Rowan Miller
5/8/2025
Understanding Large Language Models
Final Project Report

The power of AI in understanding code is astonishing. However, a simple "plug-and-play" approach is often a poor method of utilizing a model. For more narrow use cases, fine-tuning a model is a highly effective approach. To demonstrate this, I fine-tuned the `CodeT5-small` model by Salesforce to identify and correct errors in Python code. The code for my project can be found in the following GitHub repository: LLM-Final

Bug fixing is an ever-present weight on developers. The majority of my time working on this project was spent fixing bugs! I will build off of an existing model designed to parse and understand code and fine-tune it to fix Python bugs, an enormously useful skill. The model will need to be able to correctly identify bugs beyond those of the training data and make correct changes to bugs it finds. I chose Salesforce's `CodeT5-small` model on HuggingFace because it is designed with fine-tuning in mind. The model has 60,492,288 parameters, uses `float32` for instructions, and has 6 layers. It is a T5 model developed by Salesforce in 2021. When run on 100 test prompts, the model increased CPU utilization by about 20% for the 16 seconds it took to run. In order to fine-tune this model, I used the dataset `python-bugs-name-noise-1` by HuggingFace user linyalan. It is a set of 1000 examples of Python code, with a column for buggy code and another for the fixed version of the same code. I split this dataset into 700 training examples, 200 validation examples to use during fine-tuning, and 100 test examples to evaluate the model before and after fine-tuning. Many bugs in the set are subtle, such as the very first data point:

`correct_code`	`prompt_code`
`def place_types(self): """https://familysearch.org/developers/docs/api/places/Place_Types_resource""" return self.places_base + "types"`	`def place_types(self): """https://familysearch.org/developers/docs/api/places/Place_Types_resource""" return self.places_base / "types"`

In this case, the bug is in using the / operator instead of + in the return statement. This code is perfectly runnable, but will not function correctly, making it a bug that would slip by simple bug detectors.

Before fine-tuning, the `CodeT5` model performed exceptionally poorly. It failed to create a complete response to the prompts provided to it. I believe it was unable to understand the specific task that was required of it when simply being presented with runnable, but flawed code. I calculated an overall BLEU score for the model of 0.00348, an abject failure and little to no similarity between the model's output and the correct code. The model's response to the example from above was as follows:

correct_code	prediction
`def place_types(self): """https://familysearch.org/developers/docs/api/places/Place_Types_resource""" return self.places_base + "types"`	`self.person_base`

In order to improve this performance, I used a `Trainer` object from the HuggingFace `Transformer` library, set its parameters, then trained the model on my train dataset. I started with parameters suggested by ChatGPT as a baseline. Each training epoch output a Training Loss and Validation Loss, showing the ability of the fine-tuned model to accurately predict examples in the training and validation data respectively. After 5 epochs, the Validation Loss ceased to decrease significantly and even began to increase as the model became overfit to its training data. To mitigate this, I set epochs to 5 for future testing. I also determined 0.00005 to be an appropriate learning rate that mitigated overfitting while minimizing both Training and Validation loss.

After fine-tuning a few times and adjusting training parameters, I had a satisfactory model. This is saved in the `bugfixer-finetuned` folder of my repository, and can be loaded without training by running box 3.4 of my Jupyter notebook. I then tested my model with the testing dataset and compared its outputs to the correct responses in the test set with BLEU as before. The fine-tuned model boasted a BLEU score of 0.7097, a much more reasonable resu;t than the base model. In the first test example, the fine-tuned model correctly changed the / operator to + as seen below:

correct_code	prediction
`def place_types(self): """https://familysearch.org/developers/docs/api/places/Place_Types_resource""" return self.places_base + "types"`	`def place_types(self): """https://familysearch.org/developers/docs/api/places/Place_Types_resource""" return self.places_base / "types"`

Fine-tuning proved extraordinarily effective at improving the performance of the model, but the scope of what it can do is likely quite narrow. If a user needs any explanation of a bug, my model will be unable to provide one. It also cannot take as input natural language explaining what a user knows of a bug. The model is also extremely small. Its 60 million parameters pale in comparison to the billions used for ChatGPT. Ultimately, in the competition to create the best AI models, there seems to be no substitute for a gargantuan dataset.