Benchmarking of LLM Based Generative AI for CSE Undergraduate Curriculum

Garrett Goodman[1], Suman Bhunia[1], Peter Jamieson[2]
goodmag@miamioh.edu, bhunias@miamioh.edu, jamiespa@miamioh.edu
[1]Department of Computer Science and Software Engineering
[2]Department of Electrical and Computer Engineering
Miami University, Oxford, Ohio, USA 45056

Abstract

In the past few years, rapid research of Natural Language Processing (NLP) has enabled Artificial Intelligence (AI) tools to answer a wide array of questions with reasonable accuracy by an AI tool. Furthermore, the rapid inclusion of this technology into Computer Science practice requires investigating how it can affect our pedagogical strategies. With the rise of Large Language Model (LLM) based Generative AI tools (Gen-AI), they have shown tremendous efficiency in solving Computer Science-related questions found in common course curricula. Thus, this paper presents a novel investigation into the performance of these Gen-AI tools within the context of two courses in the Computer Science and Software Engineering (CSE) undergraduate curricula at Miami University.

We introduce a systematic benchmarking methodology to measure LLM performance against course material found in CSE 174 (Fundamentals of Problem Solving and Programming) and CSE 274 (Data Abstraction and Data Structures). This includes exams, lab assignments, and projects. Our benchmarking results reveal the strengths and limitations of these Gen-AI tools in Computer Science educational tasks, providing crucial insights for curriculum adaptation. Utilizing these results, we provide insight on how they might assist in the evolution of Computer Science education. Specifically, we highlight the areas where Gen-AI tools could enhance learning and where human skills remain explicitly required. This work continues the discussion on how AI capabilities can be integrated into the academic setting. Such benchmarking efforts are essential to ensure that our students receive the knowledge they require to move from the Computer Science academic setting into the professional setting, which largely utilizes such Gen-AI tools. Thus, this will prepare the next generation of computer scientists to be effective AI-augmented professionals.

1 Introduction

Artificial Intelligence (AI) introduces challenges to the traditional educational landscape. Educators need to teach students crucial concepts for their field but also need to prepare students on how to use relevant tools for their future in the workforce. One such tool is the emergence of Large Language Models (LLMs) like ChatGPT-4, Copilot, and Claude, which show a significant leap in AI capabilities. Such technologies require educators to reevaluate teaching methodologies, curriculum design, and how we prepare students for the workforce [1].

The first step to utilize the aforementioned LLMs in our classes is to apply a benchmark to the existing curriculum with respect to LLMs. Specifically, we need to understand how LLMs perform in the context of LLM "being a student." This understanding can provide insight into AI's strengths and weaknesses in various assessments with which we currently evaluate student learning. Similarly, we should consider the purpose of specific student assessments and why an AI tool *might not be* beneficial. In contrast, the same can be said about why an AI tool *might be* beneficial. By benchmarking LLM performance on assessments within our Computer Science and Software Engineering (CSE) curriculum, we can identify where AI excels and falls short and how it might impact future iterations of the educational landscape.

This study aims to benchmark LLM performance on assessments of two courses within the CSE undergraduate curriculum. We methodically evaluated how one of the most prominent LLMs, ChatGPT-4, performs on all relevant assessments within the subset of chosen core courses. Our preliminary results provide valuable insight into the prospects of AI in Computer Science and Software Engineering educational tasks. Our findings suggest possible areas of improvement within our curriculum that need to both leverage AI capabilities and to teach the necessary prerequisite concepts for our students to fully utilize AI for the workforce.

The contributions of this work are as follows:

1. An applied methodology for benchmarking LLM performance against CSE assessments.
2. Initial performance data of an advanced LLM on the assessments of two CSE core courses.
3. Provide insights into potential curriculum adjustments in light of AI advancements.

The remainder of this paper is organized as follows. Section 2 provides background on AI models, curriculum analysis, and design. Section 3 introduces our methodology to classify our LLM prompts. In Section 4, we present an overview of our CSE curriculum and note which courses have been evaluated in this study. Section 5 presents our benchmarking results. We discuss the implications of our results in Section 6 and conclude with future work in Section 7.

2 Background - AI Models, Curriculum Analysis, and Curriculum Design

To understand the context of the work in this paper, we provide a brief background of the utilized topics. First, in Section 2.1, we provide a brief overview of what an AI model is concerning LLMs. Next, in Section 2.2, we briefly define what learning outcomes are and how they relate to

course design. Finally, in Section 2.3, we briefly discuss the overall curriculum design and analysis of our CSE curriculum.

2.1 An Overview of AI and LLMs

An LLM is an AI model trained on large datasets of text and designed to perform natural language processing tasks involving understanding and generating human language [2]. In the context of Computer Science tasks, they can generate code in various programming languages [3]. As for this work, we use Agrawal, Gans, and Goldfarb's [1] model of AI as "prediction machines." LLMs are prediction machines that use the intersection between computational resources (computers, smartphones, etc.) and the given data and datasets to train them continuously. Thus, LLMs are extraordinarily effective at responding to various queries.

2.2 Learning Objectives and Designing Courses

A *Learning Objective* (LO), otherwise known as a Learning Outcome, is an educational goal for a student such that they can perform the objective after learning it. From Bloom's Taxonomy [4], an LO belongs in the hierarchy of cognitive processes that range from "lower-order thinking skills", such as recalling and classifying information, to "higher-order thinking skills", such as creating or planning. As for course design, "Understanding by Design" [5] provides a methodology for designing courses where the designer starts from LOs, documents how to assess students on individual LOs, then creates class activities and assignments to prepare them for said assessment. Thus, it shows the student is competent in any given LO.

2.3 Curriculum Design and Analysis

Curriculum can be defined as a way of ordering content and purposes for teaching and learning in schools [6]. We expand on this definition and apply it specifically to the CSE curriculum at the undergraduate level, where the goal of the curriculum is to develop individuals who can professionally perform Computer Science or Software Engineering tasks. It is worth noting that the CSE department at Miami University contains multiple undergraduate degrees. Namely, Computer Science (Bachelor of Arts and Bachelor of Science), Software Engineering, and Cyber Security. Fortunately, all our majors share the same beginning core courses and diverge into their specialties around the end of their second year. Computer Science focuses on the theoretical foundations of computing, Software Engineering focuses on the design, development, and testing of software, and Cyber Security focuses on local and remote systems, networks, and data protection from cyber threats. Even though each major diverges in the LOs we teach, they all require the same foundation of problem-solving and programming, object-oriented programming, and data structures.

As for the design of these curricula, we start with aims [7] or objectives [8] that are determined by accrediting agencies, such as the Accreditation Board for Engineering and Technology (ABET), as well as the needs of companies and practitioners of the respective fields. Following, we determine a period of study and progression of courses the student takes in the curriculum. Examining a case study for curriculum design provides great insight into the process, as shown in

Patil and Ghatage's [9].

Designing an effective curriculum has many challenges, some of which are not immediately obvious. First, Molontay *et. al.* [10] utilized directed graphs to analyze curriculum prerequisite chains. Similarly, Padhye *et. al.* [11] used data analytic techniques to determine curriculum design patterns to analyze the complexity of a curriculum. To assist in curriculum design, Heileman *et. al.* [12] used curricular analytic techniques to examine how certain curriculum reforms improve graduation rates. Even with these curriculum reform improvements, aspects of the curriculum were never intentionally created. These aspects are called Hidden Curriculum, from which the reader can learn more in works such as Villanueva *et. al.* [13].

3 LLM Benchmarking Methodology

This work utilizes Jamieson's LLM Prompt Taxonomy [14]. Based on existing research, this taxonomy follows a three-level classification system for LLM prompts. The taxonomy is as follows:

1. **LLM Agent Architecture** [15]
    - Prompt/Response (PR): Simple input-output model
    - Multi-Prompt/Automated Response (AR): Generated response gives additional prompts
    - Human in the LLM Loop (HiLL): Human guides task based on LLM responses
    - Retrieval Augmented Generation (RAG) systems [16, 17] with memory and retrieval capabilities
2. **LLM Reasoning of Thought**
    - Nothing of Thought (NoT): Baseline without reflection
    - Self-improved of Thought (SoT): Reflects and improves on the given prompt [18, 19]
    - Chain of Thought (CoT): Linear generations with reasoning [20]
    - Tree of Thought (ToT): Branching generation paths for alternatives [21]
    - Graph of Thought (GoT): Network of generated ideas with dependencies [22]
    - Program of Thought (PoT): Separates inference from computation [23]
3. **LLM Shot Type** [24]
    - zero-shot: Prediction without specific examples [25]
    - few-shot: Prediction with examples [26]
    - multi-shot: Multiple separate actions; can combine with other shot types [27, 28]

Thus, Jamieson's LLM Prompt Taxonomy is represented as $\{$*architecture; reasoning; shot-type*$\}$. Utilizing this taxonomy, a prompt can be classified based on the LLM's complexity and capabilities. One generally needs to increase one or more of the three taxonomy pieces to achieve a more advanced prompt. That is, provide a more complex architecture, higher-level reasoning of thought, and/or a few-shot or multi-shot prompt approach. We use this taxonomy to record and analyze the prompts used in our course assessments.

To utilize this taxonomy in the work for this paper, we used a chatbot, namely ChatGPT-4, and prompted it to complete the assessments of a course. Each course will be evaluated independently. The methodology for using this taxonomy is as follows:

1. Using Jamieson's LLM Prompt Taxonomy, we prompt ChatGPT-4 via the web interface[1] with the zero-shot shot type to complete the given assessment. We chose this shot type as ChatGPT-4 can understand PDFs in full. Thus, the prompt "Solve the attached assignment." is used, and we attached the lab instructions and other relevant files. Each prompt will have the taxonomy recorded.
2. Execute the prompts and record the generated results.
3. Using the generated results, the course instructor will grade the assessment as if a student-produced it. A letter grade or numerical value should be assigned to the assessment. We acknowledge that this approach has bias as the instructor knows that ChatGPT-4 generated the result for the assessment.
4. After all assessments have been prompted and graded, they are aggregated, and a final grade for the course is assigned to ChatGPT-4. Thus, this allows us to make claims such as "How well is an LLM at completing a course?" or with more specifics such as "How well does an LLM write code to traverse a binary tree recursively?".
5. All assessments given to ChatGPT-4 will have an "Assumed LLM Score" given to it by the instructor. This assumed score is what the instructor believes the LLM's grade will be on a given assessment before it is generated. These assumptions, while speculative, can provide an idea of what could be achieved if more advanced aspects of the taxonomy are used.

Lastly, we note that not all assessments, such as class participation, can be given to an LLM. Thus, to provide a final grade, we give full credit as we would any student who fully participates in the course.

4 Miami University's CSE Curriculum

Miami University's CSE Curriculum, as previously discussed, contains three undergraduate majors: Computer Science (Bachelor of Arts and Bachelor of Science), Software Engineering, and Cyber Security. All three disciplines share the same year and a half of core courses. Thus, for the sake of brevity, we will discuss only the Computer Science track. As seen in Figure 1, the three introductory courses taken linearly are CSE 174 - Fundamentals of Problem Solving and Programming, CSE 271 - Object-Oriented Programming, and CSE 274 - Data Abstraction and Data Structures. We applied Jamieson's LLM Prompt Taxonomy to CSE 174 and CSE 274 as all CSE students take these courses and are fundamental for every discipline in the department. Moving past CSE 274, the curriculum diverges depending on the chosen degree. The figure also excludes prerequisites from mathematics as well as electives. Note that our curricula were designed well before AI and LLMs were prominent. Thus, the following questions are posed:

1. How do LLMs interact with our courses?
2. How and where must our curriculum change for these readily available LLMs?

We aim to provide the beginnings of an answer to both of these questions. You could expand on the second question by discussing the ordering of courses, the topics being taught within the

[1] `https://chatgpt.com/`

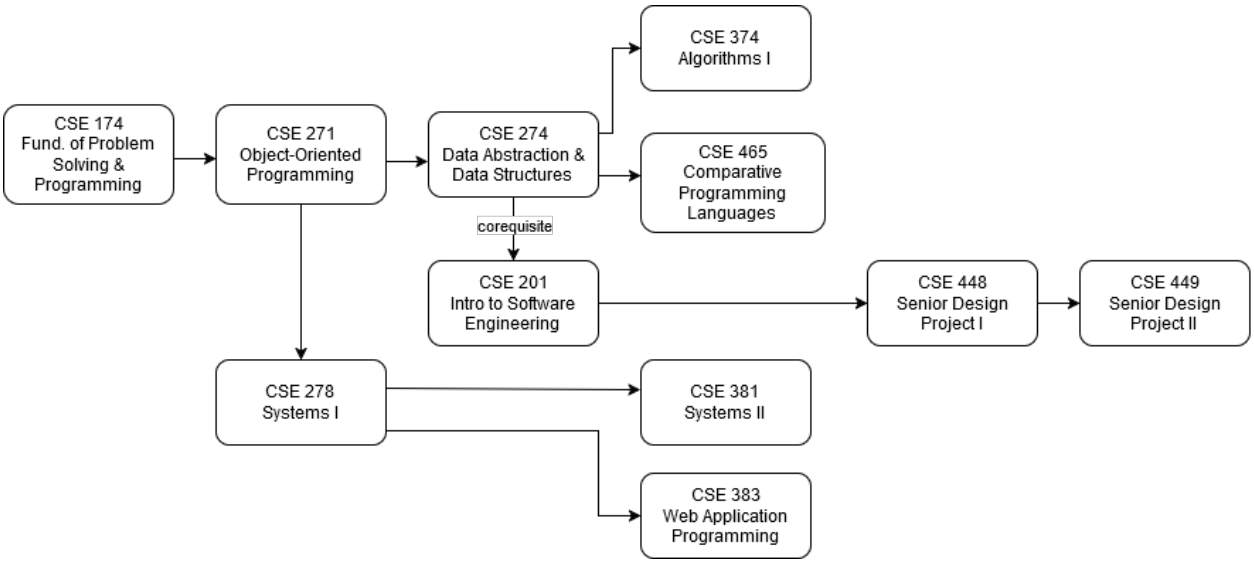

Figure 1: CSE curriculum flowchart of core courses.

courses, etc. However, these questions are outside the scope of this work and will be examined in future work.

4.1 CSE 174 - Fundamentals of Problem Solving and Programming

CSE 174, **Fundamentals of Problem Solving and Programming**, focuses on the fundamental programming principles that nearly all programming languages utilize and how these principles can be applied to solve practical problems. The curriculum uses Python to cover variable creation, file input and output, control structures, loops, functions, recursion, lists, and basic searching and sorting algorithms.

The primary goal of this course is to teach and reinforce programming and problem-solving principles to students with no previous programming knowledge. The LOs emphasize a progression through multiple levels of Bloom's taxonomy [4]. First, at the **Understanding** level, students must explain how specific principles work in order to apply them effectively. Next, at the **Applying** level, students implement the code of these principles to solve introductory problems. Then, at the **Analyzing** level, students must examine their output to determine if it matches the expected output of the test cases as well as analyze why either it did or did not. Finally, at the **Creating** level, students must combine multiple principles, such as nested for loops within a function, to address real-world problems. Thus, this course's assessments nourish critical thinking, problem-solving, and coding principles.

Assessments in this course are all hands-on and are designed to coincide with the topic being presented immediately. Furthermore, each Lab and Project has a premise, such as "Creating an Online Business Customer Management Program," to provide real-world context for a particular programming principle. The students are tasked with implementing the principle, such as nested loops, in a different and slightly more involved way than presented in class. Thus, additional time is required to walk through the essential problem-solving steps to produce the correct output. To

showcase this, below is a sample lab assignment, shortened for brevity, that would be given to students in the course.

Sample Lab 10 Assignment

> You decided to volunteer at a local homeless shelter and food pantry. The shelter offers greatly discounted foods and supplies to individuals in need. The shelter also knows you are a programmer and asked if you could help them create a program to assist with their order data.
>
> Using the example input text files list1.txt and list2.txt, create a lab_10.py file, ask the user for a file name, read the text file line-by-line, calculate the total profit and average order profit, and display the output as shown in the example output. To do this, create and implement the following functions:
>
> ```
> def get_order_profits(data: list) -> list:
>
> def display(data: list) -> None:
>
> def get_average_profit(data: list) -> float:
>
> def get_total_profit(data: list) -> float:
> ```
>
> **Grading Rubric:** Total 100 points. The program prompts the user for a filename, reads the file, and handles a FileNotFoundError with a try-except block, 20 points. The program calculates the profit per order, 20 points. The program displays the output, 10 points. The program calculates the average and total profit of all orders, 10 points. The program closes the file, 10 points. The program uses proper variable names, follows style guidelines, and has an appropriate amount of comments, 30 points.

The lab instruction is associated with an input text file of data, as given below.

> 12342 Pen 10 .25
> 12343 Book 10 1.35
> 12353 Mouse 1 12.0
> 12359 Keyboard 1 43.99
> 12313 Chips 2 2.89
> 12314 Icecream 5 1.30

Below is a sample of the code that ChatGPT-4 generated for the **get_order_profits()** function.

```
def get_order_profits(order_list: list) -> list:
    """
    Extracts the amount and price of each order, calculates the profit, and
    ↪  returns a list of profits.
    :param order_list: List of all order data.
    :return: List of individual order profits.
    """
    profits = []
    for i in range(2, len(order_list), 4):
```

```
        amount = int(order_list[i])
        price = float(order_list[i + 1])
        profits.append(amount * price)
    return profits
```

4.2 CSE 274 - Data Abstraction and Data Structures

CSE 274, **Data Abstraction and Data Structures**, focuses on the design, implementation, and analysis of Abstract Data Types (ADTs) using object-oriented programming principles. The curriculum covers a range of fundamental data structures, such as lists, stacks, queues, tables, trees, and graphs, along with their sequential and linked storage representations. Additionally, the course emphasizes recursion, sorting and searching algorithms, and the analysis of algorithm complexity, equipping students with theoretical and practical understanding.

The primary goal of this course is to enable students to select and implement appropriate data structures and algorithms to solve computational problems effectively. The LOs emphasize a progression through multiple levels of Bloom's taxonomy [4]. First, at the **Understanding** level, students articulate the purpose, behavior, and applications of key data structures and ADTs. Next, at the **Applying** level, they implement these structures and algorithms using object-oriented programming principles in Java. The course advances to the **Analyzing** level as students evaluate the performance and suitability of various data structures for specific problems, considering algorithm complexity. Finally, at the **Creating** level, students design and implement innovative solutions by combining data structures and algorithms to address real-world challenges. This comprehensive approach cultivates critical thinking, problem-solving, and coding proficiency.

Assessments in this course are hands-on and designed to build practical coding skills. Students work on implementing data structures iteratively and recursively, using array-based and linked-node-based approaches. They also develop traversal algorithms and leverage Java's standard collection libraries, ensuring a well-rounded experience in data structure implementation and application. To showcase this, below is a sample lab assignment, shortened for brevity, that would be given to students in the course.

Sample Lab 6 Assignment

> In this lab, you will define a stack and then use it to complete multiple tasks (methods) defined in the StackApplications.java class. You can then use StackApplicationTest to test all your methods using JUnit tests.
>
> Write a class named Stack, which implements the StackInterface using a singly linked list. You cannot have any other methods in the Stack class. Also, you can have only one instance variable referencing the first node named "topNode".
>
> Implement the incomplete methods in the StackApplications class so that your code passes all JUnit tests.
>
> **Grading Rubric:** Total 100 points. Stack class is worth 30 points (each method 6 points). Each method in StackApplications is worth approximately 10 points. There are seven methods, total 70 points.

The lab instruction is associated with a Java file skeleton, as given below.

```
public class StackApplications {
  /**
   * Computes the sum of all the numbers in the stack.
   */
  public static int sum(Stack<Integer> data) {
    return 0;
  }

  /**
   * Computes the sum of all the numbers in the stack. However, if two or more
   * numbers in a row are equal, only add one of them. So, for example, if the
   * stack contained 4, 1, 2, 2, 7, 2, 8, 8, 8, 4, then the numbers that would
   * be added would be 4 + 1 + 2 + 7 + 2 + 8 + 4 = 28. Here 4 is the top of
   ↪  the stack.
   */
  public static int sumSkipDuplicates(Stack<Integer> data) {
    return 0;
  }

  /**
   * Puts all of the digits of an integer into a stack, with the first
   * digit of the integer at the bottom of the stack and the last digit
   * of the integer at the top of the stack.
   */
  public static Stack<Integer> integerToStack(int n) {
    return null;
  }

  /**
   * Reverses a given stack, so that the top of the stack becomes the bottom
   * and the bottom becomes the top.
   */
  public static void reverseStack(Stack<Integer> s) {
  }
}
```

Below is a sample of the code that ChatGPT-4 generated for reversing a Stack.

```
    /**
     * Reverses a given stack, so that the top of the stack becomes the bottom
     ↪  and
     * the bottom becomes the top.
     */
    public static void reverseStack(Stack<Integer> s) {
        Stack<Integer> tempStack1 = new Stack<>();
        Stack<Integer> tempStack2 = new Stack<>();

        while (!s.isEmpty()) {
            tempStack1.push(s.pop());
        }

        while (!tempStack1.isEmpty()) {
            tempStack2.push(tempStack1.pop());
```

```
}

while (!tempStack2.isEmpty()) {
    s.push(tempStack2.pop());
}
```

5 Benchmarking Results for our Selected Courses

After implementing the methodology described in Section 3, we independently showcase the results for the two courses. Section 5.1 details the results of CSE 174's curriculum, and Section 5.2 details the results of CSE 274's curriculum. Note that the two courses employ a variety of assessments to evaluate students' understanding and skills. Table 1 showcases these for CSE 174, and Table 2 showcases these for CSE 274. For both tables, Columns 1 and 2 show each category's course assessment and the corresponding number of assessments, respectively. Column 3 shows the percentage weight used in the final grade calculation for each category. Column 4 shows ChatGPT-4's evaluated and aggregated score. Finally, column 5 shows the prompt's level in Jamieson's LLM Prompt Taxonomy discussed previously.

For both courses, we assumed a score of 100% for each assessment as the LOs of these courses are at the front of the overall curriculum, are well-studied, and do not require any external equipment for any assessment. Finally, in both tables, the **In-class Participation** and **Career Assignments** (for CSE 174 only) assessments are given a score of 100% as these components require real-time engagement not directly replicable by LLMs. Students are given full credit if they submit them, regardless of their correctness.

5.1 Benchmarking CSE 174

Table 1 provides the breakdown of grades for different CSE 174 assessments. ChatGPT-4 performed quite well in the **Final Exam** and both **Midterm Exam 1** and **Midterm Exam 2**, scoring 86%, 91%, and 88%, respectively. Then, for **Labs**, ChatGPT-4 achieved a score of 84.90%. Next, in **Quizzes**, ChatGPT-4 scored 85.71%. Finally, for **Projects**, ChatGPT-4 scored 91%. The tool demonstrated consistently above-average performance across various course assessments, achieving an overall score of 88.56%, equivalent to a letter grade of $B+$.

5.2 Benchmarking CSE 274

Table 2 provides the breakdown of grades for different CSE 274 assessments. ChatGPT-4 excelled in both the **Final Exam** and **Midterm Exam**, scoring 90% in each. Then, for **Labs**, ChatGPT-4 achieved a score of 97.2%. Next, in **Quizzes**, ChatGPT-4 scored 92.14%. Finally, the highest performance was observed in **Homework Projects** scoring 98.40%. The tool consistently demonstrated high performance across various course components, achieving an overall score of 93%, equivalent to a letter grade of $A-$.

Table 1: Performance of ChatGPT-4 in CSE 174.

Course Assessment	Assessment Count	Grade Weight	ChatGPT-4 Score	Jamieson's LLM Prompt Taxonomy
Final Exam	1	20%	86%	PR; CoT; zero-shot
Midterm Exam 1	1	15%	91%	PR: CoT; zero-shot
Midterm Exam 2	1	15%	88%	PR; CoT; zero-shot
Labs	10	15%	84.90%	PR: CoT; zero-shot
Quizzes	13	10%	85.71%	PR: CoT; zero-shot
Projects	4	20%	91%	PR: CoT; zero-shot
In-class Participation	N/A	2.5%	(assumed) 100%	N/A
Career Assignments	N/A	2.5%	(assumed) 100%	N/A
Final Score		**88.56% (Letter Grade B+)**		

Table 2: Performance of ChatGPT-4 in CSE 274.

Course Assessment	Assessment Count	Grade Weight	ChatGPT-4 Score	Jamieson's LLM Prompt Taxonomy
Final Exam	1	25%	90%	PR; CoT; zero-shot
Midterm Exam	1	25%	90%	PR: CoT; zero-shot
Labs	11	15%	97.20%	PR: CoT; zero-shot
Quizzes	11	15%	92.14%	PR: CoT; zero-shot
Homework Projects	6	15%	98.40%	PR: CoT; zero-shot
In-class Participation	N/A	5%	(assumed) 100%	N/A
Final Score		**93% (Letter Grade A-)**		

6 Discussion

Examining our benchmarking results allows us to analyze the performance of ChatGPT-4 in our assessments and, therefore, is a core part of our CSE curriculum. Also, we aim to answer Question 1 posed in Section 4. First, for CSE 174, ChatGPT-4 obtained a final grade of 88.56%, which is a **B+**. The LLM did not score perfectly in any category in Table 1, which surprised us. After analyzing the generated results, a couple patterns emerged.

First, we noticed hallucinations if there were any images (jpg, png, etc.) within the pdf file showing the expected example output. That is, a hallucination is a generated response that is plausible but is either incorrect, not connected to the initial prompt, or both [29]. As CSE 174 is the first programming course many students take, we expect very explicit program outputs. The instructors act as the 'customers' while the students act as 'workers' for the company we hired. We hired them for the assessments, and we asked for specific output. When they deviate from that, they lose points as the instructor, i.e., the customer, is not getting what was asked for. This is in an attempt to connect the assessments to real-world use cases. Note, the amount of deducted points is only a few if some words do not match up in the output, for example. We note this as most assignments had the expected output in an image saved in the PDF. Thus, ChatGPT-4 hallucinated prompts for the output and lost points as reflected in the **Labs** and **Projects** categories.

Secondly, ChatGPT-4 occasionally generated incorrect mathematical equation results when quantitative tasks were posed, but not consistently, as shown in the **Quizzes** category. Overall, we found that ChatGPT-4 performed exceptionally well in generating code for assessments, and for understanding and answering more theoretical questions, both in CSE 174 and CSE 274. That means it possesses good analytical skills.

Moving to CSE 274, ChatGPT-4 performed slightly better than CSE 174 and achieved a final grade of 93%, which is an **A-**. Though again, as shown in Table 2, it did not score perfectly as hypothesized. It did, though, excel in both the Midterm Exam and Final Exam, for which it scored 90%, showcasing its strong ability to analyze, apply, and create solutions under theoretical and problem-solving contexts. Furthermore, as CSE 274 assessments are more involved than CSE 174 assessments and require a higher order of programming knowledge, ChatGPT-4 showed proficiency in implementing and debugging more complex code. This is reflected in the **Labs** and **Homework Projects** categories, highlighting ChatGPT-4's capability to execute complex programming tasks, implement algorithms, and demonstrate problem-solving at a granular level.

There are also some interesting use cases for ChatGPT-4 in CSE education, allowing us to construct an answer for Question 2 posed in Section 4. LLMs can play a pivotal role in assisting students with various assessments. For example, in CSE 274, students can use ChatGPT-4 to clarify concepts, generate code snippets for implementing data structures, or debug and optimize their solutions. These tools can also help students write JUnit tests or understand the trade-offs in algorithm complexity, fostering a deeper understanding of course concepts. Furthermore, the instructor can harness AI tools to augment learning outcomes by designing assignments that require critical analysis and customization of AI-generated outputs, ensuring students engage deeply with the material. In the case of CSE 174, tools like ChatGPT-4 can provide additional study material and short problems to practice key concepts like if-statements, loops, etc., with a correctly engineered prompt.

Conversely, the course level in the curriculum might also dictate the level of involvement students can or should have with using LLMs. For introductory programming courses such as CSE 174, students have no conception of a for loop, a try-except block, a function, etc. To fully utilize the capabilities of LLMs as a pair programmer or programming assistant, they first need to understand the fundamental concepts it generates. Without that, it is difficult to integrate what is generated into a larger program without creating bugs, thus failing test cases. The programs for our CSE 174 course are small enough that LLMs can generate the entire solution. However, the programs get significantly larger in later courses in the third and fourth years. Therefore, students will have to understand what the code that LLMs are generating is doing so that it can be smoothly integrated into these large programs. So, to answer the second question, the curriculum needs to sufficiently educate students on learning from the information generated from LLMs and not rely solely on what is produced. It is already standard practice in the industry for programmers to utilize LLMs to increase productivity. So, we as educators must integrate LLMs into our classes gradually throughout the curriculum so that students learn the material and how to effectively utilize LLMs.

Using ChatGPT as a tutor or troubleshooter for assessments can significantly enhance student learning by providing on-demand explanations, debugging assistance, and guidance in

understanding fundamental programming concepts, complex data structures, and algorithms. For example, students can use ChatGPT to clarify doubts about pieces of code, generate sample code, or troubleshoot errors in their assignments. This self-paced, interactive learning approach enables students to explore alternative solutions and deepen their understanding. However, instructors should guide students on the ethical and effective use of such tools, emphasizing that ChatGPT should supplement learning rather than replace critical thinking and original problem-solving. To ensure academic integrity, instructors can design assessments that require personalized and innovative application of concepts, making it harder to rely solely on AI-generated solutions. To reduce opportunities for misconduct, the course should emphasize in-class quizzes and exams rather than take-home assignments. In-class assessments are harder to manipulate and provide a better measure of individual understanding. Adding more weight to these in-class evaluations than take-home assignments will encourage students to focus on mastering the content.

As for using the generated responses of our assessments from ChatGPT-4, or similar LLMs, to better teach a topic is not yet clear. That is, the LLM's reasoning of thought for these introductory courses is effectively one-to-one, i.e., not as described in Section 3. The assessments are too linear at the beginning of our curriculum to warrant a more complex reasoning of thought. Perhaps later in the curriculum with more complex assessments, the generated assessment responses can give additional insight on how to present a topic. However, the instructor can adopt a balanced approach, teaching the main concepts in lectures and demonstrating their application through live programming examples. This approach helps students build a strong theoretical foundation and observe practical problem-solving techniques firsthand. AI tools can then be positioned as practice aids, allowing students to refine their understanding by debugging code, exploring alternative solutions, or clarifying doubts. By using these tools for practice, students can strengthen their grasp of data structures and programming techniques without bypassing the learning process.

7 Conclusions

In conclusion, we utilized Jamieson's LLM Prompt Taxonomy to benchmark two courses in Miami University's CSE curriculum. Specifically, we benchmarked CSE 174 - Fundamentals of Problem Solving and Programming and CSE 274 - Data Abstraction and Data Structures, two core courses at the beginning of our curriculum. Tables 1 and 2 summarize how well an LLM, specifically ChatGPT-4, performs in CSE 174 and CSE 274, respectively. From these results and by analyzing the generated outputs from ChatGPT-4, we aimed to answer two questions: "How do LLMs interact with our courses?" and "How and where must our curriculum change for these readily available LLMs?"

Our findings suggest that while LLMs perform exceptionally well over various assessments in both courses, they are not as perfect as expected. To answer the first question, LLMs can be viewed as above-average students, able to analyze the problem and generate solutions under theoretical and problem-solving contexts. However, it hallucinates when dealing with images embedded in PDF files and is inconsistent with mathematical operations. Thus, it achieved a final grade of **B+** and **A-** in CSE 174 and CSE 274, respectively. As for answering the second question, we acknowledge the need to incorporate LLMs gradually throughout the curriculum. Specifically

balancing their usage such that students effectively learn the presented concepts and how to interact with an LLM. For CSE students to utilize the generated code from an LLM, they must understand what that code does to integrate it into a larger program without causing problems.

Future work will have us benchmark later classes, such as CSE 374 - Algorithms 1 in the curriculum, as CSE 174 and CSE 274 are at the front of our curriculum. Then, we can compare the benchmarking results to make more concrete suggestions for CSE curriculum improvement.

8 Acknowledgments

Miami University's College of Engineering and Computing provided support for this project. This work was performed in tandem with the ECE department at Miami University under the guidance of Dr. Peter Jamieson, but with our differing CSE curriculum [30].

References

[1] A. Agrawal, J. Gans, and A. Goldfarb, *Prediction Machines, Updated and Expanded: The Simple Economics of Artificial Intelligence*. Harvard Business Press, 2022.

[2] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent Abilities of Large Language Models."

[3] T. Coignion, C. Quinton, and R. Rouvoy, "A Performance Study of LLM-Generated Code on Leetcode," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, 2024, pp. 79–89.

[4] L. W. Anderson, D. R. Krathwohl, and B. S. Bloom, *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon, 2001.

[5] G. P. Wiggins, J. McTighe, L. J. Kiernan, and F. Frost, *Understanding by design*. Association for Supervision and Curriculum Development Alexandria, VA, 1998.

[6] D. F. Walker, *Fundamentals of curriculum: Passion and professionalism*. Routledge, 2002.

[7] J. Prasad, A. Goswami, B. Kumbhani, C. Mishra, H. Tyagi, J. H. Jun, K. K. Choudhary, M. Kumar, N. James, V. R. S. Reddy *et al.*, "Engineering curriculum development based on education theories," *Current Science*, pp. 1829–1834, 2018.

[8] T. Dion and K. C. Bower, "Integrating Learning Outcomes Throughout the Civil Engineering Curriculum to Meet Site Engineering Prerequisite Needs," in *ASEE Southeast Section Annual Conference, Louisville, KY*, 2007.

[9] S. R. Patil and P. S. Ghatage, "Curriculum Development of an Engineering PG Program at an Autonomous Institute–A Case Study," *Journal of Engineering Education Transformations*, vol. 32, no. 4, 2019.

[10] R. Molontay, N. Horváth, J. Bergmann, D. Szekrényes, and M. Szabó, "Characterizing curriculum prerequisite networks by a student flow approach," *IEEE Transactions on Learning Technologies*, vol. 13, no. 3, pp. 491–501, 2020.

[11] S. M. Padhye, D. Reeping, and N. Rashedi, "Analyzing Trends in Curricular Complexity and Extracting Common Curricular Design Patterns," in *2024 ASEE Annual Conference & Exposition*, 2024.

[12] G. L. Heileman, M. Hickman, A. Slim, and C. T. Abdallah, "Characterizing the complexity of curricular patterns in engineering programs," in *2017 ASEE Annual Conference & Exposition*, 2017.

[13] I. Villanueva, "What does hidden curriculum in engineering look like and how can it be explored?" in *Proceedings of the American society of engineering education annual conference and exposition, minorities in engineering division*, 2018.

[14] P. Jamieson, G. Ricco, B. Swanson, B. V. Scoy, and S. Bhunia, "LLM Prompting Methodology and Taxonomy to Benchmark our Engineering Curriculums," in *2025 ASEE Annual Conference & Exposition*, 2025.

[15] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[16] A. Krithara, A. Nentidis, K. Bougiatiotis, and G. Paliouras, "BioASQ-QA: A manually curated corpus for Biomedical Question Answering," *Scientific Data*, vol. 10, no. 1, p. 170, 2023.

[17] G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," *arXiv preprint arXiv:2007.01282*, 2020.

[18] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han, "Large language models can self-improve," *arXiv preprint arXiv:2210.11610*, 2022.

[19] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[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.

[21] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[22] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk *et al.*, "Graph of thoughts: Solving elaborate problems with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 682–17 690.

[23] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2022.

[24] S. Kanti Karmaker Santu and D. Feng, "TELeR: A General Taxonomy of LLM Prompts for Benchmarking Complex Tasks," *arXiv e-prints*, pp. arXiv–2305, 2023.

[25] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, "Zero-shot learning with semantic output codes," *Advances in neural information processing systems*, vol. 22, 2009.

[26] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[27] 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.

[28] M. Mizrahi, G. Kaplan, D. Malkin, R. Dror, D. Shahaf, and G. Stanovsky, "State of What Art? A Call for Multi-Prompt LLM Evaluation," *CoRR*, 2024.

[29] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin *et al.*, "A survey on

hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, 2023.

[30] P. Jamieson, G. Ricco, B. Swanson, and B. V. Scoy, "Results and Evaluation of an Early LLM Benchmarking of our ECE Undergraduate Curriculums," in *2025 ASEE Annual Conference & Exposition*, 2025.