Michael Younger

# MAT 422 Project Plan

**Introduction:**
LLMs are making headlines about being really good at code, alas they are only actually very good at simple coding that only has a couple hundred lines tops. However an AI agent may be able to fix that. If the LLM was prompted to write code to simulate the evolution to a schrodinger equation for instance and was able to then take a look at its output and then update the code if it does not work and finish its process if it does. AI agents are LLMs that have access to certain defined functions that they can then use to do certain tasks much better. A calculator for instance where the ai requests an expression to be evaluated and returns the output. This is nice because LLMs synonymously struggle to do basic math as the values increase to large integers, but when said LLM has access to a calculator this error goes away. So then, is there a way to use LLM agents to generate better and more complex python code without human input?

**Related Work:**
Construction of agents themselves is more or less the easy part. An article written by Incletech Admin on Medium demonstrates how to create a simple model. To construct an agent an LLM is downloaded and a home server is set up, commercial servers with chatGPT also can be used. From here we have a main directive that dictates are instructions to the LLM. For example if the directive says that a calculator exists and can be used to attain results from using the command

```
USE: calculator
ACTION: “input equation”
```

Then when the AI want to calculate 2*2 it will print

```
USE: calculator
ACTION: “2*2”
```

This will result in an output (4) that will be fed back into the LLM and will then be printed as the result[1].
The former is a simple example, however this only works somewhat. Agents are heavily reliant on the LLM behaving as anticipated. What if instead of USE:calculator\nACTION:”input equation” it leaves out the “\n” and results in everything being stated on a single line. Of course coding around that would be a simple problem, but the big picture here is that coding for every edge case is a waste of time when working with LLMs because it is impossible without knowing the output beforehand. There are other issues as well, namely that there is a limited context window. Currently the AI that was demoed in the above article uses an array that is continuously updated to feed back into the LLM, eventually the stack buffer and/or the AI will no longer be able to hold onto all of the information in the string forgetting or losing potentially important details also known as having a finite attention span. Instead there are other ways of getting the LLM to behave more amicably. Reviewing a paper written by Lilian Weng, Titled LLM Powered Autonomous Agents[2] we can review a few different methodologies that can aid with this process. First, when it comes to memory there are 2 kinds long and short. Short term memory can be equated to our context window, our long term memory we can create by adding a tool that can

store information that the AI finds important for later there are multiple methods of doing this outlined in the paper with varying degrees of memory retrieval speed, and Scalable Nearest Neighbors was by far the fastest with all other methodologies outlined in the paper roughly the same speed. Behavior on the other hand is a bit more complex. Self reflection is one of the first methodologies reviewed in the paper. This method instructs the AI to reflect on its action and determine whether or not it likes its result given an input heuristic that is calculated from the previous result, then if it does not like its previous answer is able to try the same task again. In the paper self reflection caused greater successes overall.

**Proposed methodology:**
In this project anaconda python will be used to code this project using a locally installed llama 3.8B from their AI navigator. From here the AI will be given the following list of tools:
Calculator: To better self check desired outputs
Google search: To be able to look up things it needs relevant information
Write To Python: This function will write the code produced into a python file named AI.py
Test Output: This function will run a batch run command on the created python file and output its output to a txt file called output.txt
The AI will be instructed to code something specified by the user using the above tools, the write to python function will automatically run test output as a return, which will be reflected upon by said AI to know if it did a good job. Once the AI thinks it has done a good job it will end its test loop. There will also be another AI to test out Scalable Nearest Neighbors method of long term memory with reflection to see if there are speed and accuracy improvements to the agents as seen in the papers.

**Experiment setups:**
There will be 3 LLMs used, one that is the base LLM that will take its produced code and print it to a python file. One LLM agent that will have reflection based on the output and the last LLM which will also have some form of long term memory. They will be instructed to complete the following tasks each with an increasing level of difficulty (from a human perspective anyway):
Code a classical implementation of Shor's algorithm (Easy)
Code a python script that can propagate a schrodinger equation using the Numerov method and animate it (Medium)
Code a machine learning algorithm that can classify butterflies using provided dataset in a folder titled: butterflies, with both train and test CSVs and folders (Hard)
These tests were chosen arbitrarily and will be increased in difficulty should all models pass. Each will be graded according to whether or not the code works and the time it took to arrive at an answer. If the code does not compile or work as intended the agents will be instructed to try again, every try is +1 error points and at 5 points this process will stop. Agent with the fewest error points will win.

**Expected results:**

The LLMs with reflection are expected to perform the best with the additional long term memory LLM expected to be faster and more accurate than their brother, but slower than the base model output. The results here will tell us at what complexity of coding task is there an improvement in code or if there is not any at all.

## Works Cited

[1]Incletech Admin *et al.,* “How to Create your own LLM Agent from Scratch: A Step-by-Step Guide,”https://medium.com/@incle/how-to-create-your-own-llm-agent-from-scratch-a-step-by-step-guide-14b763e5b3b8
[2]Lilian Weng *et al.,*”LLM Powered Autonomous Agents”https://lilianweng.github.io/posts/2023-06-23-agent/