Agentic Coding with AI: A case study to develop a program for testing synthetic drum models

Robert Manzke[a]

[a]Kiel University of Applied Sciences, Faculty of Computer Science and Electrical Engineering, Grenzstraße 5, 24149 Kiel, Germany

ABSTRACT

The advent of Agentic Coding, where an AI generates code based on requirements prompted by a developer, allows one to rapidly prototype ideas. The focus of the developer shifts from programming to prompting things into existence. AI-based tools for programmers become more powerful on a daily basis: Tools like Github Copilot, Claude Code and their integration into IDEs such as Jetbrains, Cursors or VSCode promise to accelerate development times and shift the developer attention away from implementation to integration and validation.

This paper evaluates the usefulness of AI-systems to rapidly implement a C++ program for real-time of testing synthetic drum models. Two publications written in Swedish language – which the Author of this paper is not proficient of –, available in pdf format were used as input data. They contain text and graphics illustrating the DSP-blocks of the synthesis models at an abstract level. As a first step ChatGPT was instructed to interpret those papers, which were uploaded in their original pdf form, and generate C++ classes of their implementation including a description of the model parameters. The resulting source code was moved to an empty Jetbrains CLion project. With the help of the plugin version of Github Copilot in Agent mode (preview at the time of the writing), the AI was instructed to create a project structure, using CMake as a built system, DearIMGui and RtAudio as libraries for graphical UI and real-time audio IO and Github Actions for pipelined CI/CD.

It was possible to create a working C++ application in a few hours, which faithfully represents the drum models, allows for DSP parameter tweaking and real-time visualization of the generated waveforms both in time-base and spectrogram view.

In conclusion, Agentic Coding allows rapid prototyping within hours compared to days or weeks. The generated code requires further quality assessment and review.

Keywords: Agentic Coding, AI-Assisted Programming, Drum Synthesis, Digital Musical Instruments, C++ Programming, Real-Time Audio Programming

Author e-mail: robert.manzke@fh-kiel.de

1. INTRODUCTION

Automated software development has been a long-term goal in artificial intelligence and software engineering.[1] The availability of large language models (LLMs) such as GPT-4,[2] Claude[3] and LLaMas[4] has opened new possibilities for automated code generation.[5] These models can understand natural language and are able to generate code in various programming languages based on textual descriptions of desired functionality and requirements. They are able to answer complex domain-specific programming questions, explain code, generate documentation and tests, are able to refactor code and even debug it.[6,7]

This has led to the emergence of AI-powered coding assistants such as GitHub Copilot,[8] Amazon CodeWhisperer[9] and Tabnine,[10] which are integrated into popular IDEs and provide real-time code suggestions and completions. These systems have been extended to support multi-modal input, including images and pdf documents. This allows developers to provide more context and information to the AI system, which can lead to more accurate and relevant code generation. Only recently, the AI is able to autonomously use tools and APIs to allow for agentic behaviour.[11–13] Agentic coding describes a workflow where the AI system is given a high-level goal or task and is able to decompose it into sub-tasks, use external tools and APIs to gather information, generate code and validate the results. The developer's role shifts from writing code to defining the task, providing feedback and validating the results (see Figure 1).

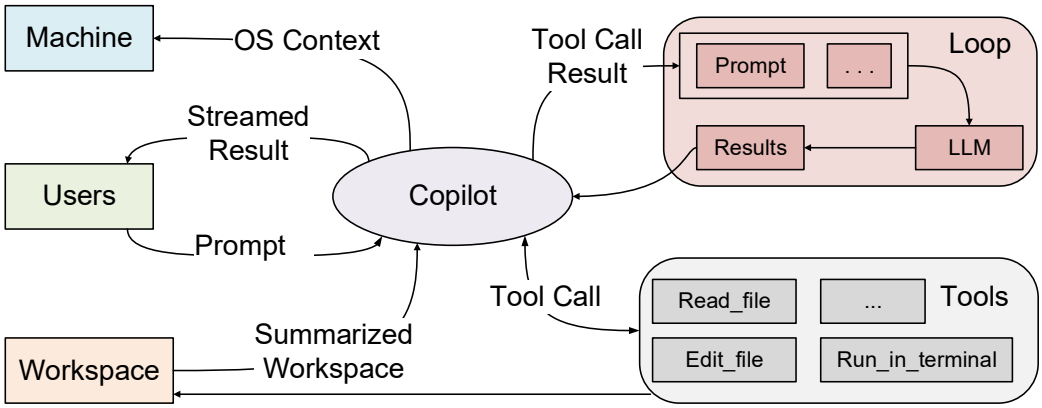

Figure 1. Agentic coding workflow using Copilot.[13]

Vibe-coding[14,15] describes the concept of agentic coding where the human developer accepts AI-suggested completions liberally avoiding detailed managing of the code. The focus is more on the overall *vibe* of the project and its direction rather than on the details of the implementation. Following a fully agentic approach, the developer would only provide a high-level description of the desired functionality and let the AI system handle the rest. Eventually, the human is completely removed from the loop, and the AI system is given full control over the development process (see Figure 2).

Current toolchains are designed in a human centric way, abstracting away the internal complexity. AI agents would benefit from AI-centric toolchains and development environments with fine grained access to internal states and data structures. This poses the question if current tools are still suitable for AI-assisted programming or if new tools and paradigms are needed.[13] Another limiting factor is the context window size of current LLMs, which limits the amount of information that can be processed at once and reasoning over long context histories. This is especially relevant for complex software systems and large codebases.[13]

It is still an open question how well these systems help in real-world software development scenarios, especially for complex and domain-specific tasks.[16]

For algorithm research and evaluation, a system that can process scientific papers and generate code based on the described algorithms is highly desirable. It could allow researchers to quickly prototype and test new ideas without having to manually implement the algorithms from scratch. This paper presents a case study where an AI system is used to generate a C++ program for testing synthetic drum models based on two scientific papers[17,18] written in Swedish language. The papers describe various digital drum synthesis models, including

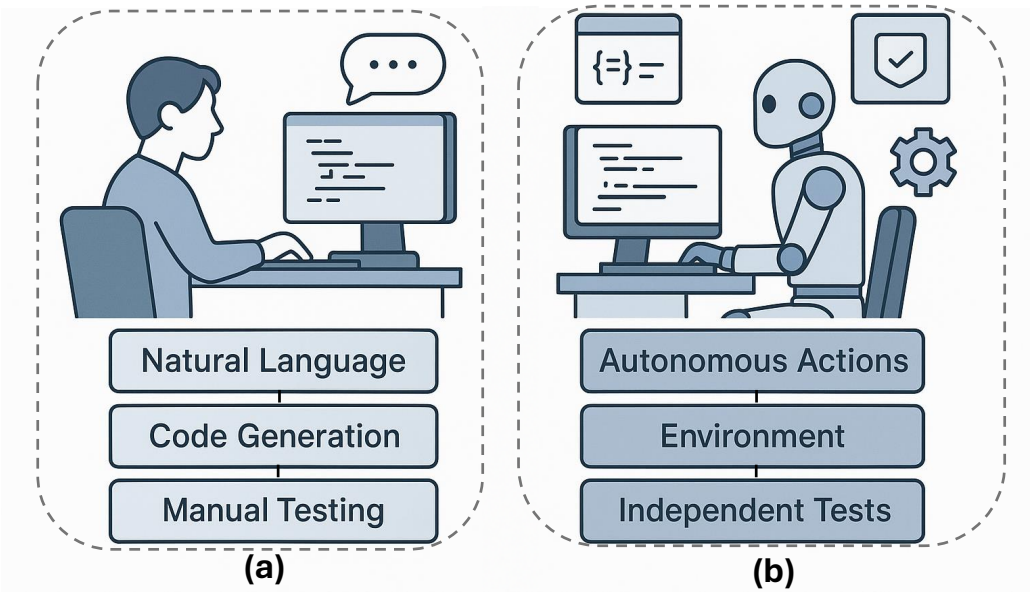

Figure 2. Vibe Coding (human interaction) vs. Fully Agentic Coding (no human interaction).[15]

their mathematical formulations and block diagrams. The goal of the case study is to evaluate the effectiveness of the AI system in generating a working C++ program that implements the described drum models and provides a user interface for parameter manipulation and real-time audio processing.

The paper is structured as follows: Section 2 outlines the case study, including methods, tools and desired outcome. Section 3 illustrates the results, including the generated code, its functionality and performance. Finally, Section 4 summarizes the findings and proposes future work.

2. CASE STUDY

The case study aims to evaluate the effectiveness of an AI system in generating a C++ program for testing synthetic drum models based on two scientific papers[17,18] written in Swedish language. The papers describe various digital drum synthesis models, including their mathematical formulations and block diagrams. The goal is to create a working C++ program that implements the described drum models, provides a user interface for parameter tweaking and allows for sound and algorithm evaluation.

Multiple AI systems were used in combination, as no single system was able to handle the entire workflow at the time of the case study. The AI systems included www-based ChatGPT-4.x with vision capabilities for interpreting the pdf documents and generating C++ code, and GitHub Copilot in Agent mode (preview at the time of the case study) integrated into Jetbrains CLion IDE* for interactive vibe-coding. Within Copilot the models GPT-4.x and Claude Sonnet 4 were used.

The project setup included CMake† as build system, DearIMGui‡ for graphical UI and RtAudio§ for real-time audio output. In order to test the generated C++ program, real-time sound output on a standard laptop sound card was used. A basic graphical user interface (GUI) was created using DearIMGui to allow users to manipulate model parameters in real-time. The program was set up for continuous integration and deployment (CI/CD) using GitHub Actions. All of the above was done on a standard Apple M3 laptop using a vibe-coding approach with minimal human code editing and review.

*CLion
†cmake.org
‡DearIMGui
§RtAudio

The development steps of the case study were as follows:

- Upload the pdf documents to www-based ChatGPT-4.x and prompt it to extract the relevant information about the drum synthesis models, including their mathematical formulations and block diagrams.
- Instruct ChatGPT-4.x to generate C++ classes for each drum model, including member variables for model parameters and methods for processing audio samples.
- Create a new empty C++ project in Jetbrains CLion and set up CMake as the build system.
- Use GitHub Copilot in Agent mode to generate the project structure, including CMakeLists.txt, main.cpp and necessary header files.
- Manual migration (copy-paste) of the generated drum model C++ classes from www-based ChatGPT-4.x to the CLion project.
- Prompt Copilot to add DearIMGui for the graphical user interface, allowing users to manipulate model parameters in real-time.
- Instruct GitHub Copilot to integrate RtAudio for real-time audio input/output, allowing users to hear the synthesized drum sounds.
- Prompt Copilot to add save/load functionality for model parameters using a simple text format.
- Use GitHub Copilot to set up GitHub Actions for continuous integration and deployment (CI/CD), including building the project.
- Iterate prompts and generated code as needed to add more models (from www-based ChatGPT) and to improve the program (object-oriented approach, parameters, feedback).

For embellishment of the GUI, the AI was additionally prompted to add real-time visualizations of the generated waveforms in both time-base and spectrogram view. Furthermore, the AI was instructed to add GLSL shaders[¶] to create visually appealing effects depending on the real-time audio output. A background image was automatically generated using GPT-4.o based on a text prompt and integrated into the GUI by the AI. Finally, the AI was prompted to write a README.md file for the GitHub repository, including instructions for building and using the program.

Prompts were partially recorded, but not systematically. Stages of the development process were committed to a GitHub repository,[19,20] where the reader may inspect the code evolution.

3. RESULTS

For the sake of brevity, only a subset of the results will be presented here. We focus on the AI implementation of the FM Kick Drum model (Sec. 4.1 in[17]) shown in Figure 3.

The pdf document was available as a scanned version of the original paper submission from Chalmers University of Technology. It was uploaded to ChatGPT-4.o. Then the AI was prompted to generate a C++ class implementing the FM Kick Drum model, embedded in a basic CMake-based project with RTAudio to provide real-time audio output and with a simple user interface to test the model parameters. The generated code was downloaded from ChatGPT as zip file and manually migrated to a new empty CLion project. Results are shown as a first commit in the GitHub repository[19] and in Listing 1.

Looking at the generated code and comparing it to the block diagram in Figure 3, it can be seen that the AI was mostly able to correctly understand the paper and generate a corresponding C++ class. All three modulation envelopes are implemented correctly. The amplitude modulation envelope correctly applies to the final output, creating the typical transient sound of a kick drum. The frequency modulation envelope (left branch of the block diagram) modulates the frequency of the carrier oscillator, creating the characteristic pitch drop of a kick drum.

[¶] OpenGL Shading Language

4.1 Bastrumma

Denna modell består av grundmodellen minus återkoppling på bärvågen (fig. 4.1). Rattarna på Machinedrum är kopplade till följande parametrar:

- *Frekvens (f_b)*
- *Decay (utklingningstid för volymenveloppen) (d_b)*
- *Modulatorfrekvens (f_m)*
- *Modulationsdjup (I)*
- *Modulatordecay (d_m)*
- *Modulatoråterkoppling (b_m)*
- *Amplitud för frekvensenveloppen (A_f)*
- *Decay för frekvensenveloppen (d_f)*

För ett mer distinkt anslag startas oscillatorerna med en fasförskjutning på $\pi/2$ rad (dvs. cosinus). Det skarpa steget i början av vågformen skapar ett högfrekvent "klickljud" som påminner lite om när hammaren träffar trumskinnet. Mest realistiska ljud får man när modulatorn återkopplas så att den brusar och modulatorenveloppen är relativt snabb. Annars kan modellen varieras i oändlighet och skapa alla möjliga underliga ljud.

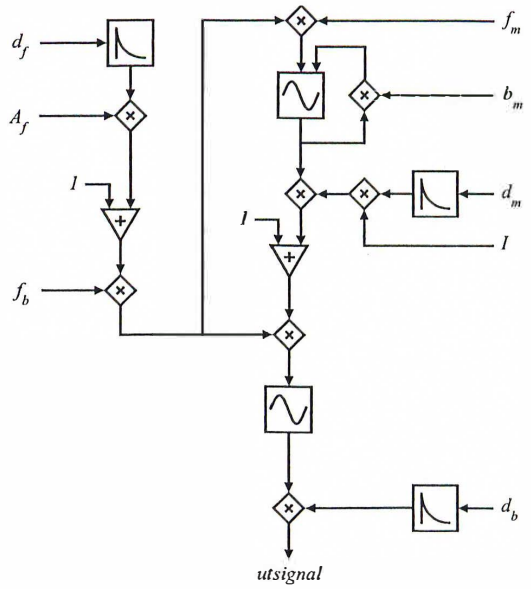

Figure 3. FM Kick Drum synthesis model block diagram.[17]

```cpp
constexpr float PI = 3.14159265f;
constexpr float TWO_PI = 2.0f * PI;
constexpr float SAMPLE_RATE = 48000.0f;
constexpr size_t BUFFER_SIZE = 256;

float WrapPhase(float phase) {
    while (phase >= TWO_PI) phase -= TWO_PI;
    while (phase < 0.0f) phase += TWO_PI;
    return phase;
}

float ExpDecay(float t, float decay_time) {
    return std::expf(-t / decay_time);
}

class FmKick {
public:
    void Init() {
        mod_phase = car_phase = t = 0;
        prev_mod = 0;
    }

    void Trigger(float f_b_, float d_b_, float f_m_, float I_, float d_m_,
                 float b_m_, float A_f_, float d_f_) {
        f_b = f_b_; d_b = d_b_; f_m = f_m_; I = I_; d_m = d_m_;
        b_m = b_m_; A_f = A_f_; d_f = d_f_;
        t = 0.0f; mod_phase = 0.0f; car_phase = PI / 2.0f; // Cosine start
    }

    float Process() {
        float dt = 1.0f / SAMPLE_RATE;
        float amp_env = ExpDecay(t, d_b);
        float mod_env = ExpDecay(t, d_m);
        float freq_env = A_f * ExpDecay(t, d_f);

        float mod_feedback = b_m * prev_mod;
        mod_phase = WrapPhase(mod_phase + TWO_PI * f_m * dt + mod_feedback);
        float mod_out = std::sin(mod_phase);
        prev_mod = mod_out;

        car_phase = WrapPhase(car_phase + TWO_PI * (f_b + freq_env) * dt + I * mod_env * mod_out);
        float out = std::sin(car_phase) * amp_env;

        t += dt;
        return out;
    }

private:
    float f_b, d_b, f_m, I, d_m, b_m, A_f, d_f;
    float mod_phase = 0.0f, car_phase = 0.0f, prev_mod = 0.0f, t = 0.0f;
};
```

Listing 1: FmKick class, first commit.

The AI, however, made a mistake, as the frequency modulation envelope should also modulate the modulator frequency (upper right branch of the block diagram). This would scale the modulator frequency in sync with the carrier frequency, keeping their ratio constant, which is important for the timbre of the sound. The modulation of the carrier (right branch) by the modulator is implemented correctly as the modulation index I is multiplied with the modulator output and modulation envelope. This creates the specific transients pertinent to the initial attack of the kick drum sound. The AI did not correctly implement adding the modulation index I as a constant both in the carrier frequency modulation path (left branch) and in the carrier modulation path (right branch). This would create a base frequency offset for both paths, which is important for the overall pitch of the sound. The overall pitch of the sound in the AI's implementation is part of f_b, in contrast to the block diagram. Otherwise, the code is well-structured and readable, with variable names reflecting the block diagram's annotations.

Later on in the vibe-coding process an object-oriented design *was prompted* in CLion with Copilot with a common base class for all drum models and modularization into separate files. The GUI was added, including the required functionality.

The AI was prompted to extract more drum models and generate C++ classes for them, including the generated inheritance structure. This was done in the www-based ChatGPT-4.x interface with manual migration to the CLion project.

Evaluating the final program[20] (available to the reader on Github, including sound examples) the AI produced well-structured code, with a modular design. The GUI allows for real-time manipulation of model parameters and visualization of the generated waveforms in both time-base and spectrogram view. The program performs real-time audio processing on a desktop computer, allowing for model evaluation and tweaking. Peer review by an expert developer revealed, however, that some basic real-time audio programming best practices were not followed, such as avoiding dynamic memory allocation in the audio callback and use of potentially blocking mutexes.

The author of this paper estimates that for the presented case study, the development time was reduced by approximately 90%, where the focus shifted from programming to vibe-coding and reviewing.

4. CONCLUSION

The case study demonstrates the potential of AI systems in generating a working C++ program for testing algorithms described in scientific papers. The AI was able to interpret the block diagrams, descriptions and mathematical formulations in the papers and generate corresponding C++ classes.

It was able to generate a working program in a significantly reduced time compared to manual implementation. The exact time savings, however, are difficult to quantify as it depends on the complexity of the models and the developer's expertise. More time was available for actual testing and evaluating the models, tweaking their parameters which was the main goal of reimplementing the models. Less time was spent on the actual coding.

A significant risk in the development process is the accuracy of the generated code, that is, how well the code reflects the original algorithms and models described in the papers. In this case study, the AI did make some mistakes or rather *creatively misinterpreted* the block diagrams, which could lead to incorrect or unexpected behavior in the generated program. This requires careful review and validation by expert developers, which is time-consuming and offsets the gain of development speed. For non-expert developers, the temptation is to accept the generated code as is, which can lead to potential issues in the program. Before using AI for the interpretation of papers, a critical reflection is needed. In cases where high accuracy is required, an expert review is mandatory.

In future, special adaptations will be *vibe-coded* to allow the algorithms to work on embedded platforms. This will involve the usage of existing embedded DSP libraries. AI may be used here to explain existing libraries and how to best apply them to the drum models.

ACKNOWLEDGMENTS

Thanks to J. Mamarella for pointers to relevant literature and K. Duske for code review.

REFERENCES

[1] Solar-Lezama, A., *Ph.D. Dissertation: Program synthesis by sketching*, PhD thesis, UC Berkeley (2008).

[2] OpenAI, "Gpt-4 technical report." https://arxiv.org/abs/2303.08774 (2023). Accessed: August 2025.

[3] Anthropic, "Introducing claude: A next-generation ai assistant." https://www.anthropic.com/index/introducing-claude (2023). Accessed: August 2025.

[4] AI, M., "Llama: Open and efficient foundation language models." https://ai.meta.com/llama/ (2025). Accessed: August 2025.

[5] Xinyi, H. and et al., "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology* **33**(8), 1–79 (2024).

[6] Chen, M. and et al., "Evaluating large language models trained on code." https://arxiv.org/abs/2107.03374 (2021).

[7] Wang, X. and et al., "Self-consistency improves chain of thought reasoning in language models." https://arxiv.org/abs/2203.11171 (2023).

[8] "Github copilot: Your ai pair programmer." https://github.com/features/copilot (2022). Accessed: August 2025.

[9] "Amazon codewhisperer: Your ai coding companion." https://aws.amazon.com/codewhisperer/ (2022). Accessed: August 2025.

[10] "Tabnine: Ai code completion tool." https://www.tabnine.com/ (2023). Accessed: August 2025.

[11] Schick, T. and et al., "Toolformer: Language models can teach themselves to use tools," *Advances in Neural Information Processing Systems* **36**, 68539–68551 (2023).

[12] "Agentic coding project website." https://agentic-coding.github.io/ (2025). Accessed: August 2025.

[13] Wang, H., Gong, J., Zhang, H., and Wang, Z., "Ai agentic programming: A survey of techniques, challenges, and opportunities." https://arxiv.org/abs/2508.11126 (2025).

[14] Karpathy, A., "Will the future of software development run on vibes?." https://arstechnica.com/ai/2025/03/is-vibe-coding-with-ai-gnarly-or-reckless-maybe-some-of-both/ (2025). Accessed: August 2025.

[15] Sapkota, R., Roumeliotis, K. I., and Karkee, M., "Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic ai." https://arxiv.org/abs/2505.19443 (2025).

[16] Uplevel, "Can genai actually improve developer productivity?." https://resources.uplevelteam.com/gen-ai-for-coding (2024). Accessed: August 2025.

[17] Larsson, E., "Syntes av trum-och percussionljud med hjälp av frekvensmodulering," (2000).

[18] Möllerstedt", D., "Digital drum synthesis trx for the elektron machinedrum sps-1," (2004).

[19] Manzke, R., "Initial commit of md-drum-synth." https://github.com/ctag-fh-kiel/md-drum-synth/tree/c0c7c232136b1c7b44e4f237e2860258329c98e8 (2025). Accessed: August 2025.

[20] Manzke, R., "Public repository of md-drum-synth." https://github.com/ctag-fh-kiel/md-drum-synth (2025). Accessed: August 2025.