Changes in Vocabulary Between Human-created and LLM-assisted Code Repositories

Taylor Lineman
Dept. of Software Engineering
Rochester Institute of Technology
Rochester, NY, United States of America
zml9650@rit.edu

***Abstract*—Detecting whether code was created by a human or generated by an LLM is a currently unsolved problem. Tracking code contributions by LLMs is incredibly important, especially as approximately 40% to 60% of code generated by LLMs has been found to be insecure [6], [7]. As more contributions created by AI Agents show up in open-source projects, it will become increasingly difficult for project maintainers to ensure the submitted code was vetted by a human.**

This paper attempted to solve this problem by comparing token frequencies and lexical diversity metrics between two datasets of Python software repositories. The human dataset is a filtered version of the repository dataset by N. Munaiah et al. [14]. A dataset of LLM-assisted code was created through an automated incremental search and analysis of GitHub repositories released after May 1, 2025 (the release of Claude Code). Repositories were chosen based on the inclusion of AI Agent configuration files, or LLM commit authorship. The human and LLM-assisted repository datasets were mined, then analyzed by the `strata` tool.

The analysis revealed a significant difference in the vocabulary humans and LLMs use in code comments. Identifiers and commit messages did not present any significant difference. This difference in comment vocabulary is expressed by LLMs repeating words 1.73x more often than a human. This paper shows that linguistic analysis could be a path forward for detecting the origin of a code comment. Alongside other metrics, the work from this paper could aid in the creation of a tool that would help engineers better manage their time and secure their software by detecting what code came from a human versus an LLM.

***Index Terms*—source code, software repository mining, generative ai, GPT code, vibe code, linguistic analysis**

I. Introduction

With the advent of Generative Artificial Intelligence (AI) / Large Language Models (LLMs), more of the software engineering process is being automated [1]. As pieces of the process are being abstracted and automated, more tools continue to emerge. The combination of LLMs with decision-making abilities and tool use has resulted in the creation of AI Agents. An AI Agent, such as Claude Code (introduced in early 2025), has the ability to create entire software projects with little to no human interaction [2].

Many of these fully automated projects, similar to human-created projects, are published to existing open-source software hosts, such as GitHub. These projects are then used as references and dependencies within other projects. This muddles the chain of security that is normally present in open-source software [3].

Some AI Agents participate as an equal citizen in the open-source community, through the use of git co-authorship [4], [5]. This provides a transparent view into what code was written with the help of an AI Agent. However, these attributions are not provided by all AI Agents and are not added when code is copied from an LLM frontend such as ChatGPT or Gemini.

This paper presents an experimental method for identifying Generative AI contributions to programming projects. Tracking code contributions by LLMs is incredibly important, especially as approximately 40% to 60% of code generated by LLMs has been found to be insecure [6], [7]. As more contributions created by AI Agents show up in open-source projects, it will become increasingly difficult for project maintainers to ensure the submitted code was vetted by a human.

This paper attempts to answer the question: **Do human-created projects and AI-assisted projects have any significant differences in vocabulary?** The approach taken in this paper used a frequency analysis of repeated vocabulary used in projects created by humans and by AI agents. An analysis of vocabulary clustering was also used to support frequency differences.

The core research question is a precursor to the true question of interest: **Can a computer program detect if code came from an LLM or a human**? Developing an answer to this question will be a significant undertaking, but I aim to describe the foundational steps in this paper.

II. Related Work

Large Language Models generate text based on large training datasets primarily comprised of human-created works [8]. Since models have been trained on data from humans, they generate text that closely resembles human-created content. This similarity makes it particularly challenging to detect whether text originated from an LLM or a human [9]. Many existing works have found methods to detect LLM-generated text, with varying success rates. This section will explore key findings from this previous literature.

A. Distinguishing AI- and Human-Generated Code: A Case Study [3]

Bukhari et al. explore the use of lexical and syntactic features to distinguish between AI and human-generated code.

Their research aims to find a way to distinguish between AI and human-generated code for the purpose of reducing the supply chain risk of including code written by an AI. They argue for the importance of tracking generated AI code so adequate risk mitigation can be performed.

The lexical features used in their paper are tailored to features present within source code: text length, keyword usage, and line length. The syntactic features track occurrences of generalized program elements, such as: arithmetic, comments, conditions, and declarations. These feature sets were then used to create lexical and syntactic feature classifiers. Bukhari et al. found that the syntactic classifiers performed better than lexical classifiers, and a combination of the two performed marginally better than just a syntactic one. I did not pursue the creation of a classification model, as they require large, clean datasets. The goal of this research was to find a mathematical difference rather than relying on a classification model.

The work by Bukhari et al. uses TreeSitter [10] to create their abstract syntax trees (ASTs), whereas I used srcML to create my ASTs. I did consider TreeSitter for my paper since it supports 423 languages, many more than the 6 offered by srcML [11]. However, the time needed to switch the existing srcML integration to TreeSitter was not available before the completion of this paper. Despite their implementation differences, srcML and TreeSitter produce the same final result, an abstract representation a code source file.

B. Playing with words: Comparing the vocabulary and lexical diversity of ChatGPT and humans [8]

To determine the difference between LLM-generated and human-created code, it is necessary to analyze how both write essays. P. Reviriego et al. explore the difference in lexical diversity of English essays written by ChatGPT and humans. They assert that the diversity of human language will change as LLMs become more prevalent.

P. Reviriego et al. used three metrics: Type-Token-Ratio (TTR), Root-Type-Token-Ratio (RTTR), and Maas. The paper cites Hout & Vermeer, 2007; however, that paper refers to RTTR as "Indice de richesse / Guiraud" and only includes an alternate copy of the Maas equation which is referred to as the "Index Uber". The Maas equation actually originates from H.D. Maas (1972) [12]. In addition to these inconsistencies, the TTR and RTTR metrics have been proven to be inconsistent at large text sizes [13], and therefore will not be used for analysis in this paper. Maas, however, stays consistent throughout text sizes and will be used in this paper.

The work of P. Reviriego et al. presented the foundational idea of using lexical diversity metrics for the detection of human and LLM text. The current paper expands on this idea, by including work from F. Zenker and K. Kyle.

C. Investigating minimum text lengths for lexical diversity indices [13]

Measuring the diversity of language is a vast area of research; F. Zenker and K. Kyle examined 9 metrics for lexical diversity for their consistency across text lengths. This is necessary as many classical metrics for lexical diversity struggle to produce consistent results with varying text lengths.

The results from this paper, were used to select the Maas [12], MATTR [16], and HD-D [17] metrics. These were selected since were consistent across text-lengths and produce a value that represents how diverse the input text is.

III. Methodology

A. Tooling

A tool named `strata` was created for the execution of this research project. Originally created for the completion of SWEN 640: Research Methods, this tool was adapted to fit the needs of this paper. This tool was built as a swiss-army-knife for Software Repository Mining. It includes features such as: identifier extraction, comment extraction, commit analysis and cleaning, vocabulary clustering, and frequency analysis. It also allows for the training of cluster identification models.

B. Data Collection

To properly analyze the difference between human-created and AI-assisted code, two datasets were generated. One containing human-created repositories, the other containing projects with identifiable AI contributions. These repositories had a couple of common requirements for consideration: they needed to have more than 100 stars on GitHub and be labeled as a Python project by GitHub.

1) Human Repository Dataset: Human repositories were extracted from a dataset of engineered software projects, created by N. Munaiah et al. [14]. This dataset was then filtered using the common requirements discussed earlier. Although it cannot be confirmed that the repositories in this dataset are AI-free, the likelihood is miniscule. The foundational paper [15] that led to the creation of Transformers, the key component in LLMs, was published two months after the dataset by N. Munaiah. Generative AI did not gain popularity as a tool for programming until much later, with the release of ChatGPT to the public in 2022. See the Repository Mining subsection for the rest of the steps taken to limit the chance of AI influence on the human dataset.

2) LLM-assisted Repository Dataset: Finding a set of repositories created with the assistance of LLMs or created wholly by LLMs proved to be a challenging task. Few of these repositories are labeled with disclosures, and many of the methods of disclosure are polluted with tooling for making LLM projects. The first approach explored was to use GitHub's topic feature to find projects tagged with *vibe-coding*, *vibe*, *vibe-coded*, and *vibe coded*. These tags did not result in nearly enough repositories; many of the projects appearing under them were either unpopulated or just tooling.

As discussed earlier, some AI Agents and AI Coding Tools tag their contributions to projects through the use of Git Attributions and configuration folders [4], [5]. These files are usually committed to a repository to allow for reproducibility across different environments, or for giving AI Agents the ability to orchestrate sub-environments and agents. These commit attributions and configuration files are a great indicator

of the use of LLMs for code generation. Attributed commits are concrete proof that an LLM was used to generate code within a commit, and therefore the repository. Configuration files are less concrete. However, their existence in the repository accepts and supports AI Agent work within the project, qualifying the repository as LLM-assisted programming.

With a set of characteristics defined, a tool was needed to perform repetitive searches on GitHub to find a list of valid repositories. For this job, a `find` command was added to the `strata` mining tool. The first stage of this command is to find a list of repositories that were created in a set date range; this paper's range was from May 1, 2025, to April 11, 2026. The release of Claude Code [2] was chosen as the starting date because the release of Claude Code started a wave of AI Agent advancements. The second step is to search within every discovered repository for a commit or co-commit from an AI Agent, as well as agent configuration files. Searching was performed on a single repository at a time to avoid the 1000 result maximum from the GitHub API. Performing a search wider than a single repository is possible and will yield results faster; however, you are limited to an incredibly small portion of the available repositories.

Listing 1. Find command used to search for LLM assisted repositories.
```
./strata find --languages Python --
    minimum-stars 100
```

3) Dataset Sampling: After the collection of both datasets, a sample of the data was created. This sample used a proportional sample size of 385 and the uniform sampler built into `strata`. Both samples were taken using the same seed (`719838331`) to ensure the samples were replicable for future work.

Listing 2. Reproducible sampling command for human created repositories.
```
./strata sample --seed 719838331 --csv-
    has-header --csv paper/data/large/
    datasets/human-repositories.csv
```

Listing 3. Reproducible sampling command for LLM-assisted repositories.
```
./strata sample --seed 719838331 --csv-
    has-header --csv paper/data/large/
    datasets/ai-assisted-repositories.csv
```

C. Repository Mining

The strata tool supports mining one repository at a time. To avoid major refactoring, this aspect of the tool was not altered. Instead, a secondary script was introduced `mine_all.py {human or ai}`. This tool runs the `strata` miner on every repository present in either the human or LLM-assisted dataset. The arguments passed to the miner differ slightly depending on the dataset being mined. The common set of arguments for both datasets is `--no-prs-issues` and `--no-ci`, since neither pull requests, issues, nor continuous integration is being investigated.

When mining the human repository, an end date for commits is added, `--ending-commit-year 2020`. This prevents commits from 2021 onward from appearing within the human dataset. The year 2020 was chosen, as it is far enough before the public introduction of LLMs (2022), as well as being within a decade of the introduction of AI agents. Like language, coding standards evolve over time; analyzing code with the smallest difference in time-delta is incredibly important for reducing the impact of standards drift.

Another limitation of the `strata` tool is that it does not support two datasets being present in the same database. Because of this, a `database` command was introduced which allows a user to switch between mined datasets.

Listing 4. The sequence of commands for mining Human repositories.
```
# Mine all human repositories
python3 mine_all.py human
# Dump the mined repositories.
./strata database --dump human.tar.gz
# Load a database for analysis
./strata database --reset --restore human
    .tar.gz
# Replace "human" with "ai" to run on the
    LLM-assisted dataset.
```

D. Analysis

1) Sampling: Without any sampling, each dataset contains upwards of 20 million data tokens. This is not an unreasonable amount of data for `strata` to handle; analysis can easily be run on this many tokens. Despite this, there are validity concerns for analyzing the entire database. Since one repository can contain significantly more tokens than others, there is a possibility of that singular repository skewing results. To mitigate this risk, I opted to analyze a stratified sample of identifiers, commits, and comments, using the source repository as the stratified group. A *proportion* sample size method produced a sample size of 385 for both the human and LLM dataset.

Listing 5. A command invocation for a sampled analysis.
```
./strata analyze --sample --sample-method
    stratified --seed 719838331
```

2) Linguistic Analysis: Six lexical diversity metrics are calculated after all tokens have been extracted from the sampled identifiers, commits, and comments. These metrics were selected based on the work from F. Zenker and K. Kyle [13]. Many metrics fail to produce a stable indicator of lexical richness as text size increases. To reduce the impact of unstable metrics, I have chosen the metrics that are most stable: Maas' Index [12], MATTR [16], and HD-D [17].

Stability across text size is critical for comparing the human and LLM-assisted datasets. Since the two datasets contain a different number of tokens, the stability across text lengths lets me compare the lexical diversity metrics without requiring the exact same text sizes.

3) Comparison: Comparing the vocabulary of human and LLM-assisted code addresses the core question of this paper. To start, I chose to compare the datasets using the clusters generated during `strata` analysis. The metrics I use are: cluster similarity, vector similarity, vocabulary overlap percentage, and shared vocabulary size. These metrics give a picture of the language map used by humans and AI. By analyzing the results, conclusions can be drawn about the categories of language used. I found that the best way to interpret the clusterings was by performing T-SNE dimensional reduction on the vocabulary vectors and graphing both datasets on the same graph. This produced three graphs: identifiers, comments, and commits.

I then performed a frequency comparison. To do this, I took the average number of repeats of every token within the dataset. This data allowed me to perform a Shapiro-Wilk test for normality, alongside a Mann-Whitney U test to generate a p-value. To help visualize the word frequency, I found a word cloud to work best for contextual interpretation of the data.

The next portion of the comparison analysis calculates the three lexical diversity metrics discussed earlier. These metrics are calculated individually and then compared in bar charts and a tabular output with specific values. The LexicalRichness Python package was chosen to perform these calculations [18].

To support all of these calculations, `strata` was extended with a `compare` subprogram. Invocation for this command is lengthy, and therefore not provided in this paper. See Data and Code VII for the GitHub repository, specifically the `REPRODUCTION.md` file for the command invocation.

IV. Evaluation

There is little, if any, difference in lexical diversity as seen across the three chosen metrics. See Table I for a full breakdown of the lexical diversity. Frequency analysis proved to be more decisive; *Comments*, showed a significant difference ($p = 4.952 - e28$) between human and LLM-assisted code (Table II). However, the commits and identifier categories did not present a significant difference.

The word clouds generated for Human (Figure 1), and LLM-assisted (Figure 2) code show the difference between popular tokens in each of the datasets. Figure 3 shows the difference in word embeddings between human and LLM-assisted code. Many of the embeddings appeared to be centered around the same points, but the LLM-assisted vocabulary tends to be more tightly grouped and has less of a spread than the human vocabulary. However, this is not quantifiable and therefore not addressed in my discussion.

V. Discussion

This paper asked, **Do human-created projects and AI-assisted projects have any significant differences in vocabulary?** I expected to find a significant difference between all parts of code vocabulary in projects created by humans versus LLMs. The research presented in this paper does not support this hypothesis.

TABLE I
Lexical Diversity of Tokens Used.

Category	Metric	Human	LLM	Delta
Identifiers	Maas' Index	0.011	0.011	0.000
Identifiers	MATTR	0.870	0.889	–0.019
Identifiers	HD-D	0.881	0.900	–0.019
Comments	Maas' Index	0.011	0.008	0.004
Comments	MATTR	0.753	0.843	-0.090
Comments	HD-D	0.763	0.836	-0.073
Commits	Maas' Index	0.018	0.018	0.000
Commits	MATTR	0.597	0.7292	-0.132
Commits	HD-D	0.951	0.967	-0.016

TABLE II
Average Token Repetition.

Category	Human	LLM	p-value
Identifiers	1.477	1.309	0.560
Comments	3.686	6.389	4.952-e28
Commits	1.580	1.657	0.171

In the commits and identifiers categories, the qualitative data from token repetition and lexical diversity did not indicate a significant difference between humans and LLM-assisted programming.

The comments category, however, shows a significant difference in the average repetition of tokens (Human: 3.686, LLM-assisted: 6.389, p-value: 4.952-e28). This indicates that when writing comments, LLM-assisted code will have noticeably different vocabulary used in comments. Individual words will be repeated 1.73x more often than if a human wrote a comment. This result can be generalized to python programs, and could be used to analyze a set of comments for their originator.

While not as conclusive as anticipated, these results do show that there is a difference in the vocabulary humans and LLMs use in code comments. A detection program should use this difference in cooperation with other metrics such as AST analysis [3].

At the present, the results of this paper do not have any practical implications for engineers currently in the field. No conclusive way to detect LLM-assisted code was found. However, this paper does show that linguistic analysis is a path forward for detecting the origin of a code comment.

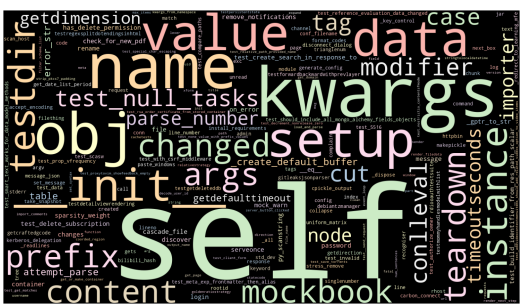

Fig. 1. Word cloud of comment tokens used in **human** written code.

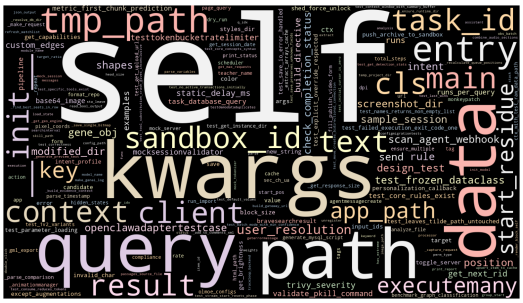

Fig. 2. Word cloud of comment tokens used in **LLM-assisted** code.

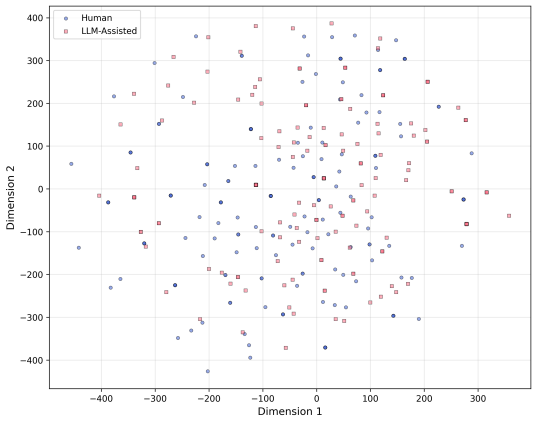

Fig. 3. Token embeddings for human and LLM-assisted **comment** tokens

Alongside other metrics, the work from this paper could aid in the creation of a tool that would help engineers better manage their time and secure their software by detecting what code came from a human versus an LLM.

VI. Threats to Validity

In this section, I identify potential threats to the validity of this paper.

A. Internal Validity

This paper assumes that the all code within the human dataset was created by humans, and all code in the LLM-assisted dataset was created with the assistance of an LLM. While the collection of human and LLM-assisted programs can be improved, these assumptions appear to be sound.

The human code dataset could be improved by limiting the mined code to code committed before the advent of the transformer model in 2017 [15]. This would remove the possibility of any LLM-generated code appearing in the human dataset. However, the current limit of 2020 is satisfactory to reasonably prevent the inclusion of LLM-assisted code while minimizing the time-delta. This is important because coding standards change over time, and these changes could skew results.

The LLM-assisted dataset faces the same kind of problem; it is not possible to ensure all mined code was written with LLM assistance. This problem can be solved in two ways: mining only code from commits authored only by an AI-Agent, and generating a dataset of projects using an AI-Agent specifically for this paper. Mining only AI-Agent-authored commits was considered an improvement to the `strata` program but was ultimately not included because of time constraints. Generating a dataset of AI-Agent code was avoided because of environmental concerns and monetary limitations.

B. External Validity

This paper only explores vocabulary used in Python projects. Results from this paper may not generalize to other programming languages, such as C, C++, or Java. Despite this lack in the current research, the `strata` tool is more than capable of running against more than just Python code. Future work could include improving the generalizability by expanding the programming languages analyzed.

C. Construct Validity

The lexical diversity metrics used in this paper are accurate to existing research in the area [8], and have been backed up by countless studies in lexical diversity [8], [12], [13], [16], [17], [19]–[21]. The chance for a threat to construct validity is incredibly small, and therefore not a concern for this paper.

D. Conclusion Validity

The statistical analysis in this paper is weak. Only a single output (the token frequency) was able to be checked against a statistical test. If future work is performed on the lexical differences between human and LLM-assisted code, this limitation should be addressed. I believe that a better approach would be the training of a classification model, similar to the work done by S. Bukhari et al. [3].

VII. Data and Code

The data and code for this paper are available on GitHub: https://github.com/ActuallyTaylor/strata. The repository contains a detailed `README` explaining the functionality and structure of the program. A file, `REPRODUCTION.md` is provided, detailing the exact commands to reproduce all data and figures in this paper.

VIII. Conclusion

Detecting whether code was created by a human or generated by an LLM is a currently unsolved problem. This research attempted to solve this problem through linguistic analysis of identifiers, commits, and comments mined from Python software repositories. Two datasets were created: one for human created code and the other for LLM-assisted code. These datasets were compared to find a difference in their vocabulary patterns.

This comparison presented a significant difference in the vocabulary humans and LLMs use to write code comments. Identifiers and commit messages did not present any significant difference.

As for future research to detect the difference between human-created and LLM-generated identifiers and commit messages, I suggest that a classification model should be

trained. Classification models are capable of deriving complex mathematical representations of relationships that humans are not easily capable of. This would mitigate many of the problems this paper faced while finding a proper mathematical representation of the difference in vocabulary.

Another future step would be to expand research to include analysis of a program's abstract syntax tree (AST). Classification of an AST has already proven to be slightly more accurate than lexical analysis and is the most promising path forward. [3]. This paper did not include any work on ASTs, as the goal was to find a mathematical difference that did not rely on classification models. Classification models require extremely large datasets, and can be easily biased by errors. Despite this, a combination of the techniques presented in this paper and AST analysis is the most promising direction for future research.

References

[1] N. Marques, R. S. Rodrigo, and J. Bernardino, "Using chatgpt in software requirements engineering: A comprehensive review," *Future Internet*, vol. 16, no. 6, p. 180, 2024.

[2] Anthropic, "Claude code github." https://github.com/anthropics/claude-code, April 2026.

[3] 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*, SCORED '23, (New York, NY, USA), pp. 17–25, Association for Computing Machinery, 2023.

[4] Anthropic, "Claude code attribution settings." https://code.claude.com/docs/en/settings#attribution-settings, April 2026, April 2026.

[5] A. C. AI, "Cursor ide documentation." https://cursor.com/help/integrations/git#how-does-agent-attribution-work, April 2026.

[6] 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, pp. 96–105, Jan. 2025.

[7] M. Vero, N. Mündler, V. Chibotaru, V. Raychev, M. Baader, N. Jovanović, J. He, and M. Vechev, "Baxbench: Can llms generate correct and secure backends?," 02 2025.

[8] P. Reviriego, J. Conde, E. Merino-Gómez, G. Martínez, and J. A. Hernández, "Playing with words: Comparing the vocabulary and lexical diversity of chatgpt and humans," *Machine Learning with Applications*, vol. 18, p. 100602, 2024.

[9] A. Boutadjine, F. Harrag, and K. Shaalan, "Human vs. machine: A comparative study on the detection of ai-generated content," *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, vol. 24, Feb. 2025.

[10] "Tree-sitter." https://tree-sitter.github.io/tree-sitter/, April 2026.

[11] "srcml." https://www.srcml.org/, April 2026.

[12] H.-D. Maas, "Über den zusammenhang zwischen wortschatzumfang und länge eines textes [on the relationship between vocabulary and the length of a text]," in *Zeitschrift für Literaturwissenschaft und Linguistik [Journal for Literary Studies and Linguistics]* (R. G. u. W. H. Herausgegeben von Helmut Kreuzer, Wolfgang Klein, ed.), vol. 08, 1972.

[13] F. Zenker and K. Kyle, "Investigating minimum text lengths for lexical diversity indices," *Assessing Writing*, vol. 47, p. 100505, 2021.

[14] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical software engineering : an international journal*, vol. 22, no. 6, pp. 3219–3253, 2017.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 06 2017.

[16] M. A. Covington and J. D. McFall, "Cutting the gordian knot: The moving-average type–token ratio (mattr)," *Journal of Quantitative Linguistics*, vol. 17, no. 2, pp. 94–100, 2010.

[17] T. Wu, "An accurate computation of the hypergeometric distribution function," *ACM Trans. Math. Softw.*, vol. 19, pp. 33–43, Mar. 1993.

[18] L. Shen, "Lexicalrichness: A small module to compute textual lexical richness," 2022.

[19] W. Johnson, "Studies in language behavior," *Psychological Monographs*, vol. 1: A program of research, pp. 56, 1–15., 1944.

[20] P. Guiraud, "Problèmes et méthodes de la statistique linguistique," *(No Title)*, 1959.

[21] J. W. Chotlos, "A statistical and comparative analysis of individual written language samples.," *Psychological Monographs*, vol. 56, no. 2, p. 75, 1944.

Generative AI Usage

This paper used Generative AI for spelling and grammar checking. The specific tool used was *Apple Intelligence Writing Tools: Proofread*.