

**ECE 1786 – Creative Applications of**

**Natural Language Processing**

**“Eng2Py” Project Proposal**

**28th October 2022**

**Course Instructor: Prof. Jonathan Rose**

**Team Members**

**Dhairya Parmar - 1006859516**

**Yuchen Dai - 1008242313**

**Word Count - 1098**

**Introduction:**

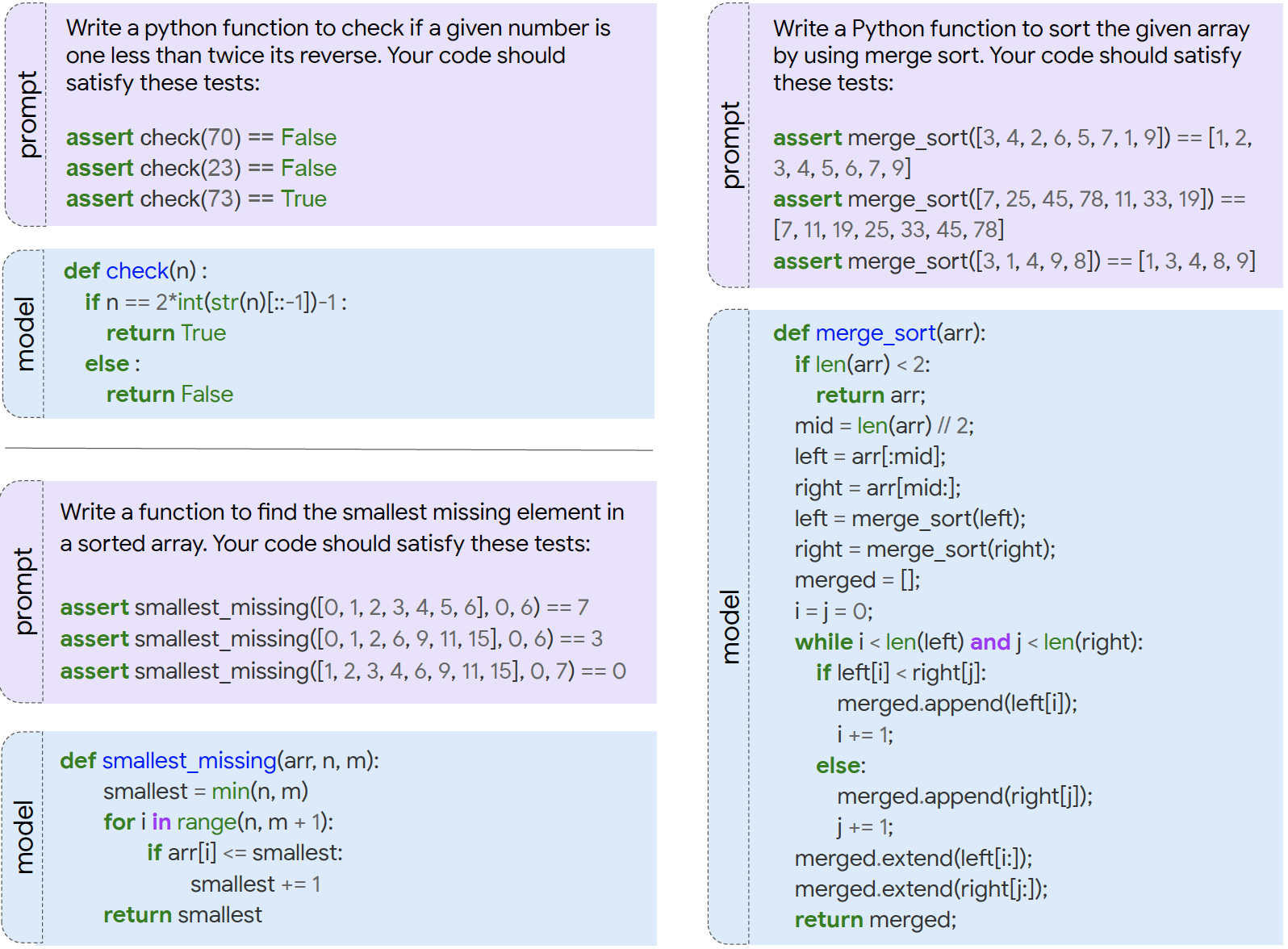
All programmers, especially beginners but even experts, often reach a point in a program where they conceptually understand what needs to be done next but lack the ability to create a robust implementation of their ideas or prefer not to have to type it in if they can avoid it.

The goal of our project is to generate code solutions to basic python programming problems pertaining to sorting algorithms using natural language queries.

The motivation behind our project is creating a piece of software that can help software developers save keystrokes or avoid writing dull pieces of code and help non-programmers in other fields, who require computation in their daily work, in creating data manipulation scripts.

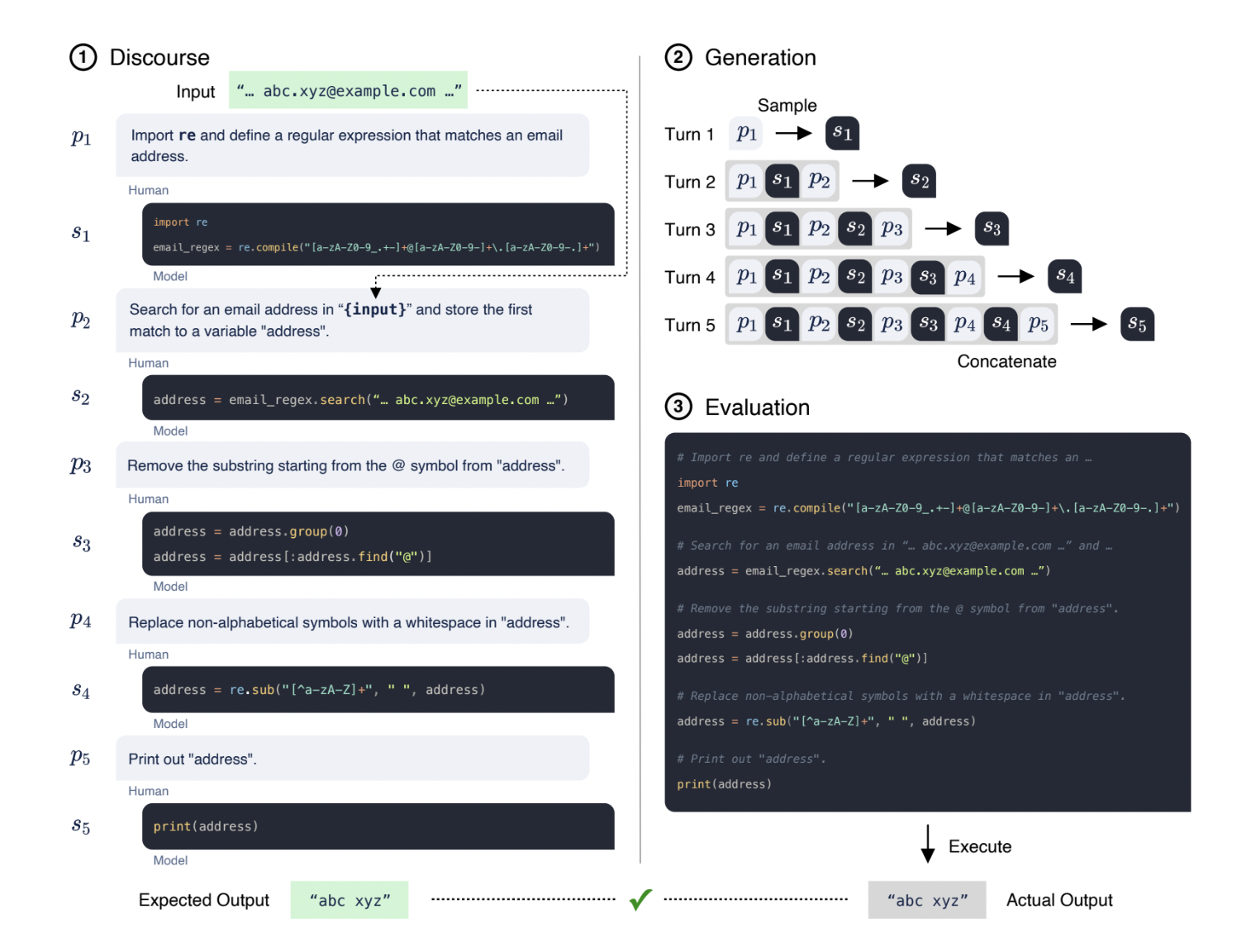
**Background:**

Code Generation or as **(Austin et al., 2021)** defines it “Program Synthesis” is a longstanding goal of artificial intelligence. Two recent developments from a research literature point of view indicates potential in the task of code generation. First, large language models have demonstrated astonishing new capabilities to produce natural language text and to resolve a constantly expanding set of modeling and reasoning problems. Austin et al. studies how a collection of large Transformer language models performs when applied to the synthesis of short programs written in general purpose programming languages like python. Examples of problems and model generated code is shown in the figure below,



This study concludes that large language models perform surprisingly well in the task of generating code for short python problems, with the largest model generating code solutions that pass the test cases for the majority of standard problems.

In a newer research of program synthesis, **(Nijkamp et al., 2022)** investigates the multi-step paradigm for it. To achieve program synthesis, there are two key challenges: (1) the intractability of the search space, and (2) the difficulty of properly specifying user intent. To overcome these challenges, **(Nijkamp et al., 2022)** proposes a multi-turn program synthesis approach which lets a user to communicate with the system by progressively providing specifications in natural language while receiving responses as synthesized subprograms. An example of extracting the user name of an email address is shown in the figure below,



The research team constructs an open benchmark called Multi-Turn Programming Benchmark (MTPB), and concludes that the same intent provided in multi-turn fashion has significantly better performance than that provided as a single turn.

Another study, **(Perez et al., 2021)**, explores the potential of using pre-trained language models for automatic code generation. This study uses an RNN based baseline model to compare the results generated by a pre-trained GPT-2 model fine-tuned on code data and it concludes that the best performing model is the Transformer model.

**Source of Data and Processing:**

For the proposed project, we will make use of the following two datasets by filter them for sorting algorithms only as part of our dataset,

(A) Mostly Basic Python Problems (MBPP) Dataset:-

<https://huggingface.co/datasets/mbpp>

(B) Custom Dataset curated by The School of AI:-

<https://drive.google.com/file/d/1rHb0FQ5z5ZpaY2HpyFGY6CeyDG0kTLoO/view>

For the Data Collection/Labeling aspect, we are planning to generate a sorting algorithms dataset by ourselves consisting of (input, target) tuples, where input would be a problem statement in English and target would be a code snippet solution to the problem.

We would process all three of these data sources to have a common general format and then concatenate them to form a single dataset, which can be further used to create training, validation and/or test splits.

Lastly, we will make use of data augmentations to further increase the size of our dataset.

The reason we chose to focus on sorting algorithms is that we wanted to explore if the model learns the specific set of problems better than very general data. By focusing on a specific category of algorithms instead of random algorithms, the model may understand this specific class better and raise the performance.

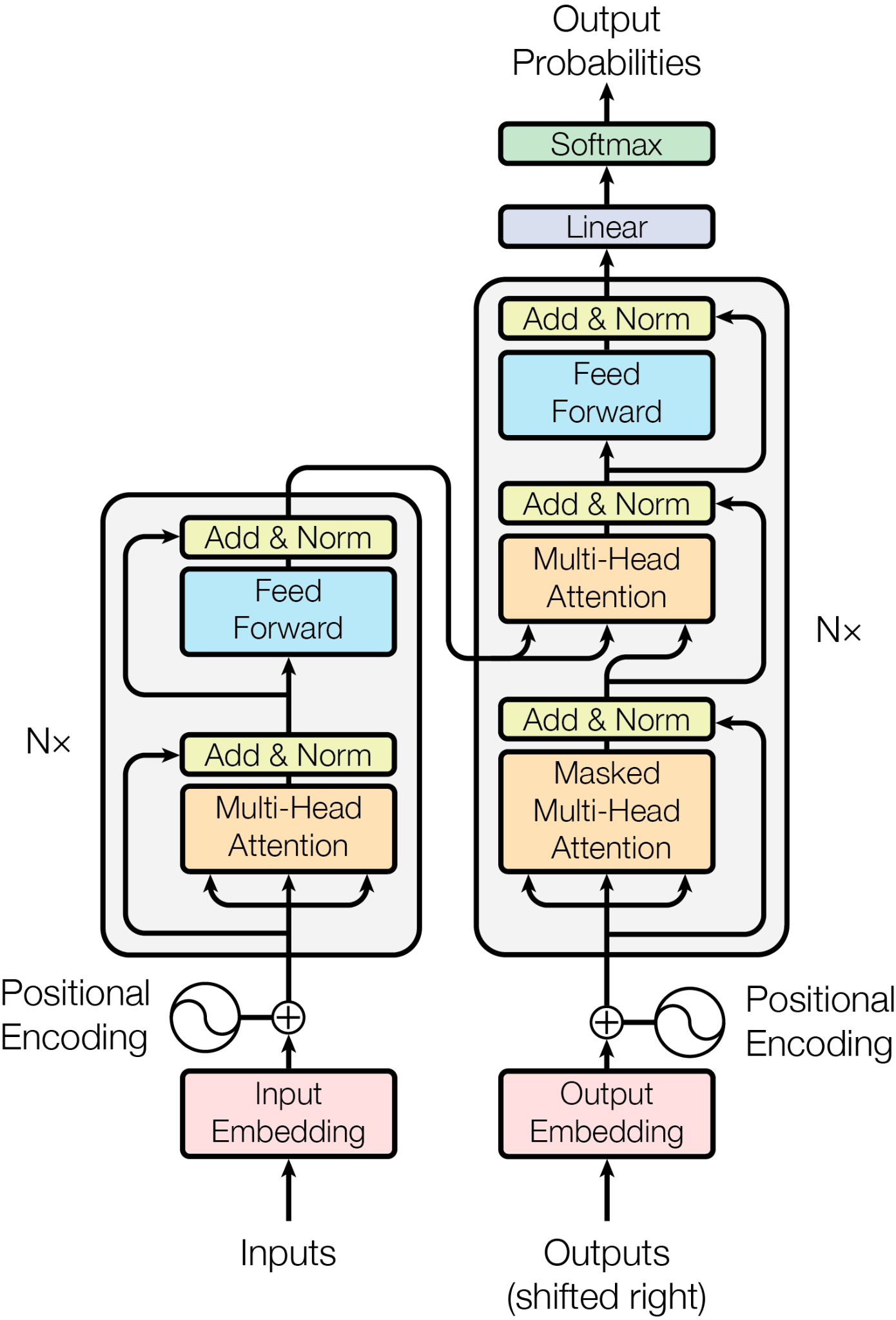
Sample data points in the combined training set would look like as follows,

**Input string:** "Write a function to sort a list in a dictionary."

**Target string:** "def sorted\_dict(dict1): sorted\_dict = {x: sorted(y) for x, y in dict1.items()} return sorted\_dict"

**Architecture of the Model:**

The model architecture for this “Sequence to Sequence” problem is going to be an Encoder-Decoder Transformer Model, which can be visualized from the following figure found in **(Vaswani et al., 2017)**



The Transformer model will be trained on all kinds of python problems and then fine-tuned on a curated dataset of sorting algorithms.

**Baseline Model:**

Since it is not clear what a simple baseline model is in the case of code generation, we may not include a baseline model in our program. Instead of comparing results from a baseline model, we choose to hand-label everything that our model will generate, as there is no other way to judge the quality of our results.

To be concrete, we are going to hand-label each output of our model upon their completeness and correctness. At the end, we will specify the percentage of the acceptable results as a direct observation.

**Plan:**

The project is divided into following subtasks and the corresponding time estimate is provided alongside each task:

| **Task** | **Time Estimate** |
| --- | --- |
| Data Collection/Labeling | Nov 02 - Nov 05 (3 - 4 Days) |
| Programming the Model Architecture | Nov 14 - Nov 21 (7 - 8 Days) |
| Progress Report | Nov 20 - Nov 21 (1 - 2 Days) |
| Training/Evaluation/Baseline Comparison | Nov 22 - Nov 28 (6 - 7 Days) |
| Schedule Buffer | Nov 28 - Dec 02 (5 - 6 Days) |
| Final Presentation | Dec 02 - Dec05 (3 - 4 Days) |
| Final Report | Dec 06 - Dec 13 (6 - 7 Days) |
|  | Tentative Project Completion - 40 Days |

**Team Collaboration Strategy:**

* We will use github as required by the project to commit all our source code in a repository
* We plan to meet once a week to discuss progress, issues and mitigations
* We will have regular online meetings to discuss about the project as required

**Risks:**

1. Since both of the team members have other duties e.g. Assignment 4, other courses, etc., changes may go beyond our plan. So we have reserved a schedule buffer around 5 to 6 days to deal with any unexpected events. If everything keeps up with the plan and no problem is encountered, then we could use the schedule buffer to further improve the existing model or the training process.
2. The training process may take up a large amount of time and limited things can be done during the waiting time. So we plan to create the model as soon as possible to start training so that we can prepare the progress report at the same time.

**References:**

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., & Sutton, C. (2021). Program Synthesis with Large Language Models. 10.48550/ARXIV.2108.07732

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., & Xiong, C. (2022). CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. 10.48550/ARXIV.2203.13474

Perez, L., Ottens, L., & Vishwanathan, S. (2021). Automatic Code Generation using Pre-Trained Language Models. 10.48550/ARXIV.2102.10535

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. 10.48550/ARXIV.1706.03762