Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #45592

BOARD # 72: Leveraging Large Language Models to Create Interactive Online Resources for Digital Systems and Computer Architecture Education

Dr. Peter Jamieson, Miami University

Dr. Jamieson is an assistant professor in the Electrical and Computer Engineering department at Miami University. His research focuses on Education, Games, and FPGAs.

Ricardo Ferreira, Universidade Federal de Viçosa
José Nacif, Universidade Federal de Viçosa

Leveraging Large Language Models to Create Interactive Online Resources for Digital Systems and Computer Architecture Education

Abstract

This paper explores the innovative use of Large Language Models (LLMs) to help create interactive online educational resources for digital system design and computer architecture courses, typically in electrical and computer engineering undergraduate courses. We present a framework that uses LLMs and existing online tools such as Google Colab and Gradio to rapidly develop and deploy interactive simulations, exercises, and automated grading systems to help learners work in the space with minimal need for up-front teacher development or full-end industrial tools associated with them. Our approach shows how to reduce the time and effort required for instructors to create engaging, personalized online learning experiences. We evaluate the effectiveness of this method through a series of case studies and provide guidelines for instructors to leverage these technologies in their courses.

1 Introduction

Large Language Models (LLMs) and their emerging skills provide educators with new capabilities to improve our teaching and save time. LLMs like ChatGPT have emerged as powerful tools that can assist in creating educational content and interactive learning experiences [1].

For digital system design and computer architecture, traditional education often relies on expensive hardware, specialized software, and physical laboratory spaces. These requirements can limit access to hands-on learning experiences, particularly for students in resource-constrained environments or those engaged in distance learning. Our previous work demonstrated the effectiveness of using Google Colab and Gradio to create interactive online learning modules for these subjects [2]. This paper builds upon that foundation, exploring how LLMs can further enhance the creation and customization of these educational resources. Additionally, we use these modules in various settings, but these tools have not been formally added to specific courses and evaluated; instead, this work seeks to expose other educators to the benefit and ease with which these tools can be created.

Integrating LLMs with platforms like Google Colab and Gradio offers several significant benefits for educators and learners. First, it dramatically reduces the time and effort required for instructors to create interactive learning materials. LLMs can quickly generate initial code for simulations,

exercises, and demonstrations, which instructors can refine and customize. Second, this approach allows for rapid iteration and personalization of learning resources, enabling instructors to tailor content to specific learning outcomes or student needs. The downside of these approaches is that the LLMs can be used to solve the exercises, but this challenge is outside the scope of this work.

This work is key to democratizing access to high-quality, interactive learning experiences in digital system design and computer architecture. Students can access sophisticated simulations and exercises via a web browser, eliminating the need for expensive hardware or software licenses. This is particularly beneficial for institutions with limited resources, students in remote locations, or those engaged in self-directed learning. By leveraging cloud computing resources, learners can engage with computationally intensive simulations that might otherwise be beyond the capabilities of their devices and the high cost of getting local systems to allow for these experiences.

The main contributions of this paper are:

- A framework for using LLMs to generate interactive simulations and exercises for digital logic and computer architecture courses.
- Evaluation of ChatGPT's capabilities in creating Python code for various digital system components.
- Guidelines for instructors on effectively using LLMs to develop educational resources.
- Case studies (and shared code) demonstrating the application of our approach for course settings.

The rest of this paper is organized as follows. Section 2 provides a comprehensive background on using Jupyter Notebooks, Google Colab, Gradio, and LLMs in educational contexts. Section 3 details our methods for using LLMs to create the activities on Google Colab and Gradio as interactive learning resources. Section 4 presents several case studies showcasing the application of our approach in digital systems and computer architecture education. Section 5 discusses the results of our evaluation, including the effectiveness of LLM-generated code and the impact on student learning. Finally, Section 6 concludes the paper and outlines directions for future work.

2 Background

Here, we briefly review the tools we advocate for creating educational resources for learners and briefly reference work performed by others using these tools.

2.1 Jupyter Notebooks in Education

Jupyter Notebooks, most notably used in AI and data science spaces, have become increasingly popular in educational settings, offering an interactive environment that combines code execution, rich text, and visualization [3]. They provide a powerful platform for creating and sharing computational narratives, making them particularly useful for teaching programming and data science concepts [4].

Educators have recently explored various ways to leverage Jupyter Notebooks in their courses. For instance, Davies *et. al.* [5] demonstrated the effectiveness of interactive digital notebooks for

bioscience and informatics education. These notebooks allow students to dynamically engage with course material, promoting active learning and experimentation.

However, the adoption of Jupyter Notebooks in education is not without challenges. Lau *et. al.* [6] analyzed 60 different notebook systems, highlighting the complexity of the notebook ecosystem and the need for a better understanding of their design tradeoffs. To address some of these challenges, tools like JupyterHub and *nbgrader* have been developed to facilitate the deployment and grading of notebook-based assignments [7] [8].

2.2 Google Colab and Cloud-based Learning

Google Colab is a popular platform for cloud-based Jupyter Notebooks, offering free access to computing resources and easy sharing capabilities, serving over 10 million active users and being particularly well-suited for machine learning, data analysis, and education [9]. Kim and Henke [10] explored using Google Colab for teaching data science, emphasizing its accessibility and ease of use for students. The performance capabilities of Google Colab for deep learning applications are analyzed by Carneiro *et. al.* [11], demonstrating its potential for resource-intensive computational tasks in educational settings.

2.3 Gradio and Interactive Interfaces

Gradio [12] is a Python library that rapidly creates web-based interfaces for machine learning models and other computational tasks. While Gradio was initially focused on machine learning applications, educators in various fields have recognized its potential for creating interactive educational tools. In particular, our previous work [2] was one of the early works demonstrating the effectiveness of combining Google Colab and Gradio to create interactive online learning modules for digital systems education. This approach provides a flexible environment for developing and deploying educational simulations and exercises, addressing some of the limitations of traditional Jupyter Notebooks.

2.4 Large Language Models in Education

Large Language Models (LLMs) like ChatGPT have shown great potential in various educational applications, including content generation, question answering, and personalized tutoring [13, 14]. In engineering education, LLMs can assist instructors in quickly and efficiently creating diverse learning materials and interactive simulations.

Recent studies have explored the use of LLMs in educational settings. For example, Lee and Perret [15] investigated the preparation of high school teachers to integrate AI methods into STEM classrooms, highlighting the potential of tools like Google Colab in this process. LLMs as chatbots are a technology seeing great promise and hype, but still, it is so early that the possibilities are unknown. This work illustrates one approach to leveraging their potential benefit to education.

2.5 Technology Adoption for Education

Despite the advantages of these educational technologies, challenges still exist. Chattopadhyay *et. al.* [16] identified challenges to using computational notebooks, including setup difficulties and issues with customization. Additionally, the retention rates in MOOCs remain a concern, with studies showing that less than 10% of learners complete their courses [17]. However, these challenges also present opportunities for innovation. The combination of Jupyter Notebooks, cloud platforms like Google Colab, interactive tools like Gradio, and the emerging capabilities of LLMs offers a promising avenue for creating more engaging and effective online learning experiences.

Building upon our previous work [2], this paper explores how LLMs can be leveraged to enhance the creation of interactive resources for digital systems and computer architecture education, addressing some of the existing challenges in online education.

3 Methodology

Our approach combines the capabilities of LLMs with Jupyter Notebook libraries, Python graphical libraries, Google Colab, and Gradio to create interactive learning resources. In our previous work, we provide examples of interactive tools that include:

- MQQT Client/Server approach for creating and distributing assignments with Auto-grading
- Circuit Simulation with different inputs
- Input Value Stimulus and Waveform Viewer for observing and modifying circuit output responses
- Cache Memory Simulation and Design
- RISC-V Single Cycle and Pipeline
- Interactive Testbench Creation
- Combinational Logic Assignments (e.g., ripple carry structural assignment)
- Sequential Logic Assignments (e.g., traffic light finite state machine)
- Command Line Tool Integration (e.g., Cachegrind, gem5, ABC Berkeley synthesis package)

These examples are hosted on a Jupyter Notebook here `https://github.com/lesc-ufv/GradioColab/example1` and provide instructors with the basic form of creating these activities. However, this work aims to allow instructors to quickly create these types of interactions.

Our LLM-based approach involves the following steps:

1. Prompt Engineering: Develop effective prompts for LLMs to generate code for interactive components and exercises.
2. Code Generation: Use LLMs to generate initial code for simulations and exercises.
3. Refinement and Integration: Refine the generated code and integrate it with Gradio for interactive interfaces.
4. Deployment: Deploy the interactive resources on Google Colab for student access.

For example, to model a simple flip-flop, the prompt is:

Prompt

Please create an interactive FlipFlop Type D, where the user has a clock and data check-box switches, and an output color LED. Using svgwrite and iwidgets in Google Colab.

Using claude.ai [18] the response is:

Response - Claude 3.5

This Python script:

```
import svgwrite
from IPython.display import display, HTML
import ipywidgets as widgets
from IPython.display import clear_output
class DFlipFlop:
    def __init__(self):
        self.state = 0  # Output Q
        self.prev_clock = 0

    def update(self, clock, data):
        # Update state on rising edge of clock
        if clock == 1 and self.prev_clock == 0:
            self.state = data
        self.prev_clock = clock
        return self.state
... SNIPPETS ...
# Connect widgets
clock_widget.observe(update_display, 'value')
data_widget.observe(update_display, 'value')
# Initial display
update_display()
# Create layout
widgets_box = widgets.HBox([clock_widget, data_widget])
display(widgets_box)
display(output_widget)
```

This code creates an interactive D Flip-Flop with the following features:
Two checkbox widgets for Clock and Data inputs An SVG visualization showing:
The D Flip-Flop symbol Input lines with colored indicators for Clock and Data states An output LED that changes color based on the Q output
The D Flip-Flop logic updates on the rising edge of the clock
To use this code:
First, install the required packages if you haven't already:
- pip install svgwrite ipywidgets
Then, copy and run the code above in a Colab cell.
The D Flip-Flop will update according to these rules: - The output (Q) only changes when there's a rising edge on the clock (transition from 0 to 1) - When the clock rises, Q takes the value of the Data input - The output holds its state when the clock is low or high (no change on falling edge)

You can interact with it by: - Toggling the Data checkbox to set the input value - Toggling the Clock checkbox to trigger updates - Watching the output LED change color based on the state (green for 1, red for 0)

Figure 1 shows the interactive flip-flop simulation. The code from the prompt includes source

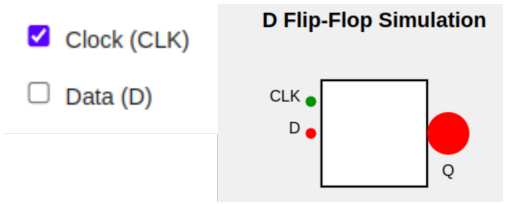


Figure 1: An interactive Flip Flop Type D simulation.

code in Python and has installation instructions. In other words, creating a simple demonstration is simple with a little learning effort with the LLM, Colab, and the additional technology.

4 Case Studies

This section provides specific examples of the resources we have created to help teach computer architecture and digital system design using Verilog HDL. In each case, we provide details of the LLM prompt and show what is generated by the LLM. However, similar to the above, we don't include the LLM response details and link to our cases later so that educators can leverage our examples to help them create their own.

4.1 Decoder Simulation

In this case, we illustrate a 3:8 decoder combinational circuit to show a simple simulation of a digital logic component using the "svgwrite" library. We used the following prompt to generate a decoder simulation:

Prompt

"Write Python code to draw an SVG box for a decoder with 3-bit input (A2, A1, A0) to 8-bit (one-hot) output (O0, O1, ..., O7), using widgets to set A2, A1, A0 that draw which output is active in Google Colab."

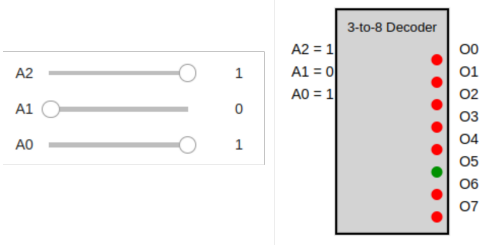


Figure 2: An interactive 3:8 decoder simulation.

The LLM generated Python code that creates an interactive SVG visualization of a 3-to-8 decoder using Ipython library "iwidgets" as shown in Figure 2. This allows the learner to interact with the component to observe how the inputs impact the outputs.

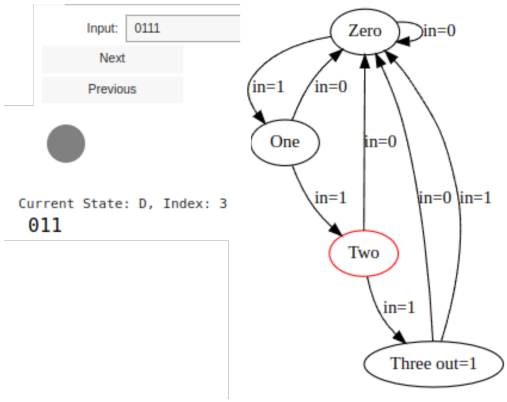
Prompt

”Give the state diagram in graphviz python

```
fsm.node('Z','Zero')
...
fsm.edge('Z','Z','in=0')
...
```

That has one binary input ”in” and one output ”out”. Create an interactive simulator to show step-by-step the highlighted state, show an input sequence as a field in the widget, highlight the current bit in the input, use a ”Next” button in the widget to go to the next state, and one to get the previous state.”

(a) Prompt Description

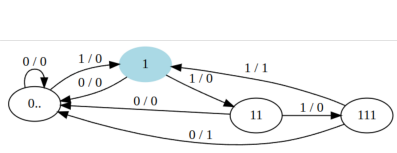


(b) An interactive state diagram simulator

4.2 Finite State Machine

This finite state machine (FSM) example features a state diagram created using the Python Graphviz library. The code was generated with the help of Microsoft’s Copilot LLM. Figure 3a shows the prompt used to specify the state diagram. The LLM successfully creates a simulator that highlights the current state in red, provides an input field for entering a bit sequence, and indicates the current position within the sequence. The simulation, controlled by the Next and Previous buttons, is illustrated in Figure 3b.

A prompt should provide clear and precise instructions [19], using delimiters such as quotation marks or braces to separate the instructions from examples or excerpts to be improved. One popular strategy in this vane is the “Chain-of-Thought” (CoT) prompting approach [20], where the request prompt solves a complex problem but is broken down into a sequence of smaller steps, where the LLM provides reasoning that explains each step of the solution.



Input String:
1011101

Output String:
0

Step
Reset
Input String: 1011101
Set Input

FSM Table:

Address	State	Input	Next State	Output
000	00	0	00	0
001	00	1	01	0
010	01	0	00	0
011	01	1	10	0
100	10	0	00	0
101	10	1	11	0
110	11	0	00	1
111	11	1	01	1

Figure 4: An interactive state machine simulator: Diagram and Table.

By applying a short sequence of interactions, creating a state machine simulator that simultaneously displays the state diagram and its implementation as a table is possible. The simulator allows for configurable inputs and supports step-by-step execution, with the current state highlighted during the process as shown in Figure 4.

4.3 RISC-V Emulator

LLMs are able to create an interactive RISC-V assembly emulator with a few interactive prompts. For example, we start with the following prompt to generate a RISC-V emulator:

Prompt

"Please create a Python RISC-V Assembly simulator for a subset with a text editor window, a sample program, use a sub-set of RISC-V instructions: lw, sw, add, addi, and, or, beq, bne, sub, for example. The user could initialize the memory, and show the memory and the register bank after the execution or even step by step. Using iwidget."

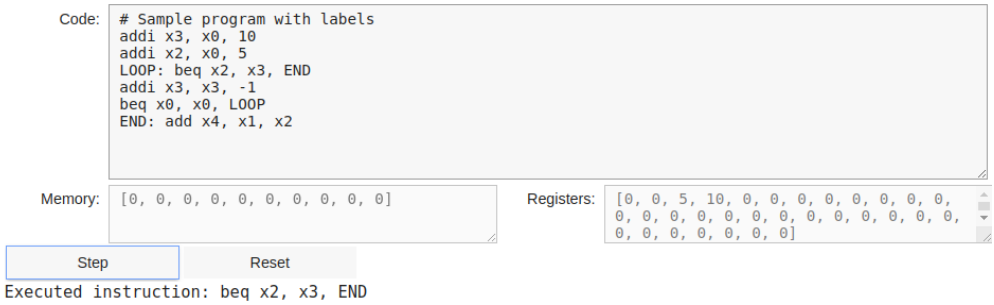


Figure 5: An interactive RISC-V emulator.

This example was generated with the free Chatgpt 4.0 mini OpenAI. The response included 169 lines of Python code. However, there is an error when handling commas in the instruction parser. Next, we follow up with the LLM, asking it to add labels to branch instructions, and within three progressing prompts, we can create a functional RISC-V emulator as shown in Figure 5.

This emulator simulates a RISC-V code step-by-step, showing the memory and register banking values. The students can edit the Python code to add more instructions and create a more sophisticated emulator. Similarly, students can take the code into the prompt and work with an LLM to make the sophisticated emulator.

4.4 Cache Emulator

A cache emulator serves as an example to demonstrate the ability of LLMs to create interactive interfaces and emulators for topics commonly covered in computer architecture courses. The following prompt can be used to generate a configurable direct-mapped cache emulator:

Prompt

”Please create a cache emulator for direct mapping using ipywidgets in Python for Google Colab. The emulator should allow the user to specify the cache size as a power of two, selected from a dropdown menu, with an additional dropdown to choose the unit (B, KB, or MB). The user should also provide the memory size, which must be a power of two, with units selected from KB or MB, and the block size, which must also be a power of two. Include a button to create the cache and display the calculated tag, line, and block address sizes in bits. Additionally, include a text area where users can input a series of memory addresses, separated by commas, to read from. Provide a button to submit a single memory address and display the resulting cache contents, including the tag and data. Assume that the main memory i contains data corresponding to its address.”

This example was generated using the free version of ChatGPT 4.0 by OpenAI. The code consists of 166 lines. Figure 6 illustrates the interface where students can specify cache parameters and provide a sequence of address accesses. The emulator demonstrates how the addresses are processed, tracks hit/miss occurrences, and displays the final cache state in binary, decimal, or hexadecimal format.

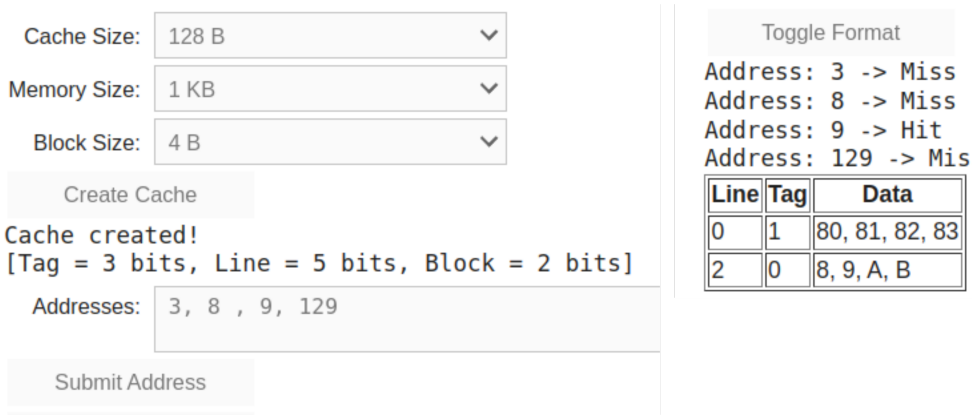


Figure 6: An interactive direct mapping cache emulator.

5 Results and Discussion

Our case studies for LLM-generated interactive code for digital system components reveal several key findings:

- Speed of Development: LLMs significantly reduce the time required to create initial prototypes of interactive simulations.
- Code Quality: The generated code was generally good quality, requiring minimal refinement for most basic components.
- Limitations: More complex systems, such as multi-cache protocols, required additional human intervention and expertise to implement correctly.

Note, however, that at the undergraduate level, in digital systems and computer architecture, most of the systems and ideas being learned are toys themselves, and the LLM's capability to create visualizations of these simplistic systems is adequate.

We also observed that the effectiveness of using LLMs for this purpose depends heavily on the quality of the prompts. Clear, specific prompts that include desired functionality and output format yielded the best results, and using CoT prompts significantly improves the quality of the results. Still, just as in the education of engineering undergraduates (who are using these LLM tools), one of the key ideas is that the LLM as a co-pilot needs to be reviewed by humans, and blindly trusting the tool tends to lead to error.

Based on our experiences, we propose the following guidelines for instructors looking to leverage LLMs in creating educational resources:

1. Start with Simple Components: Begin by generating code for basic digital components to familiarize yourself with the LLM's capabilities.
2. Treat prompting like you treated coding. The Integrated Developer Interface with the LLM should be similar to the Jupyter Notebook, where each prompt is a text file, and the response should be copied into a text file such that the process becomes iterative. The native cloud and the LLM chats (the IDE of sorts) are best maintained locally.
3. Iterate on Prompts: Refine your prompts based on the initial outputs to achieve desired results.
4. Verify and Test: Always verify the correctness of generated code and test thoroughly before deploying to students.
5. Combine with Existing Tools: Integrate LLM-generated code with established tools like Gradio for enhanced interactivity.
6. Encourage Critical Thinking: Use LLM-generated resources as a starting point for students to modify and expand upon.

6 Conclusion and Future Work

This paper demonstrates the potential of using LLMs in conjunction with Google Colab and Gradio to create interactive online resources for digital systems and computer architecture education. Our approach significantly reduces the time and effort required for instructors to develop engaging learning materials while maintaining educational quality. Our examples are maintained at: `https://github.com/lesc-ufv/GradioColab/example1` for other educators to access and iterate on.

Future work will focus on expanding the range of components and systems that can be effectively generated using LLMs and developing more sophisticated prompt engineering techniques.

7 Acknowledgements

We acknowledge the use of Claude 3.5 (https://claude.ai/, Accessed August-December 2024) to improve this document's organization and academic writing. We prompted the LLM tool, among

others in this paper, as referenced with various ideas and used generated results as starting points for aspects of our writing, noting that the ideas in the prompts were our own and that the authors edited and checked responses. We also acknowledge the use of Grammarly as a tool to improve our writing. No part of this document was written by an AI tool alone.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] R. Ferreira, M. Canesche, P. Jamieson, O. P. V. Neto, and J. A. Nacif, "Examples and tutorials on using google colab and gradio to create online interactive student-learning modules," *Computer Applications in Engineering Education*, p. e22729, 2024.

[3] M. Beg, J. Taka, T. Kluyver, A. Konovalov, M. Ragan-Kelley, N. M. Thiéry, and H. Fangohr, "Using jupyter for reproducible scientific workflows," *Computing in Science & Engineering*, vol. 23, no. 2, pp. 36–46, 2021.

[4] A. Rule, A. Birmingham, C. Zuniga, I. Altintas, S.-C. Huang, R. Knight, N. Moshiri, M. H. Nguyen, S. B. Rosenthal, F. Pérez, and P. W. Rose, "Ten simple rules for writing and sharing computational analyses in jupyter notebooks," *PLOS Computational Biology*, vol. 15, no. 7, pp. 1–8, 07 2019. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1007007

[5] A. Davies, F. Hooley, P. Causey-Freeman, I. Eleftheriou, and G. Moulton, "Using interactive digital notebooks for bioscience and informatics education," *PLoS computational biology*, vol. 16, no. 11, p. e1008326, 2020.

[6] S. Lau, I. Drosos, J. M. Markel, and P. J. Guo, "The design space of computational notebooks: An analysis of 60 systems in academia and industry," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2020, pp. 1–11.

[7] R. De Smet, S. Thielemans, J. Lemeire, A. Braeken, and K. Steenhaut, "Educational software-as-a-service based on jupyterhub and nbgrader running on kubernetes," in *2022 IEEE 9th International Conference on e-Learning in Industrial Electronics (ICELIE)*. IEEE, 2022, pp. 1–6.

[8] C. D. González-Carrillo, F. Restrepo-Calle, J. J. Ramírez-Echeverry, and F. A. González, "Automatic grading tool for jupyter notebooks in artificial intelligence courses," *Sustainability*, vol. 13, no. 21, p. 12050, 2021.

[9] K. Edwards, C. Scalisi, J. DeMars-Smith, and K. Lee, "Google colab for teaching cs and ml," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, 2024, pp. 1925–1925.

[10] B. Kim and G. Henke, "Easy-to-use cloud computing for teaching data science," *Journal of Statistics and Data Science Education*, vol. 29, no. sup1, pp. S103–S111, 2021.

[11] T. Carneiro, R. V. M. Da Nóbrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. Reboucas Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61 677–61 685, 2018.

[12] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, and J. Zou, "Gradio: Hassle-free sharing and testing of ml models in the wild," *arXiv preprint arXiv:1906.02569*, 2019.

[13] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.

[14] H. Xu, W. Gan, Z. Qi, J. Wu, and P. S. Yu, "Large language models for education: A survey," *arXiv preprint arXiv:2405.13001*, 2024.

[15] I. Lee and B. Perret, "Preparing high school teachers to integrate ai methods into stem classrooms," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, 2022, pp. 12 783–12 791.

[16] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "What's wrong with computational notebooks? pain points, needs, and design opportunities," in *Proceedings of the 2020 CHI conference on human factors in computing systems*, 2020, pp. 1–12.

[17] C. Reparaz, M. Aznárez-Sanado, and G. Mendoza, "Self-regulation of learning and mooc retention," *Computers in Human Behavior*, vol. 111, p. 106423, 2020.

[18] Anthropic, "Claude (ai assistant)," https://claude.ai, 2024, accessed December 14, 2024.

[19] X. Xu, C. Tao, T. Shen, C. Xu, H. Xu, G. Long, and J.-g. Lou, "Re-reading improves reasoning in language models," *arXiv preprint arXiv:2309.06275*, 2023.

[20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.