AI Assisted Programming
(AISoLA 2023 Track Introduction)

Wolfgang Ahrendt[1] and Klaus Havelund[2]*

[1] Chalmers University of Technology, SE
ahrendt@chalmers.se
[2] NASA Jet Propulsion Laboratory, California Inst. of Technology, US
klaus.havelund@jpl.nasa.gov

Abstract. The paper is an introduction to the track 'AI Assisted Programming', organized at the AISoLA conference during the period October 23-28, 2023. The theme of AISoLA 2023 is: 'Bridging the Gap Between AI and Reality'. The motivation behind the track is the emerging use of Large Language Models for construction and analysis of software artifacts. An overview of the track presentations is provided.

1 Introduction

Neural program synthesis, using Large Language Models (LLMs) which are trained on open source code, are quickly becoming a popular addition to the software developer's toolbox. Services like, for instance, OpenAI's ChatGPT [8], Google's Bard [6], and GitHub's Copilot [5] can generate code in many different programming languages from natural language requirements. This opens up for fascinating new perspectives, such as increased productivity and accessibility of programming also for non-experts. However, neural systems do not come with guarantees of producing correct, safe, or secure code. They produce the most probable output, based on the training data, and there are countless examples of coherent but erroneous results. Even alert users fall victim to automation bias: the well studied tendency of humans to be over-reliant on computer generated suggestions. The area of software development is no exception to this automation bias.

The track *AI Assisted Programming* at AISoLA 2023 is devoted to discussions and exchange of ideas on questions like: What are the capabilities of this technology when it comes to software development? What are the limitations? What are the challenges and research areas that need to be addressed? How can we facilitate the rising power of code co-piloting while achieving a high level of correctness, safety, and security? What does the future look like? How should these developments impact future approaches and technologies in software development and quality assurance? What is the role of models, tests, specification,

* The research performed by this author was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

verification, and documentation in conjunction with code co-piloting? Can quality assurance methods and technologies themselves profit from the new power of LLMs?

2 Contributions

These questions are taken up by the participants of the track in 10 talks. Three talks [2,3,4] were associated with regular papers, one talk [7] was associated with an extended abstract, and one talk [1] with a one page abstract. The remaining five talks do not have associated papers in the proceedings. Presenters have been offered to publish regular papers in a subsequent post-conference proceedings.

2.1 Talks with Papers in the Proceedings

Lenz Belzner, Thomas Gabor, and Martin Wirsing [2] (*Large Language Model Assisted Software Engineering: Prospects, Challenges, and a Case Study*) discuss the potential benefits and challenges associated with the adoption of LLMs in software engineering. They explore the opportunities for requirements engineering, system design, code generation, test generation, code reviews, and software processes. The paper includes a comprehensive review of the state-of-the-art of the use of LLMs for software development. A case study is presented, illustrating the prompt-based development of a simple "search and rescue" application.

Daniel Busch, Gerrit Nolte, Alexander Bainczyk, and Bernhard Steffen [3] (*ChatGPT in the Loop: A Natural Language Extension for Domain-Specific Modeling Languages*) present an approach that combines the use of Domain-Specific Languages (DSLs) and natural language prompting using LLMs. The user provides first a context in the form of a DSL model, creating a frame in which ChatGPT can generate code that fits the code skeleton generated from the model. The resulting code is verified using automata learning and subsequent model checking. The ideas are demonstrated with the development of a classical river crossing game.

Itay Cohen and Doron Peled [4] (*Integrating Distributed Component-Based Systems through Deep Reinforcement Learning*) present the idea of using deep reinforcement learning to learn how concurrently executing components can communicate in a more optimal way, in order to avoid failed communication attempts, where one components attempts to communicate with another component, which, however, is not willing to communicate at that moment. The components are considered "black boxes", where their internal structure is not known, and the learning is performed in a distributed manner.

Moa Johansson [7] (*What can Large Language Models do for Theorem Proving and Formal Methods?* - extended abstract) investigates how to best combine the capabilities of LLMs with symbolic verification systems such as theorem provers. LLMs are noted to be unreliable and prone to hallucinate, also in mathematical reasoning. It is suggested that a more reliable way is to let the LLM provide inputs, specifically invent lemmas and conjectures, to a theorem prover, which can then do the formal reasoning. In a case study it is explored how GPT-4 performs on lemma discovery for the Isabelle/HOL proof assistant.

Bernhard K. Aichernig and Klaus Havelund [1] (*AI-Assisted Programming with Test-based Refinement* - abstract) explore the idea of program development in Scala by refinement using ChatGPT. The authors refine a classic bridge controller, originally used as an example illustrating Event-B, through several steps, each generated by ChatGPT from natural language prompts. The refinements are tested using refinement mappings and property-oriented testing with ScalaCheck. This in contrast to the Event-B effort, which proves the refinements correct.

2.2 Talks without Papers in the Proceedings

Wolfgang Ahrendt, Dilian Gurov, Moa Johansson, and Philipp Rümmer (*TriCo - LLM supported Development of Robust Software*) suggest an agile software development workflow which addresses the commonly seen lack of trust in code generated by LLMs. The proposed approach, named TriCo (Triple Co-piloting), integrates in an IDE a LLM with formal methods, automated testing, and machine learning. A change in one of these three artifacts, will cause the IDE to suggest changes to the other two artifacts, keeping them consistent.

Saddek Bensalem, Kaiwen Cai, Yi Dong, Andre Freitas, Wei Huang, Xiaowei Huang, Gaojie Jin, Ronghui Mu, Mustafa A. Mustafa Yi Qi, Wenjie Ruan, Changshun WU, Dengyu Wu, Sihao Wu, Peipei Xu, Yanghao Zhang, and Xingyu Zhao (*A Survey of Safety and Trustworthiness of Large Language Models through the Lens of Verification and Validation*) present a survey exploring the safety and trustworthiness of LLMs. The authors review and categorize known vulnerabilities of LLMs. They then investigate if and how verification and validation techniques developed for traditional software and deep learning models can be integrated during the life-cycle of LLMs to make them safer and more trustworthy.

Dirk Beyer (*Software Verification in the Presence of Generated Programs*) discusses the problem of formally verifying that a program generated by a LLM satisfies a formal specification. In order to prove this, it is commonly necessary to formulate and prove program invariants. The talk focuses on automatic construction of such invariants as first-class interchangeable objects, not just code annotations, and their automatic verification. It is also discussed how to produce comprehensible error reports when a specification is violated.

Dan Boneh, Deepak Kumar, Neil Perry, and Megha Srivastava (*Do Users Write More Insecure Code with AI Assistants?*) conduct a study examining how users interact with an AI Code assistant to solve a variety of security related tasks across different programming languages. The authors observe that the uncritical use of an AI code assistant generally results in less secure code. Additionally, users with access to an AI assistant seem more likely to believe they write secure code. The authors perform an analysis of the users' language and interaction behaviours, and release an interface to conduct similar studies in the future.

Martin Leucker and Gerardo Schneider (*Some Experiments in Chatbot-Assisted Program Development*) report on experiments with ChatGPT. In particular, the authors explore the use of ChatGPT to generate simple programs, point out deficiencies in programs, generate test cases, generate temporal logic formulae, and deal with automata specifications. Furthermore it is shown how to build a simple chatbot that can run dedicated analysis tools, such as a model checker, locally, as a step towards full-scale chatbot-assisted program development.

3 Conclusion

The presentations in this track cover the use of LLMs in the context of all phases of software development, including requirements, designs, coding, testing and verification. This includes their use in combination with specification languages and domain-specific languages. It is explored how LLMs can be used to support formal methods and testing, and in the other direction it is explored how these techniques can support the use of LLMs, both wrt. safety, security, and correctness of software. Other machine learning topics are covered as well. Some case studies are furthermore presented. This covers an already interesting spectrum of AI assisted programming at this very early stage of LLMs. We hope that this track, with its talks, discussions, and papers, contributes to a future of AI assisted programming which exploits the strengths of arising AI technologies while mitigating the corresponding risks. We are convinced that many communities within computing have a lot to contribute to such a development, and look forward to future initiatives and contributions towards this aim.

References

1. B. K. Aichernig and K. Havelund. AI-assisted programming with test-based refinement. In *Proc. of AISoLA 2023 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, LNCS. Springer International Publishing, 2023. [in this volume].
2. L. Belzner, T. Gabor, and M. Wirsing. Large language model assisted software engineering: Prospects, challenges, and a case study. In *Proc. of AISoLA 2023 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, LNCS. Springer International Publishing, 2023. [in this volume].
3. D. Busch, G. Nolte, A. Bainczyk, and B. Steffen. ChatGPT in the loop: A natural language extension for domain-specific modeling languages. In *Proc. of AISoLA 2023 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, LNCS. Springer International Publishing, 2023. [in this volume].
4. I. Cohen and D. Peled. Integrating distributed component-based systems through deep reinforcement learning. In *Proc. of AISoLA 2023 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, LNCS. Springer International Publishing, 2023. [in this volume].
5. GitHub. Copilot. https://copilot.github.com, 2023. Accessed: August 27, 2023.
6. Google. Bard. https://bard.google.com, 2023. Accessed: August 27, 2023.

7. M. Johansson. What can large language models do for theorem proving and formal methods? In *Proc. of AISoLA 2023 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, LNCS. Springer International Publishing, 2023. [in this volume].
8. OpenAI. ChatGPT. https://chat.openai.com, 2023. Accessed: August 27, 2023.