**CS 260 Project Report**

Name: **Puneet Singhania**

SID: **862327375,** NetID: psing088

## **Project Design:**

**Artifact:** CodeGraphBERT - a pre-trained code representation model. I chose refinement in GraphCodeBERT. Link:

<https://github.com/microsoft/CodeBERT/tree/master/GraphCodeBERT/refinement>

**Difficulty Level:** 1. Downloading the tool and running it on a subject program.

## **Design Details:** The subject program is an automated code refinement system. Its goal is to automatically resolve bugs in given Java code, helping developers cut down on the time and cost involved in bug fixing. This program is built on the model GraphCodeBERT, which has been fine-tuned for this particular task. Transformers and the tree-sitter library are utilized in this program's parsing. The program can manage multi-GPU training, train on a GPU, and optimize the model's performance over epochs. The accuracy and BLEU scores are utilized to evaluate the model's performance. Also, it is an existing program provided in the GitHub repository I mentioned earlier.

**Functionalities: Training**: The system can fine-tune a pre-trained GraphCodeBERT model utilizing a dataset of Java code. This dataset consists of both buggy and fixed Java code. The model learns to map buggy code to its fixed counterpart. **Inference**: It makes use of the fine-tuned model for performing inference on new, unseen data, and outputs the refined code by fixing bugs in the input Java code.

**Inputs:** For training, we take the input as a small subset(scale=small). This is based on function length. The dataset present in the 'data' folder basically consists of pairs of Java functions. This has a buggy function on the source side and the corresponding fixed version on the target side. For inference, the input consists of buggy Java code that has to be fixed.

**Outputs:** The system outputs a fine-tuned model, saved checkpoints(in saved\_models), and log files(train and test log files) with details about the training procedure during the training phase. The system outputs the refined Java code with bugs fixed during the inference phase.

**Project Implementation-** [**Colab Notebook**](https://colab.research.google.com/drive/1CuUByKZDHSes2nctuRDEt37RuFPV6UU0?usp=sharing)(refer Figure 2)

I upload the contents of the GraphCodeBERT/refinement path to Google Colab. I had reduced the dataset size present in data.zip so that it takes less amount of time to run the entire process as I had GPU for a limited time from Colab. I provision GPU in Google Colab by going to Runtime->Change Runtime Type and then selecting the GPU(refer Figure 1). Initially, I install the necessary dependencies through three commands: **pip install torch** installs PyTorch. **pip install transformers** installs the Transformers library by Hugging Face. It is employed for jobs involving natural language processing, such as using models like BERT and RoBERTa. **pip install tree\_sitter** installs the tree-sitter library, which assists in parsing source code.

Next, I perform the extraction of the parser and dataset files. **unzip parser.zip** and **unzip data.zip** (extracts the dataset files used for training and evaluation). I then use the command **cd parser/** to go to the parser directory. I modified the last step of the build.sh file by making it as **python parser/build.py.** I use **bash parser/build.sh** to run a shell script contained within the parser directory**.** This script is used to build or compile the parser that processes source code. Then I change the directory back to the parent folder using **cd ..** .After preparing the parser, I install a text file format conversion tool named dos2unix using **apt-get install -y dos2unix**. To make sure that the shell scripts are in the right format for execution in the UNIX environment, this tool is utilized. Then, I convert script1.sh to UNIX format using **dos2unix script1.sh** and execute it using **bash script1.sh**(refer Figure 3). script1.sh consists of commands for fine-tuning(refer GitHub) the GraphCodeBERT model on the given dataset. I reduced the batch size in the script to 10 and epochs=3. The batch size was reduced because my Colab GPU started crashing and the epochs value was reduced to execute the entire process in a lesser amount of time. I convert script2Updated.sh to UNIX format using **dos2unix script2Updated.sh** and execute it using **bash script2Updated.sh**(refer Figure 5). script2Updated.sh consists of commands for inference(refer GitHub), which is the process of taking new data and predicting the output(refined code) using the trained model.

**Results:** The provided output from executing script1.sh (refer Figure 4) shows fine-tuning a machine-learning model using the GraphCodeBERT architecture. The model is started with a pre-trained checkpoint, trained on a dataset, and its performance is evaluated using BLEU-4 and perplexity scores. When compared to reference sentences, BLEU-4 evaluates the quality of the generated sentences, whereas Perplexity examines how well the model predicts a sample. Also, we have taken around 127 examples while running the evaluation. The script keeps track of settings, training progress, and evaluation outcomes. A lower perplexity and a BLEU-4 score of 9.48 in the final evaluation show that the model improved during training. Additionally, there are a few dependencies and deprecation warnings that have no impact on the primary process.  
Next, a machine learning model was evaluated on the small dataset via the script script2Updated.sh(refer Figure 6). A pre-trained model from "microsoft/graphcodebert-base" and a fine-tuned model from "saved\_models/small/checkpoint-best-blue/pytorch\_model.bin" were first loaded. BLEU-4 scores, which gauge the quality of generated sentences in comparison to references, were employed as the evaluation metrics. The model achieved a BLEU-4 score of 10.32 on the validation dataset and 8.38 on the test dataset. Both evaluations resulted in an xMatch score of 0.0.

Low BLEU-4 scores and the model's poor performance in producing close matches to reference sentences suggest that the model was loaded with weight initialization problems. It might also be that the training data weren't indicative of the test data. A score of 0.0 for xMatch indicates that no exact matches were discovered. These results point to the need for additional model and training improvement.

**Demo Video Link:** [**https://youtu.be/Y5zMyOpnq3c**](https://youtu.be/Y5zMyOpnq3c)

**Screenshots:**  
 Figure 1: Provisioning of GPU on Colab

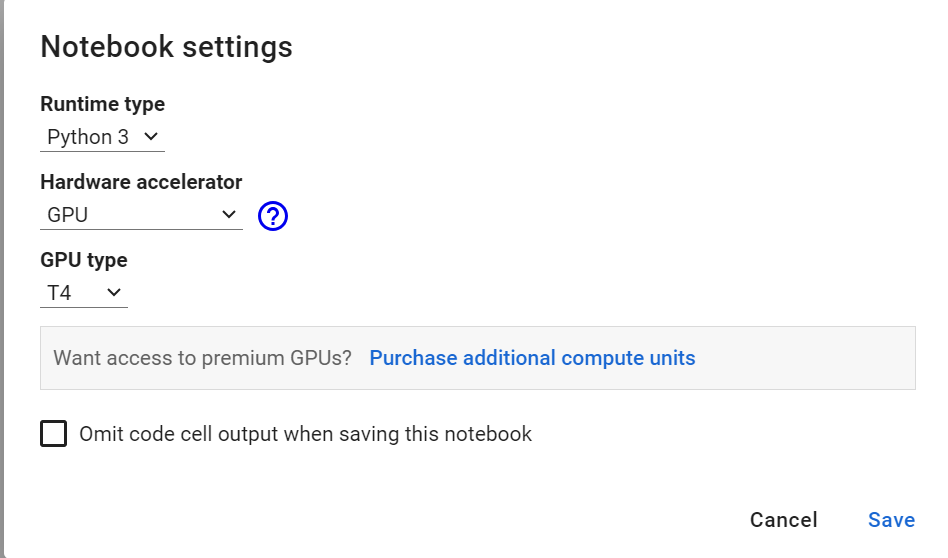


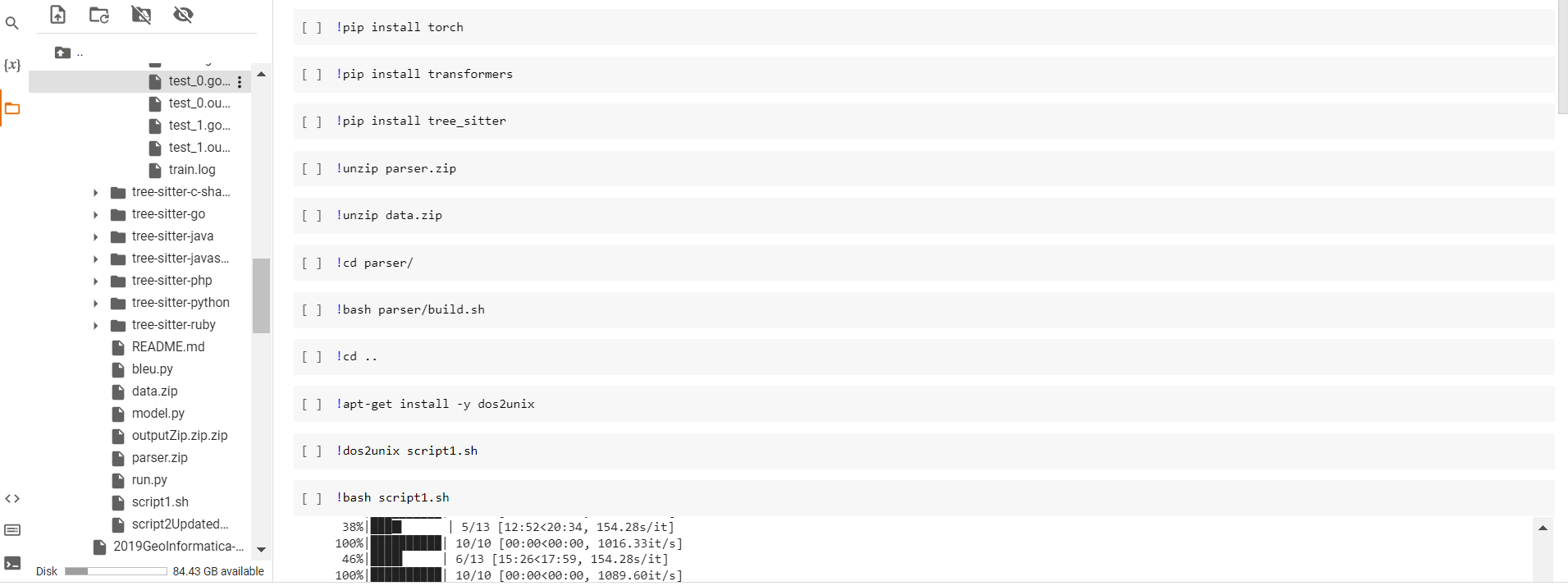
Figure 2: Implementation of the project on Colab  


Figure 3: Execution of Fine-tune script - execute

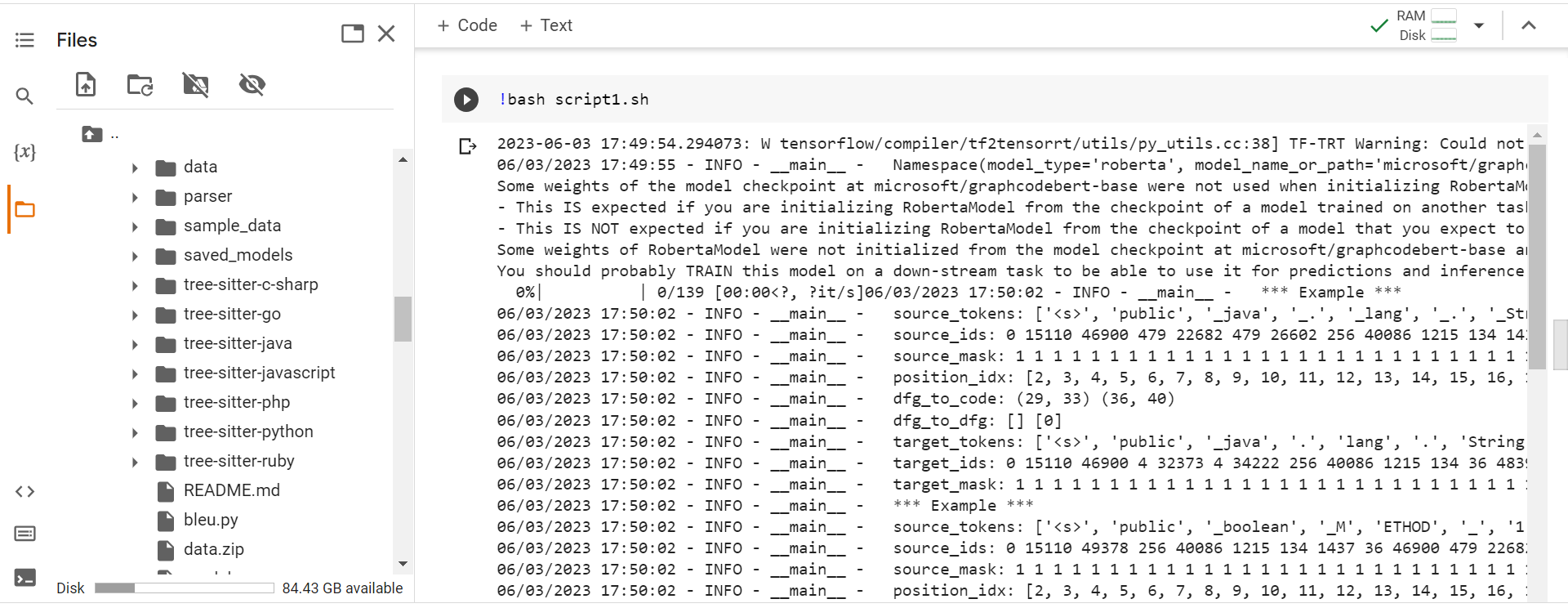


Figure 4: Execution of Fine-tune script - Results

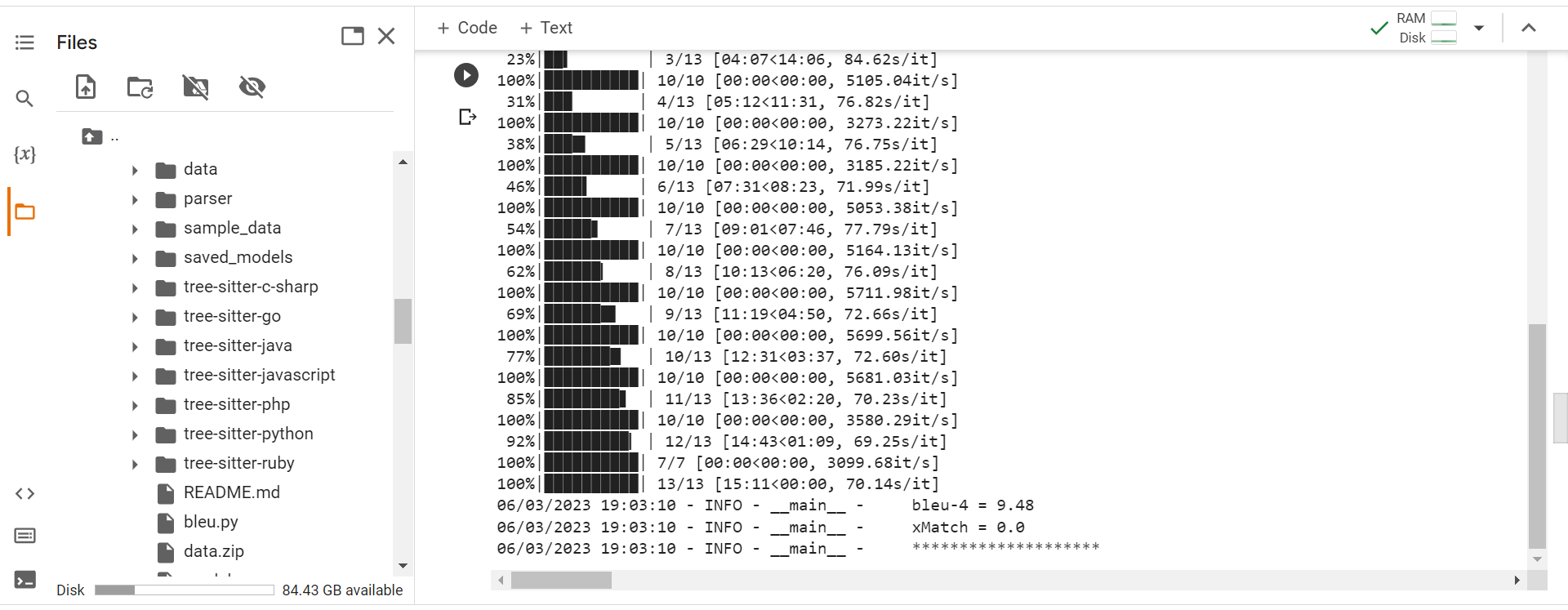


Figure 5: Execution of Inference script - execute

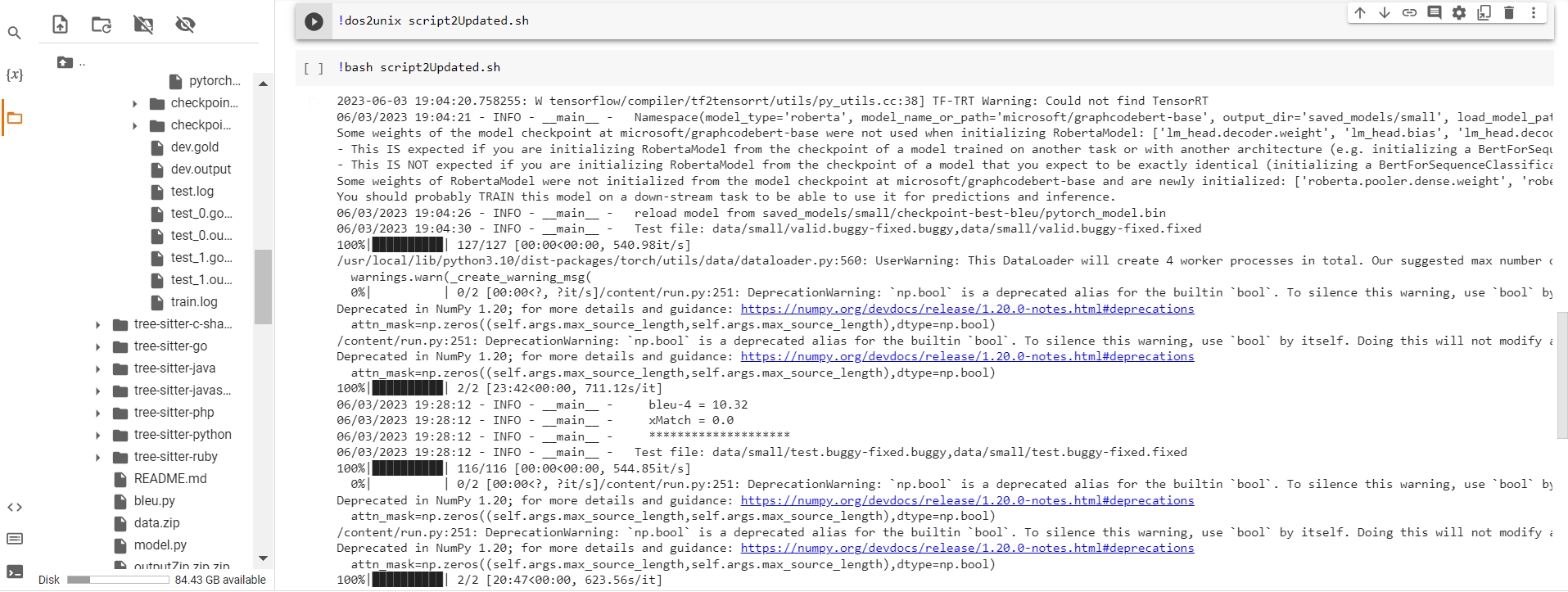


Figure 6: Execution of Inference script - Results

