Molecular Distributions and Abundances in the Binary-shaped Outflow of V Hya

Downloaded from: https://research.chalmers.se, 2025-03-28 00:48 UTC

Citation for the original published paper (version of record):
Siebert, M., Sahai, R., Scibelli, S. et al (2025). Molecular Distributions and Abundances in the Binary-shaped Outflow of V Hya. Astrophysical Journal, 979(2).
http://dx.doi.org/10.3847/1538-4357/ad8e34

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

AI Assisted Programming
(AISoLA 2024 Track Introduction)

Wolfgang Ahrendt[1], Bernhard K. Aichernig[2], and Klaus Havelund[3](✉)

[1] Chalmers University of Technology, Goteborg, Sweden
ahrendt@chalmers.se
[2] Institute of Software Technology, Graz University of Technology, Graz, Austria
aichernig@ist.tugraz.at
[3] NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA
klaus.havelund@jpl.nasa.gov

Abstract. This is an introduction to the track 'AI Assisted Programming' (AIAP), organized at the second instance of the AISoLA conference during the period October 30 - November 3, 2024. AISoLA as a whole aims to study opportunities and risks of late advances of AI. The motivation behind the AIAP track in particular, which also takes place the second time, is the emerging use of large language models for the 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, have quickly become a popular addition to the software developer's toolbox. Services like, for instance, OpenAI's ChatGPT [8], Google's Bard [7], and GitHub's Copilot [6] 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 2024 is the second of its kind, after the first instance in 2023 [1]. It is devoted to discussions and exchange of ideas on questions like: What are the capabilities of this technology when it

K. Havelund—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.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
B. Steffen (Ed.): AISoLA 2024, LNCS 15217, pp. 101–106, 2025.
https://doi.org/10.1007/978-3-031-75434-0_7

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, 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

The above questions are taken up by the participants of the track in eleven talks. Four talks [2–5] are associated with regular papers. The remaining seven talks do not have associated papers in the proceedings. Presenters have been offered to publish regular papers in subsequent post-conference proceedings.

2.1 Talks with Papers in the Proceedings

Gerhard Stenzel, Kyrill Schmid, Michael Kölle, Philipp Altmann, Marian Lingsch-Rosenfeld, Maximilian Zorn, Tim Bücher, Thomas Gabor, Martin Wirsing, and Lenz Belzner (*SEGym: Optimizing Large Language Model Assisted Software Engineering Agents with Reinforcement Learning* [4]) propose a new approach for software development agents based on LLMs. They model software development agents over LLMs as partially observable Markov decision processes to enable data-driven optimization. This is in contrast to the currently dominating approach for LLM based software development agents, which is largely heuristic. The work simplifies the setup of optimization experiments for software development agents, making it more accessible for researchers engaging in this field.

Minal Suresh Patil, Gustav Ung, and Mattias Nyberg (*Towards Specification-Driven LLM-Based Generation of Embedded Automotive Software* [3]) present an approach for generating C-code from specifications written in the specification language ACSL. The approach uses LLMs to generate code candidates from ACSL, and employs the Frama-C tool for verifying the correctness of the generated code. The feedback from potentially failed proof attempts is then used to fine-tune the prompt for a new LLM query. This loop can be iterated until the code meets the specification. The paper also reports on industrial case studies from the heavy vehicle manufacturer Scania, where a one-iteration instance of the approach is applied to generate code of critical embedded software units.

Amer Tahat, David Hardin, Adam Petz, and Perry Alexander (*Proof Repair Utilizing Large Language Models: A Case Study on the Copland Remote Attestation Proofbase* [5]) introduce the CoqDog Copilot, which leverages the neurosymbolic interplay between generative AI and the Coq theorem prover to form a "generate-and-test" loop for proving Coq lemmas provided as prompts. The proofs are generated incrementally based on failure information and human

hints until valid proofs are achieved. The authors also define metrics for measuring proof repair progress, and an evaluation system for quality assessment. The authors provide an evaluation of CoqDog Copilot's performance in proof generation across multiple samples from the Copland Coq proofbase, a domain-specific language for developing attestation protocols, which consists of a total of 21,000 lines of Coq code. The approach is robust in the sense that Coq can formally verify the output from the LLM.

Itay Cohen and Doron Peled (*LLM-based Scheme for Synthesis of Formal Verification Algorithms* [2]) present an approach to LLM code generation by first providing an underlying programming pattern as a prompt, followed by prompts requesting the generation of programs for specific problems that can be solved using this pattern. The pattern in this case is dynamic programming, which involves an algorithm where components are structured within a graph, with each component possessing values that impact and are affected by neighboring components. The LLM is subsequently requested to generate code for both model checking and runtime verification algorithms that utilize the common underlying dynamic programming scheme. The approach is compared with an approach where the dynamic programming pattern is not initially provided, which does not perform as well.

2.2 Talks Without Papers in the Proceedings

Moez Ben Hajhmida and Edward A. Lee (*Context Engineering for AI-Assisted Programming for Domain-Specific Languages*) report on experiments with using LLMs to write Lingua Franca (LF) programs. LF is a polyglot coordination language, where the logic of concurrent components is written in C, C++, Python, Rust, or TypeScript, and the architecture and communication between components is specified using the domain-specific syntax of LF. The authors observe that this situation is challenging for the LLMs because context data, such as user documentation and programming examples, mix the LF syntax with the programming languages. The problem addressed is that LLMs have mostly been trained on only traditional programming languages, and less so on emerging languages and domain-specific languages such as LF.

George Granberry, Wolfgang Ahrendt, and Moa Johansson (*Specify What? Enhancing Neural Specification Synthesis by Symbolic Methods*) study the impact of symbolic analysis results on LLM generation of specifications from code. As a base line, the LLM is prompted with C code and asked to generate ACSL specifications. In two alternative setups, the results of two different symbolic analyses are added to the prompts. In the first, the Frama-C tool Pathcrawler is used to generate input-output pairs. In the second, the Frama-C tool EVA performs value analysis for avoidance of run-time errors. In both cases, the respective output is added to the LLM prompt. The results show that adding symbolic analysis results to the prompts reduces the quantity of specification annotations, while the quality of the annotations is increased, in the sense that the generated specifications become more abstract, intentional, and focused. By applying the same techniques to buggy mutants of the code examples, the reported experiments also

show that the LLM generates specifications which reflect the intended behaviour better than the actual (buggy) behaviour of the code.

Daniel Busch, Maximilian Schlüter, and Bernhard Steffen (*Automation vs Autonomy*) observe that LLMs demonstrate a potential for automation but ask the question whether LLMs or future AI systems can achieve true autonomy. The authors point out that answering this question requires a clear distinction between automation and autonomy. True autonomy requires that improvement is achieved by the AI system itself, without external assistance. Alan Turing proposed the Turing Test to distinguish machines from humans through conversation. The authors propose a new class of tests to establish the boundary between automation and autonomy through formal methods-based behavioral analysis, and an AI-Workbench, based on process algebra, abstract interpretation, and automata learning, to study AI systems and their capability to achieve autonomy.

Tom van Dijk and Vadim Zaytsev (*Generative Artificial Intelligence Tools in Project-Based Learning*) report on a pilot study on using Generative Artificial Intelligence (GAI) to complete a large programming assignment in an introductory Java course. Students were divided into four groups: a control group, a group allowed to use Github Copilot, a group allowed to use ChatGPT, and a group allowed to use both tools. The participants maintained a reflective journal, the results were assessed using the official assessment rubric of the course and the projects were discussed in focus group meetings. The goal of the study was to understand the extent to which students can rely on GAI tools to complete their assignment, to investigate the impact on student understanding of fundamental programming concepts, and to explore the implications of integrating AI assisted coding tools in the learning process.

Johan Martinson, Yannic Noller, and Thorsten Berger (*On Using Large Language Models to 'Featurize' Software*) present some experiments with software featurization, which involves identifying and documenting distinctive characteristics of a software system, in order to facilitate transformation of the software to make it more customizable and adaptable to different situations. The study explores AI assisted programming for this purpose, evaluating LLMs like LLAMA and CodeBERT. Through experiments, this research aims to enhance developer productivity by automating parts of the featurization process. The study focuses on refactoring code to make features variable and involves literature reviews and industry interviews to determine essential criteria for successful featurization.

Bernhard Aichernig and Klaus Havelund (*Correct-ish by Design: From Upfront Verification to Continuous Monitoring of LLM Code*) argue that as developers increasingly rely on Large Language Models (LLMs) to generate code, the pace of software development is accelerating beyond the capabilities of traditional design-time verification and testing methods. The authors predict a paradigm shift towards continuous monitoring to complement and eventually supersede upfront verification. By embracing a "correct-ish by design" philosophy, they acknowledge the inevitability of imperfections in LLM-generated code.

They advocate for an adaptive approach where real-time monitoring and feedback mechanisms are employed to detect, diagnose, and rectify issues as they emerge in the field. The authors experiment with the integration of automated monitoring and testing tools.

Ezio Bartocci (*AI-Assisted Requirements Specification*) presents an overview of his and his co-workers' approaches to addressing the complexity of translating requirements from different formats, including natural language descriptions, timing diagrams and raw data, into formal specification languages such as temporal logics, timing diagrams, and state machines. He argues that AI assisted techniques have the potential to bridge the gap between requirements representations preferred by different stakeholders and their precise definition offered by formal methods. This is essential as the definition of requirements is a crucial task along all the lifecycle of cyber-physical engineering from their model-based design, simulation and testing, to their deployment and runtime monitoring.

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 verification methods, and in the other direction it is explored how verification methods can support the use and evaluation of LLMs. 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. Ahrendt, W., Havelund, K.: AI assisted programming. In: Steffen, B. (ed.) Bridging the Gap Between AI and Reality. LNCS, vol. 14380, pp. 351–354. Springer (2024)
2. Cohen, I., Peled, D.: LLM-based scheme for synthesis of formal verification algorithms. In: Proceedings of AISoLA 2024 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming, LNCS. Springer (2024). [in this volume]
3. Patil, M.S., Ung, G., Nyberg, M.: Towards specification-driven LLM-based generation of embedded automotive software. In: Proceedings of AISoLA 2024 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming. LNCS. Springer (2024). [in this volume]
4. Stenzel, G., et al.: SEGym: optimizing large language model assisted software engineering agents with reinforcement learning. In: Proceedings of AISoLA 2024 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming, LNCS. Springer (2024). [in this volume]

5. Tahat, A., Hardin, D., Petz, A., Alexander, P.: Proof repair utilizing large language models: a case study on the Copland remote attestation proofbase. In: Proceedings of AISoLA 2024 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming, LNCS. Springer (2024). [in this volume]
6. Web GitHub. Copilot (2024). https://copilot.github.com. Accessed 28 July 2024
7. Web Google. Bard (2024). https://bard.google.com. Accessed 28 July 2024
8. Web OpenAI. ChatGPT (2024). https://chat.openai.com. Accessed 28 July 2024