construct representations solely from relational structure, ensuring that graph-derived features remain independent of natural-language and statistical signals, which are modeled separately and fused at a later stage.

4.1.3 Statistical Features

We incorporate a combination of 33 novel and 7 established [43] statistical features (see Appendix C). The features

primarily model temporal relationships between entities that are not captured by the graph neural network (e.g., number of stars received in the first seven days of a repository's lifetime). To make statistical features compatible with feature fusion, we standardize each continuous feature using z-score normalization: for each feature dimension, we estimate the mean and variance on the training split only, then reuse those fixed statistics for validation, test, and online inference.

4.1.4 Modal Fusion

In the model fusion stage, the features extracted in each modality are combined into a single feature vector v, or:

$$v = \texttt{AGG}(\texttt{NLP}(\mathbf{x}), \texttt{GNN}(\mathbf{G}_x), \texttt{Stat}(\mathbf{x}))$$

We implement the aggregation function with a simple affine (or 'fully-connected') layer:

$$\texttt{AGG}(x, y, z) = W[x; y; z] + b$$

4.1.5 Supported Operational Workflows

The OctoWatch platform supports two operational workflows: (i) *blocklisting*, which requires highly precise binary classification under extreme class imbalance, and (ii) *analyst-driven pivoting*, which benefits from an embedding space where semantically and behaviorally similar repositories are colocated for similarity search and clustering.

For automated blocklisting, we train a classification head on top of the fused representation v using a weighted cross-entropy objective optimized for extremely imbalanced data. We use inverse class weighting so that errors on malicious samples incur a higher penalty than errors on benign samples.

To support analyst-driven pivoting, we learn an embedding space using a supervised contrastive objective. Contrastive learning models the geometry of the representation space, yielding embeddings that are better aligned with distance-based retrieval and clustering [46]. We include details on computing contrastive and cross-entropy loss in Appendix A.

5 Datasets

We evaluate our system using multiple malicious-artifact datasets collected from GitHub, a widely used open-source platform (OSP) with publicly accessible metadata. Table 2 summarizes the datasets used in our evaluation. Dataset selection is guided by real-world threat reporting and platform enforcement activity, rather than synthetic or purely academic benchmarks. Where possible, we rely on ground truth released by industry or prior scholarly work; in cases where such data is unavailable or incomplete, we construct new datasets through careful manual investigation and verification. Together, these datasets reflect realistic abuse scenarios observed in the wild and provide high-confidence labels suitable for operational evaluation.

Table 2: **Datasets used.** *Deleted* reports the fraction of malicious ground truth repositories that were no longer accessible on GitHub at the time of offline data collection.

	Dataset	Date	Samples		Deleted
			Benign	Malicious	
Offline	Shai-Hulud [40]	2025	100,000	11,199	100%
	Phishing scams	2025	100,000	2,100	13%
	Stargazer [42]	2024	100,000	1,414	100%
Online	Shai-Hulud [40]	2025	2,516,587	11,784	–
	Phishing scams	2025	13,591,491	4,261	–
	Stargazer [42]	2025	13,591,491	87	–
	90-day longitudinal study	2025-26	25,712,161	10,406	–

5.1 GitHub Threat Reporting

5.1.1 "Shai-Hulud" Worm

To evaluate our model's performance on a historic attack, we first construct a dataset derived from a large-scale supply-chain attack associated with the *Shai-Hulud* worm, disclosed in November 2025 [40]. This attack propagated through nearly 800 compromised npm packages and targeted developer environments to exfiltrate secrets such as API tokens and passwords. Infected hosts scanned for secrets using TruffleHog, a popular open-source, regex-based secret-scanning tool. Discovered secrets were then published to public GitHub repositories in one of two ways: if a GitHub API token was present, the malware created a new public repository under the corresponding account; otherwise, it searched for another infected user whose leaked secrets contained a valid token and published the data under that user's account. In all cases, repositories were created with randomly generated names consisting of 16–20 lowercase alphanumeric characters. We hypothesize that MMFE will be capable of identifying these algorithmically generated patterns.

To construct high-confidence ground truth, we began by sampling 5,000 repositories whose names were 16–20 characters long and lowercase alphanumeric and were created on November 24, the first day of the attack. We applied a series of conservative filters to remove obvious false positives, excluding repository names that (i) consisted of at least 75% of the characters from one of the 1,000 most common English words, (ii) contained the creator's username, or (iii) followed a simple word–number concatenation pattern (e.g., `reponame2025` or `2025compsciproject`). The remaining candidates were manually reviewed to eliminate residual false positives, resulting in a final dataset of 4,896 high-confidence malicious repositories associated with 411 compromised users.

5.1.2 Phishing Scams

Next, to evaluate each model's ability to surface previously unknown abuse starting with only a handful of samples from an active investigation, we construct a dataset of unreported phishing repositories luring users into downloading Windows-

based malware. This dataset reflects live malware campaigns that were ongoing at the time of discovery and had not yet been reported by existing threat intelligence sources. Specifically, we identified phishing repositories using a human-in-the-loop label propagation process.

Dataset construction began with a set of 11 seed repositories manually discovered through broad keyword searches related to communities frequently targeted by threat actors, including gaming cheats, cryptocurrency tools, and productivity utilities. From these seeds, we iteratively derived signatures to identify additional candidate repositories. For example, after identifying a credential-stealing lure masquerading as a Roblox cheat engine, we extracted its command-and-control URL, which directly led to the discovery of five additional repositories hosting the same malware campaign.

To ensure high-confidence ground truth, each candidate repository was subjected to automated verification using VirusTotal. Only repositories that linked to executables flagged as malicious by at least 15 independent antivirus vendors were included in the final dataset. This conservative verification criterion substantially reduces the likelihood of false positives while preserving coverage of the underlying campaign. Applying this process resulted in a final dataset of approximately 2,100 verifiably malicious repositories.

Case Study. Repositories from this campaign exhibit several distinct characteristics, making the campaign an interesting subject for ML-based detection. One such repository is `zuckkiller/kirmanjiku-2`, shown in Appendix B. This repository, like most others, contains an AI-generated README describing the project and directing users to download a (malicious) file to run the project. Other than the README, the only file in the repository is a zip file containing a Lua-based data stealer. The repository takes several steps to increase its reputation, including stuffing topical keywords in the README and tags, mimicking the name of a benign repository created days prior, and most notably, automatically pushing superficial commits to the repository several times a day to give it the illusion of recency. In Section 7.2, we discuss several variations of these patterns discovered during an online evaluation in late 2025.

5.1.3 Stargazer Ghosts

Finally, to assess the feasibility of graph-based modeling, we evaluate our model on the *Stargazer Ghosts* dataset, which consists of publicly released GitHub repositories implicated in campaigns that fraudulently inflate repository popularity using networks of compromised or attacker-controlled accounts [42]. This dataset was originally disclosed as part of an industry threat report and has since served as a benchmark in prior academic work [18]. The report notes that bot-driven star inflation is widespread and spans multiple independent campaigns, resulting in a dataset that is heterogeneous with respect to activity patterns and coordination structure.

All repositories in this dataset have since been deleted, and fewer than 1% appear in archival sources such as the Wayback Machine [18]. As a result, repository content is largely unavailable. We therefore treat this dataset as a stress test for detection under conditions of ephemeral content, and hypothesize that the underlying abuse can be captured through graph-structured and statistical features derived from starring behavior and temporal activity patterns. Further, many of the repositories in this dataset are themed scams (e.g., `Roblox-Hack`), which can be modeled by our NLP model.

5.2 Benign Samples

To simulate realistic operational conditions, we construct a test set that reflects real-world class imbalance. We construct a benign background set by sampling 100,000 repositories with `PushEvent` activity in the year prior to the experiments.

To minimize the risk of inadvertently including malicious projects, we apply two conservative filters. First, we exclude any repositories that were removed from GitHub due to DMCA takedowns or ToS violations. Second, for each remaining repository, we query VirusTotal for both the repository URL and all associated release artifacts; any repository for which VirusTotal reports malicious detections—either for the URL or for any release file—is removed from the benign set.

6 Results

In what follows, we evaluate our approach to address each RQ1–3 (see Section 1 for details). For **RQ1 (real-world effectiveness)**, we report offline benchmark results and a 45-day live deployment measuring detection capabilities under extreme class imbalance. For **RQ2 (threat coverage)**, we characterize the types of campaigns uncovered online and analyze the scope of several case studies. Finally, for **RQ3 (ecosystem impact)**, we quantify downstream harm by tracing cryptocurrency-draining malware discovered during deployment and estimating associated financial losses.

6.1 Experiment Setup

For training and inference, we use a single NVIDIA A30 GPU with 24 GB of VRAM, a configuration commonly available in mid-range production servers. Historical data storage and low-latency feature retrieval are handled using a ClickHouse database deployed on a standard CPU server with 40 cores and 256 GB of RAM. This setup reflects commodity workstation and server-class hardware, demonstrating that the system can be trained and operated without reliance on specialized or high-end infrastructure.

6.1.1 Data Sources

We rely primarily on GHArchive's public event logs to collect repository activity in order to minimize direct API calls and CDN probes to GitHub. However, extracting content-based features requires limited active scraping of repository data. During system design, we observed that the `git` client retrieves repository content via an alternative content delivery path distinct from GitHub's standard CDN (`githubusercontent.com`), and that this path supports higher sustained throughput.

To extract NLP features, we probe each repository's `README.md` file, if it is available. While malicious activity can use other files, `README.md` is the primary file displayed on a repository's landing page, and is therefore widely prevalent in scams. Our crawling relies on Git's sparse checkout functionality—supported by GitHub since 2020 [39]—to avoid downloading full repository contents. To determine whether a repository has been removed or made inaccessible, we employ the lightweight `git ls-remote` command. Following guidance from the GitHub team (see the Ethical Considerations section), we limited our crawling rate to 80% of the empirically observed threshold (approximately 300,000 files per hour) to maintain responsible usage.

6.1.2 Data Filtering

GitHub's interaction graph contains a small number of extremely high-degree nodes, such as automation accounts (e.g., `dependabot[bot]`) that interact with millions of repositories. These nodes introduce spurious short paths between otherwise unrelated repositories and substantially increase both the computational and memory cost of message passing. To prevent these artifacts, we remove all events associated with known GitHub automation accounts prior to graph construction. Even after bot filtering, the GitHub event graph remains too large (70 million edges). Following standard GraphSAGE practice, we adopt an inductive, neighborhood sampling approach. For each seed node (e.g., a repository in a training minibatch or labeled evaluation set), we construct an induced subgraph by sampling its one- or two-hop neighborhood. To bound per-node computation and control variance introduced by high-degree nodes, we cap the neighborhood fanout at 250 neighbors per relationship type. This reduces the number of edges by 70%.

6.2 Offline Evaluation

Model Selection. We selected architectures and hyperparameters over a five-fold cross-validation, selecting the model that maximized the mean Matthews Correlation Coefficient (MCC) over the validation folds. MCC is more appropriate than metrics like ROC-AUC and F1 in settings with highly imbalanced data [5] like ours. In particular, although the average MCC across datasets in our offline evaluation was 0.95, the corresponding average ROC-AUC was 0.99 and the average F1 score was 0.97, indicating that these metrics can mask substantial misclassification errors – an observation also noted by Yao et al. [50]. Accordingly, we report and optimize MCC throughout validation and model selection.

To select the best content extraction model, we evaluated four pretrained bidirectional Transformer encoders, namely BERT, CodeBERT, ModernBERT, and RoBERTa, against the scam dataset. Overall performance differences across models are small (less than 0.05 percentage points across evaluation metrics), with ModernBERT consistently achieving the highest scores. As a result, we adopt ModernBERT for all subsequent experiments. We also explored additional architectural and training refinements, including explicit regularization and class-imbalance–aware loss functions. These modifications yield at most a 2% improvement in performance. Because our research questions focus on comparative behavior and operational realism rather than maximizing absolute accuracy, we retain a simpler model configuration. This choice provides a lower bound on performance while improving reproducibility and reducing sensitivity to hyperparameter tuning.

Table 3: **Performance by dataset.** Metrics are for MCC-maximizing decision threshold over the validation set. Shown: Bag of Words (BoW) [32], Word2Vec (W2V) [32], Repo2Vec (R2V) [33], StarScout [18].

	Baseline				OctoWatch	
Dataset	**BoW**	**W2V**	**R2V**	**StarScout**	**Offline**	**Online**
Shai-Hulud [40]	0.62	0.88	0.91	0.00	0.97	0.77
Phishing	0.10	0.36	0.37	0.00	0.99	0.94
Stargazer [42]	0.00	0.00	0.07	0.45	0.90	0.35

Comparison to Baselines. Table 3 summarizes the performance of OctoWatch and four baseline models [32, 33]. Multimodal fusion consistently outperforms specialized NLP baselines. For the scam dataset, the only dataset with recoverable repository contents, Transformer-based semantic features achieve substantially higher accuracy than prior malicious-repository classifiers built on shallow bag-of-words representations or pretrained word2vec embeddings of repository metadata and contents [32]. These gains are most pronounced when file content is unavailable or unreliable. In the Shai-Hulud campaign—where repository names are randomly generated, hash-like strings—the multimodal approach achieves solid performance (0.97 MCC) using repository names alone, outperforming prior baselines by nearly a factor of three. While earlier NLP-based methods perform competitively on this dataset (reaching up to 0.88 MCC), they still fall short of the multimodal approach. Repo2Vec [33] models perform similarly across each dataset.

This gap is explained in part by limitations in feature expressivity and model capacity. Classical bag-of-words and word2vec baselines rely on tokenization schemes that struggle

Feature	MCC		
	Shai-Hulud	**Phishing**	**Stargazer**
Content	⊥	**0.94**	⊥
Repository	**0.97**	0.08	0.09
Graph	0.52	0.90	**0.69**
Stat	0.51	0.70	0.51
All	**0.97**	**0.99**	**0.90**

Table 4: **Feature contributions.** MCC for individual unimodal feature families and the full multimodal model. ⊥ denotes cases where content-based evaluation was not possible due to deleted repositories.

to segment repository and username strings into meaningful units, especially for camelCase identifiers, typosquatting variants, and randomly generated or hash-like names. This issue is most visible in the Shai-Hulud dataset: inter-token relationships in random identifiers are sparse, and smaller NLP baselines do not have enough capacity or training data to learn robust representations of these strings. Modern Transformer encoders instead use subword tokenization, such as byte-pair encoding, which better preserves useful lexical structure in repository identifiers even when the text is far from natural language. Although this improves performance, OctoWatch still outperforms the best NLP baseline by 15% on Shai-Hulud, indicating that the gains are not solely due to tokenization.

We also evaluate graph-based baselines, motivated by the prevalence of automated starring behavior in the Stargazer dataset. Specifically, we reproduce StarScout [18], a graph-based method designed to detect anomalous starring activity. StarScout identifies fake stars using two signatures derived from GitHub event logs: (1) a *low-activity* signature, which flags accounts that become dormant after starring only a few repositories, and (2) a *lockstep* signature, which detects groups of accounts repeatedly co-starring the same repositories. We compare StarScout against OctoWatch's graph-only ablation to isolate the contribution of our relational modeling. On the Stargazer dataset, OctoWatch's graph-only model achieves 1.5× higher recall than StarScout, largely because StarScout filters repositories with fewer than 50 stars during post-processing to reduce false positives. Moreover, StarScout fails to detect *any* repositories in the Phishing or Shai-Hulud datasets. In both cases, over 75% of repositories have degree one, typically consisting of a single maintainer with little or no observable neighborhood structure, limiting the effectiveness of graph-based detection. Other graph-based approaches [15, 30, 31] additionally rely on manually curated content keywords, which complicates reproduction and increases susceptibility to mimicry attacks. In contrast, OctoWatch combines semantic, statistical, and relational signals, enabling robust performance even when individual modalities are sparse, weak, or unavailable.

Feature Contribution. By conducting a series of feature ablation experiments (Table 4), we find that multimodal feature extraction can adapt to the graph, statistical, and natural-language signals that are most informative for a given ground-truth dataset. These results show that no single modality is sufficient for threat modeling across all settings. For example, on the botnet-driven Stargazer dataset, a graph-only model achieves an MCC of 0.69, whereas an NLP-only model performs poorly (0.09 MCC).

In contrast, for the Shai-Hulud dataset—whose repositories exhibit distinctive textual patterns—an NLP-only model achieves near-perfect performance, reaching 0.97 MCC. Interestingly, the actor behind the phishing campaign appears to have deliberately attempted to evade content-based detection by manipulating repository names, resulting in poor performance for this modality. In our analysis of the Phishing dataset, we found that 95% of malicious repository names were *mimicked*: each name had appeared in a benign repository less than ten days before the corresponding malicious repository was created.

7 Online Evaluation

While benchmark datasets enable consistent model comparison and selection, security-focused machine learning systems must be evaluated under realistic deployment conditions to accurately assess their operational effectiveness [2]. To this end, we simulate a multi-day online deployment where a detection model processes GitHub's live event stream—comprising nearly one million newly active repositories per day—and identifies previously unseen malicious repositories hosted on the site. We evaluate online performance across two tasks: attack backtesting and campaign expansion.

In the attack backtesting task, the model is trained on a partial sample of repositories published on the first day of the Shai-Hulud supply-chain attack and is then evaluated on its ability to flag newly emerging secret-exposing repositories associated with the same campaign. In the campaign expansion task, the model is trained on labeled examples drawn from multiple known campaigns modeled in the Phishing and Stargazer datasets. Although we primarily evaluate OctoWatch on its ability to discover new examples of these scams, we also find that OctoWatch can detect malicious content beyond the campaigns we directly modeled.

It is infeasible for a human analyst to validate millions of results each day, so we present our online analysis through the lens of an analyst concerned with the trade-off between discovering new examples and trudging through false-positive results. The analyst evaluates the first 1,000 results, then continues to evaluate results until the precision decreases below 80%. We define precision at rank r as the proportion of results up to rank r correctly identified as true positives.

Two authors independently verified the ranked results, simulating analysts triaging results in a blocklist. They reviewed

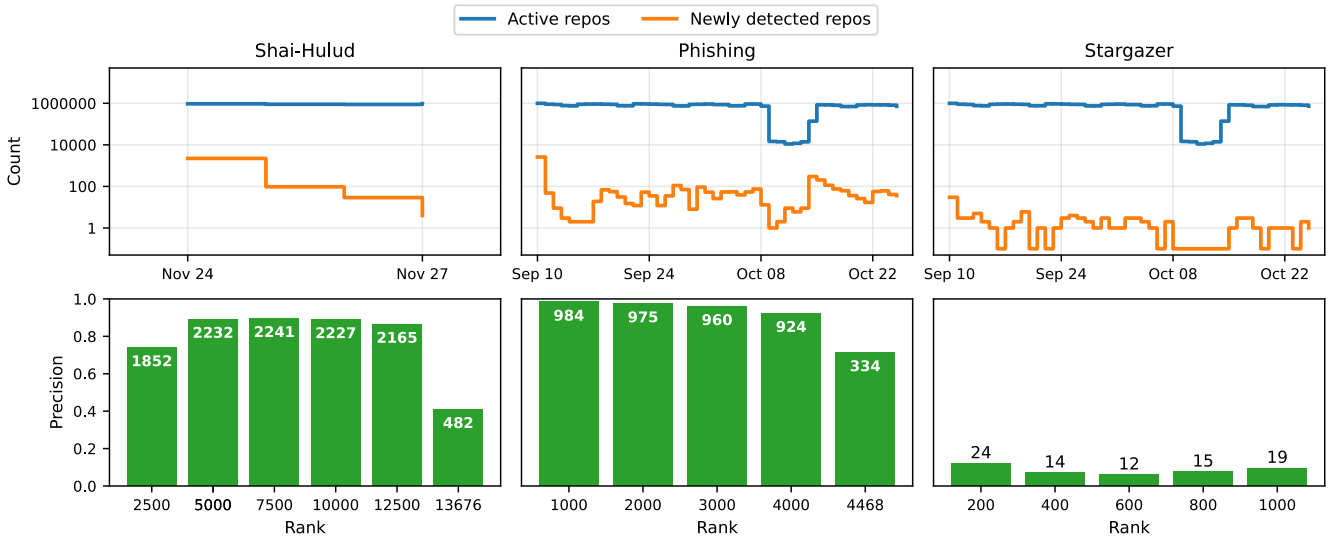

Figure 2: **Malicious detections in online evaluations.** Top: the daily number of active repositories and newly detected malicious repositories in each evaluation. Bottom: the precision across ranges of rank-ordered, manually validated malicious repositories found in the aggregated results for each evaluation, along with the corresponding counts of malicious repositories identified in each range.

daily results in ranked order, classifying each repository as either violating GitHub's abuse policies (true positive) or benign (false positive). One author labeled 4,468 repositories until the target precision threshold was met, while the second independently labeled a 50% sample as a reliability check. The resulting inter-rater agreement was high, with only 16 disagreements and a Cohen's Kappa of 0.88.

In both evaluations, we found that generating predictions for a full day of GitHub activity using all four feature classes took around 30 minutes on our hardware. Live repository crawling was the main bottleneck—however, at 300,000 files per hour, crawling all READMEs published each day was easily attainable in around three hours.

7.1 Backtesting on Known Attacks

We first turn to a backtest of our model in detecting repositories from the Shai-Hulud attack, which spanned November 24–27, 2025 and generated thousands of repositories hosting leaked secrets [40]. Given a training set of 5,000 repositories, OctoWatch discovered an additional 10,573 secret-exposing repositories, with the peak in detections on November 24 mirroring the findings of threat intelligence reporting.

Figure 2 shows how our approach reliably detects Shai-Hulud in terms of precision. In the second row, precision is computed over each rank bucket (e.g., for Shai-Hulud, the first bar represents a 78% precision over the first 2,500 results). The number of true positives is denoted on each bar. In practical terms, an analyst would identify over 10,500 true positive repositories while incurring less than 1,500 total false positives. Beyond this operating point, precision declines due to a sharp increase in false positives. Manual inspection indicates that the majority of these false positives correspond to automatically generated repositories used for orchestrating GitHub workflows. Notably, Shai-Hulud repositories exhibit nearly identical file names and directory structures, suggesting that incorporating content-based features could further improve detection performance, albeit at the cost of additional crawling overhead. We do not evaluate this extension directly, as all known Shai-Hulud repositories had been removed by GitHub as of November 28, 2025.

Table 3 indicates a decrease in model quality between online and offline mode. When evaluating each repository for a given day, we found a significant number of false positives concentrated near rank 1—on the first day of the evaluation, this led to a 40% decrease in precision for the first 2,000 repositories. The main sources of these FPs were newly created repositories with hexadecimal repository names, which were not present in our benign background noise dataset and thus are out-of-distribution. Retraining with just 100 of the randomly generated hexadecimal repository names increased MCC by 4.3% (0.95 MCC).

While heuristic techniques can perform similarly to our model in identifying Shai-Hulud repositories from GitHub's activity stream [40], an automated, data-driven approach has significant benefits. Most importantly, an ML system could reduce exposure of developer secrets by accelerating automated rapid response efforts. The impact of Shai-Hulud could have been considerably reduced had an ML system automatically disabled public visibility for repositories likely to have

been created by a Shai-Hulud infection. A rapid deployment of OctoWatch in this setting could have reduced the number of victims with secrets exposed by 36.3% with under 1,000 benign repositories incorrectly made private.

7.2 Campaign Expansion

To assess **RQ1 (real-world effectiveness)**, we evaluated the models trained on the Phishing Scam and Stargazer datasets over 45 days. In total, OctoWatch identified 4,261 new malicious repositories not present in our ground truth datasets or public threat reporting from URLHaus. As 4,177/4,261 repositories were identified by the Phishing model, we primarily focus our analysis there. The output from the Phishing model primarily consists of repositories attributable to the same threat actors modeled in the ground truth (98%), as well as repositories associated with previously unseen campaigns. We return to a discussion of one such case in Section 7.4.3.

Figure 2 illustrates the daily output and performance of the model against each dataset. On average, 46.8 new scam repositories were created every day, each from a unique GitHub account. This suggests that the adversary perpetrating this campaign relies heavily on automation to post high quantities of repositories. Typically, we identified new malicious repositories within 24 hours of their creation dates. The most notable exception to this is the influx of detections on the first day of the experiment, where OctoWatch identified more than 100× as many detections as future days. Many scam repositories were created *before* the beginning of our evaluation period but received new automated commits multiple times a day. As a result, these repositories were included in the daily feed and were quickly detected on day 1. The dip in October is due to GHArchive producing limited data for a 5-day period, but the repositories were quickly flagged when the data was restored.

Figure 2 shows the effectiveness on the phishing dataset. When output is triaged in ranked order, an analyst encounters only 48 false positives among the first 2,000 flagged repositories. This operating behavior enables large numbers of malicious repositories to be detected, blocked, and ultimately taken down before false positives become prevalent.

Running the model trained on the Stargazer dataset also led to the discovery of several campaigns (on average, four per day) not included in the ground truth dataset, demonstrating our model's effectiveness even when content takedowns rendered file contents unavailable. However, the model's performance is impaired significantly without access to NLP data. Many Stargazer repositories have few stars, reducing the amount of data available to the GNN. Despite evidence of botnet-driven starring, we found that the average number of stars per repository was 15—this low-volume starring activity would not have been detected by prior work [18], where repositories are required to have at least 50 stars. In some cases, our model combined NLP features with graph structure to identify malicious repositories with just one star. Interestingly, some true positive repositories impersonating popular open-source projects and brands intentionally inserted typos or abbreviations into their repository names (e.g., AdobeeCreativeeCloud, Offi-2025-Win) to evade detection by a system or human relying on search keywords. Transformer-based models, on the other hand, capture semantic and lexical similarities in repository names and therefore are better suited for modeling these modifications. Our error analysis suggests that false positives primarily consisted of AI-generated repositories, as well as Web3, proxy/VPN, and automation projects. False negatives included repositories with weak graph evidence or sparse metadata, including low-reputation software cracking tools.

This analysis shows how with only 291 false positives, online deployment of our model led to a 3× increase in malicious repositories versus our Phishing ground truth set.

7.3 Operational Considerations

To further assess **RQ1 (real-world effectiveness)**, we evaluated two optimizations that improve the model's suitability for operational use: automatic triage and retraining.

7.3.1 Automatic Triage

In a large-scale deployment like ours, manual triaging of results could take hours due to the sheer volume of true positives in the result set. We therefore evaluate how well the system's outputs can be triaged automatically, either by restricting all flagged repositories or by verifying maliciousness with threat intelligence platforms like VirusTotal.

One strategy would be to take down or restrict public access to *all* repositories flagged by OctoWatch and then manually resolve false positives as they are reported by users. On the Phishing dataset, such an approach would reduce analyst workload by 13×, but would decrease precision by 8%. While effective, this strategy is likely inoperable in practice, given that it could lead to serious collateral damage.

A more conservative approach for improving precision at the cost of lower recall is to apply a post-processing step that extracts potential malicious artifacts from flagged repositories and to externally verify their maliciousness using VirusTotal. We evaluated the efficacy of this method on the Phishing dataset by uniformly sampling 5,000 repositories from the 13.6M repositories observed during the 45-day evaluation. Each sampled repository contained either releases or references to .zip files in their README.md files—two malware distribution methods employed by the scammers. Within this sample of 5,000, 139 were malicious and 4,861 were benign. We queried VirusTotal for the verdicts of each of these files, finding that 106/139 (76%) of the repositories were malicious.

This result shows that VirusTotal can substantially increase confidence in automatically triaged true positives, but can

introduce considerable numbers of false negatives, namely scams that do not link to malicious executables from the GitHub README. As a result, this automation reduces the recall of the model to 67.8% (40/59) versus the manual labeling process. Further, a malicious classification from VirusTotal does not necessitate repository removal—for example, legitimate repositories linking to legitimate software frequently co-opted by adversaries (e.g., `ngrok`) and cybersecurity labs linking to non-harmful malware and hacking tools could lead to incorrect takedown if using VirusTotal as a sole ground truth source.

7.3.2 Retraining

To further understand OctoWatch's suitability for operational deployment, we evaluated its ability to leverage feedback from humans and secondary systems to improve performance.

Shai-Hulud. We found that small-sample retraining can reduce false positives when identifying historic malicious repositories. In the Shai-Hulud backtest, the original model assigned high scores to newly created benign repositories with hexadecimal, randomly generated names. These repositories resembled Shai-Hulud artifacts under the repository-name modality, but manual inspection showed that many were benign repositories created by automation tools for workflow orchestration. Because this behavior was absent from the original benign background set, these examples appeared near the top of the daily ranking and produced a large early precision drop. Adding only 100 such benign hard negatives to the training set increased MCC by 4.3%, reaching 0.95.

Phishing Scams. Beyond the primary 45-day evaluation, we conducted a separate retraining study to assess OctoWatch's ability to adapt to concept drift over a longer operational horizon. Despite our efforts leading to the takedown of 4,261 scam repositories created prior to October 24, 2025, the actor modeled in the Phishing dataset continued to create scam repositories beyond this period. This provided us with labeled data spanning an extended 90-day period from October 25, 2025 to January 23, 2026. We divided the full evaluation into three segments: P1 (September 10 – October 24, 2025), P2 (October 25 – December 1, 2025), and P3 (December 2, 2025 – January 23, 2026). Every seven days within the P2 and P3 windows, we selected newly created repositories using stratified sampling over OctoWatch's output distribution. We then ran the VirusTotal-based automatic triage process on the sampled repositories to obtain labels for retraining and evaluation.

We then compared a baseline model trained solely on the offline ground truth with a retrained model that incorporated 80% of the repositories from each period, evaluating both on the held-out 20%. Model performance on P2 and P3 with and without retraining remained high. Table 5 shows that performance does not significantly degrade over time, and is only marginally improved with retraining. Although we did not observe significant behavioral changes on this specific campaign during the extended evaluation period, we still recommend that periodic retraining be used in operational settings to detect and combat concept drift. We note that because the stratified sampling technique used in this evaluation equally samples each part of the class probability distribution, it could still undersample drifted samples.

Table 5: **Baseline versus retrained model performance by period on the Phishing scam dataset.**

Model	P1		P2		P3	
	F1	MCC	F1	MCC	F1	MCC
Baseline	0.967	0.878	0.996	0.925	0.991	0.942
Retrain	–	–	0.994	0.881	0.993	0.954

7.4 Case Studies

To provide more insight into the threat activity clusters we identified, we characterize **RQ2 (threat coverage)** and **RQ3 (ecosystem impact)** through an in-depth analysis of two case studies derived from the online results.

7.4.1 Scams Targeting Windows Users

The vast majority (98%) of repositories led users to .zip files that matched the tactics used by the SmartLoader malware family [28]. A common characteristic of this campaign was the extensive use of generative AI to populate repositories with superficially plausible content. One README file consisted only of a message refusing to generate malicious content in an apparent invocation of model safeguards. Interestingly, while nearly all repositories used similar malicious tactics, such as commit inflation and SEO-optimizing tags, around 50% of repositories linked users to GitHub releases while the other 50% referenced code committed to the repository. While we did not find conclusive evidence pointing toward actor attribution, one repository contained a mocking message to security defenders (written in Russian). In the 45-day online evaluation, a primary source of false positives for each dataset was persistent behavior from automated repositories. These tools produce large commit volumes, interfering with the model's statistical features.

Engagement & Impact. To estimate engagement with these scams, we aggregated historic star events for each repository and, when available, we scraped file download counts for each scam repository surfaced by the online evaluation.

We observed a considerable amount of starring behavior that inflated scam repositories' popularity—more than 1,000 users starred repositories later identified as malicious. 64 of these users were affiliated with legitimate GitHub organizations, implying that they were authentic users unknow-

ingly promoting malicious repositories. Assuming these accounts are not compromised, this provides a conservative lower bound on exposure and indicates that the attacker's SEO efforts succeeded.

10% of the scam repositories exceeded 10,000 downloads, representing 4× the activity of the daily median repository. Such extreme amplification is consistent with automated download inflation used to manipulate repository visibility, a tactic previously observed in the npm ecosystem [35]. Excluding these amplified repositories, releases from scam repositories averaged 44 downloads each.

To quantify the broader value of our findings to the TI community, we also measure *exclusive contribution* [25] against two public threat intelligence sources: URLHaus and VirusTotal. Over a 45-day deployment, our approach identified 2,124 unique malicious repositories, 90% of which were absent from URLHaus. In fact, a 45-day deployment of OctoWatch produced 2× more true positives than all URLHaus contributors combined over the last three years. Only nine repository URLs were flagged by any VirusTotal vendor, highlighting the limited coverage of existing feeds and the operational benefit of proactive, platform-native discovery.

We also extracted the alleged malware binaries associated with each repository verified in the 45-day evaluation. Compared to the samples in the training set, which were flagged as malicious by 20 vendors on average, some more recent binaries had only been flagged by two vendors. Upon closer inspection, we found that 75% of binaries had never been uploaded to VirusTotal, suggesting a severe gap between GitHub-based threats and TI production and sharing. The other 25% were uploaded up to 180 days before the repositories were created, indicating that although this actor rapidly created GitHub repositories, they reused malware files.

Besides the samples attributable to SmartLoader malware, we discovered instances of Android malware (5 repositories), PowerShell-based malware (3 repositories), and malware hosted on external file sharing websites (3 repositories). The presence of these results in the data implies that while the blocklisting model primarily flags malicious repositories from the same activity cluster as the training data, it can generalize to other groups when there is overlap in malicious tactics.

7.4.2 Pivoting

While the abundance of Windows-targeting scams in our dataset demonstrates OctoWatch's ability to surface large volumes of repositories given ground truth samples drawn from the same campaign, the results also contained examples of other scam campaigns not represented in our ground truth. An analyst may wish to pivot from this subclass of repositories to uncover additional, related hosting infrastructure belonging to that campaign. We therefore evaluate the suitability of our architecture to a pivoting workflow, then use the resulting expanded cluster to catalyze a further investigation into an active cryptocurrency campaign.

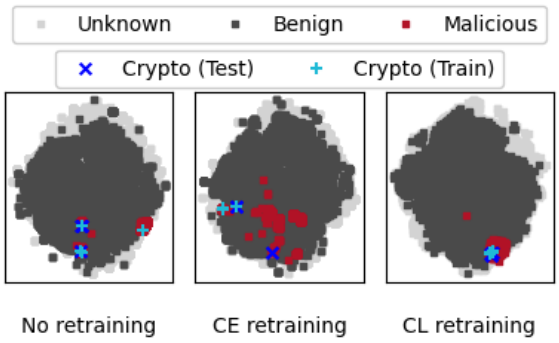

Figure 3: **t-SNE projection**, online evaluation

Pivoting places different demands on the model than binary classification. High classification accuracy alone does not guarantee that related artifacts are embedded near each other in feature space. Figure 3 illustrates this distinction. When the model is trained solely with a cross-entropy (CE) objective, repositories associated with cryptocurrency scams are often separated in embedding space, even though they are individually classified as malicious. Exchanging the cross-entropy loss function with a contrastive loss (CL) function causes all the crypto repositories to be embedded near each other.

We fine-tuned the model on the crypto case study using CL and manually analyzed the 50 nearest neighbors, a process which took under five minutes on our hardware. We found that pivoting can increase the number of repositories for a given threat cluster by up to 3× (15 new repositories). To quantify pivoting performance more systematically, we compute the mean reciprocal rank (MRR) over a held-out set of artifacts associated with each case study. CE retraining produced meaningful gains versus a baseline (13% increase in MRR), indicating that objectives optimized for binary discrimination do not reliably structure embeddings for nearest-neighbor retrieval. In all, we found that while the CE-based model generates precise results, CL narrows the analyst's search space by up to 99% when just three examples are provided.

13 of the 15 repositories directed users to send funds to a malicious smart contract written in Solidity, while the other two repositories enticed victims to run a JavaScript-based cryptocurrency stealer. In what follows, we further analyze the scope and impact of the 13 malicious smart contract scams.

7.4.3 Cryptocurrency Scams

Repositories in the malicious smart contract cluster distributed software masquerading as legitimate profit-generating tools, including automated trading strategies and compromised mining pool services. These scams were specifically crafted to target sophisticated users with experience deploying and interacting with smart contracts. Lures typically included doc-

umentation, tutorial-style videos (e.g., hosted on YouTube), and fabricated user comments asserting successful profits. But in reality, the advertised contracts simply drained victims' wallets by redirecting funds to attacker-controlled wallets.

To assess how successful those lures were, we collected the smart contracts that were directly embedded in repositories within the cryptocurrency scam cluster or hosted on external platforms linked from repository `README` files, skipping cases where externally linked files are no longer accessible. Due to the simplicity of the obfuscation, we manually analyzed the contracts to recover two wallet addresses, though we note that automatic wallet address deobfuscation techniques exist [24]. Similar to the expansion done using similar contract bytecode [24], we used GitHub's code search to locate contracts that share matching obfuscation function signatures.

While code search is effective at identifying contracts that implement the same scam logic but use different destination wallets, it is insufficient as a standalone signal for flagging malicious activity. In practice, this approach also matches benign repositories maintained for educational, research, or threat-hunting purposes. In our analysis, 24 out of the 71 (34%) repositories identified via code search fell into these benign categories and would therefore constitute false positives if acted upon. Therefore, we restrict the use of code search to expanding the set of attacker-controlled wallets associated with a given scam, rather than as a basis for reporting repositories.

To expand our analysis to repositories that were created earlier than our observation window and are no longer actively maintained, we searched Etherscan for smart contract source code that transferred funds to the extracted wallet addresses. Despite source-code availability for smart contracts on Etherscan being rare [53], this process revealed an additional obfuscation signature, which we used to further expand our GitHub search, leading to 30 new wallet addresses.

Engagement & Impact. To estimate the financial impact of these scams, we adopt a conservative methodology following the guidelines of Gomez et al. [14]. We restrict analysis to direct deposits into attacker-controlled wallets, explicitly exclude addresses labeled as exchange services by Etherscan, and compute revenue using both the historical Ethereum price at the time of each transaction and its present-day value.

Using this approach, we identify over 5.9 million USD in cryptocurrency transactions on the Ethereum blockchain alone — worth 6.9 million USD (as of January 2026) — that have been funneled into attacker-controlled wallets. This figure likely underestimates [17, 23] the true amount as we do not consider alternative smart contract platforms (e.g., Solana, Cardano), and very few contracts on Ethereum have associated Solidity source code (as opposed to EVM bytecode) [7].

The wallet addresses we recover include known cryptocurrency scam wallets [24, 26], yet several remain active, with direct deposits as recent as January 24, 2026. This persistence suggests that emerging scam campaigns evade timely detection and reporting. In contrast to older campaigns that relied on fewer, large-value transactions, recent campaigns favor swindling smaller amounts from a broader victim base, as depicted in Figure 4. Our multimodal approach effectively identifies wallets regardless of transaction strategy by learning relational patterns and dynamics of scam repositories.

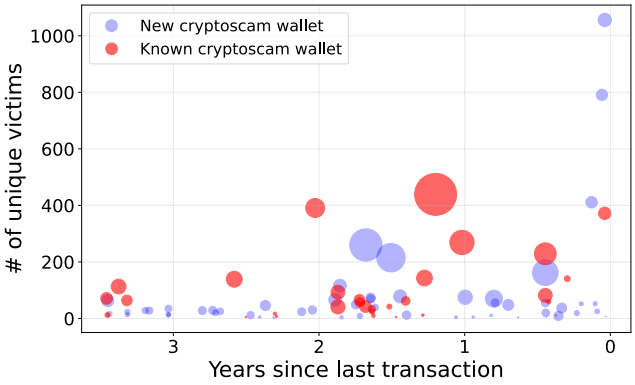

Figure 4: **Wallets sorted by recency of deposits.** Color indicates whether the wallet is associated with cryptoscams in Etherscan or Google results; size indicates lifetime deposits.

Analysis of the repositories surfaced by the Stargazer model revealed an additional set of 79 cryptocurrency-related scams advertising a so-called "flashing" technique for spoofing cryptocurrency transactions. These repositories primarily function as lures, redirecting users to off-platform websites that describe the purported spoofing capabilities and solicit payment through web-based checkout interfaces. Among the 68 repositories containing external URLs, 15 pointed to websites that were still active at the time of analysis; notably, five of these URLs had no detections on VirusTotal. In addition, 37 repositories instructed users to contact "customer support" via Telegram, further indicating an effort to move victims off-platform and evade ecosystem-level detection.

7.4.4 Vendor and Community Disclosure

Beginning in November 2025, we responsibly disclosed the abusive repositories identified during our study to GitHub's Trust and Safety team. This disclosure helped mitigate ongoing harm by enabling the removal of active malicious content, and allowed us to assess the operational relevance of our findings by observing whether platform enforcement actions aligned with our detections [25]. All reports were submitted through their abuse-reporting workflow. In addition to the list of repositories, we provided the team with context surrounding the types of threats, details of our VirusTotal validation process, and highlighted particularly risky repositories (e.g., those that were gaining traction as identified by their growing star counts). We also offered access to our internal validation tool to assist in triaging the large number of repositories.

GitHub responded promptly to our reports: in some cases,

repositories were removed within four hours, and in nearly all cases within 48 hours. This responsiveness underscores both the urgency of the reported threats and the practical utility of our findings. In total, these disclosures resulted in the removal of thousands of phishing repositories. For the off-platform findings, we reported all associated domains and messaging accounts to the appropriate abuse reporting channels, including domain registrars and Telegram. We give more details of our disclosure process in the Ethical Considerations section.

8 Discussion

Implications and Recommendations. Our findings highlight several implications for both security practitioners and the broader research community. For practitioners, our analysis reveals unexplored potential for ML-based systems trained on publicly available information to combat widespread scam campaigns. For researchers, we caution that while machine learning approaches for detecting malicious packages and repositories hold significant promise, their utility depends on careful consideration of class imbalance, ground truth biases, and reproducibility. When feasible, releasing curated datasets, including ephemeral artifacts such as file contents that may be deleted shortly after discovery as a result of takedown actions, can substantially improve reproducibility and longitudinal analyses. That said, researchers should remain cognizant that certain repository categories (e.g., those involving copyright infringement or DMCA violations [12]) may carry legal or ethical considerations that restrict data sharing.

Adversarial Attacks. Attackers aware of detection systems may attempt to manipulate repository names, README contents, graph structure, or temporal activity patterns to reduce detection confidence. We observed such behaviors in the wild. In the Phishing dataset, 95% of malicious repositories adopted names previously associated with benign repositories fewer than ten days earlier, a tactic that directly degrades NLP features derived from repository names. Attackers could also compromise user accounts to evade statistical and graph features that capture account age and reputation.

Despite these attacks, OctoWatch's multimodal design substantially increases the cost of evasion: because detections are driven by multiple independent signals, manipulating any single modality is often insufficient to evade detection.

Another concern is content collection latency. In our deployment, inference over one day of GitHub activity requires approximately 30 minutes, while crawling all READMEs takes roughly three hours. Attackers could exploit this gap by completing campaigns before content features are collected. Although this represents a valid attack vector, it does not undermine OctoWatch's practical utility: the crawl interval bounds detection delay and increases the operational cost of abuse by forcing adversaries to rotate accounts and infrastructure more frequently. Future deployments could further mitigate this risk by prioritizing content collection for high-risk repositories identified using partial feature vectors.

Limitations. Our evaluation considers a set of high-impact abuse campaigns rather than exhaustively covering all types of abuse from prior work. While the proposed approach is designed to generalize to other categories of GitHub abuse, including supply-chain compromise and leaked secrets, we do not evaluate these in depth. Extending the methodology to such settings may require incorporating additional data sources or altering feature extraction processes, which we leave to future work. A particularly interesting area of study would be the combination of these techniques with multi-file embeddings or static/dynamic code analysis techniques, which are natural extensions of our framework.

9 Conclusion

This work demonstrates that despite the critical role of OSP vendors in the Internet ecosystem, there is a gap between frontier ML techniques and their implementation in the security community's battle against abuse on these OSPs. This gap is driven partially by a lack of real-world insight in academic evaluations of these systems and a small amount of publicly available ground truth. By combining semantic, relational, and behavioral features, we show that it is possible to outperform baseline models and to quickly and reliably detect new instances of widespread scam campaigns over an extended 45-day period. Many of these scam campaigns were absent from widely used open-source intelligence feeds. Our work led to the removal of thousands of malicious repositories as part of our consultations with GitHub's Trust and Safety team.

Acknowledgment

The authors thank the anonymous reviewers for their detailed feedback that helped improve the paper. We also express our profound gratitude to the GitHub employees who engaged with us during the responsible disclosure process. This material is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001123C0035. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. Department of Defense (DoD).

Ethical Considerations

This research impacts three stakeholders: GitHub and its users, the security community, and researchers.

GitHub and its users: We took numerous steps to minimize the risk of negative impacts to GitHub and its users during data collection. Prior to our experiments, we coordinated with GitHub to establish a point of contact and ensure compliance

with their Acceptable Use Policy, including scraping from a single system and IP address [13]. We further reduced risk by limiting our crawling rate to 80% of the empirically observed threshold (approximately 300,000 files per hour). Disclosure of our findings to GitHub's Trust and Safety team resulted in the takedown of over 4,000 malicious repositories. Taken together, our precautions and outcomes suggest that our work provided a net benefit to GitHub and its users.

The security community: Beyond the interactions discussed in Section 7.4.4, we also shared URL and domain indicators of compromise (IoCs) derived from our analysis with the broader security community by submitting them to several established cyber threat intelligence exchanges (URLHaus and PhishTank). Although the set of exchanges we reported to was not exhaustive, they routinely redistribute submitted IoCs to downstream vendors and platforms, enabling wider detection and mitigation throughout the threat intelligence ecosystem [9]. We also disclosed cryptocurrency wallet addresses associated with confirmed scams to Etherscan, a public blockchain intelligence platform, to increase awareness among investigators, exchanges, and wallet providers. For cases where we found malware hosted on attacker-owned websites, we reported the associated domain and URL to both the appropriate registrar and URLHaus. While criminals could adapt to our detection methods once this study becomes public, this limitation is inherent to much of security research and, in our view, does not outweigh the benefits of disclosure.

Researchers: This study poses minimal risk to the researchers, with the primary hazard arising from accidental exposure to malware from analyzed repositories. We mitigated this risk by relying on state-of-the-art cloud-based sandboxes for inspecting suspicious binaries.

Open Science

Our code and datasets are available online (link). This repository includes the code used for model training, evaluation, and analysis. The Shai-Hulud dataset is available upon request to verified researchers—we omit this dataset from public release as it includes information about the victims of the Shai-Hulud attacks.

References

[1] Bushra A Alahmadi, Louise Axon, and Ivan Martinovic. 99% false positives: A qualitative study of SOC analysts' perspectives on security alarms. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2783–2800, 2022.

[2] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, 2022.

[3] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[4] Alan Cao and Brendan Dolan-Gavitt. What the fork? finding and analyzing malware in GitHub forks. In *Network and Distributed System Security (NDSS) Symposium 2022*. Internet Society, 2022.

[5] Davide Chicco and Giuseppe Jurman. The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. *BioData Mining*, 16(1):4, 2023.

[6] Cybersecurity and Infrastructure Security Agency (CISA). Publicly available tools seen in cyber incidents worldwide. https://www.cisa.gov/news-events/cybersecurity-advisories/aa18-284a, June 2020. Alert Code AA18-284A; Last Revised June 30, 2020.

[7] Isaac David, Liyi Zhou, Dawn Song, Arthur Gervais, and Kaihua Qin. Decompiling smart contracts with a large language model, 2025.

[8] Ruian Duan, Omar Alrawi, Rohan Kasturi, and Brendan Saltaformaggio. Towards measuring supply chain attacks on package managers for interpreted languages. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[9] Tillson Galloway, Omar Alrawi, Allen Chang, Athanasios Avgetidis, Manos Antonakakis, and Fabian Monrose. Actively understanding the dynamics and risks of the threat intelligence ecosystem. In *Network and Distributed System Security (NDSS) Symposium 2026*. Internet Society, 2026.

[10] Tillson Galloway, Kleanthis Karakolios, Zane Ma, Roberto Perdisci, Angelos Keromytis, and Manos Antonakakis. Practical attacks against DNS reputation systems. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4516–4534. IEEE, 2024.

[11] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. Some from here, some from there: Cross-project code reuse in GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 291–301. IEEE, 2017.

[12] GitHub. DMCA takedown notices. https://github.com/github/dmca, 2025.

[13] GitHub, Inc. GitHub acceptable use policies. https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies, 2025.

[14] Gibran Gomez, Kevin van Liebergen, and Juan Caballero. Cybercrime Bitcoin revenue estimations: Quantifying the impact of methodology and coverage. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3183–3197, 2023.

[15] Qingyuan Gong, Jiayun Zhang, Yang Chen, Qi Li, Yu Xiao, Xin Wang, and Pan Hui. Detecting malicious accounts in online developer communities using deep learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1251–1260, 2019.

[16] Danielle Gonzalez, Thomas Zimmermann, Patrice Godefroid, and Max Schäfer. Anomalicious: Automated detection of anomalous and potentially malicious commits on GitHub. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 258–267. IEEE, 2021.

[17] Bowen He, Yuan Chen, Zhuo Chen, Xiaohui Hu, Yufeng Hu, Lei Wu, Rui Chang, Haoyu Wang, and Yajin Zhou. Txphishscope: Towards detecting and understanding transaction-based phishing on Ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 120–134, 2023.

[18] Hao He, Haoqin Yang, Philipp Burckhardt, Alexandros Kapravelos, Bogdan Vasilescu, and Christian Kästner. Six million (suspected) fake stars on GitHub: A growing spiral of popularity contests, spam, and malware. In *Proceedings of the 48th International Conference on Software Engineering (ICSE'26)*, 2026.

[19] Risul Islam, Md Omar Faruk Rokon, Ahmad Darki, and Michalis Faloutsos. Hackerscope: The dynamics of a massive hacker online ecosystem. *Social Network Analysis and Mining*, 11(1):56, 2021.

[20] Jonathan Jiang and Mark Stamp. Multimodal techniques for malware classification. *arXiv preprint arXiv:2501.10956*, 2025.

[21] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Josh Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[22] Aklil Kiflay, Athanasios Tsokanos, Mahmood Fazlali, and Raimund Kirner. Network intrusion detection leveraging multimodal features. *Array*, 22:100349, 2024.

[23] Seunghyeon Lee, Changhoon Yoon, Heedo Kang, Yeonkeun Kim, Yongdae Kim, Dongsu Han, Sooel Son, and Seungwon Shin. Cybercriminal minds: an investigative study of cryptocurrency abuses in the dark web. In *Network and Distributed System Security (NDSS) Symposium 2019*. The Internet Society, 2019.

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

[25] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. Reading the tea leaves: A comparative analysis of threat intelligence. In *28th USENIX security symposium (USENIX Security 19)*, pages 851–867, 2019.

[26] Xigao Li, Anurag Yepuri, and Nick Nikiforakis. Double and nothing: Understanding and detecting cryptocurrency giveaway scams. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium 2023*, 2023.

[27] Antonio Lima, Luca Rossi, and Mirco Musolesi. Coding together at scale: GitHub as a collaborative social network. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 295–304, 2014.

[28] Cj Arsley Mateo, Jacob Santos, Darrel Tristan Virtusio, Junestherry Dela Cruz, and Paul John Bardon. AI-assisted fake GitHub repositories fuel SmartLoader and LummaStealer distribution, 2025.

[29] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–43. Springer, 2020.

[30] Yiyue Qian, Yiming Zhang, Nitesh Chawla, Yanfang Ye, and Chuxu Zhang. Malicious repositories detection with adversarial heterogeneous graph contrastive learning. In *Proceedings of the 31st ACM international conference on information & knowledge management*, pages 1645–1654, 2022.

[31] Yiyue Qian, Yiming Zhang, Yanfang Ye, and Chuxu Zhang. Adapting meta knowledge with heterogeneous information network for COVID-19 themed malicious repository detection. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[32] Md Omar Faruk Rokon, Risul Islam, Ahmad Darki, Evangelos E Papalexakis, and Michalis Faloutsos. SourceFinder: Finding malware source-code from publicly available repositories in GitHub. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 149–163, 2020.

[33] Md Omar Faruk Rokon, Pei Yan, Risul Islam, and Michalis Faloutsos. Repo2Vec: A comprehensive embedding approach for determining repository similarity. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 355–365. IEEE, 2021.

[34] Stewart Scott, Sara Ann Brackett, Trey Herr, and Maia Hamin. *Avoiding the success trap: Toward policy for open-source software as infrastructure*. Atlantic Council, 2023.

[35] Ax Sharma. Counterfeit ESLint and Node 'types' libraries downloaded thousands of times abuse Pastebin, December 2024.

[36] Varun Sharma. Harden-Runner detection: tj-actions/changed-files action is compromised. https://www.stepsecurity.io/blog/harden-runner-detection-tj-actions-changed-files-action-is-compromised, March 14 2025.

[37] Hossein Siadati et al. Devphish: Exploring social engineering in software supply chain attacks on developers, 2024. arXiv:2402.18401.

[38] Sonatype. State of the software supply chain 2024, October 2024.

[39] Derrick Stolee. Bring your monorepo down to size with sparse-checkout, January 2020. Updated May 16, 2022.

[40] Christophe Tafani-Dereeper and Sebastian Obregoso. The Shai-Hulud 2.0 npm worm: analysis, and what you need to know, November 2025. Last updated November 26, 2025.

[41] Nishat Ara Tania, Md Rayhanul Masud, Md Omar Faruk Rokon, Qian Zhang, and Michalis Faloutsos. Who is creating malware repositories on GitHub and why? In *Companion Proceedings of the ACM Web Conference 2024*, pages 955–958, 2024.

[42] Antonis Terefos. Stargazers ghost network. Technical report, Check Point Research, July 2024.

[43] Christoph Treude, Larissa Leite, and Maurício Aniche. Unusual events in GitHub repositories. *Journal of Systems and Software*, 142:237–247, 2018.

[44] Margaret Tucker, Rose Coogan, and Will Pace. Nuances and challenges of moderating a code collaboration platform. *Journal of Online Trust and Safety*, 2(4), 2024.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[46] Feng Wang and Huaping Liu. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2495–2504, 2021.

[47] Dominik Wermke, Jan H. Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl. "Always Contribute Back": A qualitative study on security challenges of the open source supply chain. In *Proceedings of the 44th IEEE Symposium on Security and Privacy*, May 2023.

[48] Teng Xu, Gerard Goossen, Huseyin Kerem Cevahir, Sara Khodeir, Yingyezhe Jin, Frank Li, Shawn Shan, Sagar Patel, David Freeman, and Paul Pearce. Deep entity classification: Abusive account detection for online social networks. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[49] Luming Yang, Lin Liu, Jun-Jie Huang, Zhuotao Liu, Shiyu Liang, Shaojing Fu, and Yongjun Wang. MM4flow: A pre-trained multi-modal model for versatile network traffic analysis. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25)*, New York, NY, USA, 2025. Association for Computing Machinery.

[50] Jingxiu Yao and Martin Shepperd. Assessing software defection prediction performance: why using the matthews correlation coefficient matters. In *International Conference on Evaluation and Assessment in Software Engineering*, page 120–129, 2020.

[51] Yiming Zhang, Yujie Fan, Shifu Hou, Yanfang Ye, Xusheng Xiao, Pan Li, Chuan Shi, Liang Zhao, and Shouhuai Xu. Cyber-guided deep neural network for malicious repository detection in GitHub. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 458–465. IEEE, 2020.

[52] Xiaoyan Zhou, Ying Zhang, Wenjia Niu, Jiqiang Liu, Haining Wang, and Qiang Li. An analysis of malicious packages in open-source software in the wild. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 371–385. IEEE, 2025.

[53] Huanhuan Zou, Zongwei Li, and Xiaoqi Li. Malicious code detection in smart contracts via opcode vectorization. *arXiv preprint arXiv:2504.12720*, 2025.

A Computing Loss Functions

Let (x_i, y_i) denote training samples where x_i is a repository's feature vector and y_i is its class label.

We first employ cross-entropy loss to directly supervise the classification task. Let $f(\cdot)$ denote a classifier that maps a repository feature vector x_i to a vector of logits $\ell_i \in \mathbb{R}^C$, where C is the number of classes and $\ell_{i,c}$ corresponds to class c. Applying the softmax function yields a predicted class distribution

$$p_{i,c} = \frac{\exp(\ell_{i,c})}{\sum_{c'=1}^{C} \exp(\ell_{i,c'})}.$$

The cross-entropy loss for sample (x_i, y_i) is then defined as

$$\mathcal{L}_{\text{CE}}(x_i, y_i) = -\log p_{i,y_i} = -\log\left(\frac{\exp(\ell_{i,y_i})}{\sum_{c'=1}^{C} \exp(\ell_{i,c'})}\right).$$

This objective encourages the model to assign high probability to the ground-truth class while penalizing incorrect predictions. The overall loss is computed by averaging $\mathcal{L}_{\text{CE}}$ over all training samples.

In addition to this classification objective, we incorporate a supervised contrastive loss [21] to structure the representation space. To compute contrastive loss, we first normalize each feature vector x_i to obtain z_i. For each sample belonging to class c, we compare it against all other samples from the same class (or, in set notation, $P \setminus \{z_i\}$, where P contains all samples of class c and $\setminus$ is the minus operator). All other samples act as implicit negative samples.

The contrastive loss for a sample z_i is defined as

$$\mathcal{L}_{z_i} = -\frac{1}{|P \setminus \{z_i\}|} \sum_{z_p \in P \setminus \{z_i\}} \log\left(\frac{\exp(z_i^\top z_p/\tau)}{\sum_{z_a \neq z_i} \exp(z_i^\top z_a/\tau)}\right),$$

where $\tau > 0$ is a temperature hyperparameter that controls the concentration of the similarity distribution.

The overall supervised contrastive loss is obtained by averaging over all samples in the positive class:

$$\mathcal{L}_{\text{contrastive}} = \frac{1}{|P|} \sum_{z_i \in P} \mathcal{L}_{z_i}.$$

B Example GitHub Repository

Figure B1 shows an example of a malicious GitHub repository from the unreported phishing campaign. A few distinctive characteristics from this campaign can be seen when viewing the repository in a web browser, such as the AI-generated README, the large set of keywords in the right column, and the time since the latest commit. The repository is consistently updated so that the timestamp of the latest commit remains within the last few hours.

C Statistical Features

Table C1 includes descriptions of the 40 statistical features used, indicating the 7 features also seen in prior work. All of the features are accessible through both the GitHub API and the publicly available event logs. We rely on the event logs to collect these features to support online identification of malicious repositories while limiting the burden on GitHub's API endpoints to minimal collection of README contents.

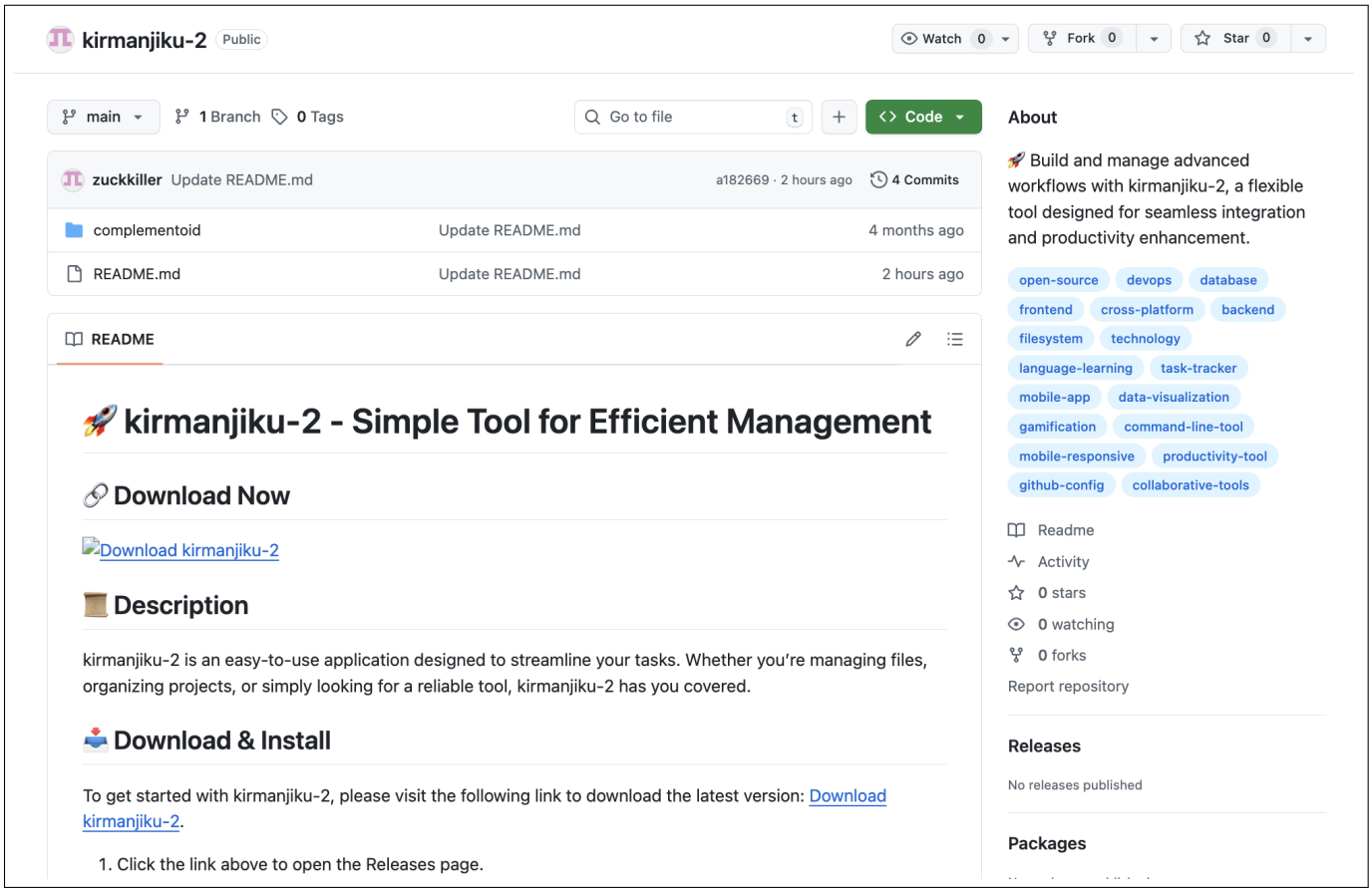

Figure B1: **Example GitHub repository.** Exhibits characteristics of a malicious scam campaign directing a user to a malicious binary. Shown tactics include AI-generated README, keyword stuffing, automatic commits.

Feature	Description	Event	API	Prior Work
Repository Features				
first_seen	Timestamp of the first event observed	✓	✓	
stars_received_total	Total stars received	✓	✓	
stars_received_1d	Stars received within 1 day of first event	✓	✓	
stars_received_3d	Stars received within 3 days of first event	✓	✓	
stars_received_14d	Stars received within 14 days of first event	✓	✓	
forks_received_total	Total forks received	✓	✓	
unique_contributors	Number of unique actors interacting with repository	✓	✓	
total_events	Total number of events	✓	✓	
unique_event_types	Count of distinct GitHub event types	✓	✓	
issues_opened	Issues opened	✓	✓	[43]
issues_closed	Issues closed	✓	✓	[43]
pull_requests_opened	Pull requests opened	✓	✓	[43]
pull_requests_merged	Pull requests merged	✓	✓	[43]
comments_received	Total issue and PR comments received	✓	✓	[43]
pushes_received	Push events received	✓	✓	
avg_time_between_events	Average time between consecutive events (seconds)	✓	✓	
days_active	Unique days with activity	✓	✓	
max_stars_received_per_hour	Maximum stars received in a single hour	✓	✓	
max_forks_received_per_hour	Maximum forks received in a single hour	✓	✓	
avg_issue_resolution_time	Average time to close issues (seconds)	✓	✓	[43]
is_fork	Whether repository is a fork	✓	✓	
User Features				
first_seen	Timestamp of first activity	✓	✓	
stars_made_total	Total stars given	✓	✓	
stars_made_1/3/14d	Stars given within 1/3/14 days of first event	✓	✓	
pushes_made_total	Pushes made	✓	✓	
stars_received_total	Stars received on repositories created by the actor	✓	✓	
unique_logins	Distinct actor logins used	✓	✓	
total_events	Total number of events	✓	✓	
unique_repos	Number of distinct repositories interacted with	✓	✓	
unique_event_types	Number of distinct event types	✓	✓	
issues_opened	Issues opened	✓	✓	
pull_requests_opened	Pull requests opened	✓	✓	[43]
comments_made	Comments on issues and PRs	✓	✓	
forks_made	Forks made	✓	✓	
repos_created	Repositories created	✓	✓	
avg_time_between_events	Average time between actions (seconds)	✓	✓	
days_active	Number of distinct active days	✓	✓	
longest_streak	Longest streak of consecutive active days	✓	✓	
max_stars_given_per_hour	Maximum stars given in a single hour	✓	✓	
max_stars_received_per_hour	Maximum stars received on owned repositories in an hour	✓	✓	

Table C1: **Repository/User features extracted from GitHub event logs.**