A Survey Study on the State of the Art of Programming Exercise Generation using Large Language Models

1st Eduard Frankford
Computer Science
University of Innsbruck
Innsbruck, Austria
eduard.frankford@uibk.ac.at

2nd Ingo Höhn
Computer Science
University of Innsbruck
Innsbruck, Austria
ingo.hoehn@student.uibk.ac.at

3rd Clemens Sauerwein
Computer Science
University of Innsbruck
Innsbruck, Austria
clemens.sauerwein@uibk.ac.at

4th Ruth Breu
Computer Science
University of Innsbruck
Innsbruck, Austria
ruth.breu@uibk.ac.at

***Abstract*—This paper analyzes Large Language Models (LLMs) with regard to their programming exercise generation capabilities. Through a survey study, we defined the state of the art, extracted their strengths and weaknesses and finally derived an evaluation matrix, helping researchers and educators to decide which LLM is the best fitting for programming exercise generation. The findings reveal that multiple LLMs are capable of producing useful programming exercises. Nevertheless, there exist several challenges like the ease with which LLMs might solve exercises generated by LLMs and their limited creative capacity. The proposed evaluation matrix offers a structured approach to assessing LLMs and contains three main factors: (1) A general assessment, (2) Program Analysis and (3) Qualitative Assessment. This paper contributes to the ongoing discourse on the integration of AI in education, offering insights into the capabilities and limitations of LLMs in improving programming education.**

***Index Terms*—Programming Education, Programming Exercise Generation, Large Language Models, Artificial Intelligence**

I. Introduction

The advent of the mainstream use of LLM based applications such as ChatGPT has marked a significant milestone in the field of artificial intelligence. State of the art LLMs have demonstrated remarkable capabilities in a variety of applications, ranging from feedback creation to code generation. In the realm of programming education, the potential of LLMs to automate and enhance the generation of learning materials, such as programming exercises, is a promising area of exploration. Despite a significant number of publications, a dedicated survey study that examines and compiles these first attempts at generating programming exercises is still missing.

This research seeks to address this gap by analyzing and synthesizing state-of-the-art literature about the application of LLMs in the field of programming exercise generation. This study is therefore guided by the following research questions:

- **RQ1:** What is the current state of the art in programming exercise generation using LLMs?
- **RQ2:** What are the strengths and weaknesses of existing solutions in programming exercise generation using LLMs?
- **RQ3:** How can different LLMs be evaluated regarding their suitability for exercise generation?

The significance of this research lies in its potential to contribute to the optimization of educational resources in programming education, by offering insights into the capabilities and limitations of LLMs in creating diverse, innovative, and pedagogically sound programming exercises.

The structure of this paper is organized as follows: Section II offers a review of the related literature. Section III details the methodologies employed in the research. The core findings of the study are outlined in Section IV, with a deeper discussion provided in Section V. Key limitations of the study are addressed in Section VI, and the paper concludes with Section VII, encapsulating the primary insights and discussing the wider implications of the study.

II. Related Work

The utilization of LLMs across various domains, including software engineering and education in general, has been extensively documented, with significant contributions from Hou et al. [1], Hadi et al. [2], and Yan et al. [3]. Hou et al. categorize LLMs' applications in software engineering, highlighting their adaptability, a feature crucial for educational applications like programming exercise generation. Hadi et al. extend this perspective by exploring LLMs' broader implications, challenges, and architectures, emphasizing their transformative potential and ethical considerations. Yan et al. conducted a comprehensive review on the ethical and practical challenges of LLMs in education, revealing their ver-

satility across various educational tasks, from content generation to feedback provision.

While these studies collectively describe the capabilities, applications, and challenges of LLMs in software engineering and education, there is still no survey study specifically addressing the generation and benchmarking of programming exercises using LLMs. This research aims to bridge this gap by focusing on the potential of LLMs to automate programming exercise generation and provide an evaluation matrix to help educators choose the right LLM for their programming exercise generation use case.

III. Methodological Approach

This study made use of the methodological approach of a survey study to assess the current state of the art in programming exercise generation using LLMs. The methodology is designed to offer a broad view of the field, highlighting significant contributions and identifying potential areas for future research. The approach encompasses the following key components.

A. Literature Collection Strategy

As search strategy a database search was conducted. Since the topic is novel, a Google Scholar search was included alongside classical databases. The classical databases include: (1) ACM Digital Library, (2) IEEE Explore, (3) ScienceDirect, (4) Springer Link.

After searching the classical databases and Google Scholar, a round of snowballing was conducted.

B. Search Terms

The search terms were derived from the core topics of interest: LLMs and programming exercise generation. A combination of keywords related to "Large Language Models", "Programming Exercise Generation" and "Educational Technology" was used to capture the relevant literature. The search was iteratively refined to include emerging terms and synonyms to ensure comprehensiveness.

C. Inclusion & Exclusion Criteria

In alignment with Garousi et al.'s [4] guidelines, this survey study adopts inclusion and exclusion criteria to ensure the relevance and timeliness of the literature reviewed. We focus on studies published from 2018 to 2023, reflecting the rapid developments in AI and the emergence of programming exercise generation using LLMs around 2022. We only selected studies that offer insights into LLM applications in programming exercise generation and are accessible in full text. Last but not least, only articles available in English were considered.

D. Stopping Criteria

For white literature as well as gray literature, the stopping criterion of data exhaustion was used. In practice, this survey study, examined the entire body of available literature found for the defined search string. Given the limited scope of existing research, all relevant studies were covered before reaching a point of data saturation. This point is supported by the fact that even snowballing did not add new sources to the body of included literature.

IV. Results

This section presents the findings from the survey study.

A. RQ1: Current State in Programming Exercise Generation

Working implementations for programming exercise generation exist and produce sensible results [5]–[8]. The study by Sarsa [5] offers the most detailed results, claiming LLMs are capable of generating sensible, novel, and readily applicable programming exercises. Of 120 investigated exercises in the study, 75% were sensible, 81.8% were novel, and 76.7% had a matching sample solution. Denny [7] reported similar results, which is expected since they used the same underlying model (Codex) and applied it to a similar task. Freitas, Haluptzok et al. [8] generated programming exercises, not with the intention of providing them to humans, but rather for model training purposes. The purpose was to enhance the code generation capabilities of the tested model through the creation and solving of programming exercises.

One reoccurring technique is the decomposition of exercise parts into: (1) problem statement, (2) template code, (3) solution and (4) test cases. This logical deconstruction of the exercise also overlaps with the creation of programming exercises in Automated Programming Assessment Systems. For exercise generation, this allows to reduce the context size and increases the likelihood of generating a more precise result [9]. Freitas [6] used two different models, for different components of the exercise. For the problem statement, the Google T5 model was used, while for the generation of template code, Google CodeT5 was used. According to the researchers, this demonstrated equally capable results to OpenAI's GPT-3 model, while only using open source models.

In general, programming exercise generation systems have been using various LLMs. Sarsa and Denny [5], [7] utilized OpenAI Codex [10], which is based on OpenAI's GPT-3 [11] architecture but fine-tuned specifically for programming tasks, originally powering the GitHub Copilot. Freitas [6], in contrast, employed Google T5 and CodeT5 models. Haluptzok et al. [8] experimented with the open-source GPT-Neo [12] model in its various forms (125M, 1.3B, and 2.7B versions) along with OpenAI Codex. GPT-Neo is similar to GPT-3 and was trained

on the Pile dataset [13], which includes a substantial amount of GitHub code, providing a robust foundation for programming-related tasks.

B. RQ2: Strengths and Weaknesses of Existing Solutions

The main advantage of automated exercise generation lies in its remarkable ability to create high-quality learning material in a time efficient way. The traditional process of developing educational resources is labor-intensive and often demands a considerable degree of expertise. This efficiency opens up unprecedented possibilities for creating an extensive array of novel learning resources, including detailed code explanations and comprehensive code examples, on a virtually limitless scale [14].

Haluptzok et al. [8] underscore the significant benefits of fine-tuning in the context of programming exercise generation. Specifically, fine-tuning Neo models on verified synthetic puzzle-solution pairs resulted in a two to five times improvement in puzzle-solving capabilities compared to the baseline model. This demonstrates a key strength of LLMs, a model with the general capacity of, for example GPT-4, when fine-tuned for programming exercise generation, could likely lead to impressive results.

Additionally, generated exercises, even if imperfect, can serve as a starting point for further modification by teachers or as a basis for student activities, like code reviewing and debugging [5]. Similar observations are made by [15], [16]. On the one hand, they emphasize personalizing the context of exercises, making them more engaging to the respective student. On the other hand, programming exercise generation enables more granular scaling by difficulty.

A weakness mentioned by Sarsa [5] is the quality of the generated tests or test suites. Less than a third of exercises with tests pass successfully, and only about 70% of the exercises include tests at all. Therefore, the general precision of the generation method has still to be improved.

Another significant challenge, mentioned by [7], is that exercises generated using LLMs seem to be easily solvable by LLMs as well. This aspect introduces the risk of creating a counterproductive cycle where the exercises produced by LLMs fail to adequately challenge students. This is because students might find it tempting to simply input these exercises into a LLM-based application, such as ChatGPT, and obtain solutions with minimal effort.

Freitas [6] mentioned a quality trade-off between costly commercial models like GPT-3, GPT-4 and freely accessible open source models, like Google T5 or GPT-Neo.

Other studies [14]–[16] highlight disadvantages of LLMs such as over-reliance on the student as well as the teacher side, ethical issues, code reuse and licensing issues.

C. RQ3: Evaluating LLMs Suitability for Exercise Generation

Most benchmarks for LLMs are based on a set of problems the LLM is expected to solve, and they are then evaluated by the percentage of correctly solved problems [10], [11], [17], [18]. As the models are getting better, there has been the trend of changing benchmarks to contain more complex problems [17], [19]. Earlier benchmarks like the *Winograd Schema Challenge* are being phased out since recent models neared human-level performance [17]. Improved benchmarks, like the *MMLU* [17], include harder and more specialized subjects. This aims to measure real-world text understanding, by evaluating the model's ability to extract knowledge from its extensive training corpus.

In programming, the *HumanEval* metric, introduced in the Codex paper by OpenAI researchers [10], is a well-established benchmark [18]. A subsequent study comparing several LLMs' coding abilities also based its evaluation on *HumanEval* [20]. Liu et al. [21] proposed *EvalPlus*, which builds upon *HumanEval*, adding the ability to detect more incorrectly synthesized code.

During the survey study, we have found both manual and automated LLM assessment approaches. In [5] they evaluated the generated exercises both manually and automatically. Regarding manual assessment, the study presents four metrics: (1) Sensibleness, (2) Novelty, (3) Topicality and (4) Readiness for Use. *Sensibleness* is defined as whether the problem could be given to students to solve. *Novelty* is true if the programming exercise is not findable in Google or Github. *Readiness of use* considered the amount of manual work needed by a teacher before being able to use the generated problem. The concept, *topicality*, is related to *novelty* and examines how well a concept given in the prompt is accounted for in the created exercise.

The studies by Sarsa and Denny [5], [7] also reported five metrics that can be evaluated automatically: (1) Has sample solution, (2) Sample solution is executable, (3) Has tests, (4) All tests pass and (5) Test coverage.

The general drawbacks of a manual evaluation are subjectivity, bias and cost when compared to automated evaluations [10].

Having analyzed multiple approaches for benchmarking in general, it is important to derive a benchmark tailored to benchmarking programming exercise generation instead of general programming abilities of LLMs.

As a first step, it is important to define the key requirements of programming exercise generation. Only then, metrics can be established to measure model performance. Defining what constitutes a good exercise, which

results in effective learning outcomes for students is challenging. Typically, high-quality exercises share characteristics such as a well-defined problem statement, good structure, a specific competency focus and a progressively increasing difficulty for sub-tasks. Therefore, to generate a programming exercise the LLM must provide a problem statement, template code, a sample solution and test cases for the exercise. Additionally, for the use in higher education, costs need to be taken into account, because decomposing the exercises and including previously generated exercise parts in the prompt, can lead to expensive requests when using paid models like GPT-4 [6]. The constructs of novelty and readiness of use have been proposed before and should also be taken into account to measure the quality of the generated exercises [5], [7]. Additionally, a viable approach to establish a quantitative benchmark for programming exercise generation using LLMs could be a match-based multiple choice approach, similar to the *hellaSWAG* benchmark [22]. The *hellaSWAG* dataset evaluates the ability to complete unfinished sentences with multiple choice questions. Similarly, a dataset could be developed consisting of partial exercises. The model would then need to determine which of the provided answers best completes the exercise. This approach could be implemented as an automated benchmark, and named the *Programming Exercise Generation (PEG) Benchmark*.

A more complete picture of LLM performance for programming exercise generation can be formed when evaluating it using the evaluation matrix presented in table I. This takes into account general dimensions like cost, ability to generate code and data privacy, as well as specialized benchmarks like the before defined *PEG Benchmark* and constructs like sensibility, novelty and readiness of use. Using this matrix, educators may form a more evidence-based choice for a LLM to power their programming exercise generation service.

TABLE I: Evaluation Matrix

	LLM
General Assessment	
Costs / Semester	Numeric
Data Privacy	Boolean
HumanEval Score	Percentage
PEG Benchmark	Percentage
Program Analysis	
Has sample solution	Percentage
Runnable sample solution	Percentage
Test Cases are present	Percentage
All tests pass	Percentage
Test coverage	Percentage
Qualitative Assessment	
Sensibility	Percentage
Novelity	Percentage
Readiness of Use	Percentage

V. Discussion

The results presented in Section IV provide a starting point to better understand the current landscape and potential of generative AI in programming education, particularly in the context of programming exercise generation.

The current state of the art in programming exercise generation (RQ1), with Sarsa [5], Freitas [6], Denny [7], and Haluptzok et al. [8], reveal a promising trend towards the practical application of LLMs to improve educational resource creation. These studies demonstrate the capability of LLMs to produce novel, sensible, and ready-to-use programming exercises, which could significantly alleviate the workload on educators and enable personalized learning experiences for students. The use of various LLMs, including OpenAI Codex, Google T5, CodeT5, and GPT-Neo and GPT-3 reflects a diverse technological landscape.

In RQ2 we analyzed the strength and weaknesses of existing solutions and identified that the main strength of LLMs lays in their efficiency in content creation. However, the highlighted weaknesses, such as the large failure quote of generated test suites and the potential for generated exercises to be solvable by LLMs themselves, suggest areas for improvement.

Last but not least, by answering RQ3 we found that there exist both automated and manual assessment approaches, which reflect the complexity of assessing AI-generated content's educational value. We further proposed PEG Benchmark and a detailed evaluation matrix, which help to structure benchmarking LLMs.

VI. Limitations

This study's comprehensiveness might be limited by the search strategy and terms used to gather relevant publications. Although a broad range of keywords and search techniques, including snowballing, were employed to ensure an extensive collection, there's still a possibility that some important studies were overlooked. Notably, the snowballing process did not reveal any new documents, suggesting the initial search was thorough.

Additionally, the initial screening and selection of studies were primarily conducted by a single individual, raising concerns about potential subjective bias in choosing which studies to include. This risk was partially mitigated through feedback from other researchers, yet the potential for bias cannot be fully discounted.

VII. Conclusion

This study has investigated the use of LLMs for generating programming exercises, revealing their potential to revolutionize programming education. While LLMs

like Codex and GPT-3 can create engaging and novel exercises, these often require educator adjustments to make them usable in the classroom. However, the main advantages include significant time savings for educators and the ability to easily tailor exercises to student needs.

Despite this, the reliance on LLMs poses risks of diminishing exercise quality and student learning outcomes, especially when LLM generated exercises seem again to be easily solvable by LLMs.

To navigate the evolving landscape of LLMs in education, we proposed an evaluation matrix to offer a structured approach for assessing LLM-generated exercises, considering factors like cost, data privacy, HumanEval Scores, and the PEG Benchmark.

Looking ahead, regularly using the evaluation matrix to assess emerging LLMs will be crucial in guiding educators and researchers to identify the most effective models for their current needs.

References

[1] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *arXiv preprint arXiv:2308.10620*, 2023.

[2] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, S. Mirjalili *et al.*, "A survey on large language models: Applications, challenges, limitations, and practical usage," *Authorea Preprints*, 2023.

[3] L. Yan, L. Sha, L. Zhao, Y. Li, R. Martinez-Maldonado, G. Chen, X. Li, Y. Jin, and D. Gašević, "Practical and ethical challenges of large language models in education: A systematic scoping review," *British Journal of Educational Technology*, vol. 55, no. 1, pp. 90–112, Aug. 2023.

[4] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, vol. 106, 2019.

[5] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, vol. 1, 2022.

[6] T. C. Freitas, A. Costa Neto, M. J. a. V. Pereira, and P. R. Henriques, "NLP/AI Based Techniques for Programming Exercises Generation," in *4th International Computer Programming Education Conference (ICPEC 2023)*, ser. Open Access Series in Informatics (OASIcs), R. A. Peixoto de Queirós and M. P. Teixeira Pinto, Eds., vol. 112. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 9:1–9:12. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.ICPEC.2023.9

[7] P. Denny, S. Sarsa, A. Hellas, and J. Leinonen, "Robosourcing educational resources - leveraging large language models for learnersourcing," in *CEUR Workshop Proceedings*, vol. 3410, 2022.

[8] P. Haluptzok, M. Bowers, and A. T. Kalai, "Language models can teach themselves to program better," arXiv, 2023, 2207.14502.

[9] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, "Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3544548.3581388

[10] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[11] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[12] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, "Gpt-neo: Large scale autoregressive language modeling with meshtensorflow," https://doi.org/10.5281/zenodo.5551208, 2021, accessed: 2024-01-02.

[13] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," *CoRR*, vol. abs/2101.00027, 2021. [Online]. Available: https://arxiv.org/abs/2101.00027

[14] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard - or at least it used to be: Educational opportunities and challenges of ai code generation," in *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, vol. 1, 2023.

[15] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn, and G. Kasneci, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, 2023.

[16] J. Prather, P. Denny, J. Leinonen, B. A. Becker, I. Albluwi, M. Craig, H. Keuning, N. Kiesler, T. Kohn, A. Luxton-Reilly, S. MacNeil, A. Peterson, R. Pettit, B. N. Reeves, and J. Savelka, "The robots are here: Navigating the generative ai revolution in computing education," arXiv, 2023, 2310.00658.

[17] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *CoRR*, vol. abs/2009.03300, 2020. [Online]. Available: https://arxiv.org/abs/2009.03300

[18] Z. Guo, R. Jin, C. Liu, Y. Huang, D. Shi, Supryadi, L. Yu, Y. Liu, J. Li, B. Xiong, and D. Xiong, "Evaluating large language models: A comprehensive survey," arXiv, 2023, 2310.19736.

[19] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *CoRR*, vol. abs/1905.00537, 2019. [Online]. Available: http://arxiv.org/abs/1905.00537

[20] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, ser. MAPS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–10. [Online]. Available: https://doi.org/10.1145/3520312.3534862

[21] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," arXiv, 2023, 10.48550/ARXIV.2305.01210. [Online]. Available: https://arxiv.org/abs/2305.01210

[22] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019. [Online]. Available: http://dx.doi.org/10.18653/v1/p19-1472