# Assistant Teaching System for Computer Hardware Courses Based on Large Language Model

Dongdong Zhang[1,2](✉), Qian Cao[1], Yuchen Guo[1,2], and Lisheng Wang[1,2]

[1] College of Electronics and Information Engineering, Tongji University, Shanghai, China
ddzhang@tongji.edu.cn

[2] National Computer and Information Technology Experiment Teaching Demonstration Center, Tongji University, Shanghai, China

**Abstract.** Recently, Large Language Models (LLMs), represented by ch1ChatGPT, have garnered significant attention in the field of education due to its impressive capabilities in text generation, comprehension, logical reasoning, and conversational abilities. We incorporate LLMs into the theoretical and experiment teaching of our Digital Logic and Computer Organization courses to enhance the teaching process. Specifically, we propose and implement an assistant teaching system consisting of a knowledge-based Question and Answer (Q&A) system and an assistant debugging and checking system. For the theoretical teaching session, the Q&A system utilizes historical Q&A records and ChatGPT to answer students' questions. This system reduces the repetitive workload for teachers by answering similar questions, and allows students to receive answers in time. For the Field-Programmable Gate Array (FPGA)-based experiment teaching session, the assistant debugging and checking system employ debug assistance module to explain error messages for students. Furthermore, a LLM-generated code checking module assists teachers in detecting academic misconduct among students' code submissions.

**Keywords:** Computer Science · Hardware Courses · Assistant Teaching System · Large Language Models · FPGA

## 1 Introduction

Digital Logic and Computer Organization Principles are fundamental courses in computer science education, significantly influencing students' comprehension of the underlying principles of computer. In addition to theoretical teaching in the fundamental concepts and principles of digital circuits and computer architecture, we have introduced experiment courses to culture students' practical skills [1]. These experiments require students to apply Hardware Description Languages (HDLs) like Verilog [2] to implement functional modules on Xilinx

W. Hong and G. Kanaparan (Eds.): ICCSE 2023, CCIS 2023, pp. 301–313, 2024.
https://doi.org/10.1007/978-981-97-0730-0_27

Nexys 4 FPGA development boards [3]. In the Digital Logic course, students engage in the implementation of basic modules of digital circuits, including basic gate circuits, data selector and distributor, encoders and decoders, barrel shifter, data comparator, trigger, PC register, RAM, ALU and so on. In the Computer Organization Principles course, students progress beyond these foundational modules to tackle more advanced experiments, such as multiplier, divider, interrupt handling experiments, ultimately culminating in the design of a MIPS CPU. Previously, we have meticulously crafted a comprehensive teaching strategy [1] and developed an automatic testing system for HDL practice teaching [4–6].

During our teaching practice, we have encountered several challenges in the Digital Logic and Computer Organization courses. These courses are distinguished by their extensive content, high complexity, and inherent concept abstraction. Both in theoretical teaching and laboratory exercises, students frequently encounter difficulties and experience delays in obtaining answers to their inquiries. Many of these difficulties are recurrent and shared among students, placing additional burden on educators to repeatedly address similar questions. Furthermore, a significant portion of these challenges is concentrated in the FPGA practical process of the courses. HDLs differ substantially from the software programming languages that students have previously encountered, and the error messages generated by Integrated Development Environments (IDEs) such as Vivado [7] and ModelSim [8] are often challenging to understand. These obstacles hinder the learning process, especially for novice learners.

Recent advancements in Large Language Models (LLMs), including ChatGPT [9], ChatGLM [10], and Llama [11], have garnered significant attention for their exceptional capabilities in text generation, comprehension, logical reasoning, and dialogue engagement. This growing interest extends to their potential applications in the field of education [12–15]. In this paper, we integrate LLMs into our computer hardware course teaching and propose an assistant teaching system, consisting of two main components: a question and answer (Q&A) system and an assistant debugging and checking system. The Q&A system first utilizes historical records and ChatGPT to promptly respond to questions submitted by students. Teacher intervention is reserved solely for new questions that cannot be resolved by the system, thereby reducing the redundancy in teacher workload. The Q&A system continuously updates and accumulates language data through the Q&A module, improving its response capabilities over time. Furthermore, we develop an assistant debugging and checking system for experiment courses. On the one hand, debug assistance module utilizes ChatGPT to explain error messages of the Verilog programs, aiding students in debugging. On the other hand, considering that the advent of LLMs also introduces potential academic integrity risks, a LLM-generated code checking module is introduced to check whether students' submitted code is generated by GPT. In summary, our assistant teaching system aims to enhance teaching efficiency, improve students' understanding of course content, and mitigate potential academic integrity risks in our computer hardware courses.

## 2 Related Technical Methods

Recently, LLMs usch as BERT [16], GPT3 [17], GPT4 [18] have emerged as a groundbreaking technology in the field of Natural Language Processing (NLP). Through extensive pre-training on vast text corpora and fine-tuning with specific instructions, they have exhibited remarkable performance in language generation and comprehension tasks. One common application scenario for LLMs is Q&A systems, such as ChatGPT [9], ChatGLM [10], claude [19], and LLaMA2Chat [11]. While LLMs showcase impressive capabilities in general domains, their limitations become evident when applied to specialized fields. As LLMs are pretrained on general corpora, they may lack domain-specific expertise in specialized fields. To address this issue, two primary approaches are considered. The first approach involves fine-tuning of pre-trained LLM on a domain-specific dataset, such as ChatDoctor [20], ChatLaw [21], EduChat [13]. This method allows LLMs to acquire specific domain knowledge and communicate effectively within a particular knowledge context. However, this fine-tuning approach requires a substantial amount of domain-specific data and significant GPU computing resources. The second approach utilizes embedding technology [22–24]. It involves computing vectors for specific knowledge using embedding models and then storing these vectors, along with their corresponding content, in a database. During the query phase, similarity queries are performed to match relevant top K results. At present, there is not enough corpus available to fine-tune large models to create a specialized LLM for hardware course teaching. We have opted the second approach to construct a Q&A system that responds to students' questions based on existing QA records within our courses.

As widely recognized, compiler error messages often pose a significant obstacle to beginners when learning programming. Previous studies have harnessed LLMs to generate explanations of error messages in programming education in languages such as C [15] and Python [14], making them easier to understand. Our course includes practical exercises where students write Verilog HDL code and deploy it onto FPGA. We have previously proposed an automatic testing system for Verilog HDL programs [5,6]. This testing system categorizes errors into two types. The first, Compile Error (CE), indicates compilation errors and returns error messages from ModelSim. The second, Wrong Answer (WA), indicates that the code compiles but produces incorrect simulation and returns a comparison report between simulation result and standard one. While this testing system can correctly assess the correctness of submitted code, it often fails to provide helpful guidance due to the incomprehensible error messages. Therefore, inspired by methods [14,15], we introduce a debug assistance module to enhance students' understanding of error messages.

Measure of Software Similarity (MoSS) [25] is widely used as a code plagiarism detection tool in programming courses, coding competitions, and research. MoSS's primary function is to identify similarity between two or more source code files, and it also supports HDLs like Verilog and VHDL. To counteract the possibility of specific students utilizing ChatGPT to generate code for their assignments, we introduce an LLM-generated code checking system based on code similarity.

## 3 Proposed Assistant Teaching System

### 3.1 Q&A System

We construct a question-and-answer(Q&A) system based on LLM. The workflow is shown as Fig. 1.

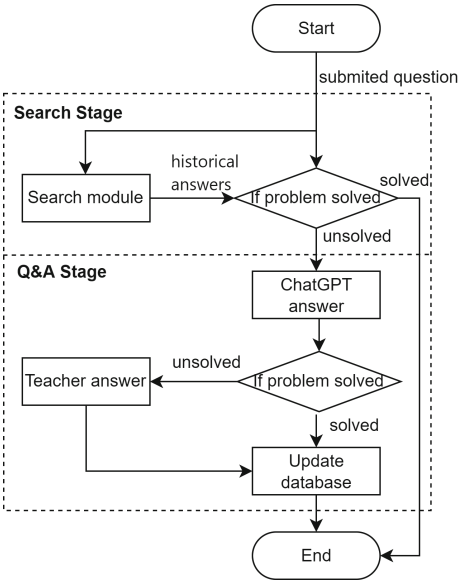

**Fig. 1.** Flow chart of Q&A system.

When a student submits a query, the system follows a 2-stage process to address the question. Search Stage: The system initially employs a search module to query the knowledge base for relevant historical question-and-answer pairs. This step aims to provide the student with immediate access to existing solutions to similar queries from the past. Q&A Stage: If the student's question remains unanswered after the search module's results, the system proceeds to the Q&A stage. First, it connects to the OpenAI API, utilizing ChatGPT to attempt to answer the student's question. ChatGPT engages in a conversation with the student, providing responses. Questions that cannot be solved by the GPT are then answered by the teacher. Once the question is resolved, whether by ChatGPT or a teacher, the system logs this Q&A interaction into the knowledge base. This ensures that the newly answered question becomes part of the knowledge repository for future reference, helping other students with similar queries. In addition, the embedding-based search module can find the most relevant content with greater accuracy and efficiency.

### 3.2 Assistant Debugging and Checking System

The emergence of LLM brings new changes in teaching and learning. Through experimental testing, we have observed that when design objectives, desired functionalities, and well-defined input-output interfaces are explicitly described within the prompt, ChatGPT is capable to generate Verilog code and successfully pass our automatic testing procedures. This signifies that ChatGPT has the potential to assist the learning of the HDL verilog in the practical course. However, it also raises the concern that students may resort to direct copying the Verilog code generated by ChatGPT.

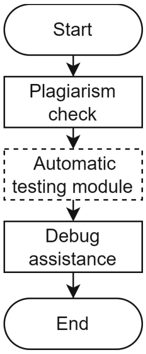

**Fig. 2.** Flow chart of assistant debugging and checking system.

In previous work, we have implemented an automatic detection system (dotted line parts in Fig. 2). To harness the potential of LLMs in educational assistance while mitigating the risk of academic misconduct, We propose an assistant debugging and checking system. Considering that students often encounter challenges during the practical phase of writing Verilog code, we propose a debug assistance module to utilize ChatGPT for interpreting error messages, thus assisting students in the debugging process. To avoid the risks that students may cheat by directly utilizing ChatGPT to generate Verilog code, we introduce a LLM-generated code checking module.

## 4 System Implementation

### 4.1 Q&A System Implementation

For the scheme above, we develop a Q&A website for a concrete implementation, as shown in Fig. 3. The entire system consisting of a front-end pages and a back-end server is implemented based on the Python Flask framework. The front-end serves as an interfacial interaction for students to submit questions and query results, and allows teachers to provide answers to unresolved inquiries.

The backend consists of two distinct modules. The Q&A module collaborates with ChatGPT and the teacher to answer the student's question and inserts the solved quastion-answer records into the database. The search module retrieves relevant Q&A records from the existing records in the database as references for students.

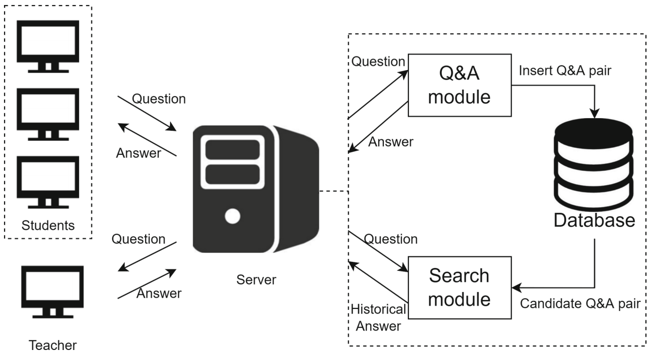

**Fig. 3.** Q&A system framework.

During the early stages of system operation, the primary focus is on knowledge accumulation through the Q&A module. Given that ChatGPT may not always provide satisfactory answers, teacher involvement is often required. After the system has been running for a while, the Q&A module has accumulated a sufficient number of question-and-answer records in the database. At this point, relying solely on the historical records returned by the search module is sufficient to address the majority of issues. Only when new questions arise that cannot be answered by the existing knowledge, is the Q&A module and the perhaps involvement of teachers required to further supplement the database.

**Embedding-Based Search Module.** The embedding-based search module evaluates similarity with the feature embeddings of question sentence and returns the top K most relevant QA records. The user's input question sentence is first vectorized into an embedding with the OpenAI's API text-embedding-ada-002. Afterwards, vector similarity is measured using cosine similarity based on the embeddings, and the system searches the database to retrieve the top K questions that are most similar to the submitted question.

For traditional relational databases, the specific process is as illustrated in Fig. 4. The query module first retrieves two columns, the question IDs and question embeddings from the database. After calculating the cosine similarity between the submitted question's embedding and all existing embeddings, the top K most similar records are selected. Finally, the complete Q&A records are retrieved according to the selected question IDs from the database and sent to the

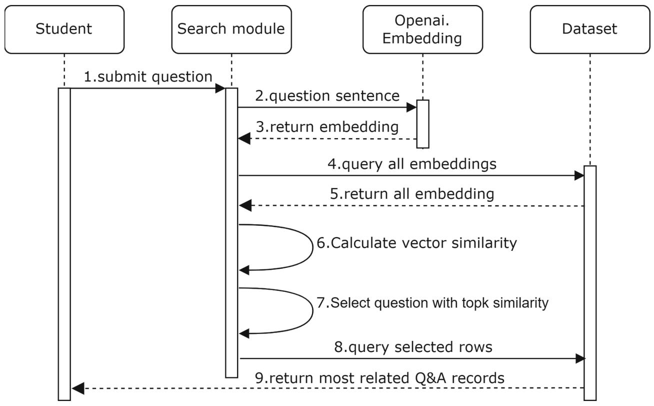

**Fig. 4.** UML of search module with relational database.

student's terminal. Embeddings are typically high-dimensional vectors. Specifically, OpenAI's text-embedding-ada-002 embeds text into a 1536-dimensional vector. In traditional relational databases, the need to extract entire embedding columns results in a significant amount of read and write operations, which can be time-consuming. Additionally, similarity computation for all rows also imposes a large computational effort. These limitations indeed restrict the scale of databases.

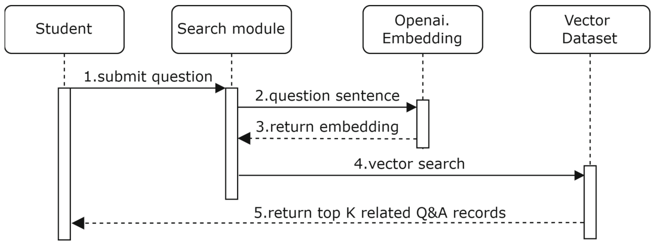

**Fig. 5.** UML of search module with vector database.

Vector databases are specifically designed for the storage and processing of vector data, offering efficient vector retrieval capabilities. The workflow as shown in Fig. 5, involves the creation of embedding, subsequently querying the database for similar vectors, and directly returning the corresponding content associated with these vectors. The core functionality of vector databases lies in similarity search, accomplished by calculating the distance between a vector and all other vectors to identify the most similar ones. Most vector databases provide efficient similarity measurement methods, such as cosine similarity, and are optimized to accelerate these queries. Vector databases align seamlessly with our solution.

Specifically, we chose Milvus [26] vector database to implement efficient similarity search.

**Q&A Module.** The Q&A module is responsible for addressing questions that cannot be answered by the historical records. It consists of two stages.

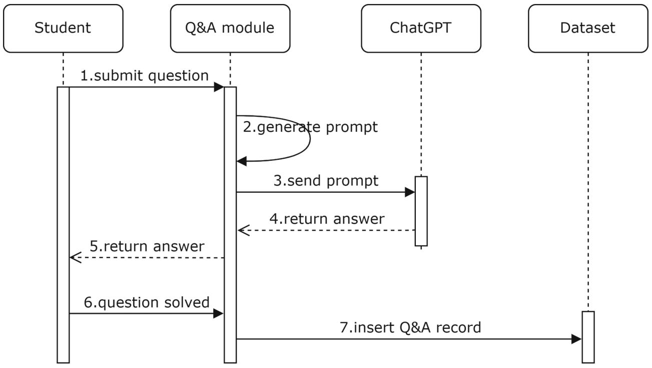

**Fig. 6.** UML of Q&A stage 1.

In the first stage, the Q&A module begins by attempting to address inquiries using ChatGPT as shown in Fig. 6. This module generates prompts by incorporating information including the course name, submitted question into a predefined template. When applied to specialized vertical domains such as knowledge within the course and Verilog programming, ChatGPT lacks domain-specific knowledge and often fails to provide effective answers. Although the historical records returned by the search module cannot directly provide a satisfactory solution to the submitted problem, they can serve as contextual prompts for ChatGPT to reference. ChatGPT suffers from the hallucination problem, where the generated response may be not truthful and mislead students. Therefore, it is essential to emphasize this point in the prompt. The prompt template example is as follows:

*You are an expert in digital circuit and computer architecture. And you will help students study ¡course name¿.*

*I face the following problems ¡submitted question¿, please answer them.*

*Here are some historical records that might help ¡related Q&A¿*

*If you don't know the answer just say you don't know. DO NOT try to make up an answer.*

The module sends the generated prompts via an HTTP API call to OpenAI's gpt-3.5-turbo, and then forwards the answers to the student's terminal. If the student feedback that the question has been resolved, the question is marked as resolved, and both the question and answer records are inserted into the database.

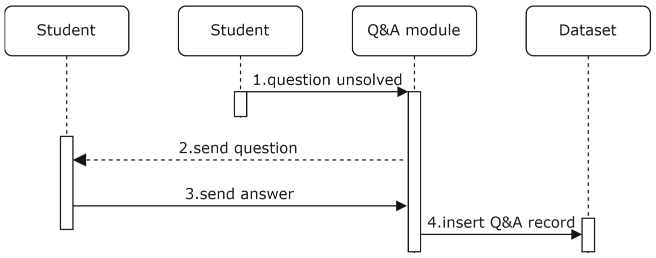

**Fig. 7.** UML of Q&A stage 2.

Otherwise, it proceeds to the second stage, where the teacher takes responsibility for answering as shown in Fig. 7. The question is marked as unsolved and sent to the teacher's terminal. Once the teacher answers it, both the question and answer records are inserted into the database.

## 4.2 Assistant Debugging and Checking System Implementation

**Debug Assistance Module.** In order to help students debug, the debug assistance module leverages ChatGPT to provide more comprehensible explanations for errors occurring during the compilation and simulation procedures of Verilog code. The specific workflow is as shown in Fig. 8.

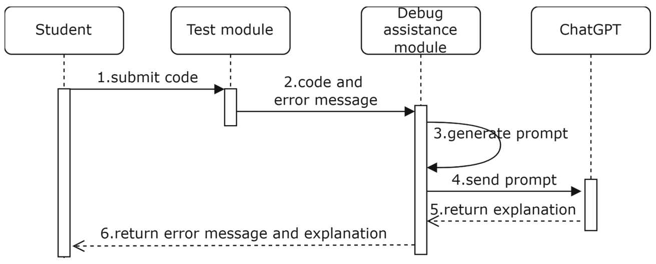

**Fig. 8.** UML of debug assistance module.

It first generates a prompt based on the information from the testing module. The prompt is then transmitted to ChatGPT through an API call, seeking explanations for the error message. Ultimately, the ChatGPT-generated error explanations are appended to the test report and delivered to the students' terminals.

The prompt is generated with a predefined template. It begins by outlining the objectives of the code, including functional description and interface definitions, just as in the previous section. Next, it incorporates the submitted design code along with the testbench code from the server. Finally, it populates the error types and error messages based on the report generated by the testing module. Additionally, it is important to emphasize that the generated feedback serves as an educational tool to aid students rather than a replacement for their efforts. The prompt template is as follows:

*You are a tutor helping a student to explain the error in verilog program. Do not fix the program. Do not provide code.*

*I want to implement a module.: <module description>.*

*This is my Verilog design code: <submitted source code>.*

*This is my Verilog testbench code: <testbench code>.*

*An error occurred during the <error types>, help me understand this error message: <error reports>.*

*Remember, you are tutor helping a student. Do not write code for the student.*

**LLM-Generated Code Checking Module.** Current LLMs have the capability to generate accurate Verilog code and pass the automatic testing. However, this also means that students can potentially plagiarize from LLMs, which is a situation we want to avoid. To address this concern, we propose a LLM-generated code checking scheme.

Before the module is put into operation, we need to utilize current leading LLMs such as ChatGPT and Claude to generate reference code as a basis for LLM-generated code checking. To enable LLMs to generate effective Verilog code, the prompt necessitates the inclusion of the functional descriptions, along with explicit input/output(I/O) interface definitions. Here is a prompt example of a data selector:

*Write the Verilogg code for the data selector. A Data Selector is a fundamental logic circuit that is used to choose one of several input signals based on the state of input switching signals.*

*The interface is defined as: the switching is controlled by the encoding of two control signals s0 and s1, c0∼c3 are the data inputs and z is the output.*

*selector41(*
*input [3:0] iC0, //four-bit input signal c0*
*input [3:0] iC1, //four-bit input signal c1*
*input [3:0] iC2, //four-bit input signal c2*
*input [3:0] iC3, //four-bit input signal c3*
*input iS1, //selection signal s1*
*input iS0, //selection signal s0*
*output [3:0] oZ //four-bit output signal z*
*);*

The workflow of this plagiarism detection module is shown in Fig. 9. When a student submits a code for a particular assignment, the detection module will perform a similarity check between the submitted code and the reference code generated by the LLMs for that specific experiment. Specifically, we utilize the MoSS [25] provided by stanford to implement code similarity checks and obtain detection reports. If the similarity is too high, it raises suspicion of using LLM-generated content. It's important to note that code similarity doesn't certainly

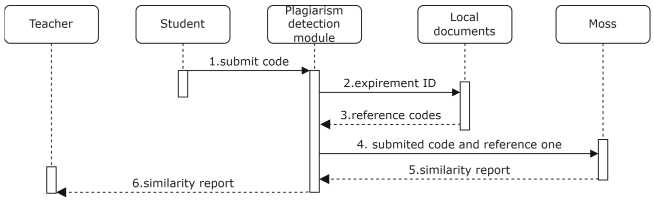

**Fig. 9.** UML of LLM-generated code checking module.

indicate plagiarism, but this module flags it and provides a report for administrators and teachers to review. This module serves only as an aid, and the final judgment is made by the teacher.

## 5 Conclusion

In this paper, we propose an assistant teaching system based on large language models for our courses in digital logic and computer organization. This system effectively improves teaching efficiency in both theoretical and experiment courses, enhances student understanding of course content, and reduces potential academic integrity risks. In the practical application of teaching, this system can greatly facilitate both students and teachers. The Q&A system can swiftly answer students' questions, helping them quickly resolve uncertainties in their learning process and providing real-time support. By storing historical Q&A records, a knowledge base can be built, thereby accumulating and disseminating knowledge. The search module utilizes LLM to extract semantics and matches relevant Q&A records based on feature embeddings, enabling the system to answer similar questions more effectively in subsequent queries. By utilising knowledge base and ChatGPT, Q&A systems can address a significant portion of questions, thereby substantially reducing the burden on teachers. In experiment teaching, using ChatGPT to assist students in debugging can expedite issue identification and resolution, thereby boosting efficiency in students' programming and experimental learning. Finally, the LLM-generated code checking system helps mitigate potential academic integrity risks.

## References

1. Wang, L., Huang, R.: A comprehensive experiment scheme for computer science and technology. In: Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS), p. 1. The Steering Committee of The World Congress in Computer Science, Computer(2012)
2. Thomas, D., Moorby, P.: The Verilog® Hardware Description Language. Springer, Heidelberg (2008). https://doi.org/10.1007/978-0-387-85344-4
3. Berg, M., Poivey, C., Petrick, D., Espinosa, D., Lesea, A., LaBel, K.A., Friendlich, M., Kim, H., Phan, A.: Effectiveness of internal versus external seu scrubbing mitigation strategies in a xilinx fpga: Design, test, and analysis. IEEE Trans. Nucl. Sci. **55**(4), 2259–2266 (2008)

4. Lu, S., Li, G., Wang, Y.: CPU design for computer integrated experiment. In: Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS), p. 1. The Steering Committee of The World Congress in Computer Science, Computer (2012)
5. Wang, L., Zhou, H., Zhang, D.: Automatic testing scheme of hardware description language programs for practice teaching. In: 2017 12th International Conference on Computer Science and Education (ICCSE), pp. 659–662. IEEE (2017)
6. Wang, L., Ruan, J., Zhang, D.: MIPS CPU test system for practice teaching. In: 2017 12th International Conference on Computer Science and Education (ICCSE), pp. 663–666. IEEE (2017)
7. Feist, T.: Vivado design suite. White Paper **5**, 30 (2012)
8. Ha, J., Jeong, H.: Modelsim simulation for real-time stereo matching using DP algorithm. In: 2012 International Conference for Internet Technology and Secured Transactions, pp. 244–248. IEEE (2012)
9. Chatgpt: Optimizing language models for dialogue. https://chat.openai.com/
10. Du, Z., et al.: GLM: general language model pretraining with autoregressive blank infilling. arXiv preprint arXiv:2103.10360 (2021)
11. Touvron, H., et al.: Llama: open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
12. Baladn, A., Sastre, I., Chiruzzo, L., Ros, A.: Retuyt-inco at bea 2023 shared task: tuning open-source llms for generating teacher responses. In: Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023), pp. 756–765 (2023)
13. Dan, Y., et al.: Educhat: a large-scale language model-based chatbot system for intelligent education. arXiv preprint arXiv:2308.02773 (2023)
14. Leinonen, J., Hellas, A., Sarsa, S., Reeves, B., Denny, P., Prather, J., Becker, B.A.: Using large language models to enhance programming error messages. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education, vol. 1, pp. 563–569 (2023)
15. Taylor, A., Vassar, A., Renzella, J., Pearce, H.: Integrating large language models into the debugging c compiler for generating contextual error explanations. arXiv preprint arXiv:2308.11873 (2023)
16. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
17. Brown, T., et al.: Language models are few-shot learners. Adv. Neural. Inf. Process. Syst. **33**, 1877–1901 (2020)
18. Gpt-4 technical report. https://cdn.openai.com/papers/gpt-4.pdf
19. Claude 2. https://www.anthropic.com/index/claude-2
20. Li, Y., Li, Z., Zhang, K., Dan, R., Jiang, S., Zhang, Y.: Chatdoctor: a medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge. Cureus **15**(6) (2023)
21. Cui, J., Li, Z., Yan, Y., Chen, B., Yuan, L.: Chatlaw: open-source legal large language model with integrated external knowledge bases. arXiv preprint arXiv:2306.16092 (2023)
22. Ma, X., Zhu, Q., Zhou, Y., Li, X.: Improving question generation with sentence-level semantic matching and answer position inferring. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 8464–8471 (2020)
23. Reimers, N., Gurevych, I.: Sentence-bert: sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084 (2019)

24. Muennighoff, N.: SGPT: GPT sentence embeddings for semantic search. arXiv preprint arXiv:2202.08904 (2022)
25. A system for detecting software similarity. http://theory.stanford.edu/~aiken/moss/
26. Vector database built for scalable similarity search. https://milvus.io/