AI Assisted Programming
(AISoLA 2026 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 proceedings for the track 'AI Assisted Programming' (AIAP), organized at the fourth instance of the AISoLA conference during the period October 27 - 31, 2026. AISoLA as a whole aims to study opportunities and risks of late advances of AI. The motivation behind the AIAP track in particular 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 its reasoning models [8], Anthropic's Claude [9], Google's Gemini [11], xAI's Grok [12], Meta's LLaMA [13], DeepSeek [10]; and various LLM-enhanced IDEs and coding agents such as Copilot [15], Cursor [16], Devin [17], and Claude Code [14], 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 2026, October 27-31, on the Greek island of Kos, was the fourth of its kind, after the first instance in 2023 [3], the second instance in 2024 [1], and the third instance in 2025 [2].

* 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.

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, specifications, 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. Four talks are associated with papers in these proceedings. Other talks given in the track will be associated with papers in subsequent post meeting proceedings. In the following we provide a survey of the contributions in these proceedings.

From Maritime Regulations to Verifiable Properties with LLM Assistance [4]. Masoud Ebrahimi, Jackiie Samuelsson, Carolina Smigan, Abu Naser Masud, Robbert Jongeling, and Marjan Sirjani present an approach for transforming maritime collision-avoidance regulations (COLREGs), encoded in the Legata domain-specific language, into formal safety properties suitable for model checking Timed Rebeca actor models. The method is realized as a four-stage pipeline in which the first three stages are specialized LLM agents that extract regulatory context, map Legata elements and attributes to Rebeca actors and state variables, and define atomic propositions and invariants, while the final stage applies formal verification with the Rebeca Model Checker. They introduce Translation Quality Criteria, a 10-point rubric assessing whether the generated property is syntactically correct, references the relevant actors and attributes of the source clause, is free of hallucinated constructs, and is expressed at an appropriate level of logical granularity. Evaluating GPT-4o first without domain context and then with added Rebeca knowledge and worked examples, on a benchmark of six COLREGs mapped to nine Legata clauses, the better-informed configuration raises the average score from 8.06 to 8.72 out of 10, references all relevant actors, and eliminates syntax errors, though expert review remains necessary.

From Manual to LLM-Assisted Formal Verification of a PKCS#1 Signature Parser [5]. Martin Hána, Nikolai Kosmatov, Virgile Prevosto, and Julien Signoles evaluate the ability of LLMs to assist engineers in the deductive verification of critical code, using a PKCS#1 v1.5 signature parser written in C that was previously specified manually in ACSL and verified with the Frama-C platform to resist Bleichenbacher-style signature forgery attacks. Their approach prompts the LLM to generate ACSL annotations directly from the C code, cross-checks the generated specifications against known security requirements, and iteratively refines them using feedback from Frama-C; for failing proof obligations, the LLM is instructed to supply sequences of ACSL assertions that guide the prover. The

experiment is repeated across several GPT versions and on buggy variants of the parser to observe whether the LLM reproduces existing bugs in the specification. Using structured, generic prompts they repeatedly obtain a complete proof, demonstrating that substantial parts of the verification workflow can be mechanized, and they compare the result with the prior manual proof in terms of how easily the specification can be reviewed and proved, and the human effort required.

When Words Change the Model: Sensitivity of LLMs for Constraint Programming Modelling [6]. Alessio Pellegrino and Jacopo Mauro investigate the robustness of LLM-based constraint modelling, questioning whether apparent success in generating constraint programming (CP) models from natural language reflects genuine reasoning about combinatorial structure or merely the reproduction of textual patterns seen during training, possibly due to data contamination. To probe this, they reformulate and perturb ten CP problems drawn from CSPLib, altering contextual elements and introducing distracting or misleading information while preserving the underlying combinatorial structure. Comparing the models generated by three representative LLMs across the original and modified descriptions, their qualitative analysis shows that, although the models are frequently syntactically valid and semantically plausible, performance degrades significantly under contextual and linguistic variation, revealing strong sensitivity to wording. Building on these observations, they articulate a set of guidelines to help practitioners craft prompts and modelling instructions that mitigate these failure modes.

AMADEUS: A Conceptual Framework for Integrating Agentic AI into Model-Based Systems Engineering [7]. Nick Wiele, Denny Marx, and Martin Wirsing present AMADEUS, a conceptual framework for integrating agentic AI into model-based systems engineering (MBSE) workflows for automated driving, exploiting the textual notation of SysML v2 that turns model construction into a text-to-text generation problem. The framework offers three contributions: a work package concept that operationalizes a systems engineering methodology into discrete, agent-executable units and matches each to a suitable pattern for coordinating the agents; two complementary architecture variants, one using a dedicated model server that exposes typed operations and one using direct SysML v2 text editing in a Git-based workflow, together with a comparative analysis; and an empirical benchmark evaluating three late-2025 frontier agents (OpenAI Codex, Google Gemini 3 Pro, Anthropic Claude Sonnet 4.5) against a human expert on seven modeling tasks. The agents achieve comparable quality (3.69 vs. 3.86 on a 0–4 scale) while reducing modeling time by roughly 12× and per-task cost by about 3,500×, suggesting the engineer's role shifts from model creator to model reviewer.

3 Conclusion

The presentations in this track covered the use of LLMs across several phases of software and systems development, including requirements formalization, sys-

tem design and modelling, and verification. This included such topics as LLM support for formalization of natural-language regulations, generation of formal specifications and properties for model checking and deductive verification, proof debugging, agentic AI workflows for model-based systems engineering, generation of system models from textual notations, constraint programming and optimization modelling, iterative repair guided by verification feedback, metrics and benchmarks for evaluating LLM output quality, and the robustness and reliability of LLM-generated artifacts. This covers an interesting spectrum of AI assisted programming in the still young era of LLMs, with a recurring emphasis on keeping the human expert in the loop as reviewer and validator. 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., Aichernig, B.K., Havelund, K.: AI assisted programming. In: Steffen, B. (ed.) AISoLA 2024 - Bridging the Gap Between AI and Reality. LNCS, vol. 15217, pp. 101–106. Springer International Publishing (2025)
2. Ahrendt, W., Aichernig, B.K., Havelund, K.: AI assisted programming (AISoLA 2025 track introduction). In: Steffen, B. (ed.) AISoLA 2025 - Bridging the Gap Between AI and Reality. LNCS, vol. 16220. Springer International Publishing (2026)
3. Ahrendt, W., Havelund, K.: AI assisted programming. In: Steffen, B. (ed.) AISoLA 2023 - Bridging the Gap Between AI and Reality. LNCS, vol. 14380, pp. 351–354. Springer International Publishing (2024)
4. Ebrahimi, M., Samuelsson, J., Smigan, C., Masud, A.N., Jongeling, R., Sirjani, M.: From maritime regulations to verifiable properties with LLM assistance. In: Margaria, T., Steffen, B. (eds.) Proc. of AISoLA 2026 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming. LNCS, Springer International Publishing (2026), [in this volume]
5. Hána, M., Kosmatov, N., Prevosto, V., Signoles, J.: From manual to LLM-assisted formal verification of a PKCS#1 signature parser. In: Margaria, T., Steffen, B. (eds.) Proc. of AISoLA 2026 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming. LNCS, Springer International Publishing (2026), [in this volume]
6. Pellegrino, A., Mauro, J.: When words change the model: Sensitivity of LLMs for constraint programming modelling. In: Margaria, T., Steffen, B. (eds.) Proc. of AISoLA 2026 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming. LNCS, Springer International Publishing (2026), [in this volume]
7. Wiele, N., Marx, D., Wirsing, M.: AMADEUS: A conceptual framework for integrating agentic AI into model-based systems engineering. In: Margaria, T., Steffen, B. (eds.) Proc. of AISoLA 2026 - Bridging the Gap Between AI and Reality. Track: AI Assisted Programming. LNCS, Springer International Publishing (2026), [in this volume]
8. ChatGPT LLMs (OpenAI). https://chat.openai.com
9. Claude LLMs (Anthropic). https://claude.ai

10. DeepSeek LLMs. https://www.deepseek.com
11. Gemini LLMs (Google). https://gemini.google.com
12. Grok LLMs (xAI). https://grok.com
13. LLaMA LLMs (Meta). https://www.llama.com
14. Claude code. https://www.anthropic.com/claude-code
15. GitHub copilot. https://github.com/features/copilot
16. Cursor IDE. https://cursor.com
17. Devin. https://devin.ai

Disclaimer