AI Assisted Programming
(AISoLA 2025 Post-Proceedings Track Introduction)

Wolfgang Ahrendt[1], Bernhard K. Aichernig[2], and Klaus Havelund[3⋆]

[1] Chalmers University of Technology and University of Gothenburg, Sweden
ahrendt@chalmers.se
[2] Inst. for Formal Models and Verification, Johannes Kepler University Linz, Austria
bernhard.aichernig@jku.at
[3] NASA Jet Propulsion Laboratory, California Inst. of Technology, USA
klaus.havelund@jpl.nasa.gov

Abstract. This is an introduction to the post-proceedings for the track 'AI Assisted Programming' (AIAP), organized at the third instance of the AISoLA conference during the period November 1 - 5, 2025. 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 took place the third 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. LLMs like, for instance, OpenAI's ChatGPT and o1/o3 reasoning models [15], Anthropic's Claude [16], Google's Gemini [18], xAI's Grok [19], Meta's LLaMA [20], DeepSeek [17]; and various LLM enhanced IDEs such as Copilot [22], Cursor [23], Devin [24], and Claude Code [21], 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 2025, November 1-5, 2025 on the Greek island of Rhodes, was the third of its kind, after the first instance

⋆ 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.

in 2023 [2] and the second instance in 2024 [1]. It was 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, 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 sixteen talks. Five talks are associated with papers in these post-proceedings. Six talks are associated with papers in the previously published meeting proceedings available at the conference. The remaining five talks do not have associated papers. In the following we provide a survey of the contributions, with summaries of the papers published in these proeceedings.

2.1 Papers in these Proceedings

The following papers are included in these proceedings.

From Natural Language to Monitors via LLM-Assisted Disambiguation [3]. Itay Cohen, Klaus Havelund, Doron Peled, and Yoav Goldberg present an LLM-based method and interactive tool for synthesizing runtime verification monitors from natural-language specifications. The tool addresses key challenges: bridging the gap between natural language and temporal logic, handling descriptions beyond standard temporal logic expressiveness, and resolving ambiguities. The approach operates in two stages. First, it helps users define a collection of monitorable temporal operators in natural language, assisting in choosing among possible interpretations to create a custom temporal logic with a corresponding monitor library. Second, users define application-specific monitors as compositions of library monitors, with the tool comparing automata to help select intended interpretations, increasing reliability in synthesizing monitors from natural-language specifications.

Agentic Software Engineering is a Search Problem [5]. Moritz Gärtner, Lenz Belzner, Thomas Gabor, and Martin Wirsing frame agentic software development as a strategic search problem, where Large Language Model agents operate as stochastic algorithms traversing a complex solution space defined by formal specifications and environmental constraints. By operationalizing development through iterative feedback loops and tool-integrated search operators, these systems shift human expertise toward formulating precise objectives and validating search outcomes. They introduce LIFT (LLM-based iterative feedback-driven test suite generation), an agentic framework treating test suite evolution as a

directed search process. A case study on the *simplejson* library shows LIFT effectively achieves near-complete structural coverage but remains constrained by the oracle problem and gaps between code execution and semantic validation.

Model-Driven Formally Verified LLM Program Synthesis for Solidity Smart Contracts [10]. Jonas Schiffl, Samuel Teuber, and Bernhard Beckert present an approach to automatically synthesize formally verified Solidity smart contracts using LLMs within the Scar model-driven verification-based development process. In Scar, developers first create an abstract model with security and correctness properties, from which a formally specified code skeleton is generated. Traditionally, developers manually implement this skeleton and verify it against the specification. Here, an LLM generates the implementation directly from the specification, followed by automated formal verification using Certora and solc-verify. If verification succeeds, the code is guaranteed correct. The method is evaluated on multiple use cases, comparing both verification tools and reporting practical insights.

LLM-Assisted Translation and Bounded Model Checking of Python Code [11]. Shivkumar Shivaji, Klaus Havelund, Natalia Lobakhina, Lucas C. Cordeiro, and Alessandro Pinto present an approach that uses an LLM orchestrator to iteratively coordinate static, dynamic, and formal verification tools for verifying Python programs. The system translates Python code into C suitable for formal verification using ESBMC, enabling detection of critical bugs including arithmetic overflows, array bounds violations, and concurrency errors. They evaluate their approach on a benchmark of 23 carefully designed Python programs (15-50 lines) with planted bugs representing common verification challenges. The LLM orchestrator successfully detected all planted bugs, demonstrating that LLM-assisted translation can make mature C verification tools accessible for Python code analysis.

AI-based Code Generation with Feedback from Formal Analysis [14]. Cheng Wang, Florian Lorber, Edi Muškardin, Bernhard Aichernig, and Martin Leucker propose a formal evaluation method for LLM code generation, using finite-state machines as ground truth specifications. These models are automatically translated into natural language descriptions and provided to an LLM, which generates Python programs intended to match the original behavior. The generated programs are then analyzed with active automata learning (using the AALpy automata learning library) to infer their input–output behavior and compare it against the ground truth, producing a similarity score. The method also supports iterative repair of faulty code using counterexamples. Initial experiments with four popular LLMs on randomly generated Mealy machines reveal differences in accuracy and robustness.

2.2 Papers in the Meeting Proceedings

The following papers were included in the previously published meeting proceedings [12] available at the conference.

- *LLM-based Property-based Test Generation for Guardrailing Cyber-Physical Systems* [4]. Khashayar Etemadi, Marjan Sirjani, Mahshid Helali Moghadam, Per Strandberg, and Paul Pettersson
- *RAG and Agentic Assistant: A Combined Approach* [6]. Moez Ben Hajhmida and Edward A. Lee
- *CASP: An Evaluation Dataset for Formal Verification of C code* [7]. Niclas Hertzberg, Merlijn Sevenhuijsen, Liv Kåreborn, and Anna Lokrantz
- *A Voice-Enabled Query Framework for Systems Engineering Artefacts* [8]. Lennart Landt, Martin Leucker, and Carsten Burchardt
- *Integrating LLMs with QC-OpenDRIVE: Ensuring Normative Correctness in Autonomous Driving Scenarios* [9]. Julian Müller, Thies de Graaff, and Eike Möhlmann
- *AGREE-Dog Copilot: A Neuro-Symbolic Approach to Enhanced Model-Based Systems Engineering* [13]. Amer Tahat, Isaac Amundson, David Hardin, and Darren Cofer

2.3 Talks without Papers in Proceedings

The following talks were given without associated papers.

- *PolyVer: A Compositional Approach for Polyglot System Modeling and Verification.* Pei-Wei Chen, Shaokai Lin, Adwait Godbole, Ramneet Singh, Elizabeth Polgreen, Edward Lee, and Sanjit Seshia
- *Techniques and Experiments in Retrieval-Augmented Neural Theorem Proving.* João F. Ferreira
- *TD-Interpreter: Enhancing the Understanding of Timing Diagrams with Visual-Language Learning.* Jie He, Vincent Theo Willem Kenbeek, Zhantao Yang, Meixun Qu, Ezio Bartocci, Dejan Ničković, and Radu Grosu
- *ChatGPT in the Loop - Revisited.* Marco Krumrey, Daniel Busch, and Bernhard Steffen
- *LLM-Assisted Program Correctness: Generating Lemmas, Assertions, and Repairs in Dafny.* Alexandra Mendes

3 Conclusion

The presentations in this track covered the use of LLMs in the context of all phases of software development, including requirements, designs, coding, testing and verification. This included such topics as LLM support for specification generation, test-case generation, runtime verification, formal verification, automated repair, translation of high-level design models and specifications to code, translation between programming languages, human comprehension of models, and benchmarks. This covered an 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. W. Ahrendt, B. Aichernig, and K. Havelund. AI assisted programming. In B. Steffen, editor, *AISoLA 2024 - Bridging the Gap Between AI and Reality*, volume 15217 of *LNCS*, pages 101–106. Springer International Publishing, 2025.
2. W. Ahrendt and K. Havelund. AI assisted programming. In B. Steffen, editor, *AISoLA 2023 - Bridging the Gap Between AI and Reality*, volume 14380 of *LNCS*, pages 351–354. Springer International Publishing, 2024.
3. I. Cohen, K. Havelund, D. Peled, and Y. Goldberg. From natural language to monitors via LLM-assisted disambiguation. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming (Post Proceedings)*, LNCS. Springer International Publishing, 2026. [in this volume].
4. K. Etemadi, M. Sirjani, M. H. Moghadam, P. Strandberg, and P. Pettersson. LLM-based property-based test generation for guardrailing cyber-physical systems. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, volume 16220 of *LNCS*. Springer International Publishing, 2025.
5. M. Gärtner, L. Belzner, T. Gabor, and M. Wirsing. Agentic software engineering is a search problem. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming (Post Proceedings)*, LNCS. Springer International Publishing, 2026. [in this volume].
6. M. B. Hajhmida and E. A. Lee. RAG and agentic assistant: A combined approach. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, volume 16220 of *LNCS*. Springer International Publishing, 2025.
7. N. Hertzberg, M. Sevenhuijsen, L. Kåreborn, and A. Lokrantz. CASP: An evaluation dataset for formal verification of C code. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, volume 16220 of *LNCS*. Springer International Publishing, 2025.
8. L. Landt, M. Leucker, and C. Burchardt. A voice-enabled query framework for systems engineering artefacts. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, volume 16220 of *LNCS*. Springer International Publishing, 2025.
9. J. Müller, T. de Graaff, and E. Möhlmann. Integrating LLMs with QC-OpenDRIVE: Ensuring normative correctness in autonomous driving scenarios. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, volume 16220 of *LNCS*. Springer International Publishing, 2025.
10. J. Schiffl, S. Teuber, and B. Beckert. Model-driven formally verified LLM program synthesis for solidity smart contracts. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming (Post Proceedings)*, LNCS. Springer International Publishing, 2026. [in this volume].
11. S. Shivaji, K. Havelund, N. Lobakhina, L. C. Cordeiro, and A. Pinto. LLM-assisted translation and bounded model checking of Python code. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted*

Programming (Post Proceedings), LNCS. Springer International Publishing, 2026. [in this volume].
12. B. Steffen, editor. *AISoLA 2025 - Bridging the Gap Between AI and Reality*, volume 16220 of *LNCS*. Springer International Publishing, 2026.
13. A. Tahat, I. Amundson, D. Hardin, and D. Cofer. AGREE-Dog Copilot: A neuro-symbolic approach to enhanced model-based systems engineering. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming*, volume 16220 of *LNCS*. Springer International Publishing, 2025.
14. C. Wang, F. Lorber, E. Muškardin, B. K. Aichernig, and M. Leucker. AI-based code generation with feedback from formal analysis. In B. Steffen, editor, *Proc. of AISoLA 2025 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming (Post Proceedings)*, LNCS. Springer International Publishing, 2026. [in this volume].
15. ChatGPT LLMs (OpenAI). https://chat.openai.com.
16. Claude LLMs (Anthropic). https://claude.ai.
17. DeepSeek LLMs. https://www.deepseek.com.
18. Gemini LLMs (Google). https://gemini.google.com.
19. Grok LLMs (xAI). https://grok.com.
20. LLaMA LLMs (Meta). https://www.llama.com.
21. Claude code IDE. https://claude.ai/code.
22. GitHub copilot. https://github.com/features/copilot.
23. Cursor IDE. https://cursor.com.
24. Devin desktop. https://devin.ai/desktop.

Disclaimer