Vibe-Coding Physics Simulations to Enhance Instruction

Vladimir Lopez
The Kinkaid School
Email: vladimir.lopez@kinkaid.org

July 16, 2025

Abstract

"Vibe-coding"—using natural language to generate code with AI—now allows educators to create custom teaching tools. This paper describes how the author used this method with GitHub Copilot and Visual Studio Code to build two high school physics simulations: a tangent-line analyzer for kinematics and an electrostatics visualization tool. This approach empowers teachers to develop and deploy their own pedagogically targeted simulations.

1 Introduction

There are many excellent online simulations for physics, but even the best rarely align perfectly with a teacher's personal style, specific lesson goals, or the needs of a particular group of students. One way to address this misalignment is to adapt our teaching to fit the available tools. A more empowering alternative is to vibe-code our own simulations.

The term vibe-coding was coined by Andrej Karpathy, cofounder of OpenAI, in early 2025 [1]. In simple terms, it refers to the practice of describing your programming goals in natural language to an AI agent, which then generates the necessary code.

Like many teachers, I learned to code in college, but rarely used those skills outside of helping students in robotics. In this paper, I share my personal experience using this approach to create two simulations tailored to the specific needs of my high school physics class.

2 Implementation

My initial vibe-coding project in 2024, a simple roleplaying game chatbox, highlighted the limitations of early AI workflows. Although AI platforms have significantly improved since then, I eventually settled into a more productive workflow using GitHub Copilot inside Visual Studio Code, which remains my preferred setup.

My current setup includes:

- Visual Studio Code as the coding environment
- A GitHub Education account that provides free access to Copilot Pro
- GitHub repositories to organize and sync projects
- GitHub Pages to deploy simulations online

Although this might seem complicated at first, once everything is installed, the process becomes simple: create a repository, open it in VS Code, and have the Copilot agent create the codespace. More information is included in Appendix A, and a detailed, hands-on guide is available online (URL withheld for anonymous review).

3 Tangent-Line Simulation

In our high school physics unit on motion, students analyze position-time graphs from Galileo's ramp experiment, fitting quadratic curves and estimating instantaneous velocity via tangent lines. An unsupported software tool previously aided this, but its replacement lacked the ability to analyze tangents between data points, hindering students' visualization of instantaneous velocity as a continuous slope. To address this, I vibe-coded an online app that plots data, fits a parabola, and allows interactive tangent-line analysis anywhere along the curve.

These are the steps I followed:

1. Created a public repository in GitHub.
2. Opened the repository in VS Code. This can be done by asking the Copilot chat agent to do it for you. As a general rule, I asked the agent to commit and push the code to the repository instead of doing it manually.
3. Prompted Copilot with the following:

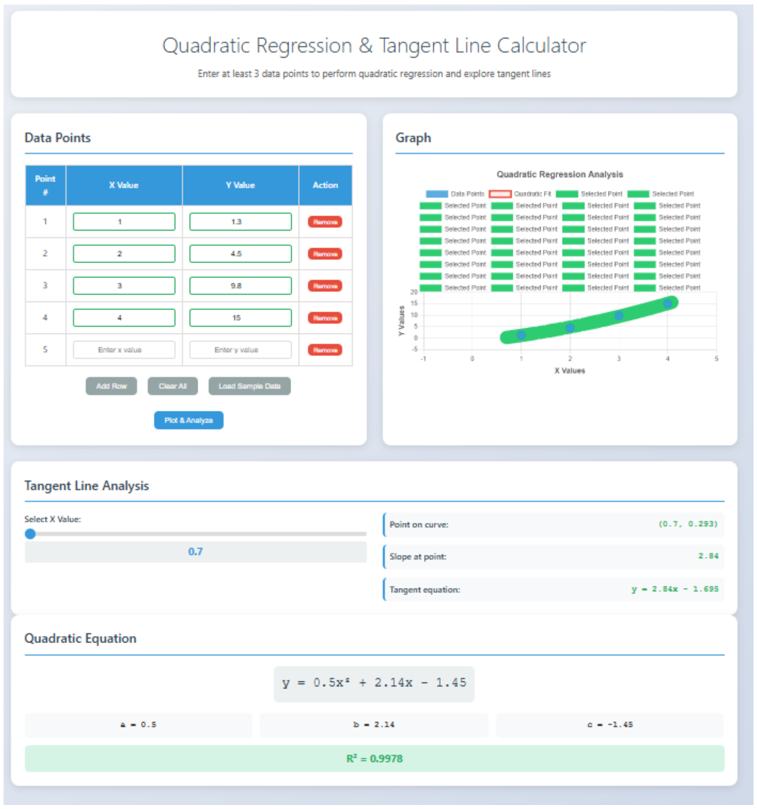
   Create an HTML5 web application using HTML, CSS, and JavaScript. The app should have the following features:

   - A table where users can input x and y values (minimum of 3 points required for a quadratic fit).
   - A button to plot the data and perform a quadratic regression (least squares fit).
   - A canvas (e.g., using Chart.js or plain HTML5 Canvas) to display:
     - The input data points.
     - The best-fit quadratic curve based on the input data.
   - An interactive slider or input box to select an x-value within the domain of the fitted parabola.
   - When an x-value is selected:

  - Highlight the corresponding point on the quadratic curve.
  - Calculate and display the slope of the tangent line at that point.
  - Draw the tangent line on the graph.
- Style the app with simple, responsive CSS for clarity and usability.

Do not use any backend code. All functionality should be done client-side. Keep the code modular and well-commented. Use plain JavaScript unless a library like Chart.js or math.js makes things clearer.

Copilot created the files and folders, added the code, and displayed a working prototype (see Figure 1).



Quadratic Regression & Tangent Line Calculator

Enter at least 3 data points to perform quadratic regression and explore tangent lines

Data Points

Point #	X Value	Y Value	Action
1	1	1.3	Remove
2	2	4.5	Remove
3	3	9.8	Remove
4	4	15	Remove
5	Enter x value	Enter y value	Remove

Graph

Tangent Line Analysis

Select X Value: 0.7

Point on curve: (0.7, 0.293)

Slope at point: 2.84

Tangent equation: y = 2.84x - 1.695

Quadratic Equation

y = 0.5x² + 2.14x - 1.45

a = 0.5 b = 2.14 c = -1.45

R² = 0.9978

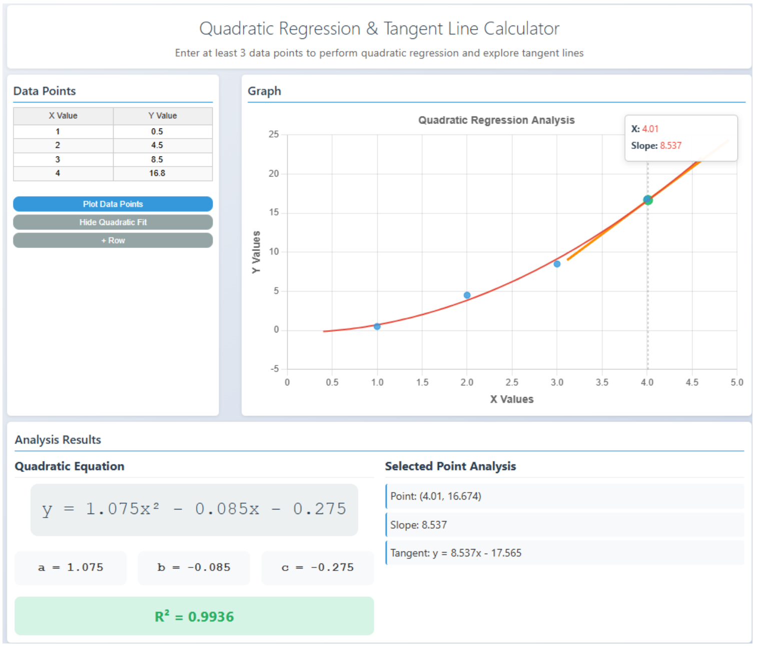
Figure 1: The initial prototype of the tangent-line simulation, generated by the AI agent. Note the basic layout and functionality.

As expected, some adjustments were needed. I continued prompting Copilot with refinements:

- Simplify the data table (remove unnecessary columns).
- Make points on the graph smaller and add the quadratic curve.
- Remove unneeded labels and legends.

- Add a vertical line at the selected point.
- Replace the slider with direct graph interaction.
- Simplify the overall layout.

The final product allowed my students to input data, visualize a quadratic fit, and explore slopes of tangent lines intuitively. It was deployed to GitHub Pages and used in class (see Appendix B).


Quadratic Regression & Tangent Line Calculator

Enter at least 3 data points to perform quadratic regression and explore tangent lines

Data Points

X Value	Y Value
1	0.5
2	4.5
3	8.5
4	16.8

Analysis Results

Quadratic Equation

y = 1.075x² − 0.085x − 0.275

a = 1.075 b = −0.085 c = −0.275

R² = 0.9936

Selected Point Analysis

Point: (4.01, 16.674)

Slope: 8.537

Tangent: y = 8.537x - 17.565

Figure 2: The final, refined version of the tangent-line simulation after several iterations of vibe-coding.

4 Electrostatics Simulation

During our unit on charging by conduction and induction, my students often struggled to visualize how electrons move in different scenarios. While I used a traditional setup with a hanging pith ball and charged rods, the demonstration felt too passive and didn't clearly show the direction of charge flow. Students particularly struggled to visualize the dynamic redistribution of charge and the flow of electrons during these processes, which traditional demonstrations often fail to convey clearly. That inspired me to create an interactive simulation that would let students explore these processes directly.

As with the first project, I began by creating a file with lesson goals and key physics concepts to help "ground" the AI. This involved providing the AI with our teacher's guide and other relevant pedagogical documents, ensuring the generated simulation aligned with our specific instructional objectives and addressed common student misconceptions. This document also became the embedded reference guide within the simulation.

I then prompted Copilot to generate a simulation where users could:

- Select a material (e.g., rod, conductor)
- Apply positive or negative charges
- Move objects around
- Observe the effects of conduction and induction visually
- Add or remove grounding

The AI-generated code wasn't perfect on the first try, but through a series of refinements—adding draggable elements, labeling charges, adjusting UI—Copilot and I created a usable, engaging model of electrostatic interactions. Students were able to explore and compare different configurations, and the simulation made abstract charge behavior much more visible. The final product is in Figure 3.

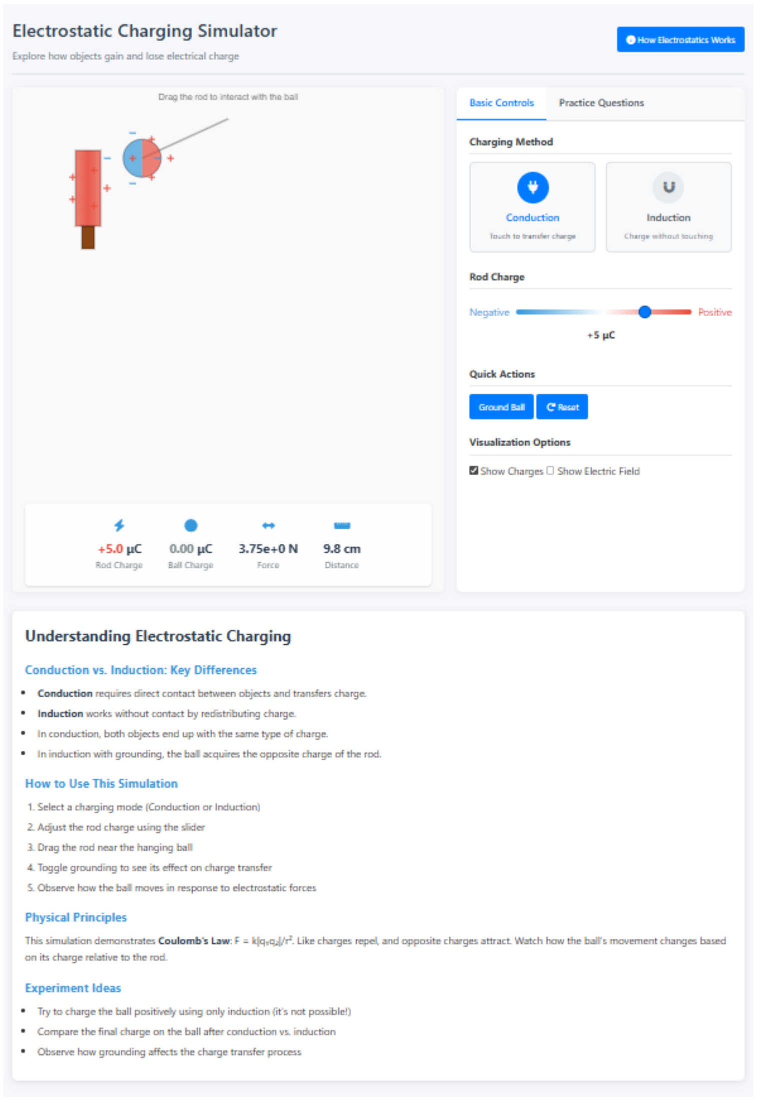
Both simulations were created entirely with vibe-coding, deployed online, and are shared in Appendix B.

5 Reflections

Since I began experimenting with vibe-coding in 2024, the tools have improved dramatically. The introduction of agent-based functionality in GitHub Copilot has been a major step forward. The "agent" can now create files, write code, and even execute and analyze terminal commands. The language models themselves are now far more capable. For example, Claude Sonnet 4 can generate nearly functional simulations after a single, well-structured prompt. What once required a dozen back-and-forth iterations can now be accomplished in just a few. These developments have made the process faster and more accessible, though not without its challenges.

Even with better tools, the process can still be frustrating. There are times when the AI doesn't implement requested changes correctly or repeats earlier mistakes. One workaround I've found effective is taking screenshots of the simulation and sharing them with the Copilot agent. This approach only works with multimodal languages like Claude, Gemini, or GPT 4.1, but significantly improves the feedback loop. That said, human oversight remains essential, especially to ensure that the physics is correct. Grounding AI with the lesson documents and clearly defined concepts made a noticeable difference in accuracy and relevance.

I deliberately avoided editing the code by hand, staying true to the spirit of vibe-coding: building functional tools through language-based interaction. This approach helped keep the process accessible and replicable for other educators. Students' response was encouraging, particularly for the electrostatics simulation, which helped them better visualize the electrostatic charging and the flow of electrons. The tangent-line simulation served more as a digital tool. Both gained some attention in class and gave me a bit of 'street cred' for building custom tools tailored to our lessons.



Electrostatic Charging Simulator

Explore how objects gain and lose electrical charge

Charging Method: Conduction

Rod Charge: +5 μC

Quick Actions

Visualization Options: [✓ Show Charges] Show Electric Field

+5.0 μC Rod Charge | 0.00 μC Ball Charge | 3.75e+0 N Force | 9.8 cm Distance

Understanding Electrostatic Charging

Conduction vs. Induction: Key Differences

- **Conduction** requires direct contact between objects and transfers charge.
- **Induction** works without contact by redistributing charge.
- In conduction, both objects end up with the same type of charge.
- In induction with grounding, the ball acquires the opposite charge of the rod.

How to Use This Simulation

1. Select a charging mode (Conduction or Induction)
2. Adjust the rod charge using the slider
3. Drag the rod near the hanging ball
4. Toggle grounding to see its effect on charge transfer
5. Observe how the ball moves in response to electrostatic forces

Physical Principles

This simulation demonstrates **Coulomb's Law**: $F = k|q_1q_2|/r^2$. Like charges repel, and opposite charges attract. Watch how the ball's movement changes based on its charge relative to the rod.

Experiment Ideas

- Try to charge the ball positively using only induction (it's not possible!)
- Compare the final charge on the ball after conduction vs. induction
- Observe how grounding affects the charge transfer process

Figure 3: The final version of the electrostatics simulation, showing the interaction between a charged rod and a neutral conductor.

6 Recommendations

Based on my experience building these simulations, I've identified a few suggestions for teachers interested in trying vibe-coding in their classrooms. These tips are meant to lower the barrier to entry and help others avoid some of the early frustrations I encountered.

There are many platforms for vibe-coding, but the combination of GitHub Copilot and Visual Studio Code stands out for its flexibility and power—especially since it's free for educators through GitHub's education program. VS Code supports a wide range of languages, including Python and JavaScript, making it suitable for everything from simple sketches to more complex simulations. Just as important, this setup models real-world coding workflows, with version control and clean, shareable repositories.

Patience is essential. Code rarely works on the first try—not even for professionals.

Adopting a mindset of iterative development and feedback loops is key. Vibe-coding is not about getting the perfect simulation immediately, but about gradually shaping it through collaboration with the AI.

Above all, start with problems you know well. When you build tools grounded in your own teaching expertise, the AI's output becomes more relevant, and your feedback to the agent more accurate. The human-in-the-loop aspect isn't optional—it's what ensures the simulation works, both technically and pedagogically.

7 Conclusion

AI is already changing how teachers work by helping with lesson planning, grading, and creating instructional materials. More than just a technical skill, vibe-coding is a pedagogical strategy. It directly supports established teaching frameworks like Modeling Instruction, where students build and test their own conceptual models of the physical world [2]. By creating a simulation, the teacher engages in a similar process, refining their own understanding of a concept to make it explicit and interactive. Furthermore, these custom-built tools are ideal for inquiry-based learning, allowing students to ask "what if" questions and explore the consequences in a virtual environment that is perfectly tailored to the lesson's objectives. This moves students from passive observers to active participants in the scientific process.

But for teachers willing to go a step further, vibe-coding offers something more: the ability to design custom tools that match their teaching style and their students' needs. It empowers educators to become creators, not just consumers, of digital content.

For teachers with even a little coding background, no matter how out-of-date, vibe-coding offers a way to reconnect with that knowledge and put it to work in a new, meaningful context. It is useful, it is creative, and it is surprisingly fun (once you overcome the frustration of not-working iterations of the code). Unlike using AI as a chat assistant or a document writer, vibe-coding is an active, hands-on process that blends pedagogy, problem solving, and personal invention. For me, it has opened a door to building the kinds of tools I always wished I had. I believe other teachers can do the same.

AI Assistance Acknowledgment

The author utilized a large language model (Gemini) for assistance with grammar, clarity, and stylistic improvements during the preparation of this manuscript. The content, analysis, and conclusions remain the sole responsibility of the author.

References

[1] A. Karpathy, "Prompt engineering is the new software engineering," X (formerly Twitter), `https://twitter.com/karpathy/status/1753124588808980917` (Feb. 2, 2025).

[2] M. Wells, D. Hestenes, and G. Swackhamer, "A modeling method for high school physics instruction," Am. J. Phys. **63**, 606–619 (1995).

A GitHub and Setup Resources

- **GitHub Education:** Apply for a free teacher or student account, which includes Copilot Pro access.
- **GitHub Copilot Documentation:** Learn how to use Copilot and Copilot Chat inside Visual Studio Code.
- **GitHub Pages:** Free static site hosting for simulations and web apps.
- **Hello World: GitHub Quickstart:** Step-by-step guide to creating and publishing your first repository.
- **GitHub Docs:** Official documentation homepage.

B Simulation Links

B.1 Tangent Line Simulation

- Live site: `https://vladimirlopez.github.io/slope_tangents/`
- Source code: `https://github.com/vladimirlopez/slope_tangents.git`

B.2 Electrostatics: Conduction and Induction

- Live site: `https://vladimirlopez.github.io/chargingDemo/`
- Source code: `https://github.com/vladimirlopez/chargingDemo.git`