12-2025

AgentGuard: An active threat discovery system for package confusion using multi-agent collaboration

Wei MA
Singapore Management University, weima@smu.edu.sg

Yu LI

Zhi CHEN
Singapore Management University, zhi.chen.2023@phdcs.smu.edu.sg

Ye LIU

Lingxiao JIANG
Singapore Management University, lxjiang@smu.edu.sg

See next page for additional authors

AgentGuard: An Active Threat Discovery System for Package Confusion using Multi-Agent Collaboration

Anonymous

Abstract. The proliferation of open-source software (OSS) has made software supply chains prime targets for attacks like Package Confusion, where adversaries publish malicious packages with names deceptively similar to legitimate ones. Existing detection methods often rely on simple lexical similarity or passive analysis of known package pairs, struggle with high false positive rates (FPR), fail to proactively identify emerging threats, and are vulnerable to adversarial evasion. To overcome these limitations, we introduce AgentGuard, a novel framework for proactive, single-input package confusion detection. AgentGuard employs a multi-agent architecture that autonomously discovers potential confusion targets using fine-tuned word embedding model to hybird semantic search and subsequently evaluates the risk via a machine learning model incorporating multi-dimensional feature groups to enhance robustness. This design enables scalable, real-time monitoring across diverse software ecosystems. We evaluate AgentGuard on the challenging ConfuDB and NeupaneDB datasets. Our results demonstrate that AgentGuard significantly outperforms state-of-the-art baselines, improving accuracy by 10%-24% while simultaneously reducing the false positive rate by 9%-31%.

Keywords: package confusion detection · LLM agent · Cybersecurity

1 Introduction

Modern software development relies heavily on open-source package ecosystems, with registries like NPM and PyPI hosting millions of packages and serving billions of weekly downloads [11, 12, 15, 17, 19, 23]. This scale and openness create significant attack surfaces for software supply chain threats [5, 16]. Among these, package confusion attacks pose a particularly insidious risk: adversaries publish malicious packages with names designed to closely resemble legitimate ones, deceiving developers into installing them and achieving arbitrary code execution. These attacks exploit diverse confusion mechanisms spanning lexical (typosquatting), syntactic (delimiter modifications, reordering), and semantic (synonym substitution) levels [7, 10].

Prior work on package confusion detection has evolved from simple lexical matching [20,21] to semantic analysis [10] and metadata-based filtering [6,13,25]. However, existing approaches face three key limitations: high false positive rates

due to benign naming similarities, susceptibility to adversarial metadata manipulation, and reliance on pre-defined package pairs that hinder proactive detection of newly published threats. First, *high false-positive rates* arise because the inherent naming similarity among benign packages makes distinguishing innocuous resemblance from deliberate impersonation difficult, leading to excessive alerts and alert fatigue [2]. Second, *susceptibility to adversarial evasion* remains problematic as defenses relying on static rules or easily manipulable metadata features remain fragile against attackers who can forge or obfuscate these signals. Third, *retrospective and pair-dependent analysis* limits prior work, as most existing approaches operate reactively on known package pairs, misaligning with the *proactive* need to assess newly published packages of unknown intent against the entire ecosystem.

To address these challenges, we introduce AgentGuard, a proactive multi-agent framework for package confusion detection. Given only a single package name, AgentGuard autonomously discovers potential legitimate targets using similarity search and evaluates confusion risk using a machine learning model fortified with multi-dimensional features. The main contributions are:

1. A robust detection model incorporating multi-dimensional feature groups, achieving significantly improved resilience compared to easily-forgeable feature-based approaches.
2. A proactive, single-input detection framework that autonomously discovers confusion targets across diverse package ecosystems without requiring pre-defined pairs.
3. Empirical evaluation on NeupaneDB and ConfuDB showing AgentGuard outperforms state-of-the-art baselines by 10%-24% in accuracy and 9%-31% in false positive reduction. The source code of AgentGuard is publicly available at: `https://sites.google.com/view/agentguard/home`.

Roadmap. The rest of this paper is organized as follows. We first introduce the relevant background and related work in Section 2, followed by the design of AgentGuard in Section 3. We then present our experimental setup and the results in Section 4. After that, we discuss the limitations in Section 5. Finally, Section 6 concludes this paper.

2 Background and Related Work

Package confusion attacks exploit naming similarities to deceive developers into installing malicious packages that impersonate legitimate ones. These attacks employ diverse confusion techniques [7, 10]: lexical typosquatting, syntactic delimiter manipulation, and semantic synonym substitution. Once installed, malicious packages can execute arbitrary code, steal credentials, or inject backdoors into the software supply chain.

Name-Based Detection. Early approaches focused on detecting lexical similarities using edit distance metrics. Taylor et al. [20] and Vu et al. [21] applied Levenshtein distance to identify typosquatting. Neupane et al. [10] extended this by using FastText embeddings [3] to capture semantic similarities, categorizing 13 distinct confusion mechanisms. While effective for identifying name-level resemblance, these methods suffer from high false positives when legitimate packages share similar names, and struggle to distinguish malicious intent from benign similarity.

Metadata-Based Detection. To reduce false positives, subsequent work incorporated package metadata as additional signals. Zimmermann et al. [25] correlated maintainer count with security risk. Ohm et al. [13] employed machine learning with package metadata and dependency information. Sejfia et al. [18] verified code reproducibility as a trust signal. ConfuGuard [6] combined deep metadata analysis with heuristic rules in a "Benignity Filter", achieving significant false positive reduction. However, these approaches either require pre-identified candidate pairs [6] or rely on metadata that sophisticated adversaries can forge [13,25], limiting their effectiveness against adaptive attackers.

Adversarial Robustness. MeMPtec [4] introduced the Easy-to-Manipulate(ETM)/ Difficult-to-Manipulate(DTM) framework, distinguishing between ETM (e.g., descriptions, homepages) and DTM features (e.g., package age, release history). This framework provides a principled approach to building evasion-resistant detectors. However, prior work has not systematically integrated DTM features into a comprehensive detection pipeline that addresses both target discovery and classification, nor empirically quantified robustness under adversarial manipulation.

3 Design of AgentGuard

We begin by formalizing the problem and establishing our threat model, then articulate our design goals, followed by a system overview, and finally detail each agent's design.

3.1 Problem Formulation, Threat Model and Design Goals

Problem Statement. Given a newly published package p with name n_p in ecosystem $\mathcal{E}$, our goal is to determine whether p is a confusion attack impersonating a legitimate package. Unlike prior pair-based approaches that assume knowledge of both the suspicious package and its target, we address the *single-input detection problem*: given only p, the system must autonomously (1) discover potential legitimate targets $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$ that p may impersonate, and (2) classify whether p constitutes a malicious confusion attack.

Threat Model. We consider an adversary aiming to deceive developers into installing malicious packages by exploiting naming similarities with legitimate ones. The adversary's primary weapon is the package name itself, which can be crafted using diverse confusion techniques spanning Lexical (typosquatting), Syntactic (delimiter changes, reordering), and Semantic (synonym substitution).

To maximize the deception's effectiveness and evade automated detectors, this primary name-based attack is often accompanied by adversarial metadata manipulation. The adversary may attempt to forge signals related to what our model defines as Metadata Quality (MQ) featuresy (e.g., repository url).

However, we assume the adversary cannot trivially forge the signals central to our model's robustness, particularly those we categorize as DTM features. These include immutable temporal signals like package age and release history. We assume package registries maintain basic integrity (e.g., immutable publication time) and are not fully compromised.

Design Goals. To address the challenges identified in Section 1 and operate effectively under our threat model, AgentGuard is designed to satisfy the following objectives:

- G1 **Proactive Single-Input Detection.** Operate with only a package name as input, autonomously discovering confusion targets without pre-defined pairs, enabling real-time monitoring of newly published packages.
- G2 **Robust Multi-Dimensional Model.** Effectively distinguish malicious confusion from benign similarities while resisting adversarial metadata forgery through four-dimensional feature groups, achieving high precision and recall with resilience against evasion.
- G3 **Multi-Ecosystem Generalization.** Generalize across diverse package ecosystems (NPM, PyPI, RubyGems, etc.) without ecosystem-specific hardcoding.
- G4 **Practical Deployment Readiness.** Provide interpretable results with actionable context for human decision-making, while supporting concurrent analysis with reasonable latency for large-scale monitoring.

3.2 System Overview

AgentGuard is a proactive multi-agent framework that integrates LLM-based reasoning, lightweight machine learning, and tool-augmented analysis for automated package confusion detection. As illustrated in Figure 1, AgentGuard comprises three agents: the *Orchestrator Agent* coordinates task scheduling and data flow; the *Threat Analyst Agent* performs semantic search to discover potential legitimate targets (**G1**); the *Confusion Checker Agent* conducts ML-based risk classification using multi-dimensional features (**G2**); we use multiple ecosystems as our datasets (**G3**); and the real-world *Human Analyst* receives aggregated final reports for final decision-making (**G4**).

Why Multi-Agent Architecture? The multi-agent design is necessitated by three key characteristics: *(1) Task Heterogeneity*: Target discovery (hybrid similarity

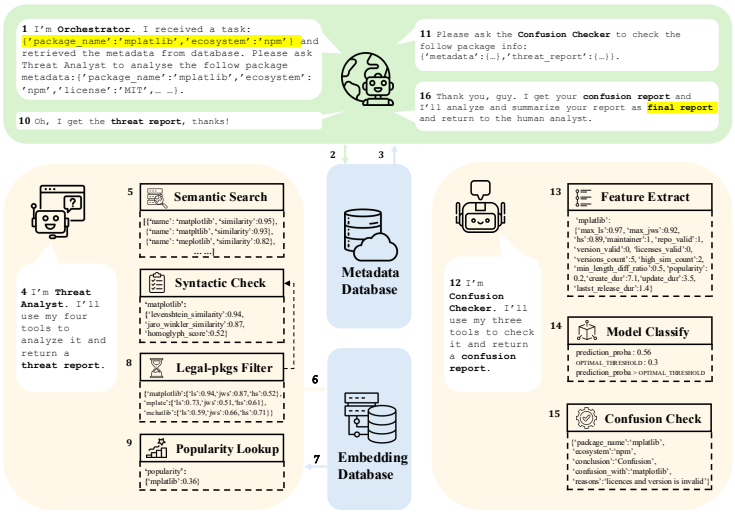


Fig. 1: The overall architecture and workflow of the AgentGuard system. The Orchestrator Agent coordinates three specialized agents through a discovery-evaluation pipeline, processing packages from detection request to final threat assessment.

search) and classification prediction require fundamentally different computational paradigms, which independent agents handle optimally without architectural compromises. *(2) Independent Evolution*: The agent-based architecture enables updating individual components (e.g., retraining word embedding model for new ecosystems or incorporating new confusion patterns) without modifying downstream agents, reducing deployment risk (**G2**). *(3) Graceful Degradation*: When external services fail (e.g., `Libraries.io` [1] API unavailable), agents can fall back to cached data and continue operation, ensuring continuous availability for large-scale monitoring (**G4**). This design maintains independent state, explicit failure handling, and asynchronous communication, distinguishing it from simple modular pipelines.

Overview. The Orchestrator receives a target package and retrieves its metadata from the metadata database (steps 1–3), then assigns it to the Threat Analyst for rapid analysis (steps 4). The Threat Analyst queries an embedding database to perform hybrid semantic–syntactic retrieval of the K most similar popular packages (steps 5–7), computes Levenshtein, Jaro–Winkler, and homoglyph similarities, applies legitimate-package filtering and popularity lookup (steps 8–9), and returns a structured threat report (step 10). The Orchestrator then forwards the results to the Confusion Checker for final confusion prediction and report generation (steps 11–16). At steps 1, 4, 10, 11, 12, and 16, AgentGuard interacts with the LLM, and the corresponding prompts are provided in our public code.

In summary, AgentGuard leverages the strengths of LLMs for adaptive reasoning, small embedding model, and machine learning models for reliable classification, forming a cooperative architecture that achieves both analytical depth and operational robustness.

3.3 Orchestrator Agent

The Orchestrator Agent serves as the central coordinator of AgentGuard, managing task initialization, inter-agent communication, and result aggregation (**G4**). Powered by the LLM for adaptive decision-making, it bridges high-level reasoning with tool-assisted data operations. Given an input package p with name n_p and ecosystem $\mathcal{E}$, it executes the following workflow:

Step 1: *Resilient Data Acquisition.* The Orchestrator first retrieves package metadata from the local database. If local data is unavailable or incomplete, it automatically falls back to the API service provided by `Libraries.io`, ensuring robustness in data retrieval.

Step 2: *Task Dispatch and Conditional Scheduling.* The Orchestrator constructs a detection request $\langle n_p, \mathcal{E} \rangle$ and dispatches it to the Threat Analyst Agent for target discovery. Upon receiving the threat report (containing identified targets $\mathcal{T}$), it validates metadata completeness. If sufficient, it forwards the enriched task (including p, $\mathcal{T}$, and all metadata) to the Confusion Checker Agent for classification; otherwise, it flags the case for manual review.

Step 3: *Result Aggregation and Reporting.* The Orchestrator consolidates the threat report and confusion report (if available) into a comprehensive assessment, annotating the confidence level and forwarding it to the human analyst for final decision-making.

3.4 Threat Analyst Agent

The Threat Analyst Agent addresses autonomous target discovery (**G1**), transforming single-input detection into a comparison problem by identifying potential legitimate targets $\mathcal{T}$ that a suspicious package may impersonate. Guided by the LLM, it coordinates semantic embedding queries and syntactic similarity analyses, balancing semantic breadth (capturing synonym-based confusion) with syntactic precision (detecting typosquatting).

Step 1: *Semantic Embedding and Hybrid Search.* The agent encodes the input package name n_p using a fine-tuned word embedding model, whose subword-based mechanism captures morphological variations critical for handling diverse naming patterns across 34 ecosystems. The agent queries a pre-built embedding database [14] using a hybrid strategy combining *Cosine Similarity* (semantic relatedness between embeddings) and *Trigram Similarity* (syntactic overlap of character trigrams). After retrieving, merging, and deduplicating candidates ranked by both metrics, this yields up to 30 packages balancing semantic breadth with lexical precision.

Step 2: *Legitimate-Package Filtering and Ranking.* Since attackers typically mimic popular packages to maximize impact, the agent filters candidates by

download counts and repository verification. For each retained candidate c_i, it computes three syntactic similarity scores: Levenshtein distance [8], Jaro–Winkler similarity [22], and Homoglyph score [24]. Candidates are ranked by max{Lev, Jaro, Homo}, and the top three form the final target set $\mathcal{T} = \{t_1, t_2, t_3\}$.

Step 3: *Popularity Assessment and Report Generation.* For each target $t_i \in \mathcal{T}$, the agent computes a normalized popularity score (0 to 1) by aggregating download counts, stars, forks, and dependencies. It generates a *threat report* containing the target set $\mathcal{T}$, similarity scores, and popularity scores, which is returned to the Orchestrator for further classification. This LLM-guided and tool-augmented process enables interpretable and high-precision target discovery under minimal prior knowledge.

3.5 Confusion Checker Agent

The Confusion Checker Agent performs the final stage of analysis (**G2**), integrating metadata and similarity results to assess the likelihood of a confusion attack. This agent complements the LLM-driven reasoning of upstream components with a lightweight machine learning classifier to ensure reliability and interpretability.

Step 1: *Multi-Dimensional Feature Extraction.* Given the input package p, the target set $\mathcal{T}$, and metadata for both, the agent extracts a 14-dimensional feature vector $X \in \mathbb{R}^{14}$ organized into four categories (Table 1). The features span *Syntactic Similarity (SS)* (name-level resemblance), *Metadata Quality (MQ)* (maintenance and validity signals), *Contextual and Differential (CD)* (comparative ecosystem context), and critically, *Difficult-to-Manipulate (DTM)* [4] (temporal signals like package age that resist forgery), thereby enhancing robustness (**G3**).

Step 2: *Random Forest Classification.* The feature vector $X \in \mathbb{R}^{14}$ is input into a pre-trained Random Forest (RF) classifier with decision trees (tuned via cross-validation). The RF aggregates individual tree predictions via majority voting, outputting a confusion probability $\hat{P}(p \text{ is confusion} \mid X) \in [0, 1]$ that provides both a classification decision and confidence estimate for human review prioritization.

Step 3: *Threshold-Based Decision and Report Generation.* The agent compares $\hat{P}$ with an empirically optimized threshold (0.29, determined via F1-maximization in Section 4). If $\hat{P} > 0.29$, the package is classified as *Confusion*; otherwise, as *Benign*. The agent generates a *confusion report* with the classification decision, confidence score, and low-confidence flag for manual review (**G4**), which is returned to the Orchestrator.

Through this modular multi-agent design, AgentGuard achieves a balance between scalability, analytical precision, and robustness against adversarial behavior. The empirical validation is presented in Section 4.

Table 1: Extracted features and their corresponding categories.

Feature Name	**Feature Category**
Maximum Levenshtein Similarity	Syntactic Similarity (SS)
Maximum Jaro–Winkler Similarity	
Maximum Homoglyph Score	
Maintainer Adequacy	Metadata Quality (MQ)
Repository URL Validity	
Version Format Validity	
License Validity	
Version Count	
High-Similarity Count	Contextual and Differential (CD)
Minimum Length Difference Ratio	
Target Popularity	
Package Age	Difficult-to-Manipulate (DTM)
Time Since Last Release	
Time Since Last Update	

4 EXPERIMENTS

4.1 Experimental Setups

Baselines. We compared AgentGuard against two established baselines: *Typomind* [10], which utilizes semantic embeddings alongside lexical analysis to detect confusion primarily on package pairs, and *ConfuGuard* [6] that employs deep metadata analysis.

Datasets. Two datasets were used for evaluation. *ConfuDB* [6] contains 2,361 packages analyzed by security experts during tool development and deployment, representing complex real-world cases with confirmed attacks, stealthy threats, and benign samples. *NeupaneDB* [10] comprises 1,840 packages from prior research, featuring a higher proportion of confirmed and publicly documented confusion attacks.

Fine-tuned word embedding model. While advanced Transformer embedding models (like Qwen3) excel at general semantic tasks, we selected FastText(cc.en.300.bin) because its architecture, which is based on subword (n-gram) information, makes it inherently adept at generating robust vector representations for misspellings and Out-of-Vocabulary words. To further enhance its domain adaptation, we fine-tuned the cc.en.300.bin model using a comprehensive corpus of 10,015,794 package names spanning 34 distinct ecosystems.

Classification Model. AgentGuard's Confusion Checker Agent utilizes a Random Forest classifier. We employed 5-fold stratified cross-validation on subsets from ConfuDB and NeupaneDB to robustly evaluate performance using the 14 defined features and select optimal settings. A single, final model was then trained using these settings on the entire combined dataset (all 5 folds) for deployment.

4.2 Research Questions

To comprehensively evaluate AgentGuard, we designed experiments addressing three research questions:

- **RQ1 (Model Capability):** How does AgentGuard select its classification threshold, and what is its overall classification capability?
- **RQ2 (Baseline Comparison):** How does AgentGuard compare to state-of-the-art tools (Typomind, ConfuGuard) in detecting real-world confusion attacks?
- **RQ3 (Feature Robustness):** What is the contribution of each feature category (SS, MQ, CD, DTM), and how robust is the model against adversarial metadata manipulation?

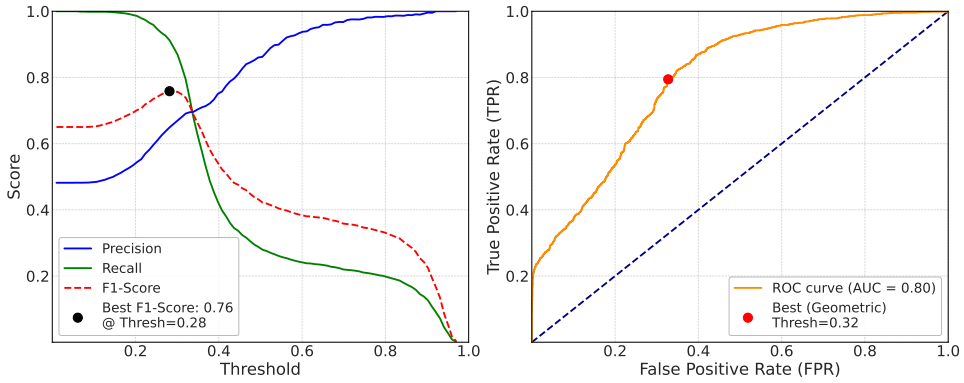
4.3 RQ1 - Threshold Selection and Overall Capability

Methodology. Rather than using an arbitrary threshold (e.g., 0.5), we employ a principled approach by conducting a grid search over the range [0.01, 0.99] with step size 0.01 on a validation set, selecting the threshold that maximizes the F1 score (a harmonic mean that balances precision and recall). This data-driven strategy ensures the threshold is optimized for the specific characteristics of package confusion detection.
Results. As shown in Fig. 2a, the F1 score peaks at threshold = 0.29, significantly lower than the conventional 0.5. This deviation reflects an important characteristic of confusion detection: the severe consequences of false negatives (missing malicious packages) often outweigh those of false positives (flagging benign packages for human review). The threshold of 0.29 favors recall while maintaining acceptable precision, aligning with the practical requirement of minimizing security risks in supply chain monitoring (**G2**).

The ROC curve in Fig. 2b demonstrates robust classification capability with AUC ≈ 0.8. This indicates that our model assigns higher confidence scores to malicious confusion packages than to benign ones in approximately 80% of all possible pairwise comparisons, which represents strong discriminative performance considering the subtle differences between sophisticated confusion attacks and legitimate similarly-named packages.
Takeaway. The optimized threshold of 0.29 and strong AUC validate that AgentGuard's hybrid feature-based classifier possesses solid foundational classification capability. The threshold's deviation from 0.5 appropriately reflects the asymmetric costs in security-critical applications.

4.4 RQ2 - Performance Comparison with Baselines

Overall Performance. We evaluated AgentGuard against two state-of-the-art baselines (Typomind and ConfuGuard) on both ConfuDB (2,361 packages,



(a) Relationship of precision, recall and F1 with threshold

(b) ROC curve showing FPR vs TPR trade-off

Fig. 2: Performance evaluation: (a) Threshold selection analysis showing F1 score optimization at 0.29, and (b) ROC curve demonstrating model classification capability with AUC ≈ 0.8

analyst-triaged real-world cases) and NeupaneDB (1,841 packages, publicly documented attacks). Table 2 presents the comprehensive results. AgentGuard consistently outperforms both baselines across all metrics on both datasets, achieving improvements of 6%-43% in precision, 10%-24% in recall/accuracy, and 16%-42% in F1-score.

Dataset-Specific Analysis. On *ConfuDB*, the most challenging real-world dataset, AgentGuard achieves weighted F1 of 0.78 versus 0.46 (ConfuGuard) and 0.36 (Typomind), representing a 70% and 117% relative improvement, respectively. Notably, AgentGuard's recall for the *Confusion* class (0.76) substantially exceeds both baselines (0.53 and 0.31), demonstrating superior capability in identifying actual attacks (**G2**). This is critical in security contexts where false negatives carry severe consequences.

On *NeupaneDB*, which contains more clear-cut, publicly documented attacks, all methods perform better, but AgentGuard maintains its advantage with weighted F1 of 0.92 versus 0.76 (ConfuGuard) and 0.69 (Typomind). The smaller performance gap on this dataset suggests that AgentGuard's advantages are most pronounced when handling subtle, analyst-triaged real-world cases, precisely the challenging scenarios where automated detection provides the most value.

Understanding the Performance Gains. AgentGuard's superior performance stems from three key design choices that address fundamental limitations of prior work:

(1) Proactive Target Discovery. Unlike baselines that require pre-identified package pairs, AgentGuard's Threat Analyst Agent autonomously discovers confusion targets using fine-tuned word embedding model to semantic search (**G1**). This enables detection of sophisticated semantic confusion attacks that purely

Table 2: Experiments result for AgentGuard with Typomind and ConfuGuard on the ConfuDB and NeupaneDB datasets

Class	Metric	ConfuDB			NeupaneDB		
		AgentGuard	ConfuGuard	Typomind	**AgentGuard**	ConfuGuard	Typomind
Benign	Precision	0.93	0.40	0.31	0.89	0.91	0.38
	Recall	0.76	0.51	0.69	0.93	0.62	0.84
	F1-score	0.84	0.45	0.30	0.91	0.67	0.81
Confusion	Precision	0.53	0.55	0.31	0.87	0.79	0.74
	Recall	0.76	0.53	0.31	0.85	0.82	0.68
	F1-score	0.62	0.49	0.52	0.93	0.79	0.65
Weighted	Precision	**0.82**	0.44	0.31	**0.88**	0.82	0.65
	Recall	**0.76**	0.52	0.61	**0.87**	0.77	0.72
	F1-score	**0.78**	0.46	0.36	**0.92**	0.76	0.69
	Accuracy	**0.76**	0.52	0.61	**0.87**	0.77	0.72
Total Support		2361	2361	2361	1840	1840	1840

lexical approaches (like aspects of Typomind) may miss, while avoiding the coverage limitations of pair-based methods.

(2) Multi-Dimensional Feature Engineering. Rather than relying primarily on name similarity or simple heuristics, AgentGuard employs 14 features spanning four categories (SS, MQ, CD, DTM, as shown in Table 1). This holistic feature space captures signals beyond superficial naming patterns, including metadata quality, contextual relationships, and temporal characteristics, enabling more nuanced discrimination between malicious packages and benign similarly-named ones.

(3) ML-Based Classification. The Random Forest classifier adaptively learns the complex, non-linear relationships between features and maliciousness from labeled data, providing more flexible decision boundaries than rule-based systems. This explains AgentGuard's higher precision compared to systems relying on fixed heuristic thresholds.

Takeaway. The consistent and substantial performance improvements across both datasets validate AgentGuard's core hypothesis: combining proactive semantic discovery with multi-dimensional ML-based evaluation effectively addresses the limitations of prior confusion detection approaches. The particularly strong performance on ConfuDB's challenging real-world cases demonstrates practical deployment readiness.

4.5 RQ3 - Feature Contribution and Robustness

To understand which feature categories drive AgentGuard's performance and assess its resilience against adversarial manipulation, we conducted controlled ablation experiments comparing four model configurations: *Baseline* (SS only), *Vulnerable* (SS + MQ), *Robust* (SS + CD + DTM, excluding MQ), and *AgentGuard* (all features). Each model was evaluated on two test conditions:
Clean: Original, unmodified test data from ConfuDB and NeupaneDB.

Adversarial (Adv.): Simulated attack where all MQ feature values in *Confusion* samples are flipped (0→1, 1→0), mimicking an adversary's attempt to forge legitimate-appearing metadata, while *Benign* samples remain unchanged.

Table 3: The ablation experiment results using four model configurations

Model	**Dataset**	**Clean**			**Adv.**	ΔRecall$_{(c.)}$
		AUC	F1	Recall$_{(c.)}$	Recall$_{(c.)}$	
Baseline	ConfuDB	0.54	0.65	0.28	0.28	0.00
	NeupaneDB	0.78	0.73	0.73	0.73	0.00
Vulnerable	ConfuDB	0.77	0.76	0.41	0.08	0.33
	NeupaneDB	0.95	0.90	0.88	0.30	0.58
Robust	ConfuDB	0.75	0.74	0.36	0.36	0.00
	NeupaneDB	0.94	0.90	0.88	0.88	0.00
AgentGuard	ConfuDB	**0.81**	**0.78**	**0.68**	0.22	0.46
	NeupaneDB	**0.96**	**0.92**	**0.89**	0.74	0.15

The key metric is the recall drop, $\Delta\text{Recall} = Recall_{\text{Clean}} - Recall_{\text{Adv.}}$ for the *Confusion* class, quantifying performance degradation under attack. Table 3 presents the complete results.

Feature Contribution (Clean Data). On unmodified data, each feature group demonstrates clear value. The Baseline (SS only) achieves modest performance (F1: 0.65 on ConfuDB, 0.73 on NeupaneDB), confirming that syntactic similarity alone is insufficient (**G2**). Adding MQ features (Vulnerable) substantially improves performance (F1: 0.76 and 0.90), indicating that metadata quality signals are highly informative for identifying poorly-maintained malicious packages. The Robust configuration (SS + CD + DTM) also achieves strong performance (F1: 0.74 and 0.90) through different signals, specifically contextual relationships and temporal patterns. Finally, AgentGuard with all features achieves the best performance (F1: 0.78 and 0.92, AUC: 0.81 and 0.96), demonstrating that combining complementary feature types captures the most comprehensive view of package maliciousness.

Robustness Against Adversarial Manipulation. The adversarial evaluation reveals stark differences in resilience. The Vulnerable model, relying on SS and easily-forged MQ features, suffers catastrophic performance collapse under attack, with Recall Drop of 0.33 (ConfuDB) and 0.58 (NeupaneDB). This represents up to 66% of its detection capability lost, rendering it vulnerable when adversaries manipulate metadata.

In sharp contrast, the Robust model exhibits *perfect resilience*: $\Delta Recall = 0.00$ on both datasets. Since it excludes MQ features entirely, flipping MQ values has no effect whatsoever. This validates the fundamental premise behind DTM features [4]: temporal characteristics like package age and release history

are inherently difficult for adversaries to manipulate without waiting extended periods, providing attack-resistant signals (**G2**).

The complete AgentGuard model strikes a crucial balance. While it includes MQ features (and thus is affected by their manipulation), the presence of robust DTM and CD features provides substantial protection. Its $\Delta Recall$ (0.15-0.46) is dramatically lower than the vulnerable model's (0.33-0.58), demonstrating that DTM features act as a *defensive hedge*. Even when some features are compromised, the model retains significant detection capability through manipulation-resistant signals.

Takeaway. All feature categories contribute to detection performance, but they differ fundamentally in robustness. MQ features provide strong signals on clean data but are trivially manipulable. DTM features, while individually less powerful, are inherently attack-resistant and essential for maintaining performance against adversarial evasion (**G2**). AgentGuard's multi-dimensional feature design achieves both high baseline accuracy and substantial resilience, a necessary combination for practical security applications.

5 Discussion

Defense Against Adaptive Adversaries. Our RQ3 results reveal a critical deployment insight: adversaries will likely attempt metadata manipulation as a low-cost evasion strategy. Systems relying primarily on ETM features become vulnerable to trivial circumvention (up to 66% detection capability lost). In contrast, AgentGuard's incorporation of DTM features provides crucial defensive advantage, retaining substantial detection capability (Recall Drop of 15%-46% vs. 33%-58%) through manipulation-resistant temporal signals. This *defensive-in-depth* strategy is essential for production environments facing adaptive adversaries.

Cross-Ecosystem Generalization. AgentGuard's design explicitly targets multi-ecosystem applicability (**G3**). Our evaluation validates this: both datasets contain packages from diverse ecosystems, yet our experiments make *no ecosystem-specific adaptations*. This generalization stems from (1) FastText fine-tuning on 10M package names across 34 ecosystems, and (2) universal feature signals that transcend individual package managers. Current results demonstrate practical multi-ecosystem deployment feasibility.

Limitations and Future Work. AgentGuard has several limitations that suggest future research directions. The semantic search (**G1**) may struggle with very short names, acronyms, or domain-specific terms, suggesting exploration of hybrid approaches with character-level models. While DTM features resist basic metadata forgery (**G2**), sophisticated adversaries might employ long-term strategies such as account nurturing or Sybil attacks [9], requiring complementary approaches like behavioral anomaly detection. Additionally, continuous model retraining and feedback loops are essential for adapting to evolving attacks, and full-scale deployment requires further optimization (**G4**), including incremental index updates, caching strategies, and distributed processing.

6 Conclusion

This paper proposed AgentGuard, an innovative multi-agent system that fundamentally transforms package confusion detection from traditional passive, pair-based analysis to an active, single-input detection model. By integrating a word embedding model for autonomous target discovery with a machine learning classifier that fuses a multi-dimensional feature groups, AgentGuard achieves superior performance: evaluation on the ConfuDB and NeupaneDB datasets shows its accuracy is 10%-24% higher than state-of-the-art baselines, while reducing the false positive rate by 9%-31%, and it also significantly enhances robustness against adversarial metadata forgery. Agent Guard provides a practical, scalable, and adversarially robust automated defense tool, which is crucial for protecting the modern software supply chain from complex and evasive threats.

References

1. libraries.io. https://libraries.io/api (2025)
2. Agency for Healthcare Research and Quality: Alert fatigue. https://psnet.ahrq.gov/primer/alert-fatigue (2019)
3. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information (2017), https://arxiv.org/abs/1607.04606
4. Halder, S., Bewong, M., Mahboubi, A., Jiang, Y., Islam, M.R., Islam, M.Z., Ip, R.H., Ahmed, M.E., Ramachandran, G.S., Ali Babar, M.: Malicious package detection using metadata information. In: Proceedings of the ACM Web Conference 2024. p. 1779–1789. WWW '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3589334.3645543, https://doi.org/10.1145/3589334.3645543
5. Herr, T.: Breaking trust – shades of crisis across an insecure software supply chain. USENIX Association (Feb 2021)
6. Jiang, W., Çakar, B., Lysenko, M., Davis, J.C.: ConfuGuard: Using Metadata to Detect Active and Stealthy Package Confusion Attacks Accurately and at Scale. arXiv e-prints arXiv:2502.20528 (Feb 2025). https://doi.org/10.48550/arXiv.2502.20528
7. Kaplan, B., Qian, J.: A survey on common threats in npm and pypi registries (2021), https://arxiv.org/abs/2108.09576
8. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics. Doklady **10**, 707–710 (1965), https://api.semanticscholar.org/CorpusID:60827152
9. Lynn Neary: Real 'Sybil' Admits Multiple Personalities Were Fake. NPR. (2011)
10. Neupane, S., Holmes, G., Wyss, E., Davidson, D., De Carli, L.: Beyond typosquatting: an in-depth look at package confusion. In: Proceedings of the 32nd USENIX Conference on Security Symposium. SEC '23, USENIX Association, USA (2023)
11. npm: . https://www.npmjs.com/ (2024)
12. NPM Contributors: Npm package json: name. https://docs.npmjs.com/cli/v9/configuring-npm/package-json#name. (2024)
13. Ohm, M., Stuke, C.: Sok: Practical detection of software supply chain attacks. In: Proceedings of the 18th International Conference on Availability, Reliability and Security. ARES '23, Association for Computing Machinery, New York, NY,

USA (2023). https://doi.org/10.1145/3600160.3600162, https://doi.org/10.1145/3600160.3600162
14. pgvector Contributors: pgvector: A vector extension for postgresql. https://github.com/pgvector/pgvector (2023)
15. Python Software Foundation: Acceptable use policy. https://policies.python.org/pypi.org/Acceptable-Use-Policy/ (2024)
16. Scalco, S., Paramitha, R., Vu, D.L., Massacci, F.: On the feasibility of detecting injections in malicious npm packages. In: Proceedings of the 17th International Conference on Availability, Reliability and Security. ARES '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3538969.3543815, https://doi.org/10.1145/3538969.3543815
17. Schorlemmer, T.R., Kalu, K.G., Chigges, L., Ko, K.M., Isghair, E.A., Baghi, S., Torres-Arias, S., Davis, J.C.: Signing in four public software package registries: Quantity, quality, and influencing factors (2024), https://arxiv.org/abs/2401.14635
18. Sejfia, A., Schäfer, M.: Practical automated detection of malicious npm packages. In: Proceedings of the 44th International Conference on Software Engineering. p. 1681–1692. ACM (May 2022). https://doi.org/10.1145/3510003.3510104, http://dx.doi.org/10.1145/3510003.3510104
19. Soto-Valero, C., Benelallam, A., Harrand, N., Barais, O., Baudry, B.: The emergence of software diversity in maven central. In: Proceedings of the 16th International Conference on Mining Software Repositories. p. 333–343. MSR '19, IEEE Press (2019). https://doi.org/10.1109/MSR.2019.00059, https://doi.org/10.1109/MSR.2019.00059
20. Taylor, M., Vaidya, R., Davidson, D., De Carli, L., Rastogi, V.: Defending against package typosquatting. In: Network and System Security: 14th International Conference, NSS 2020, Melbourne, VIC, Australia, November 25–27, 2020, Proceedings. p. 112–131. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-65745-1_7, https://doi.org/10.1007/978-3-030-65745-1_7
21. Vu, D.L., Pashchenko, I., Massacci, F., Plate, H., Sabetta, A.: Typosquatting and combosquatting attacks on the python ecosystem. 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) pp. 509–514 (2020), https://api.semanticscholar.org/CorpusID:220792403
22. Winkler, W.E.: String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. (1990), https://api.semanticscholar.org/CorpusID:54580585
23. Wyss, E., De Carli, L., Davidson, D.: What the fork? finding hidden code clones in npm. In: Proceedings of the 44th International Conference on Software Engineering. p. 2415–2426. ICSE '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3510003.3510168, https://doi.org/10.1145/3510003.3510168
24. Yazdani, R., van der Toorn, O., Sperotto, A.: A case of identity: Detection of suspicious idn homograph domains using active dns measurements. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 559–564 (2020). https://doi.org/10.1109/EuroSPW51379.2020.00082
25. Zimmermann, M., Staicu, C.A., Tenny, C., Pradel, M.: Smallworld with high risks: a study of security threats in the npm ecosystem. In: Proceedings of the 28th USENIX Conference on Security Symposium. p. 995–1010. SEC'19, USENIX Association, USA (2019)