CS100 - Virtual Teaching Assistant

Problem Motivation:

The problem we have identified is the special need for beginners at coding, with particular emphasis on students of CS-100 at LUMS, in receiving much needed assistance with their coding concepts, critical thinking and developing problem solving skills. Such beginners to programming have often not built the algorithmic approach towards solving problems, and the ease of access to multiple LLMs has only worsened the situation. More often than not, students find it easier (and more beneficial in the short term) to obtain the final solutions without ever building the problem solving capabilities that are essential for developing as a successful programmer. This, coupled with the lack of possibility of 24/7 availability of teaching staff leads to at least one, if not all students leaving the course with a decent enough grade and next to no learning. The potential impact of a solution to this problem will strive to help beginners to develop the much needed algorithmic approach to problem solving, by forcing them through the "mental gears shifting" process so that the students leave such courses with some form of learning, and confidence on how to approach unseen problems.

Goal of the Project:

We hope to devise a model, which contrary to the GPTs and other LLMs on the web, does not provide the final solution on one request, but rather serves the role of a virtual teaching assistant which can guide, quiz and nurture the fundamental coding concepts and problem solving skills, whenever the students require help in solving their coding problems, be it with syntax, or logical hints on how to approach a problem. Seminal work has already been done in this regard, primarily focusing on Python (CodeHelp and CodeAid). However, little work is done focusing on C++ or other lower level languages. We hope to use this project to fill this gap.

The primary audience of our model will be the students enrolled in CS-100 at LUMS, especially those from a non-CS background. Our work focuses on making C++ the main language for assisting, since CS-100 is in C++. In this regard, we limit our approach to addressing the following:

1) guiding on how to solve the requested problem in an appropriate difficulty that is expected from the problem [i.e. attempting the problem by neither too easy concepts (such as using arrays to store multiple names just for the sake of brute-forcing the problem) nor too difficult concepts (such as OOP for the sake of OOP)].

2) similar to 1), evaluate whether the user presents as his final understood code solution in an appropriate manner by comparing against certain reference response codes generated (possibly by incorporating other LLM APIs like CodeLLaMa, fine-tuned GPT)

3) taking multiple quizzes while guiding to the solution of the problem based on informed responses to gauge whether the student gets the concept or not, and providing multiple responses in different styles to give another intuition if they don't.

4) giving a similar problem to the one just solved as an additional exercise to gauge understanding

Solution Approaches:

CodeHelp (page 5, Fig 4) details a pipeline for its response. We plan to use this as a baseline:

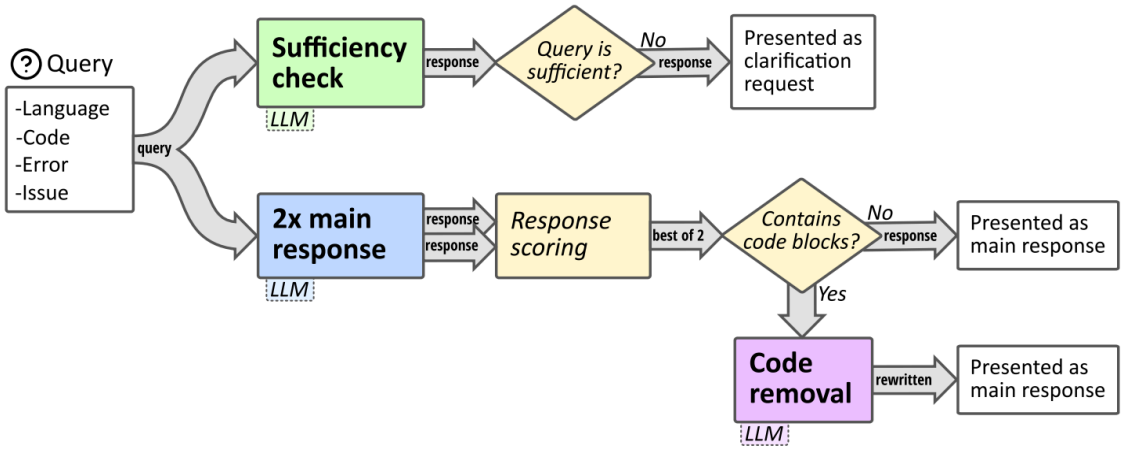


Figure 4: CodeHelp's response workflow. Steps using a large language model completion are tagged *LLM*.

There are 3 streams highlighted:

1) A query sufficiency check stream where an LLM is used to gauge whether the query is clear enough for a sufficient response

2) A code removal stream which checks if the main response has code, if it doesn't, it is forwarded as is, if it does, the code is removed and response is rephrased to remove any mention of code.

3) The response scoring stream which pools multiple response and chooses the best one out of them

We plan to take the initial query and pass it through a checkpointing stream that devises a multiple step response based on the initial query (i.e. the best approach to make the user interactively reach the problem solution by detailing checkpoints to be achieved in the form of user response). This can be done by feeding the main response from the initial query back into another LLM and fine-tuning it to break the response into multiple checkpoints. The first of these responses is then fed into the LLM (with or without the main response (TBD)) to generate the first question/exercise/explanation that the model presents to the user. This will be presented as

the main response. Subsequently, the main response and the user input is fed back to the checkpointing stream which removes the stored checkpoints (which have been achieved)/ adds new ones as a recurrent update, and forwards the next checkpoint to be addressed. This might require appending the whole chat stream and feeding it back, or only maintaining a small amount of history (previous two-three query-response pairs).

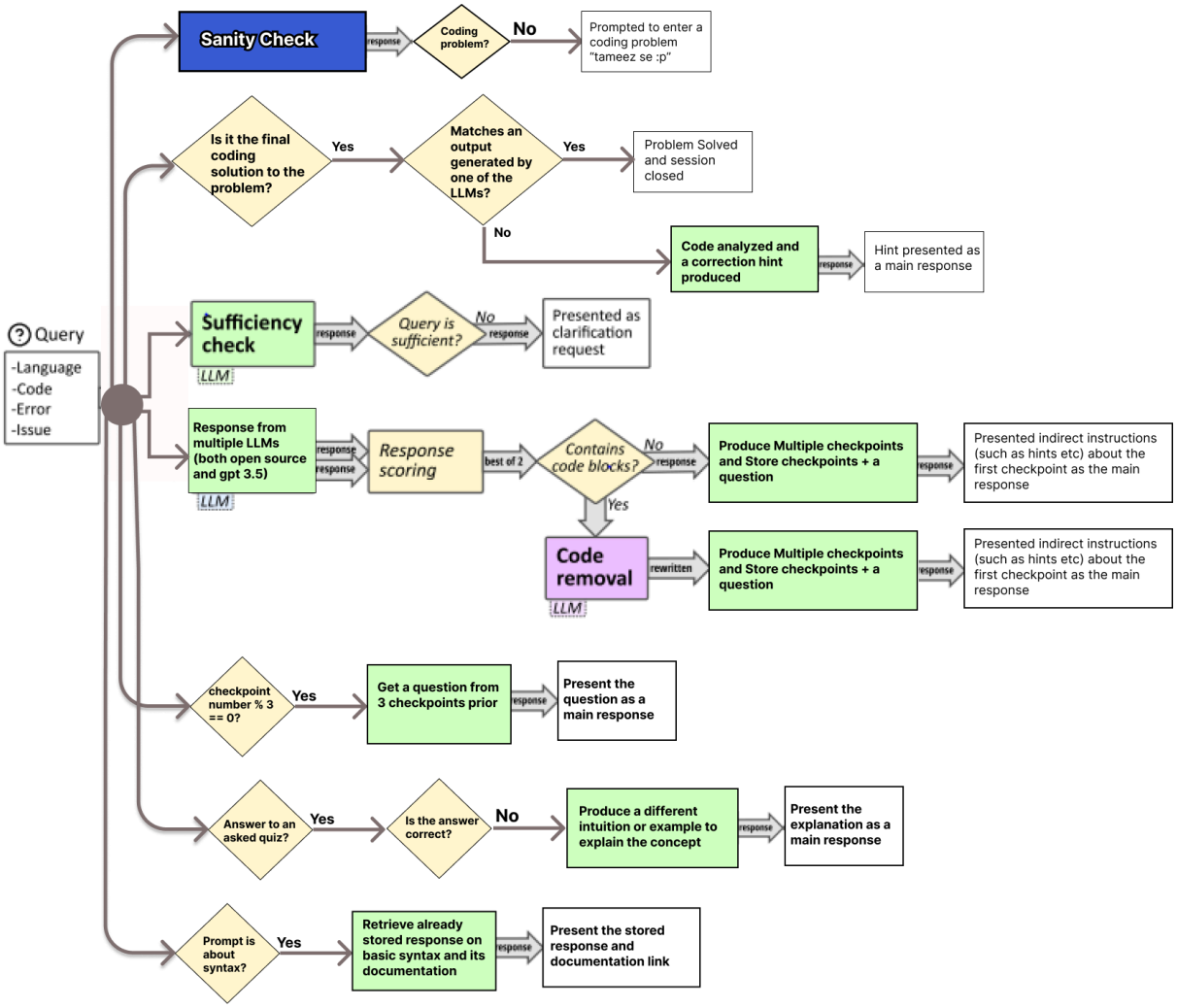
An initial checkpoint might be to prompt the user on how he wants to approach the problem, feedback the response-query pair, update the checkpoints, and move on to the next checkpoint. This process can be repeated until a solution is achieved (without any mention of code, or pseudo code, at max) which can be indicated by user input or the user provides a complete correct code without any additional query (the typical response of GPTs in this case would be to explain the code). The analysis of a correct code given an input code by the user can simply be done by maintaining a list of “correct responses” which can include the codes removed in the first step or through additional code generated at the time of input through LLMs like GPTs and CodeLLaMa. This means an addition of the following streams:

1) A checkpointing stream which analyzes which checkpoints have been achieved/ need be added and forwards the one required in sequence

2) A feedback stream that maintains history of a few (2-3) or more response-query pairs to allow the checkpointing LLM to deduce which checkpoints have been achieved

3) A reference code production stream which produces a reference code to be compared against in case the user gives a code based input with a query of analyzing the code.

The main challenges in this part are in fine-tuning the multiple streams which may harness one or more LLMs to work in unison. Our approach is to first replicate the CodeHelp paper by fine-tuning on C++ data/problems instead, followed by (or working in parallel) to add the three additional streams outlined above. The final envisioned pipeline (as described above) can be detailed below:



Envisioned pipeline

Our model will primarily use large LLMs (40B+ parameters) . This reason for such selection is the diverse and multifaceted nature of challenges faced by CS-100 students at LUMS. These challenges extend from understanding intricate coding concepts to developing effective problem-solving skills and responding to open-ended queries. By utilizing LLMs with a substantial parameter count, the virtual teaching assistant is equipped to provide nuanced and comprehensive assistance across a broad range of topics. This ensures that students receive personalized and in-depth support, irrespective of the complexity of their queries or problems.

We plan to use a combination of both open source and proprietary models, with the hope that we can continue with open source models only provided that the quality of the response is not compromised. For the generation of the reference final expected code, we plan to integrate CodeLLaMa and similar code generation models, by fine tuning them on C++ problems, documentations, and tutorials, ensuring precise and relevant final response. These might be pooled or consulted individually to verify the accuracy of the user's final answer. We will mainly be fine tuning such pre-existing models, however we also plan to employ RAG in handling

straightforward inquiries, such as basic C++ syntax or introductory concepts like loops and functions. By utilizing pre-existing knowledge bases, RAG retrieves relevant information, aggregates it, and generates concise yet accurate responses, ensuring students receive clear explanations essential for building a robust programming foundation. Conversely, fine-tuning LLMs like the GPT-3.5 API is ideal for addressing more ambiguous or open-ended prompts.

Our goal is to create a virtual teaching assistant that is not only effective but also practical and user-friendly for CS-100 students at LUMS. To achieve this, we will design an intuitive and visually appealing user interface (UI) to make navigation simple and straightforward, as well as adding feedback mechanisms for students to rate the quality of assistance and provide suggestions for improvement (RLHF maybe??).

Rough Timeline:

Week	Checkpoints to achieve
1	**Data collection for C++ specific tuning**
2	**Replication of baseline model**
3	**Replication of baseline model + RAG based training**
4	**Implementing Quiz functionality + Sanity Check**
5	**Final Code Check against Reference Functionality + Final Checkpointing Testing**
6	**Deployment**
7-8	**Contingencies**

References:

https://arxiv.org/abs/2401.11314 (CodeAid)
https://arxiv.org/pdf/2308.06921 (CodeHelp)

Questions Answered (To be Condensed):

The proposed solution for the CS-100 course at LUMS involves a combination of advanced language models (LLMs), particularly the GPT-3.5 API and OpenAI's GPT models. Trained over 175 billion parameters, the GPT-3.5 API performs exceptionally at generating human-like text for various tasks, such as query comprehension, checkpoint generation, and feedback provision. Additionally, the suite of OpenAI's GPT models, including GPT-3, can be leveraged to fine-tune the assistant on C++ problems, ensuring precise and relevant guidance for students. The integration of CodeLLaMa, a tailored LLM for programming tasks, further strengthens the assistant's capabilities by offering support with C++ code syntax, logical reasoning, and problem-solving approaches. CodeLLaMa also aids in producing reference code snippets for comparing and evaluating student solutions. In essence, this combination of LLMs equips the virtual teaching assistant with the tools to provide comprehensive and tailored aid to CS-100 students at LUMS, facilitating the cultivation of algorithmic thinking, problem-solving acumen, and an enhanced understanding of C++ concepts.

Do you aim to focus more on proprietary models or open-source models?
Both, will try open-source first to develop learning and then later integrate gpt 3.5 api for more quality final product

The project's initial emphasis is on open-source models, prioritizing the creation of a robust foundation and learning structure. The decision to utilize open-source models is informed by their abundant resources and supportive community. This aligns with the project's objective of nurturing a collaborative learning atmosphere. By capitalizing on the large user base and collaborative nature of open-source models, the project benefits from a broad spectrum of viewpoints and contributions.

As the project progresses, the goal is to integrate proprietary models like the GPT-3.5 API to enhance the quality and functionality of the final product. Proprietary models such as GPT-3.5 API have state-of-the-art capabilities and can generate high-quality human-like text across various tasks. The central aim of integrating these models is to refine the virtual teaching assistant's ability to provide accurate and nuanced guidance, thereby enhancing the learning experience for CS-100 students at LUMS.

By combining the strengths of both open-source and proprietary models, the project seeks to create a robust and effective virtual teaching assistant that offers comprehensive and personalized assistance to students. This approach facilitates their learning and development of algorithmic thinking, problem-solving skills, and a deeper understanding of coding concepts in C++.

What techniques do you aim to use (e.g. RAG, fine-tuning, training from scratch), and how do you justify them?
Our strategy involves employing techniques such as RAG and fine-tuning. RAG is suitable for addressing straightforward queries like basic C++ syntax or introductory concepts such as loops and functions. Conversely, fine-tuning large language models (LLMs) is justified for handling more ambiguous or open-ended prompts.

Our strategy encompasses a dual approach, incorporating both the RAG (Retrieve, Aggregate, Generate) technique and the fine-tuning of large language models (LLMs) to cater to the diverse range of queries and challenges encountered by CS-100 students at LUMS. RAG proves effective in handling straightforward inquiries, such as basic C++ syntax or introductory concepts like loops and functions. By utilizing pre-existing knowledge bases, RAG retrieves relevant information, aggregates it, and generates concise yet accurate responses, ensuring students receive clear explanations essential for building a robust programming foundation. Conversely, fine-tuning LLMs like the GPT-3.5 API is ideal for addressing more ambiguous or open-ended prompts. This process involves training the LLM on specific tasks, such as providing guidance and feedback on C++ coding problems, thereby enhancing its proficiency in generating detailed and contextually relevant responses to complex queries or problem-solving tasks. Through the synergistic application of RAG and fine-tuning techniques, the virtual teaching assistant delivers comprehensive and personalized support, adeptly addressing both fundamental and advanced challenges, thereby enriching the learning experience for CS-100 students at LUMS.

How large are the models you aim to use, and how do you justify this in terms of the problem scope?

Our aim is to deploy cutting-edge large language models (LLMs), particularly focusing on OpenAI's GPT-3.5 API, which has over 175 billion parameters. This reason this particular model was chosen is owed to the diverse and multifaceted nature of challenges faced by CS-100 students at LUMS. These challenges extend from understanding intricate coding concepts to developing effective problem-solving skills and responding to open-ended queries. By utilizing LLMs with a substantial parameter count, the virtual teaching assistant is equipped to provide nuanced and comprehensive assistance across a broad range of topics. This ensures that students receive personalized and in-depth support, irrespective of the complexity of their queries or problems.

How do you aim to make your work practical in terms of usability for the end-user?

Our goal is to create a virtual teaching assistant that is not only effective but also practical and user-friendly for CS-100 students at LUMS. To achieve this, we will focus on several key areas. Firstly, we will design an intuitive and visually appealing user interface (UI) to make navigation simple and straightforward. Secondly, we will ensure the assistant is accessible to all students, including those with disabilities. Thirdly, we will ensure the assistant is available 24/7 for students to access assistance whenever needed. Fourthly, we will incorporate a feedback mechanism for students to rate the quality of assistance and provide suggestions for improvement. Additionally, the assistant will offer multimedia support, like interactive exercises/quizzes, to accommodate different learning styles. Lastly, the assistant will seamlessly integrate with existing learning management systems at LUMS, ensuring students can access assistance within their familiar learning environment. Through these efforts, we aim to make the virtual teaching assistant practical and easy to use, providing a positive and effective learning experience for CS-100 students at LUMS.