EVALUATING AI CODING ASSISTANTS IN AUDIO DSP EDUCATION: A SMALL SCALE STUDY

Leonardo Gabrielli and Michael Fioretti and Giuseppe Bergamino

Dept. of Information Engineering
Università Politecnica delle Marche
Ancona, IT
<l.gabrielli>@staff.univpm.it
<m.fioretti,g.bergamino>@pm.univpm.it

ABSTRACT

Recent advances in AI-assisted coding tools raise questions about how programming-intensive subjects such as audio digital signal processing should be taught and how exam projects should be evaluated. This paper presents a small-scale controlled exploratory study conducted in a graduate course on music DSP. As a final project at the end of the course, the students implemented a modular synthesizer plugin in C++. Half of the students had access to AI-assisted coding support, while the other group developed the plugin manually. All students had to follow a protocol and provide data at the end of the project, together with their code, which was discussed with them as part of the course exam. Although the scale of the study is small, the paper shares a qualitative analysis of the results and a few takeaway messages for future reference among lecturers in the field. Overall, AI assisted coding does provide some advantage to students but only in certain regards. The used AI tools, trained on GitHub repositories, seem to have only partial awareness of the state of the art in digital audio processing (e.g. antialiasing oscillators, virtual analog filters, etc.). Finally, the use of AI seems to not interfere excessively with the ability of the students to learn from their practical experience. These results, however, should be considered preliminary and cannot be extended to other cases (different subjects, student age, kind of assignment, etc.).

1. INTRODUCTION

Large language models (LLMs) have rapidly transformed software development workflows, with AI-assisted programming tools demonstrating significant acceleration of development tasks across several domains. While productivity gains are well-documented, the implications for engineering education remain uncertain - particularly in audio digital signal processing (DSP), where programming assignments traditionally serve as a primary vehicle for developing conceptual understanding. Specialized environments such as VCV Rack and general-purpose frameworks like JUCE have become popular teaching tools precisely because they allow DSP concepts to be implemented and evaluated interactively.

The availability of generative coding assistants raises an important pedagogical question: if complex DSP modules can be generated through AI-assisted programming, what aspects of audio DSP assignments still serve as reliable indicators of student understanding? Existing literature has examined generative AI in general software engineering workflows [1, 2], and educational settings [3] but relatively little work addresses specialized domains such as audio DSP. It remains unclear how AI assistance affects the correctness of domain-specific implementation choices and the interpretability of student work as evidence of understanding. When students can produce plausible, functional systems with AI support, the distinction between superficial task completion and domain-specific correctness becomes increasingly consequential, raising questions not only about how students work, but also about how their work should be evaluated.

This paper presents an empirical study conducted within a university master-level course on audio DSP. Eight students were assigned the task of implementing a moderately complex synthesizer module for VCV Rack, comprising oscillator waveform selection, noise input, modulation routing, an ADSR envelope, and a resonant filter stage. Two groups were compared: one using conventional resources (documentation, forums, textbooks), the other one was additionally instructed to use GitHub Copilot. Submissions were evaluated across functional and formal correctness, design choices, interface usability, and the emergence of features beyond the explicit specification. Rather than focusing on productivity or task completion, the study examines implementation-level evidence, such as specification adherence, parameter mapping, DSP design choices, and code organization, to assess how AI-assisted coding influences technical structure, error profiles, and domain-specific correctness. The goal is to identify which aspects of audio DSP assignments remain informative as indicators of student understanding in the presence of AI-assisted code generation.

The paper is structured as follows: Section 2 frames the work within the broader literature on generative AI and audio DSP education; Section 3 describes the study methodology; Section 4 presents results based on a custom evaluation rubric; Section 5 offers interpretations and general observations; Section 6 discusses key findings in relation to the limitation of the present study and possible future directions.

2. RELATED WORK

2.1. AI Coding Assistants

Transformer-based models trained on large-scale code corpora have demonstrated a remarkable capacity to synthesize functional programs from natural language prompts. Foundational work on Codex [4, 5] established that neural language models could solve sub-

stantial portions of standard programming benchmarks, and subsequent systems such as CodeGen [6] and AlphaCode [7] extended this capability to complex tasks and competitive programming contests, driving rapid adoption of AI-assisted tools in real-world development workflows.

According to the 2025 Stack Overflow Developer Survey, GitHub Copilot was the second most reported out-of-the-box AI agent tool among survey respondents.[1] Observational studies reveal that developers adopt distinct interaction modes depending on task familiarity: fluid generation for routine work and more exploratory prompting when navigating unfamiliar APIs or algorithms [8]. In practice, AI-generated code is rarely accepted verbatim — users routinely combine generated outputs with manual editing and debugging to arrive at correct solutions [9]. Broader impact studies suggest meaningful productivity gains in automating repetitive and boilerplate tasks [10], though these benefits are tempered by documented risks: automatically generated code can introduce security vulnerabilities when adopted without careful review [11], and the computational overhead of large-scale assistants carries non-negligible environmental costs [12].

2.2. LLMs in Engineering Education

A growing amount of research work is being done in the field of education. The number of potential applications in education is very large, ranging from learning, to teaching and evaluating [13]. Even though LLMs have been widely known to the public only after the public release of ChatGPT in 2022, several very large meta-studies are already available [14, 15, 16].

A large review of 78 studies [17] finds that while GenAI tools are broadly appreciated for personalized learning, concerns persist around academic integrity, over-reliance, and concept understanding. A complementary review of 40 empirical studies [18] emphasizes that effectiveness depends strongly on instructional embedding: structured integration, targeted assessment, and explicit guidance are necessary to preserve foundational skills and mitigate risks to higher-order reasoning. More broadly, analyses of generative AI in higher education [19, 20, 21] highlight a recurring tension between the benefits of personalized support and the risk of undermining independent problem-solving. Emerging perspectives suggest that programming curricula may need to shift emphasis toward system design, code evaluation, and critical assessment of generated output [22, 23], treating AI as an integrated component of the development environment rather than an external aid, a reorientation that includes explicitly teaching prompt design, code verification, and model limitations [24].

The report [16] analyzes a corpus of 147 research papers and highlights that the most prominent topic is the one related to assessments and their integrity (84 works). It is out of the scope of this paper to provide an overview of such a large number of papers, but it is interesting to note that policies proposed by papers may vary: some explicitly treat the use of AI as misconduct, some allow students to openly disclose the use of AI, some others recommend instructors to adopt assessment formats less vulnerable to AI. This last point calls for the redesign of exams and coursework [25, 26]. Finally a small number of studies suggest the use of plagiarism detection software, with the risk - however - of reliability.

A semester-long field study [27] reports significant improvements in final course performance among AI-assisted students, yet also documents declining reliance on the assistant over time and persistent concerns around critical thinking development. However, controlled experiments in [28] seems to contradict this picture, showing that access to ChatGPT does not necessarily yield measurable learning gains in introductory programming.

[1] Stack Overflow, 2025 Developer Survey, AI section, `https://survey.stackoverflow.co/2025/ai/`, accessed June 25, 2026.

2.3. Issues in DSP education

Audio DSP education often combines theoretical signal processing with implementation-oriented programming assignments, which may contribute to course assessment. The increasing availability of AI-assisted coding tools raises questions about the extent to which programming artifacts alone remain reliable indicators of students' individual competencies. Currently available LLMs are trained on large code repositories and general web corpora [4, 5], and can incorporate additional domain-specific knowledge provided at inference time. Their coding and reasoning capabilities have been extensively documented [6, 7], and these properties directly challenge the assumption that implementation quality reflects individual student competence.

To the best of our knowledge no specific research work addresses the topic of LLMs in audio DSP education. Some works discuss methods to enhance learning and teaching in general DSP education using LLMs [29, 30, 31, 32, 33]. The work in [34] evaluates the ability of several commercial LLMs in writing code for Matlab, Octave and Python for signal processing tasks such as denoising and data visualization. In [35] and [36] experiments are conducted to understand how LLMs (Claude 3.5 Sonnet and GPT-4, respectively) are capable of solving electrical engineering questions, including circuit theory, digital systems and DSP. Results are mixed, but AI tools can take over part of the students' work, reducing the ability of the teacher to assess their work and verify their comprehension of the subjects.

In Audio DSP education a successful assignment implementation demands not only general programming ability but also domain-specific knowledge of signal processing, real-time system design, and perceptually meaningful parameterization. This raises a key open question: when students are able to produce plausible and functional systems with AI assistance, which aspects of their work remain informative as indicators of domain-specific understanding? In particular, it remains unclear which error categories - specification misunderstandings, weak parameter mappings, DSP implementation issues, or software design limitations - persist across development conditions. The present study attempts to address this gap by analyzing student implementations of a subtractive synthesizer assignment under both AI-assisted and traditional conditions, examining artifact structure and characteristics rather than task completion alone.

3. EXPERIMENTAL DESIGN

The present study is a small-scale comparative classroom experiment in higher education, methodologically close to recent university-level studies that compares student programming performance with and without generative AI access. Controlled experiments such as [28] and [37] provide direct precedent: both employ carefully structured small-sample designs to examine not only performance outcomes but also how AI assistance reshapes student development practices, supporting the validity of this approach when artifact quality and process differences are the primary object of inquiry.

Our study adapts this family of designs to a domain-specific context. Submitted synthesizer modules are evaluated through a purpose-built rubric (see Table 1) targeting dimensions particularly salient in audio programming: interpretation of underspecified requirements, musically meaningful parameter ranges, DSP implementation choices, signal-level management, and code structure. The study should thus be understood as a pilot investigation into how AI-assisted coding affects not only programming productivity, but the quality of DSP objects such as filters and oscillators.

3.1. Course Context

The experiment was conducted within the graduate course *digital circuits for music processing and sound synthesis, a compulsory course in the Master's program Electronic Systems for Digital Audio Applications at Università Politecnica delle Marche (UNIVPM)*. Students typically hold a bachelor's degree in Electronic Engineering and possess foundational DSP knowledge, C coding, though many have no prior experience with C++ or audio-specific DSP implementations. The 72-hour course covers discrete-time implementations of acoustic and electronic circuits, synthesis algorithms (additive, FM, subtractive, wavetable, physical modeling), digital filters, time-frequency analysis, audio coding, MIDI, and DAW architecture, with particular attention to real-time constraints, numerical stability, and optimization on ARM and x86 platforms. Practical exercises require students to implement synthesis and audio processing algorithms in C++, developing audio plugins and synthesizer modules using modern frameworks. Course assessment includes an individual programming project constituting a substantial portion of the final grade, complemented by an oral examination in which students present and defend their implementation choices.

3.2. Assignment Design

Students were asked to implement a virtual synthesizer module within VCV Rack, guided by a schematic diagram (Figure 1) and a written functional specification. The architecture follows a classical subtractive synthesis signal flow: two VCOs generating band-limited square and sawtooth waveforms share a global pitch control with optional V/Oct modulation input, while the second oscillator additionally supports ± 1 octave detuning relative to the first; each oscillator has an independent level control. A noise generator with adjustable amplitude is summed with the oscillator outputs and routed through a second-order resonant low-pass filter with V/Oct-modulatable cutoff. The filtered signal passes through a VCA controlled by an ADSR envelope generator. Additionally, an LFO provides sinusoidal modulation of oscillator pitch and noise amplitude, with controls for rate and depth, and an exposed output port for external routing. Students were also required to design the module's graphical panel within VCV Rack, including knobs, switches, labels, and I/O ports, employing a predefined background graphic.

The specification intentionally left certain design decisions underspecified: gate signal handling for the ADSR was not included in the diagram nor the text; the LFO-to-noise modulation routing was ambiguously prescribed (the LFO out arrow points towards the LFO amount knob, however the specification text required the noise to be amplitude-modulated by the LFO). Furthermore, the diagram in Figure 1 showed triangle and square waves, while the text referred to sawtooth and square waves. These ambiguities were introduced to assess whether the students relied on both sources and how they interpreted incomplete specifications and how AI assistance may influence such interpretive design choices.

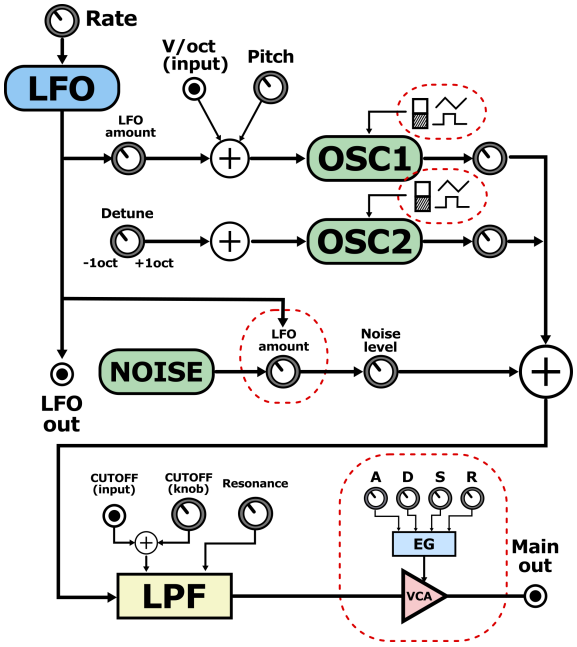

Figure 1: Synth module diagram given as additional visual instruction to participants. The dashed, red circles highlight ambiguous indications. From top to bottom: selectable square-triangular wave instead of square-saw as in the verbal instructions; unspecified LFO-noise modulation behavior; unspecified gate input.

3.3. Experimental Protocol

The small-scale study employed a between-subjects design and was conducted between January 2025 and February 2026. Eight students were assigned individually to either an AI-assisted group using GitHub Copilot in Visual Studio Code, or a control group relying on conventional resources such as official documentation, textbooks, and technical forums. We did not constrain Copilot to a specific model: students mostly used its automatic model-selection mode. Because the available model pool changed during the study period, the results should be interpreted as reflecting GitHub Copilot as a deployed educational tool rather than the behavior of a single fixed LLM.

To guarantee anonymity all participants were assigned a label, S for master students and D for PhD students. Labels S1, S2, S3, and D1 identify control group's students, while labels S4, S5, S6 and D2 identify the Copilot group. Both groups worked from the same development environment and base project template. Students were instructed not to exchange information during development to preserve independence.

Students were introduced to the Pomodoro technique [38], a time management method that concentrates work in 25-minute interval sessions followed by short breaks. This allowed the students to effectively log the elapsed time and the kind of work conducted during that session (coding, debugging, studying). The initial setup of the development environment was not counted.

Group-specific process metrics were also collected: AI-assisted students recorded the number of individual prompts sub-

mitted to Copilot (i.e., each single message sent within a chat session, aggregated across all sessions), while control group students logged the number and type of external resources consulted. At the conclusion of the development period, each student submitted their completed plugin alongside a short report describing their process and any difficulties encountered.

3.4. Evaluation Metrics

Submitted plugins were assessed qualitatively by 3 instructors through a structured rubric spanning five categories: specification comprehension; formal and interface correctness; musical parameter mapping; DSP implementation quality; code quality and organization. Table 1 summarizes the evaluation criteria employed for each category. Each instructor provided textual notes and comments for each submission according to the rubric criteria, then, to ensure consistency, a single instructor converted the qualitative evaluations into quantitative scores.

4. RESULTS

The following analysis foregrounds recurring implementation patterns across submissions. At a surface level, the majority of submissions were functional: oscillators, filter stages, and envelope-controlled amplification were generally present, and all projects produced audible output. Closer inspection, however, exposes systematic categories of errors and inconsistencies that reflect meaningful differences in how the specification was interpreted and realized.

4.1. Observed Strategies

Interpretation of implicit requirements varied across both groups. Gate signal handling for the ADSR was correctly implemented in most submissions, with the exception of S2, which substituted it with a local trigger button. More substantial divergence emerged in LFO-to-noise modulation routing: while S4, D1, and S6 correctly implemented amplitude modulation, several control-group submissions (S2, S3, S1) and two Copilot-assisted cases (S5, D2) deviated through misinterpretation of the LFO output as a multiplicative factor, an additive offset, or by omitting the modulation stage entirely. A further source of ambiguity arose from a discrepancy between the written specification, which required square and sawtooth waveforms, and the schematic diagram, showing selectable square and triangular waveforms. This resulted in mixed implementations across both groups with no particular preference.

Knob ranges: Knob ranges were adequate in all submissions (except for S2, who set the LFO rate range in the range 100 Hz-1 kHz), but in many cases pitch and cutoff frequency were linearly mapped, or constrained in nonstandard ranges (e.g. filter cutoff maximum at 8 kHz). Envelope time (e.g. attack time) was often inconsistent with the expected duration indicated on the knob. Submissions achieving musically plausible parameterizations (D1, S3, S6) were distributed across both groups, and no clear pattern of advantage or deficiency was attributable to AI assistance alone.

DSP implementation quality: Oscillator design showed the widest implementation variance, ranging from naive phase accumulation without anti-aliasing (S2, S3, D2, S5) to lookup tables (D1), DPW[39] (S1, S4), and PolyBLEP [40] (S6). Reports from

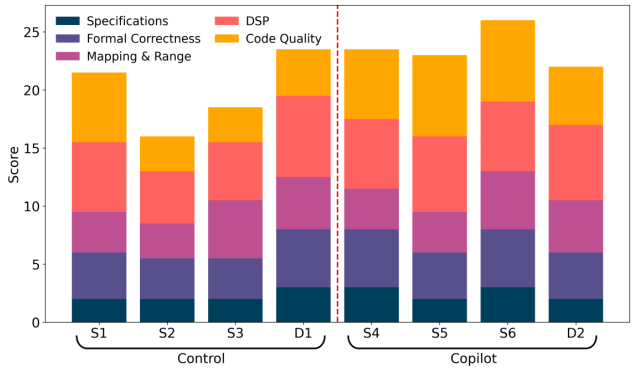

Figure 2: Score breakdown for each participant across the five evaluation categories.

the students tell that Copilot always proposed trivial oscillators when not asked explicitly for alias reduction. When S4 asked Copilot for a bandlimited oscillator, a minBLEP implementation was proposed (reportedly, no other method was known to Copilot), but at the end, the student provided lecture material about DPW which was used by Copilot to implement the oscillator correctly. In the same situation S6, instead was provided with a basic polyBLEP implementation.

Filter implementations were comparatively more homogeneous, with most participants adopting state-variable topologies (either Chamberlin [41] or Zavalishin form [42]) derived from course material (all except D2, S5). Direct-form biquad designs appeared predominantly in Copilot-assisted assignments - an implementation choice that, unlike state-variable topologies, is prone to artifacts under real-time parameter modulation (e.g. cutoff sweeps). However, two students from the Copilot group provided lecture materials asking Copilot to adopt the SVF topology instead. Signal-level management and numerical robustness were recurrent concerns across both groups, manifesting as excessive internal amplitudes (S4, S2), absent output clamping (D2, S5), initialization instabilities (S2, S1), and unintended inter-instance coupling through shared global state (S3).

Code organization: Code structure varied from modular designs with dedicated DSP classes and clear file separation (S1, S5, S6, D1) to monolithic implementations with limited abstraction. Naming consistency and overall readability followed a similar pattern: several submissions showed evidence of incremental writing without an initial design or a later systematic refactoring. Not all copilot works were well organized, but all Copilot-based works were tidy in the indentation and spacing, with a balanced presence of comments.

4.2. Score Distribution and Category-Level Comparison

Figure 2 reports per-participant total score decomposed across the five evaluation categories introduced in Section 3.4 and summarized in Table 1. Results for the control group are shown on the left, the copilot group on the right. Variability is observed in the mapping and range category, where some submissions exhibit reduced scores due to non-musical parameter scaling or inconsistent control behavior.

Table 1: Structured evaluation rubric used for the assessment of student synthesizer implementations.

Category	Evaluation criteria
Specification comprehension	• Presence of a gate signal for ADSR envelope • Correct routing of LFO modulation to noise through amount parameter • Oscillator waveform selection (tri or saw)
Formal and interface correctness	• Presence of incorrect or misleading labels • Completeness of controls (knobs, switches, inputs/outputs) • Overall correctness of system implementation • Panel readability and organization • Consistency in number and type of UI elements
Musical parameter mapping	• Pitch and detune range appropriateness • Filter cutoff range and mapping • Resonance range appropriateness • LFO frequency range and polarity • ADSR parameter ranges and scaling
DSP implementation quality	• Use of anti-aliasing techniques in oscillators • Filter topology and compliance with specification • Linear vs nonlinear filter implementation • LFO waveform generation method • Noise generator type • VCA implementation • Signal dynamic range management • Numerical stability and initialization
Code quality and organization	• Modular file structure (use of multiple headers) • Encapsulation into classes • Code readability and indentation • Consistency of naming conventions

The most pronounced between-group differentiation emerges in code quality as shown in Figure 3: AI-assisted submissions cluster toward the upper range of the scoring scale, whereas control group submissions exhibit a broader distribution, including lower scores associated with limited modularization and inconsistent naming conventions. No comparable separation is observed across the remaining categories, where score distributions overlap substantially between groups. This pattern suggests that AI assistance had a more measurable effect on code organization and structural quality than on signal processing design, specification adherence, or parameter mapping.

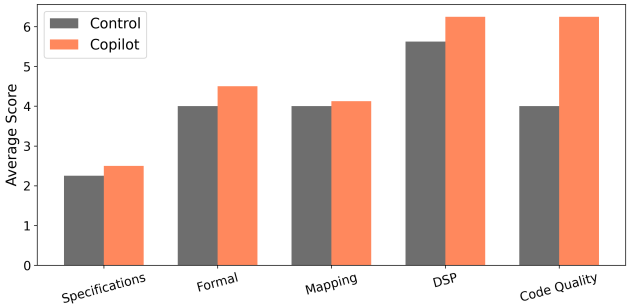

Figure 3: Comparison of code quality scores between control and AI-assisted groups.

4.3. Elapsed Time

Figure 4 summarizes the total development effort for each participant, decomposed into coding, debugging, and study time. The total time varies substantially across subjects. The impact of AI in the overall elapsed time is remarkable, however it does not shrink the coding and study effort by orders of magnitude if we consider the students S1...S6. The two PhD students, instead, though comparably skilled, exhibit a huge difference in the elapsed time. Their study time is similar and is much shorter than the other students, given their prior preparation; coding time undergoes a huge reduction for D2, thanks to AI assistance and the ability to formulate prompts correctly; debug time is also quite short thanks to the ability to drive the process with awareness (although D2 required more debug time than D1, because the AI-written code needed to be understood and tested thoroughly).

Figure 5 places each project in the score-time plane, relating total development time to the corresponding evaluation score. A tentative positive trend is visible across both groups, with greater development effort broadly associated with higher scores, though the small sample size precludes any strong interpretive claim. The doctoral-level participants' submissions deviate more noticeably from the general pattern, likely reflecting differences in prior experience and problem-solving strategies.

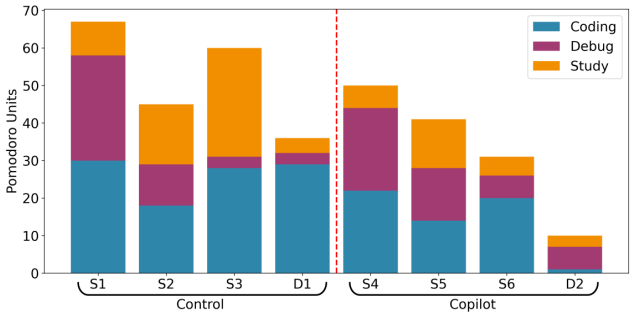

Figure 4: Development time distribution across participants, expressed in Pomodoro units and divided into coding, debugging, and study activities.

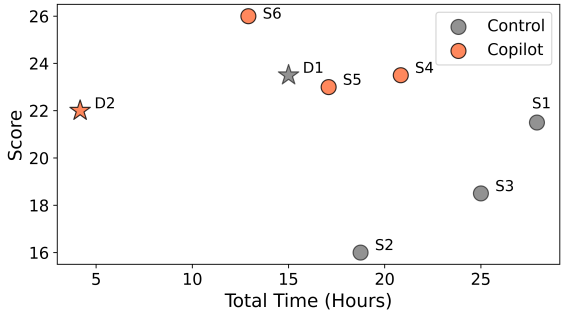

Figure 5: Relationship between total development time (hours) and overall score. Control and AI-assisted groups are distinguished by color, while phD students (D) are indicated with a different marker.

4.4. Student perceptions of Copilot usefulness

All students were asked to explain the inner functioning of their synthesizer and relate that to theoretical DSP knowledge. The level was almost identical between groups and no signs of cheating were detected.

Copilot group students were also asked to describe their experience during the assignment execution. Table 2 summarizes recurrent comments. Across the four reports, Copilot was appreciated for explaining existing code, comparing alternative implementations, retrieving project context, and accelerating technical or repetitive tasks such as parameter configuration and structural edits. Participants consistently reported that the assistant was most effective once a project architecture was already in place. One student noted that it was *"especially useful in the more advanced stages, when the project was already well structured"*, while proving less reliable for building components from scratch. This limitation was particularly evident for GUI layout and domain-specific DSP structures such as oscillators, envelope generators, and filters, as one participant put it, *"it struggles if you provide it with a theme to develop from scratch"*. Students also described Copilot as requiring active supervision: prompts frequently needed reformulation, suggestions occasionally drifted beyond the intended scope, and the inline suggestion interface was experienced as distracting and difficult to constrain. One student captured this dynamic concisely, observing that *"sometimes it takes the initiative and strays from the plan, it needs to be kept on a leash"*.

Table 2: Summary of comments from students

Category	Common observations
What Copilot helped with	Code explanation, comparison of alternatives, repetitive coding tasks, local refinements.
When it helped most	After the project architecture had already been established.
Where it struggled	GUI organization, from-scratch generation, specific DSP components.
What students still had to do	Constrain prompts, reformulate requests, critically supervise the generated output.
Overall perception	Useful as a support tool, but not as a fully reliable autonomous developer.

Student reflections suggest that Copilot functioned for them more as a guided support tool than as an autonomous development partner. It was valued primarily for local scaffolding, code interpretation, and iterative refinement, rather than for reliable first-pass generation of domain-specific DSP structures.

5. DISCUSSION

5.1. Technical outcomes

By reading the reports, discussing with the students and analyzing their code, we can draw some interesting technical information to share with the DAFx community.

- Although the multimodal AI models available with Copilot can build a synthesizer prototype from a drawing or from a text file, they do not have the ability to overcome all the details, find suitable DSP recipes to avoid aliasing and finalize the project in one single iteration;
- At the moment, Copilot has low expertise on DAFx-related state-of-the-art findings such as bandlimited methods (only polyBLEP and minBLEP are known, only the Biquad topology is known), although knowledge can be supplemented with additional material;
- Copilot is not proficient with arranging graphical items on the GUI;
- Copilot is very proficient with code styling (variable naming, indentation, spacing and comments), encapsulation and use of certain C++ features such as namespaces vary.

These results are only preliminary, and may depend on the fact that Copilot had to deal with a lesser known audio SDK, i.e. that of VCV Rack. Inconsistencies in the behavior of Copilot seen during the work may not be only due to LLMs *temperature* providing randomness, but most probably due to the use of the "Auto" feature in VS Code, i.e. the automatic selection of AI models. A thorough comparative of such models is outside the scope of this paper.

5.2. Guidelines for lecturers

Students in the control group may be pushed towards cheating by secretly using AI coding tools. However, this was not the case, as

their code is quite clearly their own: bad code formatting can be a suggestive indicator, though not a definitive one. Other disclosed design choices, such as copying existing VCV Rack designs, were also openly reported. During the oral exam all students were able to clearly explain their project.

This result can be imputed by the lack of incentives for developing fast. They knew the elapsed time would not be used to judge their work. The amount of work was compatible or lower than other final projects the students have, therefore the effort was not perceived as too heavy, and they had no particular push towards employing the least possible time.

Students also understood that it was crucial to be able to explain the development step by step. The students in the AI group, showed to be able to drive the decision process to get the desired results and accept AI decisions only if they were convinced by the outcome.

It is noteworthy to say that part of this positive result is the moral involvement of the students, who knew they were participating to a scientific study and, therefore, had to behave fairly and report real data.

By discussing with the students, we observed that the Copilot group generally showed more willingness to improve their code after a first draft, while the control group worked towards a first version and stopped there. This is explained by the different affordability of coding given by AI assistants. Therefore, the AI group developed slightly better projects. We can conclude that AI can help in getting stretched goals: this must be taken in mind when assessing a final project, since extra features can bias positively the evaluator in spite of other small errors.

Details in the specifications that need to be figured out can help understand whether the student is working on his/her own. The GATE detail, for instance, was spotted by most students, and some of them contacted the teacher to discuss alternatives. This was a sign that they were reflecting on the outcome and made them think about potential solutions. The AI, instead, would try to solve alone or not spot the problem.

To conclude, for lecturers and teachers who wish to include a final project to their DSP courses, but desire the students not to use AI tools, we can suggest the following:

1. Leave enough time to students, to avoid incentives for AI assisted tools;
2. If students are not very proficient with coding, consider looking at the style and formatting as a hint for AI usage;
3. Hide details that the students need to figure out themselves and see how they solve these, allowing them to contact the teacher;
4. Leave some arbitrary decisions to the students and let them motivate their choices.

6. CONCLUSIONS

This paper described a small-scale study on the use of AI-assisted coding within an audio DSP university-grade course, with the intent of understanding how AI coding tools can affect the development of a student project. Although preliminary, the results seem to indicate that despite the promise of a reduction in the elapsed time, there is no large difference in the quality of the outcome and that - at the moment - AI tools from Github Copilot do not master the development of a software synthesizer to the point that the student is relieved from the cognitive activity that leads to designing and learning.

AI is more effective in reducing the development time for experienced users, such as the PhD students in this study, but regular students spend more time in studying, and iterating through code development and debug, which is definitely what helps them understand DSP basics.

This study did not investigate what are the real capabilities of currently available AI models in developing DSP plugins, nor what embedded knowledge they have about state-of-the-art techniques in digital audio algorithms such as bandlimited oscillators, virtual analog filters, differentiable DSP, physical modeling, etc. All these topics are left for future works.

7. REFERENCES

[1] I. Solohubov *et al.*, "Accelerating software development with AI: Exploring the impact of ChatGPT and GitHub Copilot," in *Proc. of the 11th Workshop on Cloud Technologies in Education (CTE 2023)*, ser. CEUR Workshop Proceedings, vol. 3679, Kryvyi Rih, Ukraine, 2024, pp. 76–86.

[2] R. Pandey *et al.*, "Transforming software development: Evaluating the efficiency and challenges of GitHub Copilot in real-world projects," 2024, preprint:2406.17910.

[3] M. Alanazi *et al.*, "The influence of artificial intelligence tools on learning outcomes in computer programming: A systematic review and meta-analysis," *Computers*, vol. 14, p. 185, 2025.

[4] M. Chen *et al.*, "Evaluating large language models trained on code," 2021, preprint:2107.03374.

[5] J. Austin *et al.*, "Program synthesis with large language models," 2021, preprint:2108.07732.

[6] E. Nijkamp *et al.*, "CodeGen: An open large language model for code with multi-turn program synthesis," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2023, preprint:2203.13474.

[7] Y. Li *et al.*, "Competition-level code generation with AlphaCode," *Science*, vol. 378, pp. 1092–1097, 2022.

[8] S. Barke *et al.*, "Grounded Copilot: How programmers interact with code-generating models," *Proc. of the ACM on Programming Languages*, vol. 7, pp. 85–111, 2023.

[9] P. Vaithilingam *et al.*, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Extended Abstracts of the 2022 CHI Conf. on Human Factors in Computing Systems*, 2022, pp. 1–7.

[10] C. Bird *et al.*, "Taking flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools," *Queue*, vol. 20, pp. 35–57, 2023.

[11] H. Pearce *et al.*, "Asleep at the keyboard? assessing the security of GitHub Copilot's code contributions," in *Proc. of the 2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 754–768.

[12] T. Coignion *et al.*, "Green my LLM: Studying the key factors affecting the energy consumption of code assistants," 2024, preprint:2411.11892.

[13] S. Wang *et al.*, "Large language models for education: A survey and outlook," *IEEE Signal Processing Magazine*, vol. 42, pp. 51–63, 2025.

[14] A. Rana *et al.*, "Transformative role of large language models in education and comparative analysis with traditional technologies," *Education and Information Technologies*, 2025, early access.

[15] B. Dong *et al.*, "Large language models in education: A systematic review," in *Proc. of the 2024 6th Int. Conf. on Computer Science and Technologies in Education (CSTE)*, 2024, pp. 131–134.

[16] D. Bektik *et al.*, "Use of LLM tools within higher education: Report 2," Quality Enhancement and Innovation, Institute of Educational Technology, The Open University, Tech. Rep., 2025.

[17] F. J. Agbo *et al.*, "Computing education using generative artificial intelligence tools: A systematic literature review," *Computers and Education Open*, vol. 9, p. 100266, 2025.

[18] J. Nathaniel *et al.*, "Literature review on the integration of generative AI in programming education," *Int. Journal of Artificial Intelligence in Education*, vol. 35, 2025.

[19] E. Kasneci *et al.*, "ChatGPT for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, 2023.

[20] C. K. Lo, "What is the impact of ChatGPT on education? a rapid review of the literature," *Education Sciences*, vol. 13, p. 410, 2023.

[21] D. R. E. Cotton *et al.*, "Chatting and cheating: Ensuring academic integrity in the era of ChatGPT," *Innovations in Education and Teaching Int.*, vol. 61, pp. 228–239, 2024.

[22] D. Baidoo-Anu and L. O. Ansah, "Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning," *Journal of AI*, vol. 7, pp. 52–62, 2023.

[23] S. Filippi and B. Motyl, "Large language models (LLMs) in engineering education: A systematic review and suggestions for practical adoption," *Information*, vol. 15, p. 345, 2024.

[24] D. Kohen-Vacs *et al.*, "Integrating generative AI into programming education: Student perceptions and the challenge of correcting AI errors," *Int. Journal of Artificial Intelligence in Education*, vol. 35, pp. 3166–3184, 2025.

[25] R. Arum *et al.*, "ChatGPT early adoption in higher education: Variation in student usage, instructional support, and educational equity," *AERA Open*, vol. 11, 2025.

[26] D. Agostini and F. Picasso, "Large language models for sustainable assessment and feedback in higher education: Towards a pedagogical and technological framework," *Intelligenza Artificiale*, vol. 18, pp. 121–138, 2024.

[27] W. Lyu *et al.*, "Evaluating the effectiveness of LLMs in introductory computer science education: A semester-long field study," in *Proc. of the Eleventh ACM Conf. on Learning @ Scale*, 2024, pp. 63–74.

[28] Y. Xue *et al.*, "Does ChatGPT help with introductory programming? an experiment of students using ChatGPT in CS1," in *Proc. of the 2024 IEEE/ACM 46th Int. Conf. on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 2024, pp. 331–341.

[29] D. Bansal *et al.*, "Transforming digital signal and image processing education: An AI-driven approach to pedagogical advancements," *Sustainability*, vol. 17, p. 10741, 2025.

[30] A. Kwasinski *et al.*, "Lessons from two roundtables on artificial intelligence and signal processing education: Addressing the emergence of a new era and a new discipline," *IEEE Signal Processing Magazine*, vol. 42, pp. 39–50, 2025.

[31] J. Haupt *et al.*, "Deploying AI for signal processing education: Selected challenges and intriguing opportunities," *IEEE Signal Processing Magazine*, vol. 43, pp. 32–46, 2026.

[32] S. Nivithan *et al.*, "Exploring state-of-the-art methods in mastering digital signal processing through blended learning," in *Proc. of the 2024 Third Int. Conf. on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, 2024, pp. 1–9.

[33] Z. Xu and X. Hei, "Application of generative AI in experimental teaching of communication principles," in *Proc. of the 2025 IEEE Global Engineering Education Conf. (EDUCON)*, 2025, pp. 1–5.

[34] C. G. Druga *et al.*, "On the use of AI in signal processing programming applications at the master of science level," in *Proc. of the 2025 9th Int. Symposium on Electrical and Electronics Engineering (ISEEE)*, 2025, pp. 1–6.

[35] Y. Mormul *et al.*, "Gauging the capability of artificial intelligence chatbot tools to answer textbook coursework exercises in circuit design education," in *Proc. of the 2024 21st Int. Conf. on Information Technology Based Higher Education and Training (ITHET)*, 2024, pp. 1–7.

[36] Z. Zhong *et al.*, "Exploring the performance of generative AI tools in electrical engineering education," in *Proc. of the 2023 IEEE Int. Conf. on Teaching, Assessment and Learning for Engineering (TALE)*, 2023, pp. 1–6.

[37] M. I. H. Shihab *et al.*, "The effects of GitHub Copilot on computing students' programming effectiveness, efficiency, and processes in brownfield coding tasks," in *Proc. of the 2025 ACM Conf. on Int. Computing Education Research, Vol. 1*, 2025, pp. 407–420.

[38] F. Biwer *et al.*, "Understanding effort regulation: Comparing "Pomodoro" breaks and self-regulated breaks," *British Journal of Educational Psychology*, vol. 93, pp. 353–367, 2023.

[39] V. Välimäki *et al.*, "Alias-suppressed oscillators based on differentiated polynomial waveforms," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 18, pp. 786–798, 2010.

[40] V. Välimäki and A. Huovilainen, "Antialiasing oscillators in subtractive synthesis," *IEEE Signal Processing Magazine*, vol. 24, pp. 116–125, 2007.

[41] H. Chamberlin, *Musical Applications of Microprocessors*. Rochelle Park, NJ: Hayden Book Company, 1980.

[42] V. Zavalishin, "The art of VA filter design," Native Instruments, Berlin, Germany, Tech. Rep., 2012.