# Quality Assessment of ChatGPT Generated Code and their Use by Developers

Mohammed Latif Siddiq
msiddiq3@nd.edu
University of Notre Dame
Notre Dame, IN, USA

Lindsay Roney
lroney@nd.edu
University of Notre Dame
Notre Dame, IN, USA

Jiahao Zhang*
jzhang38@nd.edu
University of Notre Dame
Notre Dame, IN, USA

Joanna C. S. Santos
joannacss@nd.edu
University of Notre Dame
Notre Dame, IN, USA

## ABSTRACT

The release of large language models (LLMs) like ChatGPT has revolutionized software development. Numerous studies are exploring the generated response quality of ChatGPT, the effectiveness of different prompting techniques, and its performance in programming contests, among other aspects. However, there is limited information regarding the practical usage of ChatGPT by software developers. This data mining challenge focuses on DevGPT, a curated dataset of developer-ChatGPT conversations encompassing prompts with ChatGPT's responses, including code snippets. Our paper leverages this dataset to investigate (RQ1) whether ChatGPT generates Python & Java code with quality issues; (RQ2) whether ChatGPT-generated code is merged into a repository, and, if it does, to what extent developers change them; and (RQ3) what are the main use cases for ChatGPT besides code generation. We found that ChatGPT-generated code suffers from using undefined/unused variables and improper documentation. They are also suffering from improper resources and exception management-related security issues. Our results show that ChatGPT-generated codes are hardly merged, and they are significantly modified before merging. Instead, based on an analysis of developers' discussions and the developer-ChatGPT chats, we found that developers use this model for every stage of software development and leverage it to learn about new frameworks and development kits.

## CCS CONCEPTS

• **Software and its engineering → Software performance**; **Software usability**; *Empirical software validation.*

## KEYWORDS

datasets, chatgpt, security, quality, pull-request, open-coding

**ACM Reference Format:**

Mohammed Latif Siddiq, Lindsay Roney, Jiahao Zhang, and Joanna C. S. Santos. 2024. Quality Assessment of ChatGPT Generated Code and their Use by Developers. In *Proceedings of 21st International Conference on Mining Software Repositories, Mining Challenge Track (MSR'24).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

The release of GitHub Copilot [16] and ChatGPT [2], two code generation assistants based on Large Language Models (LLMs), is reshaping software development [27]. A recent survey [30] showed that 92% of 500 US-based developers are using LLM-based code generation tools for both work and personal use. Code generation models help developers automate repetitive tasks, focusing on higher-level challenging tasks [43].

Although these code generation assistants are increasingly popular among developers [43], prior studies have shown that they can generate code that contains quality issues, such as code smells, security smells, and vulnerabilities [25, 33]. A recent study [32] also showed that code generation models are fine-tuned with samples containing code/security smells that leak to the generated code. With the increasing use of LLM-based code assistants, these problematic code snippets can get deployed into production, negatively affecting the software system's security and reliability.

Although there are studies about code generation models [25, 32], there is no study about chat-style multimodel, *i.e.*, ChatGPT [2]. As developers can make conversations about software development that are not limited to traditional code generation, ChatGPT can provide answers to developers in every step of software development (*e.g.*, software management, testing, deployment *etc.*). Moreover,

*J. Zhang was a visiting student at ND from SUSTech (Southern University of Science and Technology) when this research was performed.


*MSR'24, April 2024, Lisbon, Portugal*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

there is no study about the ChatGPT-generated code from the perspective of code quality issues mined from real software developers' prompts.

Thus, in this data mining challenge, our work investigates the quality issues of ChatGPT-generated code and their influence on developers' discussions. We used the DevGPT dataset [42] to investigate the code and security smells in the generated Python and Java code using three static analyzers, Pylint [4], Bandit [1] and CodeQL [17]. We also investigated if the ChatGPT-generated Code merged into the code base. Furthermore, we did an open coding of the discussion and conversations in the pull requests where developers posted ChatGPT share links.

Our work provides the first view of quality issues in the code generated by ChatGPT and how the code is used among software developers. Our result shows that ChatGPT-generated codes have issues using undefined and unused variables and no proper documentation. They also suffer from improper resources, exceptions, and cryptography security management and use hard-coded credentials. Our result also shows that most ChatGPT-generated codes are not merged from the pull requests. Developers use the conversation to learn about libraries, frameworks, refactoring, and debugging code. The replication package can be found here: https://anonymous.4open.science/r/DevGPT-Study/.

## 2 BACKGROUND AND RELATED WORK

This section provides a high-level overview of code generation and code quality issues and related papers.

### 2.1 LLM-based Code Generation

LLM-based code generation techniques produce source code from av given *prompt*. A prompt can be a combination of natural language and source code. The code generation problem can be treated as a *sequence-to-sequence (seq2seq) learning* problem [35] and prior works used Recurrent Neural Networks (RNN) and neural networks based on Long Short-Term Memory (LSTM) [31, 35] to generate source code. More recently, the attention-based transformer architecture revolutionized the field of language learning [40]. There are several transformer-based deep learning models that are fine-tuned with code-related datasets for source code generation [18, 20, 36], search [11], and summarization [14]. Examples of this type of model include CodeBERT [11], CodeT5 [41], and Codex [8]. Codex models are powering GitHub Copilot [16], which are descendants of GPT-3 model [6]. ChatGPT is optimized for conversation and also can produce source code.

### 2.2 Code Smell

A **code smell** ("bad code smell" or "smell") is an indicator of an improper choice of system design and implementation strategy [12, 13]. These flaws can stifle software development or increase the chance of future errors or failures [26]. An example of a smell taken from Siddiq *et al.* [32] is *using the wrong exception catching order*, as shown in Figure 1. The `TypeError` block is never reached, as the `Exception` block will catch ***all*** exceptions.

**Security code smells** (or simply "security smells") are a subset of code smells. They are common code patterns that may lead to security flaws [28, 29, 32]. Although security smells may not be a vulnerability per se, they are symptoms that signal the prospect of a vulnerability [15]. For example, the code snippet in Figure 1 has a security smell related to the *use of hard-coded credentials* (CWE-798) [32]. This code snippet checks the password to a hard-coded string (*i.e.*, `"passw0rd!"`), which can be exploited using a list of commonly used passwords.

Code smell example

```
try:
    age = int(input())
except Exception: raise
except TypeError: raise
```

Security smell example

```
def verifyAdmin(password):
  if password != "passw0rd!":
    return False
  return True
```

**Figure 1: Examples of a code smell and a security smell**

Code and security smells introduce quality issues and negatively affect a project's maintainability, readability, and security [13]. As manually identifying smells is time-consuming and may not scale [21], several works [7, 19, 24] developed techniques to detect code smells. Lanza and Marinescu [21] proposed a metric-based detection strategy to detect code smells, while Chen *et al.* [9] implemented a code smell detection tool, Pysmell, which can detect **11** code smells in a Python project. Di Nucci *et al.* [10] described machine learning-based experiments with a new dataset configuration that includes instances of multiple types of smells.

### 2.3 Empirical Studies on Code LLMs

Prior works investigated the usability of code generation models [38], whether they can generate vulnerable code [25] or whether it is as bad as humans in generating insecure code [5]. Siddiq *et al.* [32] focused on quality issues in the code generation datasets and their leakage from the closed and open source models. In other projects, they created a dataset and framework to systematically evaluate generated code from the security perspective [33, 34]. Liu *et al.* [22] focused on the quality issues of ChatGPT-generated code, but they are limited to programming problems from LeetCode [3]. Unlike these prior studies, we study *real* prompts made by software developers who used ChatGPT to generate code.

## 3 METHODOLOGY

In this study, we answer three research questions:

**RQ1** ***Does ChatGPT generate code with smells?***

A recent study [32] showed that datasets used to train *open-source* LLMs contain code and security smells that leak to the output generated by these models. In this RQ, we investigate whether ChatGPT, a closed-source LLM, also generates code with quality issues, *i.e.*, containing code smells and security smells.

**RQ2** ***Are ChatGPT-generated source codes merged into open source projects?***

In this RQ, we investigate whether developers reuse code generated by ChatGPT in merged pull requests. In doing so, we also inspect what type of changes (if any) developers make prior to merging the generated code into production.

**RQ3** ***How are developers using ChatGPT?***

In this question, we aim to understand the use cases for ChatGPT beyond code generation. We investigate how developers use ChatGPT by analyzing the chat conversations.

To answer these RQs, we use the DevGPT dataset [42]. This dataset contains ChatGPT *sharing links* found in posts made on HackerNews as well as in code files, commits, issues, pull requests, and discussions posted on GitHub. This dataset has **2,345** share links. Each shared link corresponds to a chat between a developer and ChatGPT. This chat conversation can include not only natural language but also code written in different programming languages. In this study, we focus on the generation of code written in Python and Java as these are two popular languages among developers [23]. The next sections describe how we answer each of these RQs.

## 3.1 RQ1: Smells in Generated Code

To answer RQ1, we retrieved all share links mined from GitHub, *i.e.*, links found in files, commits, issues, pull requests (PRs), or PRs' discussions. Next, we excluded conversations that did not include code written in Python or Java. This way, we obtained 696 Python files from 357 ChatGPT conversations. Since 52 of these Python files (7.47%) had syntax errors, in the end, we had **644** ChatGPT-generated Python codes. In the case of Java, we initially obtained 147 Java files from 69 ChatGPT conversations. We converted them into Maven projects, named the files according to the public class' name, and added dependencies as needed in order to make the code compilable. Yet, not all the generated Java codes could be compiled as some used classes that were not available in the conversation. Once we discarded uncompilable samples, in the end, we had a total of **67** Java source codes from **35** ChatGPT conversations. To identify quality issues, we ran **Pylint** (v3.0.2) [4] and **Bandit** (v1.7.5) [1] on the Python files and **CodeQL** (v2.14.6) [17] for the Java files.

***Code Smell Analyzers.***

- **Pylint** [4] is a static analyzer for Python that can display five message types: *fatal* (*i.e.*, unable to process the file), *error* (*i.e.*, code smells that may lead to runtime errors), *warning* (*i.e.*, Python-specific smells), *convention* (*i.e.*, coding standard violations), and *refactor* (*i.e.*, code smells that can be fixed through refactoring). Similar to a prior study [32], we ignored messages related to style issues about whitespaces, newlines, and invalid names[1] as well as import-related messages[2] as Pylint is not reliable in checking for `import` statements' usage [39]. We also omitted *fatal* messages.
- **Bandit** [1] is a static analyzer that detects *security smells* in Python code. Each detected security smell maps to an ID from the Common Weakness Enumeration (CWE ID) [37].
- **CodeQL** [17] is a static analyzer that checks for style issues and vulnerabilities in a project by executing QL queries against a database generated from the source code. We used CodeQL to check for issues related to the following categories: *Advisory*, *Architecture*, *Compatibility*, *Complexity*, *Dead Code*, *Language Abuse*, *Performance*, *Violations of Best Practices* and *Security*.

## 3.2 RQ2: Generated Code in Merged PRs

To answer RQ2, we leveraged the metadata data from pull requests (PRs) where ChatGPT share links were mentioned. We only included PRs whose status was "*merged*" and that had *at least one* modified/added Python or Java file, obtaining a total of **50** pull requests. We then manually inspect each of these PRs and compare them with the source file(s) in the pull request in order to identify the files that contain verbatim copies or similar code to the ones generated by ChatGPT. For each PR, we also note what changes (if any) were made by developers to the generated code before making the pull request.

## 3.3 RQ3: ChatGPT Use Cases

In RQ3, we analyzed the *developers' discussion* in 39 out of the 50 pull requests retrieved in RQ2, along with the *chat conversation* within the ChatGPT share link. We could not analyze 11 PR discussions as they were not in English or had no discussion texts. During this open coding, we analyzed the pull request's title, body, and each comment made by the developers where the ChatGPT conversation is mentioned (if it is not mentioned in the PR body). We analyzed the context in order to annotate them with concepts (codes). This coding was performed by one of the authors of this paper, who has a software development experience of 2 years. The open code was then vetted by two other authors with 3 and 10 years of experience each. After reviewing the pull request information, we collaboratively highlighted the key points (which are presented in Section 4.3).

# 4 RESULTS

This section describes our findings and answers our RQs.

## 4.1 RQ1: Generated Code Quality

Table 1 shows the quality issues found by Pylint, Bandit, and CodeQL to the Python/Java code generated by ChatGPT. We found that most messages for Python files are *error* and *convention* type. The top messages for each category are *undefined variables*, *missing docstrings*, *too few public methods in a generated class*, and *redefining a variable declared in the outer scope*. ChatGPT-generated code also suffers from security issues. Results from Bandit [1] show that there are 84 security issues and the top three of them are *requests without a timeout* (CWE-400: Uncontrolled Resource Consumption), *using pseudo-random generators* (CWE-330: Use of Insufficiently Random Values) and *using asserts* (CWE-703: Improper Check or Handling of Exceptional Conditions).

For Java code, the issues are about *not using proper JavaDoc* in public methods, constructors, or variables, *dead code*, *unused maven dependencies*, and *unused and unread local variables*. In terms of security smells, ChatGPT mainly provides hard-coded credentials in the code, which is related to *CWE-798: Use of Hard-coded Credentials*.

[1] *C0303-trailing-whitespace*, *C0304-missing-final-newline*, *C0305-trailing-newlines*, and *C0103-invalid-name*.

[2] *W0611-unused-import*, *W0401-wildcard-import*, *W0404-reimported*, *W0614-unused-wildcard-import*, *C0410-multiple-imports*, *C0411-wrong-import-order*, *C0412-ungrouped-imports*, *C0413-wrong-import-position*, *C0414-useless-import-alias*, *C0415-import-outside-toplevel*, *C2403-non-ascii-module-import*, *R0402-consider-using-from-import*, and *E0401-import-error*.

**Table 1: Code quality issues in ChatGPT-generated code**

| | Category | Total Msgs. | # Msgs. | Top-3 Message Types |
|---|---|---|---|---|
| Python | Convention | 1,146 | 638 | C0114: Missing module docstring |
| | | | 235 | C0116: Missing docstring |
| | | | 189 | C0301: Line too long |
| | Error | 718 | 701 | E0602: Undefined variable |
| | | | 4 | E1101: No member |
| | | | 4 | E0104: Return outside function |
| | Warning | 328 | 125 | W0621: Redefined outer name |
| | | | 42 | W0613: Unused argument |
| | | | 36 | W0612: Unused variable |
| | Refactor | 143 | 47 | R0903: Too few public methods |
| | | | 27 | R0801: Duplicate code |
| | | | 16 | R1705: No else return |
| | Security | 84 | 14 | B113: Request without timeout |
| | | | 19 | B311: Pseudo-random generators |
| | | | 19 | B101: Assert used |
| Java | Advisory | 122 | 87 | No Javadoc for public method |
| | | | 11 | Non-final immutable field |
| | Architecture | 8 | 8 | No Javadoc for public type |
| | | | 8 | Unused Maven dependency |
| | Dead Code | 36 | 19 | Dead method |
| | | | 9 | Dead class |
| | | | 5 | Dead field |
| | Violations | 42 | 17 | Auto boxing or unboxing |
| | | | 17 | Unused local variable |
| | | | 5 | Unread local variable |
| | Security | 2 | 1 | Hard-coded credential in API call |
| | | | 1 | Hard-coded credential in sensitive call |

## 4.2 RQ2: Generated Code in Merged PRs

Out of the 50 analyzed PRs containing ChatGPT share links, only **6** of them (12%) contained generated code that was merged into the repository. Out of these 6 merged codes, **3** of them were verbatim copies of the generated code, whereas the other **3** contained modifications. To make the modification, developers changed the code to make it compatible with their code base. For example, the following code has been generated by ChatGPT, which uses Jira rest API to check permission to create an issue.

ChatGPT-Generated code

```
if response.status_code == 200:
    permissions = response.json().get('permissions', {})
    create_issue_permission =
        permissions.get('CREATE_ISSUES', {}).get('havePermission', False)
```

The following code, taken from a merged pull request[3], uses the example above and is modified for the same purpose.

Code from a merger PR

```
def can_i(self, permission: str) -> bool:
    return bool(
        self.jira.my_permissions(projectKey=self.project)["permissions"][
            permission]["havePermission"]
    )

def can_create_issues(self) -> bool:
    return self.can_i("CREATE_ISSUES")
```

## 4.3 RQ3: ChatGPT Use Cases

After doing an open coding of PRs' discussions and the corresponding conversations with ChatGPT, we observed the following use cases for ChatGPT:

- **Libraries & Frameworks**: We found that ChatGPT was used in 36% of cases to get help with the APIs for **(1)** standard (built-in) Python modules/ Java packages and **(2)** external Python modules / Java APIs. For example, we observed conversations in which developers sought help with Python's string manipulation methods, datetime module, and deepcopy of objects. ChatGPT's conversations were also used for understanding public libraries, especially for machine and deep learning frameworks, *e.g.*, Pytorch, TensorFlow, pandas, skrub, pydantic, SparkAI *etc.*
- **Code Formatting and Refactoring**: We found 6 (15%) chats in which developers used ChatGPT for code formatting and standardization. For example, ChatGPT provided suggestions to a developer on how to re-structure a developer's Python script into multiple files and folders. In another use case, we found that developers used ChatGPT to obtain advice on how to automatically check type hints in a Python project, to which case ChatGPT suggested using the library mypy.
- **Testing and Deploying**: We found 9 chats (23%) in which ChatGPT provided suggestions to developers related to version controlling, unit & integration testing, dependency management and documentation. Developers also use it to obtain help for deployment frameworks like Jira.
- **SDK Help**: In 4 of the analyzed conversations, ChatGPT was used to understand and troubleshoot software development kits (SDKs). The most commonly mentioned SDKs were Nylas-SDK, AWS SDK for Python (Boto3), and konfuzio-SDK.
- **Debugging**: Although ChatGPT prompts are limited to 4,000 tokens, we found 3 conversations in which developers used ChatGPT to debug file uploading issues, audio conversation problems, and arbitrary code execution.
- **Miscellaneous**: In 8 cases, ChatGPT was also used in miscellaneous use cases such as networking, bioinformatics-related data manipulation, message exchange, and audio streaming.

# 5 DISCUSSION

***Quality Issues in the Generated Code.*** Our findings in RQ1 demonstrated that ChatGPT-generated codes have quality issues like not following standard coding practice, need refactoring, and security smells. The generated codes use undefined variables and return data outside of the functions. They are also poorly documented and keep undefined variables and arguments in the method. ChatGPT-generated code also faced security issues like not using timeout in resource-sensitive API calling, providing `assert` without proper exception management, and using hardcoded credentials.

***Implication for the Developers.*** Developers are using ChatGPT for various purposes, as our findings in RQ3. They can be used to learn about unfamiliar libraries and development kits. Developers make conversations about code formatting, testing, and deployment. They also can be used for code debugging. However, our results in RQ1 show they are not free from code and security smells. They should be appropriately vetted before using the code base.

## 5.1 Threats of Validity

In our work, the dataset is used for a mining challenge, DevGPT [42], which can introduce external threats to validity. However, the dataset is vetted by the organizers. Our work focused on Python and Java, two of the top-used programming languages among developers, according to a survey from Stack Overflow [23].

[3]https://github.com/app-sre/qontract-reconcile/pull/3630/files

Another validity threat to this work is that we used three static analyzers (Pylint [4], Bandit [1], and CodeQL [17]) to detect code quality issues. They can introduce false positives, but their precision is significantly high according to the literature [25, 32]. Another threat to this work concerns the manual evaluation of the PRs' discussions and chats. The author's professional software development experience and the peer review by the other senior authors mitigated this internal validity threat.

## 6 CONCLUSION AND FUTURE WORK

ChatGPT is revitalizing the software development process. Our work shows developers use them for code formatting, debugging, testing, and deployment. They are using it for learning new frameworks. However, the generated code from ChatGPT can be substandard and have security smells. In addition to that, they can be merged into the software code repositories. In the future, we are going to extend our work to other programming languages and investigate how the developers are making conversation to reach the solution.

## REFERENCES


[1] 2023. *Bandit.* https://bandit.readthedocs.io/

[2] 2023. Chat completions. Accessed Mar 25, 2023. https://platform.openai.com/docs/guides/chat

[3] 2023. LeetCode. https://leetcode.com [Online; accessed 10. Oct. 2023].

[4] 2023. *Pylint.* https://pylint.pycqa.org/

[5] Owura Asare, Meiyappan Nagappan, and N Asokan. 2022. Is GitHub's Copilot as Bad As Humans at Introducing Vulnerabilities in Code? *arXiv preprint arXiv:2204.04741* (2022).

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.

[7] Frederico Luiz Caram, Bruno Rafael De Oliveira Rodrigues, Amadeu Silveira Campanelli, and Fernando Silva Parreiras. 2019. Machine learning techniques for code smells detection: a systematic mapping study. *International Journal of Software Engineering and Knowledge Engineering* 29, 02 (2019), 285–316.

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[9] Zhifei Chen, Lin Chen, Wanwangying Ma, and Baowen Xu. 2016. Detecting Code Smells in Python Programs. In *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*. 18–23. https://doi.org/10.1109/SATE.2016.10

[10] Dario Di Nucci, Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, and Andrea De Lucia. 2018. Detecting code smells using machine learning techniques: Are we there yet?. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 612–621. https://doi.org/10.1109/SANER.2018.8330266

[11] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[12] Martin Fowler. [n. d.]. CodeSmell. https://martinfowler.com/bliki/CodeSmell.html

[13] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, Boston, MA, USA.

[14] Yuexiu Gao and Chen Lyu. 2022. M2TS: Multi-Scale Multi-Modal Approach Based on Transformer for Source Code Summarization. In *Proc. of the 30th IEEE/ACM Intl. Conf. on Program Comprehension* (Virtual Event) *(ICPC '22)*. Association for Computing Machinery, New York, NY, USA, 24–35. https://doi.org/10.1145/3524610.3527907

[15] Mohammad Ghafari, Pascal Gadient, and Oscar Nierstrasz. 2017. Security smells in android. In *2017 IEEE 17th international working conference on source code analysis and manipulation (SCAM)*. IEEE, 121–130. https://doi.org/10.1109/SCAM.2017.24

[16] GitHub Inc. 2022. GitHub Copilot : Your AI pair programmer. https://copilot.github.com [Online; accessed 10. Oct. 2022].

[17] GitHub Inc. 2022. Use of a broken or weak cryptographic hashing algorithm on sensitive data. https://codeql.github.com/codeql-query-help/python/py-weak-sensitive-data-hashing/ [Online; accessed 30. Oct. 2022].

[18] Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. 2022. CodeFill: Multi-token Code Completion by Jointly Learning from Structure and Naming Sequences. In *44th Intl. Conf. on Software Engineering (ICSE)*.

[19] Wael Kessentini, Marouane Kessentini, Houari Sahraoui, Slim Bechikh, and Ali Ouni. 2014. A cooperative parallel search-based software engineering approach for code-smells detection. *IEEE Transactions on Software Engineering* 40, 9 (2014), 841–861.

[20] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. 2021. Code prediction by feeding trees to transformers. In *2021 IEEE/ACM 43rd Intl. Conf. on Software Engineering (ICSE)*. IEEE, 150–162.

[21] Michele Lanza and Radu Marinescu. 2007. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems.* Springer Science & Business Media.

[22] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D Le, and David Lo. 2023. Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. *arXiv preprint arXiv:2307.12596* (2023).

[23] Stack Overflow. 2023. Stack Overflow Devlopers Survey. https://survey.stackoverflow.co/2023/#most-popular-technologies-language-prof

[24] Thanis Paiva, Amanda Damasceno, Eduardo Figueiredo, and Cláudio Sant'Anna. 2017. On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development* 5, 1 (2017), 1–28.

[25] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In *2022 2022 IEEE Symposium on Security and Privacy (SP) (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 980–994. https://doi.org/10.1109/SP46214.2022.00057

[26] José Pereira dos Reis, Fernando Brito e Abreu, Glauco de Figueiredo Carneiro, and Craig Anslow. 2022. Code Smells Detection and Visualization: A Systematic Literature Review. *Archives of Computational Methods in Engineering* 29, 1 (Jan. 2022), 47–94. https://doi.org/10.1007/s11831-021-09566-x

[27] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2022. Do Users Write More Insecure Code with AI Assistants? *arXiv preprint arXiv:2211.03622* (2022).

[28] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The Seven Sins: Security Smells in Infrastructure as Code Scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, Montreal, QC, Canada, 164–175. https://doi.org/10.1109/ICSE.2019.00033

[29] Md Rayhanur Rahman, Akond Rahman, and Laurie Williams. 2019. Share, But be Aware: Security Smells in Python Gists. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 536–540. https://doi.org/10.1109/ICSME.2019.00087

[30] Inbal Shani. 2023. Survey reveals AI's impact on the developer experience | The GitHub Blog. *GitHub Blog* (June 2023). https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/#methodology

[31] Alex Sherstinsky. 2020. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena* 404 (mar 2020), 132306. https://doi.org/10.1016/j.physd.2019.132306

[32] Mohammed Latif Siddiq, Shafayat H Majumder, Maisha R Mim, Sourov Jajodia, and Joanna CS Santos. 2022. An Empirical Study of Code Smells in Transformer-based Code Generation Techniques. In *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 71–82.

[33] Mohammed Latif Siddiq and Joanna Santos. 2023. Generate and Pray: Using SALLMS to Evaluate the Security of LLM Generated Code. *arXiv preprint arXiv:2311.00889* (2023).

[34] Mohammed Latif Siddiq and Joanna CS Santos. 2022. SecurityEval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques. In *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security*. 29–33.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks.

[36] Alexey Svyatkovskiy, Sebastian Lee, Anna Hadjitofi, Maik Riechert, Juliana Vicente Franco, and Miltiadis Allamanis. 2021. Fast and memory-efficient neural code completion. In *2021 IEEE/ACM 18th Intl. Conf. on Mining Software Repositories (MSR)*. IEEE, 329–340.

[37] The MITRE Corporation. 2022. CWE - Common Weakness Enumeration. http://cwe.mitre.org/

[38] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[39] Bart van Oort, Luís Cruz, Maurício Aniche, and Arie van Deursen. 2021. The Prevalence of Code Smells in Machine Learning projects. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 1–8.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.

[41] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[42] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*.

[43] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity Assessment of Neural Code Completion. In *Proc. of the 6th ACM SIGPLAN Int'l Symposium on Machine Programming* (San Diego, CA, USA) *(MAPS 2022)*. ACM, New York, NY, USA, 21–29. https://doi.org/10.1145/3520312.3534864