Editor in Chief: **Ipek Ozkaya**
Carnegie Mellon Software Engineering Institute
ipek.ozkaya@computer.org

Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications

Ipek Ozkaya

HAS THE DAY we all have been waiting for really arrived? Have advances in deep learning and machine learning (ML) finally reached a turning point and have started to produce "accurate enough" assistants to help us in a variety of tasks, including software development? Are large language models (LLM) going to turn us all into better writers, artists, translators, programmers, health-care workers, not to mention software engineers? Or are we at a risky turning point where we will not be able to separate artificial intelligence (AI)-generated content from user-created ones, drowning in misinformation and perfect sounding yet fake and incorrect information and AI-generated faulty programs?

Digital Object Identifier 10.1109/MS.2023.3248401
Date of current version: 18 April 2023

Recently released LLMs, such as Generative Pretrained Transformer (GTP) 4 used in ChatGPT by OpenAI and BERT used in Bard by Google, disrupt the search engine model that we have been used to. Use of these models shifts the end-user computer interaction from "here are a list of places to look at to potentially find an answer to your question" to "here is a suggested answer to your questions with well-constructed syntax, what is your next question based on this?"

Without a doubt, LLMs have use cases in assisting software engineering tasks as well, including code generation models trained in programming languages, such as CoPilot by GitHub. The reaction of the software engineering community to the accelerated advances that LLMs have been enjoying since 2022 has been varied, ranging from considering capabilities offered by these models as "snake oil"[1] to "end of programming and computer science education as we know it."[2] In this article, after a brief overview of LLMs, I will focus on the opportunities LLMs open up for software development and implications of incorporating LLMs into systems as well as assisting with software development tasks.

What Are LMMs?

An LLM is a deep neural network model which has been trained on large amounts of data, such as books, code, articles, and websites, to learn the underlying patterns and relationships in the language that it was trained for. By doing so, the model is able to generate coherent content such as grammatically correct sentences and paragraphs that mimic human language or syntactically correct code snippets.

 0740-7459/23©2023IEEE

LLMs have applications in a variety of tasks, including language translation, summarization, and question answering and have potential in many fields as long as the data that the models have been trained on provide the appropriate input. While the content generated by LLMs are often grammatically correct, they may not always be semantically correct. The probabilistic and randomized selection of the "next token" in constructing the outputs on one hand gives the end user the impressions of correctness and style, on the other hand may result in mistakes.[3]

While the recently released versions of LLMs, ChatGPT driving the pack, have made significant improvements, there are several areas of caution around their generation and use:

- *Data quality and bias concerns:* LLMs require enormous amounts of training data to learn language patterns and their outputs are highly dependent on the data that they are trained on. Any of the issues that exist in the training data, such as biases and mistakes, will be amplified by LLMs, potentially resulting in models that exhibit discriminatory behavior, such as making prejudiced recommendations. This means that the quality and representativeness of the training data can significantly impact the model's performance and generalizability, mistakes can propagate. For example, language models that are used to recommend code patterns have been found to carry security flaws forward.[4] This creates risks in not only generating buggy code, but also perpetuating immature implementation practices in developers.
- *Privacy and content ownership concerns:* LLMs are generated using content developed by others which both may contain private information as well as content creators' unique creativity characteristics. Training on such data using patterns in recommended output creates plagiarism concerns. Some content is boiler plate and the ability to generate output in correct and understandable ways creates opportunities for improved efficiency. But content, including code, where individual contributions matter becomes difficult to differentiate. In the long run, increasing popularity of language models will likely create boundaries around data sharing and open source software and open science. Techniques to indicate ownership or even preventing certain data to be used to train such models will likely emerge. However, such techniques and attributes to complement LLMS are yet to come.
- *Environmental concerns:* The vast amounts of computing power required in training deep learning models has been increasingly a concern related to their impact on carbon footprint. Research in different training techniques, algorithmic efficiencies, and varying allocation of computing resources during training will likely increase. In addition, improved data collection and storage techniques are anticipated to eventually reduce the impact of LLMs on the environment, but development of such techniques are still in their early phases.[5]
- *Explainability and unintended consequence concerns:* Explainability of deep learning and ML models is a general concern in AI, including but not limited to LLMs. Users seek to understand the reasoning behind the recommendations,

CONTACT US

AUTHORS

For detailed information on submitting articles, visit the "Write for Us" section at www.computer.org/software

LETTERS TO THE EDITOR

Send letters to software@computer.org

ON THE WEB

www.computer.org/software

SUBSCRIBE

www.computer.org/subscribe

SUBSCRIPTION CHANGE OF ADDRESS

address.change@ieee.org
(please specify *IEEE Software*.)

MEMBERSHIP CHANGE OF ADDRESS

member.services@ieee.org

MISSING OR DAMAGED COPIES

contactcenter@ieee.org

REPRINT PERMISSION

IEEE utilizes Rightslink for permissions requests. For more information, visit www.ieee.org/publications/rights/rights-link.html

Digital Object Identifier 10.1109/MS.2022.3228353

especially if such models are to be used in safety or business critical settings. Dependence on the quality of the data and inability to trace the recommendations to the source increase trust concerns.[6] In addition, since the sequences are generated using a randomized probabilistic approach, explainability of correctness of the recommendations create added challenges. Explainability as well as responsible AI practices are critical since such models can easily be used to spread misinformation.

The application programming interfaces (API) of GPT and BERT are now also available to other developers. This contributes to both accelerating the use and improvements on LLMs as well as increasing the number of opportunities of their misuse. OpenAI researchers are open about their lessons learned and have no choice but rely on software engineering best practices. They recommend policy enforcement as a mechanism to enforce avoiding misuses.[7] Applications which help detect text written by such models have been quick to come, such as GPTZero written for educators to detect such text, and ironically it uses ChatGPT in doing so.[8] It is safe to say LLMs have attracted a fair share of confusion, criticism, and excitement all at the same time.

Applications in Software Engineering

Research agendas developed recently had already shined the light on the future of software engineering to be an AI-augmented development lifecycle where both software engineering and AI assistants share roles from copilot to student, expert, and supervisor.[9] In the National Agenda for Software Engineering, my colleagues and I had suggested that developers will need to guide and consequently improve the AI assistants. AI assistants will also take on a supervisory role by providing real-time feedback and, in time, demonstrating repeated mistakes to developers. On a developer team, there will always be some developers who you trust more than others (perhaps due to experience, skill sets, or demonstrated performance). The AI-assisted development workflows will trigger the need to think of AI "partners" in the same way.[9]

While with caution, software engineers need to think about LLMs as partners and focus on where their optimal application can be. There are quite a number of software engineering tasks which can effectively benefit from using LLMs. Indulge me for a moment to assume that we solved the trust and unethical use issues as I enumerate potential use cases where LLMs can create strides of advances in improved productivity of software engineering tasks, and where the risks can still be manageable.

- *Specification generation:* Quite a number of requirements can be common across applications, yet oftentimes requirements are also incomplete. LLMs can assist in generating more complete specifications significantly quicker.
- *Just in time developer feedback:* Applications of LLMs in software development has been received with much skepticism, rightly so at the time being. While the code generated by current AI assistants, such as Copilot, have been found to carry more security issues,[4] in time this will change. AI-based and other approaches which give developers

syntactic corrections and suggestions have been around a while. LLMs carry the promise of going the extra mile and recommending not just corrections, but next steps.
- *Improved testing:* Generating unit tests is one of the tasks where developers shortcut the most. Ability to generate test cases at ease would increase overall test effectiveness and coverage, and consequently system quality.
- *Documentation:* Ranging from contracting language to regulatory requirements, there are many applications of LLMs to software development documentation.
- *Language translation:* Legacy software and brownfield development is the norm of system development today, and many organizations need to go through language translation efforts when they need to modernize their systems. This process is often manual and error prone, while some tools do exist to support developers. While will not work at scale, portions of code can potentially be translated to other programming languages using LLMs. Rewriting a system in an other programming language is not just a language translation exercise, it is mostly also a re-architecting exercise; however, ability to rewrite selected portions at ease would be a welcomed capability.

LLMs will also require software engineers to become more savvy in how they incorporate them into systems as elements. Example areas include the following:

- *LLMs as functional components:* LLMs will definitely change some of the ways capabilities are bundled and delivered as well, where pretrained models become parts of systems or parts of external systems. APIs to LLMs will drive different system composition scenarios and will be available as services.
- *Operations informing development:* Data is the first-class citizen in LLM tools. Operational data will need to be more timely fed back to both the development process, e.g., areas where users make most mistakes, as well as functionality development, e.g., inform functionality that users do not use to be deprecated.

> Computer science and software engineering programs need to start a shift in their curricula today.

These examples focus on existing software engineering tasks that can be done better or faster because such models exist. There are also, however, task flows that will change, and new activities will likely emerge while time spent on others get reduced. An AI-augment software development lifecycle will likely have different task flows, efficiencies, and roadblocks than the current development lifecycles of agile and iterative development workflows. For example, rather than thinking about steps of development as requirements, design, implementation, test, and deploy, LLMs can enable bundling these tasks together. This would change the number of hand-offs and where they happen, shifting task dependencies within the software development lifecycle.

Going Forward

All the areas of cautions and risks related to LLMs are areas where we need new research and innovations. These need to be targeted at improving correctness of LLM recommendations, improving their generalizability, as well as improving the ethical implications of data use and content creation.

We are likely to see most advances in generalizability of models, development of integrated development environments with new paradigms, and reliable data collection and use techniques in the near future. Curricula development and education of the next generation of computer scientists and software engineers cannot stay blind to the implications of such developments in generative AI either.

Generalizability of Models

Currently, LLMs work by pretraining on a large corpus of content followed by fine-tuning on a specific task. What this implies is that the architecture of the model is task independent; however, its application for specific tasks requires further fine-tuning with significantly large numbers of examples. Generalizability of these models to applications where data are sparse, few-shot settings, is already a focus area by researchers.[10]

New Development Environments

If we are convinced by the argument that some tasks can be accelerated and improved in correctness by AI assistants including LLMs, that also implies that the current integrated development tools will need

to incorporate these assistants. When assistants are integrated in, then development becomes a more interactive process with the tool environment. Software engineering bots are already pushing the envelope of the development environments in the direction of incorporating developer assistants.[11]

Data as a Unit of Computation

The most critical input which drives this next generation of AI innovations is not only the algorithms, but also data. Not only will a significant portion of computer science and software engineering talent shift to data science and data engineer careers, but also, we will need more tool-supported innovations in data collection, data quality assessment, and data ownership rights management. This is an area with huge gaps that requires skill sets that span computer science, policy, engineering, as well as deep knowledge in security, privacy, and ethics.

Computer Science and Software Engineering Education

The biggest implications of LLMs are in how we teach programming languages and system design. LLMs are likely to take already existing platforms such as StackOverflow and Reddit, which have become indispensable resources for developers, to a new level of reduced barrier of entry. Computer science and software engineering programs need to start a shift in their curricula today. Software engineering and computer science education has already missed the boat by continuing to focus on teaching green field development while today the reality of system development is brownfield. Students are not adequately exposed to theories and techniques to support system development by composition, legacy evolution, and using heterogeneous platforms and programming languages in concert. We teach students hello world development, while we should be teaching them how to read millions of lines of code, triage and fix bugs that they have not contributed to and understand the structure and behavior of the software rather than the single class or story card they are responsible for. With LLMs and their sister AI-driven apps assisting developers, we need to be teaching next-generation software engineers when to trust, how to create evidence to trust, how to do trust assessment rapidly and correctly, and how to improve such assistants. We need to teach them how to evolve systems to incorporate such components, and we need to teach them to treat data as code. We need to make ethics courses mandatory every year of the curriculum. The list goes on.

After the two winters of AI, generally attributed to late 1970s and early 1990s, we have entered not only a period of AI blossoms, but also exponential growth in funding, in use, and in scare from AI. Advances in LLMs without a doubt are huge contributors to this growth. What will determine if the next phase includes innovations beyond our imagination or another AI winter is largely dependent on not our ability to continue technical innovations, but on our ability to practice software engineering and computer science through the highest level of ethics and responsible practices. We need to be bold in experimenting with the potential of LLMs in improving software development, and we need to be cautious and not forget fundamentals of engineering ethics and rigor.

References

1. S. Shankland. "Computing guru criticizes ChatGPT AI tech for making things up." CNET. Accessed: Feb. 2023. [Online]. Available: https://www.cnet.com/tech/computing/computing-guru-criticizes-chatgpt-ai-tech-for-making-things-up/
2. M. Welsh, "The end of programming," *Commun. ACM*, vol. 66, no. 1, pp. 34–35, Jan. 2023, doi: 10.1145/3570220.
3. S. Wolfram. "What is ChatGPT doing … and why does it work?" Stephen Wolfram. Accessed: Feb. 2023. [Online]. Available: https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/
4. N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with AI assistants?" 2022, *arXiv:2211.03622*.
5. D. A. Patterson et al., "The carbon footprint of machine learning training will plateau, then shrink," *Computer*, vol. 55, no. 7, pp. 18–28, Jul. 2022, doi: 10.1109/MC.2022.3148714.
6. C. Tantithamthavorn, J. Cito, H. Hemati, and S. Chandra, "Explainable AI for SE: Experts' interviews, challenges, and future directions," *IEEE Softw.*, vol. 40, no. 4, 2023.
7. M. Brundage et al., "Lessons learned on language model safety and misuse." OpenAI. Accessed: Feb. 2023. [Online]. Available: https://openai.com/blog/language-model-safety-and-misuse/
8. GPTZero. Accessed: Feb. 2023. [Online] Available: https://gptzero.me/faq
9. A. Carleton et al., "Architecting the future of software engineering: A national agenda for software engineering research and development," Softw. Eng. Inst., Pittsburgh, PA, USA, AD1152714, 2021.
10. T. B. Brown et al., "Language models are few-shot learners," 2020, *arXiv:2005.14165*.
11. I. Ozkaya, "A paradigm shift in automating software engineering tasks: Bots," *IEEE Softw.*, vol. 39, no. 5, pp. 4–8, Sep./Oct. 2022, doi: 10.1109/MS.2022.3167801.