UNIVERSITY OF
TORONTO

ECE 1786 – Creative Applications of Natural Language Processing

"Eng2Py" Progress Report

18th November 2022

Course Instructor: Prof. Jonathan Rose

Team Members

Dhairya Parmar - 1006859516
Yuchen Dai - 1008242313

Word Count - 980

Introduction:

Since the project proposal, the goal of our project: to generate code solutions to basic python programming problems pertaining to sorting algorithms using natural language queries remains the same.

Data Processing:

In the proposal stage, our original thought was to use the two datasets (MBPP Dataset & Custom Dataset from TSAI) for our project. We planned to train our model with general python problems and fine-tune it with a sorting algorithms dataset we created.

However, we found a more appropriate model which has been pre-trained on coding problems (we'll talk about this model in detail in the Architecture section) and there are not many sorting problems in the above two datasets, so we plan to fine-tune the new model directly with our own dataset, curated to be comprised of sorting problems only.

As a test running for the progress report, we created our own dataset with 89 data entries in total. We split the dataset into train, validation, and test set and they contain 72, 8, and 9 entries respectively.

```
training_set = training_set.remove_columns([list(training_set.features.keys())[-1]])
training_set
```

```
Dataset({
    features: ['text', 'code'],
    num_rows: 72
})
```

```
validation_set = validation_set.remove_columns([list(validation_set.features.keys())[-1]])
validation_set
```

```
Dataset({
    features: ['text', 'code'],
    num_rows: 8
})
```

Each data entry consists of (input, target) tuples, where input would be a problem statement in English and target would be a code snippet solution to the problem. For example, the input asks for generating a Python program to sort an array using bubble sort in the figure below:

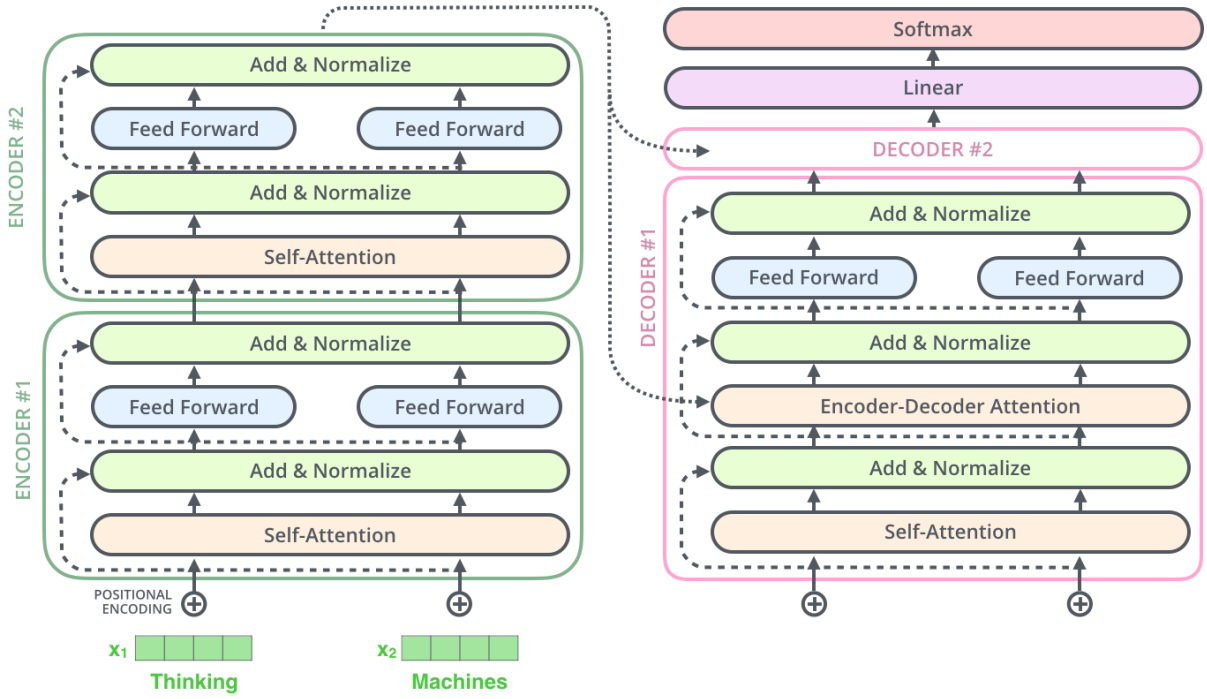
```
### SAMPLE INPUT
# write a python program to sort an array in ascending order using bubble sort

### TRUE LABEL (TRUE CODE)
def bubbleSort(array):
    swapped = False
    for i in range(len(array)-1,0,-1):
        for j in range(i):
            if array[j]>array[j+1]:
                array[j], array[j+1] = array[j+1], array[j]
                swapped= True
        if swapped:
            swapped=False
        else:
            break
    return array
arr = [9,4,3,5,6,1,2,7]
print(bubbleSort(arr))
```

Baseline Model:

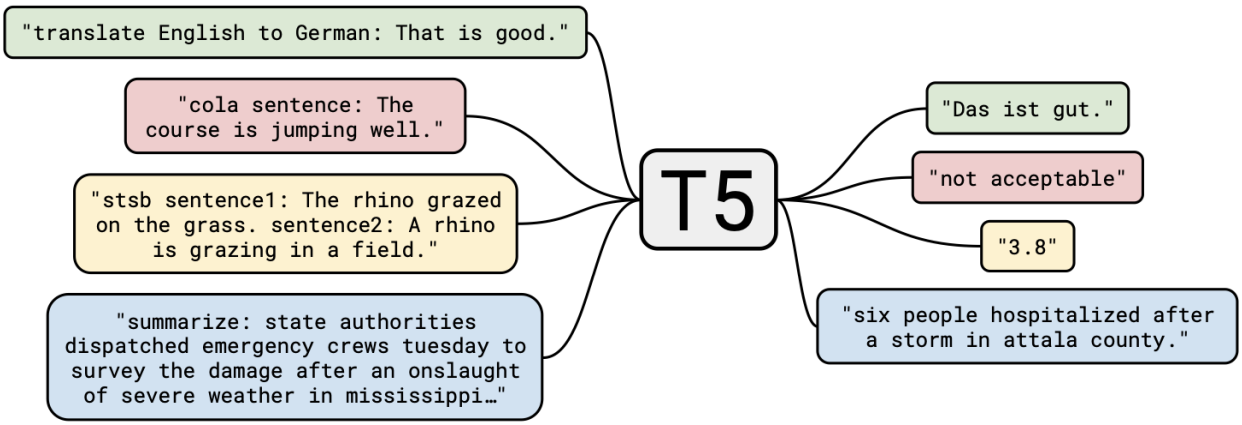
Since it is not clear what a simple baseline model is in the case of code generation, we will not include a baseline model, as described in the project proposal. Instead, we are going to hand-label each output of our model upon their completeness and correctness. Details will be shown in the later sections.

Architecture:

In very simple terms, what we are trying to achieve is given an English instruction for writing a python program (associated to sorting algorithms) our model should output a python code snippet. Hence we were in search of transformer models, which were pre-trained in a text-to-text setting so that we can fine-tune it with our dataset for code-generation as a downstream task. The most appropriate model we found for text-to-text problems is a T5 Transformer model. T5 stands for **T**ext-**T**o-**T**ext **T**ransfer **T**ransformer. The T5 model is a standard encoder-decoder model which can be seen in the figure below,

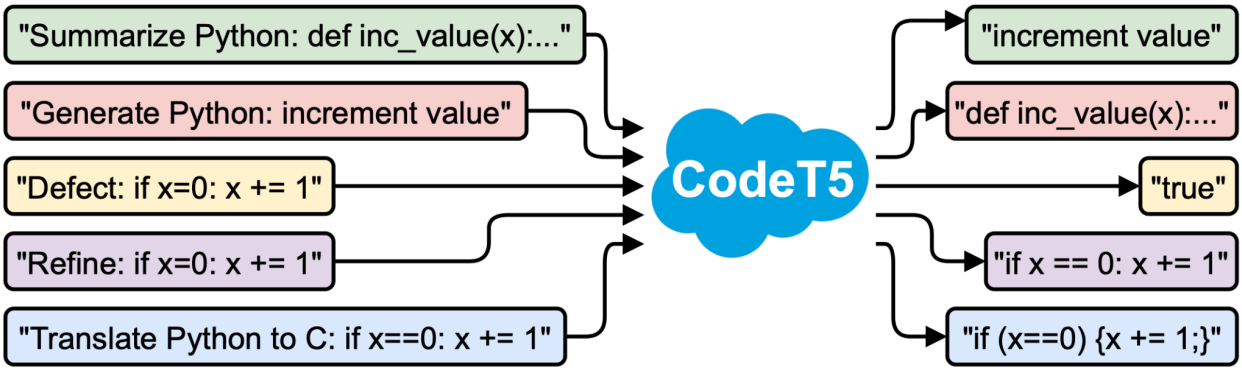


The T5 model can perform multiple tasks like machine translation, classification, regression (for example, predict how similar two sentences are, the similarity score is in range 1 to 5), and text summarization simultaneously,



After fine-tuning this model for our code generation task we found that the results were no where near satisfactory, since this model has never been introduced to python code in pre-training mode.

In further search for a model pre-trained on code, we found the **CodeT5 model** which has the same model structure as T5 but has a different tokenizer which leverages the syntactical structure a code snippet possesses. The CodeT5 model, analogous to the T5 model, can also perform multiple tasks,



The task "Generate Python:" is a direct application of what we aim to achieve. For this project, the largest CodeT5 model we can implement, complying with the computational resources we have, is the base-sized model. The encoder and decoder layer stacks consist 12 transformer blocks each (each block comprising self-attention, optional encoder-decoder attention and a feed-forward network). The feed-forward networks in each block consist of a dense layer with an output dimensionality of $d_{ff} = 3072$ followed by a ReLU nonlinearity and another dense layer. The "key" and "value" matrices of all attention mechanisms have an inner dimensionality of $d_{kv} = 64$ and all attention mechanisms have 12 heads. All other sub-layers and embeddings have a dimensionality of $d_{model} = 768$. In total, this results in a model with about 220 million parameters.

This model can be found on huggingface and to reproduce this one can follow the code provided in the figure below,

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("Salesforce/codet5-base")

model = AutoModelForSeq2SeqLM.from_pretrained("Salesforce/codet5-base")
```

Result:

Since we don't have a baseline model, we plan to evaluate our results manually and give a qualitative comparison between the true labels and the generated labels as follows,

Sample Result 1:

```
### SAMPLE INPUT
# Write a python program for Bitonic Sort

### TRUE LABEL (TRUE CODE)
def compAndSwap(a, i, j, dire):
    if (dire == 1 and a[i] > a[j]) or (dire == 0 and a[i] > a[j]):
        a[i], a[j] = a[j], a[i]

def bitonicMerge(a, low, cnt, dire):
    if cnt > 1:
        k = cnt//2
        for i in range(low, low+k):
            compAndSwap(a, i, i+k, dire)
        bitonicMerge(a, low, k, dire)
        bitonicMerge(a, low+k, k, dire)

def bitonicSort(a, low, cnt, dire):
    if cnt > 1:
        k = cnt//2
        bitonicSort(a, low, k, 1)
        bitonicSort(a, low+k, k, 0)
        bitonicMerge(a, low, cnt, dire)

def sort(a, N, up):
    bitonicSort(a, 0, N, up)

### GENERATED CODE
def bitonicSort(arr):
    n = len(arr)
    i = 0
    while i < n and arr[i] > arr[i+1]:
        i += 1
        arr[i], arr[i+1] = arr[i+1], arr[i]
            i += 1
        arr[i], arr[i+1] = arr[i+1], arr[i]
    return arr
```

As we can see, the generated code only has one function compared to 4 functions in the true code. Inside the generated function, we observe that it has some repeating code, and the second "i += 1" has wrongly indented. To be concise, the generated code under original settings does not pass upon completeness and correctness. However, it does have a proper function structure and the general syntax is correct (even though it did not perform the task).

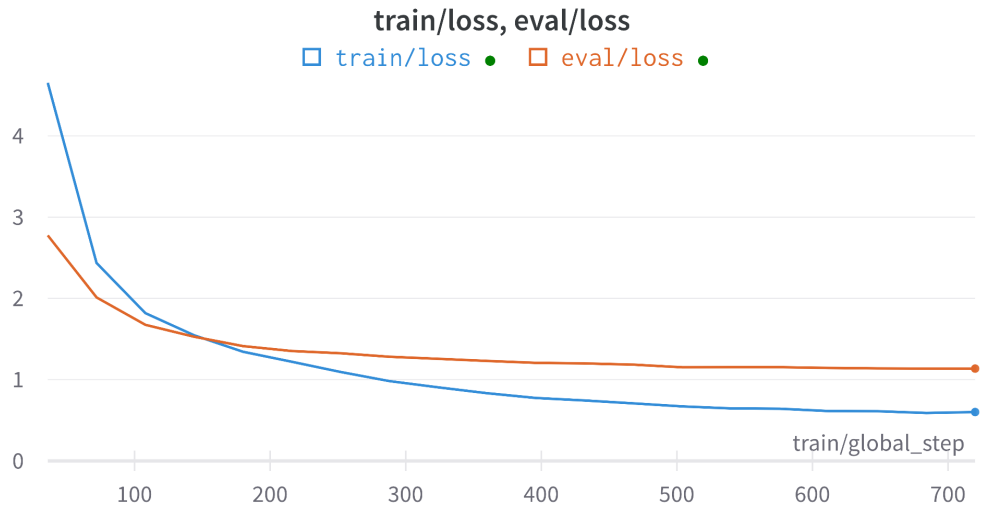
Sample Output 2:

```
### SAMPLE INPUT
# write a python program to find the third distinct
# maximum number in this array using bubble sort

### TRUE LABEL (TRUE CODE)
def bubbleSort(array):
    swapped = False
    for i in range(len(array)-1,0,-1):
        for j in range(i):
            if array[j]>array[j+1]:
                array[j], array[j+1] = array[j+1], array[j]
                swapped= True
        if swapped:
            swapped=False
        else:
            break
    return array
arr = [2,2,3,1]
arr = bubbleSort(arr)
count = 1
prev = arr[0]
for i in range(len(arr)):
    if arr[i] != prev:
        count += 1
        prev = arr[i]
        if count == 3:
            print(arr[i])

### GENERATED CODE
def bubbleSort(array):
    swapped = False
    for i in range(len(array)-1,0,-1):
        if array[i] > array[j+1]:
                array[j+1], array[j] = array[j+1], array[j]
            swapped= True
        else:
            array[j+1] = array[j]
    return array
arr = [9,4,3,5,6,1,2,7]
print(bubbleSort(arr))
```

For this example output, the generated code doesn't aim towards the intended purpose of problem rather it just displays a function to implement bubble sort (part of the problem). The function name is appropriate but the attempt to implement bubble sort has some bugs when compared to the true code, for example, the iterator variable i and j are not used correctly, the swapped variable wasn't changed back to False and the else block doesn't have a break statement. Overall, the generated output doesn't satisfy completeness and correctness criteria, but for a first-run output it is not worse.

Discussion:

Following are the loss curves plotted during training,



One thing we observed while comparing the results is that the model generated exactly the same array example for bubble sort in the code as we provided in training. This observation and the above loss curve provides proof of overfitting. Our model as of now doesn't perform well on the correctness and completeness criteria but it is sufficient enough as an initial step towards achieving our goal.

Team Work and Progress:

In general, we cooperate pretty well and we have completed all the tasks we planned on time. To increase communication efficiency, we had frequent meetings for any updates. Each member contributed to creating half of the dataset and we worked together on finding the appropriate model and making it run.

References:

Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. *arXiv [cs.CL]*. Ανακτήθηκε από http://arxiv.org/abs/2109.00859

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., … Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, *21*(140), 1–67. Ανακτήθηκε από http://jmlr.org/papers/v21/20-074.html