Unlocking Code Simplicity: The Role of Prompt Patterns in Managing LLM Code Complexity

Antonio Della Porta, Gilberto Recupito, Stefano Lambiase, Dario Di Nucci, Fabio Palomba
SeSa Lab - University of Salerno, Fisciano, Italy
{adellaporta, grecupito, slambiase, ddinucci, fpalomba}@unisa.it

***Abstract*—The rapid growth of generative artificial intelligence, especially Large Language Models (LLMs), has greatly influenced software engineering by automating code generation tasks. Despite the potential, challenges in code maintainability and quality persist, mainly due to prompt design. This study examines how various prompt patterns influence the complexity of Python code generated by LLMs, using the Dev-GPT dataset. Four prompt patterns were analyzed: Zero-shot, Few-shot, Chain-of-Thought, and Personas. Complexity metrics assessed include Lines of Code (LOC), Cyclomatic Complexity, and Halstead metrics, with statistical analyses conducted using the Kruskal-Wallis test and post-hoc pairwise comparisons. Results showed significant differences in LOC and related sub-metrics among these prompt patterns. Notably, the Chain-of-Thought pattern consistently generated more concise and efficient code, offering strategies to enhance LLM-generated code quality and maintainability.**

***Index Terms*—Prompt Engineering; Prompt Patterns; Code Complexity; Empirical Software Engineering.**

I. Introduction

The rapid advancement of generative artificial intelligence (AI) and, in particular, Large Language Models (LLMs) has initiated a transformative shift in various domains, including software engineering [1], [2]. Within this field, LLMs demonstrate significant potential by automating code generation, assisting in component development, and supporting decision-making processes [2]. Despite these advancements, the adoption of LLMs in software development workflows remains fraught with challenges, particularly regarding the quality and consistency of generated code [3], [4].Furthermore, the complexity of AI-generated source code, an important determinant of its maintainability, remains underexplored, particularly in the context of systematic prompt engineering strategies.

While existing research highlights the utility of prompt patterns in improving LLM performance [5], [6], current studies often fail to comprehensively address their influence on critical software quality attributes, such as code complexity.

This study investigates how different prompt patterns affect the complexity of Python code generated by large language models (LLMs). Python was chosen for its prominence in AI and ML development, where maintaining reliable and efficient code is critical. The research focuses on complexity metrics—such as Lines of Code, Cyclomatic Complexity, and Halstead measures—across four prompt categories: Zero-shot, Few-shot, Chain-of-Thought, and Personas. Using an empirical approach, the study analyzed whether prompt patterns significantly influence code complexity, aiming to improve the maintainability of ML-enabled systems.

The findings revealed that the use of different prompt patterns significantly influences size-related metrics, such as *SLOC*, *Multi*, and *Blank* lines, while no significant impact was observed on the number of *comments*. Specifically, the Chain-of-Thought pattern demonstrated its effectiveness in generating concise and efficient code through step-by-step reasoning, making it a suitable approach for improving code readability and maintainability. In contrast, the exclusive use of Few-Shot tended to produce verbose or unnecessarily complex code, potentially introducing quality issues.

II. Background and Related Work

Research on source code complexity often revolves around both traditional and advanced metrics. Object-oriented metrics, such as Chidamber and Kemerer (CK) metrics, lines of code (LOC), McCabe's cyclomatic complexity (CC), and counts of methods and attributes, are commonly employed in the context of object-oriented programming [7]. Entropy-based measures are also used to quantify code change complexity and predict future challenges [8]. While CC and LOC remain widely used for internal quality assessment, debates persist over their redundancy due to strong linear correlations [9].

Metrics play a pivotal role during the design stage, offering early predictions of code quality by evaluating specifications prior to development. This approach can reduce development time by highlighting potential issues early [10]. Studies on open-source projects reveal that complexity evolves over time, aligning with Lehman's laws of software evolution, and often shifts between design hierarchy levels [11].

LLMs have demonstrated significant potential in automating code generation, though their effectiveness varies depending on the complexity of tasks. Models such as GPT-4 and GPT-3.5 have shown improved performance in code generation when holistic strategies are employed, whereas incremental approaches yield better results for other models. Incorporating programming practices, such as creating high-level sketches before implementation, has further enhanced the performance of LLMs [12]. Sasaki et al. [13] introduced the concept of *prompt engineering patterns*, describing them as a *"systematic approach to structuring interactions, providing a versatile framework applicable across various domains"*. White et al. [14] highlighted the importance of prompt engineering in software engineering and reported that effective, prompt usage improves the early stages of the software development lifecycle. A significant contribution to the field was made by

Xiao et al. [4], who developed DEV-GPT, a dataset aimed at studying how developers interact with CHATGPT in software development contexts. In light of these prior studies, we complement existing research by focusing on the influence of prompt patterns on quality-related metrics describing the complexity of generated source code.

III. OBJECTIVE AND RESEARCH QUESTION

The *goal* of this work is to determine the extent to which specific prompt patterns influence the complexity of source code generated by LLMs, with the *purpose* of expanding current knowledge in prompt engineering and potentially uncovering the side effects of code generation.

To reach the defined goal and test the working hypothesis, we formulated a research question (RQ) whose main aim is to shape and guide the research process.

> **RQ** — *Are there significant differences in the complexity of Python source code generated by LLMs when using varying prompt patterns?*

In order to represent complexity, we operationalized five well-known metrics, i.e., Lines of Code (LOC), Cyclomatic Complexity, Volume, Difficulty, and Effort (deepened in Section IV-A). Thus, we formulated the following hypotheses:

- **Null Hypotheses:** There are no significant differences in ($H1_0$) LOC, ($H2_0$) Cyclomatic Complexity, ($H3_0$) Volume, ($H4_0$) Difficulty, and ($H5_0$) Effort complexity metrics of the Python source code generated by LLMs based on the prompt pattern used.
- **Alternative Hypotheses:** There are significant differences in ($H1_A$) LOC, ($H2_A$) Cyclomatic Complexity, ($H3_A$) Volume, ($H4_A$) Difficulty, and ($H5_A$) Effort complexity metrics of the Python source code generated by LLMs based on the prompt pattern used.

The research question defined serves to guide the statistical analysis of the potential relationship between different prompt patterns and the complexity of generated Python source code. Addressing this question is essential for achieving the study's overall objective, as it provides insights into how specific prompt patterns may affect the overall complexity of code generated by LLMs.

IV. RESEARCH DESIGN

The study analyzed developer-LLM conversations by cleaning the dataset, examining prompt patterns, formatting code snippets for analysis, and using static analysis and statistical methods to calculate code complexity metrics and answer the research question.

A. Variables of the Study

The independent variable was operationalized using a categorical scale consisting of four prompt pattern categories and their combination, frequently discussed in the literature:

- *Zero-shot* (ZS) [15]: It involves the use of a specific prompt with no examples but relies on the knowledge of the model based on training data to perform the task.
- *Few-shot* (FS) [16]: It describes the addition of task execution examples within the prompt to increase the level of detail that the model can use, providing solutions to perform the target task.
- *Chain-of-Thought* (CoT) [17]: it includes a step-by-step reasoning description of the logical flow to execute the task. It enhances the model's logical consistency and detail.
- *Personas* (Per) [3]: It requests to "interpret" a specific character within the prompt, helping the model to comprehend the referring context and perspective of the role.

Regarding the dependent variable, we used all the complexity metrics made available by the RADON[1] library, well-known for analyzing Python code. Specifically, we used:

- *Cyclomatic Complexity* (CC): The number of decisions a block of code contains plus 1.
- *Halstead Volume* (V): It is the actual size of a program if a uniform binary encoding for the vocabulary is used.
- *Difficulty* (D): The difficulty level or error-proneness of the program is proportional to the number of unique operators in the program.
- *Effort* (E): The effort to implement or understand a program is proportional to the volume and to the difficulty level of the program.
- *Lines of Code* (LOC): The total number of lines of code. This metric is calculated as the sum of *Source Lines of Code* (SLOC), *Multi-lines strings* (M), *Comments* (CO), and *Blank lines* (B).

B. Dataset

The DEV-GPT dataset [4] comprises developer interactions with CHATGPT across various software development tasks, collected using OPENAI's conversation-sharing feature. It includes 6 `json` files representing data from sources like GITHUB issues, pull requests, commits, code files, and HACKER NEWS threads. The dataset, containing 5494 conversation rounds and 29 788 prompts, spans 13 988 code snippets across 113 programming languages. For this study, only Python-related interactions were analyzed. Conversations with mixed-language code were filtered to include only Python segments. This selection process yielded 329 conversations and 2405 Python code files for analysis.

C. Data Processing and Analysis

To address our research question, the analysis began with the cleaned dataset and proceeded through the steps necessary to identify prompt patterns. Specifically, the identification of prompt patterns in user conversations was conducted using the capabilities of ChatGPT 4o-mini (snapshot *2024-07-18*). Following the automated classification by the LLM, the first three authors manually reviewed a sample of the results to ensure the accuracy of the classification process.

Then, all Python snippets were converted into code files, and static analysis was performed to calculate code complexity

[1] https://pypi.org/project/radon/

TABLE I: Kruskal Wallis for **LOC** metrics.

Factor	Statistic	df	p	Rank ε^2
SLOC	19.476	5	0.002	0.009
Multi	13.527	5	0.019	0.006
Comments	9.390	5	0.095	0.004
Blank	16.270	5	0.006	0.007

Note. Factor: Variable tested, Statistic: Test statistic (H), df: Degrees of freedom, p: Significance level, Rank ε^2: Effect size.

TABLE II: Descriptive Statistics.

Variable	Prompt Pattern	Median	Mean	Std. Deviation
SLOC	ZS-CoT	7.000	10.089	10.539
	FS	11.000	15.954	16.168
	FS-Per	5.500	8.800	8.788
Multi	ZS	0.000	0.180	1.675
	ZS-CoT	0.000	0.400	1.517
Blank	ZS-CoT	1.000	2.578	3.329
	FS	3.000	4.160	4.695

metrics using the RADON library. The metrics described in Section IV-A were extracted, and the dataset was augmented with this additional information for further analysis.

Subsequently, statistical analyses were performed on the dependent variables across various sets of conversations, categorized based on prompt patterns. Due to violations of ANOVA assumptions, the non-parametric Kruskal-Wallis test was utilized, followed by Dunn's post hoc test for pairwise comparisons. A significance threshold of $\alpha = 0.05$ was applied throughout the analysis. Since some of our metrics (e.g., LOC) are themselves composed of finer-grained sub-metrics, in cases where the primary metrics were found to be significant, the same analysis was conducted for their sub-metrics. The analysis was conducted using the JASP statistical analysis software.[2]

V. ANALYSIS OF THE RESULTS AND DISCUSSION

In this section we present the results of our analysis.[3]

The first part of the analysis involved using kruskal wallis to test significant differences between difference prompt patterns for the following source code metrics: LOC, Cyclomatic Complexity, Volume, Difficulty, and Effort. The instances for ZS-CoT-Per (5 instances) and FS-CoT-Per (4 instances) were excluded from consideration due to their insufficient sample size, as both contained fewer than 10 instances.

The test revealed a statistically significant difference for LOC ($\rho = 0.003$), with a small effect size (Rank $\varepsilon^2 = 0.008$). However, no statistically significant differences were observed for Cyclomatic Complexity ($\rho = 0.380$), Volume ($\rho = 0.354$), Difficulty ($\rho = 0.374$), or Effort ($\rho = 0.369$). The effect sizes for these metrics were minimal (Rank $\varepsilon^2 = 0.002$ for each).

These results suggest that while no significant differences are found across prompt patterns for the other metrics, LOC varies significantly across the use of different prompt patterns. For the sake of readness and clarity, we reported only statistically significant differences among our variables. More details on other relationships found in this study are available in our online appendix [18].

Since LOC was found to be significant, a deeper analysis was conducted on its sub-measures, as shown in Table I. The sub-measures selected are SLOC, Multi, Comments, and Blank, described in Section IV-A.

The results indicate that SLOC ($\rho = 0.002$, Rank $\varepsilon^2 = 0.009$), Multi ($\rho = 0.019$, Rank $\varepsilon^2 = 0.006$), and Blank ($\rho = 0.006$, Rank $\varepsilon^2 = 0.007$) were statistically significant, with small effect sizes. Conversely, Comments did not show a statistically significant difference ($\rho = 0.095$, Rank $\varepsilon^2 = 0.004$). These findings suggest that differences in LOC across prompt patterns are primarily driven by variations in SLOC, Multi-line strings, and Blank lines, while comments are relatively consistent across groups. **For this reason, $\mathbf{H1_A}$ is considered supported**.

To further investigate the significant results identified for the SLOC, Multi, and Blank metrics, a post-hoc Dunn test was conducted.

- The Dunn test results identified statistically significant differences in the SLOC metric among specific group comparisons. After applying the Holm correction to account for multiple comparisons, significant differences were observed between ZS-CoT and FS ($\rho_{\text{holm}} = 0.022$) and between FS and FS-Per ($\rho_{\text{holm}} = 0.021$). These findings suggest meaningful distinctions in SLOC values between these pairs of groups. Notably, the effect size for FS vs. FS-Per was substantial ($rrb = 0.603$), indicating a considerable difference, while the effect size for ZS-CoT vs. FS ($rrb = 0.233$) was smaller but still noteworthy.
- Regarding the Multi metric, the results indicate one statistically significant difference. Specifically, the comparison between ZS and ZS-CoT ($\rho_{\text{holm}} = 0.006$) is significant, with a small effect size ($rrb = 0.051$). This result suggests a meaningful distinction between these groups in the Multi metric, though the effect size indicates the difference is relatively minor.
- The Dunn test results for the Blank metric indicate one statistically significant difference after applying the Holm correction. Specifically, the comparison between ZS-CoT and FS ($\rho_{\text{holm}} = 0.045$) is significant, with a small effect size ($rrb = 0.206$). This suggests a meaningful difference between these groups in the Blank metric.

The Dunn test results revealed statistically significant differences across multiple metrics. For the SLOC metric, notable distinctions were identified between ZS-CoT and FS, as well as between FS and FS-Per, with varying effect sizes indicating both moderate and substantial differences. In the Multi metric, a significant but minor distinction was observed between ZS and ZS-CoT. Similarly, the Blank metric highlighted a meaningful, albeit small, difference between ZS-CoT and FS. These findings suggest that differences across groups vary in magnitude depending on the metric analyzed, with some com-

[2]JASP website: https://jasp-stats.org

[3]Detailed results are reported in our online appendix [18].

parisons showing stronger effects than others. We analyzed the effects of different prompt patterns on code generation metrics, focusing on size-related aspects. The results show that the ZS-CoT pattern produces significantly shorter code compared to FS (median: 7 vs. 11 SLOC), indicating greater efficiency. The FS-Per pattern further reduces code length (median: 5.5 SLOC) compared to FS, though it is not significantly different from ZS-CoT. Regarding blank lines, ZS-CoT generates fewer than FS, with a median of one blank line. Additionally, ZS-CoT includes fewer multi-line strings than ZS, demonstrating differences in how patterns influence code structure. The analysis demonstrates significant differences in how various prompt patterns influence the complexity of the generated code, particularly in terms of size-related metrics. Furthermore, the choice of prompt pattern plays a crucial role, depending on the specific goals of the user when generating LLM responses.

One major finding of the study highlights the fact that the application of the Zero-Shot combined with the CoT prompt pattern consistently results in shorter code compared to other prompt patterns, highlighting its effectiveness, so practitioners should prefer those patterns to generate more short and maintainable code.

VI. Limitations and Threats to Validity

This study acknowledges several threats to validity, identified based on its design.

Regarding *Construct Validity*, the classification of prompt patterns into categories such as Zero-shot, Few-shot, Chain-of-Thought, and Personas may oversimplify the nuanced ways in which these patterns influence code generation. Although the categorization was informed by established literature, it may not fully account for variations in pattern design and implementation. Moreover, the complexity metrics of source code were calculated using the RADON library and so the findings are heavily dependent on the accuracy and implementation of the metrics as computed by this tool.

With respect to *External Validity*, the reliance on a single dataset of conversations (DEV-GPT) obtained by the use of a single LLM (Chat-GPT) may limit the applicability of the results to other datasets or language models. Future research should extend the analysis to include additional datasets to validate the findings in broader contexts.

Finally, for *Conclusion Validity*, the use of the Kruskal-Wallis test, a non-parametric alternative to ANOVA, mitigated issues related to normality and variance assumptions. However, rank-based analysis can limit the detection of subtle effects. To enhance robustness, significance thresholds were carefully set to reduce the risk of Type I and II errors, thereby increasing the reliability of the study's conclusions.

Acknowledgment

This work has been partially supported by the *Qual-AI* national research project, which has been funded by the MUR under the PRIN 2022 program (Code: D53D23008570006).

References

[1] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

[2] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 2023, pp. 31–53.

[3] A. Kong, S. Zhao, H. Chen, Q. Li, Y. Qin, R. Sun, X. Zhou, E. Wang, and X. Dong, "Better zero-shot reasoning with role-play prompting," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024, pp. 4099–4113.

[4] T. Xiao, C. Treude, H. Hata, and K. Matsumoto, "Devgpt: Studying developer-chatgpt conversations," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 2024, pp. 227–230.

[5] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," in *Generative AI for Effective Software Development*. Springer, 2024, pp. 71–108.

[6] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "Prompt tuning in code intelligence: An experimental evaluation," *IEEE Transactions on Software Engineering*, vol. 49, no. 11, pp. 4869–4885, 2023.

[7] A. S. N. Varela, H. G. Pérez-González, F. E. M. Pérez, and C. Soubervielle-Montalvo, "Source code metrics: A systematic mapping study," *J. Syst. Softw.*, vol. 128, pp. 164–197, 2017.

[8] K. K. Chaturvedi, P. K. Kapur, S. Anand, and V. B. Singh, "Predicting the complexity of code changes using entropy based measures," *International Journal of System Assurance Engineering and Management*, vol. 5, pp. 155–164, 2014.

[9] D. Landman, A. Serebrenik, and J. Vinju, "Empirical analysis of the relationship between cc and sloc in a large corpus of java methods and c functions," *Journal of Software: Evolution and Process*, vol. 28, pp. 589 – 618, 2016.

[10] S. Henry and C. Selig, "Predicting source-code complexity at the design stage," *IEEE Software*, vol. 7, pp. 36–44, 1990.

[11] R. Sangwan, P. Vercellone-Smith, and P. Laplante, "Structural epochs in the complexity of software over time," *IEEE Software*, vol. 25, pp. 66–73, 2008.

[12] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation," *ArXiv*, vol. abs/2308.01861, 2023.

[13] Y. Sasaki, H. Washizaki, J. Li, D. Sander, N. Yoshioka, and Y. Fukazawa, "Systematic literature review of prompt engineering patterns in software engineering," in *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2024, pp. 670–675.

[14] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, *ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design*. Cham: Springer Nature Switzerland, 2024, pp. 71–108. [Online]. Available: https://doi.org/10.1007/978-3-031-55642-5_4

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[16] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, vol. 1, 2020.

[17] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[18] A. Della Porta *et al.*, "Unlocking code simplicity: The role of prompt patterns in managing llm code complexity." [Online]. Available: https://figshare.com/s/b616f6ed0ee8a2e6cca2