An Exploratory Study on LLM-Generated Code and Comments in Code Repositories

Yongyi Ji, Jiaji Wang, Yi Zhou, Fuxiang Chen*, Hongji Yang

School of Computing and Mathematical Sciences, University of Leicester, University Road, Leicester, LE1 7RH, United Kingdom

Abstract

The use of LLMs in software development has become increasingly widespread on tasks such as code generation and summarization. Reports from large technology companies showed that around 20% to 30% of their code are generated by LLMs. However, there remains skepticism about the practical usage of LLM-generated code and comments, such as concerns on more time for debugging the generated code and the unnaturalness of the generated comments. In this paper, we study the code and comments detected as likely to be generated by LLMs and their characteristics, the differences between company- and community-maintained repositories, and how likely bugs are associated with LLM-generated code.We conduct extensive experiments on active company- and community-maintained repositories from 2021 to 2025 using various tools and techniques that detect code and comments generated by LLMs. Based on our detector-based proxy analysis, the results suggest that code detected as likely to be generated by LLMs decreased over time and appeared frequently in test cases, while that of comments remains relatively stable. Proxy results further suggest that code detected as likely to be generated by LLMs shows substantial intra-repository code clones, whereas comments exhibit a relatively low proportion of grammatically correct sentences. In addition, the company-maintained repositories show a higher percentage of code and comments detected as likely to be generated by LLMs, and only a small percentage of the human-labelled bugs are detected as being likely associated with LLM-generated code.

Keywords: LLM-generated detection on Code and Comments, Empirical Study, Bug analysis, Code Clone Detection

1. Introduction

The rapid advancement of Large Language Models (LLMs) has resulted in multiple adaptation of LLMs in various downstream tasks [1]. In Software Engineering (SE), it was reported that some developers are already using LLMs in code generation and summarization, the two most important activities in software development and comprehension [2, 3, 4, 5, 6, 7, 8]. Moreover, the JetBrains Developer Ecosystem Survey [9] found that developers use LLMs to generate code and comments frequently. There have also been reports suggesting that the use of LLMs is gaining popularity among software companies. For example, in 2024, Google stated that 25% of the company's new code is generated by LLMs [10]. Similarly, in 2025, Microsoft stated that 20% to 30% of the code within the company's repositories is written by LLMs [11]. In addition, the 2025 Google's DORA report [12], which surveyed nearly 5,000 professional developers from around the world, found that 90% of the surveyed developers reportedly use some elements of LLMs at work, and over 80% of them believed this had improved their productivity.

On the other hand, there is reported skepticism on the accuracy of LLM-generated content. For code, Pearce et al. [13] found that when Github Copilot [14] is prompted to generate code in security-related scenarios, 40% of the code generated by Github Copilot was vulnerable. In the paper, the Github Copilot refers to the Copilot AI service hosted by Github. This is not to be mistaken for the Copilot model developed by Microsoft [15]. Liang et al. [16] surveyed 410 developers and found that the primary reason for not using LLMs was that the generated code did not meet their requirements. For comments, Sergeyuk et al. [17] collected opinions from 481 developers and found that one of the top reasons for not using comments generated by LLMs is that the generated comments are often unnatural and that they do not match the required tone and clarity of a human developer. Moreover, the code and comments that were generated by LLMs raised the uncertainty on whether a generated software artifact is created by an individual or a machine [18, 19].

In light of the controversial reports on the use of LLMs by developers, there are multiple unknowns: 1) What proportion of the code and comments in developers' repositories are likely to be generated by LLMs? 2) What characteristics do these code and comments possess? 3) Are there any differences between company- and community-maintained repositories on these code and comments? 4) Are there bugs in repositories that were associated with LLM-generated code?By answering these questions, we can better establish the gaps between developers' LLMs usages and the current state of LLMs. Previous work have proposed different detectors to detect text that is likely generated by LLMs [20, 21, 22]. Our work differs from previous research by applying existing detectors to repositories in order to quantify the proportion of code and comments that

*Corresponding author: Fuxiang Chen (email: fuxiang.chen@leicester.ac.uk)

are likely to be generated by LLMs and to analyze their characteristics, rather than proposing a new detector. In this study, we attempt to answer these questions by focusing on detecting code and comments that are likely to be generated by LLMs in the developers' repositories as writing code and comments are the two most important activities in software development as mentioned earlier. From this point onwards, we use likely to be LLM-generated to refer to code or comments that are likely to be generated by LLMs. We emphasize that all our findings are proxy-based observations derived from detector outputs, given the absence of ground-truth labels.

In our experiments, we apply existing LLM-generated content detectors, including Binoculars [23], Log-Likelihood [24], Entropy [25], Rank [20], Log-Rank [20], LRR [26], DetectGPT [21], Fast-DetectGPT [22] and DetectCodeGPT [18], to analyze developers' use of LLMs on code generation and summarization in their repositories. These two tasks correspond to generating code and comments, respectively. Throughout the paper, we use the term *detectors* to refer to LLM-generated content detectors. We like to stress that unlike previous studies that focus on proposing new detectors such as DetectCodeGPT, our focus is different: we investigate the proportion of code and comments that are likely to be generated by LLMs in the repositories, their characteristics, the differences between company- and community-maintained repositories as well as the bugs associated with LLM-generated code. Based on the inclusion and exclusion criteria, we selected 8 company- and community-maintained repositories from 2021 to 2025: Go-github, Guava, Liquid, Zap, Act, Jadx, Kafka and Pandas. We selected the repositories from 2021 because several LLMs, such as GPT-Neo [27], Codex [8] and Github Copilot [14], were released or open-sourced in 2021. One of the major challenges in performing the detection is the absence of ground truth data (human-labelled LLMs generated content). Since detectors rely on thresholds to classify inputs as likely to be generated by LLMs, it is challenging to determine optimal thresholds without human-labelled data, as these thresholds vary depending on the dataset used. For comments, while previous benchmarks like DetectRL benchmark [28] offer thresholds for general domains (e.g., Yelp reviews on businesses, human-written stories, etc), these may not accurately reflect the unique linguistic and structural patterns of comments, given that optimal thresholds vary across domains. To ensure that the thresholds are suitable for comments, we utilized the AISE dataset [29], which consists of original repository comments and LLM-generated comments, to derive optimal thresholds for our analysis. Since we cannot guarantee that the original comments were written by humans, we further filtered the AISE dataset to include only comments from files last modified before 2021. For code, we applied the default threshold from DetectCodeGPT as it was evaluated on developers' repositories, which is similar to our study. After using detectors to find code and comments that were likely to be LLM-generated, we performed a coding process to categorize them. We note that a previous study has found that LLM-generated code contains code clones [30]. Thus, we studied the characteristics of these code by analyzing code clones within repositories, across repositories, and in the GPTCloneBench [31]. We also examined whether comments detected as likely to be LLM-generated exhibit similar properties (e.g., frequent use of AI-related vocabulary such as *aspect* and *capturing*, and having high grammatical accuracy) as summarized in previous work on LLM-based textual output [32]. To analyze whether bugs in repositories were associated with LLM-generated code, we used the PreciseBugs dataset [33]. The PreciseBugs dataset was selected because it contains human-labelled bugs introduced after 2021, and several LLMs were released or open-sourced in 2021. This is in contrast to other datasets, such as Defects4J [34] and InferredBugs [35] where the human-labelled bugs are before 2020 (the pre-LLM era), and they are unlikely to be LLM-generated. In this paper, we refer to the use of detectors to detect code and comments that are likely to be LLM-generated as a detector-based proxy analysis, since the analysis is derived from detector outputs rather than ground-truth annotations of actual LLMs' usage.

We open-sourced the scripts used in our experiment [1].

Based on detector-based proxy analysis, we observed several interesting phenomena: 1) In active repositories, there is a decreasing trend of code detected as likely to be LLM-generated, while comments detected as likely to be LLM-generated remain stable. Code detected as likely to be LLM-generated primarily appear in test cases, and comments detected as likely to be LLM-generated mainly in Explanation or Meta category; 2) For code detected as likely to be LLM-generated, majority of the repositories have greater than 70% of file-level code clones and for method-level code clones, only a small percentage was found, with the highest being 34.21%. When compared with GPTCloneBench, we found that most of the code detected as likely to be generated by LLMs had no code clones. For comments detected as likely to be generated by LLMs, there is a high proportion of proper punctuation, with an average of around 90%, which correlates to previous study [32] summarizing the properties of text classified as LLM-generated. However, these comments showed a relatively low percentage of grammatically correct sentences and limited usage of AI-related vocabulary; 3) Company-maintained repositories have a higher percentage of code and comments that are likely to be generated by LLMs, and a higher percentage of code clones found; 4) In code repositories, we found that only a small percentage of the human-labelled bugs (10.79% and 5.56% from NVD and OSS-Fuzz, respectively) from the PreciseBugs dataset are likely to be generated by LLMs.

Overall, this paper makes the following contributions:

- We conducted extensive experiments on multiple different repositories across 2021 to 2025 using various detectors to investigate if code and comments are likely to be generated by LLMs in the repositories under a detector-based proxy analysis framework. Moreover, we tested a set of thresholds for detecting comments that are likely to be generated by LLMs.
- We compared company- and community-maintained repositories and found that, based on our detector-based

[1] https://github.com/yongyiji/LLM-Generated

proxy analysis, the percentage of code that is likely to be generated by LLMs is higher in company-maintained repositories.

- Based on our detector-based proxy analysis, we analyzed the human-labelled bugs in the repositories and found that only a small percentage of them (10.79% and 5.56% from NVD and OSS-Fuzz, respectively) is likely associated with LLM-generated code.

The rest of this paper is organized as follows. Section 2 provides background for our paper and Section 3 introduces related work. Section 4 outlines research questions, data collection and experiment setup. Section 5 presents the results of our analysis. This is followed by a discussion in Section 6 and an analysis of threats and validity in Section 7. Finally, Section 8 concludes and outlines directions for future work.

2. Background

LLM-generated text is reported to exhibit common traits: it has higher average log probability [20], it tends to favor high-probability tokens, resulting in lower ranks [20], and it is more predictable and typically has lower entropy due to concentrated probability distributions [20]. There are various existing zero-shot methods for detecting LLM-generated content, which can mainly be categorized into three types: statistical-based methods, perturbation-based methods, and model-comparison-based methods [28, 36]. In this work, we used all the three methods in our experiments as described below.

Statistical-based methods detect LLM-generated text by analyzing token-level statistical features derived from a language model's probability distribution, such as token log probabilities, token ranks, or entropy. Token log probabilities calculate the average log probability of a text, token ranks refer to the rank of a token within the model's probability distribution for a given text and entropy measures the uncertainty of a model's token prediction distribution. As an advanced statistical approach, Fast-DetectGPT examines the conditional probability curvature of the log-probability [22]. Unlike previous methods that look at a single average value, Fast-DetectGPT analyzes the distribution of alternative token choices at each position.

Perturbation-based methods, such as DetectGPT and Detect-CodeGPT, identify LLM-generated text by perturbing the input text and analyzing changes in the model's output [21, 18]. The core idea of perturbation-based methods is that, when text is perturbed or slightly rewritten by alternative LLMs, LLM-generated text tends to be less robust than human-written text. For example, DetectGPT observes that LLM-generated text is highly sensitive to perturbations, resulting in larger drops in log probability compared to human-written text. This is because LLM-generated text is usually more predictable and is constrained by the patterns the model was trained on. Unlike statistical-based methods, which only calculate statistics for the original text, perturbation-based methods calculate the statistics for both original and its perturbed variants.

Model-comparison-based methods, such as Binoculars [23], detect LLM-generated text by comparing the outputs of multiple LLMs. Unlike perturbation-based methods, model-comparison-based methods do not rely on perturbing the input text. Instead, they leverage the observation that LLM-generated text tends to exhibit greater consistency across different models than human-written text.

3. Related Work

Studies on Developers' Use of LLMs

Previous studies have examined the practices and challenges of using LLMs for developers. Zhang et al. [37] studied discussions on Stack Overflow and GitHub and found that developers often express hesitation and face challenges when incorporating Github Copilot into their workflows. Similarly, Jaworski and Piotrkowski [38] surveyed 42 developers and found that most participants did not want to use Github Copilot. Liang et al. [16] conducted a survey of 410 developers. They found that the main reasons developers avoid using LLM-based tools are that these tools often fail to generate satisfactory code meeting functional or non-functional requirements, and that developers have difficulty controlling the tools to produce the desired output. At the same time, the study found that developers are motivated to use LLM-based tools because they help reduce keystrokes and can complete programming tasks more quickly.

There were also studies examining the impact of LLMs on developers' productivity and have reported varying results. Some studies have shown that using LLMs can enhance productivity. For example, Ziegler et al. [39] investigated the use of Github Copilot and found that it can offer useful suggestions that guide developers' progress. Peng et al. [40] conducted a controlled experiment to assess the impact of LLM-based tools on professional software development and found that Github Copilot had a statistically significant effect on developers' productivity. Moreover, Weisz et al. [2] analyzed the impact of using LLMs to assist developers through surveys at a large technology company. They found that while LLM-based tools could increase developers' productivity, these benefits were not experienced equally by all users. Kuttal et al. [41] compared human-human and human-agent pair programming and found that LLM-based agents can serve as effective pair programming partners, matching humans in productivity, code quality, and learning experience. On the contrary, some studies found that using LLMs does not always improve work quality. Imai [42] compared Github Copilot to a human pair programmer and found that although Github Copilot increased productivity, the quality of the code produced was lower than that of human pair programming.

Our work differs from previous research by analyzing the developers' repositories to check how likely the code and comments are generated by LLMs.

LLM-generated Text Detection

In recent years, researchers have been working on detecting LLM-generated text. There are mainly three categories of approaches for LLM-generated text detection: zero-shot methods, watermarking methods and supervised models.

Zero-shot methods are usually based on the discrepancy between the Log-likelihood and ranking information of human and LLM-generated texts. Gehrmann et al. proposed GLTR [20], a zero-shot detection method that identifies LLM-generated text by analyzing each token's rank and Log-likelihood within the probability distribution of a pretrained language model. DetectGPT [21] detect LLM-generated text by analyzing how small perturbations affect a language model's log-likelihood of a passage, leveraging the observation that generated text tends to occupy regions of negative curvature in the model's probability surface. Based on DetectGPT, Bao et al. proposed Fast-DetectGPT [22], which accelerates zero-shot detection by replacing perturbations with a more efficient sampling strategy. Due to their statistical nature, zero-shot methods generally tend to achieve higher detection accuracy on longer passages [25, 26]. Shi et al. proposed DetectCodeGPT [18] which extends the framework of DetectGPT by perturbing stylistic tokens that capture the distinctive patterns between LLM-generated and human-written code, rather than perturbing arbitrary tokens. DetectCodeGPT is model agnostic and can detect code generated by various LLMs, as it does not rely on training or fine-tuning on any model but instead analyzes the differences between human-written and LLM-generated code [18].

Watermarking methods embed token-level markers within generated text, which are invisible to humans, enabling reliable detection of LLM-generated content. These approaches explore the potential of incorporating watermarks into language models to make LLM-generated texts easier to identify. Kirchenbauer et al. [43] proposed a statistical watermarking method that divides the vocabulary into green and red token lists based on hash values of preceding n-grams, and softly increases the logits of green tokens during generation to embed a watermark. Based on the work of Kirchenbauer et al., Zhao [44] used a fixed green-red split to propose a more robust watermarking method. However, watermarking approaches depend on model owners, such as OpenAI, to embed the watermark, which limits its broader utilization [45]. Moreover, Singh and Zou [46] found that watermarking method affects text quality, especially in reducing the coherence and depth of the generated responses.

Supervised models are trained on human-labelled datasets to distinguish human-written from LLM-generated text [47]. Previous studies have fine-tuned pretrained models to detect synthetic text across various domains, including peer review corpora [48] and news [49, 50]. For example, Fagni et al. [51] trained several supervised models to classify LLM-generated content on social media platforms. Similarly, Guo et al. [52] fine-tuned a RoBERTa-based classifier to distinguish between human-written text and ChatGPT-generated text. In the context of code, Nguyen et al. [53] proposed GPTSniffer, which fine-tunes CodeBERT to detect LLM-generated code snippets. However, supervised models trained to detect LLM-generated content may overfit to their training data [49].

We do not consider watermarking methods or supervised classifiers in our analysis because they require training of human-labelled datasets, which we do not possess. Moreover, although existing detectors such as DetectCodeGPT have been used to detect code in repositories, they primarily focused on proposing a detector. Contrary to that, our focus is on analyzing the proportion of code and comments that are likely to be generated by LLMs and their characteristics, comparing company- and community-maintained repositories and identifying the bugs likely associated with LLM-generated code.

Bugs in LLMs-generated code

Previous studies have found that code generated by LLMs contains bugs. Fan et al. [54] analyzed bugs in Codex-generated code and found that such code shares common mistakes with human-written code and exhibits several negative symptoms, including names that indicate incorrect algorithms, duplicated or similar code blocks, and irrelevant helper functions. Dakhel et al. [55] analyzed the quality of code generated by GitHub Copilot and found that some of the generated code contains bugs and is non-reproducible. Liu et al. [56] studied the quality of code generated by ChatGPT and summarized the common issues in ChatGPT-generated code. Recently, Tambon et al. [57] conducted an empirical study on bugs in code generated by LLMs and summarized 10 bug patterns that differ from bugs in human-written code. While the findings of these studies show that code generated by LLMs may contain different kinds of bugs, none of them have examined how many and what bugs are likely to be LLM-generated in repositories. To the best of our knowledge, our study is the first to analyze the proportion and category of bugs in repositories that are likely to be LLM-generated.

4. Research Design

4.1. Research questions

The research questions are described as follow:

- RQ1: How does the proportion of LLM-generated code and comments in repositories, as identified by the detectors, change over time?

  According to Google's DORA report [12], 90% of the surveyed developers reportedly use LLMs in their work. However, there is skepticism on the usages of LLMs due to concerns that LLMs generate non-functional code and the unnaturalness of the generated comments [16, 17]. This controversy motivated us to investigate whether the repositories contain LLM-generated content, and how the proportion of LLM-generated content changes over time. Moreover, we aim to detect code and comments that are likely to be LLM-generated, given the widespread adoption of LLMs techniques in SE tasks such as code generation and summarization. For each repository, our analysis covers the period from October 2021 to October 2025 because several LLMs, such as GPT-Neo, Codex and Github Copilot, were released or open-sourced in 2021. Starting from this time, developers could use LLMs to generate code and comments. We used the zero-shot based detectors (Binoculars, Log-Likelihood, Entropy, Rank, Log-Rank, LRR, DetectGPT, Fast-DetectGPT and DetectCodeGPT) to detect code and comments that are likely to

be generated by LLMs as these detectors analyzed the differences between LLM-generated and human-written content in a LLM agnostic manner. We examined yearly snapshots corresponding to the state of the repository each October. By using detectors to detect content that is likely generated by LLMs at these time points, we can analyze potential LLM usage evolves within their real-world work. Moreover, we conducted a coding process to categorize the code and comments that were detected as likely to be LLM-generated.

- RQ2: What are the characteristics of LLM-generated content detected by detectors in repositories?

  Previous studies have analyzed the characteristics of LLM-generated content. For natural language text, Russell et al. [32] hired annotators to classify non-fiction English articles into either human-written or LLM-generated, and summarized properties of LLM-generated content to distinguish between LLM-generated and human-written texts. They also reported the frequency of these properties based on annotators' explanations. We follow their summarized properties to analyze whether the comments detected as likely to be LLM-generated exhibit similar properties. In our experiment, since some properties such as originality are too subjective to be evaluated, we only considered the properties that can be automatically detected, including vocabulary, grammar, punctuation, and spelling.

  For code, Wu et al. [30] found that LLM-generated code can contain code clones, and commercial AI code generators produce Type-1 and Type-2 code clones. Because we lack human-labelled ground truth for LLM-generated code, we examine whether code detected as likely to be LLM-generated exhibits such clone patterns. To explore this, we utilized CCFinderX [58], a token-based code clone detector, to analyze whether the code detected as likely to be LLM-generated by DetectCodeGPT contain code clones. Moreover, we studied code detected as likely to be LLM-generated have code clone in GPTCloneBench [31] which contains code generated by GPT models. This allows us to find whether code detected as likely to be LLM-generated has code clones with known GPT code.

- RQ3: How does LLMs usage detected by the detectors differ between community-maintained and company-maintained repositories?

  According to statements from large technology companies, LLM-generated content can increase development efficiency, with approximately 20% to 30% of code being generated by LLMs in their repositories [10, 11]. In repositories not maintained by large technology companies, it remains unclear whether the proportion of LLM-generated code is similar. To analyze whether there are differences between company-maintained and community-maintained repositories, we collected data from both sources. We compare whether there is a difference between the two types of repositories in the proportion of code and comments detected as likely to be LLM-generated, as well as in the characteristics of code and comment detected as likely to be LLM-generated.

- RQ4: How likely are bugs associated with LLM-generated code?

  Liang et al. [16] found that developers are unwilling to use LLMs not only because the LLM-generated code does not meet their requirements and it is difficult to control the code generation tools, but also because developers need to spend too much time debugging the code produced by LLMs. It remains unknown whether bugs are associated with code detected as likely to be LLM-generated. We utilized bugs collected by Ye et al. [33] and applied DetectCodeGPT to analyze which of these human-labelled bugs were likely to be generated by LLM.

4.2. Project Selection

To ensure the reliability and analytical relevance of our dataset, we established a set of inclusion and exclusion criteria for selecting software repositories from GitHub. The inclusion criteria are defined as follows:

Inclusion Criteria:

- Active software development repositories. The repository must contain source code files and show at least one commit or merged pull request every month over the past year. This ensures that the project remains active and reflects current software development practices.

- Dominant use of mainstream programming languages. The primary programming language of the repository must be supported by our LLMs-based detection framework including Java, Python, Go, Ruby, Javascript and PHP. This requirement guarantees compatibility with the tools used in our analysis.

- Presence of analyzable textual content. The repository must include a sufficient amount of code comments or docstrings, enabling meaningful LLMs-based content analysis and interpretation.

- Diverse development entities. The collected repositories should include projects maintained by community developers as well as by companies, ensuring that the dataset is representative and reflects the practices and styles of different types of development entities.

Exclusion Criteria:

- Automatically generated or template-based repositories. Repositories whose content is primarily produced by non-LLM tools or explicitly labelled as "boilerplate," "template," or "generated" in their names or descriptions are excluded.

- Forked repositories. To avoid redundancy, only original repositories are analyzed. Exceptions are made if a forked repository demonstrates significantly higher activity than its source project.

- Non-software repositories. Repositories that primarily contain datasets, configuration files, or textual materials with minimal or no source code are excluded from the analysis.

After applying these criteria, we obtained a refined set of active and analyzable software repositories that align with the goals of our study. The selected repositories cover a wide range of application types (foundational libraries, developer tools, distributed systems, data processing frameworks and template engines) and are written in different programming languages (Python, Java, Go, and Ruby).

4.3. Data Collection

According to the inclusion and exclusion criteria, we selected eight repositories in GitHub [59], four company-maintained repositories and four community-maintained projects repositories. For the selected repositories, we fixed our analysis on a specific version snapshot, recording the commit hash, timestamp, and license. The time period spans from October 2021 to October 2025, with data collected yearly. We analyzed the repository starting from 2021 because major breakthroughs in LLMs occurred in 2021 [60] – for example, the release of Codex [61] and the launch of GitHub Copilot [14] enabled developers to generate code and comments using LLMs [60]. We employed zero-shot detectors to detect code and comments that are likely to be generated by LLMs, without restriction to any specific LLM [21, 18].

For all repositories, we applied the same data preprocessing procedure to extract the file content and we classified them into two categories: code, and comment. Since LLMs have been widely used in the field of SE for code generation and code summarization [3, 4, 6, 7, 8], we focus on detecting LLM-generated code and comments. We analyzed source files that DetectCodeGPT can analyze, including those written in Java, JavaScript, Python, Go, Ruby, and PHP.

4.4. Experiment Setup

All experiments are conducted on 10 NVIDIA A100 GPUs with 40GB memory. For the hyperparameters used in DetectCodeGPT, we followed previous work [18] by setting the span length to 2 and masking 50% of the words during text perturbation. The perturbation type was random-insert-space+newline. For the hyperparameters used in Binoculars, Log-Likelihood, Entropy, Rank, Log-Rank, LRR, DetectGPT and Fast-DetectGPT, we used the same settings as those used in DetectRL [28].

Figure 1 shows the overview of the workflow in this study. We apply detectors to identify code and comments that are likely to be generated by LLMs on both company- and community-maintained repositories. The outputs from these detectors are then used to address the four research questions.

- RQ1: How does the proportion of LLM-generated code and comments in repositories, as identified by the detectors, change over time?

In the experiment, we employed different zero-shot detectors, including Binoculars, Log-Likelihood, Entropy, Rank, Log-Rank, LRR, DetectGPT, Fast-DetectGPT and DetectCodeGPT. To detect comments written in natural language, we used Binoculars, Log-Likelihood, Entropy, Rank, Log-Rank, LRR, DetectGPT, and Fast-DetectGPT. For detecting code, we used DetectCodeGPT, which is specifically designed to identify whether code is LLM-generated or not.

For each detector, the optimal decision threshold depends on the dataset and domain. Existing non-code based detectors do not offer a predefined threshold to determine whether a given input is LLM-generated, as the optimal threshold varies across datasets and is typically derived using the Area Under the Receiver Operating Characteristic Curve (AUROC) [62]. AUROC metric is widely used for evaluating zero-shot methods [21], as it considers both true positive and false positive rates across different decision thresholds. However, in our experiment, there is no ground-truth labels for our dataset, as all data were collected from public repositories. Thus, we were unable to compute the AUROC metric and to find an optimal threshold for each repository. Moreover, Wu et al. [28] proposed DetectRL benchmark which evaluates a set of detectors, including Binoculars, Log-Likelihood, Entropy, Rank, Log-Rank, LRR, DetectGPT and Fast-DetectGPT. It performs different domain evaluation and assesses the generalization of detectors by applying thresholds obtained from one domain to other domains. The DetectRL provided four benchmark datasets: the arXiv dataset for scientific papers, the XSum dataset for news articles, WritingPrompts (WP) for creative stories, and Yelp Reviews for social reviews. However, optimal thresholds vary significantly across domains. The threshold set for each detector in DetectRL was shown in Table 1. Given that code comments possess unique structural and linguistic patterns distinct from news or social reviews, directly applying the non-SE thresholds provided by DetectRL could lead to inaccurate results. To ensure that the thresholds are suitable for the SE domain, we used the AISE dataset [29] to derive thresholds for code comments. The AISE dataset was proposed by Katzy et al. [29], who conducted an empirical study on LLM-generated code comments and created a dataset consisting of original comments and comments generated by LLMs. They first compiled a list of common words and used GitHub to collect files containing these words. For each file, comments were extracted, and CodeGemma, CodeLlama, CodeQwen, GraniteCode, and StarCoder were used to generate corresponding comments. Since it is unclear whether the original comments were written by humans or generated by LLMs, we filtered out files modified after 2021 to ensure that the retained comments were written by humans.

However, there is a difference in token length between the AISE dataset and the comments in our dataset, as shown in Table 2. While the AISE dataset primarily consists of

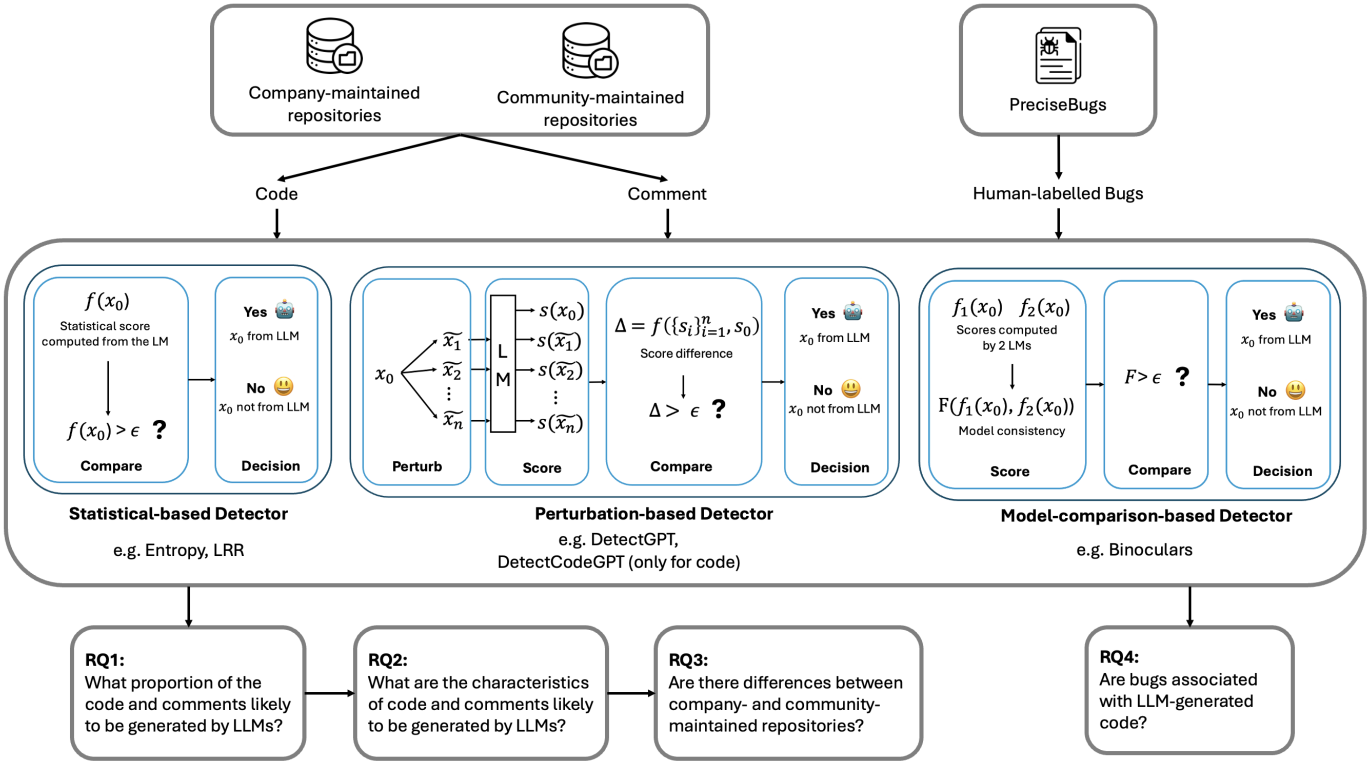


Figure 1: Overall framework of our analysis. The framework illustrates how detectors are applied to identify LLM-generated code and comments in repositories.

Table 1: Detector threshold across four datasets in DetectRL benchmark: ArXiv, XSum, WP, and Yelp.

Detector	ArXiv	XSum	WP	Yelp
Binoculars	-0.92	-0.92	-0.91	-0.93
Log-Likelihood	-2.24	-2.25	-2.64	-2.57
Entropy	2.96	2.95	3.47	3.79
Rank	-52.51	-17.72	-46.88	-39.83
Log-Rank	-1.12	-1.09	-1.28	-1.328
LRR	2.07	2.11	2.03	2.03
DetectGPT	1.03	0.94	0.54	0.50
Fast-DetectGPT	5.48	5.16	4.64	4.19

shorter comments, real-world repositories often contain much longer comments. Previous study has shown that detectors exhibit a performance degradation when processing short text [63]. Therefore, we filtered the AISE dataset to exclude comments containing fewer than 10 tokens and computed the AUROC metric to determine the threshold. The threshold for SE domain is shown in Table 3. Although detectors are primarily designed for general domains, comments are written in natural language. Therefore, zero-shot detectors developed for natural language are inherently applicable to the SE domain. For DetectCodeGPT, we used the dataset provided in DetectCodeGPT [18] to obtain the threshold, which was then applied in our experiment. The threshold was derived from open-source Github projects aross multiple programming languages. This is consistent with our experiment.

After using detectors to detect code and comments that are

Table 2: Distribution of comment lengths in the AISE dataset and our repository dataset.

Length	Distribution	
	Repos	AISE
0-10	7.01%	19.16%
10-20	10.53%	23.82%
20-50	17.51%	31.19%
50-100	13.49%	14.81%
100-500	33.55%	11.03%
500+	17.91%	0.00%

Table 3: Detector thresholds for comments in the software engineering domain, derived from the AISE dataset after filtering out comments with fewer than 10 tokens.

Detector	Threshold	F1	TP	FP	TN	FN
Binoculars	-1.00	0.83	553	71	154	149
Log-Likelihood	-3.20	0.58	303	36	189	299
Entropy	2.12	0.61	339	62	163	363
Rank	-163.38	0.78	516	108	117	186
Log-Rank	-1.13	0.72	437	68	157	286
LRR	1.83	0.70	416	68	157	286
DetectGPT	0.04	0.71	457	127	98	245
Fast-DetectGPT	1.18	0.79	486	35	190	216

likely to be LLM-generated, we conducted a coding process to identify the categories these code and comments appeared in. For comments, we followed the previous work [64] which summarized the comment classifications. Padioleau et al. [64] classified comments into different angles including “what", “who", “when", “where". We only applied the “what" dimension from Padioleau et al.’s work because we are concerned on the purpose of the comments detected as likely to be LLM-generated. The codebook for labelling comments is shown in Table 4. As Padioleau et al.’s work studied comments from open-source software written in C, we adapt their categories to support other programming languages. The categories, Code Relationship, PastFuture, Meta, and Explanation, retain the same definitions as in Padioleau et al.’s work, whereas the categories, Type and Interface, are adapted to broader interpretations. The original Type category included C specific subcategories such as Unit, IntRange, and BitsBytes. We refine this category to capture value constraints, units, ranges, formats, and conceptual types that appear across languages. Similarly, the Interface category is broadened into a more general interface contract that captures how functions or modules should be used. For each repository, we considered all comments that were detected as likely to be LLM-generated by any detector between 2021 and 2025, and randomly sampled 370 comments from this combined set. The sampled dataset satisfy a 95% confidence level and a 5% margin of error [65]. Two authors independently coded the first 100 comments in the sampled dataset. For the initial coding, the Cohen’s kappa score is 0.63 [66]. The two authors then discussed their disagreements and reached an agreement on the definitions. Subsequently, they independently coded the remaining sampled comments, achieving a Cohen’s Kappa score of 0.76 [66]. For any inconsistencies, two authors discussed and reached an agreement.

For code, we created a codebook to categorize code, since previous studies did not provide such categorization for code. For each repository, we randomly sample 370 code that is detected as likely to be LLM-generated between 2021 and 2025. The sample size for each repository satisfies a 95% confidence level and a 5% margin of error [65]. Following that, two authors independently analyzed the first 370 samples to create an initial list of categories (codebook). They then annotated the remaining sampled code based on the initial codebook. During the annotation process, when code was encountered that do not fit into the initial codebook, the authors met to discuss and refine the codebook. Through iterative discussions, two authors collaboratively developed a finalized codebook shown in Table 4 and independently coded all the sampled code, with a Cohen’s Kappa score of 0.78 [66]. Finally, the authors discussed any disagreements to reach a consensus.

- RQ2: What are the characteristics of LLM-generated content detected by detectors in repositories?

  To analyze the code detected as likely to be LLM-generated, we employed CCFinderX, a token-based clone detection tool designed for identifying similar code fragments and is widely used in the SE field [58]. CCFinderX detects code clones by tokenizing code into token sequences and identifying repeated or structurally equivalent subsequences across files. During preprocessing, the tool applies normalization to the code, which enables the detection of both Type-1 and Type-2 clones. In Type-2 clones, identifiers and literals may differ, but the overall syntactic structure remains the same. In our experiment, we conducted code clone detection within each repository to identify intra-repository code clones and between each repository and the other seven repositories to identify inter-repository clones. Moreover, we conducted a code clone analysis between code detected as likely to be LLM-generated and the GPTCloneBench dataset [31]. The GPTCloneBench [31] is a benchmark dataset for GPT-generated semantic and cross-language code clones, validated through both manual and automated verification. The GPTCloneBench dataset includes code clones across four programming languages: Python, Java, C, and C#. As the selected repositories (Guava, Zap, Jadx, Kafka, and Pandas) are primarily written in Java and Python, our comparison with GPTCloneBench was limited to these repositories.

  For comments, we analyzed the characteristics of comments detected as likely to be LLM-generated applying the guidelines proposed by Russell et al. [32], which provide a guide to distinguish LLM-generated text from human writing. Since some of the criteria are subjective like originality and tone, we only applied those that can be assessed automatically, including detecting words frequently generated by AI and checking spelling, punctuation and

Table 4: Definitions of categories used to classify code or comments detected as likely to be LLM-generated. Categories listed in the upper section refer to code detected as likely to be LLM-generated, whereas those listed in the lower section refer to comments detected as likely to be LLM-generated.

Category	Description
Core logic	Contains the core rules, business logic, and main algorithms that implement the system’s essential functionality.
Interfaces	Defines the programming contracts, public APIs, and extension points for interacting with the core logic.
Common	Provides shared infrastructure, utilities, and technical implementations.
Tests	Includes test cases used for verification and validation.
Type	Specifies value meanings or constraints.
Interface	Describes the behavioral contract of functions, methods, or modules.
Code Relationship	Specifies some code relationships.
PastFuture	Describes code evolution aspects, such as past changes, current issues, or future tasks.
Meta	Provides authorship, licensing, or other non-behavioral metadata.
Explanation	Comments not covered by the other five categories.

grammar. Following the work of Russell et al. [32], the AI-related words include nouns, verbs, adjectives, and adverbs. To evaluate spelling, punctuation and grammar, we applied automated text analysis at the sentence level using the language_tool_python library [67], which provides sentence-level linguistic error detection.

- RQ3: How does LLMs usage detected by the detectors differ between community-maintained and company-maintained repositories?

  Following the inclusion and exclusion criteria described in Section 4.2, we selected the repositories shown below, which contain both company- and community-maintained repositories.

  The company-maintained repositories are Go-github [68], Guava [69], Liquid [70]and Zap [71]. These repositories are maintained by major technology companies, including Google, Meta, Uber, and Shopify.

  The community-maintained repositories are Act [72], Jadx [73], Kafka [74], and Pandas[75]. These repositories are primarily maintained by the open-source communities, rather than by large technology companies.

  For all the repositories, we followed the same process described in the RQ1 experiment design to detect comments, and code separately, and to compare differences in code and comments detected as likely to be LLM-generated between company-maintained and community-maintained repositories.

- RQ4: How likely are bugs associated with LLM-generated code?

  To answer this question, we detect the human-labelled bugs to see if they were LLM-generated. However, as our dataset does not contain human-labelled bugs, we used a bug dataset, PreciseBugs, curated by Ye et al. [33], containing 1,057,818 bugs from 2,968 open-source repositories and it includes multiple programming languages, such as C/C++, Rust, Go, Python, and Java/JVM. We focused on the Rust, Go, Python, and Java bugs due to the limitation of DetectCodeGPT, which cannot detect code written in C and C++. In the PreciseBugs dataset, there are three sources: two are human-labelled bugs (NVD and OSS-Fuzz) found in repositories, and one is a synthesized version. NVD (National Vulnerability Database) stores the standardized vulnerability reports, and OSS-Fuzz discovers bugs in open-source software. We only considered the human-labelled bugs, as we aimed to analyze whether the bugs in the repositories are likely to be generated by LLMs. In addition, we limited our analysis to the human-labelled bugs generated after October 2021, which aligns with the period covered in our study. Moreover, we define the unit of analysis for this study as the entire source code file containing the labelled bug.

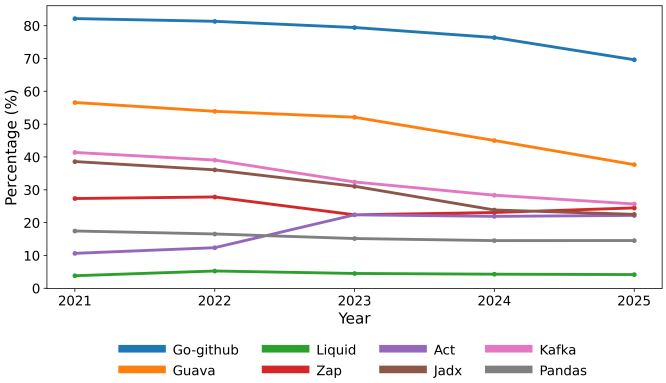


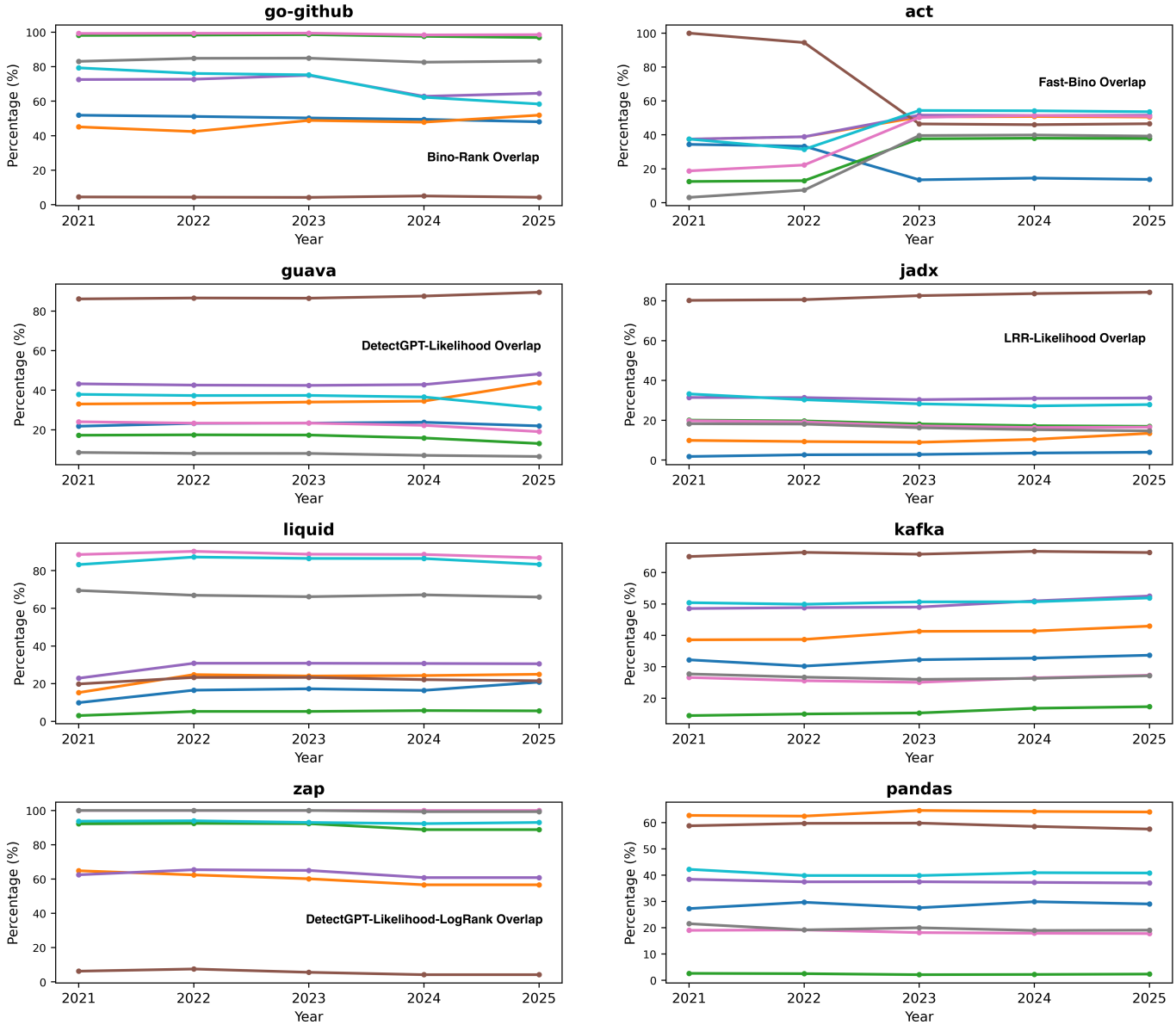
Figure 2: The percentage of code detected as likely to be LLM-generated in repositories detected by DetectCodeGPT. Each line represents one repository. The repositories, Go-github, Guava, Liquid, and Zap represent the company-maintained repositories, while the repositories, Act, Jadx, Kafka, and Pandas represent the community-maintained repositories.

5. Results

5.1. RQ1: How does the proportion of LLM-generated code and comments in repositories, as identified by the detectors, change over time?

Figure 2, 3 show the proportions of likely to be LLM-generated content in each repository detected by each detector. Overall, the proportion of code detected as likely to be LLM-generated by the detectors decreased over time. As shown in Figure 2, out of the eight code repositories, 50% showed a decrease in the proportion of code detected as likely to be LLM-generated. For example, the Go-github and Guava repositories showed a substantial decrease in the proportion of code detected as likely to be LLM-generated, dropping from 82.16% to 69.62% and 56.6% to 37.67% respectively, between 2021 and 2025. In the Kafka repository, the proportion of code detected as likely to be LLM-generated decreased from 41.38% to 25.68%. 12.5% of the repositories showed an increasing trend in the proportion of code detected as likely to be LLM-generated. The Act repository showed a substantial increase from 10.64% to 22.19%. For the remaining repositories, the changes were relatively small. The proportion of code detected as likely to be LLM-generated in the Liquid repository showed a slight increase, while the Zap repository exhibited a slight decrease. However, the relatively high proportions detected in earlier years (2021–2023) should not be interpreted as direct evidence of actual LLM usage. Possible explanations for these early-year signals are discussed further in Section 6. Therefore, the early-year results should be interpreted primarily as baseline reference points for relative temporal comparison rather than definitive indicators of real-world LLM adoption. In addition, the threshold sensitivity analysis discussed in Section 6 suggests that the overall trends remain relatively consistent under threshold variations.

For comments, many of the repositories exhibited a relatively stable proportion of comments detected as likely to be LLM-generated throughout the analysis period, as shown in Figure 3. Different detectors showed varying proportions, but similar



(a) Company-maintained repositories

(b) Community-maintained repositories

Figure 3: The percentage of comments detected as likely to be LLM-generated in each repository over time, using the threshold obtained from AISE dataset. The different colored lines represent the different detectors: the blue line represents DetectGPT, the orange line represents Fast-DetectGPT, the green line represents LRR, the purple line represents Binoculars, the brown line represents Entropy, the pink line represents Log-likelihood, the gray line represents Log-Rank, and the sky-blue line represents Rank. Figure 3(a) shows the percentage change in likely to be LLM-generated comments from the company-maintained repositories between 2021 and 2025, while Figure 3(b) shows the percentage change in likely to be LLM-generated comments from the community-maintained repositories between 2021 and 2025. The overlapped lines are mentioned in wordings in the plots. The following detector lines overlap in the corresponding repositories: in Go-github: Binoculars and Rank; in Guava: DetectGPT and Log-likelihood; in Zap: DetectGPT, Log-likelihood and Log-Rank; in Act: Fast-DetectGPT and Binoculars; in Jadx: LRR and Log-Likelihood.

trends were observed within each repository. The Act repository showed fluctuations in the proportion of comment detected as likely to be LLM-generated before 2023. Among the eight detectors, five detectors, including Log-Rank, Log-likelihood, Rank, LRR, and Binoculars showed an increase, while the other three showed a decrease. After 2023, the proportions of comments detected as likely to be LLM-generated became stable for each detector, with negligible changes in 2024 and 2025. The other repositories showed a stable proportion of comments detected as likely to be LLM-generated. In addition, the threshold sensitivity analysis discussed in Section 6 indicates that these overall patterns remain relatively stable under threshold varia-

Table 5: The proportion of the code and comments detected as likely to be LLM-generated within each category. Categories listed in the upper section refer to code detected as likely to be LLM-generated, whereas those listed in the lower section refer to comments detected as likely to be LLM-generated. The first four columns represent the company-maintained repositories, while the last four columns represent the community-maintained repositories.

Category	Go-github	Guava	Liquid	Zap	Act	Jadx	Kafka	Pandas
Core logic	0.5%	26.1%	40.0%	47.3%	25.8%	21.2%	17.1%	20.2%
Interfaces	37.4%	14.3%	0.0%	7.1%	25.8%	17.2%	32.7%	4.8%
Common	0.5%	26.7%	16.7%	12.5%	30.9%	12.1%	25.1%	7.5%
Tests	61.6%	32.9%	43.3%	33.1%	17.5%	49.5%	25.1%	67.5%
Type	0.0%	3.0%	6.5%	0.0%	1.0%	0.0%	0.0%	2.0%
Interface	0.0%	3.0%	1.0%	0.0%	3.0%	0.0%	0.0%	1.5%
Code Relationship	1.0%	1.0%	1.0%	0.0%	3.5%	5.5%	5.5%	0.0%
PastFuture	2.0%	26.0%	3.0%	0.0%	7.0%	13.5%	1.5%	43.5%
Meta	34.0%	0.0%	74.0%	71.5%	60.0%	31.5%	11.5%	1.0%
Explanation	63.0%	67.0%	14.5%	28.5%	22.5%	49.5%	81.5%	52.0%

tions.

Table 5 shows the results of the coding process used to categorize the code and comments detected as likely to be LLM-generated. We filter out code from third-party dependencies; auto-generated artifacts, such as OpenAPI stubs and minified bundles; standardized library files containing external copyright notices (e.g., Microsoft Corp.) The majority of the code detected as likely to be LLM-generated is in the Tests category, with 62.5% of the repositories showing the highest proportion as likely to be LLM-generated in this category. For instance, the Go-github and Pandas repositories also showed high proportions of Tests category detected as likely to be LLM-generated, with 61.6% and 67.5%, respectively. Compared with the Tests category, the other categories contained a smaller proportion of code detected as likely to be LLM-generated. For comments detected as likely to be LLM-generated, the Meta and Explanation categories account for a large proportion of comments, while the Type, Interface, Code relationship and PastFuture made up a relatively small proportion. For example, in Go-github repository, 97% of the comments detected as likely to be LLM-generated were in the Meta and Explanation categories.

Answer to RQ1

Code detected as likely to be LLM-generated decreased for majority of the repositories, while comments detected as likely to be LLM-generated remained relatively stable, with negligible changes across the years. Code detected as likely to be LLM-generated were primarily in the Tests category, while comments detected as likely to be LLM-generated were mostly found in Meta or Explanation category.

5.2. RQ2: What are the characteristics of LLM-generated content detected by detectors in repositories?

Table 6 presented the code clone results for code detected as likely to be LLM-generated, analyzed within individual repositories and across other repositories. A large proportion of code detected as likely to be LLM-generated exhibits file-level clones within their own repository, with intra-repository file-level clone percentages exceeding 70% in most cases, except for Pandas (45.32%). For the Go-github, Guava, Liquid, Zap and Act repositories, the proportion of code detected as likely to be LLM-generated approaches 100%. For intra-repository method-level clones, the percentage is relatively low, ranging from around 10% to 35%. In the inter-repository clone analysis, there is no code clones detected, and we hypothesized that this is due to the analyzed repositories written in different programming languages and that they have different code structures. Table 7 showed the code clone results for the code detected as likely to be LLM-generated when compare with GPTCloneBench. Only a small proportion of the code detected as likely to be LLM-generated contains clones in GPTCloneBench. In the Zap and Pandas repositories, no code detected as likely to be LLM-generated had code clones in GPTCloneBench, while in the Guava, Jadx, and Kafka repositories, the proportion of likely to be LLM-generated code with clones in GPTCloneBench was mostly lower than 1%, with the highest being 3.09%. This indicates that developers may not directly use GPT outputs into repositories.

Table 6: The percentage of LLM-generated code detected by DetectCodeGPT that contains code clones. Intra represents comparisons within each repository. Inter refers to comparisons between a given repository and the other nine repositories at the file level. We define file-level clones as clones identified by comparing entire files containing code detected as likely to be LLM-generated. Method-level clones are identified by splitting those files into individual methods and applying CCFinderX to detect clones at the method level. Company-maintained repositories are shown above, and community-maintained repositories are shown below.

Repository	Intra		Inter	
	File	Method	File	Method
Go-github	97.89%	34.98%	0.00%	0.00%
Guava	91.56%	26.30%	0.00%	0.00%
Liquid	100%	34.21%	0.00%	0.00%
Zap	90.5%	33.33%	0.00%	0.00%
Act	93.05%	22.81%	0.00%	0.00%
Jadx	85.98%	14.55%	0.00%	0.00%
Kafka	77.04%	23.16%	0.00%	0.00%
Pandas	45.32%	26.29%	0.00%	0.00%

Tables 8 and 9 showed the characteristics of likely to be LLM-generated comments, as detected by the detectors using the average thresholds. The percentage of AI-related words used in in the SE domain is relatively low, possibly because

the AI vocabulary proposed by Russell et al. [32] is primarily derived from the article domain. However, for likely to be LLM-generated comments, the proportion of proper punctuation is high.

Table 7: The proportion of LLM-generated code detected by DetectCodeGPT that forms code clones with code in GPTCloneBench. File-level clones refer to code clones found by comparing entire files containing code detected as likely to be LLM-generated with GPTCloneBench, while method-level clones refer to code clones obtained by splitting those files into individual methods for clone detection. Company-maintained repositories are shown above, and community-maintained repositories are shown below.

Repos	File-level	Method-level
Guava	0.54%	0.05%
Zap	0.00%	0.00%
Jadx	0.57%	0.31%
Kafka	3.09%	0.95%
Pandas	0.00%	0.00%

Answer to RQ2

The code detected as likely to be LLM-generated contains code clones within their own repositories with majority of the repositories exceeding 70% at file-level and 20% at method-level, whereas almost few code is detected as likely to be LLM-generated has clones in GPTCloneBench. Unlike the previous work reporting on the properties of text generated by LLMs, the comments detected as likely to be generated by LLMs show a high proportion of correct punctuation usage, but a low frequency of AI-related vocabulary and correct grammar.

5.3. RQ3: How does LLMs usage detected by the detectors differ between community-maintained and company-maintained repositories?

The company-maintained repositories showed a higher average proportion of likely to be LLM-generated content across code and comment, as detected by the detectors. As shown in Figure 2, the company-maintained repositories generally exhibited a higher proportion of code detected as likely to be LLM-generated. However, the Liquid repository showed very low percentages of code detected as likely to be LLM-generated. Among the community-maintained repositories, the proportion of code detected as likely to be LLM-generated was around 20%, which showed a lower percentage.

As illustrated in Figure 3, although the different detectors showed different results, the highest proportions of comments detected as likely to be LLM-generated in company-maintained repositories were higher than those in community-maintained repositories. For example, Log-Rank, LRR and Log-likelihood showed a high proportion of comment detected as likely to be LLM-generated, with values exceeding 80% for the Go-github. For the Zap repository, Log-Rank, Log-likelihood and DetectGPT detected almost all comments (around 100%) as likely to be LLM-generated. In the Liquid repository, some detectors identify more than 50% of the comments as likely to be LLM-generated. In community-maintained repositories, nearly all the repositories had less than 70% of comments detected as likely to be LLM-generated by all detectors. For the Kafka, and Pandas repositories, all detectors detected less than 70% of the comments as likely to be LLM-generated. In the Jadx repository, the proportion of comments detected as likely to be LLM-generated was below 40% for all detectors except Entropy. Similarly, In the Act repository, the proportion of comments detected as likely LLM-generated remains below 60% for all detectors, except for Entropy in 2021 and 2022.

As shown in Table 6, company-maintained repositories have a higher percentage of likely to be LLM-generated code containing file-level code clones, with the Go-github, Guava, Liquid, and Zap repositories all exceeding 90%. In contrast, the Kafka and Pandas repositories, which are community-maintained repositories, showed relatively lower percentages of likely to be LLM-generated code containing file-level code clones. In particular, the Pandas repository has only 45.32% in the intra-repository file-level code clone comparison. In terms of intra-repository method-level comparisons, the proportions are quite similar between company-maintained and community-maintained repositories. Company-maintained repositories show relatively higher code clone percentages, with the lowest at 26.3% and the highest at 34.98%, while community-maintained repositories have relatively lower percentages, with the lowest at 14.55% and the highest at 31.89%. When compared with GPTCloneBench, both company-maintained and community-maintained repositories had a very low percentage of code detected as likely to be likely LLM-generated that contained code clones. For the characteristics of comments detected as likely to be LLM-generated, as shown in Tables 8, and 9, community-maintained repositories tend to be more linguistically correct and cleaner compared with company-maintained repositories.

As reported in Table 5, although both company-maintained and community-maintained repositories tended to have relatively high proportions of code detected as likely to be LLM-generated in the Tests category, the overall distribution across categories differed between company- and community-maintained repositories. In the company-maintained repositories, the Core logic category has a higher proportion of code detected as likely to be LLM-generated, while in the community-maintained repositories, it showed a more distributed pattern across the categories. For comments detected as likely to be LLM-generated, the company-maintained repositories showed a higher concentration in the Explanation or Meta categories. For example, the Go-github and Guava repositories showed 63% and 67% of comments in the Explanation category, respectively, and the Liquid and Zap repositories showed 74% and 71.5% in the Meta category respectively. In comparison, the community-maintained repositories showed a more varied pattern. For example, in the Pandas repository, 43.5% of the comments detected as likely to be LLM-generated belonged to the PastFuture category.

Table 8: Percentage of comments in company-maintained repositories detected as LLM-generated by different detectors across linguistic characteristics, using the average threshold. S represents Spelling ; P represents Punctuation ; G represents Grammar ; and V represents Vocabulary of AI-related word . Higher spelling, punctuation, and grammar percentage suggest more linguistically polished comments, while a higher vocabulary percentage indicates a higher Log-likelihood that the comment was generated by LLMs due to the frequent use of AI-related words. - indicates that no comments were detected as LLM-generated by a detector. All numbers are in percentage.

Repos	Detectors	S	P	G	V
Go-github	Binoculars	0.20	99.8	80.57	18.00
	Detectgpt	0.00	93.89	32.67	44.67
	Entropy	100	100	100	0.00
	Fast-DetectGPT	0.00	93.25	30.97	50.41
	Log-likelihood	0.05	97.74	55.23	20.40
	Log-Rank	0.32	96.99	49.93	23.09
	LRR	0.06	96.53	44.26	18.69
	Rank	0.11	98.84	79.66	22.23
Guava	Binoculars	58.38	100	74.60	10.46
	Detectgpt	11.59	74.6	28.24	59.56
	Entropy	49.65	96.10	88.79	12.33
	Fast-DetectGPT	0.00	78.38	70.27	72.97
	Log-likelihood	32.97	92.19	44.38	18.12
	Log-Rank	29.08	91.07	39.36	35.33
	LRR	43.63	88.24	40.69	22.06
	Rank	55.05	90.38	53.12	28.12
Liquid	Binoculars	27.17	100	20.65	11.96
	Detectgpt	11.84	89.47	6.58	34.21
	Entropy	-	-	-	-
	Fast-DetectGPT	0.00	100	0.00	8.7
	Log-likelihood	76.94	100	75.65	1.94
	Log-Rank	69.45	97.87	67.50	6.57
	LRR	0.00	100	0.00	8.00
	Rank	84.51	98.86	84.97	2.51
Zap	Binoculars	0.00	100	33.33	66.67
	Detectgpt	0.00	94.89	5.24	28.23
	Entropy	37.5	100	50.00	0.00
	Fast-DetectGPT	0.00	85.78	11.85	68.25
	Log-likelihood	0.00	94.84	4.60	26.36
	Log-Rank	0.00	94.19	6.34	28.27
	LRR	0.00	96.23	3.37	24.6
	Rank	0.00	96.81	3.42	23.46
Average		22.21	94.74	41.49	25.68

Table 9: Percentage of comments in community-maintained repositories detected as LLM-generated by different detectors across linguistic characteristics, using the average threshold. S represents Spelling ; P represents Punctuation ; G represents Grammar ; and V represents Vocabulary of AI-related word . Higher spelling, punctuation, and grammar percentage suggest more linguistically polished comments, while a higher vocabulary percentage indicates a higher Log-likelihood that the comment was generated by LLMs due to the frequent use of AI-related words. - indicates that no comments were detected as LLM-generated by a detector. All numbers are in percentage.

Repos	Detectors	S	P	G	V
Act	Binoculars	74.16	100	77.00	1.81
	Detectgpt	0.00	84.94	3.77	69.04
	Entropy	52.94	95.29	81.18	8.82
	Fast-DetectGPT	25.53	97.87	27.66	31.91
	Log-likelihood	54.60	100	50.19	8.81
	Log-Rank	44.73	98.00	43.93	18.02
	LRR	63.48	98.09	60.08	10.83
	Rank	67.37	100	75.86	4.77
Jadx	Binoculars	93.03	100	92.13	0.22
	Detectgpt	81.82	97.73	82.95	9.09
	Entropy	61.33	99.84	83.28	6.72
	Fast-DetectGPT	-	-	-	-
	Log-likelihood	62.50	100	65.62	9.38
	Log-Rank	92.13	98.69	92.65	2.62
	LRR	99.10	100	97.01	0.30
	Rank	96.98	100	96.70	0.55
Kafka	Binoculars	60.69	93.86	78.54	19.19
	Detectgpt	9.41	79.24	21.66	50.02
	Entropy	72.07	99.29	91.97	9.60
	Fast-DetectGPT	34.15	93.90	70.73	36.59
	Log-likelihood	20.09	92.11	33.41	26.64
	Log-Rank	20.43	84.52	44.04	44.00
	LRR	41.58	84.65	56.93	30.20
	Rank	24.98	88.33	38.99	33.46
Pandas	Binoculars	41.37	93.88	62.59	16.91
	Detectgpt	0.38	79.72	3.50	70.59
	Entropy	52.89	99.41	90.34	8.01
	Fast-DetectGPT	0.00	75.68	1.24	82.63
	Log-likelihood	27.41	90.34	24.61	29.60
	Log-Rank	7.60	87.74	16.75	49.85
	LRR	55.56	100	38.89	3.70
	Rank	36.41	90.76	26.63	27.17
Average		47.57	93.67	55.83	23.26

Answer to RQ3

Compared with community-maintained projects, company-maintained repositories showed a higher proportion of code and comments detected as likely to be LLM-generated and a higher proportion of intra-repository code clones in the code detected as likely to be LLM-generated. Majority of the code detected as likely to be LLM-generated in the company-maintained repositories are test cases.

5.4. RQ4: How likely are bugs associated with LLM-generated code?

We found that only a small portion of the PreciseBugs dataset was detected as likely to be LLM-generated. Specifically, in the NVD data source, 10.79% of the human-labelled bugs are detected as likely to be LLM-generated. Among these, the highest percentage of the human-labelled bugs (12.57%) are in the CWE-287 type (e.g., a system failing to verify a user's claimed identity), followed by 10.34% in the CWE-129 type (e.g., a program using an untrusted array index without proper validation). In the OSS-Fuzz data source, 5.56% of the human-labelled bugs are detected as likely to be LLM-generated, with 100% in the Uncaught exception type (e.g., an exception that is thrown but not handled).

Answer to RQ4

Only a small percentage of the human-labelled bugs (10.79% and 5.56% from NVD and OSS-Fuzz, respectively) in repositories analyzed by DetectCodeGPT are likely to be generated by LLMs.

6. Discussion

Trends of LLM-Generated Content

Our results should be interpreted as proxy-based observations derived from detector outputs, rather than direct measurements of LLM usage. Given the absence of ground truth labels in repository data, the reported proportions reflect detector behaviour under specific thresholds and configurations. Moreover, we emphasize that high detection rates in early years should not be interpreted as evidence of actual LLMs adoption in software development workflows. The high detection rates may be driven by several factors, including detector bias and false positives, repetitive or template-based code patterns, retrospective repository modifications and structural similarities between human-written test code and LLM-generated outputs. Therefore, we treat the early year detection rates as a baseline for comparison, rather than as meaningful indicators of absolute LLM usage. Our analysis focuses on relative changes over time, rather than the absolute values. The observed decreasing trend in the proportion of code detected as likely to be LLM-generated may reflect multiple factors rather than a single underlying cause. One possible interpretation is that it may also reflect changes in developer usage patterns. Developers may begin to use LLMs more strategically. Instead of relying on LLMs to generate code without modification, developers may employ it for tasks such as code completion, which is not the focus in this work. Another possible explanation is that newer generations of LLM-generated code may increasingly resemble human-written code, reducing the sensitivity of existing detectors over time.

As LLMs rapidly improve and become more widely adopted, our observation may not change significantly. As found in the previous work, code generated by LLMs contain bugs [56, 57, 55] and requires developers to review the code generated by LLMs. In our study, we observed that the proportion of code detected as likely to be LLM-generated decreased over time in 50% of the repositories (this is despite more advanced LLMs were introduced in the latter years), while the proportion of comments detected as likely to be LLM-generated have negligible changes over time. We believed this pattern is likely to persist because, even with the introduction of more advanced LLMs, developers still need to carefully review the code generated by LLMs. Comments detected as likely to be LLM-generated are mainly found in the Meta or Explanation categories. The use of LLMs to generate comments remained stable, and we expect this trend to continue in the future.

Detector Limitations and Measurement Bias

Our detector-based approach may underestimate the actual usage of LLMs in software repositories. In earlier years, LLM-generated code may have been easier to detect, while in recent years, it may come from more advanced models, which could lead us to underestimate LLMs usage. In addition, in many cases, the LLM-generated code may undergo substantial human revision before committing to the repositories, making it difficult for detectors to identify it as LLM-generated.

Meanwhile, detectors may also overestimate LLM-generated code and comments. Human-written code with repetitive or highly structured patterns may resemble generated code and may be incorrectly detected as likely LLM-generated. Since we did not conduct surveys or interviews with developers, we could not validate whether detected content was generated with LLMs. We chose not to conduct such a survey to investigate how developers are using LLMs to generate code and comments in their repositories, because we believed that these inquiries could raise ethical concerns such as identifying a specific developer [76]. Consequently, our findings should be interpreted as detector-based proxy observations rather than precise measurements of actual LLM usage.

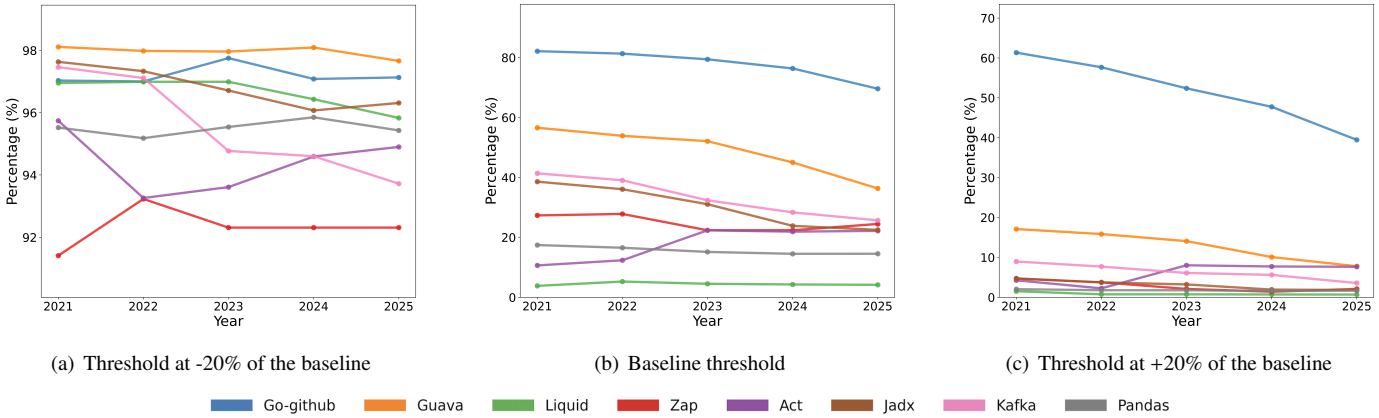
To evaluate the robustness of these observations, we conducted a sensitivity analysis by varying the threshold DetectCodeGPT by ±20% and the thresholds of the comment detectors by ±20%. The results for code are shown in Figure 4, while the results for comments are shown in Figures 5 and 6. For code detection, adjusting the DetectCodeGPT threshold from 1.3 by ±20% causes notable fluctuations in the proportions. However, the overall trend remains similar. For most repositories, the proportion of code detected with the baseline threshold and the +20% variation decreases over time. With a -20% variation, the threshold is generous that over 90% of code is detected as likely LLM-generated across all repositories. For comment detection, different detectors produced different absolute proportions under threshold variations. Nevertheless, the overall patterns remained relatively stable across repositories. These findings suggest that our observations are relatively robust. However, even thresholds adapted using the AISE dataset should still be regarded as approximations rather than definitive decision thresholds. In addition, constructing ground-truth datasets for developer comments remains an open challenge, as it is difficult to obtain verified human-written and LLM-generated comments.

Characteristics of Likely to be LLM-Generated Content

We observed that the majority of code detected as likely to be LLM-generated is concentrated in the Tests category. Through the coding process, we observed that code, detected as likely to be LLM-generated in Tests category, consist of highly repetitive code, hard-coded JSON strings, and symmetric benchmark loops. Because DetectCodeGPT perturbs input code by strategically inserting spaces and newlines [18], these perturbations may not significantly shift the perplexity, resulting in low perturbation sensitivity for such code. As a result, even human-written test code can be detected as likely to be LLM-generated because its structural patterns are likely to be detected as LLM-generated.

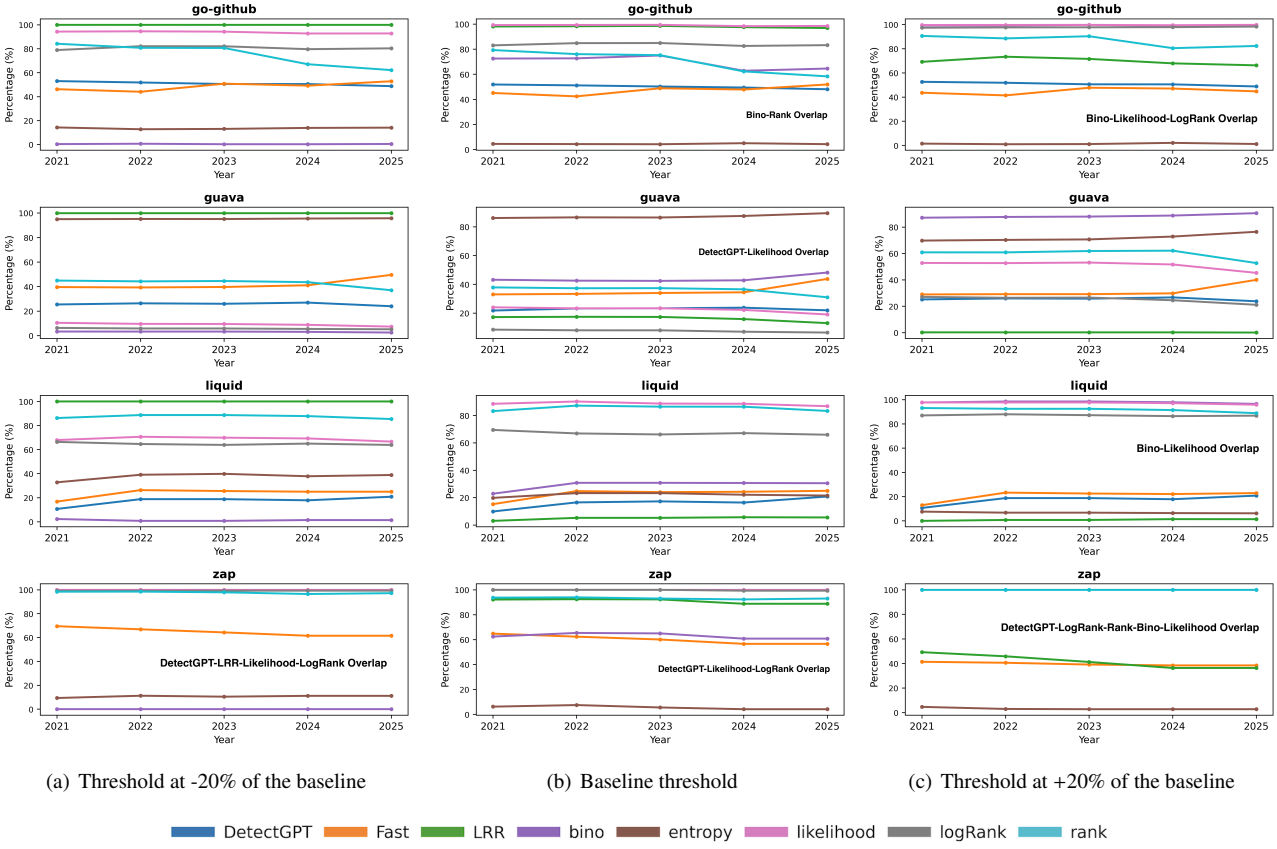
In addition, we found that likely LLM-generated code exhibited a high intra-repository clone rate. This indicates a risk of technical debt. We recommend that automated refactoring tools can be integrated into the CI/CD pipeline to detect and consolidate repetitive LLM-generated snippets, preventing long-term maintenance issues.

We found that the company-maintained repositories contained a higher proportion of LLM-generated code and comments compared to community-maintained repositories. This may be because companies tend to employ LLMs to accelerate



(a) Threshold at -20% of the baseline (b) Baseline threshold (c) Threshold at +20% of the baseline

Figure 4: Proportion of code detected as LLM-generated across different repositories, comparing the selected baseline threshold with variations of −20% and +20%. Figure 4(b) shows the overall change in the proportion of code detected as likely LLM-generated using the baseline threshold, with +20% and -20% variations illustrated in Figures 4(c) and 4(a), respectively.

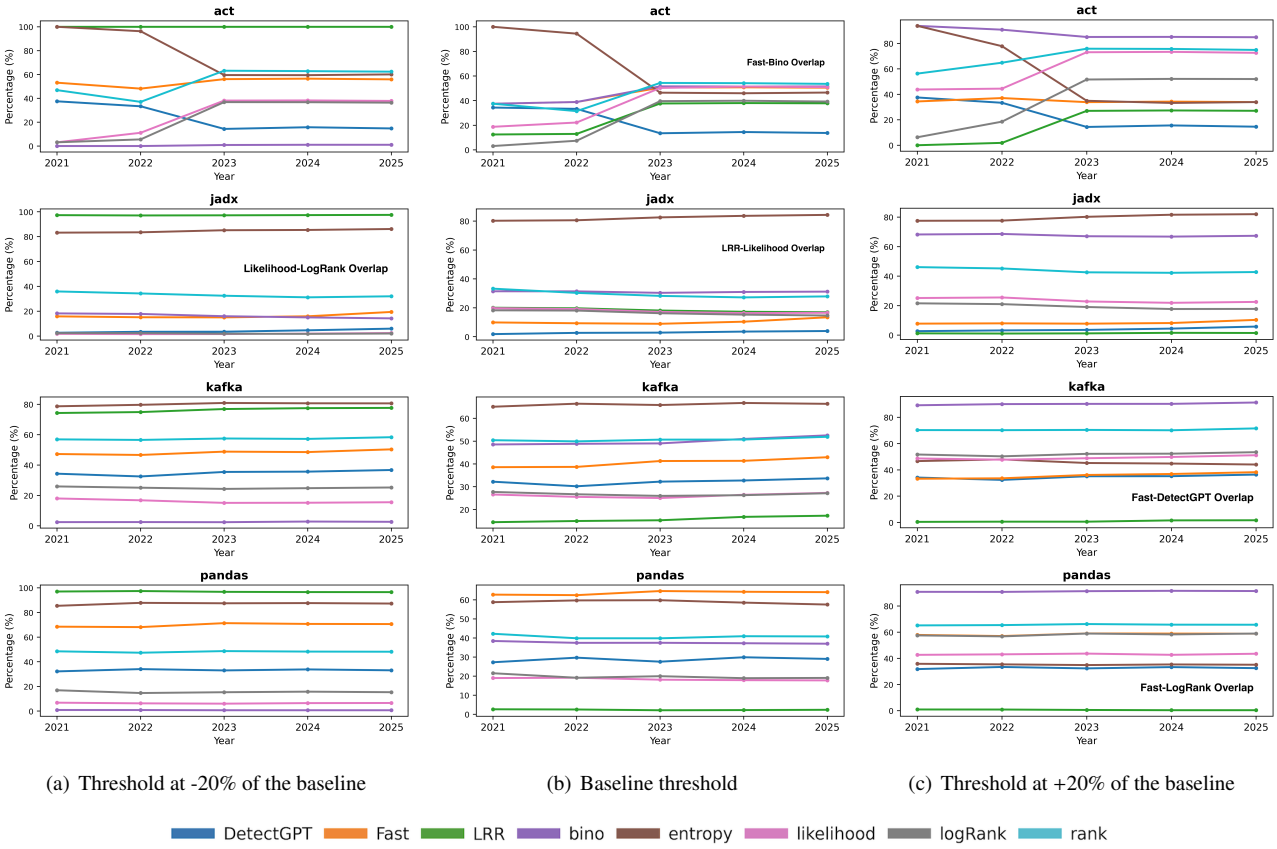


(a) Threshold at -20% of the baseline (b) Baseline threshold (c) Threshold at +20% of the baseline

Figure 5: Proportion of comment detected as LLM-generated in company-maintained repositories, comparing the AISE threshold with variations of −20% and +20%. Figure 5(b) shows the overall change in the proportion of comment detected as likely LLM-generated using the AISE dataset threshold, with +20% and -20% variations illustrated in Figures 5(c) and 5(a), respectively. The overlapped lines are mentioned in wordings in the plots. The following detector lines overlap in the corresponding repositories: in Figure 5(a): in Zap: DetectGPT, LRR, Log-Likelihood and Log-Rank. in Figure 5(c): in Go-github: Binoculars, Log-Likelihood and LogRank; in Liquid: Binoculars and Log-Likelihood; in Zap: DetectGPT, LogRank, Rank, Binoculars and Log-Likelihood.

project development and maintain internal consistency, which in turn leads to higher percentage of code and comments detected as likely to be LLM-generated.

Previous studies have found that developers are often reluctant to use LLMs because the generated code does not meet their requirements, the output is difficult to control, and they spend a lot of time debugging. However, our results showed that only a small percentage of the human-labelled bugs (10.79% and 5.56% from NVD and OSS-Fuzz, respectively) in repositories were likely to be generated by LLMs. While Detect-



(a) Threshold at -20% of the baseline (b) Baseline threshold (c) Threshold at +20% of the baseline

Figure 6: Proportion of comment detected as LLM-generated in community-maintained repositories, comparing the AISE threshold with variations of −20% and +20%. Figure 6(b) shows the overall change in the proportion of comment detected as likely LLM-generated using the AISE dataset threshold, with +20% and -20% variations illustrated in Figures 6(c) and 6(a), respectively. The overlapped lines are mentioned in wordings in the plots. The following detector lines overlap in the corresponding repositories: in Figure 6(a): in Jadx: Log-Likelihood and Log-Rank. In Figure 6(c): in Kafka: Fast-DetectGPT and DetectGPT; in Pandas: Fast-DetectGPT and LogRank.

CodeGPT may underestimate LLM-generated code that contains bugs, the low percentage suggests that developers may have modified the LLM-generated code to remove any bugs before committing to the repositories. It should be noted that the bugs in our analysis, sourced from NVD and OSS-Fuzz, primarily represent security vulnerabilities and runtime exceptions. Therefore, the results may not generalize to all types of software defects.

7. Threats to Validity

In this section, we address threats to the validity of our study.

7.1. Internal Validity

In our experiments, we used existing detectors to identify LLM-generated content in repositories. However, the repositories do not have ground truth on whether the content is LLM-generated. Therefore, our analysis relies on the existing detectors' outputs, which may not perfectly reflect the true extent of developers' LLMs usage, as they may overestimate or underestimate the LLM-generated code and comments. To mitigate this, we made every effort to use the existing resources to detect the code and comments that were likely to be generated by LLMs by using multiple detectors and threshold settings, improving the reliability of our measurements. Moreover, we conducted a sensitivity analysis with ±20% threshold variation. The results show that while absolute values change, the overall temporal trends remain stable.

7.2. External Validity

In this study, we primarily focused on repositories maintained by large companies and active communities, which have significant contributions and a high number of stars. Although these projects are active, the results are skewed toward mature and popular ecosystems, and thus may not generalize to other ecosystems. Moreover, we selected repositories developed in programming languages that can be detected by DetectCodeGPT. As a result, the generalizability of our findings may be limited to similar types of repositories, and may not directly extend to less active repositories, smaller projects, or repositories in unsupported programming languages. Finally, our dataset is restricted to open-source projects. Therefore, the results cannot be generalized to closed-source industrial repositories. However, to make our findings more representative, we analyzed repositories from different companies and organiza-

tions. These repositories were also developed in different programming language to improve the diversity of our dataset.

7.3. Construct Validity.

Although we observed the percentage of the code and human-labelled bugs detected as likely to be LLM-generated is low, developers may still use LLMs to assist them on code refactoring, debugging, code comprehension, and other tasks, and that the developers may also rewrite or paraphrase the content generated by LLMs and commit into the repositories. In such cases, we believe there is no one single detectors that can identify such LLM-assisted content that has been substantially modified. Furthermore, there is no existing study that reports how much of a modified LLM-generated content can be identified by these detectors. To mitigate this, we use a variety of detectors of various abilities, including state-of-the-art detectors, to detect content as likely to be LLM-generated. In addition, the detection of LLM-generated comments introduces further construct validity challenges due to domain differences. Programming comments often contain inline code, parameter descriptions, abbreviations, and DSL-like structures that differ substantially from the natural language data commonly used to calibrate existing detectors. As a result, natural language detectors may not generalize perfectly to software engineering comments and may introduce domain-specific detection bias.

7.4. Detection Validity.

Our analysis relies on zero-shot detectors whose thresholds are derived from proxy datasets rather than ground-truth annotations. As optimal thresholds vary across domains, the classification of LLM-generated content remains uncertain. To mitigate this, we applied domain-specific threshold derivation and sensitivity analysis. However, the results should still be interpreted as indicative rather than definitive. In addition, detector behaviour may change over time as newer LLMs evolve. Detectors calibrated on earlier generations of LLM outputs may become less effective when applied to newer models whose outputs increasingly resemble human-written content.

8. Conclusion

With the advancement of LLMs, their adoption has become increasingly popular among developers. In this study, we aim to show how code and comments detected as likely to be LLM-generated appear in repositories. We analyzed active repositories maintained by both companies and the community. Our proxy-based observations indicate that the proportion of likely to be LLM-generated code decreased over time, while the proportion of comments detected as likely to be LLM-generated remained stable. Based on detector-based proxy analysis, we also observed that company-maintained repositories had a higher percentage of code and comments detected as likely to be LLM-generated and exhibited higher intra-repository code clone rates, whereas community-maintained projects produced linguistically cleaner comments detected as likely to be LLM-generated. In addition, only a small percentage of the human-labelled bugs (10.79% and 5.56% from NVD and OSS-Fuzz, respectively) in the repositories were detected as likely to be LLM-generated. Building on this study, future work could establish guidelines for reviewing, such as designating low-priority test files for AI-assisted review, to improve efficiency.

Acknowledgement

Calculations were performed using the Sulis Tier 2 HPC platform hosted by the Scientific Computing Research Technology Platform at the University of Warwick. Sulis is funded by EPSRC Grant EP/T022108/1 and the HPC Midlands+ consortium.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work the author(s) used detectors (including Binoculars, Log-Likelihood, Entropy, Rank, Log-Rank, LRR, DetectGPT, Fast-DetectGPT and Detect-CodeGPT) in order to run the core experiments required in the manuscript. We did not use generative AI or AI-assisted technologies to produce any of the content in the manuscript.

References

[1] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, vol. 1, no. 2, 2023.

[2] J. D. Weisz, S. V. Kumar, M. Muller, K.-E. Browne, A. Goldberg, K. E. Heintze, and S. Bajpai, "Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise," in *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2025, pp. 1–13.

[3] M. Ciniselli, N. Cooper, L. Pascarella, A. Mastropaolo, E. Aghajani, D. Poshyvanyk, M. Di Penta, and G. Bavota, "An empirical study on the usage of transformer models for code completion," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 4818–4837, 2021.

[4] T. Ahmed and P. Devanbu, "Few-shot training llms for project-specific code-summarization," in *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*, 2022, pp. 1–5.

[5] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[6] P. Shojaee, A. Jain, S. Tipirneni, and C. K. Reddy, "Execution-based code generation using deep reinforcement learning," *arXiv preprint arXiv:2301.13816*, 2023.

[7] W. Sun, Y. Miao, Y. Li, H. Zhang, C. Fang, Y. Liu, G. Deng, Y. Liu, and Z. Chen, "Source code summarization in the era of large language models," *arXiv preprint arXiv:2407.07959*, 2024.

[8] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *arXiv preprint arXiv:2102.04664*, 2021.

[9] "Jetbrains developer ecosystem survey," 2023. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2023/ai/

[10] "Google ceo says more than a quarter of the company's new code is created by ai," 2024. [Online]. Available: https://news.ycombinator.com/item?id=41991291

[11] "Up to 30% of microsoft's code is now written by ai: Ceo satya nadella," 2025. [Online]. Available: https://shorturl.at/be0vn

[12] Google, "Dora report," 2025. [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report

[13] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," *Commun. ACM*, vol. 68, no. 2, p. 96–105, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3610721

[14] "Copilot hosted by github." [Online]. Available: https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[15] "Copilot hosted by microsoft," 2025. [Online]. Available: https://copilot.microsoft.com/

[16] J. T. Liang, C. Yang, and B. A. Myers, "A large-scale survey on the usability of ai programming assistants: Successes and challenges," in *Proceedings of the 46th IEEE/ACM international conference on software engineering*, 2024, pp. 1–13.

[17] A. Sergeyuk, Y. Golubev, T. Bryksin, and I. Ahmed, "Using ai-based coding assistants in practice: State of affairs, perceptions, and ways forward," *Information and Software Technology*, vol. 178, p. 107610, 2025.

[18] Y. Shi, H. Zhang, C. Wan, and X. Gu, *Between Lines of Code: Unraveling the Distinct Patterns of Machine and Human Programmers*. IEEE Press, 2025, p. 1628–1639. [Online]. Available: https://doi.org/10.1109/ICSE55347.2025.00005

[19] S. Bukhari, B. Tan, and L. De Carli, "Distinguishing ai- and human-generated code: A case study," in *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, ser. SCORED '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 17–25. [Online]. Available: https://doi.org/10.1145/3605770.3625215

[20] S. Gehrmann, H. Strobelt, and A. M. Rush, "Gltr: Statistical detection and visualization of generated text," *arXiv preprint arXiv:1906.04043*, 2019.

[21] E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn, "Detectgpt: Zero-shot machine-generated text detection using probability curvature," in *International conference on machine learning*. PMLR, 2023, pp. 24 950–24 962.

[22] G. Bao, Y. Zhao, Z. Teng, L. Yang, and Y. Zhang, "Fast-detectgpt: Efficient zero-shot detection of machine-generated text via conditional probability curvature," *arXiv preprint arXiv:2310.05130*, 2023.

[23] A. Hans, A. Schwarzschild, V. Cherepanova, H. Kazemi, A. Saha, M. Goldblum, J. Geiping, and T. Goldstein, "Spotting llms with binoculars: zero-shot detection of machine-generated text," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024.

[24] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, J. W. Kim, S. Kreps *et al.*, "Release strategies and the social impacts of language models," *arXiv preprint arXiv:1908.09203*, 2019.

[25] T. Lavergne, T. Urvoy, and F. Yvon, "Detecting fake content with relative entropy scoring." *Pan*, vol. 8, no. 27-31, p. 4, 2008.

[26] J. Su, T. Zhuo, D. Wang, and P. Nakov, "DetectLLM: Leveraging log rank information for zero-shot detection of machine-generated text," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 12 395–12 412. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.827/

[27] S. Black, G. Leo, P. Wang, C. Leahy, and S. Biderman, "Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow," *Zenodo*, 2021.

[28] J. Wu, R. Zhan, D. Wong, S. Yang, X. Yang, Y. Yuan, and L. Chao, "Detectrl: Benchmarking llm-generated text detection in real-world scenarios," *Advances in Neural Information Processing Systems*, vol. 37, pp. 100 369–100 401, 2024.

[29] J. Katzy, Y. Huang, G.-R. Panchu, M. Ziemlewski, P. Loizides, S. Vermeulen, A. van Deursen, and M. Izadi, "A qualitative investigation into llm-generated multilingual code comments and automatic evaluation metrics," in *Proceedings of the 21st International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 31–40. [Online]. Available: https://doi.org/10.1145/3727582.3728683

[30] W. Wu, H. Hu, Z. Fan, Y. Qiao, Y. Huang, Y. Li, Z. Zheng, and M. Lyu, "An empirical study of code clones from commercial ai code generators," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 2874–2896, 2025.

[31] A. I. Alam, P. R. Roy, F. Al-Omari, C. K. Roy, B. Roy, and K. A. Schneider, "Gptclonebench: A comprehensive benchmark of semantic clones and cross-language clones using gpt-3 model and semanticclonebench," in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2023, pp. 1–13.

[32] J. Russell, M. Karpinska, and M. Iyyer, "People who frequently use ChatGPT for writing tasks are accurate and robust detectors of AI-generated text," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 5342–5373. [Online]. Available: https://aclanthology.org/2025.acl-long.267/

[33] Y. He, Z. Chen, and C. Le Goues, "Precisebugcollector: Extensible, executable and precise bug-fix collection: Solution for challenge 8: Automating precise data collection for code snippets with bugs, fixes, locations, and types," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1899–1910.

[34] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 international symposium on software testing and analysis*, 2014, pp. 437–440.

[35] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy, "Inferfix: End-to-end program repair with llms," in *Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering*, 2023, pp. 1646–1656.

[36] J. Wu, S. Yang, R. Zhan, Y. Yuan, L. S. Chao, and D. F. Wong, "A survey on LLM-generated text detection: Necessity, methods, and future directions," *Computational Linguistics*, vol. 51, no. 1, pp. 275–338, Mar. 2025. [Online]. Available: https://aclanthology.org/2025.cl-1.8/

[37] B. Zhang, P. Liang, X. Zhou, A. Ahmad, and M. Waseem, "Practices and challenges of using github copilot: An empirical study," *arXiv preprint arXiv:2303.08733*, 2023.

[38] M. Jaworski and D. Piotrkowski, "Study of software developers' experience using the github copilot tool in the software development process," *arXiv preprint arXiv:2301.04991*, 2023.

[39] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Measuring github copilot's impact on productivity," *Communications of the ACM*, vol. 67, no. 3, pp. 54–63, 2024.

[40] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," *arXiv preprint arXiv:2302.06590*, 2023.

[41] S. K. Kuttal, B. Ong, K. Kwasny, and P. Robe, "Trade-offs for substituting a human with an agent in a pair programming context: the good, the bad, and the ugly," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–20.

[42] S. Imai, "Is github copilot a substitute for human pair-programming? an empirical study," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 319–321.

[43] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 17 061–17 084.

[44] X. Zhao, P. Ananth, L. Li, and Y.-X. Wang, "Provable robust watermarking for ai-generated text," *arXiv preprint arXiv:2306.17439*, 2023.

[45] W. Liang, Z. Izzo, Y. Zhang, H. Lepp, H. Cao, X. Zhao, L. Chen, H. Ye, S. Liu, Z. Huang, D. A. McFarland, and J. Y. Zou, "Monitoring ai-modified content at scale: a case study on the impact of chatgpt on ai conference peer reviews," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024.

[46] K. Singh and J. Zou, "New evaluation metrics capture quality degradation due to llm watermarking," *arXiv preprint arXiv:2312.02382*, 2023.

[47] D. Ippolito, D. Duckworth, C. Callison-Burch, and D. Eck, "Automatic detection of generated text is easiest when humans are fooled," *arXiv preprint arXiv:1911.00650*, 2019.

[48] R. Bhagat and E. Hovy, "What is a paraphrase?" *Computational linguistics*, vol. 39, no. 3, pp. 463–472, 2013.

[49] A. Uchendu, T. Le, K. Shu, and D. Lee, "Authorship attribution for neural text generation," in *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, 2020, pp. 8384–8395.

[50] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi, "Defending against neural fake news," *Advances in neural information processing systems*, vol. 32, 2019.

[51] T. Fagni, F. Falchi, M. Gambini, A. Martella, and M. Tesconi, "Tweepfake: About detecting deepfake tweets," *Plos one*, vol. 16, no. 5, p. e0251415, 2021.

[52] B. Guo, X. Zhang, Z. Wang, M. Jiang, J. Nie, Y. Ding, J. Yue, and Y. Wu, "How close is chatgpt to human experts? comparison corpus, evaluation, and detection," *arXiv preprint arXiv:2301.07597*, 2023.

[53] P. T. Nguyen, J. Di Rocco, C. Di Sipio, R. Rubei, D. Di Ruscio, and M. Di Penta, "Is this snippet written by chatgpt? an empirical study with a codebert-based classifier," *arXiv preprint arXiv:2307.09381*, 2023.

[54] Z. Fan, X. Gao, M. Mirchev, A. Roychoudhury, and S. H. Tan, "Automated repair of programs from large language models," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1469–1481.

[55] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. J. Jiang, "Github copilot ai pair programmer: Asset or liability?" *Journal of Systems and Software*, vol. 203, p. 111734, 2023.

[56] Y. Liu, T. Le-Cong, R. Widyasari, C. Tantithamthavorn, L. Li, X.-B. D. Le, and D. Lo, "Refining chatgpt-generated code: Characterizing and mitigating code quality issues," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, pp. 1–26, 2024.

[57] F. Tambon, A. Moradi-Dakhel, A. Nikanjam, F. Khomh, M. C. Desmarais, and G. Antoniol, "Bugs in large language models generated code: An empirical study," *Empirical Software Engineering*, vol. 30, no. 3, p. 65, 2025.

[58] T. Kamiya, S. Kusumoto, and K. Inoue, "Ccfinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE transactions on software engineering*, vol. 28, no. 7, pp. 654–670, 2002.

[59] "Github graphql api," 2024. [Online]. Available: https://docs.github.com/en/graphql

[60] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, Apr. 2023. [Online]. Available: https://doi.org/10.1145/3586030

[61] M. Chen, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[62] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[63] S. Chakraborty, A. S. Bedi, S. Zhu, B. An, D. Manocha, and F. Huang, "On the possibilities of ai-generated text detection," *arXiv preprint arXiv:2304.04736*, 2023.

[64] Y. Padioleau, L. Tan, and Y. Zhou, "Listening to programmers—taxonomies and characteristics of comments in operating system code," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 331–341.

[65] H. Yang, W. Lian, S. Wang, and H. Cai, "Demystifying issues, challenges, and solutions for multilingual software development," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1840–1852.

[66] T. Byrt *et al.*, "How good is that agreement?" *Epidemiology*, vol. 7, no. 5, p. 561, 1996.

[67] "language_tool_python," 2025. [Online]. Available: https://pypi.org/project/language-tool-python/

[68] Google, "go-github," 2025. [Online]. Available: https://github.com/google/go-github

[69] ——, "Guava," 2025. [Online]. Available: https://github.com/google/guava

[70] Shopify, "Liquid," 2025. [Online]. Available: https://github.com/Shopify/liquid

[71] Uber, "Zap," 2025. [Online]. Available: https://github.com/uber-go/zap

[72] "act," 2025. [Online]. Available: https://github.com/nektos/act

[73] "jadx," 2025. [Online]. Available: https://github.com/skylot/jadx

[74] Apache, "Kafka," 2025. [Online]. Available: https://github.com/apache/kafka

[75] "Pandas," 2025. [Online]. Available: https://github.com/pandas-dev/pandas

[76] M. Tahaei, D. Wilkinson, A. Frik, M. Muller, R. Abu-Salma, and L. Wilcox, "Surveys considered harmful? reflecting on the use of surveys in ai research, development, and governance," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, vol. 7, 2024, pp. 1416–1433.