AI-assisted software development

Introduction

Verification and validation (V&V) are two critical features in software engineering that help to secure both the intended behaviour and quality of outputs. Verification is checking that the software correctly implements the required features from the specifications provided and validation ensures that however what was developed actually meets the user's needs and expectations. Traditional V&V processes are complemented with manual code reviews, static analysis and long test phases.[1]

These processes have been thoroughly revolutionised by the emergence of Large Language Models (LLMs), and AI-assisted tools that automate many V&V tasks. LLMs can be helpful in automatically creating new test cases, detecting bugs and even validating code by creating natural language descriptions of what a user expects the program to do based on the input and output and how it behaves within its intended environment.

They also use AI-powered tools for dynamic testing (validates the functionality of an application by executing the code) involving simulating users' interactions to reveal behind the scene issues [2]. This is very useful especially for regression testing (runs after every change to ensure that the change introduces no unintended breaks) or in a continuous integration environment since you need to validate the code changes through multiple scenarios [3]. Since AI can automate repetitive tasks, developers are free to concentrate on more complex aspects of software quality assurance like performance tuning and security analysis.

AI integration into V&V processes is extremely helpful as it boosts efficiency, reduces errors and helps create more reliable software systems, becoming a must have for the modern software development community.

What is a LLM

A Large Language Model (LLM) is an advanced artificial intelligence model extensively trained on diverse data sources, including books, code, articles, and websites. LLM models, such as GPT-3, BERT, CodeT5, and XLNet, explore inherent patterns and relationships within the language they are trained on, allowing them to produce coherent content, including grammatically accurate sentences, human-like paragraphs, and syntactically correct code snippets.[4]

We have different categories for the large language models. An encoder-only model works with an encoder network without a separate decoder network, taking an input sequence and mapping it to a lower-dimensional representation to learn and encoding the input data. Encoder-decoder models have an additional decoder network that generates an output sequence by iteratively generating tokens based on the context vector and previously generated tokens, used for tasks like machine translation or text generation. Decoder-only models lack an encoder component and generate the output based on a given context or input. It is used in architectures like autoregressive models, where the output is generated token by token, based on previously generated tokens and the context.[5]

Promises and guarantees

As of today, LLMs hold a great deal of promises in code generation, unit testing, bug detection and software debugging, but are still in exploratory phase with limited implementation.

LLMs have been successful in tasks like unit test generation, test oracle creation and bug repair. However, these are still not fully integrated into large-scale or production-level testing environments and aren't yet applied in the early stages of software testing process, test oracle problem, rigorous evaluations, and real-world application. Their integration in those fields are areas for future exploration.

While LLMs like GPT-3 and ChatGPT are promised to be capable of generating unit tests, the effectiveness still varies across different contexts, and the generated tests often require refinement and validation by human experts. Although LLMs can automatically repair buggy code, they are not yet fully reliable for complex bug fixes without human intervention.

LLMs could potentially automate the process of identifying and repairing code issues based on natural language descriptions or specific bugs. They're expected to increase test coverage, using techniques like differential and mutation testing to generate more test cases.

LLMs are leading the way in AI improvements, showcasing capabilities previously never thought possible. However, they still have difficulty understanding complex requirements specific to context, such as generating the entire function or script based on prompts, which tools like OpenAI's Codex can appear to do. The lack of clarity in this domain makes them impractical for deep understanding methods or large scale processing with existing software projects.

Additionally, the real-world application of LLMs is still limited. While they can automate a lot of the repetitive tasks (e.g. unit test creation), human oversight is necessary to ensure the machine's output is correct and comprehensive

While developers can save time on more important code and simple bug fixes, models to which they apply automated tests are not yet sophisticated enough for more complex software engineering tasks like performance optimization or secure coding practices.

LLMs coverage is still far from complete, therefore, combining LLMs with other techniques optimises their strengths and weaknesses to achieve better outcomes in specific scenarios.[6]

Current state of AI

LLMs have been used for various coding-related tasks including code generation and recommendation. In software testing, there are many tasks related to code generation, such as unit test generation, where the utilisation of LLMs is expected to yield good performance. LLMs are commonly used for test case preparation (including unit test case generation, test oracle generation, and system test input generation), program debugging, and bug repair.

An example of an AI for test driven development (TDD) is GAI4-TDD, a PyCharm plugin to support the Green Phase of TDD. TDD consists of three phases: Red Phase, where you write a unit test for a small chunk of functionality not yet implemented and watch the newly written test fail; Green Phase, where you make the newly written test pass as quickly as possible, committing whatever "sin" is necessary to do so, and watch all unit tests pass; Refactor Phase, where you refactor the code, thus remedying any "sin" previously committed, and watch all unit tests pass. GAI4-TDD generates production code from tests

written in the Red Phase, making these tests pass. It often successfully implements the required functionality in the Green Phase, allowing developers to focus on the Red and Refactor phases, which are often deemed less important or even skipped. Although it sometimes fails on the first try, this is common for humans too. Moreover, the generated code becomes progressively complex, encouraging refactoring.[7]

Another example is Bard, which generates text, translates languages and writes different kinds of creative content. It finds and shows information from Google tools like Gmail, Google Maps, Youtube in more than 40 languages.

MusicLM is a text-to-music model which can make music from text, images, video and humming.

Duet AI assists in Google Workspace and Google Cloud by helping users create images, analysing spreadsheets, drafting and summarising emails and chat messages, and summarising meetings.[8]

Some AIs such as GitHub's Copilot can have some applications in software development. It learns from existing code repositories to know how to complete code, while catching potential bugs and recommending refactoring in real-time. Errors are embedded in the Integrated Development Environment (IDE) like Visual Studio Code and PyCharm to give real-time recommendation and error detections.[5]

AI for software development is the latest and greatest evolution when it comes to AI technology for developers, encompassing not only LLMs such as GPT-4 but also other models and tools created to help in different stages of the software development lifecycle. For instance, the transformer model CodeT5 which is created to understand and generate code has made amazing progress in translating it from a programming language to another often producing more accurate comments to unannotated code, all very useful features for managing large codebases, and make code more readable and maintainable.[6]

One trend that has emerged over the last year is the use of AI in continuous integration and deployment (CI/CD) pipelines (series of steps that must be performed in order to deliver a new version of software). It predicts and resolves problems before they enter production, as well as automates the deployment of applications using AI.[9]

Tasks benefited by AI

LLMs are most commonly used in unit testing, followed by system testing. However, there is still no research on the use of LLMs in integration testing and acceptance testing.
LLMs can generate not only the source code for testing DL libraries but also the textual input for testing mobile apps, even the models for testing CPS. LLMs show remarkable potential in test case preparation, by generating good test inputs addressing challenges for better coverage, bug detection and repair, offering suggestions for code corrections and automating some debugging task, regression testing, automating the generation of new test cases when software is updated, and test execution and automation, since LLMs have learning-based capabilities enable them to adapt to different software systems.[6]

Studies tell us that most participants consider AI-based programming tools important for reducing programming effort and online searches for specific code snippets, programming syntax, or API calls, and to discover better alternatives or starting points for a solution. Additionally, almost half of the participants perceive AI-based programming assistants as important to auto-complete or reduce typing effort.

When asked to describe successful situations in which they used AI-based programming assistants, the main use-cases were the exploration of unfamiliar solutions to

problems such as "Problems I've never needed to solve before with programming" and for solution building, for code quality assurance and coding support. For coding support - the most frequent - participants described using this type of tool for assistance in understanding, optimising, or refactoring a snippet of code; adoption of cutting-edge technology or design pattern; and searching syntax examples.[10]

AI-based natural language processing tools can be used to automate the translation of user stories and requirements into technical specifications as part of the requirements gathering and analysis phase. The results will be smoother developments, by reducing the ambiguities for both sides and thus reducing misunderstandings between stakeholders and the development teams.

During the design and architecture stage, AI can help generate design patterns, recommend architectural backing up from historical data and industry standards. For example, an AI tool might suggest microservices architectures or serverless systems based on the expected load and use cases of an application.

AI in the testing phase is very helpful. For this type of problem AI will locate the pain points in code and thus be able to focus on these areas first during testing, hence improving both efficiency and effectiveness, while updating with the codebase and constantly creating new tests as the software grows.

Tools ready for use

The most commonly utilised LLM in software testing tasks is ChatGPT. Codex, an LLM based on GPT-3, is the second most commonly used LLM. There are already 14 studies that utilise GPT-4, ranking at the fourth place. Several studies directly utilise this state-of-the-art LLM of OpenAI, since it demonstrates excellent performance across a wide range of generation and reasoning tasks, such as utilising GPT-4 to generate fuzzing inputs and employing it to generate property-based tests with the assistance of API documentation.[6]

Fuzzing is an application security testing technique that feeds invalid inputs to a software program to expose vulnerabilities. It is often a black box testing technique that is carried out without the knowledge of the subject program's internal structure.[11]

The APITestGenie is a tool based in LLMs that uses OpenAPI to generate executable API test scripts from business requirements written in natural language and API specifications documents. It autonomously generates, improves and executes test cases.

APITestGenie is composed of three modular flows, Test Generation, Test Improvement and Test Execution. They can work independently but are designed to work together to make better solutions. The Generation flow picks the input from the API Specification and the business requirement and, with that information, generates a test script. The Test Improvement, based on previous test results and improvement user instructions, improves the generated script. The Execution flow executes the generated test script and reports the results.

LLMs models have the context window size limitation. To simplify the raw API specification, all the <img> tags content and the admin deprecated resources are removed. It also uses RAG (Retrieval Augmented Generation) to select the relevant information.[12]

Another example is Microsoft 365 Copilot that helps users resume, write and respond to questions. It works based on LLMs Pre-trained Generative Transformers (GPT), such as GPT-4. It uses information allowed from the user in Microsoft Graph as emails, conversations, and documents. It also works with Microsoft 365.

The Microsoft 365 Copilot can be described in five steps. It receives an order from Microsoft 365 and others, like from Word, Excel, PowerPoint, Outlook, Teams and Loop, pre-processes the incoming request with contextualization, which helps to get relevant requests, sends the contextualization request to the LLM that uses that to generate a response relevant to the user's task and accepts this response from LLM and performs post-processing. To finish, it makes a post-processing that includes contextualization calls to Microsoft Graph, responsible AI checks, security, compliance, and privacy reviews, and command generation.

The Copilot returns the answer to the user that can consult and evaluate the response. It has a history that can be reviewed by the user.

To understand better, Microsoft Graph includes information about the relationships between users, activities and data. It brings more information from customer signals, such as information from emails, conversations, documents, and meetings.[13]

Risks and challenges of using AI

The generated code from these models can sometimes be syntactically incorrect, leading to potential errors and reduced usability.

Although software testing with LLMs has undergone significant growth in the past two years, there are still challenges in achieving high coverage of the testing, test oracle problem, rigorous evaluations, and real-world application of LLMs in software testing. Adopting a human-computer interaction schema for tackling early-stage testing tasks would harness the domain-specific knowledge of human developers and leverage the general knowledge embedded in LLMs.

Integration testing requires diversified data to be generated to sufficiently test the interface among multiple modules. The size and complexity of the input data in this circumstance may exceed the capacity of the LLM to process and analyse, which can lead to errors or unreliable results.[6]

Although with AI assistance programmers developers are writing code faster, we have some studies that show us that a developer that uses an AI assistant to make code is more likely to write insecure code and more likely to rate their answers as secure compared to the programmers that don’t use AI assistant to generate code.[14]

Nowadays, some critics consider that developing and deploying solutions with hybrid techniques that use LLMs and Software Engineering are the best solution, because they consider hallucination a pervasive problem in LLMS. Hallucination means that the LLM can create fictitious outputs. That in engineering means that artefacts created could be incorrect, even when they appear plausible. GitHub's Copilot was an example of how to hide this problem since it only gives recommendations to the developer, instead of giving a complete and correct answer.

LLMs are non deterministics. This means that they can generate different answers in different executions. To avoid that and to provide robust conclusions, we use parametric and nonparametric statistics.

In conclusion, we can consider extensive programming dataset for training and evaluation, large and efficient-to-sample transformer-based architectures and Large-scale model sampling to explore the reach space, followed by behaviour-based filtering the main problems that are critical for LLMs to achieve reliable performance.[5]

The study “An Industry Case Study on Adoption of AI-based Programming Assistants“ asked participants about challenges associated with using AI right now. The

most recurrent challenges faced are out-of-context and non-functional code provided as answers by AI-based programming assistants. The most used technique by these users to cope with inaccurate responses is including detailed descriptions of the problem and the steps needed to solve it, so that their requests are contextualised.[10]

Conclusions

FunnyHow generated with AI a full commercial for Domino's to avoid the Burgerpizzas. This was the first 100% AI generated commercial made in Portugal.[15]

Nowadays, AI is used in almost everything. A recent report said that videos and avatars generated by AI have been used by the Ukrainian government to inform citizens about social programs and for public service announcements. This is a partnership between the Ukrainian government and Synthesia, a company based in London. This is a great example of AI that can be used in many situations, even in war scenery.[16]

AI tools used in writing and checking code, especially through the technology of Large Language Models (LLMs), have been a major disruption in traditional verification and validation (V&V) processes. LLMs automate numerous repetitive operations involved such as unit test generation, bug detection and code repair, among others, hence lifting the burden of developers with less sophisticated factors of software quality. While AI is further developed with tools like GAI4-TDD and GitHub's Copilot, models such as Codex and GPT-4 point towards its automatable use in later parts of the software development lifecycle.

In conclusion, AI in software development is still a new field, and has challenges of code integrity, test coverage and risk of generating bugs or bad code. LLMs also struggle to concatenate complicated landscapes, and human intervention may be needed for the following step of quality assurance in AI-generated outputs. However, hybrid methods using both LLMs with human expertise are proving to be the most effective. As the technology continues to evolve, LLMs are likely to become more sophisticated, making them indispensable tools in modern software engineering.

References

[1] Wikipedia contributors. (2024, September 12). *Verification and validation*. Wikipedia. https://en.wikipedia.org/wiki/Verification_and_validation

[2] *What is Dynamic Testing? (Types and Methodologies)* | *BrowserStack*. (2023, February 15). BrowserStack. https://www.browserstack.com/guide/dynamic-testing

[3] Daityari, S. (2023, June 24). *Regression Testing : Definition, How it works* | *BrowserStack*. BrowserStack. https://www.browserstack.com/guide/regression-testing

[4] Santos, R., Santos, I., Magalhaes, C., & De Souza Santos, R. (2024). Are we testing or being tested? Exploring the practical applications of large language models in software testing. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)* (pp. 353–360). https://doi.org/10.1109/icst60714.2024.00039

[5] Fan, A., Generative AI Team, Gokkaya, B., PyTorch Team, Harman, M., Instagram Product Foundation, Lyubarskiy, M., Developer Infrastructure, Sengupta, S., FAIR, Yoo, S., School of Computing, KAIST, Zhang, J. M., & Department of Informatics, King's College London. (2023). 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)* (p. 31). https://doi.org/10.1109/ICSE-FoSE59343.2023.00008

[6] Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., & Qing Wang. (2024). Software Testing With Large Language Models: Survey, Landscape, and Vision. In IEEE, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING* (Vol. 50, Issue 4, pp. 911–912) [Journal-article]. https://doi.org/10.1109/TSE.2024.3368208
[7] Cassieri, P., Romano, S., & Scanniello, G. (2024). Generative Artificial Intelligence for Test-Driven Development: GAI4- TDD. *2024 IEEE Conference on Software Testing, Verification and Validation (SANER)*. https://doi.org/10.1109/saner60148.2024.00098
[8] *2023: a year of groundbreaking advances in AI and computing*. (2024, September 26). Google DeepMind. https://deepmind.google/discover/blog/2023-a-year-of-groundbreaking-advances-in-ai-and-computing/
[9] *What is a CI/CD pipeline?* (n.d.-b). https://www.redhat.com/en/topics/devops/what-cicd-pipeline
[10] Davila, N., Wiese, I., Steinmacher, I., Da Silva, L. L., Kawamoto, A., Favaro, G. J. P., & Nunes, I. (2024). An Industry Case Study on Adoption of AI-based Programming Assistants. *2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE- SEIP)*. https://doi.org/10.1145/3639477.3643648
[11] TechTarget, "Fuzz testing," *SearchSecurity* (2024). https://www.techtarget.com/searchsecurity/definition/fuzz-testing
[12] Pereira, A., Lima, B., & Faria, J. P. (2024). APITestGenie: Automated API Test Generation through Generative AI. In IEEE Computer Society, *IEEE Computer Society* [Journal-article]. https://doi.org/10.1109/XXX.0000.0
[13] Camillepack. (2024, September 27). *Descrição geral do Microsoft 365 Copilot*. Microsoft Learn. https://learn.microsoft.com/pt-pt/copilot/microsoft-365/microsoft-365-copilot-overview
[14] Nash, P. (2024, April 2). Avoiding the dangers of AI-generated code. *InfoWorld*. https://www.infoworld.com/article/2336657/avoiding-the-dangers-of-ai-generated-code.html
[15] M. (2024, September 27). IA ‘assina’ anúncio da FunnyHow para Domino’s. *ECO*. https://eco.sapo.pt/2024/09/27/ia-assina-anuncio-da-funnyhow-para-dominos/
[16] Davies, P. (2024, September 27). Governo da Ucrânia vai utilizar avatares e vídeos com IA para ajudar a informar cidadãos sobre assistência social. *Euronews*. https://pt.euronews.com/next/2024/09/27/governo-da-ucrania-vai-utilizar-avatares-e-videos-com-ia-para-ajudar-a-informar-cidadaos-s