NET IMPACT OF LARGE LANGUAGE MODELS TRAINED ON CODE

Pavel GHERCIU

Department of Computer Science and Systems Engineering, group IA-201, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Republic of Moldova

Corresponding author: Pavel Gherciu, pavel.gherciu@iis.utm.md

Abstract. *Natural language processing has seen many improvements in recent years, particularly driven by machine learning models such as OpenAI's GPT-3. This paper aims to present the various language models, as well as OpenAI Codex, which is considered to be an AI revolution in the field of programming. This system has been trained on Python code from more than 50 million GitHub repositories and is capable of generating and explaining code, translating it between various programming languages and more. In addition, the benefits and potential dangers of its use will be analysed and presented.*

Keywords: *artificial intelligence, code generation, language models, OpenAI, Codex, natural language processing*

Introduction

In recent times, Large language models have seen a significant spike in popularity, which could be attributed to their updated capabilities, which allow them to serve as solutions for a variety of applications including helping developers perform general coding tasks. Specialised tools based on LLMs have been trained on vast amounts of source code so that they can predict likely code completions given some prompt comprising comments, function names, and other code elements. The newest commercial models have such large datasets that they are now claimed to be general purpose coding helpers, capable of performing many various tasks, such as translation between programming languages, code completion, and even explanation.

Defining Large language models

Language models are statistical and probabilistic tools which are usually used for predicting the next word in a sentence, that is to say, they are systems trained on predicting the likelihood of a token (character, word or string) given either its preceding context or its surrounding context. They are unsupervised and when deployed, take text as input and output scores or string predictions [1].

These systems are called large language models when they are trained on enormous amounts of data. Some examples of LLMs are Google's BERT and Switch-C, as well as OpenAI's GPT-2 and GPT-3, as can be seen in *Table 1*. Switch-C is currently known as the largest language model with 1.6 trillion parameters trained on 745 gigabytes of text. These models have capabilities ranging from writing a simple essay to generating complex computer code with limited to no supervision.

When it comes to OpenAI's language models, compared to the initial GPT-1 architecture, GPT-3 has virtually nothing novel. But it is much larger, having 175 billion parameters and was trained on Common Crawl, which was considered to be the largest corpus a model could have been trained on at the time of GPT-3's release. This is mainly possible due to the semi-supervised training strategy of a language model, which implies that a text can be used as a training example with some words omitted. The impressive power of GPT-3 comes from the fact that it read more or less all text that appeared on the whole internet in the past years, and has the capability to reflect most of the complexity that natural language has [2].

Table 1

Overview of recent large language models [1]

Year	Model	# of Parameters	Dataset Size
2019	BERT	3.4E+08	16GB
2019	DistilBERT	6.60E+07	16GB
2019	ALBERT	2.23E+08	16GB
2019	XLNet (Large)	3.40E+08	126GB
2020	ERNIE-GEn (Large)	3.40E+08	16GB
2019	RoBERTa (Large)	3.55E+08	161GB
2019	MegatronLM	8.30E+09	174GB
2020	T5-11B	1.10E+10	745GB
2020	T-NLG	1.70E+10	174GB
2020	GPT-3	1.75E+11	570GB
2020	GShard	6.00E+11	–
2021	Switch-C	1.57E+12	745GB

OpenAI Codex

At the time of GPT-3's release, it was revealed that it could generate simple programs from Python docstrings. While rudimentary, this capability was exciting, considering that GPT-3 was not specifically trained for code generation. Given the substantial success of large language models in other applications and the abundance of publicly available code, a specialized GPT model could hypothetically be used to excel at a variety of coding tasks.

This very conclusion has given rise to OpenAI's own artificial intelligence model called Codex. OpenAI Codex, like GPT-3, has much of the same natural language understanding, but it produces working code - meaning that commands can be issued in English to any piece of software with an API. Its training data contains natural language as well as billions of lines of source code from publicly available sources, including code in public GitHub repositories. Although Codex isn't the first LLM trained on code, it is currently given special attention due to its stronger performance with the Python programming language. While it is most capable in Python, it is also effective in several other languages such as JavaScript, Go, Perl, PHP, Ruby, Swift and TypeScript, and even Shell. It has a memory of 14KB for Python code, while GPT-3 has only 4KB - so it can take into account over three times as much contextual information while performing any code-related task [3].

Positive impact

Code generation and associated capabilities have several potential economic and labor market impacts. While Codex in its current state could somewhat reduce the cost of producing software by increasing programmer productivity, the effect may be limited by the fact that engineers don't only write code, but also carry out other important tasks that include conferring with colleagues, writing design specifications, and upgrading existing software stacks.

It has been found that Codex imports packages at different speeds, which could advantage some package authors over others, especially if programmers and engineers come to rely on Codex's suggestions. Over a longer period of time, the effects of this class of technologies on software-related labor markets and on the economy could be more substantial as capabilities improve.

More analysis will be required to fully understand the potential impact of code generation, , however this section will present some of the possible effects that this new tool would bring upon the industry as a whole, with an emphasis on the labor market [3].

It is possible that code generation could help create economic value by allowing engineers and programmers to write better code faster and also help with tasks such as docstrings, documentation, tests, code reviews, etc. This could change the work of engineers and programmers as well as lower the barrier to building software and even enable the development of new kinds of software.

At the same time, this could allow for fixing security bugs in code easier, as it would be faster for a Language Model to identify and fix any bugs compared to humans. Research has shown that OpenAI Codex is successful when tasked with security bug repair, however it is far too recent to replace current tools used by most developers in the field. It is expected that developers will use such LLMs to write functional and secure code in the future, as shown in *Figure 1* [4].

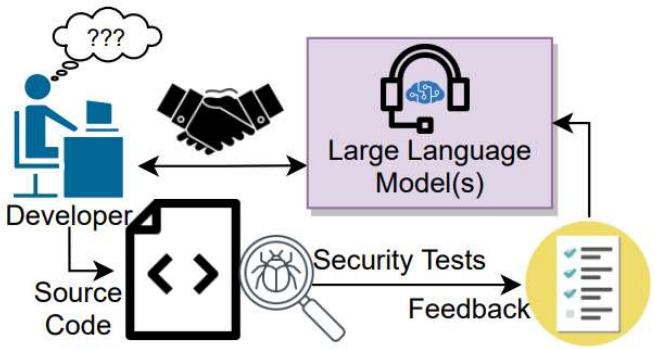

Figure 1. Cybersecurity application of LLMs [4]

Potential dangers

According to OpenAI CTO and co-founder Greg Brokman, Codex is "a tool to multiply programmers." He also said that the rationale behind it is not to make a computer in charge of programming, but to simply serve as a programmer's assistant. That helper would take conceptual ideas and produce the code to turn them into software or any other technology [5].

Although OpenAI claims this of their current model, more sophisticated future code generation tools could potentially lead to the displacement of some programmer or engineer roles, and could change the very nature of programming work. They might make the work of some engineers more efficient or they could create the illusion of increased efficiency while offloading the time spent writing code to more detailed code reviews and QA testing. While that could be the case, realistically, as can be seen with many other fields, it is more likely that many jobs will be indeed be lost. Companies would profit much more from automation and a select few "thinkers", that would then simply pass the concept and design over to the LLM, which would take care of the "repetitive work that humans don't want to do". Currently, given Codex's performance on Python, it is likely that its impact will be felt more in roles which use Python as the dominant programming language. This could in turn encourage companies to switch their codebases to programming languages where they know Codex could augment work.

The proliferation of tools such as Codex raise concerns and challenges for the future of computing education as well. It is said that the value of a tool depends on its use, and in this case there is a real danger that Codex could be used in ways that either limit learning or that make the work of educators difficult. The developers of Codex mentioned one such challenge: possible over-reliance on Codex by novice programmers.

Codex may be used by students to generate model solutions for exercises where they are not provided by instructors. If the generated solution is incorrect or uses poor style, it is likely that students are not learning optimally, and may adopt bad habits which could include faulty conventions and style. While this is true for even web-based examples, the customised solutions offered by Codex may entice students to believe that the provided solutions are more credible, similar to an intelligent

tutoring system. To ensure that students don't rely on such tools as an "oracle", it thus may become necessary to alter future programming courses to focus more on code review, or evaluation of code, to ensure that students can effectively assess the code generated by Codex and its overall quality. As such, it is likely that academia will come to rely on supervised exams even more, despite online learning becoming more popular by the day.

The fact that Codex in its current form has limits when it comes to the complexity of problems it can solve may not mean much to students. If students submit code generated by Codex, even if it's not correct, it will likely be worthy of partial credit. It would thus become hard for a teacher to tell the difference between a student who tried but eventually failed to get the right answer and a student who simply generated code using Codex. This may result in students being awarded a pass in a course without the appropriate level of knowledge required, and subsequently burden instructors and peers in group projects. At the same time, it could lead to a vast number of graduates with very little practical knowledge which would damage the industry even more than Codex itself [6].

Conclusion

In this paper, the current capabilities of Large language models have been presented, as well as of those trained on code. Special attention has been given to OpenAI Codex, which is currently a very popular example of such systems and is considered to be the most capable at doing coding tasks in the Python programming language.

While the technology is recent, it has been proven to be relatively effective at what it does, although in a limited capacity compared to real programmers. Nonetheless, it must be noted that such technologies will only continue to improve and proliferate. As such, this paper also presents the benefits and dangers that could come from widespead use of LLMs trained on code.

Tools such as Codex could potentially assist engineers in doing their work more efficiently, help students learn faster and also solve cybersecurity issues, but special emphasis must be put on the dangers associated with replacing coding with simply using an automated system, as it could lead to many people losing their jobs and computer education being compromised.

In conclusion, the reality is that this particular AI revolution has now come and there is no choice but to learn to adapt, because technological progress will not wait for anyone.

References:

1. BENDER, E., GEBRU, T., MCMILAN-MAJOR, A., SHMITCHELL, S. *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?*. ACM Conference on Fairness, Accountability, and Transparency, 2021, pp. 610-623.
2. Towards Data Science. *A beginner's guide to language models,* [accessed 24.02.2022]. Available: https://towardsdatascience.com/the-beginners-guide-to-language-models-aa47165b57f9.
3. CHEN, M., TWOREK, J., JUN. H., KAPLAN, J., EDWARDS, H., BURDA Y., BROCKMAN, G. *Evaluating Large Language Models Trained on Code,* 2021.
4. PEARCE H., TAN, B., AHMAD, B., KARRI, R., DOLAN-GAVITT, B. *Can OpenAI Codex and Other Large Language Models Help Us Fix Security Bugs?*, 2021.
5. OpenAI. *OpenAI Codex ,* [accessed 24.02.2022]. Available: https://openai.com/blog/openai-codex/.
6. FINNIE-ANSLEY, J., DENNY, P., BECKER, B., LUXTON-REILLY, A., PRATHER, J. *The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming.* Australasian Computing Education Conference, 2022, pp. 10-19.