Michael Mills
Senyo Ohene
October 25, 2023
CS5100
Foundations of Artificial Intelligence

Artificial Intelligence Final Project Proposal

Project Title

The title for our project will be "Autonomous Code Generation: A Novel Approach"

Problem Statement with Literature Review

Programming is a very complex and time consuming task. It can take days, weeks, months, if not years to complete something that could be considered a simple task. Companies all over the world spend enormous amounts of money hiring software developers to write code. For example, in the research paper "An analysis of errors and their causes in systems programs," it is suggested that one of the best ways in which we can minimize mistakes is by automating as many tasks as possible. In other words, we are able to minimize the amount of mistakes we make and thus the amount of time it takes to write a certain amount of code by automating the process of writing code as much as we can.

In addition, typical humans are very likely to make minor mistakes in the code that will prevent it from running properly, thus increasing the amount of time it takes to write. In the article "A framework and methodology for studying the causes of software errors," the amount of time it takes to fix different types of errors in code is carefully outlined, and it is evident that it certainly takes a significant amount of time to debug code.

In summary, debugging can be a very monotonous task of correcting the code and running it to see if the bug has been fixed, and researching more ways to fit it if it hasn't been fixed. This process, which is essentially a loop that will not exit itself until it has solved the problem, can certainly be automated. This is very interesting because programming is very closely related to innovation, technology, and how quickly new and innovative ideas can be developed. In a software engineering project, if the amount of time it takes to get from point A to point B can be minimized dramatically, this can greatly improve the rate at which new and innovative products can be built.

Problem Analysis

The state space for our system is a string of characters, which comes from the user-inputted prompt. This is reasonable because this is directly related to what we are looking to do with our project. We are looking to convert a prompt that the user inputs into completed code. The state space can be thought of as an infinite space, given the fact that there is an infinite number of concepts and constructs that can be generated from user prompts.

In terms of how the system will move from one state space to the next, let's consider the overall lifecycle of how the model will be trained and what it will look like as it is being trained. First, the system will be in a state where it is not able to generate any code. Next, it will transition to a state where it is able to generate code, but it is not correct and is definitely not reliable. Next, it will transition to a state where it is able to generate code that is sometimes accurate, but still not reliable. Finally, it will transition to a state where it is able to reliably generate code. This will primarily come from the training of the AI model on the dataset, but we are also hoping to add a splash of reinforcement training as well.

These transitions are stochastic, due to the fact that we are not able to predict where it will move. This is due to a lot of factors, such as how complex the code is, the data that it is trained on, and the overall random nature of how the model will generate code.

In terms of how real time corresponds to one tick of the state-space clock, it can be argued that one tick will be equal to one iteration of the training.

Because one epoch is equal to the amount of time it takes to train itself one time using the entirety of the training data, we can say that we are unable to determine the numbers of epochs it will take to properly train the model. We are planning on using functionality in tensorflow that will allow us to automatically stop the training once the validation has reached a certain predetermined number.

In terms of the goal test, the goal test for our system is whether the code generated performs the function requested by the user. This means we can find the proportion of user-inputted prompts our system is able to generate functioning code for.

Dataset or other source materials

There are a lot of interesting places to get the data. The best way that I have found so far is to look at existing models that are able to achieve a similar task, look and see what datasets are used in those existing models, and then look to see if I am able to download the dataset. I have been primarily successful in doing this through a website called Huggingface, which is a resource to not only look at AI models, but also the datasets that are used to be trained with.

The first step in converting the data into something that is usable is to clean and preprocess the data. This will consist of removing anything that is unnecessary, could be detrimental to the overall success of the model, or information that is simply not relevant to what we are trying to accomplish.

Once the first step is done, the next step will involve training the data. We will use a number of datasets in order to train this model. One of which will be The-stack-dedup dataset. This is an incredibly large dataset that supports a large number of languages. We are looking to extract only Python, and perhaps some more essential languages such as HTML, CSS, Javascript, C++, Java, Typescript, SQL.

Deliverable and Demonstration

We will produce a system that takes a user-inputted prompt for a program function, and outputs (a) line(s) of code that performs the requested function. We will demonstrate this by having the class request our system to write some type of code, and have our system generate the requested code, and test it out. If we have enough time in the development process, we are also looking to implement multiple agents, where there will be a number of agents that will be constantly looking to improve the code, write tests, write documentation, solve bugs, etc.

Evaluation of Results

In order to evaluate the model, we will primarily be using human evaluations. In other words, we will give the program a prompt, and look to see if the model is able to accurately complete the prompt. For example, if we give the AI model a prompt such as “write a block of code in python that will print “hello world” in the terminal” we will expect the output of the code in the terminal to print “Hello World!” We will also look at the models ability to complete code without bugs or errors. We will collect numerical data on the amount of times that model is able to generate code without any errors.

Major Components and Schedule

As for the major components and schedule, there are three major components that are evenly divided across the remainder of the semester. The first component will consist of training the model, tokenization, parser, and genuinely making sure that the actual model, or the file that will resemble the model. Is a good starting point for the project. We will set the due date for this portion of the project to November 7.

As for the second component of the project, we are looking to have an application that is built around the model such as a terminal interface for a user to interact with the model and generate predictions. At this point, we will also work on some type of feedback and learning mechanism, autonomously run the code to make sure there are no bugs or errors, loop through with multiple agents. We will want to have this portion of the project completed by November 21.

As for the third component of the project, we will want to develop testing strategies to figure out how good it actually is. This is a point where we will be tweaking the model in various ways in order to make it better. We will want to fix bugs and logic error.s Perhaps, we will want to manipulate the dataset and how it’s trained in various ways that can allow the model to be more accurate. We will have this portion of the project completed by December 5.

References

Hugging Face. "bigcode/the-stack-dedup." Hugging Face, 2023. Web. Accessed 2023-10-30.

Hugging Face. "togethercomputer/RedPajama-Data-1T." Hugging Face, 2023. Web. Accessed

2023-10-30.

Hugging Face. "iamtarun/python_code_instructions_18k_alpaca." Hugging Face, 2023. Web. Accessed 2023-10-30.

Hugging Face. "sahil2801/CodeAlpaca-20k." Hugging Face, 2023. Web. Accessed 2023-10-30.

Karpathy, Andrej. "nanoGPT." GitHub, 2023. Web. Accessed 2023-10-30. Karpathy, Andrej, et al. "Large Language Models are Few-Shot Learners." arXiv preprint arXiv:2201.08237 (2022).

OpenAI. "Language Unsupervised: A Unified Approach for Few-Shot, Zero-Shot, and Multi-Task Learning." OpenAI Blog, 2022. Web. Accessed 2023-10-30.

TensorFlow. "Natural Language Processing with TensorFlow." TensorFlow, 2023. Web. Accessed 2023-10-30.

TensorFlow. "A Primer on Text Generation." TensorFlow, 2023. Web. Accessed 2023-10-30. Vyas, Nitish. "Know About Zero-Shot, One-Shot, and Few-Shot Learning." Analytics Vidhya, 2022. Web. Accessed 2023-10-30.