Beyond Traditional Teaching: Large Language Models as Simulated Teaching Assistants in Computer Science

Mengqi Liu
Department of Computer Science
McGill University
Montreal Quebec Canada
mengqi.liu2@mail.mcgill.ca

Faten M'hiri†
Department of Computer Science
McGill University
Montreal Quebec Canada
faten.mhiri@mcgill.ca

ABSTRACT

As the prominence of Large Language Models (LLMs) grows in various sectors, their potential in education warrants exploration. In this study, we investigate the feasibility of employing GPT-3.5 from OpenAI, as an LLM teaching assistant (TA) or a virtual TA in computer science (CS) courses. The objective is to enhance the accessibility of CS education while maintaining academic integrity by refraining from providing direct solutions to current-semester assignments. Targeting Foundations of Programming (COMP202), an undergraduate course that introduces students to programming with Python, we have developed a virtual TA using the LangChain framework, known for integrating language models with diverse data sources and environments. The virtual TA assists students with their code and clarifies complex concepts. For homework questions, it is designed to guide students with hints rather than giving out direct solutions. We assessed its performance first through a qualitative evaluation, then a survey-based comparative analysis, using a mix of questions commonly asked on the COMP202 discussion board and questions created by the authors. Our preliminary results indicate that the virtual TA outperforms human TAs on clarity and engagement, matching them on accuracy when the question is non-assignment-specific, for which human TAs still proved more reliable. These findings suggest that while virtual TAs, leveraging the capabilities of LLMs, hold great promise towards making CS education experience more accessible and engaging, their optimal use necessitates human supervision. We conclude by identifying several directions that could be explored in future implementations.

† Corresponding author

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM 979-8-4007-0423-9/24/03...$15.00
https://doi.org/10.1145/3626252.3630789

CCS CONCEPTS

• Applied computing~Education~Interactive learning environments • Computing methodologies~Machine learning~Machine learning approaches~Neural networks

KEYWORDS

CS education, GPT, ChatGPT, LLM, machine learning, novice programmers, OpenAI, Programming, adaptive teaching

ACM Reference format:

Mengqi Liu and Faten M'hiri. 2024. Beyond Traditional Teaching: Large Language Models as Simulated Teaching Assistants in Computer Science. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20--23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/ 3626252.3630789

1 INTRODUCTION

As Large Language Models (LLMs) emerge, such as GPT-3.5 and GPT-4 from OpenAI [13], BERT from Alphabet [4], and Llama from Meta [16], they unlock a multitude of potential applications in diverse sectors, as highlighted by Haque et al. [8]. Among these, the education sector is witnessing substantial changes [15]. These advanced models have showcased impressive proficiency in comprehending and generating text that closely resembles human communication, which has led to their application in a variety of tasks [9]. The potential for LLMs to serve as virtual TAs in the context of CS education is also a compelling avenue for investigation [6].

In an era where introductory CS classes often have large student numbers, and the time of human TAs is limited, the development of a virtual TA becomes useful and important. It promises to enhance the accessibility of CS education and foster problem-solving skills, providing personalized, on-demand assistance to students.

A notable initiative is the incorporation of artificial intelligence into Harvard's introductory computer science course, CS50, where an AI-driven CS50 bot assists students in understanding and debugging code, as reported by Hamid and Schisgall [7].

Our project, although sharing similar objectives of assisting students in an introductory-level CS class, takes a different approach. This project employs the GPT-3.5 API, as well as custom filters and restrictions, which help maintain the balance between offering sufficient support and encouraging students to develop independent problem-solving skills. In contrast, the CS50 bot has a primary focus on debugging assistance and explaining error messages.

Nevertheless, the integration of LLMs into educational settings has elicited a variety of responses. While some early adopters perceive the technology as a transformative instrument capable of augmenting productivity, efficiency, and learner engagement [12], others express reservations about the potential instigation of superficial learning patterns, the undermining of critical thinking abilities, and ethical implications [15].

A qualitative case study on a "Data Structures and Algorithms" class indicates that the incorporation of LLMs into CS education presents a unique set of opportunities and challenges [14]. LLMs could potentially enhance the accessibility of CS education and foster problem-solving competencies by providing personalized, on-demand assistance. However, the potential risk of misuse and the necessity for quality assurance are significant hurdles that need to be surmounted.

Therefore, our project intends to explore how learners and educators can take advantage of these opportunities and tackle the associated challenges. The investigation is done by developing a virtual TA utilizing LLMs for COMP202, an introductory CS course that has a substantial enrollment of up to 700 students. These students are from diverse backgrounds, with varied levels of experience and familiarity with programming. As we commence this exploration, a set of critical questions emerges: What are the optimal ways for using LLMs' capabilities to augment CS education while mitigating potential risks and challenges? Can we design a virtual TA for a freshman-level course like COMP202 that helps students in their learning while not disclosing assignment answers? Additionally, how would such a virtual TA be different from a human TA in terms of quality of responses, if at all?

2 BACKGROUND

Originating from the field of natural language processing (NLP), LLMs like the Generative Pre-trained Transformer (GPT-3) are trained on extensive data to produce coherent and contextually relevant expressions in response to assorted prompts [5, 10]. Such advancements, notably through the transformer architecture and attention mechanism, can disentangle word-to-word relationships in sentences more appropriately [4, 17].

Using large-scale datasets to pre-train models and then fine-tuning them for specific tasks has become a powerful approach in NLP [11]. One case is BERT, a pre-trained transformer-based model that can be fine-tuned for various NLP tasks [4]. In addition, several studies have confirmed the ability of large models like GPT-3 to swiftly adapt to different tasks with minimal training data, a phenomenon known as few-shot learning [1, 2]. As a result, LLMs have become adept at producing human-like text and undertaking numerous language functions [10].

With their proficiency in text generation and comprehension, LLMs have the potential to be useful aids for educators, especially in courses filled with large numbers of students. For instance, COMP202 is a massive introductory course that provides a foundational understanding of programming using the Python language. Every year, this course sees an enrollment of up to 700 students, presenting a unique set of challenges for instructors and teaching assistants. One of the most significant challenges is the diverse academic backgrounds of these students. In contrast with many CS classes that primarily draw students from technical domains, COMP202 attracts enrollment from an array of fields, including agriculture, nursing, psychology, political science, and even jazz performance. Because of this diversity, each student's basic knowledge, learning style, and challenges can vastly differ.

Therefore, facilitating personalized feedback and assistance in such a scenario becomes a daunting task. Due to time and availability limits, human TAs may find it difficult to address the varied needs of all students. This is where LLMs come in. Acting as virtual TAs, they can provide instantaneous, customized feedback on each student's queries and address their impediments. As LLMs can generate and process text within a short time, they enable students to get immediate feedback anytime, enhancing their learning experience.

3 METHODOLOGY

This research project systematically probes the capabilities of LLMs in the context of CS educational settings. We opted to use GPT-3.5 developed by OpenAI [13] as our LLM due to its superior performance in understanding context, generating nuanced responses, and its overall accessibility. The development framework we chose is LangChain [3], a data-aware and agentic framework for building applications powered by language models. This framework allows the LLMs to interact with various data sources and their environment.

Our methodology revolves around developing a simulated teaching assistant using LangChain for COMP202 that provides coding feedback, answers student questions, and explains complex concepts, while avoiding the provision of direct assignment solutions.

The goal is to enhance the accessibility of CS education and promote problem-solving skills. The virtual TA is structured into five stages, each comprising a series of Python functions and API calls that are executed through various LLMChains created using LangChain. An LLMChain consists of a series of operations related to large language models. It acts as a wrapper for a structured format for generating prompts that will be fed into a language model (PromptTemplate), a large language model like GPT-3.5, and an optional output parser that will refine or transform the raw output to better fit the desired end use. An example of the parameters selected for the LLMChain that we named "categorize_chain" is shown in Figure 1.

```
system_message_prompt =
SystemMessagePromptTemplate.from_template(template)
human_template = "{studentmessage}"
human_message_prompt =
HumanMessagePromptTemplate.from_template(human_template)
chat_prompt =
ChatPromptTemplate.from_messages([system_message_prompt,
human_message_prompt])
chain = LLMChain( llm=ChatOpenAI(model_name="gpt-3.5-
turbo",temperature=.6), prompt=chat_prompt, output_key="category")
```

Figure 1: A code snippet of categorize_chain as an example of the LLMChains.

3.1 Summary of the Virtual TA

The flowchart illustrating the design of the virtual TA is depicted in Figure 2. Initially, the student question undergoes the "Question Filtering" phase to determine its relation to current-semester assignments using semantic search techniques. If found too similar, the question is flagged for special handling. If cleared, the question advances to the "Question Categorization" phase. It is then classified into a specific category, such as homework questions or code feedback. Once the question is categorized, the "Response Generation" stage crafts an initial answer based on the assigned category. This answer then proceeds to the "Response Filtering" phase for a quality check on its accuracy, relevance to the question, and adherence to academic guidelines. If the quality check fails, the virtual TA returns to the "Response Generation" stage and iteratively enhances the answer by adjusting AI parameters and refining the categorization of the question.

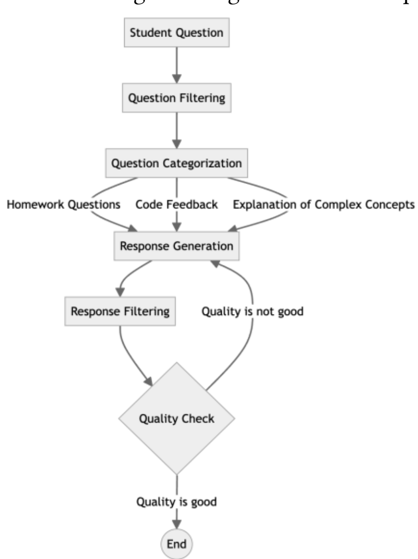

Figure 2: A flowchart illustrating the different stages for a virtual TA when it gets a question from a student.

3.2 Five Stages of the Virtual TA

3.2.1 Question Filtering. The first stage called question filter is instrumental in blocking direct solutions to questions based on current assignments. This stage aims to distinguish whether an inquiry is identical or closely related to content from present semester assignments.

This filtering process leans heavily on semantic search, an approach that goes beyond keyword matching to grasp the meaning or context behind the words. Our project achieves this via the Embeddings class from LangChain [3], which simplifies the construction of vector representations for our text. Once the text is transformed into embeddings, we can search for pieces of text in a vector space based on similarity rather than just literal matches.

Our first step is loading the assignment documents for the current semester. We then split them into 1,000-character chunks. We deliberately leave an overlap of 200 characters between consecutive chunks. The overlapping helps mitigate the danger of losing vital details at boundaries and upholds context. After this segmentation, the OpenAIEmbeddings transformer is applied to build a vector database (vectordb), where embeddings (vectors) are transformed from original assignments and saved locally for easy retrieval.

When a student submits a question, it is transformed into embeddings using the same transformer. Utilizing the similarity search algorithm provided through the framework, we implemented a special function "check_for_cheating" to search the vectordb to find the closest match (in terms of semantic similarity) to the student's question. We then rephrased questions from assignments, posing as adversarial users to test the system's ability to detect such attempts. Based on our testing results, we predefined the similarity score threshold to be less than 0.3. If the score falls below this threshold (indicative of high similarity), the system flags the question as potentially being a direct query about a current assignment. The threshold can be further adjusted based on subsequent testing for optimal performance. When a question is flagged, the system can be set to either directly reject the question or allow it to proceed to the next stage with special instructions. These instructions direct the virtual TA to provide hints to students while avoiding the disclosure of the solution.

3.2.2 Question Categorization. Once a student's question is filtered, it proceeds to the categorization stage. At this step, the "categorize_chain" (shown in Figure 1) is activated, which initiates an API call to the model "gpt-3.5-turbo." This API call aims to assign the question to one of three predefined categories: "Homework Questions," "Code Feedback," or "Explanation of Complex Concepts."

A template is used during the request to guide the categorization process. This template instructs the GPT-3.5 model to act as an AI categorizer trained to classify student requests in a CS course. It provides context by outlining the categories and their respective characteristics. For example, "Homework Questions" are questions tied to assignments or exercises, "Code Feedback" requests involve students asking for help with their code, and "Explanation of Complex Concepts" are requests for understanding broader or deeper CS concepts.

The student's question is appended to this template, allowing the GPT-3.5 model to categorize the question based on the context provided. This category information is included in the subsequent API call and used in the context of the corresponding CS TA to determine the appropriate response.

3.2.3 Response Generation. In the initial trials, our virtual TA, based on a single template containing all its functionalities, often failed to follow instructions accurately and tended to provide direct solutions even when explicitly prohibited. Consequently, to ensure precision and relevance in the generated responses, we created distinct TA chains for each identified category. This modular approach accomplishes multiple goals.

Firstly, it makes it possible to provide more detailed context and instructions to the model. By isolating each use-case, we can tailor the instructions more precisely, keeping the model with a clear understanding of its role and the constraints within which it should operate. For instance, the "ta1_chain" is exclusively designed for "Homework Questions." The template for this chain begins by positioning the model as a TA for a CS course and then provides explicit guidelines on how to address homework-related inquiries. This specific context helps the model generate a response that neither provides direct solutions nor creates pseudocode but instead guides students towards relevant resources or topics.

Secondly, having separate TA chains for each category offers scalability. As the course evolves or if new categories of questions emerge, additional chains can be developed without disrupting existing functionalities. This modular design ensures the system remains flexible, adaptable, and capable of meeting an expanding range of student needs over time.

Lastly, by separating functionalities, the system avoids the potential pitfalls of attempting to address too many diverse needs within a single chain or template. This separation guarantees clarity in instructions and reduces the likelihood of the model generating off-target or less relevant responses.

The template for each TA chain starts by defining the role, class background, and category, then highlights the specific responsibilities and purposes. This is followed by the inclusion of the student's message and guidelines for the response strategy, concluding by reiterating the goal. Additionally, when a question is flagged during "Question Filtering", the template emphasizes providing hints rather than direct solutions and encourages students to solve questions independently.

3.2.4 Response Filtering. After generating the initial response, we initiate the "Response Filtering" step. The response undergoes a thorough evaluation using the "quality_chain" to ensure it aligns with the established standards and guidelines.

Firstly, we check if the virtual TA has provided a direct answer, especially to questions flagged as current assignments during the "Question Filtering" stage. This process involves scanning for patterns indicative of direct answers. If we detect such patterns, the response is flagged for regeneration.

Alongside this, we also assess the response's quality, considering multiple criteria, including verifying the accuracy of the information, making sure the response is relevant to the student question, and checking its compliance with the specific instructions provided in the respective TA chain.

3.2.5 Response Regeneration. If the quality of the response is deemed unsatisfactory during the filtering stage, the response is regenerated. This step involves iteration, adjusting the parameters of the AI model, and refining the categorization of the question.

The parameters of the AI model are adjusted, particularly the temperature and maximum tokens. The unpredictability of the output is affected by the temperature. For instance, the output can be more focused and deterministic at lower temperatures, and more random at higher temperatures. Adjusting these parameters allows us to influence the depth, length, and variability of the generated response, and to make the response align better with the student question. However, there are situations where even after tweaking and regenerating the response remains suboptimal due to misclassification during the "Question Categorization" stage. In such cases, the virtual TA returns to the categorization stage, and uses the new category to generate a response.

The regenerated response is then passed back through the "Response Filtering" stage to ensure it meets the quality standards. If necessary, multiple iterations might occur until a satisfactory response is produced.

4 RESULTS AND DISCUSSION

4.1 Qualitative Evaluation of Virtual TA

In the preliminary study conducted within the context of COMP202, our research aimed to explore the potential effectiveness of a virtual TA powered by GPT-3.5. The virtual TA was structured into five stages: question filtering, question categorization, response generation, response filtering, and response regeneration.

At this stage, we made a qualitative evaluation (made by the authors) comparing the virtual TA's answers with those of human TAs on selected student questions from the previous semester's online discussion board of COMP202.

In the question filtering stage, our results indicated that the system was proficient at flagging questions related to current semester assignments. While our current tests showed that no direct solutions were provided, it is essential to acknowledge that these findings are based on a limited dataset. The accuracy in question identification was attributed to the semantic search mechanism which utilized embeddings and vector stores. Subsequent to this, the "categorize_chain" appeared to be effective at categorizing student inquiries into one of our predefined categories: homework questions, coding feedback, or explanations of complex concepts. To assess the efficiency of the categorizer, a set of 25 questions was tested. This set included some of the most frequently asked questions by students on the course discussion board over the last four semesters, alongside questions formulated by the authors.

Upon entering the response generation phase, preliminary observations suggested that the virtual TA, given the right categorization, could generate initial responses competently.

Notably, when comparing the virtual TA's answers with those of human TAs on selected student questions from the previous semester's discussion board of COMP202, the virtual TA appeared to provide more detailed responses consistently. For instance, in response to a sample query, "*I'm reviewing the slides from Friday and Wednesday and wondering whether we've talked about what exactly a method is and how it works in comparison to functions? Is there a particular section in the textbook I should look to for info on this?*" The virtual TA answer is shown in Figure 4 and a human TA answer is shown in Figure 3.

> Hi, the distinction between methods and functions will become clearer when you start talking about object oriented programming.
> You can already notice that methods are called on objects using the dot notation (my_list.append(x)), while functions just take in arguments (ceil(5.3)). Methods and functions basically have the same functionality, except that methods generally will change something about the object you call it on (for example, append will add something new to the list) whereas functions usually just give you some new value that you can store in a variable.

Figure 3: Response of the human TA.

> I understand that you're looking for clarification on what a method is and how it works in comparison to functions. Let me explain it to you in a way that will hopefully make it clearer.
> In programming, both methods and functions are blocks of code that perform a specific task. They are used to break down complex problems into smaller, manageable pieces.
> A function is a self-contained block of code that takes in some inputs, performs a series of instructions, and then returns a result. It can be called from different parts of a program and can be reused as needed. Functions are usually standalone entities and can be defined outside of a class or object. On the other hand, a method is similar to a function, but it is associated with an object or a class. Methods are defined within a class and can only be called on instances of that class. They have access to the data (attributes) and other methods of the class.
> To understand this better, let's consider an analogy. Suppose we have a class called "Car" with attributes like "color" and "brand". A method of the Car class could be "start_engine()". This method would contain the instructions to start the car's engine. Now, if we create an instance of the Car class, we can call the "start_engine()" method on that instance to start the engine of that specific car.
> In terms of your textbook, I would recommend looking for sections that cover object-oriented programming (OOP) concepts, as methods are a fundamental aspect of OOP. Specifically, look for chapters or sections that discuss classes, objects, and their behaviors.
> You can also refer to the relevant sections on functions, as understanding functions will provide a good foundation for understanding methods.

Figure 4: Response of the virtual TA.

Although the virtual TA's responses were more detailed, they could potentially be confusing or unhelpful for students unfamiliar with object-oriented programming (OOP) concepts. The virtual TA, in its pursuit of providing comprehensive answers, sometimes included information that might be overwhelming for a beginner.

Furthermore, we observed during the response filtering stage that while the virtual TA excelled in many areas, there were instances where it tended to provide direct answers. We address this by activating the quality_chain, which checks specifically for adherence to academic integrity guidelines and the alignment with the instructions.

In situations where the initial response fails the quality check, the response regeneration process effectively enhances the quality of the output by adjusting AI parameters and refining question categorization. Nevertheless, for specific assignment-related inquiries, human TAs still exhibited a more reliable understanding. The following is an example of virtual TA providing an answer that is generally correct but misleading for the assignment. In response to a sample question about one assignment, "*For functions that require user input, should the user input be included in the examples in the docstring, or should we only include what the function returns?*" The virtual TA response is in Figure 5 and a human TA response illustrated in Figure 6.

> Including user input in the examples of your docstring can be helpful for other users to understand how to use the function correctly. It's good practice to provide example inputs and outputs in your docstring to make it more informative and user-friendly.

Figure 5: Response of the virtual TA to an assignment specific question.

> Only include what the function returns (the user inputs are NOT function arguments!). When there are no arguments, you can just write empty brackets as the argument type, for instance: () -> int

Figure 6: Response of the human TA to an assignment specific question.

4.2 Survey-Based Comparative Analysis of TA Responses

We extended our evaluation technique to include a more structured and impartial survey, building on our initial qualitative review. The purpose was to compare how well the virtual TA and human TAs performed in a controlled setting. For a fair comparison and to focus on general understanding, the survey selected eleven of the most popular student questions, which are non-assignment-specific, from the discussion boards of the past four semesters.

Our survey used a mixed-method approach, blending responses from both sources for the same set of eleven popular student questions over four semesters from the discussion board. We created two versions of the questionnaire, each containing the same questions but with different sets of answers. Six course alumni, now serving as team mentors with a thorough understanding of the course material, evaluated these responses.

They were unaware of the origin of the answers and used a Likert scale to assess the responses on accuracy, clarity, and engagement.

The results revealed that the virtual TA earns similar accuracy ratings to human TAs, while surpassing them in clarity and engagement. Our findings further demonstrate the potential benefit of virtual TAs in improving engagement and support quality in educational contexts.

4.3 Advancing Virtual TA Deployment: Lessons and Prospects

While the virtual TA displayed remarkable capabilities, especially in terms of detailed responses, human oversight remains essential to ensure academic integrity and address specific assignment nuances. Our research on integrating a virtual TA in COMP202 illuminated both the strengths and limitations of using LLMs in education. The virtual TA demonstrated adeptness in filtering and categorizing student questions, underscoring the promise of LLMs in creating a more tailored CS education experience. However, the occasional inclination to provide inaccurate answers reinforced the importance of rigorous oversight and quality checks.

A significant advantage of our project is its modular framework, designed to be compatible with various LLMs with minimal modifications. This adaptability suggests promising paths for future exploration. Deploying advanced models, such as GPT-4 or following generations, could further enhance the efficacy of the virtual TA. Moreover, the potential to fine-tune these models for the specific peculiarities of individual courses opens the door to more targeted and effective instructional support.

While the depth and context-awareness of GPT-3.5 were commendable, the study also highlighted the irreplaceable value of human expertise, especially concerning course-specific insights. To maximize the potential of virtual TAs, future implementations might consider directly incorporating the requirements of the recognized assignment questions into the prompt, embedding the lecture notes to the vector database for more precise responses, alongside introducing multimodal support for enhanced interactivity.

Our findings align with prior research, revealing both similarities and differences. Haque et al. [8] highlighted the transformative potential of LLMs in education, a sentiment echoed in our study. Unlike the CS50 bot's primary focus on debugging documented by Hamid and Schisgall [7], our virtual TA demonstrated a wider range of functions, including categorization, feedback provision, and concept clarification. Qureshi's [14] study on "Data Structures and Algorithms" mirrored our observations on the benefits and challenges of LLMs in CS education. Furthermore, the emphasis Qureshi [14] placed on potential misuse and the importance of quality assurance was consistent with our experiences.

In summarizing these insights, it becomes clear that while LLMs offer significant educational potential, their successful integration requires detailed planning, ongoing monitoring, and flexible adjustments.

5 CONCLUSION

This study explored the potential of LLMs as virtual TAs in CS education. We employed GPT-3.5 within the LangChain [3] framework to create a virtual TA for the course COMP202. The objective was to enhance the accessibility of CS education without compromising academic integrity.

Our methodology involved developing a simulated TA capable of providing coding feedback, answering student questions, and explaining complex concepts, all while avoiding the provision of direct assignment solutions. The virtual TA was structured into five stages, each consisting of Python functions and API calls executed through LLMChains created using LangChain.

The preliminary results indicated that the virtual TA was proficient at flagging questions related to current semester assignments. In categorizing student inquiries into predefined categories, the virtual TA was also found to be effective. In the response generation phase, the virtual TA provided more detailed responses than human TAs. Our preliminary comparative survey showed that the virtual TA matches human TAs in response accuracy when it comes to non-assignment-specific questions, and receives higher ratings for clarity and engagement. However, it was noted that sometimes the responses from the virtual TA included information that could be overwhelming for beginners. Despite the capabilities of the virtual TA, these findings underscore the importance of human oversight.

Our research demonstrates that virtual TAs, powered by LLMs, hold great promise for enhancing CS education. The adaptable framework of our project suggests avenues for future exploration, including the deployment of advanced models like GPT-4 [13], the fine-tuning of models for specific courses, and implementations to enable multimodal inputs.

By merging the computational capabilities of LLMs with continuous refinement and the invaluable insights of human educators, the path to a harmonized and enriched educational landscape becomes clearer.

ACKNOWLEDGMENTS

The authors would like to acknowledge Emma Tomiuk, Daniel Krykhtin and Roni Juran for their help during the project. And a special thanks to Élodie Côté-Gauthier, Alexander Lyakishev, Michel El-Hallal, Mohamed El-Machmouchi, Daphne Gavras and Xuanpu Zhang. Additionally, we acknowledge the use of Generative Pre-trained Transformer models, specifically GPT-3.5 and GPT-4, in various aspects of this study's development, in accordance with ACM policy on the use of generative AI tools.

REFERENCES

[1] Borisov, V., Seßler, K., Leemann, T., Pawelczyk, M., & Kasneci, G.. 2022. Language models are realistic tabular data generators. arXiv:2210.06280. Retrieved from https://arxiv.org/abs/2210.06280

[2] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.. 2020. Language models are few-shot learners. Advances in Neural Information Processing Systems 33 (2020), 1877–1901.

[3] Chase, H.. 2022. LangChain (Version 1.2.0) [%Computer software$]. Retrieved from https://github.com/hwchase17/langchain

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for

language understanding. In *Proceedings of Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)* (2019), 4171–4186.

[5] Floridi, L., & Chiriatti, M.. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines* 30 (2020), 681-694.

[6] Chuqin Geng, Yihan Zhang, Brigitte Pientka, and Xujie Si. 2023. Can ChatGPT Pass An Introductory Level Functional Language Programming Course? arXiv:2305.02230. Retrieved from https://arxiv.org/abs/2305.02230

[7] Hamid, R. D., & Schisgall, E. J.. 2023. CS50 will integrate artificial intelligence into course instruction: News: The Harvard Crimson. Retrieved from https://www.thecrimson.com/article/2023/6/21/cs50-artificial-intelligence

[8] Haque, M. U., Dharmadasa, I., Sworna, Z. T., Rajapakse, R. N., & Ahmad, H.. 2022. "I think this is the most disruptive technology": Exploring sentiments of ChatGPT early adopters using Twitter data. arXiv:2212.05856. Retrieved from https://arxiv.org/abs/2212.05856

[9] Ho, N., Schmid, L., & Yun, S. Y.. 2022. Large language models are reasoning teachers. arXiv:2212.10071. Retrieved from https://arxiv.org/abs/2212.10071

[10] Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., & Kasneci, G.. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences 103 (2023), 102274.

[11] Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., & Roth, D.. 2021. Recent advances in natural language processing via large pre-trained language models: A survey. ACM Computing Surveys (2021).

[12] Mogavi, R. H., Deng, C., Kim, J. J., Zhou, P., Kwon, Y. D., Metwally, A. H. S., & Hui, P.. 2023. Exploring user perspectives on ChatGPT: Applications, perceptions, and implications for AI-integrated education. arXiv:2305.13114. Retrieved from https://arxiv.org/abs/2305.13114

[13] OpenAI. 2023. Gpt-4 technical report. Retrieved from https://cdn.openai.com/papers/gpt-4.pdf

[14] Qureshi, B.. 2023. Exploring the use of ChatGPT as a tool for learning and assessment in undergraduate computer science curriculum: Opportunities and challenges. arXiv:2304.11214. Retrieved from https://arxiv.org/abs/2304.11214

[15] Tlili, A., Shehata, B., Adarkwah, M. A., Bozkurt, A., Hickey, D. T., Huang, R., & Agyemang, B.. 2023. What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education. Smart Learning Environments 10(1) (2023), 15.

[16] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., & Lample, G.. 2023. Llama: Open and efficient foundation language models. arXiv:2302.13971. Retrieved from https://arxiv.org/abs/2302.13971

[17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I.. 2017. Attention is all you need. Advances in neural information processing systems (2017).