

# Project Report

Aditya Somani – Nandita Venkatesh – Catherine Nguyen

---

## Introduction

The project addresses the challenge of improving a pre-trained model's capability to understand and generate Python programming related responses by fine-tuning a language model on Python question and answer data. This effort is crucial for enhancing automated coding assistance tools, which are becoming integral in educational environments and development settings. We selected the Flan T5 model for fine-tuning, using a Kaggle dataset (*Python Questions from Stack Overflow*, n.d.) of Python questions and answers sourced from Stackoverflow. After data preprocessing and iterative training sessions, the model demonstrated significant improvements in generating relevant and useful answers, verified through both human evaluation and automated metrics such as ROUGE scores.

## Model

The chosen model for this project was the base Flan T5 model, an enhanced version of Google's T5 (Text-To-Text Transfer Transformer) model. Flan T5 has been fine tuned on more than 1000 additional tasks and is reported on HuggingFace documentation as "better at everything" for the same number of parameters (*Google/Flan-T5-Base · Hugging Face*, n.d.).

## Model Card for FLAN-T5 base

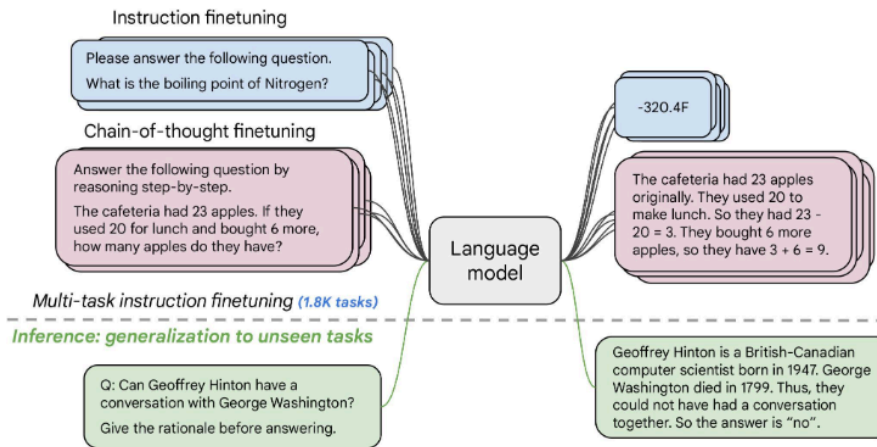


Figure 1. Model Card of FLAN-T5 base model from the HuggingFace website.

In this project, the model was configured to take a string containing a Python programming question and output a text string containing the corresponding code snippet or answer.

For this project, we are coding using Python on a Google Colaboratory notebook. The pretrained model is loaded using `T5ForConditionalGeneration` from the HuggingFace library. To train the model on sequence-to-sequence tasks, we utilized the `Trainer` class from the HuggingFace library, which provides an API for feature-complete training in PyTorch.

## Preprocessing and Data Preparation

The dataset sourced from Kaggle contained two CSV files – questions and answers in HTML format for Stack Overflow posts that contain the "Python" tag.

The preprocessing steps involved:

- **Cleaning:** Using Python's BeautifulSoup library, we converted HTML entities to plain text. We also wrote a function to check and remove any row from both Answers and Questions containing non-UTF characters in the dataset that could lead to encoding errors during model training.
- **Quality Filtering:** Answers were filtered based on the number of upvotes, with a threshold set at two upvotes to ensure the data quality. Only answers with at least two upvotes were retained.

- Diversity: To maintain diversity in the model's training data, we also limited the number of answers per question to the top two based on the number of upvotes.
- Merging the dataset: We then merged the Question and Answer data set to create the training dataset.

We ended up with approximately 320,000 entries post-filtering from an initial set of 980,000. This was then divided into training (80%) and test set (20%). The training set was further divided to get a validation set (5%). Initially, this number was set to 15%, but we noticed that the validation step was taking a lot longer, and eventually crashed the training for the 2nd batch of training data, so we decided to reduce it to 5%. This decision is based on the fact that we had 2000 test samples per batch, and we did not have early stopping implemented.

## Experiments

### Data Segmentation

Initially, the training crashed and could not be done due to GPU RAM overload. This was negated once the batch size and size of the training set were reduced.

With such a large volume of data, to manage computational load, we segmented the data into batches of 20,000. This approach allowed for more manageable training cycles and frequent model evaluation to monitor progress.

### Training Procedure

The training was carried out in a high-performance computing environment equipped with L4 GPU to handle the computational demands of the Flan T5 model.

We employed the following strategy:

**Incremental Training:** Starting with the pre-trained Flan T5, we incrementally trained the model on the segmented batches (20000 samples each). Each batch underwent three epochs of training. Both human evaluation and Rouge score evaluation on the test set were done before and after each batch training to see the gradual improvements.

We did also divide the initial test set into corresponding batches of 2000 to gauge the generalizability of the model.

Some key notes of the training process:

- Initially, we were not able to set the maximum of tokens that the model generates, so the first set of 20000 samples were tested and trained on the default setting of 20 tokens.
- The second 20000 samples were trained for 2 epochs before the Colab crashed in the middle of the validation step. This fueled our decision to change the validation set to 5% of the total 20000 samples of the training set.

## Monitoring and Evaluation

For evaluation, we employed both automated and human-based methods:

- ROUGE Scores: The ROUGE metrics (Rouge-1, Rouge-2, and Rouge-L) were used after each epoch to quantitatively measure the overlap between the model-generated answers and the actual answers in the test set. This is our chosen method of quantitative measurement of the model's performance.
- Human Evaluation: We chose a fixed set of six random questions relating to the topic of Python programming to ask the model before, during (between each batch of training set) and after training to evaluate the accuracy, relevance, and helpfulness of the model's answer. This is our qualitative measurement of the model's performance.

Our questions set is:

```
questions = [  
    "What is the difference between list and tuple?",  
    "How to create a new text file in Python?",  
    "How to install Numpy?",  
    "How to add 2 lists together?",  
    "How to remove a specific element from a list in Python?",  
    "How to find the index of an element in a list in  
    Python?",  
]
```

- This question list is chosen because they are general Python questions that we all know the answer to.





How to create a new text file in Python?	<pad> You can use os.path.join to create a new file. import os os.path.join(os.path.join(os.path.dirname(filename), '.'))</s>
How to install Numpy?	<pad> You can download the numpy package from here: <a href="http://www.np.np.ac.uk/downloads/np.np.py">http://www.np.np.ac.uk/downloads/np.np.py</a> </s>
How to add 2 lists together?	<pad> You can use zip() to combine the two lists together: >>> lst = [1,2,3] >>> lst.append([1,2,3]) >>> lst [1, 2, 3]</s>
How to remove a specific element from a list in Python?	<pad> You can use a list comprehension: >>> l = [1,2,3] >>> l.remove('a') [1, 2, 3]</s>
How to find the index of an element in a list in Python?	<pad> You can use the id() function to find the index of the element in the list: >>> l = [1,2,3] >>> id(l) 0 >>> id(l) 1 >>> id(l) 2 >>> id(l) 3 >>> id(l) 4 >>> id(l) 5</s>

Table 4.2. Qualitative results for the second fine-tuned Flan T5 model.

## Final fine-tuned model

Test Loss	Rouge 1	Rouge 2	Rouge L	Rouge Lsum
1.179	0.2005	0.06606	0.1571	0.1709

Table 5.1. Quantitative results for the first fine-tuned Flan T5 model.

Question	Model's output
What is the difference between list and tuple?	<pad> Lists are a common way to access a list or tuple. Lists are a common way to access a list or tuple. Lists are a common way to access a list or tuple. <i>[model continues to repeat this]</i>
How to create a new text file in Python?	<pad> You can use os.path.join() to create a new file. import os import sys import os.path f = open('new_file.txt', 'r') f.write('<unk> n'.join(f))</s>
How to install Numpy?	<pad> You can install numpy using pip.</s>
How to add 2 lists together?	<pad> You can use zip() to combine two lists together: >>> zip(*[1,2,3]) [1, 2, 3]</s>
How to remove a specific	<pad> You can use a list comprehension: l = [] for i in range(len(l)):

element from a list in Python?	<code>l.append(l[i])</code>
How to find the index of an element in a list in Python?	<code>&lt;pad&gt; You can use a list comprehension: l = [1,2,3,4,5]</code> <code>l.append(l[0])</code>

Table 5.2. Qualitative results for the first fine-tuned Flan T5 model.

## Analysis

Initially, we can see that the pretrained model did not do well at all on both our qualitative and quantitative evaluation metrics. The Rouge scores range from 0.01 to 0.05, with Rouge score closer to 0 indicating poor similarity between the result and the reference (*Evaluate Translation or Summarization with ROUGE Similarity Score - MATLAB RougeEvaluationScore*, n.d.). With human evaluation, we see that the model outputs either nonsensical or unhelpful outputs. For reference, the answer to “How to add 2 lists together?” is “Add the first list to the second list.” (Table 2.2).

After the first fine-tuning, we noticed that the model performs significantly better in both the quantitative and qualitative measurements. The 4 Rouge score has an average increase of 125%. The answers that the model gave were also all comprehensible English, and not just repeated random characters. It was able to output some almost correct answers, such as Numpy being a Python library and some command would be able to download it (though the command was not correct). It also gave an example of `list.append()` to join 2 lists together. Though that was not the most accurate answer, it is in the ballpark of the required subject (Python, list commands) despite the question being vague.

We noticed that the results, though improved significantly, had more than a few answers being endless repeats of the same utterances. This may be due to the fact that we initially trained the model with the default `max_length = 20` instead of a specific `generation_max_length` argument, but specified a `max_new_tokens` value of 300 in the `human_eval` function. Therefore, the model may not have been able to learn correct answers that are longer than 20 tokens.

For the second and third 20000-sample training sets, we changed the `generation_max_length` parameter to first 300, and then 256 tokens for faster computation time. The decision was based on the 95 percentile max answer length from the train set.



After the second fine-tuning round, despite the model crashing after the 2nd epoch, we observed that the Rouge scores have an average increase of 73.9% compared to the first fine-tuned model and 286% compared to the pretrained model. Upon investigating the qualitative results, we noticed that there was no answer with repeated phrases, and the model managed to give the correct command (though the example was a bit strange) to remove a specific element in a Python list.

After the final training (third fine-tuning round), we observed no significant improvement compared to the second fine-tuned model. Upon investigating the qualitative results, we noticed that there were improvements in some answers, while some worsened.

However, we do note that the test sets were different for each 20000-sample batch; hence, the comparison might not be as indicative of true performance. Though we thought that this would help measure the generalizability of our results, we did not take into account the comparability of the performance over time. Due to the limited computing power that we had access to, we were not able to rerun the predict function for each batch.

## **Future Works**

Since we had a much larger corpus of data, we can run the same finetuning training process on the rest of the data. With more computing power, we can probably train on larger batches, to improve training speed. We can also source data relating to Python on similar websites to increase data volume. Additionally, instead of passive human evaluation, we can incorporate reinforcement learning with human inputs to improve model performance.

## **Ethical Statement**

While this work is exciting and would be helpful for learners to learn Python coding, it can be misused by students who would abuse it on class assignments instead of actually learning. Furthermore, perfecting such models may lead to the increase of layoffs of software engineers (which we do not want).

## References

⚡ *LLM 04a - Fine-tuning LLMs* ⚡. (n.d.). Kaggle.com. Retrieved March 28, 2024, from

<https://www.kaggle.com/code/aliabdin1/llm-04a-fine-tuning-llms>

*Evaluate translation or summarization with ROUGE similarity score - MATLAB*

*rougeEvaluationScore*. (n.d.). Wwww.mathworks.com.

<https://www.mathworks.com/help/textanalytics/ref/rougeevaluationscore.html>

*Finetuning Large language models using QLoRA*. (n.d.). Kaggle.com. Retrieved March 28, 2024, from

<https://www.kaggle.com/code/neerajmohan/finetuning-large-language-models-using-qlora>

*google/flan-t5-base · Hugging Face*. (n.d.). Huggingface.co. <https://huggingface.co/google/flan-t5-base>

*Hugging Face: Using `max\_length`'s default (20) to control the generation length. This behaviour is deprecated*. (2023, June 19). Mark Needham.

<https://www.markhneedham.com/blog/2023/06/19/huggingface-max-length-generation-length-deprecated/>

*LLM Evaluation Metrics: Everything You Need for LLM Evaluation - Confident AI*. (n.d.).

Wwww.confident-Ai.com. Retrieved March 28, 2024, from

<https://www.confident-ai.com/blog/llm-evaluation-metrics-everything-you-need-for-llm-evaluation>

*Python Questions from Stack Overflow*. (n.d.). Wwww.kaggle.com.

<https://www.kaggle.com/datasets/stackoverflow/pythonquestions>

*RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! when resuming training*. (n.d.). Stack Overflow. Retrieved May 9, 2024, from

<https://stackoverflow.com/questions/66091226/runtimeerror-expected-all-tensors-to-be-on-the-same-device-but-found-at-least>

*Trainer*. (n.d.). Huggingface.co. Retrieved May 9, 2024, from

[https://huggingface.co/docs/transformers/en/main\\_classes/trainer#transformers.Seq2SeqTrainer](https://huggingface.co/docs/transformers/en/main_classes/trainer#transformers.Seq2SeqTrainer)