Integrating Vibe Coding and Programming Foundations in Contemporary Programming Education

Edward Abel (abel@sdu.dk)
Zhiru Sun (zhiru@sdu.dk)

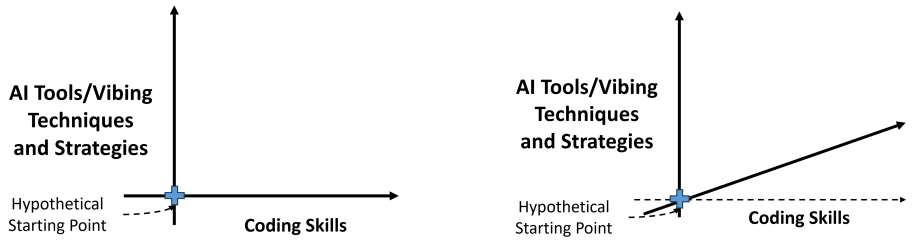
University of Southern Denmark

Abstract. AI-assisted programming tools are increasingly embedded in contemporary programming practice, reshaping both how programming work is performed and how programming competence is understood. Vibe coding, in which programmers rely on AI tools to translate high-level intent into executable code, shifts the skill set required for programming and raises important questions for programming education and the role of foundational knowledge. We conceptualise contemporary programming competence as comprising two interdependent and necessary dimensions: classical programming skills and AI-related skills. We propose and examine a course structure that integrates these dimensions within a single course, reporting empirical analysis and observations from its implementation. In this structure, students first encounter vibe coding, then engage in explicit programming instruction, before returning to AI-assisted development. We found the early exposure to vibe coding provides concrete experience of both its affordances and limitations, often surfacing frustration that motivates subsequent engagement with programming foundations. Learning programming after this exposure situates foundational skills within a broader development ecosystem, clarifying their continued relevance, while returning to vibe coding with programming knowledge enables more critical and informed use of AI tools. We argue that this structure offers a promising pathway for developing contemporary programming competence.

Keywords: Programming education · Vibe coding · Contemporary programming competence · Student learning · Course-based study

1 Introduction

AI-based programming tools are increasingly embedded in contemporary practice, reshaping how programming work is performed and how programming competence is understood [14]. In this context, vibe coding—coined by Andrej Karpathy [8]—describes a mode of programming in which developers rely on AI to help translate intent into code. The widespread adoption of AI-assisted tooling [1] reflects an expanded competence profile, including prompting, orchestration, and critical evaluation of AI-generated code. For learners, this visibility



(a) Competence dimensions

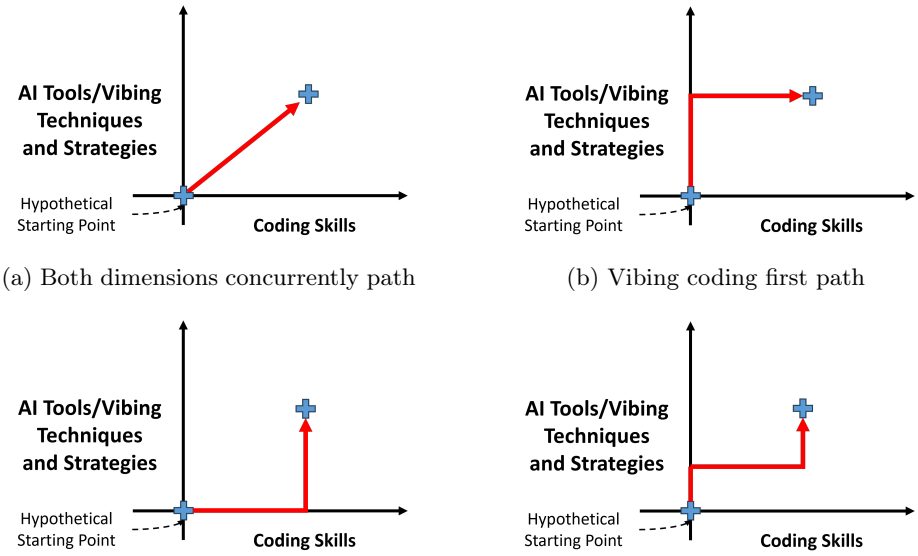
(b) Interrelation of competence dimensions

Fig. 1: Contemporary programming competence dimensions

can blur tool use and conceptual understanding, fostering beliefs that AI can substitute for foundational programming knowledge [16]; when programming is seen as *"handled by the AI"*, competencies such as abstraction, code comprehension, debugging, and reasoning about program behaviour may be undervalued. Understanding how vibe coding reshapes perceptions of programming skill and learning trajectories is therefore a key challenge for programming education.

To structure these shifts, we conceptualise contemporary programming competence as two intersecting dimensions: classical programming skills, pre-LLM competencies, and AI-related skills, prompting, steering, and evaluating AI-mediated code.These dimensions are illustrated in Figure 1a. Proficiency across both is necessary: excluding AI tools risks misalignment with contemporary practice, while emphasising AI-assisted development without foundations can obscure system behaviour and weaken evaluative capacity. Viewing the dimensions as orthogonal axes clarifies that vibe coding reconfigures, rather than replaces, how competencies combine. Although the dimensions overlap—programming expertise can reinforce vibe coding capability [11], [9] (illustrated via the skewed axis in Figure 1b)—the distinction remains analytically useful. Within this two-dimensional design space, multiple pedagogical routes can support progression across both dimensions, as illustrated in Figure 2.

In this paper, we propose and empirically examine an introductory programming course structure that incorporates both dimensions. Students first encounter vibe coding, then engage in explicit programming instruction, before returning to AI-assisted development, as illustrated in Figure 2d. We show that early exposure to vibe coding provides learners with direct experience of both its affordances and limitations, often surfacing frustration that motivates subsequent engagement with programming foundations. Learning programming after this exposure clarifies their relevance in AI-assisted workflows. Returning to vibe coding with programming knowledge then enables more critical and informed use of AI tools this time around.



(a) Both dimensions concurrently path

(b) Vibing coding first path

(c) Programming foundations first

(d) Our structure path

Fig. 2: Paths to contemporary programming competence

2 Background

As artificial intelligence reshapes programming practice, education is evolving to reflect changing expectations of programming competence [13]. However, reliance on vibe coding alone has been shown to create an illusion of competence, encouraging uncritical acceptance of generated code [15] and leaving learners unable to recover from errors [12]. Therefore, educational practice should resist narratives suggesting that AI eliminates the need for coding skills. This has prompted caution around over-reliance on AI-assisted coding [7] and highlights a *vibe coding paradox*: AI-assisted development appears most effective for those with existing programming expertise [9], reinforcing the continued importance of foundational programming skills [12]. At the same time, excluding AI-assisted practices from instruction risks leaving students underprepared for hybrid workplaces [4], where AI tooling is now widely used [1], and may lead students to adopt AI tools anyway, just in less structured and pedagogically unsupported ways [3].

Contemporary programming education should aim to develop proficiency across both dimensions, consistent with centaur models of human–machine collaboration [2]. One approach is to introduce both dimensions simultaneously within a course, as illustrated in Figure 2a. While this reflects contemporary development environments, it may confuse novice learners: without clear separation between human- and AI-generated code, students can struggle to assess their own understanding, weakening their ability to explain and reason about

system behaviour, reproducing limitations observed when relying solely on AI-assisted coding [12].

Alternatively, the two dimensions can be explored sequentially. Introducing vibe coding first, followed by formal programming instruction (Figure 2b), may motivate learners and quickly expose them to realistic tasks, but risks framing programming as slower, separate, and secondary, weakening conceptual cohesion between programming and AI-assisted development. Conversely, teaching programming foundations before introducing AI tools (Figure 2c) supports strong mental models, explicit reasoning, and learner agency, but delaying exposure to vibe coding may misalign curricula with contemporary practice and position AI assistance as optional rather than integral. This approach also risks investment in skills increasingly automated or reduced in relevance by AI [5], [11].

3 Methodology

3.1 Course Design

In contrast to the paths shown in Figure 2a–c, we explore the path shown in Figure 2d, where learners first encounter vibe coding, then engage in explicit programming instruction, before returning to AI-assisted development. Our course was organised as illustrated in Figure 3. An initial survey was administered at course commencement to capture students' backgrounds and motivations. In the course, students were first introduced to and explored vibe coding ❶ before any formal introduction to programming concepts or syntax. Students completed two aligned data preparation and visualisation tasks: one combining multiple small datasets to produce a bar plot, and another extracting subsets from a much larger dataset to generate a comparative line plot. While targeting similar outcomes, the tasks differed in scale and emphasised operations. Following this

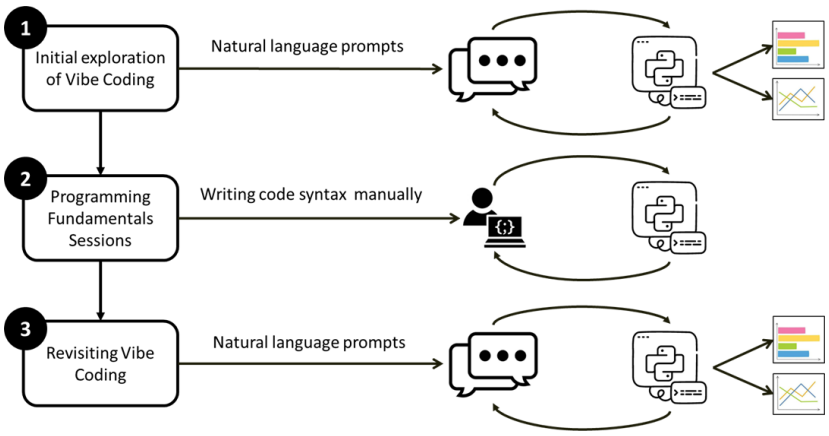


Fig. 3: Course experiment structure path

initial activity, students undertook programming sessions ❷ covering data types structures and processing, and debugging. Students then returned to vibe coding ❸ culminating in aligned tasks designed to mirror the earlier activities without repetition. These tasks shared the same learning goals but differed in datasets and implementation details.

The study was conducted in a graduate-level data science tools course for humanities students ($N = 87$). The cohort was predominantly female (72%), with a mean age of 29 years and diverse humanities backgrounds. Nearly half of students (47%) reported no prior programming experience, and only 17% had previously used AI tools for code generation. Data were collected during vibe coding ❶ and ❸, with 49 participants in the first session and 17 in the second.

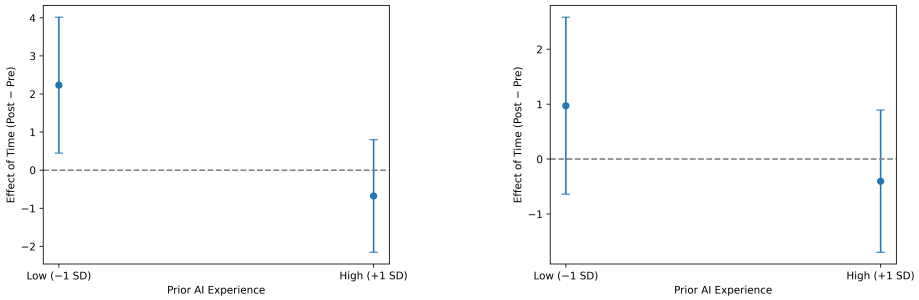
3.2 Measures and Analytical Approach

Prior AI Experience: Before initial vibe coding, familiarity with LLM tools was rated on a 5-point scale from 1 (*no prior use*) to 5 (*advanced experience*).
Perceived Task Experience: After each vibe coding session, students rated their experience on a 5-point scale from 1 (*very challenging*) to 5 (*very smooth*).
Task Accomplishment: Students' rated task accomplishment after each session on a 5-point scale from 1 (*very low*) to 5 (*very high*).
Affective Status: Students reported emotional responses to the vibe coding tasks, including *rewarding, frustrated, engaged, confused, and inspired*.
To examine whether Python instruction enhanced students' perceived task experience, we employed linear mixed-effects models (LMM) to accommodate repeated measures and unbalanced sample sizes across two time points. Separate models were estimated for Task 1 and Task 2. Each model included *time* (pre- vs. post- Python instruction), *prior AI experience* , and their *interaction* as fixed effects. To account for individual baseline differences and the non-independence of repeated observations, we included *random intercepts for each student*. Significant interactions were followed by slope analyses estimating conditional time effects at low (–1 SD) and high (+1 SD) levels of prior AI experience. All affective data (emotions) were descriptively analyzed to provide a more qualitative context for the quantitative changes in the student experience.

4 Results

4.1 Impact of Python Instruction on Vibe Coding Experience

For Task 1, results showed a significant main effect of time (β= 6.25, p = .025), indicating increased vibe coding experience following Python instruction. Prior AI experience was also positively associated with perceived experience (β= 6.59, p = .015), alongside a significant interaction effect (β= -1.83, p = .040). Simple slope analysis (Figure 4a) revealed significant pre–post improvement for students with low prior AI experience (β= 2.23, p = .014), but no significant change for those with high prior experience (β= -0.68, p = .368). This indicates that



(a) Time Effect of Vibe Coding Task 1 (b) Time Effect of Vibe Coding Task 2

Fig. 4: Comparison of vibe coding experiences of Task 1 and Task 2

Python instruction disproportionately benefited students with lower initial AI experience. For Task 2, neither the main effects of time and prior AI experience nor their interaction were significant. Simple slope analyses likewise showed no significant time effects for either experience group (Figure 4b), indicating no measurable gains following Python instruction.

4.2 Impact of Python Instruction on Vibe Coding Proficiency

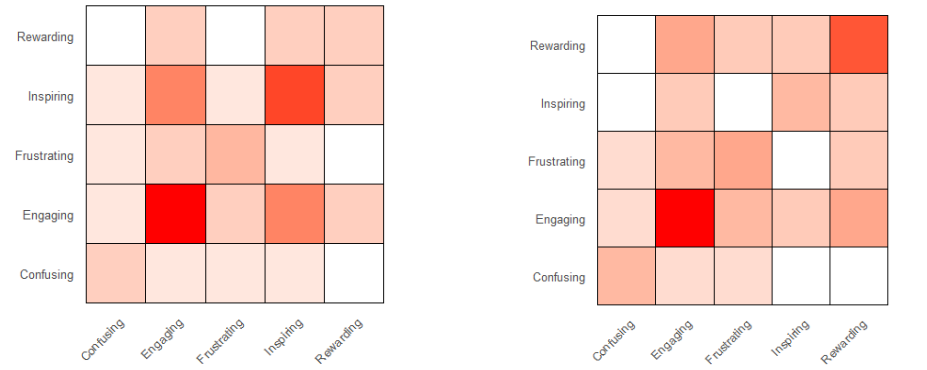
Reported task accomplishment in vibe coding was measured after ❶ and after ❸. Overall performance improved, for Task 1, mean scores increased from 3.27 (❶) to 4.00 (❸), while gains for Task 2 were more modest, rising from 2.82 to 3.27. This suggests that programming skills transferred more strongly to Task 1 than Task 2, highlighting the role of task structure in shaping subsequent vibe coding performance.

4.3 Impact of Python Instruction on Affective Status

Figure 5 shows changes in pairwise affective co-selections during vibe coding sessions ❶ and ❸. In Figure 5a, strong co-occurrence between *engaging* and *inspiring* indicates high motivational involvement during initial exposure, alongside notable levels of *frustrating*. By Figure 5b, *rewarding* is more strongly coupled with *inspiring* and *engaging*, suggesting a shift toward more achievement-oriented emotional experiences. Across both sessions, *frustrating* and *confusing* remain present. Together, these patterns suggest that Python instruction helps transform initial engagement into more rewarding learning experiences while preserving productive challenge.

5 Discussions

Our findings indicate a promising pathway towards contemporary programming competence. We found an effect of Python instruction on students' vibe coding



(a) Emotions in VibeCoding Session ❶ (b) Emotions in VibeCoding Session ❸

Fig. 5: Emotions comparisons

experiences. Vibe coding performance improved for Task 1, which closely aligned with the instructional content, particularly among students with limited prior AI experience, indicating that foundational programming skills provide conceptual scaffolding for productive engagement with vibe coding. In contrast, no significant improvement was observed for Task 2, which required greater data manipulation—competencies less directly emphasised during instruction—underscoring the importance of aligning instruction with the skills needed to support subsequent vibe coding tasks. The moderating role of prior AI experience further underscores the differentiated benefits of instruction. Students with low prior AI experience showed significant gains, suggesting the intervention helped mitigate the *"initial friction"* of programming, consistent with findings by Prather et al. [10] that AI tools can help novices bypass the *"broken code"* phase associated with frustration and confusion. As one student reflected: *"The second time was way more enjoyable than the first time, as I could understand the code that Copilot gave me. For example, learning about data types made it easier to also make assumptions on why something is not working."* By contrast, the lack of significant change among students with high prior AI experience suggests potential ceiling effects in AI-assisted learning. As Denny [6] notes, proficient users may rely on prompt-based strategies without deeply engaging with underlying logic. As one student noted: *"Even without inspecting the code in detail, it was possible to get to the right results by looking at the output of the code and make corrections based on that"*. For these students, the pedagogical challenge shifts from *enabling* access to *deepening* conceptual understanding.

6 Limitations and Future Work

As an exploratory pilot, this study is limited by a small sample size, and some use of self-reported measures, reducing statistical power and generalisability. Future

work will evaluate the proposed course structure with larger cohorts, incorporate performance-based assessments, and analyse AI chat logs. Examining iterative prompting sequences will help identify problem-solving patterns and provide a more robust account of learning in AI-augmented programming environments.

References

1. DORA Report 2025 - State of AI-assisted Software Development. Tech. rep., DORA, Google Cloud (2025)
2. Alves, P., Cipriano, B.P.: The centaur programmer – How Kasparov's Advanced Chess spans over to the software development of the future (4 2023), https://arxiv.org/pdf/2304.11172
3. Aniefuna, M.: Prohibition Didn't Stop Alcohol Use. Will It Work With AI? (2025), https://www.edsurge.com/news/2025-10-23-prohibition-didn-t-stop-alcohol-use-will-it-work-with-ai
4. Brogle, K., Bhatt, U.: Beyond Bans: Thoughtful Use of AI in the Classroom. Tech. rep., The Alan Turing Institute (2025)
5. Cotroneo, D., Improta, C., Liguori, P.: Human-Written vs. AI-Generated Code. Proceedings - International Symposium on Software Reliability Engineering, ISSRE pp. 252–263 (8 2025). https://doi.org/10.1109/ISSRE66568.2025.00035
6. Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B.A., Reeves, B.N.: Prompt Problems: A New Programming Exercise for the Generative AI Era. In: SIGCSE 2024. vol. 1, pp. 296–302 (3 2024). https://doi.org/10.1145/3626252.3630909
7. Gupta, M.: Don't be a Vibe Coder (2025), https://medium.com/data-science-in-your-pocket/dont-be-a-vibe-coder-30fa7c525971
8. Karpathy, A.k.: There's a new kind of coding I call "vibe coding". https://x.com/karpathy/status/1886192184808149383?lang=en (2025)
9. Pimenova, V., Fakhoury, S., Bird, C., Storey, M.A., Endres, M.: Good Vibrations? A Qualitative Study of Co-Creation, Communication, Flow, and Trust in Vibe Coding **1** (9 2025), https://arxiv.org/pdf/2509.12491
10. Prather, J., Reeves, B.N., Denny, P., Becker, B.A., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J., Santos, E.A.: "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers (4 2023). https://doi.org/10.1145/3617367, http://arxiv.org/abs/2304.02491
11. Saini, K.: How AI Vibe Coding Is Destroying Junior Developers' Careers (2025), https://www.finalroundai.com/blog/ai-vibe-coding-destroying-junior-developers-careers
12. Sajor, P.: A new worst coder has entered the chat: vibe coding without code knowledge (2025), https://stackoverflow.blog/2025/08/07/a-new-worst-coder-has-entered-the-chat-vibe-coding-without-code-knowledge
13. Shein, E.: The Impact of AI on Computer Science Education. Communications of the ACM **67**(9), 13–15 (9 2024). https://doi.org/10.1145/3673428
14. Storer, K.M.: AI research insights - How gen AI affects the value of development work (2025), https://dora.dev/ai/research-insights/value-of-development-work/
15. Willison, S.: Will the future of software development run on vibes? (2025), https://simonwillison.net/2025/Mar/6/vibe-coding/

16. Yilmaz, R., Karaoglan Yilmaz, F.G.: Augmented intelligence in programming learning. Computers in Human Behavior: Artificial Humans **1**(2), 100005 (8 2023). https://doi.org/10.1016/J.CHBAH.2023.100005