Leveraging Agentic Coding for Interactive Education in Electrical Engineering

Bartosz Sawicki, Tomasz Les, Tomasz Grzywacz
Warsaw University of Technology,
Warsaw, Poland
Email: {bartosz.sawicki, tomasz.les, tomasz.grzywacz}@pw.edu.pl

***Abstract*—Large language models enable the automated synthesis of interactive educational software beyond conversational tutoring. This paper presents an authoring methodology using agentic coding (an autonomous, tool-augmented workflow based on the Reason-and-Act (ReAct) paradigm) to generate standalone, browser-based simulation modules for electrical engineering curricula. It was applied across three undergraduate courses, using three distinct agentic development environments: a transient commutation simulator for second-order RLC circuits (circuit theory), an eddy-current induction laboratory (electromagnetism), and an interactive benchmark of iterative linear solvers (numerical methods). We formalize a pipeline built on (1) strict decoupling of pedagogical specification from implementation, (2) formal mathematical notation constraints, (3) single-pass holistic generation to prevent context window degradation, (4) human-in-the-loop validation gates, and (5) prompt-level regeneration rather than conversational code patching. Recurring failure modes: notation drift, context saturation, feature creep, and physically invalid simulations ("vibe physics"); are analyzed together with their mitigations, and an independent analytical audit of two generated modules is reported. Modules are synthesized in 5–10 minutes over 3–5 specification iterations, shifting instructor effort from manual programming to requirements engineering and domain verification.**

***Index Terms*—Agentic coding, LLM agents, virtual laboratories, electrical engineering education, authoring workflow, transient analysis, eddy currents, numerical methods.**

I. Introduction

Computer-assisted instructional tools in electrical engineering frequently rely on static diagrams, pre-rendered animations, or proprietary desktop simulation suites. While interactive virtual laboratories achieve learning outcomes comparable to physical setups [1]–[3], their widespread adoption is constrained by high development costs. Manual implementation of bespoke, browser-based numerical simulations requires substantial software engineering effort, limiting curriculum coverage and long-term maintainability.

Research on large language models (LLMs) in engineering education primarily focuses currently on two domains: conversational tutoring systems [4]–[7] and automated code generation within computer science pedagogy [8], [9].

This paper investigates the intersection of these domains: utilizing LLMs as autonomous software authors for interactive educational artifacts. Rather than deploying an LLM as a direct student-facing conversational interface, we utilize agentic coding to develop standalone, interactive simulation software. Unlike unstructured, conversational code generation (often termed "vibe coding"), agentic workflows follow the Reason-and-Act (ReAct) paradigm [10]: the agent autonomously plans, executes local developer tools (file I/O, execution environments), analyzes feedback, and iterates toward a specified objective. This workflow transforms the instructor's role from manual coding to formal requirements specification and domain verification [11].

To ensure pedagogical validity, design constraints derived from cognitive load theory [12] and active learning frameworks [6], [13]–[15] are embedded directly into the prompt specifications. Specifically, generated artifacts must enforce parameter exploration, step-by-step derivations, and interactive self-assessment rather than merely displaying final numerical solutions.

The contributions of this paper are:

1) A structured, multi-phase agentic authoring workflow for simple engineering simulations (Section II).
2) Three case studies spanning circuit theory, electromagnetism, and numerical methods, including a detailed physical and numerical audit (Section III).
3) A taxonomy of failure modes in agentic code generation for scientific education and practical mitigation strategies (Section IV).

II. Agentic Authoring Workflow

The authoring workflow was evaluated across three undergraduate courses using several agentic development environments (Claude Code, Gemini Antigravity, OpenAI Codex and OpenCode+GLM) each executed locally with file system and terminal access. Multiple trials and prompting strategies was applied. Despite differences in subject matter, the best results have been obtained by generation process converged onto the unified pipeline shown in Fig. 1.

A. Phases

1) Scope and audience: Explicit specification of target student level, prerequisites, and learning objectives to prevent overly broad or encyclopedic implementations.

2) Context pack: Provision of the course syllabus, a mathematical notation specification, and a minimal HTML/CSS template to enforce visual consistency with minimal token overhead. Supplying complete prior applications as context degrades output quality (Section IV).