Code Generation Capabilities of LLMs: A Comparative Analysis Using Competitive Programming Problems

Hugo Morvan
Linköping University
hugmo418@student.liu.se

Abstract

Large Language models (LLMs) have demonstrated remarkable advancements in natural language processing, but their performance on competitive programming tasks remains an open question as benchmarks usually focus on more traditional coding tasks. Competitive programming tasks require not only linguistic understanding but also algorithmic and mathematical reasoning, precision and efficiency. This paper evaluates the problem-solving capabilities of 5 small size LLMs on a set of competitive programming problems from the publicly available coding platform Open.Kattis. Each code generated is evaluated on compiling correctness, success on sample input/output combinations and finally on the online judging platform directly. The results reveal a significant gap between the ability of the LLMs to produce correct code, which is pretty good, and the ability of the LLMs to produce code that actually solves the competitive programming tasks, which is not very good. These finding offer a valuable insight to the current state of most LLMs, which is that although they appear to produce valuable outputs, they actually lack strong reasoning capabilities to produce qualitative outputs.

1 Introduction

The field of natural language processing (NLP) has undergone rapid transformation with the emergence of large language models (LLMs) capable of generating coherent and contextually appropriate text. These models demonstrate significant potential in domains ranging from creative writing to scientific research. However, one particularly challenging domain is competitive programming, where success requires not only linguistic understanding, but also precise problem-solving skills, algorithmic and mathematical reasoning, and correct code. In this project are tested five different small and open-sourced LLMs against competitive programming tasks of varying difficulty. The aim of this project is to compare the problem-solving capabilities of LLMs and estimate the current state of their reasoning capabilities through this small empirical study.

2 Data

To evaluate the effectiveness of large language models (LLMs) in generating code, it is crucial to assess their performances on a diverse range of coding problems. This requires a comprehensive dataset of coding problems with varying levels of difficulty and complexity. The International Collegiate Programming Contest (ICPC) is the world's oldest and largest programming competition for college students (icp, 2025), and they use the Kattis website to host many of their contests as the website has a large problem archive, ranging from trivial problems to very difficult ones. (kat, 2025). For this project, a set of 100 problems has been randomly selected from the 4600+ available problems.

The data collection was made using the `autokattis` python library (Saerang, 2023) and then processed to isolate the problem metadata, problem descriptions, input descriptions, output descriptions and sample input/output combinations. The difficulty distribution of the selected problems can be seen in figure 1.

3 Method

3.1 Code generation

To generate code, we use LM Studio, a powerful open-source platform designed for running and fine-tuning LLMs locally on personal or dedicated hardware. It provides users with a versatile GUI to download open-source LLMs from Hugging Face (hug, 2025) and facilitate code generation through interaction with a language server (lms, 2025). A language server is started locally, allowing to process incoming requests. In a Python script or a

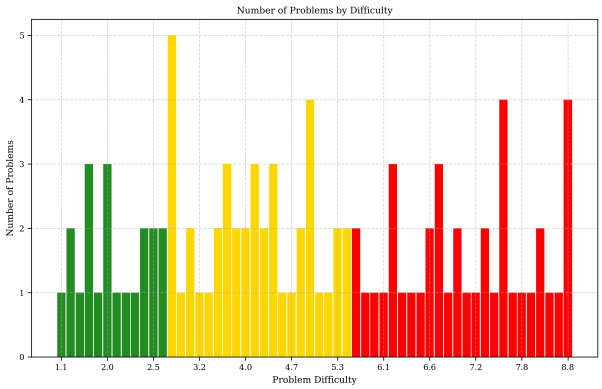

Figure 1: Distribution of the problems difficulty in the sample selection

Jupyter notebook, a request is then sent to the local server with the desired prompt/instructions. The input is processed by the server, and the language model hosted by LM Studio is utilize to generate the corresponding output. This output is then returned as a response to the request, enabling automated code generation for a variety of open-source models. The time taken to generate each script is also measured and recorded for further analysis and comparisons.

One limitation of this approach is that it relies on running the language models locally, which makes the choice of model dependent on the resources available on the local-machine. All computations for this project were performed on a system with 32GB of RAM, an NVIDIA GeForce RTX 3070 Ti Laptop GPU with 8GB of VRAM, and a 12th Gen Intel Core i9-12900H CPU.

3.2 Code evaluation

Each script generated by the LLMs will be evaluated on several levels of quality. The first level of evaluation is whether or not the code compiles properly. At this level, the code will be tested to see if it is properly written according to Python requirements, i.e. if the indentation is consistent, if the variables/functions called are defined, if there is any syntax error, etc. At this level is also verified that the code generated does not import any non-standard library (e.g. `numpy`, `pandas`) as it is one of the requirements from the Kattis website (Kattis Support, 2025).

For the second and third levels, the code is tested against the sample inputs and outputs. Level 2 is considered passed if the script runs without error when given the sample inputs. This verifies that no run-time error occurs when the program is ran with inputs. Level 3 verifies that when the script runs without error when given the sample input, it produces an output that matches the expected sample output. Since each problem contains sample I/O, each of the problems selected can be tested. However these samples are usually trivial example in the space of all possible input/output combinations for each problem. Hence, passing this level is not indicative of a solution that necessarily grasps the full complexity of a problem.

For the last level of evaluation, the code is tested on the hidden inputs and outputs on the Kattis website. Here, each script is submitted on the online judging platform provided by the competitive programming website. A successful submission on the Kattis website demonstrates a complete understanding of the problem, taking into account the possible edge cases that might have not been explicitly stated in the problem description or in the test cases.

To save on some computation, only the scripts that pass level 3 evaluation will be tested against the hidden test cases. This step is the true bottleneck of this project since the website does not allow automated submission and limits the submission attempts to one submission per minute. Previous attempts at automatizing this step resulted in a banned account on the online platform.

3.3 Models evaluated

3.3.1 Same model family, different sizes

The first comparison that will be performed is between 3 different parameterized version of the same model: Qwen2.5-Coder-3B-Instruct, Qwen2.5-Coder-7B-Instruct and Qwen2.5-Coder-14B-Instruct (Hui et al., 2024). Qwen2.5-Coder is s series of open-source model from the Alibaba Group and is available in multiple different model sizes. By comparing different model sizes, the aim is to test the claims of the technical report, which shows that bigger model systematically perform better than the smaller ones across the various benchmarks.

3.3.2 Different models, same sizes

In this comparison we will look at three different model of roughly equal size (7B parameters) from different companies, and look at their respective performances. The first model tested is the 7B-Instruct version of Qwen2.5-Coder from the Alibaba Group as mentioned previously. The second model tested DeepSeek-V2-Math-7B from

DeepSeek (Shao et al., 2024), a rival company of Alibaba's Qwen team. Although it is a mathematical reasoning model, the model was chosen for its comparable size and its description also claims coding capabilities. Moreover, many Kattis problems (including in this project's selection) necessitate good mathematical reasoning to be solved, so it will be interesting to see if this model fares differently. The final model is CodeLlama-7B-Instruct from Meta AI (Rozière et al., 2024). This model derived from Llama 2 claims good support for large input contexts and zero-shot instruction following ability for programming tasks, which is fitting for this project. With this comparison we aim to see if there is a drastic difference in performance between the models from competing companies.

3.4 Prompt

All models mentioned previously received the same prompting instructions:

> You are a highly skilled competitive python programmer. When given a prompt, you only output code. Your code should take in inputs using the input() function and return outputs using print(). Generate code for: {prompt}"

, where {prompt} is a problem's description as well as the input and output requirements for the problem.

4 Results

The success at the various evaluation levels are presented in Table 1 and Figure 2. A comparison of the average difficulty solved versus the average time taken to generate code is presented in Figure 3. A visualization of the difficulty of the problem solved by all the models is presented in Figure 4.

5 Discussion

Table 1 and Figure 2 show that in this scenario, the DeepSeek model completely failed at producing any compiling code. When examining the code produced, some elements of python syntax can be seen, but there is a lot of empty or incomplete functions, lengthy comments or even sometimes a copy of the system prompt showed in section 3.4. It is difficulty to explain why this model performed so poorly compared to the other models, especially considering that it was evaluated in the exact same conditions.

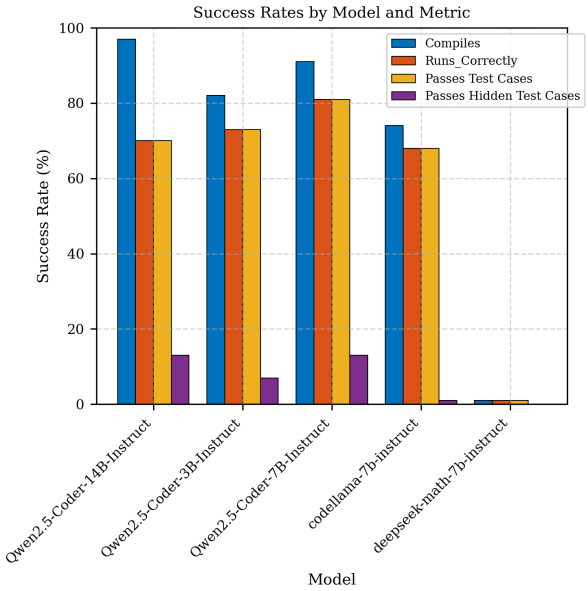

Figure 2: Success rates of each models on the 4 quality levels tests

As for the other models, the Qwen2.5-Coder-14B-Instruct was unsurprisingly the best model at producing compiling code. More surprising however was that both the 3B and 7B Qwen models performed better than the 14B model on the evaluation with sample input/outputs. The CodeLlama model slightly lesser performances compared to the Qwen models. On the last level of evaluation, where the codes generated are tested directly on the Kattis online judging platform, the 7B and 14B Qwen models tied for 1st place with 13 problems solved, the 3B version came in 2nd place with 7 problem solved, and the CodeLlama model was able to solve 1 problem. Figure 3 shows that the problem solved were mostly easy problems, with a few medium difficulty problems being solved as well but none from the hard difficulty.

These results demonstrates that there is still a big gap between producing code that compiles and code that actually solved the tasks presented. Large Language Models have the capability of producing code that looks like it could solve the problems, but they lack a deeper mathematical or algorithmic reasoning required for more complex competitive programming tasks.

This project also shows that, for consumer grade/local models, bigger does not necessarily means better, as it can be seen with the results with the Qwen models. In fact, when taking into account generating time of the tokens, as shown in Figure 3, it can be argued that the 7B model performed

Table 1: Summary results for 100 generated script per model.

Model	Compiles	Runs Correctly	Passes Test Cases	Passes Hidden Test Cases
Qwen2.5-Coder-3B-Instruct	82	73	73	7
Qwen2.5-Coder-7B-Instruct	91	**81**	**81**	**13**
Qwen2.5-Coder-14B-Instruct	**97**	70	70	**13**
codellama-7b-instruct	74	68	68	1
deepseek-math-7b-instruct	1	1	1	0

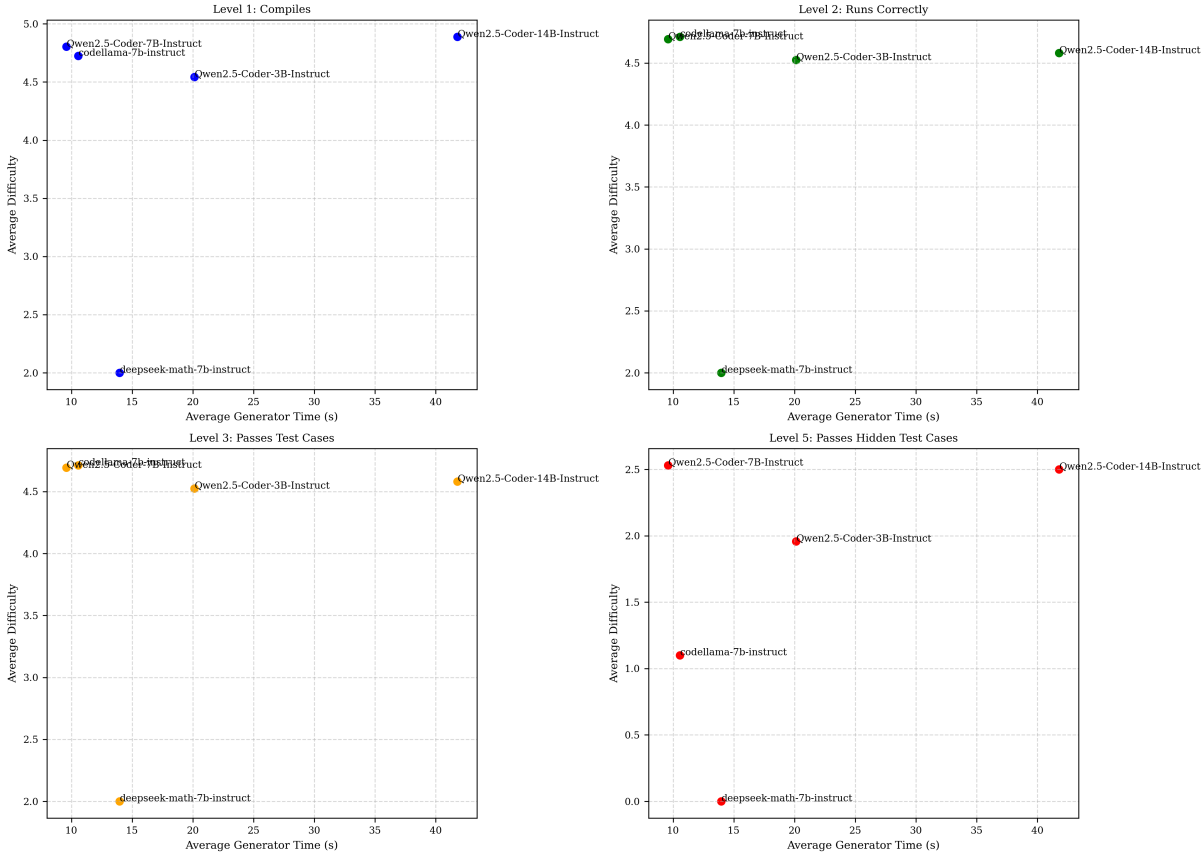

Figure 3: Average code generation time VS Average solved problem difficulty per model

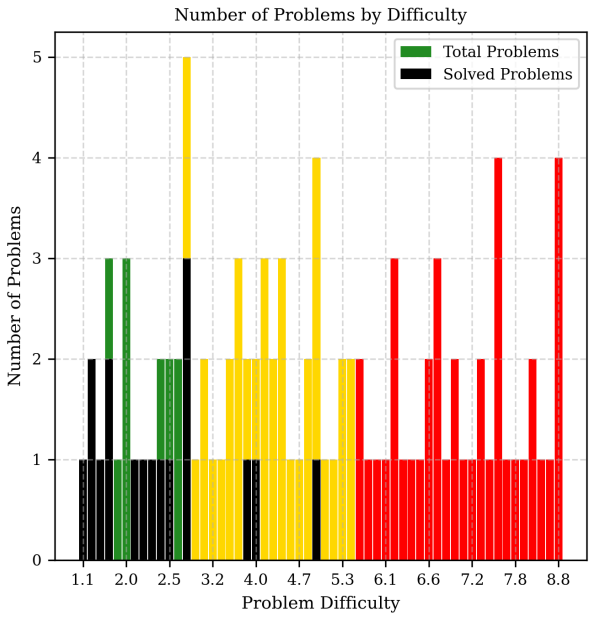

Figure 4: Difficulty of the problem solved by all the models. In black are the problems that were solved.

better than the 14B model. For a similar average difficulty of the problem solved, the 7B model produced tokens far quicker. This results are however dependent of the computational resources available and may vary.

5.1 To be improved

The model were evaluated based on a single sample of problems, taking only one generation per problem. A more statistically consistent approach would be to perform multiple generation on multiple problem sets. However this approach would be much more computationally expensive and would take much longer, therefore all the results presented in this report must be taken with reservations.

6 Conclusion

This project successfully evaluated and compared various small, code specific, models. While the models were almost all able to produce python code, only a few problems of relatively easy difficulty were actually solved. This suggest that LLMs still lack, in this scenario, the ability to reason about logical, mathematical or algorithmic problems, which makes them not very good for competitive programming tasks.

References

2025. Hugging face: The ai community building the future. https://huggingface.co/. Accessed: 2025-01-19.

2025. International collegiate programming contest. https://icpc.global/. Accessed: 2025-01-19.

2025. Kattis - programming problems archive. https://open.kattis.com/. Accessed: 2025-01-19.

2025. Lm studio: Discover, download and run local llms. https://lmstudio.ai/. Accessed: 2025-01-19.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Kattis Support. 2025. What python modules are available? https://support.kattis.com/support/solutions/articles/79000122491-what-python-modules-are-available-. Accessed: 2025-01-19.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Russell Saerang. 2023. Autokattis: Kattis problem scraper and auto-submission tool. https://github.com/RussellDash332/autokattis. Accessed: 2025-01-19.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Source Code

All the source codes for this project as well as the code generated by the different LLMs is available at https://github.com/hugo-morvan/KattisBot .