The SIGHT Advisor System

Anthony Testa
CS 7675

SCENT

The impetus for this project was the SCENT programming advisor system, designed to teach students the LISP programming language. This system utilized a unique design, employing domain experts and strategy judges to determine the goal of a program, and provide feedback to a student (Gordon I. McCalla). A total system overview can be seen in figure 1, with the observers, and strategy judges determining what strategy the student is using, the diagnosticians providing analysis of potential errors (non-syntactic), and the task expert using this information to provide the final feedback to the student.

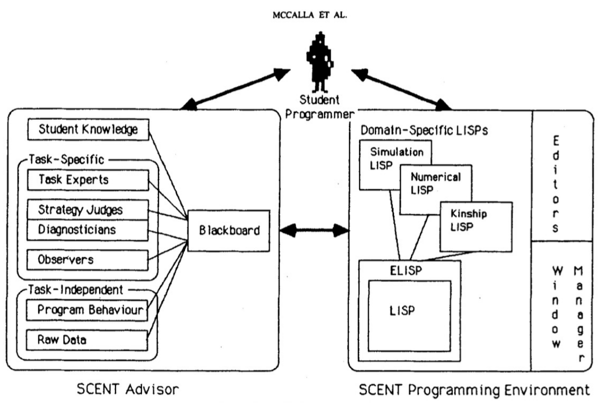

Figure 1: The design of the SCENT programming advisor

The team behind SCENT also later developed a more intricate student model, that utilized a granularity approach to represent the student's knowledge base (Greer). This structure made the system incredibly capable and proved to be a worthy area of research. However, the system was inherently inflexible in its design. Strategy judges, and task experts, were coded for a specific strategy, to solve a specific programming problem. As a general programming tool then, it was not applicable.

Modern Tools

In the years following the release of *Attention is all you need* (Ashish Vaswani) a variety of large language models (LLMs) were released to the public utilizing this approach, with varying degrees of success. Now, these tools are becoming widely available for many different applications, such as GitHub's CoPilot, OpenAI's ChatGPT, and Meta's LLaMa.

These tools have sparked debate about their usefulness in the classroom. Many believe students become reliant on these tools, going so far as to replace the student in completing assignments. The focus now of AI in education is looking to effectively use these tools to help students, rather than replace them.

SIGHT

The goal of this project is to implement a version of SCENT, that utilizes a LLM to replace the strategy judge, task expert, and other task specific components in order to provide a more flexible tool for use in the classroom.

Design

SIGHT utilizes the LLM released by Meta, LLaMa 2 (Meta AI). The model is free to download in its entirety and is designed for use by AI researchers. Three versions of this model exist, with 7B 13B and 70B parameters. For this project we will use the 7B parameter model, translated from python to cpp in order to reduce the model size, utilizing the LLaMaCPP project (ggerganov). This open source project downsizes the (7B) model from 13GB to 3.9 GB. Considering all weights must be stored in memory for evaluation, this makes the hardware requirements of running this model locally significantly more approachable. While this does impact performance, it allows for repetitive calls to the model without any cost. Alternative solutions incur a fee per API call.

LLaMa2 was trained mostly on chat data, and as such is more suited for conversation than programming. A key advantage of the design of this project is that the LLM could be swapped out at any point in the future, for a better performing, or more specifically trained model.

The interface for the original SCENT advisor was built into an IDE for LISP. To carry on that idea, SIGHT is a VSCode extension, and provides feedback through the programmer's coding pane. This can also be updated easily, as VSCode evolves, and is supported in almost all browsers as well.

Figure 2: SIGHT advisor output for a python program.

VSCode takes a programmer's input as they work in the IDE, and the SIGHT extension intercepts this to pass to LLaMa. SIGHT begins by asking LLaMa what level programmer produced this code, and then stores that as a variable. Next, SIGHT asks what the objective of the code is. Finally, SIGHT asks for advice on how to improve the program and produces that as output to the programmer as seen in figure 2. All interactions are recorded in a text file and can be viewed at any time. The transcript of a conversation can be seen in figure 3.

Capabilities

The main advantage over a design like SCENT, is that SIGHT is not restricted to one domain, or even one programming language. LLMs, depending on their training, can provide general knowledge and expertise on a range of topics. This allows the advisor to assist students tackling a range of topics, in a multitude of languages, without any adjustment from human input. The SCENT advisor was programmed for a specific task, and as such would require complete reworking for different domains. Certain tasks are broadly applicable, such as sorting, however the range of tasks and strategies that are available to solve them is simply too great to rely on human programming alone.

SIGHT is also structured in such a way to LLaMa2 is not the only advisor available. At the time of writing, gpt4all is another open-source alternative (Mulyar), and any API to a LLM could be used such as GPT3.5 from OpenAI. This means in the future as the technology progresses, and models become more trained on providing programming feedback, SIGHT can adapt with them to provide students a better experience. As such, LLaMa2 is not the best choice for this application (by quality of feedback). Figure 3 illustrates a conversation around the same python code from the previous figure.

User:def sortNums(nums: list) -> list:
```
        """
        Sort the given list of numbers
        """
        return sort(nums)
```
Given the python code above, rank the programmer that wrote it as one of the following: beginner, intermediate, advanced.
ChatLLaMa: Intermediate
User: Given the programmer is an Intermediate
What improvements can be made in the above code?
ChatLLaMa:
1) you could move the sorting line into sort function (so it will not be repeated), like this:

```
        def sort(numbers):
                new_numbers = []
                for n in numbers:
                        new_numbers.append(n)
                return new_numbers
```
2) You should create a "main()" function and call sort() inside main(), because sort() does not take any arguments, so calling sortNums(...) would raise an error.

Figure 3: An interaction with LLaMa2

It is apparent that the suggestion made by LLaMa2 does not make much sense, and in the case of the 7B parameter model, this is not a rare event. Frequently responses make little sense, sometimes consisting of emojis, and suggest that a more programming focused model would be a better fit for this application.

The user interface for SIGHT is also flexible, with VSCode offering extensive ways for the LLMs to interact with a programmer. Figure 2 shows a text output on the bottom of the screen, however VSCode supports suggestion highlighting, and popups. In the future SIGHT could utilize this to make suggestions to programmers similarly to syntax highlighting from IDEs. Only the suggestions made by SIGHT extend beyond syntax, and in an ideal world could detect minor errors that do not cause crashes, but unintentional behaviour.

Insights

One of the main goals for this project was to explore what limitations currently exist for using LLMs as programming tutors for students. Given the capabilities there are several conclusions we can draw from this project.

Prompt Engineering

The input to a LLM has a massive effect on the quality of advice a student can receive from a model (Schmidt). As such, prompt engineering has a huge effect on the design of a student tutor centered around a LLM. The prompts for SIGHT were simply loaded as variables in the VSCode extension and prepended to the code that user typed into the IDE. Future iterations could involve more complex systems, but ultimately could all be controlled by user preferences (in a configuration file).

This flexibility would allow one to adjust the prompting of the model to fit a specific application, or change it entirely based on ones needs.

Modelling

SCENT relied on the blackboard, and later for a more complex student model. SIGHT in its current iteration employs only a level for the programmer: beginner, intermediate, or advanced. The difficulty with using a LLM for output is the unreliability. As mentioned in the previous section, LLaMa2 would regularly output answers that didn't make sense, symbols, or incoherent thoughts. Along with not being useful, these do not translate well to a programmed student model. A string of text can represent an idea, but evaluating that in the context of a student's performance is difficult.

There do exist techniques to automate this task as well. Supervised machine learning can be employed to learn features and attributes of a student's work that classify their level of skill or understanding (Hung Chau). This would allow for teachers to specify what features they are looking for and allow a computer to learn the attributes of a program that predict those features (ultimately scoring a student's work based on this function).

This also brings into question the topic of knowledge representation. SIGHT's modeling is very basic, but if the tool is to advance it needs to be able to represent the depth of a student's knowledge more precisely. It has been proposed that tutors can operate without a student model (Gugerty) however the transparency that is desired from a system of this type is only truly achievable if the teacher has some control over the model of the student. Being able to understand why a student is making a mistake, and where their knowledge is lacking, is key in properly advising and nurturing their learning. The downside to LLMs is their inability to explain their actions, and by combining this black box with a student model that is explainable, we may be able to maintain some level of transparency.

Intermediate Storage

SIGHT currently relies on text files, containing the conversations with LLaMa2 to store the relevant information about the current programming session. Keeping track of a student's previous work, the progression of their ideas, and the representation of the task is difficult for LLMs. While a advisor could rely on the context (or the memory) built into a model, this is currently a massive limitation. LLaMa2 (and most other prominent models at the time of writing) have a context limit of around four thousand tokens (words). This means that after four thousand words, the model will essentially forget everything that was said. For longer sessions this is not desirable; and if an advisor is to rely on this context to store the student model and knowledge base, this will run out quickly.

As noted earlier, these models will continue to advance and with it the context will increase. However more research is needed into both the efficacy, and practicality of this technique. Alternative methods have been proposed (Hung Chau) that would automatically update given a student's progress on a task. More work is needed in general to advance this process though.

Conclusions

LLMs provide an incredible tool when used properly and could enable the future of AI in education to advance to levels once only dreamed of. If every student could be provided their own personal programming tutor, their performance would see a drastic improvement (Bloom).

Alternatively, many are worried about the effects that these tools could have on students, with potential misuse of the tools leading to students not performing assignments. However, as this project and many others show, with the correct guiderails in place these tools can be incredibly powerful. A tool of this type could become the new norm in the classroom. This would open doors for students of all backgrounds and is the future AI in education should strive for.

Student modeling needs to be refined to create a flexible, yet transparent model of what a student is capable of, and what their goals are. Knowledge representation also needs to advance to the level that storing this information can be formatted in both a human readable, and machine processable medium. Finally prompt engineering is an emerging field with much to offer. Given the right inputs, these LLMs could provide insightful feedback to programmers everywhere, and could make a tool like this of vital importance to the progression of the modern programming student.

Tools like SIGHT are necessary however, to ensure a student is not just searching for answers, but rather getting helpful feedback and being guided to discovering the correct solution on their own. Ultimately the goal is to improve a student's understanding of the given programming language, and the safeties necessary to make LLMs suitable for this task can be baked into a tool like SIGHT.

Bibliography

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. "Attention is All you Need." *Advances in Neural Information Processing Systems*. NIPS, 2017. 30.

Bloom, B. S. "The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring." *Educational Researcher* (1984): 4-16.

ggerganov. *https://github.com/ggerganov/llama.cpp*. 2023.

Gordon I. McCalla, Richard B. Bunt, and Janelle J. Harms. "The design of the scent automated advisor." *Computational Intelligence* (1986): 76–92.

Greer, Gordon I. McCalla and Jim E. "Granularity-based reasoning and belief revision in student models." *Student Modelling: The Key to Individualized Knowledge-Based Instruction* (1994): 39-62.

Gugerty, Leo. "Non-Diagnostic Intelligent Tutoring Systems: Teaching without Student Models." *Instructional Science* (vol. 25, no. 6, 1997): 409–32.

Hung Chau, Igor Labutov, Khushboo Thaker, Daqing He & Peter Brusilovsky. "Automatic Concept Extraction for Domain and Student Modeling in Adaptive Textbooks." *International Journal of Artificial Intelligence in Education* (2020): 820–846.

Meta AI. "Llama 2: Open Foundation and Fine-Tuned Chat Models." *arxiv* (2023).

Mulyar, Yuvanesh Anand and Zach Nussbaum and Brandon Duderstadt and Benjamin Schmidt and Andriy. *GPT4All: Training an Assistant-style Chatbot with Large Scale Data Distillation from GPT-3.5-Turbo: GItHub*. 2023.

Schmidt, Jules White and Quchen Fu and Sam Hays and Michael Sandborn and Carlos Olea and Henry Gilbert and Ashraf Elnashar and Jesse Spencer-Smith and Douglas C. "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT." *arXiv* (2023).