TYPE Editorial
PUBLISHED 19 August 2025
DOI 10.3389/fcomp.2025.1670939

Check for updates

OPEN ACCESS

EDITED AND REVIEWED BY
Martin Shepperd,
Brunel University London, United Kingdom

*CORRESPONDENCE
Kevin Lano
kevin.lano@kcl.ac.uk

RECEIVED 22 July 2025
ACCEPTED 01 August 2025
PUBLISHED 19 August 2025

CITATION
Lano K, Rahimi SK, Tehrani SY and Alfraihi H (2025) Editorial: Machine learning for software engineering. *Front. Comput. Sci.* 7:1670939. doi: 10.3389/fcomp.2025.1670939

COPYRIGHT
© 2025 Lano, Rahimi, Tehrani and Alfraihi. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Editorial: Machine learning for software engineering

Kevin Lano[1]*, Shekoufeh Kolahdouz Rahimi[2], Sobhan Yassipour Tehrani[3] and Hessa Alfraihi[4]

[1]Informatics Department, King's College London, London, United Kingdom, [2]Computer Science Department, Roehampton University, London, United Kingdom, [3]Computer Science Department, University College London, London, United Kingdom, [4]Computer Science Department, Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia

KEYWORDS
software engineering, artificial intelligence, machine learning, LLMs, requirements engineering

Editorial on the Research Topic
Machine learning for software engineering

1 Introduction

Artificial intelligence (AI) techniques such as machine learning (ML) and particularly deep learning (DL) tools such as large language models (LLMs) (Zhao et al., 2023) have been increasingly used to support a wide range of software engineering tasks, or even to replace traditional manually-based techniques for software development (Hou et al., 2023), and this leads to the need to consider the effectiveness and implications of such use (Sallou et al., 2024).

Historically, machine learning was divided into *symbolic* ML techniques such as inductive logic programming and *non-symbolic* techniques based on neural nets and other quantitative approaches using numeric processing. While symbolic techniques were successfully used in areas such as learning model transformations (Balogh and Varro, 2009) or code synthesis rules (Lano and Xue, 2023), they have limitations due to the need for specialized preparation of training data, and non-symbolic techniques have become predominant in practice due to their advantages in terms of scalability, flexible training requirements, and wide applicability.

In recent years, non-symbolic ML techniques have been used to perform reverse engineering of code, to generate code, and to perform many other code-based and software engineering tasks, as alternatives to rule-based techniques for these activities. The advent of LLMs from 2020 onwards as a general-purpose deep learning technique has transformed software development practice, with tools such as Copilot and Cursor being widely used to produce program code based on natural language requirements and partial specifications, such as the intended results of test cases. This approach is highly usable for practitioners, however the reliability and accuracy of LLM-based code generation is still open to question (Du et al., 2024; Jiang et al., 2024; Rabbi et al., 2025; Sadik et al., 2024; Xue and Lano, 2025), and in practice the outputs of LLMs need scrutiny by human experts before being used in operational code, as is pointed out by the paper by Uandykova et al. in this Research Topic.

In other areas of software engineering practice, ML and LLMs have been used for the reverse engineering of programs to specifications (Boronat and Mustafa, 2025; Campenello et al., 2025; Siala and Lano, 2025a,b,c; Xie et al., 2025) and for code summarization, refactoring, and program translation (Ahmad et al., 2023; Gandhi et al., 2024; Lano et al., 2025; Li et al., 2024; Malyaya et al., 2023; Zhang et al., 2022). In each of these application domains, the high usability of the conversational interfaces offered by LLM-based tools often outweighs (at least for non-critical applications) the stochastic and black-box nature of LLMs and the need to review LLM outputs.

Machine learning is particularly relevant for tasks where informal and imprecise information needs to be formalized, as occurs in requirements engineering processes such as requirements formalization. Because of the wide variation and highly expressive nature of natural language, fixed-rule systems for requirements formalization inevitably have limitations in processing natural language requirements texts, whilst ML tools such as LLMs can instead apply their broad language knowledge for the task. The paper Hemmat et al. in this Research Topic provides a survey of recent work in this field. The possible problems with LLM use in this field are identified by Borg (2024).

2 Selected papers

The six selected papers investigate different applications of AI and ML to software engineering across a range of domains.

In Uandykova et al., the authors analyze the potential for LLM use in generating industrial-quality Java code. They evaluate the ChatGPT LLM on a range of realistic tasks. They identify that the LLM can improve developer productivity and help to efficiently allocate human resources in projects. However, the study also points out limitations of the LLM for solving complex problems and the need for human intervention to verify, correct, and improve generated code.

The paper of Umar et al. defines a framework for automated requirements engineering in an agile MDE development context and uses ML models to extract essential components from textual requirements specifications, producing UML class diagrams as formalized specifications which can then be used by further development stages. They evaluate the framework on two real-world experimental studies, demonstrating accurate results when compared with manually derived class diagrams.

Also in the area of requirements engineering, the survey of Hemmat et al. carried out a systematic literature review of the field of LLM application within requirements engineering, and identified 29 primary studies in this area over the period 2020 to 2024. They conclude from analysis of these studies that LLMs have been successful in requirements specification but that deficiencies such as the presence of LLM hallucinations still remain.

The paper of Siala and Lano compares LLM-based and rule-based approaches for reverse engineering, for tasks of Python and Java reverse engineering to UML specifications. They find that the baseline performance of an LLM is prone to errors; however, fine-tuning of an LLM can improve its performance to be comparable to that of the rule-based methods.

In Guruge and Priyadarshana, the authors apply the LSTM ML technique together with Facebook Prophet to support efficient cloud computing implementation via Kubernetes. The use of ML enables proactive autoscaling, in contrast to the reactive autoscaling provided by rule-based techniques. They show that the approach improves upon existing approaches in terms of prediction accuracy.

The paper Saini et al. introduces a new technique for accurate classification of visual and textual data, with applications in data mining and natural language processing. The authors evaluate their approach on established datasets and compare the results to well-known AI classifiers such as KNN and Bayes.

3 Conclusions

This Research Topic explored the subject of AI application to software engineering, and the selected papers cover a wide range of application areas, from requirements engineering and NLP to code generation, reverse engineering, and cloud computing. We can conclude that AI techniques have shown effective capabilities for performing or supporting software engineering tasks and that there is substantial potential for extension of these AI applications to SE and for further development of the field.

Author contributions

KL: Writing – original draft, Writing – review & editing. SR: Writing – review & editing. ST: Writing – review & editing. HA: Writing – review & editing.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Gen AI was used in the creation of this manuscript.

Any alternative text (alt text) provided alongside figures in this article has been generated by Frontiers with the support of artificial intelligence and reasonable efforts have been made to ensure accuracy, including review by the authors wherever possible. If you identify any issues, please contact us.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Ahmad, W., Tushar, M., Chakraborty, S., and Chang, K.-W. (2023). AVATAR: a parallel corpus for Java-Python program translation. *arXiv preprint arXiv:2108.11590v11592*. doi: 10.18653/v1/2023.findings-acl.143

Balogh, Z., and Varro, D. (2009). Model transformation by example using inductive logic programming. *SoSyM* 8, 347–364. doi: 10.1007/s10270-008-0092-1

Borg, M. (2024). Requirements engineering and large language models: insights from a panel. *IEEE Softw*. 41, 6–10. doi: 10.1109/MS.2023.3339934

Boronat, A., and Mustafa, J. (2025). "MDRE-LLM: a tool for analysing and applying LLMs in software reverse engineering," in *SANER '25* (Montreal, QC: IEEE). doi: 10.1109/SANER64311.2025.00090

Campenello, V., Shahbaz, S., Indykov, V., and Struber, D. (2025). On the use of GPT-4 in the reverse engineering of class diagrams. *JoT* 24, 1–14. doi: 10.5381/jot.2025.24.2.a14

Du, X., Liu, M., Wang, K., Liu, J., Chen, Y., Feng, J., et al. (2024). "Evaluating large language models in class-level code generation," in *ICSE 2024* (New York, NY: IEEE/ACM). doi: 10.1145/3597503.3639219

Gandhi, S., Patwardhan, M., Khatri, J., Vig, L., and Medicherla, R. (2024). "Translation of low-resource COBOL to logically-correct and readable Java leveraging high-resource Java refinement," in *LLM4Code* (New York, NY: IEEE/ACM). doi: 10.1145/3643795.3648388

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., et al. (2023). LLMs for software engineering: a systematic literature review. *arXiv preprint arXiv*:2308.10620. doi: 10.48550/arXiv.2308.10620

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. (2024). A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*. doi: 10.48550/arXiv.2406.00515

Lano, K., Siala, H., and Jin, K. (2025). "Comparing LLM-based and model-driven program translation," in *ICoSSE* (Nice: IEEE).

Lano, K., and Xue, Q. (2023). *Code Generation by Example Using Symbolic Machine Learning*. Springer Nature Computer Science: New York. doi: 10.1007/s42979-022-01573-4

Li, X., Yuan, S., Gu, X., Chen, Y., and Shen, B. (2024). Few-shot code translation via task-adapted prompt learning. *J. Syst. Softw*. 212:112002. doi: 10.1016/j.jss.2024.112002

Malyaya, A., Zhou, K., Ray, B., and Chakraborty, S. (2023). On ML-based program translation: perils and promises. *arXiv preprint arXiv:2302.10812v101*. doi: 10.48550/arXiv.2302.10812

Rabbi, F., Ding, Z., and Yang, J. (2025). A multi-language perspective on the robustness of LLM code generation. *arXiv preprint arXiv:2504.19108v191*. doi: 10.48550/arXiv.2504.19108

Sadik, A., Brulin, S., Olhofer, M., Ceravola, A., and Joublin, F. (2024). LLM as a code generator in agile model-driven development. *arXiv preprint arXiv:2410.18489*. doi: 10.48550/arXiv.2410.18489

Sallou, J., Durieux, T., and Panichella, A. (2024). "Breaking the silence: the threats of using LLMs in software engineering," in *IEEE/ACM ICSE-NIER* (New York, NY: IEEE/ACM). doi: 10.1145/3639476.3639764

Siala, H., and Lano, K. (2025a). "Towards using LLMs in the reverse engineering of software systems to OCL," in *SANER 2025* (Montreal, QC: IEEE). doi: 10.1109/SANER64311.2025.00096

Siala, H., and Lano, K. (2025b). "Using LLMs to extract OCL specifications from Java and Python programs: an empirical study," in *OCL '25, STAF* (Aachen: XUE and LANO CEUR).

Siala, H., and Lano, K. (2025c). "Using LLMs to extract UML class diagrams from Java and Python programs: an empirical study," in *AMDE '25, STAF* (Aachen: XUE and LANO CEUR).

Xie, D., Yoo, B., Jiang, N., Kim, M., Tan, L., Zhang, X., et al. (2025). "How effective are LLMs in generating software specifications?," in *SANER 2025* (Montreal, QC: IEEE). doi: 10.1109/SANER64311.2025.00014

Xue, Q., and Lano, K. (2025). "Comparing LLM-based and MDE-based code generation for agile MDE," in *AgileMDE 2025, STAF 2025*.

Zhang, C., Wang, J., Zhou, Q., Xu, T., Tang, K., Gui, H., et al. (2022). A survey of automated code summarization. *Symmetry* 14:471. doi: 10.3390/sym14030471

Zhao, W., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223v110*. doi: 10.48550/arXiv.2303.18223