Automatically Generating Security Guidelines from Pull Request Discussions: The Case of Django Framework

Ester Silva
Department of Computer Science, Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil
estersilva@dcc.ufmg.br

Marco Tulio Valente
Department of Computer Science, Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil
mtov@dcc.ufmg.br

Hudson Borges
Faculty of Computing, Federal University of Mato Grosso do Sul (UFMS)
Campo Grande, Brazil
hudson.borges@ufms.br

Abstract

Security in open-source software relies on documentation, yet a major gap exists regarding project-specific instructions for secure usage. We introduce a multi-stage framework that automatically generates project-specific User Guidelines from historical Pull Request (PR) code reviews. Adapting PRIMES (Prompt Refinement and Insights for Mining Empirical Software repositories) [5] — a methodology for using LLMs (Large Language Models) in repository mining — our approach uses iterative prompt engineering and operational constraints to classify unstructured PR threads into OWASP Top 10 2025 categories [11]. Using a pilot validation set, the classification pipeline achieved an accuracy of 0.98, which we then applied to classify the full corpus of 18,524 Django PRs. Large-scale analysis identified 1,917 security discussions, predominantly involving the Mishandling of Exceptional Conditions (A10). Finally, a qualitative peer review by senior developers confirmed that the framework reliably transforms historical developer dialogs into production-ready security policies.

Keywords

Large Language Models, Software Security, Repository Mining, Pull Requests, OWASP Top 10, Security Documentation

1 Introduction

Security in open-source software (OSS) development is increasingly crucial for managing vulnerabilities and guiding users toward safe practices. Establishing a comprehensive, formal security policy is recognized as one of the most critical measures to minimize vulnerabilities, comply with emerging standards, and ensure the overall health of a software ecosystem [16].

Despite this recognized importance, there is a significant gap between the need for security documentation and its actual implementation in OSS projects [2]. Furthermore, existing efforts to promote security policies predominantly focus on establishing basic vulnerability reporting mechanisms. There is a critical lack of comprehensive *User Guidelines*, project-specific instructions covering the secure usage of dependencies, recommended configurations, and general project usage [4]. Consequently, this absence of developer-aligned guidelines leaves projects directly exposed to *security-by-obscurity* risks.

In this paper, we propose a multi-stage framework to automatically generate project-specific security guidelines directly from historical developer discussions. We demonstrate our approach using a dataset of 18,524 Pull Requests (PRs) from the Django framework [6]. By adapting the PRIMES methodology [5], we employ iterative prompt engineering and statistical validation to accurately classify unstructured PR discussions into OWASP Top 10 2025 [11] vulnerabilities. Our results show that the model successfully identifies prevalent issues and synthesizes these historical comments into functional *User Guidelines*. We also asked three experts to evaluate the security guidelines automatically generated for Django. Overall, the guidelines received an average score of 4.75 out of 5, with 93.3% of the generated recommendations classified as strictly correct.

The primary contribution of this work lies in the adaptation and proposition of a structured methodology for leveraging LLMs in the context of Pull Request analysis. Particularly, this pipeline involves a multi-step workflow with empirical validations, ensuring that the framework does not rely exclusively on prompting. While this methodology is applied here to derive security guidelines for a widely adopted system (Django), it can be adapted in the future to analyze other software quality attributes such as performance, usability, architectural compliance, and testing across diverse software ecosystems.

The remainder of this paper is organized as follows. Section 2 describes the proposed methodology and prompt design. Section 3 presents the experimental results, including the OWASP classification, the generated guidelines, and the expert evaluation. Section 4 discusses threats to validity, Section 5 reviews related work, and Section 6 concludes the paper.

2 Methodology

This section describes the research design and methodology used to analyze security-related discussions in Pull Requests (PRs) and generate project-specific security policies. As illustrated in Figure 1, our approach follows a multi-stage framework, adapted from the PRIMES methodology [5], which relies on data collection from real-world software repositories, iterative prompt engineering for vulnerability classification, and a multi-layered validation process to ensure the reliability and consistency of the model's outputs.

2.1 Target Project and Dataset

Our study leverages a dataset with 18,524 Pull Requests (PRs) from the Django framework [6] collected via the GitHub API [7] on September 16th, 2024, comprising all PRs in the repository regardless of status. This selection is justified by Django's role as a real-world web framework that implements extensive security practices. To transform this raw PR data into a structured format for LLM analysis, we developed a parsing pipeline that maps the unstructured messages into a unified JSON schema, containing three section keys (see the example in Listing 1):

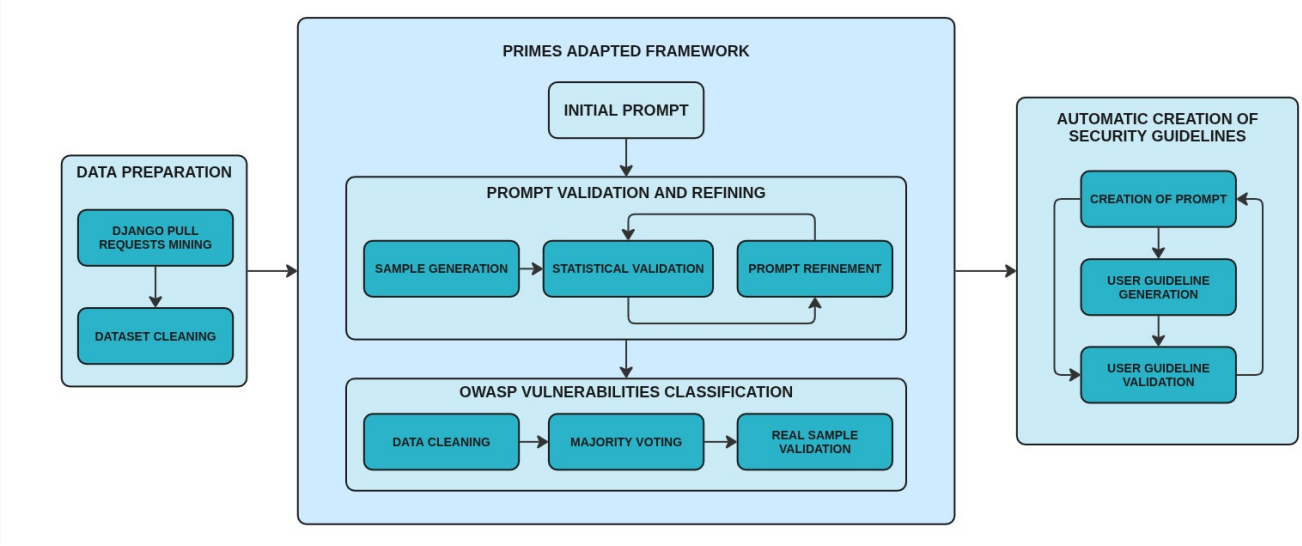

Figure 1: Overview of the multi-stage framework for security guideline generation from PR discussions.

(i) PR Context, comprising metadata such as the ID, title, and merge status.
(ii) General Discussions, corresponding to what the GitHub REST API defines as *Issue Comments*, which are conversations associated with the PR rather than specific code blocks. Thus, they are used to discuss high-level architectural issues, design critiques, and core conceptual exchanges.
(iii) Code Review Threads, corresponding to what the GitHub API [7] defines as *Review Comments*, which are discussions that target specific lines of the submitted diff.

Listing 1: Pull Request Example

```
{
  "context": {
    "repository": "django/django",
    "pr_id": "16420",
    "title": "Fix: IDOR in account endpoint",
    "is_merged": false
  },
  "general_discussion": [
    "Fixed issue where changing the '?acct=' parameter
        allowed access to other users' accounts."
  ],
  "code_review_threads": [
    {
      "scope": "FILE:django/contrib/auth/views.py",
      "comments": ["Good fix. Filtering the query by `
          request.user` prevents unauthorized access."]
    }
  ]
}
```

2.2 Initial Prompt

This phase is dedicated to the iterative refinement of the prompts used to interact with the LLM. We selected Gemini 3.1 Flash Lite for its low cost in large-scale batch processing, accessed via the official Google GenAI SDK [9] with a temperature of 0, structured JSON output, and batches of 20 PRs per call. The prompting strategy adopted the Google AI for Developers example template [8], optimized for long-context understanding and structured data extraction. The prompt can be seen in Listing 2.

Listing 2: First prompt structure

```
<role>
You are a security expert specializing in OWASP Top 10
     2025 vulnerabilities in code reviews and PR
     discussions.
</role>

<instructions>
Analyze the provided list of PRs. For each PR, execute
     the following steps:
1. Determine if the discussion DIRECTLY SPECIFIES a
     security hole.
2. Map concerns ONLY to the provided <categories>.
3. Identify the "Nature of Action": FIX/PREVENTION or
     VULNERABILITY_INTRODUCTION.
4. Present the final answer as a list of JSON objects.
</instructions>

<categories>
"A01: Broken Access Control", ..., "A10: Mishandling of
     Exceptional Conditions", "NONE"
</categories>

<output_format>
Return a JSON list:
[{
    "pr_id": "PR_ID",
    "owasp_category": "Category Name",
    "nature": "FIX/PREVENTION |
        VULNERABILITY_INTRODUCTION | NONE",
    "summary": "Short justification"
}]
</output_format>
```

2.3 Prompt Validation and Refinement

2.3.1 Sample Generation. Since our study involves a classification task, initial validation does not rely on a random sample, which could suffer from class imbalance and fail to represent all OWASP categories. Instead, we developed a controlled pilot dataset composed of 50 Pull Request discussions. To construct this dataset, we operationalized the "Example Attack Scenarios" documented

within the official OWASP Top 10 2025 framework specifications, translating them into simulated PR discussions; the sample size and per-category allocation followed the official scenario counts, excluding those incoherent with Django. For example, the PR in Listing 1 was created based on the scenario "Scenario 1: The application uses unverified data in an SQL call that accesses account information." from the Broken Access Control category of the OWASP website [11].

2.3.2 Statistical Validation and Prompt Refinement. To validate our pilot dataset, we employed a comprehensive suite of classification metrics. We utilized the scikit-learn library to calculate the output accuracy, macro-averaged precision, recall, and F1-Score, ensuring each OWASP category is treated with equal importance regardless of frequency (note that accuracy and macro metrics reflect strict multi-class OWASP categorization, whereas TP, FP, FN, and TN represent binary vulnerability detection). Furthermore, we implemented a Confusion Matrix to isolate misclassification patterns across categories, which then guided a recursive prompt-engineering cycle aimed at optimizing accuracy and mitigating systemic biases, until the prompt achieved the desired stability and precision.

2.3.3 Results. The preliminary outcomes of our initial interaction are illustrated in Table 1. Although the performance metrics demonstrate substantial precision, the presence of a false positive in our dataset of 50 PRs highlights a potential risk, particularly given the volume of PRs in production environments that do not discuss security-related issues.

Table 1: OWASP Pilot Set Categorization: Initial Iteration

Metric	Initial Iteration
Accuracy	0.9000
Macro F_1-Score	0.8946
Macro Precision	0.9136
Macro Recall	0.8991
True Positives (TP)	37
False Positives (FP)	1
False Negatives (FN)	0
True Negatives (TN)	12

The initial Confusion Matrix in Figure 2 reveals that certain OWASP categories are harder to classify than others, indicating room for improvement through iterative prompt refinement.

Leveraging this evaluation and the insights derived from our iterative refinement cycle, we established the final set of operational constraints presented below and shown in Listing 3.

- Negative Sampling Default: We instructed the model to default to *NONE* for any maintenance or functional tasks lacking an explicit security exploit.
- Intent-Based Classification: We established the *Original Intent Rule* to ensure that the nature of the change (Vulnerability Introduction vs. Fix) is evaluated based on the code impact, independent of the PR's final status (Merged/Closed).

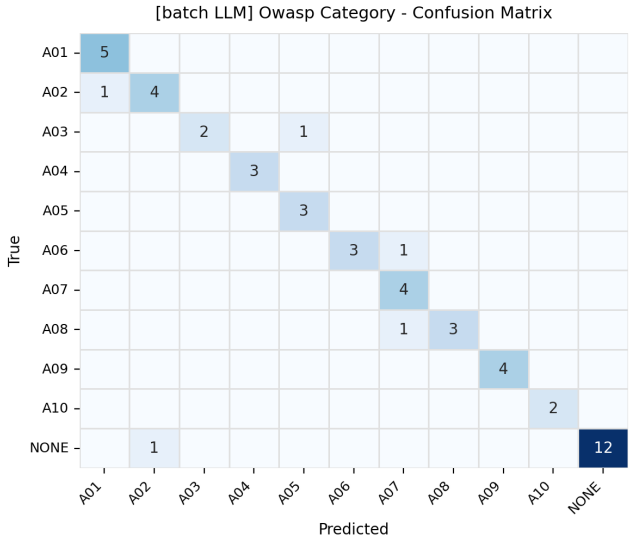

Figure 2: Initial prompt confusion matrix

- Heuristic Priority Mapping: We provided specific disambiguation rules for complex OWASP categories, such as prioritizing *Insecure Design (A06:2025)* for logic flaws and *Software and Data Integrity Failures (A08:2025)* for verification-related issues.
- Strict Grounding: A *No Inference* policy is enforced, requiring the model to rely strictly on direct evidence from the PR discussion.

Listing 3: Prompt Constraints

```
<constraints>
- If the discussion is about General maintenance, UI/UX,
performance, or bug fixes without direct security impact,
 it MUST be "NONE".
- Classify "Nature" based on the author's original code.
If the code introduces a flaw, it is
VULNERABILITY_INTRODUCTION, regardless of whether the PR
was Merged or Closed.
- Classify fundamental logic flaws (e.g., security
questions, MFA bypasses) as A06.
- Prioritize A02 for technical exposures (stack traces)
or incorrect permissions (ACLs).
- Use A08 for any lack of verification regarding
untrusted data, headers, signatures, or deserialization (
Pickle/JSON).
- If a library itself is the source of a flaw, default to
 A03.
- DO NOT assume a vulnerability exists unless the PR
discussion DIRECTLY SPECIFIES a security hole, "potential
" or "possible" are not enough.
- The PR must have a discussion to be classified, only
the title and description are not enough.
</constraints>
```

Upon executing this optimized configuration, the performance outcomes detailed in Table 2 demonstrate the achievement of zero False Positives (FP) and robust classification metrics (accuracy 0.98 and macro precision 0.9848).

This high degree of precision is further evidenced by the confusion matrix in Figure 3, which exhibits a near-perfect diagonal alignment.

Table 2: OWASP Pilot Set Categorization: Refined Iteration

Metric	Refined Iteration
Accuracy	0.9800
Macro F_1-Score	0.9816
Macro Precision	0.9848
Macro Recall	0.9818
True Positives (TP)	37
False Positives (FP)	0
False Negatives (FN)	0
True Negatives (TN)	13

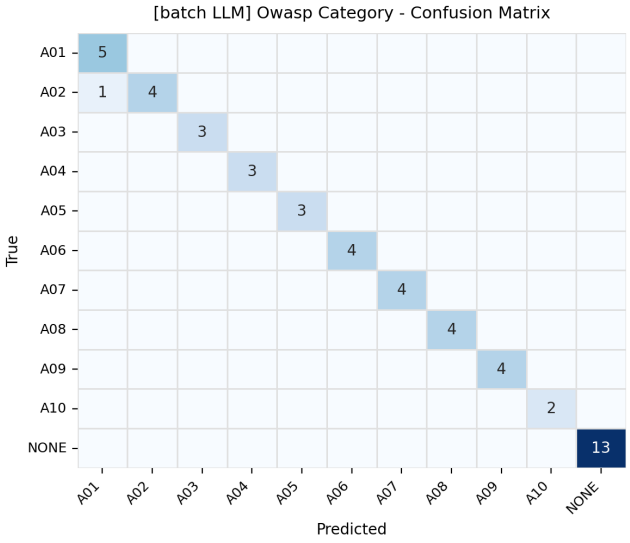

Figure 3: Final prompt confusion matrix

2.4 Automatic Creation of Security Guidelines

The final stage of our methodology involves the synthesis of the identified and classified security vulnerabilities into a security policy. According to recent empirical evidence [4], well-structured security policies are strong indicators of a project's maturity and help mitigate *security-by-obscurity* risks. To this end, we formulated a structured prompt (Listing 4) designed to synthesize our findings into a User Guideline (one of the sections recommended for a *SECURITY.md* document), providing project-specific instructions for secure usage and recommended configurations based on observed vulnerability patterns. To generate the guideline, we employed Claude Sonnet 4.6 via Anthropic's conversational interface (claude.ai) [1], given its instruction-following performance, providing the full JSON records of the 1,917 classified PRs in-context.

Listing 4: Artifact Generation Prompt

```
Based on the security classifications extracted from the
following Django Pull Request discussions, generate the
User Guideline section for the Django repository's
security policy (SECURITY.md). The guideline must provide
 security-related and project-specific instructions,
including the secure usage of dependencies, recommended
configurations, and general project usage based on the
security patterns and lessons learned identified in the
provided data. The response must be a markdown file, and
the final guideline must strictly avoid generic
recommendations, focusing only on the provided evidence.
```

3 Results

This section presents the results of applying the proposed methodology: the classification of security vulnerabilities in the analyzed PRs (Section 3.1), an overview of the security guidelines automatically generated for Django (Section 3.2), and their validation by three experienced Django developers (Section 3.3).

3.1 OWASP Vulnerabilities Classification

We applied the methodology described in the previous section to the full Django dataset of 18,524 PRs. Following the initial execution, we performed a data cleaning step to ensure all outputs adhered to the required schema. To mitigate potential hallucinations and the inherent stochasticity of the LLM, we implemented a self-consistency strategy: each PR was processed three separate times. We then applied a majority voting ensemble to determine the final OWASP category. This triple-run approach ensures that the results represent the model's consensus rather than an isolated outlier. Disagreement across the three runs occurred in only 20 PRs (0.11% of the dataset). To ensure reliability, we also audited a stratified sample, randomly drawn at 5% per category (except NONE, which was fixed at 50 PRs given the size of its pool). The audit was conducted by one author (a senior Django developer with five years of experience), who judged each classification against the PR discussion and OWASP definitions, with visibility into the LLM's output.

The classification results and the manual audit are detailed in Table 3. The distribution reveals that *A10: Mishandling of Exceptional Conditions* is the most frequent security concern, with 398 occurrences (20.76% of the identified vulnerabilities, excluding *NONE*), followed by *A02: Security Misconfiguration* (327, 17.06%) and *A01: Broken Access Control* (267, 13.93%). The manual audit of the 151 sampled PRs demonstrated an accuracy of 79.47% (120 fully correct classifications) and an overall acceptable alignment of 99.34% (including 30 partial matches, where either the category or the justification is correct, but not both).

3.2 AI-Generated Security Guidelines

After classifying the PRs into OWASP categories, we applied the prompt from Section 2.4 with the Claude model to generate the security guidelines. The resulting artifact is compiled into a document containing 30 development rules. These recommendations are categorized into 9 security domains: *Configuration and Secrets Management*, *Dependency Management*, *Authentication and Session Security*, *SQL and ORM Safety*, *Template and XSS Safety*, *Access*

Table 3: OWASP Category Sampling Analysis

Category	Total	Sample	Correct	Partial	Incorrect
A01	267	14	6	8	0
A02	327	17	8	9	0
A03	86	5	5	0	0
A04	112	6	4	2	0
A05	229	12	10	2	0
A06	167	9	7	2	0
A07	50	3	3	0	0
A08	178	9	4	4	1
A09	103	6	4	2	0
A10	398	20	19	1	0
NONE	16,607	50	50	0	0
Total	**18,524**	**151**	**120**	**30**	**1**

Control, *Cryptographic Practices*, *Sensitive Data in Logs and Tracebacks*, and *Exception Handling and Resilience*. This categorization aligns with real-world security standards, mapping effectively to the OWASP Top 10 2025 classifications adapted to the specificities of the Django framework.

To illustrate the structure and granularity of the generated policy, Table 4 presents the first three recommendations from the *Configuration and Secrets Management* domain. The complete security document is fully available within our replication package in the artifacts section.

3.3 Validation with Experts

To ensure technical accuracy and practical utility, the generated User Guideline was validated by three senior Django developers, each with at least five years of Django experience, recruited from one author's professional network. This validation process was operationalized through an online questionnaire sent to the experts. The evaluation instrument utilized a 5-point Likert scale (ranging from 1 - "Strongly Disagree" to 5 - "Strongly Agree") where experts evaluated the artifact based on three primary criteria: the technical correctness of the remediation steps, the clarity of the security guidelines, and adherence to Django's specific security best practices. Furthermore, alongside the Likert scale evaluation, the experts were tasked with explicitly classifying each individual recommendation within the final generated document into one of three categories: "Correct," "Partially Correct," or "Incorrect."

The synthesized performance outcomes and consolidated metrics derived from our expert evaluation instrument are presented in Tables 5 and 6. The quantitative results were highly positive, with the generated guidelines achieving an overall mean of 4.75 out of 5 from all participating experts. Furthermore, 93.3% of the individual recommendations were classified as strictly "Correct", where a recommendation is Correct only if all three evaluators agreed, and Partially Correct otherwise.

Qualitative feedback provided deeper insights into the framework's practical efficacy, highlighting the following key themes:

3.3.1 Technical Correctness and Framework Specificity. Evaluators praised the high degree of framework specificity, noting the accurate inclusion of native Django mechanisms (e.g., `ALLOWED_HOSTS`, `RawSQL`, `format_html`). Regarding its technical alignment, Evaluator 2 noted that *"the guide is technically correct for the most part and aligned with Django's best practices,"* adding that minor limitations exist only because *"some points depend on the Django version or the environment"*.

3.3.2 Clarity and Practical Utility. The experts confirmed the artifact's high utility for daily tasks, such as PR reviews and security hardening checklists. Evaluator 1 noted that for junior developers, it *"serves as a structured reference for adopting correct standards,"* while for senior engineers, it represents an *"opportunity to deepen knowledge."* To further enhance clarity, experts suggested structural improvements to the text. Specifically, Evaluator 2 stated that the guide *"could be more concise by separating recommendations that mix different subjects in the same item and adding quick examples of 'wrong' and 'correct'."*

3.3.3 Identified Gaps and Future Enhancements. When asked for missing topics, evaluators identified areas reflecting modern web development trends:

- Security considerations for asynchronous operations introduced in recent Django versions.
- Secure deployment strategies for architectures like Kubernetes and Serverless platforms.
- API-specific security measures, particularly integrating Django REST Framework (DRF) concepts (e.g., object-level permissions, rate limiting).
- File upload security, including strict extension validation and safe storage.

Overall, the experts concluded that the model successfully transformed historical pull request data into a production-ready security policy. The panel emphasized that the artifact serves a dual purpose: acting as a structured reference for junior developers and as a robust validation tool for senior engineers, ultimately elevating the software ecosystem's security posture.

4 Threats to Validity

The primary threat to internal and construct validity concerns the subjectivity inherent in the qualitative evaluation conducted with human experts. Although the review panel comprised senior Django developers, individual interpretations of what constitutes an optimal, clear, or actionable security guideline can vary. A further threat is attribution: recommendations are not explicitly traced back to the PRs that motivated them, making it harder to distinguish guidance grounded in mined discussions from the LLM's prior Django knowledge.

Regarding external validity, our findings may lack generalization across the broader ecosystem because the proposed solution was evaluated exclusively on historical data from the Django repository. While Django represents a high-stakes, real-world framework, the generated user guidelines reflect only the vulnerability patterns observed within its own code-base. Thus, our dataset does not account for security discussions or practical misconfigurations that occur downstream in client repositories that depend on Django.

Table 4: Initial Recommendations Sample (Configuration Domain)

ID	Title	Description
REC-01	Validate SECRET_KEY Format and Rotation	The key must be a valid, non-empty Unicode string. Django enforces this at startup. Use SECRET_KEY_FALLBACKS for seamless cryptographic rotation.
REC-02	Secret and Local Settings Exclusion	Never commit local settings or raw credentials to git. PR history records leaks of plaintext DB passwords and unsecure screenshot attachments.
REC-03	Production Disable of DEBUG = True	Debug pages leak tokens, request.user, and variables. In addition, it keeps queries in memory, causing a MemoryError during large migrations.

Table 5: Qualitative Evaluation of Security Guidelines by Expert. A1: Technical Correctness; A2: Usefulness; A3: Clarity; A4: Django Specificity; A5: Overall (mean of A1–A4)

Expert	A1	A2	A3	A4	A5
E1	5	5	5	4	4.75
E2	4	5	4	5	4.50
E3	5	5	5	5	5.00
Mean	4.67	5.00	4.67	4.67	4.75

Table 6: Accuracy of the Generated Recommendations

Expert Classification	Recommendations	Percentage
Correct	28	93.3%
Partially Correct	2	6.7%
Incorrect	0	0%
Total	**30**	

5 Related Work

Recent empirical research has examined the adoption and structure of security policies in open-source software (OSS). Zahan et al. [16] identify the establishment of a formal security policy as one of the most critical practices for minimizing vulnerabilities, motivating our use of security policy generation as the primary goal of this framework; however, Ayala et al. [2] find that only a small fraction of popular GitHub repositories define one, and Bühlmann and Ghafari [3] show that security issues are often resolved without discussion or delayed due to a lack of reproducibility data. More recently, Choetkiertikul et al. [4] propose a categorization of GitHub SECURITY.md policies to assess OSS security maturity, revealing that existing literature primarily examines vulnerability reporting mechanisms and leaves a critical gap regarding secure usage instructions, also known as User Guidelines; it is precisely this gap that motivates our approach to focus explicitly on synthesizing User Guidelines rather than generic vulnerability templates.

With the recent expansion of Large Language Models (LLMs) in Software Engineering (SE), researchers are increasingly exploring their analytical capabilities for repository mining. Hou et al. [10] propose applying LLMs to identify and remediate code flaws and categorize system specifications for secure development, and several studies exemplify targeted repository analysis with LLMs, including Silva et al. [14] for code smell detection, Serrano et al. [13] for security intelligence extraction, Della Porta et al. [12] for documentation generation, and Tufano et al. [15], who mine commits, issues, and PRs to categorize tasks developers automate using ChatGPT. However, extracting reliable data from unstructured developer discussions requires methodological governance: De Martino et al. [5] propose the PRIMES framework for extracting information from software repositories via LLMs, showing that iterative prompt engineering, sample generation, and statistical validation are essential to mitigate hallucinations and ensure reproducibility; we use an adapted version of PRIMES to govern our classification of OWASP vulnerabilities from Django Pull Requests, extending this pipeline to synthesize technical discussions into security guidelines.

6 Conclusion and Future Work

In this paper, we propose a multi-stage framework that leverages Large Language Models to automatically synthesize project-specific security guidelines from Pull Request discussions. Validated through a rigorous pipeline and qualitative expert review, the approach demonstrated high precision in classifying vulnerabilities and transforming unstructured developer dialogues into documentation.

For future work, we plan to expand this research along three main avenues. First, to overcome current generalization limits, we intend to extend the study to downstream client repositories that depend on Django. Second, we aim to generalize the underlying pipeline into an adaptable, project-agnostic tool capable of ingesting discussions from any software repository to generate custom security user guidelines. Finally, we propose to broaden the scope of the synthesis framework beyond software security, evaluating its capacity to generate structured user guidelines for other development domains, such as testing, CI/CD configurations, and architectural practices.

Artifact Availability

The replication package of this study is publicly available in https://github.com/estersassis/llm-pr-security-classifier.

Acknowledgments

This research was supported by grants from FAPEMIG (grant APQ-02419-23) and CNPq (grant 403304/2025-3). It was also partially supported by INES.IA (National Institute of Science and Technology for Software Engineering Based on and for Artificial Intelligence), www.ines.org.br, CNPq grant 408817/2024-0.

References

[1] Anthropic. 2026. Claude.ai. https://claude.ai AI assistant platform. Accessed: 2026-08-08.

[2] Jessy Ayala and Joshua Garcia. 2023. An Empirical Study on Workflows and Security Policies in Popular GitHub Repositories. In *2023 IEEE/ACM 1st International Workshop on Software Vulnerability (SVM)*. IEEE, 6–9. doi:10.1109/svm59160.2023.00006

[3] Noah Bühlmann and Mohammad Ghafari. 2022. How do developers deal with security issue reports on GitHub?. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing* (Virtual Event) *(SAC '22)*. Association for Computing Machinery, New York, NY, USA, 1580–1589. doi:10.1145/3477314.3507123

[4] Morakot Choetkiertikul, Sushawapak Kancharoendee, Chanikarn Jongyingyos, Thanat Phichitphanphong, Chaiyong Ragkhitwetsagul, Brittany Reid, Raula Kula, and Thanwadee Sunetnanta. 2026. Security by documentation? characterizing GitHub SECURITY.md policy and their adoption in Python libraries. *Empirical Software Engineering* 31 (02 2026). doi:10.1007/s10664-025-10794-z

[5] Vincenzo De Martino, Joel Castano, Fabio Palomba, Xavier Franch, and Silverio Martinez-Fernandez. 2025. A Framework for Using LLMs for Repository Mining Studies in Empirical Software Engineering . In *2025 IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE)*. IEEE Computer Society, 6–11.

[6] Django Software Foundation. 2026. Django: A high-level Python web framework. https://github.com/django/django GitHub repository. Accessed: 2026-06-17.

[7] GitHub, Inc. 2026. GitHub REST API documentation. https://docs.github.com/pt/rest Official documentation. Accessed: 2026-06-17.

[8] Google AI for Developers. 2026. Prompting strategies: Example template combining best practices. https://ai.google.dev/gemini-api/docs/prompting-strategies#example_template_combining_best_practices Gemini API Documentation. Accessed: 2026-06-17.

[9] Google Cloud. 2026. Google Gen AI SDK. https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/sdks/overview?hl=pt-br Google Cloud Documentation. Accessed: 2026-08-08.

[10] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE] https://arxiv.org/abs/2308.10620

[11] OWASP Foundation. 2025. OWASP Top 10:2025. https://owasp.org/Top10/2025/ Accessed: 2026-06-17.

[12] Antonio Della Porta, Vincenzo De Martino, Gilberto Recupito, Carmine Iemmino, Gemma Catolino, Dario Di Nucci, and Fabio Palomba. 2024. Using Large Language Models to Support Software Engineering Documentation in Waterfall Life Cycles: Are We There Yet?. In *Proceedings of the Ital-IA Intelligenza Artificiale - Thematic Workshops co-located with the 4th CINI National Lab AIIS Conference on Artificial Intelligence (Ital-IA 2024), Naples, Italy, May 29-30, 2024 (CEUR Workshop Proceedings, Vol. 3762)*, Sergio Di Martino, Carlo Sansone, Elio Masciari, Silvia Rossi, and Michela Gravina (Eds.). CEUR-WS.org, 42–47. https://ceur-ws.org/Vol-3762/523.pdf

[13] Patrick Serrano, Luwen Huangfu, Chunhua Liao, Dongqing Lin, Kai Williams, Jaden Perleoni, and Thomas Brettin. 2025. Large Language Model (LLM)-Driven Document Clustering: Improving Real-Time Security Intelligence Extraction and Threat Analysis. In *2025 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 7–14. doi:10.1109/ISI65680.2025.11201090

[14] Luciana Lourdes Silva, Janio Rosa da Silva, Joao Eduardo Montandon, Marcus Andrade, and Marco Tulio Valente. 2024. Detecting Code Smells using ChatGPT: Initial Insights. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Barcelona, Spain) *(ESEM '24)*. Association for Computing Machinery, New York, NY, USA, 400–406. doi:10.1145/3674805.3690742

[15] Rosalia Tufano, Antonio Mastropaolo, Federica Pepe, Ozren Dabić, Massimiliano Di Penta, and Gabriele Bavota. 2024. Unveiling ChatGPT's Usage in Open Source Projects: A Mining-based Study. arXiv:2402.16480 [cs.SE] https://arxiv.org/abs/2402.16480

[16] Nusrat Zahan, Shohanuzzaman Shohan, Dan Harris, and Laurie Williams. 2023. Do Software Security Practices Yield Fewer Vulnerabilities?. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 292–303. doi:10.1109/ICSE-SEIP58684.2023.00032