Progress Report 1:

My prior project plan was off base from what was really achievable/valuable, so I had to change my goals and project structure a bit. Apologies for being late on this, but I think it is for the best for my final delivery so I'm OK with taking the penalties on this intermediate assignment. Anyway, I think I was wrong about the fundamental goal of doing next-line prediction for my project. While going the next-line prediction route is slightly more interesting due to the idea that the computer should "understand" the logic of your code, I don't think it is really achievable for a few reasons. First of all, while there are some models/software that offer next-line prediction I don't think they are accurate enough to really be interrogated with any rigor from a practitioner point of view. Second, the current software that offers next-token prediction seems way better to me and is much more reconcilable with normal coding practices. I've updated my views on the creation of logic as much more difficult than understanding patterns in algorithms. I was previously thinking of these two things as conceptually similar, where understanding the structure of an algorithm must be akin to developing next-line logic, but I think I was fundamentally off on that. While they may be similar with respect to long term dependencies between data and control, I think the creation of program logic will remain the domain of human coders for a while longer due to the need for fundamental understanding of the purpose of proximal lines of code. Also, I think there is a need for some representation of the target of the program to be provided to the AI program because that is a giant piece of information that is always stored in the mind of the programmer but may actually be the last line of code in the sequence, and thus not accessible to the AI program during production of that code. With all that said, I'll now concretely go through my new project goal and my status on the project right now:

Main Goal: Do a User Study of some popular tools in production versus academic models for code completion. My goal is to evaluate the value-add of these various tools and note their strengths and weaknesses in production. I am switching to focus only on next-token prediction because that seems to be the focus of industry autocomplete software and much more realistically integrated into modern development procedures. I think this is fairly novel in that I will be evaluating the performance objectively and subjectively across many types of coding that I have done/continue to do. I hope this uncovers some potential issues with training set coverage of these various types of projects or shows a clear winner in terms of helpfulness.

Comparison Subjects: I want to review Tabnine, Kite (and will include the google Colab autocomplete if time allows because I've found it to be really helpful). As for the academic models, I'm going to spin up the GPT-2 transfer-learned model (state of the art performance) and CodeGPT (very similar performance) as the main comparisons. Interestingly enough, in the research I did about the Tabnine and Kite machine learning model behind the products, the Tabnine model actually claims to utilize GPT-2 but likely trains it on even more data as it says it was trained on “millions of files from GitHub”.

Datasets: I plan to try all of these models with only Python code (the language I am most comfortable with and is supported by all these programs). However, I plan to focus on 10 code examples across 5 different types of python code:

1. Tensorflow
2. Pandas data analysis
3. Scikit-Learn ML Pipelines
4. Flask/Django app creation
5. Plotly visualization

I suspect there will be big differences in performance across these various tasks, basically due to the training data used to build the models. I hope to get some idea of the training data for the academic models, and I have emailed the TabNine and Kite companies asking a bit more about their process for creating their ML models. Not sure if I’ll get any response there, but any info would be helpful.

For this data, I plan to use 1 code example for each category from my own code/projects and 1 code example from publicly available code on GitHub to make sure my coding style isn’t the thing messing up the performance of the models. In each file, I will target several places in the code to get the prediction of the next token (~5) and will obviously keep those locations consistent across all evaluations. I plan to target areas of the code that are important logical steps (as opposed to setup of functions or objects) and will evaluate for correctness of suggestions as well as a more subjective evaluation for “helpfulness” on whether the suggestions would help me find ways to implement that logic.

Where am I getting the code to run on? And how am I going to run through this code?

As discussed above, I’m going to basically pick from various old projects of mine and publicly available code on GitHub. I am going to make the decision on which points to evaluate the next-token prediction primarily on my perceived difficulty of that prediction. I want to pick areas that I think I would actually need help thinking about how to do it instead of simply improving my typing speed. I think this evaluation is unique because if a model can help you “think” through the difficulty of your logic then that is a step function improvement on coding efficiency instead of incremental improvement on efficiency by auto-filling what you already knew you had to write.

How am I going to evaluate the performance? Subjective & Formulaic. I plan to do they typical formulaic eval on if the top 5 suggestions contain the actual code. This will be a binary performance indicator – not accounting for rank of the suggestions unless the models are very close in this performance. As for the subjective performance evaluation, I'll take the suggestions and think to see if any of them point me in a possibly correct direction and evaluate usefulness of the suggestions on a 1-10 scale.

Value it will bring: I hope to show performance on actual code (possibly buggy/imperfect) and evaluate the models' helpfulness in aiding my programmatic thinking rather than simple correctness. I really think the goal of contributing to programmer's logical development is the end goal of AI4SE and I want to be able to give an estimate of the current value-add from major academic models and production software. Lastly, I hope to be able to provide suggestions on types of projects to be added to the training sets of these models such that they can be more helpful with various common python projects.