

# Scylla: A Unified Tool for Code Smell Classification in Python

Igor Soares de Oliveira  
PUC-Rio  
Rio de Janeiro, Brazil  
isoliveira@inf.puc-rio.br

Gabriel Gervasio  
PUC-Rio  
Rio de Janeiro, Brazil  
gabriel@aise.inf.puc-rio.br

Alexandre Andrade  
PUC-Rio  
Rio de Janeiro, Brazil  
acesar@aise.inf.puc-rio.br

Juliana Alves Pereira  
PUC-Rio  
Rio de Janeiro, Brazil  
juliana@inf.puc-rio.br

## Abstract

Code smells are structural indicators of poor design that can compromise software maintainability, readability, and scalability. Their detection and mitigation are essential to reduce technical debt and ensure the long-term quality of systems. Traditional approaches rely on static analysis and rule-based tools, which often lack contextual understanding and adaptability to evolving codebases. This paper presents Scylla, a unified tool for code smell classification in Python that unifies multiple detection approaches with a single environment: Abstract Syntax Tree (AST) analysis, Machine Learning (ML), Deep Learning (DL), and Small Language Models (SLMs). Scylla adopts an event-driven microservice architecture composed of three main components: (1) an API responsible for coordinating and processing classification tasks asynchronously, persisting AST-, ML-, and DL-based results in a PostgreSQL database; (2) a worker service that executes SLM-based classifications via Ollama and stores results in a PostgreSQL database; and (3) a web application providing an interactive interface for code submission and classification. Scylla's modular design ensures scalability, extensibility, and fault tolerance, supporting future integration of new code smells, models, and programming languages. Experimental results demonstrate the potential of Scylla as a unified and extensible framework for automated software quality assessment and comparative research on code smell detection techniques. Video: <https://youtu.be/dIyqNGGTdbE> Tool: <https://github.com/aisepucrio/lm4smells-core>

## 1 Introduction

Code smells are structural or design characteristics in software systems that suggest potential quality issues and may increase maintenance complexity. Initially introduced by Fowler et al. [11], they represent symptoms of violations of software development principles—such as excessive coupling, low cohesion, or redundant functionality—that hinder program comprehension and contribute to higher maintenance costs and defect risks. Moreover, code smells accumulate technical debt, elevate the likelihood of failures, and reduce the productivity of development teams [18]. Refactoring—the process of restructuring existing code without changing its external behavior—is the primary strategy to address these problems. However, its effectiveness strongly depends on the accurate and timely identification of code smells. Manual inspection of large codebases to identify code smells is time-consuming, error-prone, and non-scalable, particularly as modern software projects grow in size and complexity. Consequently, there is a growing need for automated, reliable detection tools that systematically analyze code and provide consistent feedback to support refactoring decisions and continuous quality improvement.

Despite Python's growing popularity, the detection of code smells in Python remains underexplored. Existing tools, such as Pylint

[31] and Pyflakes [25], focus mainly on syntactic errors and stylistic inconsistencies, offering limited insight into design-level or structural smells. Research-oriented metric-based tools, such as PySmell [6] and Dpy [3], while effective at identifying threshold violations, often lack contextual understanding, failing to capture the subtle semantic and structural relationships that characterize more complex code smells. Moreover, they are either discontinued or not openly available, leaving a gap in reliable, actively maintained, and up-to-date solutions. Furthermore, none of the existing tools provides a unified environment that supports multiple AI-driven approaches to code smell detection, such as those based on Machine Learning (ML), Deep Learning (DL), or Language Models (LMs). This limitation highlights the need for modern, extensible, and intelligent tools capable of leveraging both analytical metrics and AI-based reasoning to enhance the accuracy and interpretability of code smell detection in Python.

To address these limitations, we propose Scylla<sup>1</sup>, an open-source web-based tool for detecting code smells in Python. Unlike existing tools that typically focus on a single detection approach, Scylla integrates multiple AI-driven and analytical approaches, including Abstract Syntax Tree (AST) analysis, ML, DL, and LMs. We evaluated Scylla through a user-centered experimental study to assess users' overall experience and task effectiveness. Our evaluation captures the impact of Scylla across varying levels of analytical experience, focusing on how well it assists navigation, classification, and interpretation of code smells during the review process.

Our main contributions are threefold: (1) We present Scylla, a unified tool for classifying code smells in Python using four approaches—AST, ML, DL, and LMs. (2) We introduce an extensible architecture that supports a wide range of code smell types and designed to accommodate future integration of additional approaches, models, and programming languages. (3) We provide empirical evidence, based on controlled experiments with Python developers, that Scylla assists code review activities by streamlining the classification of diverse code smell types, enabling comparisons across detection approaches, and helping developers better interpret and reason about potential quality issues in the analyzed code.

## 2 Related Work

The detection of code smells has traditionally relied on AST-based tools, but recent years have seen growing interest in ML, DL, and LLM-based approaches. One of the earliest and most influential studies in this area was conducted by Mäntylä and Lassenius [17], who proposed a taxonomy of bad smells and demonstrated their impact on software maintainability. Their work laid the foundation for automated smell detection research. Later, Lanza and Marinescu

<sup>1</sup>Scylla takes its name from a famous creature in Greek mythology. Like the mythical guardian lurking in treacherous waters, Scylla symbolizes vigilance against hidden dangers (i.e. code smells) that can hinder code comprehension. The name reflects the tool's purpose to detect and expose such threats before they "devour" code quality.

[15] formalized the use of structural metrics and detection strategies to identify smells such as Long Method and Long Parameter List.

In the Python ecosystem, several semantic-based and AST-based tools are widely used. Pylint [31] and Flake8 [24] perform stylistic and structural checks to identify dead code, style violations, and maintainability issues. Similarly, MLpylint [14] was designed to detect ML-specific code smells. The DPY [8], PySmell [5], and PyExamine [27] use AST-based analysis to detect Python-specific code smells through structural metrics and semantic heuristics. Several studies employ ML and DL models to automatically classify code smells by learning patterns from source code representations (such as ASTs, tokens, and code embeddings) and using these learned features to distinguish smelly from non-smelly code. Azeem et al. [2] presented a systematic review of supervised algorithms, including Decision Trees, Random Forests, and SVMs, showing that ensemble and rule-based models achieve high performance. For instance, Doe et al. [7] used techniques like XGBoost and Bagging combined with SMOTE to handle class imbalance, achieving strong predictive performance for Python code smell detection. Previous studies [16, 28] use DL for smell detection, leveraging embeddings derived from syntactic structures. Recently, studies [12, 22] demonstrated that LMs can accurately identify smells, adapting dynamically to different code contexts. These learning-driven approaches represent a shift from purely static, rule-based detection to more context-aware and data-driven analysis, enabling the discovery of subtle and domain-specific code smells. In this context, Scylla aims to integrate the strengths of all these paradigms—AST-based analysis, ML and DL, and LM-based contextual reasoning—into a modular and scalable framework for code smell detection in Python.

### 3 Scylla

Scylla was developed following the principles of event-driven microservice architecture. This architectural approach allows services to operate independently while communicating asynchronously through well-defined events, ensuring robustness and fault tolerance. It also facilitates the incremental incorporation of new types of code smells, including those from different programming languages, with minimal impact on existing components. Figure 1 illustrates the architecture of Scylla, highlighting its main services and data flow. Next, we describe each of these components, explaining their roles, interactions, and how they collectively support multi-approach code smell classification within a unified tool.

**① Web:** An interactive web application was developed to facilitate the classification of Python code. Through this interface, users can apply multiple classification approaches to detect and analyze code smells, including AST-based heuristics, ML, DL, and LMs. The web interface allows users to select entities to be analyzed (e.g., classes or methods), the code smell type, the heuristics/model, upload one or more source code files, and track processing progress in real time through dynamic visual feedback. Upon task completion, Scylla provides detailed, exportable reports in CSV format, ensuring transparency, traceability, and reproducibility of the analyses performed. The web application was implemented using widely adopted technologies, JavaScript [10], CSS [20], and HTML [30].

**② code-extractor.api:** It serves as the central processing component within the proposed architecture. *code-extractor.api* is responsible for orchestrating the reception of requests from the *Web*

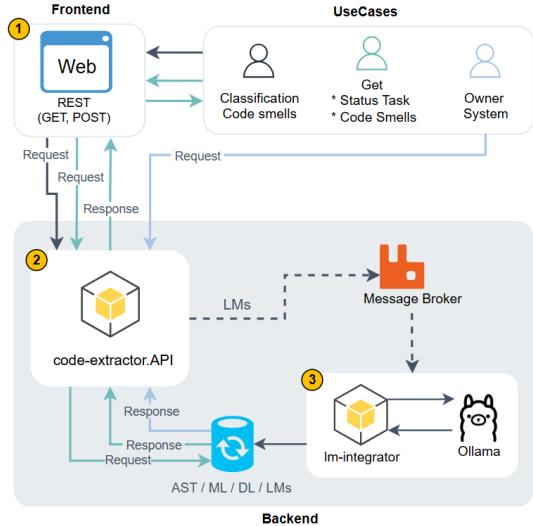


Figure 1: Scylla’s architecture.

service, coordinating classification operations, task scheduling, and querying of analytical results through well-defined endpoints. Its implementation follows a layered design structure—*Application*, *Domain*, and *Infrastructure*—according to the principles of Domain-Driven Design (DDD) [9]. This modular separation improves maintainability, scalability, and fault isolation, while fostering a high degree of decoupling among workflow stages. Furthermore, the service provides asynchronous mechanisms for managing execution states—such as scheduling, in-processing, and completion—ensuring consistent task control across distributed components. This architectural design also establishes a robust foundation for extensibility of new analytical capabilities.

**Message Broker:** The message-oriented middleware adopted in this architecture is *RabbitMQ* [26], which serves as the central broker for asynchronous communication between system components. Its primary role is to handle the events generated by the classification requests assigned to the LMs. Upon receiving these events from the *code-extractor.API*, the broker queues them for processing by the *lm-integrator* service.

**③ lm-integrator:** This service was designed to asynchronously process classification tasks and notifications received from the message broker. Operating as an event-driven consumer, it processes code classification tasks through five LMs, either integrated locally via *Ollama* [23] or accessed through RESTful APIs [13, 19]: Deepseek-r1, Qwen2.5-coder, Mistral, Gemma2, and Codellama. These LMs were chosen based on our prior work [29]. Upon receiving events, the service executes the classification of code fragments and subsequently persists the processed results in the database.

**Database:** The database adopted in this architecture is *PostgreSQL* [21], serving as the central repository for storing and managing the results of code classification tasks. All classification outputs generated for the different analytical approaches—ASTs, ML, DL, and LMs—are consolidated within this persistence layer, which ensures data consistency, integrity, and reproducibility. Furthermore, this component provides a reliable foundation for data access and retrieval, supporting potential analytical or visualization layers

233 built upon the stored results and facilitating empirical validation of  
 234 classification accuracy across the different approaches.

## 235 4 Study Design

236 The goal of the defined experiments is to observe and validate  
 237 Scylla's practical use in supporting the automated classification of  
 238 code smells. It is important to note that, in [29], we assess the quality,  
 239 robustness, and reliability of the approaches implemented in  
 240 Scylla, providing complementary evidence of its technical soundness.  
 241 The experiments were structured to reflect realistic usage  
 242 scenarios, allowing observation of user interactions with Scylla's  
 243 main functionalities and assessment of their ability to successfully  
 244 execute classification tasks. The study was organized into three  
 245 main stages: ① Preparation, ② Execution, and ③ Data Analysis.

246 **① Preparation:** We formulated an experimental framework in-  
 247 spired by a previous study by Castro et al. [4], which guided the  
 248 structuring of the user tasks into three progressively distinct cat-  
 249 egories: Filtering Tasks (FT), Basic Tasks (BT), and Assimilation  
 250 Tasks (AT). This classification was adopted to ensure a systematic  
 251 and progressive assessment of user interaction with Scylla, rang-  
 252 ing from simple operational actions to more complex cognitive  
 253 reasoning processes. The FT represents straightforward activities  
 254 designed to confirm the participants' ability to perform fundamental  
 255 interactions within the tool. The BT evaluates users' understanding  
 256 of the core functionalities, focusing on their capacity to execute  
 257 standard operations successfully. Finally, the AT involves more so-  
 258 phisticated reasoning, requiring participants to interpret outputs,  
 259 correlate information across different analytical approaches, and  
 260 extract conclusions from the results produced by the system. The  
 261 complete list of tasks can be found in our complementary material<sup>2</sup>.

262 We conducted a pilot study with two Python developers to eval-  
 263 uate the clarity and feasibility of the task. This stage allowed for ad-  
 264 justments in content, duration, and format, making the instrument  
 265 more objective and applicable. The implemented improvements  
 266 reduced the average response time to approx. 60 minutes.

267 We selected Python developers with different technological back-  
 268 grounds and varying levels of expertise. The recruitment process  
 269 was conducted through a preliminary questionnaire aimed at vali-  
 270 dating each participant's knowledge and confirming their eligibility  
 271 to take part in the experimentation phase. Moreover, we prepared  
 272 an informed consent form (ICF) to ensure that all participants were  
 273 fully aware of the nature and objectives of the study. The document  
 274 provided a detailed description of the experiment's purpose, the  
 275 types of data to be collected and how such information would be  
 276 used to evaluate the proposed tool. Participation in the study was  
 277 conditional upon the reading and signing of the consent form.

278 **② Execution:** After providing consent, the execution phase was  
 279 organized into three stages. First, participants were introduced  
 280 to the study and informed about the tasks they were required to  
 281 perform, as well as the procedures to be followed during the experi-  
 282 ment. They received a 15-minute overview of Scylla, highlighting  
 283 its interface and main functionalities. Second, participants engaged  
 284 in an interview phase where they were asked to perform code clas-  
 285 sification and analysis tasks while interacting with Scylla and  
 286 verbalizing their reasoning following the think-aloud protocol. All

287 sessions were recorded via Zoom, capturing both screen activity  
 288 and audio for later transcription and analysis. Third, upon comple-  
 289 ting the tasks of the interview, participants answered a post-session  
 290 questionnaire<sup>3</sup> designed to collect complementary data. It com-  
 291 bined 14 open-ended and scaled questions based on the Technology  
 292 Acceptance Model (TAM) [1], assessing perceived ease of use, use-  
 293 fulness, and overall impressions of the tool, while also including  
 294 demographic questions to characterize the participants.

295 **③ Data Analysis:** The final stage of the study focused on analyz-  
 296 ing the collected results. This phase examined participants' re-  
 297 sponses to assess Scylla's performance and extract empirical  
 298 evidence supporting its quality. We computed metrics such as, task  
 299 completion rate, accuracy, and time spent per section. Moreover,  
 300 we analyzed participants' perceptions about Scylla's use.

301 We defined two research questions (RQs), as follows:

302 **RQ<sub>1</sub>. How effectively can Scylla support users in performing**  
**303 tasks of varying complexity within realistic code analysis**  
**304 and classification scenarios?** This question examines Scylla  
 305 usability as experienced by participants during the execution of  
 306 experimental tasks designed to replicate realistic code analysis  
 307 workflows. Participants interacted with Scylla through filtering,  
 308 basic, and assimilation tasks, each varying in cognitive demand and  
 309 required interface interaction.

310 **RQ<sub>2</sub>. How do users perceive the usefulness and practical**  
**311 applicability of Scylla?** This question focuses on capturing par-  
 312 ticipants' evaluation of Scylla's relevance, clarity, and alignment  
 313 with their analytical needs. To address it, we analyzed responses  
 314 from a structured post-session questionnaire.

## 315 5 Results and Discussions

316 A total of nine participants contributed to the study. Most partic-  
 317 ipants are specialized in Software Engineering (88.9%), followed  
 318 by Data Science (33.3%), and AI (11.1%). Regarding professional  
 319 experience in software development, 44.4% of participants reported  
 320 1–3 years of experience, 33.3% reported 4–6 years, while the  
 321 remaining indicated more than 7 years (both 11.1%). Concerning  
 322 familiarity with code smell classification tools, 66.7% indicated high  
 323 to moderate knowledge. When asked about the main advantages  
 324 of using tools for code smell classification, participants identified  
 325 several benefits. #P1, #P2, #P5, #P6, and #P9 mentioned "*tools are*  
*326 valuable mechanisms for enhancing software quality and supporting*  
*327 developers in producing cleaner, more maintainable code*". #P1 and  
*328 #P2 emphasized their "usefulness in guiding less-experienced profes-*  
*329 sionals toward recognizing and correcting design deficiencies". P3, P4*  
*330 and P5 mentioned their "role as auditors for expediting refactoring*  
*331 processes and mitigating the spread of structural issues in large-scale*  
*332 systems". Additionally, #P5, #P7, and #P8 noted that "*these tools*  
*333 enable analytical operations that would be hard to perform manually,*  
*334 particularly those involving complex metrics*". Collectively, these  
 335 perceptions indicate that participants are aware that automated  
 336 code smell classification tools contribute to software quality.*

### 337 5.1 Scylla's Effectivity (RQ<sub>1</sub>)

338 To address RQ<sub>1</sub>, we examined participants' performance during the  
 339 experimental tasks. The tasks were divided into three progressive

2<sup>2</sup><https://github.com/aisepucrio/lm4smells-core/tree/main/docs/tasks>

3<sup>3</sup><https://github.com/aisepucrio/lm4smells-core/tree/main/docs/questionnaire>

349 categories—FT, BT, and AT—to assess the effectiveness of Scylla  
 350 across increasing levels of cognitive and operational complexity.  
 351 Overall, the results demonstrate a high level of task completion  
 352 accuracy, reflecting Scylla's clarity, consistency, and reliability  
 353 throughout different interaction stages. The detailed table results  
 354 are available in our supplementary material<sup>4</sup>.

355 Filtering tasks (FT1–FT5) showed the highest completion rates  
 356 across all participants, indicating that participants correctly under-  
 357 stood the essential functioning of the tool. Participants were able  
 358 to navigate the interface intuitively, with minimal cognitive load.  
 359 The completion times ranged from approximately 2min to 10min,  
 360 reinforcing the tool's efficiency in handling exploratory tasks. Al-  
 361 though overall performance was high, two isolated errors were  
 362 identified. In FT1, participants #P4 and #P7 had difficulty in recog-  
 363 nizing all classification approaches supported by the tool. These  
 364 misunderstandings indicate interpretive nomenclature issues in the  
 365 tool, since both participants mixed the names of the approaches  
 366 with the names of the supported extraction types. Participant #P4  
 367 mentioned: “Well, I think I should look here under Language Models.  
 368 And inside Language Models, I need to check the supported extraction  
 369 and smell type...”, a comment that illustrates that the participant  
 370 knew the tool supported the Language Model approach, but she  
 371 was uncertain about what, in fact, meant an approach.

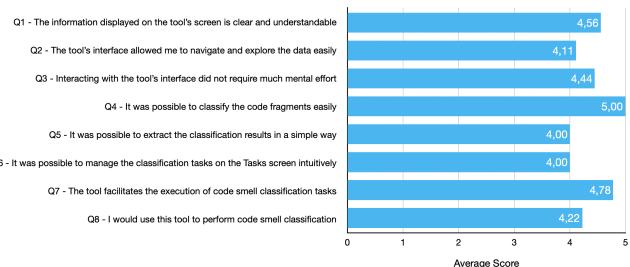
372 Basic tasks (BT1–BT6) also achieved strong results, with par-  
 373 ticipants successfully executing classifications using all analytical  
 374 approaches—AST, ML, and LMs—and exporting results without  
 375 major difficulty. Participants #P1 and #P4 experienced difficulty  
 376 understanding precisely what actions needed to be performed for  
 377 the task BT1, resulting in responses that were inconsistent with the  
 378 task's objectives. In BT5, participants #P3–#P5 and #P7 struggled to  
 379 correctly identify the exported files, compromising the accuracy of  
 380 the expected answer. Completion times ranged between 10min and  
 381 15min, except for participant #P9, who took nearly 21min, reinforc-  
 382 ing that task execution requires attention but remains manageable.  
 383 Note that in these tasks, there is also the inherent processing time  
 384 associated with model inference and task scheduling, and despite  
 385 that, these outcomes indicate that Scylla enables users to perform  
 386 the core code-classification operations efficiently.

387 Assimilation tasks (AT1–AT3) represented the most complex  
 388 tasks, requiring comparative reasoning between approaches and  
 389 interpretation of model explainability outputs. Although partici-  
 390 pants completed most tasks successfully, most of them committed  
 391 at least one error. These suggest opportunities to enhance the way  
 392 comparative analyses are presented to users, particularly regarding  
 393 the visual organization and clarity of information derived from  
 394 multiple classification approaches. For instance, the addition of  
 395 an analytical dashboard could significantly strengthen support for  
 396 comparative reasoning tasks, enabling users to explore differences  
 397 across approaches more intuitively and systematically.

## 398 5.2 Scylla's Usefulness and Applicability (RQ<sub>2</sub>)

399 To address RQ<sub>2</sub>, we analyzed participants' perceptions of Scylla's  
 400 usefulness and practical applicability through a post-session ques-  
 401 tionnaire. The aggregated scores are presented in Figure 2. Overall,  
 402 the responses revealed a positive evaluation, particularly regarding  
 403 the clarity of its interface and the intuitiveness of its workflows.

404  
 405 <sup>4</sup><https://github.com/aisepucrio/lm4smells-core/tree/main/docs/results>



415  
 416  
 417  
 418  
 419  
 420  
 421  
 422  
 423  
 424  
 425  
 426  
 427  
 428  
 429  
 430  
 431  
 432  
 433  
 434  
 435  
 436  
 437  
 438  
 439  
 440  
 441  
 442  
 443  
 444  
 445  
 446  
 447  
 448  
 449  
 450  
 451  
 452  
 453  
 454  
 455  
 456  
 457  
 458  
 459  
 460  
 461  
 462  
 463

Figure 2: Average participant scores for each question.

420 Questions Q1, Q2, and Q3, which assessed clarity, ease of nav-  
 421 igation, and cognitive effort, achieved strong ratings (4.56, 4.11,  
 422 and 4.44, respectively), indicating that participants found the  
 423 interface clear, consistent, and cognitively accessible. Similarly,  
 424 Q4 and Q7 about the classification process received the highest mean  
 425 score (5.00 and 4.78), suggesting that the classification process was  
 426 straightforward and well-integrated into Scylla's design.

427 Regarding operational aspects, Q5 and Q6—which evaluated  
 428 the simplicity of extracting results and managing classification  
 429 tasks—both obtained average scores of 4.00. In Q5, participants  
 430 reported difficulties related to the extraction and interpretation  
 431 of results. For instance, #P5 noted that “the extraction is not easy  
 432 due to the lack of information on the download screen; comparing  
 433 two approaches also requires manual work in external tools.” #P7  
 434 reinforced these concerns, commenting that “evaluating the result  
 435 requires data analysis, which can be complex when done quickly  
 436 through CSV”. For Q6, participants expressed issues related to task  
 437 identification and management. #P4 reported that “it was hard to  
 438 identify which file was which, since the file names did not clearly  
 439 differentiate the smell type”. Participants also mentioned difficulties  
 440 handling multiple files, as #P9 explained: “managing results from  
 441 different approaches is hard when relying only on task/file names”.  
 442 Finally, Q8, which measured the intention to use Scylla in real  
 443 scenarios, achieved a mean score of 4.22. Collectively, these findings  
 444 evidence the strong acceptance and perceived utility of Scylla  
 445 among participants, demonstrating high potential for its adoption.

## 446 6 Conclusion and Future Work

447 We presented Scylla, a tool that unifies four complementary ap-  
 448 proaches to code smell classification—AST-based heuristics, ML,  
 449 DL, and LMs—within a single, accessible environment. By enabling the  
 450 submission, classification, and export of results through a structured  
 451 workflow, Scylla supports reproducible and multifaceted analy-  
 452 ses of software quality. Our experiments showed that participants  
 453 successfully completed tasks of varying complexity, demonstrating  
 454 that Scylla provides a clear and intuitive interaction flow.

455 Future work will focus on two main directions. First, we plan  
 456 to incorporate analytical capabilities, enabling users to visualize,  
 457 compare, and interpret classification results directly within Scylla.  
 458 Second, we aim to extend Scylla to additional smells and pro-  
 459 gramming languages (C#, Go, JavaScript, and Java), broadening its  
 460 applicability across diverse development ecosystems. By advanc-  
 461 ing these capabilities, Scylla seeks to evolve into a comprehensive  
 462 and extensible environment for empirical software quality analysis,  
 463 supporting both research and practical code assessment workflows.

## References

- [1] Emran Aljarrah, Hamzah Elrehail, and Bashar Aababneh. 2016. E-voting in Jordan: Assessing readiness and developing a system. *Computers in Human Behavior* 63 (2016), 860–867. <https://doi.org/10.1016/j.chb.2016.05.076>
- [2] Muhammad Azeem, Muhammad Usman, and Aamer Nadeem. 2019. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology* (2019).
- [3] Aryan Boloori and Tushar Sharma. 2025. DPY: Code Smells Detection Tool for Python. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, 826–830.
- [4] Diego Castro and Marcelo Schots. 2018. Analysis of test log information through interactive visualizations. In *Proceedings of the 26th Conference on Program Comprehension*. Association for Computing Machinery, 156–166. <https://doi.org/10.1145/3196321.3196345>
- [5] Jinfeng Chen, Lin Tan, and Qiang Chen. 2016. PySmell: A Python code smell detection tool. In *Proceedings of the 24th International Conference on Software Analysis*.
- [6] Zhiwei Chen, Lin Chen, Wanwanging Ma, and Baowen Xu. 2016. Detecting Code Smells in Python Programs. In *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*, 18–23.
- [7] John Doe, Wei Zhang, and Anil Kumar. 2024. An Empirical Evaluation of Ensemble Models for Python Code Smell Detection. *Journal of Software Quality and Maintenance* 38, 2 (2024), 155–172.
- [8] DPY Developers. 2024. DPY — A Framework for Python Program Analysis. <https://github.com/python/ast>.
- [9] Eric Evans. 2003. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, Boston.
- [10] David Flanagan. 2020. *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language* (7 ed.). O'Reilly Media, Inc.
- [11] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. 2012. *Refactoring: Improving the Design of Existing Code*. Pearson Education. <https://books.google.com.br/books?id=HmHDHwgkbpCs>
- [12] Felipe Garcia, Xinyi Wang, and David Roberts. 2025. Prompt Learning for Multi-Label Code Smell Detection: A Promising Approach. *IEEE Transactions on Software Engineering* (2025), 1–12. <https://doi.org/10.1109/TSE.2025.1234567>
- [13] Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2023. Testing RESTful APIs: A Survey. *ACM Trans. Softw. Eng. Methodol.* 33, 1, Article 27 (Nov. 2023), 41 pages. <https://doi.org/10.1145/3617175>
- [14] Peter Hamfelt, Ricardo Britto, Lincoln Rocha, and Camilo Almendra. 2024. Automatic Identification of Machine Learning-Specific Code Smells. In *Proceedings of the 2024 International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 1–8.
- [15] Michele Lanza and Radu Marinescu. 2006. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. (2006).
- [16] Yifan Liu, Xinyu Yang, and Min Li. 2019. Deep learning based detection of design smells. *Journal of Systems and Software* (2019).
- [17] Mika V. Mäntylä and Casper Lassenius. 2004. Subjective evaluation of software evolvability using code smells: An empirical study. In *Proceedings of the 11th IEEE International Software Metrics Symposium*, 381–392. <https://doi.org/10.1109/METRIC.2004.1357922>
- [18] R.C. Martin. 2009. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall. <https://books.google.com.br/books?id=hjEFCAAAQBAJ>
- [19] M. Masse. 2011. *REST API Design Rulebook*. O'Reilly Media. <https://books.google.com.br/books?id=4lZesRwXo6MC>
- [20] Eric Meyer and Estelle Weyl. 2023. *CSS: The Definitive Guide: Web Layout and Presentation* (5 ed.). O'Reilly Media, Inc.
- [21] Regina O. Obe and Leo S. Hsu. 2017. *PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database* (3 ed.). O'Reilly Media.
- [22] Igor Oliveira, Joanne Carneiro, Jessica Ribas, and Juliana Pereira. 2025. Code Smell Classification in Python: Are Small Language Models Up to the Task?. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software* (Recife/PE). SBC, 699–705. <https://doi.org/10.5753/sbes.2025.11046>
- [23] Ollama. 2025. Ollama. <https://ollama.com/> Accessed on: July 2, 2025.
- [24] PyCQA. 2025. *Flake8: The modular source code checker for Python*. <https://github.com/PyCQA/flake8> Accessed: 2025-11-10.
- [25] PyCQA. 2025. *Pyflakes: A simple program which checks Python source files for errors*. <https://github.com/PyCQA/pyflakes> Accessed: 2025-11-10.
- [26] Gavin M. Roy. 2017. *RabbitMQ in Depth*. Manning Publications, Shelter Island, NY, USA.
- [27] Karthik Shivashankar and Antonio Martini. 2025. PyExamine: A Comprehensive, Un-Opinionated Smell Detection Tool for Python. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, IEEE, 763–774.
- [28] Laura Smith, Thiago Pereira, and Min Huang. 2024. EnseSmells: Deep Ensemble and Programming Language Models for Automated Code Smells Detection. *Empirical Software Engineering* (2024), 1–25.
- [29] Igor Soares de Oliveira, Joanne Carneiro, Jessica Ribas, and Juliana Alves Pereira. 2025. Code Smell Classification in Python: Are Small Language Models Up to the Task?. In *Brazilian Symposium on Software Engineering, Insightful Ideas and Emerging Results Track (SBES IIER)*. SOL, 1–7.
- [30] Raúl Tabarés. 2021. HTML5 and the evolution of HTML: tracing the origins of digital platforms. *Technology in Society* 65 (2021), 101529. <https://doi.org/10.1016/j.techsoc.2021.101529>
- [31] The Pylint Team. 2024. Pylint – Static Code Analysis for Python. <https://pylint.pycqa.org/>.