Impact of LLM-Assisted Coding in Creativity and Robustness of Robot Controllers

Paolo Baldini[1,*], Michele Braccini[1] and Andrea Roli[1,2]

[1] Department of Computer Science and Engineering (DISI), Alma Mater Studiorum — Università di Bologna, Campus of Cesena, Cesena, 47521, Italy

[2] European Centre for Living Technology, Venice, 30123, Italy

Abstract

The use of Large Language Models (LLMs) in work environments has recently started to gain attention in the research community, with many works reporting increased productivity and others reporting homogenization of the output products. Nevertheless, their use as coding assistants in robotics has been mostly overlooked, especially from the point of view of their effects compared to not-assisted programming. We claim that peculiar characteristics of robotics programming deserve special attentions, such as the robustness of the produced solution. Here we analyze the effects of using LLMs as coding assistants in robotics. We analyze their effects on the performance, in a pseudo-reality gap, and on the similarity of the produced controllers. We also briefly discuss the feedback of some participants of the experiment. The results suggest that the codes produced with the assistance of LLMs are less robust to unseen conditions, and overall more homogeneous. Additionally, we report a shorter development time when using LLMs, but a poorer coding experience.

Keywords

LLM, Robotics, Development, Creativity, Robustness, Pseudo-Reality Gap, High-Level Education

1. Introduction

The release of ChatGPT 3.5 in 2022 quickly revolutionized the world by showing how human work could be replaced or made more efficient. Among the many areas affected, software development experienced a major change. The incredible ability of Large Language Models (LLMs) to predict the code to be written and their seemingly immediate access to a huge amount of information led to massive speed-ups in code production [1]. This improvement led to the adoption of LLMs as coding assistant in many development environments.

The sudden interest in this technology captured also the curiosity of researches, who tried to assess how their specific area of knowledge could be affected by these tools. This led to the proliferation of works highlighting the positive effect of LLMs in making more effective the work of professionals. Nevertheless, also some limitation of these systems started to emerge. Specifically, it is of our interest the perceived homogenization of the output produced with the assistance of LLMs [2, 3]. Many works reported that LLMs reduce the creativity of humans, leading to the production of similar outputs. Some works argue that this can lead to a decrease in the novelty of human creations.

One field in which the impact of LLMs in development has been less considered is robotics. Specifically, to the best of the author knowledge, no work assessed their effects on the performance of the produced solution against the reality gap and their effect on the creativity of the produced solution. In this work we perform a preliminary analysis on the effects of using LLMs as coding assistants while programming robot controllers. We do so by comparing results obtained by the students of a university course in the creation of controllers for a specific task. Specifically, we explore the effects on the creativity of the produced solutions (i.e., their structural diversity), their performance, and the robustness to a

HAIC 2025 - First International Workshop on Human-AI Collaborative Systems, editors Michele Braccini, Allegra De Filippo, Michela Milano, Alessandro Saffiotti, Mauro Vallati; October 25, 2025, Bologna, Italy.

*Corresponding author.

p.baldini@unibo.it (P. Baldini); m.braccini@unibo.it (M. Braccini); andrea.roli@unibo.it (A. Roli)
0000-0002-3625-5414 (P. Baldini); 0000-0002-2335-2514 (M. Braccini); 0000-0001-9891-5441 (A. Roli)

pseudo-reality gap[1].

The article is organized as follows. In Section 2 we discuss works analyzing the use of LLMs for coding, and specifically its use in robotics. Section 3 presents our experiment and the experimental settings. Section 4 presents and explains the results obtained. In Section 5 we discuss the outcomes of the experiment and discuss how and why those should be considered when deciding to use LLMs to develop robot controllers. Finally, Section 6 summarizes the work done and proposes future works.

2. Related works

The use of LLMs in code production have recently started to increase [4, 5]. This widespread interest caught the attentions of researchers, who started to question how these tools really affect development. Some works concentrated solely on the performance of LLMs with respect to humans [6]. Nevertheless, here we are interested in their effect as coding assistants. Preliminary experiments considered small groups in context limited in time [7]. Subsequent works reported the findings in large business and companies [8, 9]. The results suggest that LLMs improve the work efficiency and quality of developers. However, the supervision of humans remains an important aspect for the effective use of these tools. One limitation of these works is that they mostly focus on the reported perception of the participants, and not on technical metrics. This could hide important flaws behind the perceived utility. He [10] analyzes the presence of vulnerabilities in produced code, with and without using LLMs. Their results suggest that the code produced with the support of LLMs presents more security issues than that produced without. This affects also the trust that humans have towards LLMs [11]. Indeed, most developers trust LLMs mostly for simple tasks such as test generation, trusting it less for what concerns code development and fix.

Multiple works considered LLMs for assisted development, but few investigated their use in robotics. We notice that the most common approach to the problem of generating code for robots consists in the generation of high-level plans (i.e., sequences of action commands) [12, 13, 14, 15]. The motivation is that most works leverage on existing (or assume the existence of) sets of basic skills. Therefore, the LLM just need to combine them, allowing the automatic generation of control plans. This approach comes to face various limitation of the LLMs themselves. For instance, the performance of LLMs tends to decrease for long texts due to difficulties in effectively using the whole context [16]. When considering the generation of complex code, this obviously becomes a problem. The generation of high-level plans mitigates this issue by reducing the length of the output required to the LLM. Nevertheless, this approach just seems to be a workaround, with complex tasks requiring long high-level plans still showing a decrease in performance [17]. Another problem that this approach solve is the generation of functioning code for specific platforms. Indeed, different robots perceive and act through devices that produce, and are controlled, by specific type of signals. LLMs do not often know how to interpret them, and therefore cannot produce the required code targeting a specific robot. Finally, combining pre-built blocks is a common approach in robotics to face the so called reality gap [18, 19]. Overall, this approach reduces the need of human developers supervising the code production. This can lead to a speed-up in code production of up to 90%[2], with a consequent reduce in costs [17].

One of the few works trying to generate robot code at a level lower than a plan is Liang et al. [20]. The authors create a system that converts human goals to an LLM-generated plan. The plan is then translated to code, iteratively implementing undefined functions. Also in this case the system assumes the existence of control primitives, but it has more control over the execution flow and the code organization. The authors report that the system struggles with commands or goals longer and more complex than those given as example, highlighting an important weakness of the system.

Recently, novel approaches started to emerge as a step forward to classical plan generation for robots. Antero et al. [17] proposes using LLMs not only as code or plan generator, but also as code evaluator. This methodology sees the contraposition of two different LLMs, one for the generation of Finite States

[1]A pseudo-reality gap simulates passing from simulation to the real robot, without actually using this latter one.
[2]This assumes that the low-level skills already exists.

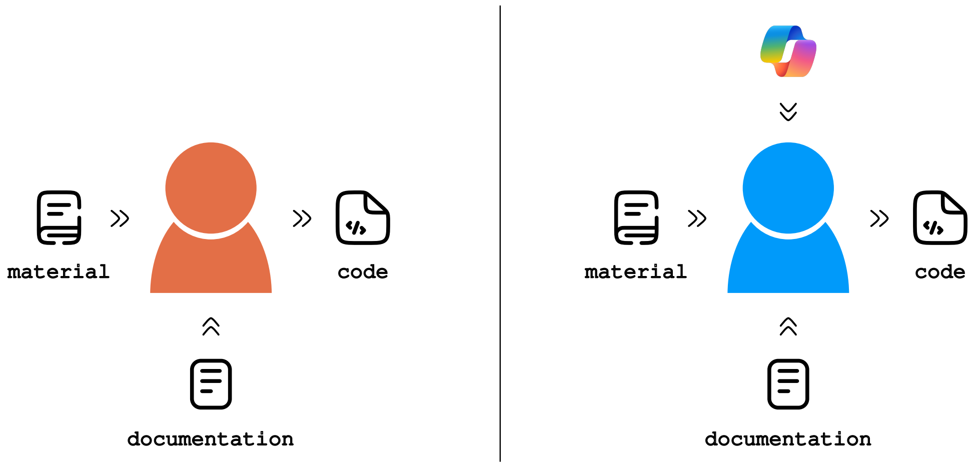


Figure 1: Representation of the artifacts used and produced during the experiment by each group: the one using Copilot (blue), and the one not using it (orange).

Machines controllers, and one for their evaluation. The controller is iteratively refined until no error is detected. This aims at removing completely the burden of code or plan generation from human developers. Differently, Schlesinger et al. [21] proposes a system that employs LLMs for the automatic generation of robot plans and the recovery from errors. The plan produced by the LLM runs until an error occurs or a human blocks the execution. The LLM then subsequently updates the code to overcome the detected problem. Also Vemprala et al. [15] proposes the real-time generation of control plans. However, rather than performing an automatic adaptation of the plan, they convert human commands to code to be immediately executed by the robot.

Most of the aforementioned works combining LLMs and robotics aims at generating high-level plans. Additionally, they see the LLMs more as a replacement for human developer rather than assistants. If the human remains part of the loop, it is mostly not expected to code, but rather to supervise. Here we argue that humans are still often the core of the robotic development, and thus we see LLMs more like assistants rather than replacements. Our approach imagines therefore a collaboration of generative systems and developers through assisted development. In this context, the code is still often produced from scratch, starting from the implementation of the low-level behaviors (that we argue requiring a great deal of effort) up to the complete control logic. Therefore, differently from other works, we examine a case of full code development, and not just planning.

3. Methods

This work aims at assessing the effect of LLMs on the development of robot controllers. As multiple solutions and approaches exist, we need to collect a set of solutions to compare. Therefore, we ask students of the University of Bologna to implement a controller for a specific task. No student reported having working experience in robotics, but all attended the course of *Intelligent Robotic Systems*, which gives them a wide perspective of development approaches and strategies in robotics. They are thus aware of the issues that could arise while programming robots, such as noise and the reality gap. However, the participants of the experiment are not aware of the analyses we perform on the controllers they create. The experiment involved thirteen male students, which we divided in two groups. Seven participants program with the assistance of LLMs, while the remaining six program without (see Figure 1). All the students are allowed to access the course material and codes of laboratories, plus the documentation of the simulation environment and programming language. Nevertheless, they are not allowed to access internet except for the aforementioned resources.

The task considered in this experiment is *path-following*. A robot deployed randomly on an arena must search for a black path, and then follow it. The robot must keep moving as fast as possible, avoiding turning on itself. This is a classical robotic task that does not constraints the use of a specific

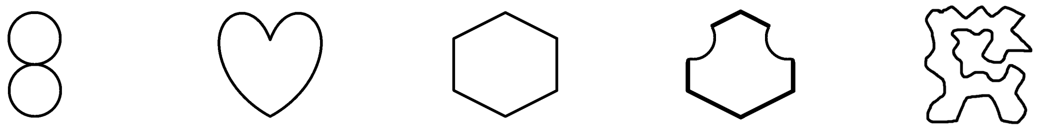


Figure 2: The paths used in the experiment. The students have had access only to arenas with the (a) *eight*, (b) *heart*, and (c) *hexagon* paths during development. The (d) *train* and (e) *freehand* paths are used in the test arenas.

control architecture. To drive the development, we present the students the evaluation function we use for assessing the performance of the controller in the experimental analysis section (see Equation 1). This computes the performance at a specific step, and thus needs to be accumulated through the entire duration of the experiment to obtain an overall evaluation. The function considers the average color of the ground, i.e., how often the robot remains on the line, and its direction and average speed. All these metrics should be maximized, as having just one of those to zero consequently zeroes the step performance. The step evaluation function is the following:

$$\frac{1}{n}\sum_{i}^{n}(1-ground(i)) \times \left(1-\frac{|M_l-M_r|}{2}\right) \times \max\left(0, \frac{M_l+M_r}{2}\right) \tag{1}$$

where:

- n is the number of ground sensors,
- $ground(i)$ is the perception of the ground color from sensor i in $[0, 1]$, where 0 indicates black and 1 indicates white,
- M_l, M_r are respectively the speed of the left and right wheels, transformed in $[-1, 1]$.

The students program a controller for the Foot-Bot robot [22, 23]. We simulate the robot and its behavior in the ARGoS3 simulator [24]. To simplify and accelerate the development, the controller is implemented in the Lua language [25], which however cannot be directly ported to the physical robots. This decision is due to the length of the experiment (i.e., three hours) which requires a fast development of the solution. Additionally, it should simplify the LLM generation, as Lua is a very permissive language similar to pseudocode.

The language model we consider in this experiment is Copilot, on its version at May 2025 [26]. Specifically, we permit its use through its online interface. This means that it has access only to the code uploaded by the students. We chose this LLM and this interaction mode for multiple reasons. First, it is a widespread tool in development. Second, it can be used remotely and without login, avoiding biases due to previous interactions or the presence of local files. Third, it can provide fast answers (i.e., Quick Response) and longer reasoned ones (i.e., Think Deeper), allowing each student to use what they prefer. Finally, it works even in incognito mode, which we require as an additional measure to avoid biases in the generated responses. We require the students belonging to the group using LLMs to query Copilot at least three times during the development.

Before performing the experiments, we require the students to take a four minutes Divergent Association Task (DAT) [27]. This aims to measure the verbal creativity and the ability to generate diverse solutions to open-ended problems [28]. We use the test to divide the students in the two groups, maintaining as balanced as possible the distribution of the DAT score (see Figure 6). The students are not aware of the test results and of the logic behind the group subdivision.

4. Results

The first metric we use to analyze the results is the performance of the produced controllers (see Figure 3). We consider the performance on the three arenas presented to the students, and on two

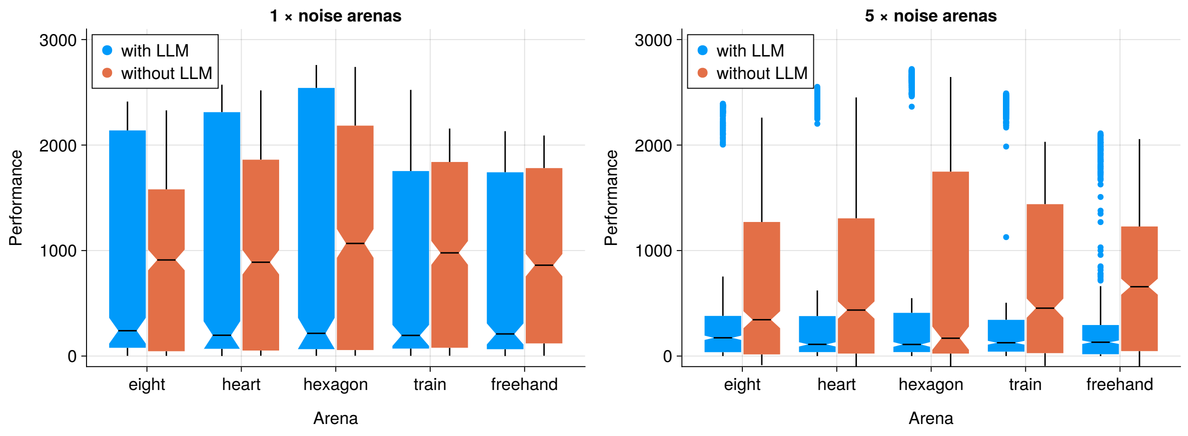


Figure 3: Performance distribution in the different arenas. The left plot shows the performance in arenas with relatively low noise. The right plot shows the performance in arenas with relatively high noise. Performances obtained with high noise simulate a reality gap between the development and production environment. The students had access to only the *eight*, *heart*, and *hexagon* arenas during the development, all with low noise. The *train* and *freehand* arenas represent a test set.

additional arenas (see Figure 2). For each controller, we perform 100 runs per arena, with a limit of 5 minutes of simulation. The group coding without the assistance of the LLM obtained averagely better results in all the arenas. However, the group using Copilot attained the highest performance in the arenas presented to the students (i.e., *eight*, *heart*, *hexagon*). This difference in the highest performance disappears when considering the two test arenas (i.e., the arenas not presented to the students: *train* and *freehand*). Even more interesting are the results obtained in the pseudo-reality gap, that in this case is, experimenting with much higher noise [29, 30, 31]. In this scenario the performances attained by the group using Copilot drop, while the performances obtained by the other group remain overall stable. This indicates a that the controllers produced with the assistance of LLM are less robust than the controllers produced by students alone.

The second metric we use to analyze the results is the code similarity. This considers the similarity between pairs of codes produced by the students of each group. For the analysis we use a software named Dolos, which aims to identify plagiarism in code [32]. This considers the similarity of codes ignoring comments, variable names and text position. Indeed, these factors concur in obscuring the results when using different metrics, such as the Normalized Compression Distance (NCD) (see Figure 7). The results show that the code produced with the assistance of Copilot is significantly more similar that the one produced by the students alone (see Figure 4). This seems to indicate a decrease in creativity and an overall homogenization of the results, as discovered by previous studies [3].

Both the performance and the code similarity seem to be affected by the use of an LLM as a coding assistant. However, it is important to avoid biases due to the creativity and expertise of the participants of the experiment. Our two groups are composed of students from the same university course, divided according to their DAT score. The aim is indeed to minimize differences between the two test groups. However, as additional check, we assess how the performance of the produced controllers correlates with the DAT score. Specifically, for each student, we take the average performance per arena over 100 runs and plot it in a scatter-plot according to the DAT score of the author (see Figure 5). This shows two interesting aspects. First, the performance seems to increase with the increase in the DAT score. Second, both groups present few students failing to produce satisfactory solutions, cancelling each other bias.

5. Discussion

This works shows the effect of LLMs in programming controllers for robots. Specifically, it highlights how using Copilot leads to less robust and more specialized code, which performs worse when some

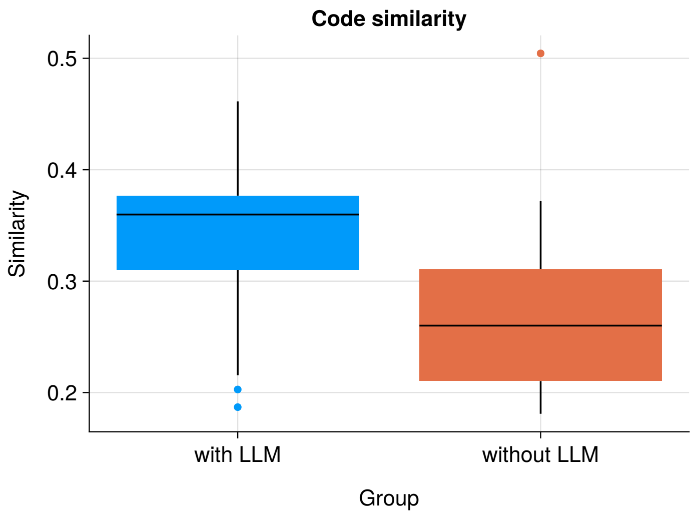


Figure 4: Pair-wise similarity of codes inside each group as calculated by Dolos.

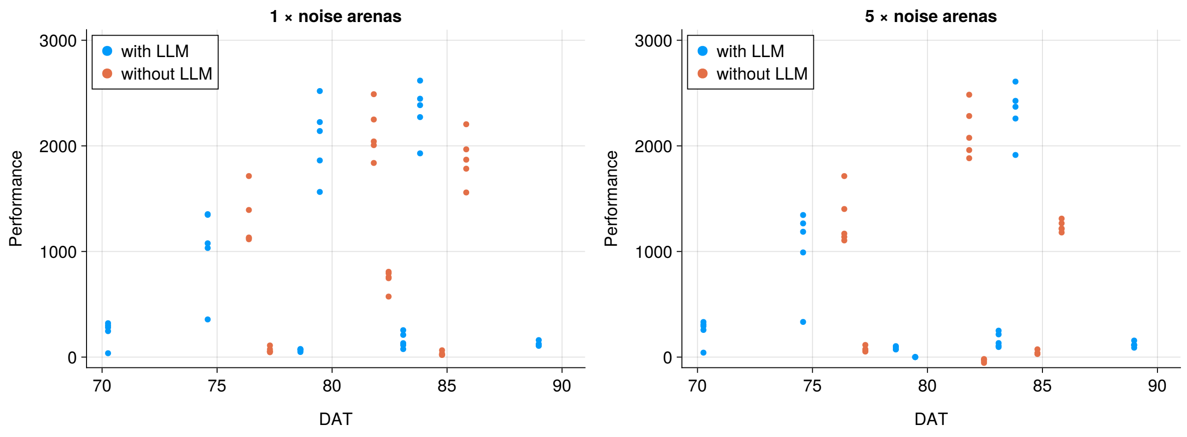


Figure 5: Average performance according to the DAT score in the five arenas, with the two levels of noise. If we ignore students which did not succeed in producing an effective controller (i.e., with performances near to 0), the performance seems to increase according to presumed creativity.

conditions change. This result is significant, as a well known issue in robotics is maintaining the performance while porting the code to different environments or platforms. Specifically, when this happens from simulation to real-world, it usually takes the name of reality gap. One of the major challenges has often been creating control systems robust to conditions different from those of development or training. To this goal, researchers proposed many strategies and architectures [29, 33, 34, 35, 36, 37, 38, 39, 40]. Nevertheless, the poor results obtained while using LLMs seem to indicate that those are not inherently able to exploit them. This could be due to the reduced availability of state-of-the-art examples present online. Indeed, most of the code available online is from examples of simulators, while the use of these architectures and strategies is often limited to minimal illustrations in papers.

Another difference detected is that using Copilot seems to produce more similar code. This could be again due to the problem presented in the previous paragraph, i.e. that the LLM does not have a broad enough knowledge of robotic systems. Alternatively, it could be related to the generally reduced variability in LLMs responses when compared to humans [41]. On its own, this is not necessarily a problem. However, the limited variability in answers can lead to fewer options presented to developers aiming to produce effective controllers.

Despite limitations, we highlight also that the use of Copilot allowed obtaining working controllers faster that by humans alone (see Figure 9). Nevertheless, this was not true in all the cases. Some students reported that asking Copilot directly for a solution led to a long process of fixes that instead elongated the development time. This led a discrepancy in the students' feedback on the quality of coding with LLM, with some students that enjoyed the process and others that did not (see Figure 8). At the end of the experiment, students reported that the best use they found for Copilot was to ask for a pseudocode or to brainstorming, but not for the code generation itself (see Table 1).

6. Conclusion

The effect of Large Language Models as coding assistants to program entire robot controllers has been until now overlooked. In this work we shed a light on the impact they have on the technical and personal perspectives of a group of developers. Specifically, we find that LLMs affect the performance of the produced controllers, making them less robust to environmental changes and thus to the reality gap. Additionally, they tend to constrain the creativity of the developers, leading to the employment of very similar strategies. Finally, although speeding up the development, the participants of the experiment reported frustration and overall less enjoyment during coding. We believe these results to be important for the creation of healthy and effective workplaces in a changing robot industry. Additionally, we notice that, in order to be effective tackling cutting-edge research problems and development in robotics, technological advancements of LLMs are still needed.

One limitation of the current work is the small number of human participants. We plan to perform additional experiments with a larger group of developers, so to get more statistically robust results. The next work could also consider multiple tasks and the comparison with a third group of developers performing pair-programming.

Acknowledgments

The authors thank the students of the *Intelligent Robotic Systems* course of the University of Bologna who participated in this experiment. Specifically, we thank Emanuele Artegiani, Samuele De Tuglie, Marco Fontana, Lorenzo Guerrini, Pablo Sebastian Vargas Grateron, and all the other participants who preferred to remain anonymous.

Declaration on generative AI

During the preparation of this work, the author(s) used Grammarly in order to: grammar and spelling check, paraphrase, and reword. After using this tool, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] C. Ziftci, S. Nikolov, A. Sjövall, B. Kim, D. Codecasa, M. Kim, Migrating code at scale with LLMs at Google, 2025. arXiv:2504.09691.

[2] A. Doshi, O. Hauser, Generative AI enhances individual creativity but reduces the collective diversity of novel content, Science Advances 10 (2024) eadn5290.

[3] B. Anderson, J. Shah, M. Kreminski, Homogenization effects of Large Language Models on human creative ideation, in: C&C'24: Proceedings of the 16th Conference on Creativity & Cognition, Association for Computing Machinery, 2024, pp. 413–425.

[4] U. Arora, A. Garg, A. Gupta, S. Jain, R. Mehta, R. Oberoi, Prachi, A. Raina, M. Saini, S. Sharma, J. Singh, S. Tyagi, D. Kumar, Analyzing LLM usage in an advanced computing class in India, in: ACE '25: Proceedings of the 27th Australasian Computing Education Conference, Association for Computing Machinery, 2025, pp. 154–163.

[5] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, B. Myers, Using an LLM to help with code understanding, in: ICSE '24: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Association for Computing Machinery, 2024, pp. 97, 1–13.

[6] S. Licorish, A. Bajpai, C. Arora, F. Wang, K. Tantithamthavorn, Comparing human and LLM generated code: The jury is still out!, 2025. arXiv:2501.16857.

[7] M. Hamza, D. Siemon, M. Akbar, T. Rahman, Human AI collaboration in software engineering: Lessons learned from a hands-on workshop, 2023. arXiv:2312.10620.

[8] Y. Gao, Research: Quantifying GitHub Copilot's impact in the enterprise with Accenture, https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-in-the-enterprise-with-accenture, 2024. Accessed on May 2025.

[9] G. Bakal, A. Dasdan, Y. Katz, M. Kaufman, G. Levin, Experience with GitHub Copilot for developer productivity at Zoominfo, 2025. arXiv:2501.13282v1.

[10] X. He, Large Language Models for code writing: Security assessment, https://medium.com/@researchgraph/large-language-models-for-code-writing-security-assessment-f305f9f01ce9, 2024. Accessed on May 2025.

[11] D. Khati, Y. Liu, D. Palacio, Y. Zhang, Mapping the trust terrain: LLMs in software engineering - insights and perspectives, 2025. arXiv:2503.13793v1.

[12] R. Wiemann, N. Terei, A. Raatz, Large Language Model for assisted robot programming in micro-assembly, in: Procedia CIRP: 57th CIRP Conference on Manufacturing Systems 2024 (CMS 2024), volume 130, Elsevier, 2024, pp. 244–249.

[13] H. Luo, J. Wu, J. Liu, M. Antwi-Afari, Large language model-based code generation for the control of construction assembly robots: A hierarchical generation approach, Developments in the Built Environment 19 (2024) 100488.

[14] Z. Hu, F. Lucchetti, C. Schlesinger, Y. Saxena, A. Freeman, S. Modak, A. Guha, J. Biswas, Deploying and evaluating LLMs to program service mobile robots, IEEE Robotics and Automation Letters 9 (2024) 2853–2860.

[15] S. Vemprala, R. Bonatti, A. Bucker, A. Kapoor, ChatGPT for robotics: Design principles and model abilities, IEEE Access 12 (2024) 55682–55696.

[16] T. Li, G. Zhang, Q. Do, X. Yue, W. Chen, Long-context LLMs struggle with long in-context learning, 2024. arXiv:2404.02060.

[17] U. Antero, F. Blanco, J. Oñativia, D. Sallé, B. Sierra, Harnessing the power of Large Language Models for automated code generation and verification, Robotics 13 (2024) 137.

[18] R. Brooks, Artificial life and real robots, in: Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, The MIT Press, 1992, pp. 3–10.

[19] M. Birattari, A. Ligot, G. Francesca, AutoMoDe: A Modular Approach to the Automatic Off-Line Design and Fine-Tuning of Control Software for Robot Swarms, Springer International Publishing, 2021, pp. 73–90.

[20] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, A. Zeng, Code as policies: Language model programs for embodied control, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), Institute of Electrical and Electronics Engineers, 2023, pp. 9493–9500.

[21] C. Schlesinger, A. Guha, J. Biswas, Creating and repairing robot programs in open-world domains, 2024. arXiv:2410.18893.

[22] M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, F. Mondada, The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Institute of Electrical and Electronics Engineers, 2010, pp. 4187–4193.

[23] M. Bonani, P. Rétornaz, S. Magnenat, H. Bleuler, F. Mondada, Physical Interactions in Swarm Robotics: The Hand-Bot Case Study, Springer Berlin Heidelberg, 2013, pp. 585–595.

[24] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. Gambardella, M. Dorigo, ARGoS: a modular, parallel,

multi-engine simulator for multi-robot systems, Swarm Intelligence 6 (2012) 271–295.

[25] R. Ierusalimschy, L. de Figueiredo, W. Filho, Lua–an extensible extension language, Software: Practice and Experience 26 (1996) 635–652.

[26] Microsoft, Copilot, https://copilot.microsoft.com, 2023. Accessed 09 May 2025.

[27] J. Olson, M. Webb, S. Cropper, E. Langer, J. Nahas, D. Chmoulevitch, The divergent association task measures verbal creativity in under 4 minutes, https://www.datcreativity.com, 2019. Accessed 09 May 2025.

[28] J. Olson, J. Nahas, D. Chmoulevitch, S. Cropper, M. Webb, Naming unrelated words predicts creativity, Proceedings of the National Academy of Sciences: Psychological and Cognitive Sciences 118 (2021) e2022340118.

[29] N. Jakobi, P. Husbands, I. Harvey, Noise and the reality gap: The use of simulation in evolutionary robotics, in: Advances in Artificial Life, Springer Berlin Heidelberg, 1995, pp. 704–720.

[30] A. Ligot, M. Birattari, On mimicking the effects of the reality gap with simulation-only experiments, in: Swarm Intelligence, volume 11172, Springer International Publishing, 2018, pp. 109–122.

[31] A. Ligot, M. Birattari, Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms, Swarm Intelligence 14 (2020) 1–24.

[32] R. Maertens, C. Van Petegem, N. Strijbol, T. Baeyens, A. Jacobs, P. Dawyndt, B. Mesuere, Dolos: Language-agnostic plagiarism detection in source code, Journal of Computer Assisted Learning 38 (2022) 1046–1061.

[33] S. Koos, J. Mouret, S. Doncieux, The transferability approach: Crossing the reality gap in evolutionary robotics, IEEE Transactions on Evolutionary Computation 17 (2013) 122–145.

[34] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, M. Birattari, AutoMoDe: A novel approach to the automatic design of control software for robot swarms, Swarm Intelligence 8 (2014) 89–112.

[35] X. Peng, M. Andrychowicz, W. Zaremba, P. Abbeel, Sim-to-real transfer of robotic control with dynamics randomization, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), Institute of Electrical and Electronics Engineers, 2018, pp. 3803–3810.

[36] E. Salvato, G. Fenu, E. Medvet, F. Pellegrino, Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning, IEEE Access 9 (2021) 153171–153187.

[37] P. Baldini, M. Braccini, A. Roli, Online adaptation of robots controlled by nanowire networks: A preliminary study, in: Artificial Life and Evolutionary Computation: 16th Italian Workshop, WIVACE 2022, Gaeta, Italy, September 14–16, 2022, Revised Selected Papers, volume 1780, Springer Nature Switzerland, 2023, pp. 171–182.

[38] P. Baldini, A. Roli, M. Braccini, On the performance of online adaptation of robots controlled by nanowire networks, IEEE Access 11 (2023) 144408–144420.

[39] M. Braccini, P. Baldini, , A. Roli, An investigation of graceful degradation in boolean network robots subject to online adaptation, in: Artificial Life and Evolutionary Computation: 17th Italian Workshop, WIVACE 2023, Venice, Italy, September 6–8, 2023, Revised Selected Papers, volume 1977, Springer Nature Switzerland, 2024, pp. 202–213.

[40] P. Baldini, M. Braccini, A. Roli, Fault recovery through online adaptation of boolean network robots, Sensors 25 (2025) 5849.

[41] M. Braccini, G. Aguzzi, P. Baldini, Unraveling creativity through variability: A comparison of LLMs and humans in an educational Q&A scenario, Submitted (2025).

Appendix

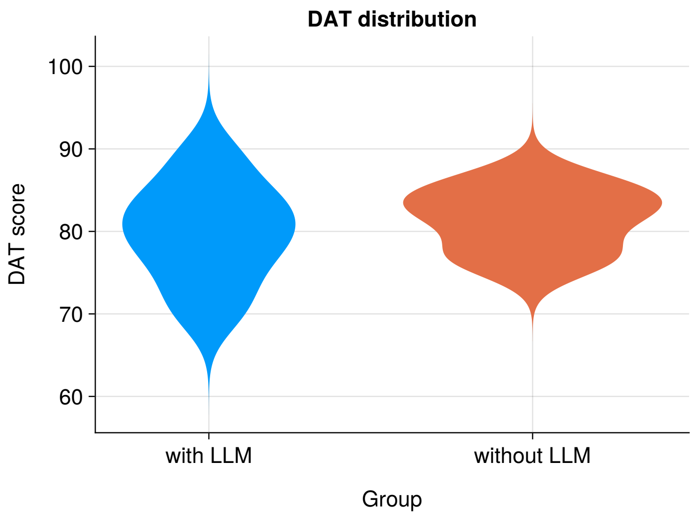
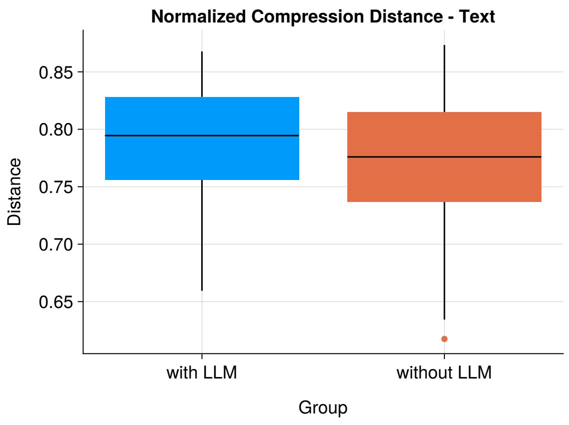
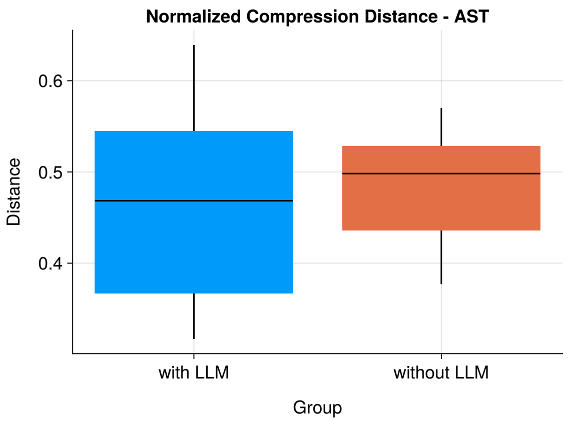


Figure 6: Distribution of the DAT scores per group.



(a)



(b)

Figure 7: Normalized Compression Distance (NCD) on pairs of students code (a) and corresponding Abstract Syntax Tree (AST) (b). We begin by taking the all combinations of two students in each group. For each combination, we compute the NCD on the codes comprehensive of comments and spaces, and on their ASTs. These two measures indicate the distance of the raw codes and the codes logic. All the codes contain a common part used for the evaluation of the controllers performance.

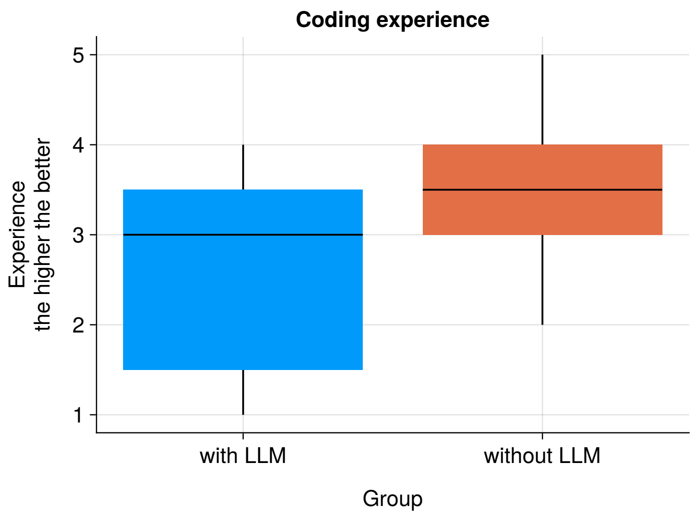


Figure 8: Coding experience as reported by the students. Each student could select a value from 0 to 5.

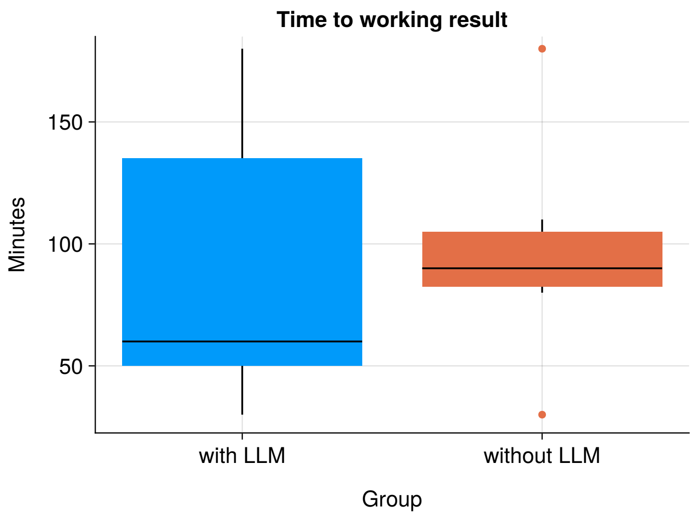


Figure 9: Time to produce a working controller as reported by the students.

	Comment
1	Students complain that code produced by Copilot needs many adjustments.
2	Students complain that asking Copilot for different solutions and/or fixes always produce similar outputs.
3	Students complain that asking Copilot for different solutions and/or fixes often breaks (or does not work with) previously generated code, thus requiring manual intervention.
4	The Quick-Answer mode of Copilot appears quite ineffective in any case, thus resulting useless.
5	The Think-Deeper mode of Copilot appears quite effective in all the cases.
6	Students using Copilot experienced overall less enjoyment during coding.
7	Students not using Copilot complain that more effort was needed to produce a working solution, and therefore they could hardly explore different strategies.
8	Students state that the code produced by Copilot was often obscure and hard to understand, limiting possible improvements.
9	Students state that Copilot was more useful during the beginning of the development. They state it is mostly useful to analyze the task characteristics, giving examples, brainstorming, producing pseudocode, producing sub-tasks, in order to start tackling the problem effectively.
10	Some students state that Copilot is useful to improve code produced autonomously; others state that its suggestions are quite ineffective.

Table 1
Summary of students' comments at the end of the experiment.