

# CSCE 5290/4290:

## Natural Language Processing Project

### Names:

Oriana Borges-Salinas, Zachary Chenausky, Anas Mohammed Nayeem, Dagar Rehan

### Project Title:

Research Paper Translation and Summarization

### Github:

<https://github.com/oriAleph/Research-Translation-Summarization>

### Project Proposal:

The project's primary goal is to translate and summarize research papers. The project will be divided into two sections. The first section focuses on research paper translation from one language to another. The second section focuses on summarizing the document to provide the user with the most important information. Four deep-learning models will be trained using the same training set and epochs for both portions of the project. There will be four models in total, two for translation and two for summarization. Furthermore, comparisons will be made to show which models performed well and which did not, and ideas for improving the models will be tested.

TL;DR: Train and compare four deep-learning models for the translation and summarization of research papers.

### Description:

#### 1. Motivation

The goal of the project is to develop software that will enable someone who doesn't speak the original language to comprehend the significant ideas of a research article. This comprises translating the paper's content from the original tongue into English and summarizing it for easier user comprehension. All the models will also be compared to one another to see which combination performed the best and how to make the models better.

## **2. Significance**

This project has many practical applications and is a key component of natural language processing. A wide range of businesses and organizations use translation and summarization, two of the most explored topics in computer science. With this study, we will concentrate on Czech and English research syntax and semantics because various languages have distinct ways of describing information. Due to the project's NLP components, we apply what we learned in class to use by designing a tool for cross-cultural research comprehension. This serves as an excellent learning tool for our team.

## **3. Objectives**

- Train two models that can translate from the native language to English
- Train two models that can summarize English text and give key points
- Compare which combination of models gives the best results
- Try to improve the models to give better results

## **4. Features**

- Natural Language Toolkit (NLTK)
- Hugging Face
- spaCy
- TextBlob
- English datasets
- Czech datasets
- REGEX
- GPT Corpus
- Collections
- TensorFlow
- Keras
- Numpy

## 5. Task Visualization

Name	Task
Dagar	Summarization - T5-small model
Zachary	Translation - RNN model with Attention
Anas	Summarization - Abstractive roBERTa model
Oriana	Czech to English NMT with a Transformer and Keras

## Increment 1:

### Related Work (Background):

#### Translation:

The task of machine translation involves converting a sentence from one target language to another, which has lately significantly advanced thanks to encoder-decoder attention-based systems like BERT. Rule-based, statistical, hybrid and neural-based methods can all be used for machine translation.

The rule-based approach, or RBMT, to machine translation, generates sentences using both the source language and the target language's grammatical rules. However, this method has the drawback of heavily relying on dictionaries and extensive editing.

The second approach is models based on statistics. The best part about statistical models is that they can be quickly translated from one language to another without taking context into account.

A hybrid machine translation system, or HMT, combines statistical models and RBMT. This performs better than prior tests conducted by other researchers, but it takes a lot more resources to develop a model like this than it does to use the other methods.

Using neural networks and their capacity to learn when given enough data, neural machine translation is a translation method that can independently pick up linguistic rules and iteratively learn how to connect the source and target languages. The issue with this method is that it requires a sizable dataset for training. NMT was used for this project; it is used by many popular translation sources, such as Google Translate.

#### Summarization:

NLP summarization is the task of breaking down large amounts of text into a paragraph or a few sentences to provide key information. As the field of NLP developed, summarization was split into two different subfields Extractive and Abstractive.

Extractive Summarization is the process of extracting key information from the text and using techniques like frequency to determine how relevant that information is. If the frequency is high enough or meets certain criteria then it will be added to the summarization paragraph and displayed to the user. This type of summarization does not involve generating new sentences, instead, sentences that appeared in the original text will appear in the summary as well. This has led to limited use of such a method even though useful it lacks the flexibility to make its predictions.

Abstractive Summarization is the process of generating a summary of text with sentences that are generated by the language model instead of just grabbing key information from the initial text. The model in Abstractive Summarization uses deep-learning models such as T5 or Bert to generate sentences that have not appeared in the text before. The models use supervised learning where you have a dataset with the actual text and a summary written by a human and the model tries to learn to make connections from text to summary.

The project is based on Abstractive Summarization.

Models such as T5 and Bert come pre-trained in which the weights and biases have already been trained on a huge dataset but for this project, the models were initialized with the default weights of the model before the training took place. The models are trained on the dataset we have chosen.

### **Overall:**

The combination of translation and summarization has already been combined in the past. This project focuses on the translation and summarization of research papers from all different types of fields and shall be fine-tuned for Computer Science related papers from the NIPS dataset.

## Datasets:

### Translation Datasets

[WMT](#) is the main event for machine translation and research. The conference is held yearly in conjunction with bigger natural language processing conferences [1].

The [wmt19\\_translate](#) dataset is based on the data from statmt.org, versions exist for various years utilizing a combination of several data sources [2].

Czech to English [parallel sentences](#) [3].

[1]“WMT,” *Machine Translate*, Sep. 2022. [Online]. Available: <https://machinetranslate.org/wmt>. [Accessed: Nov. 06, 2022]

[2]“wmt19\_translate | TensorFlow Datasets,” *TensorFlow*, Oct. 2022. [Online]. Available: [https://www.tensorflow.org/datasets/catalog/wmt19\\_translate](https://www.tensorflow.org/datasets/catalog/wmt19_translate). [Accessed: Nov. 06, 2022]

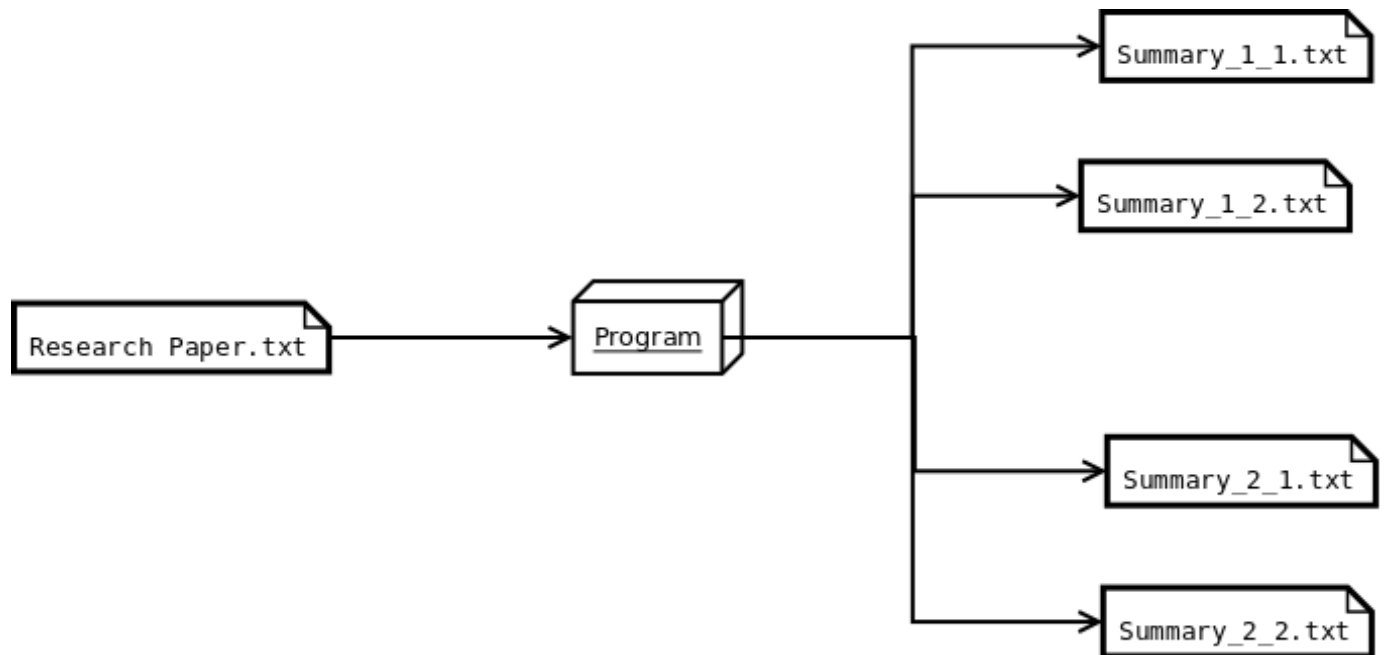
[3]“Tab-delimited Bilingual Sentence Pairs from the Tatoeba Project (Good for Anki and Similar Flashcard Applications),” *ManyThings*, 2022. [Online]. Available: <http://www.manythings.org/anki/>. [Accessed: Nov. 06, 2022]

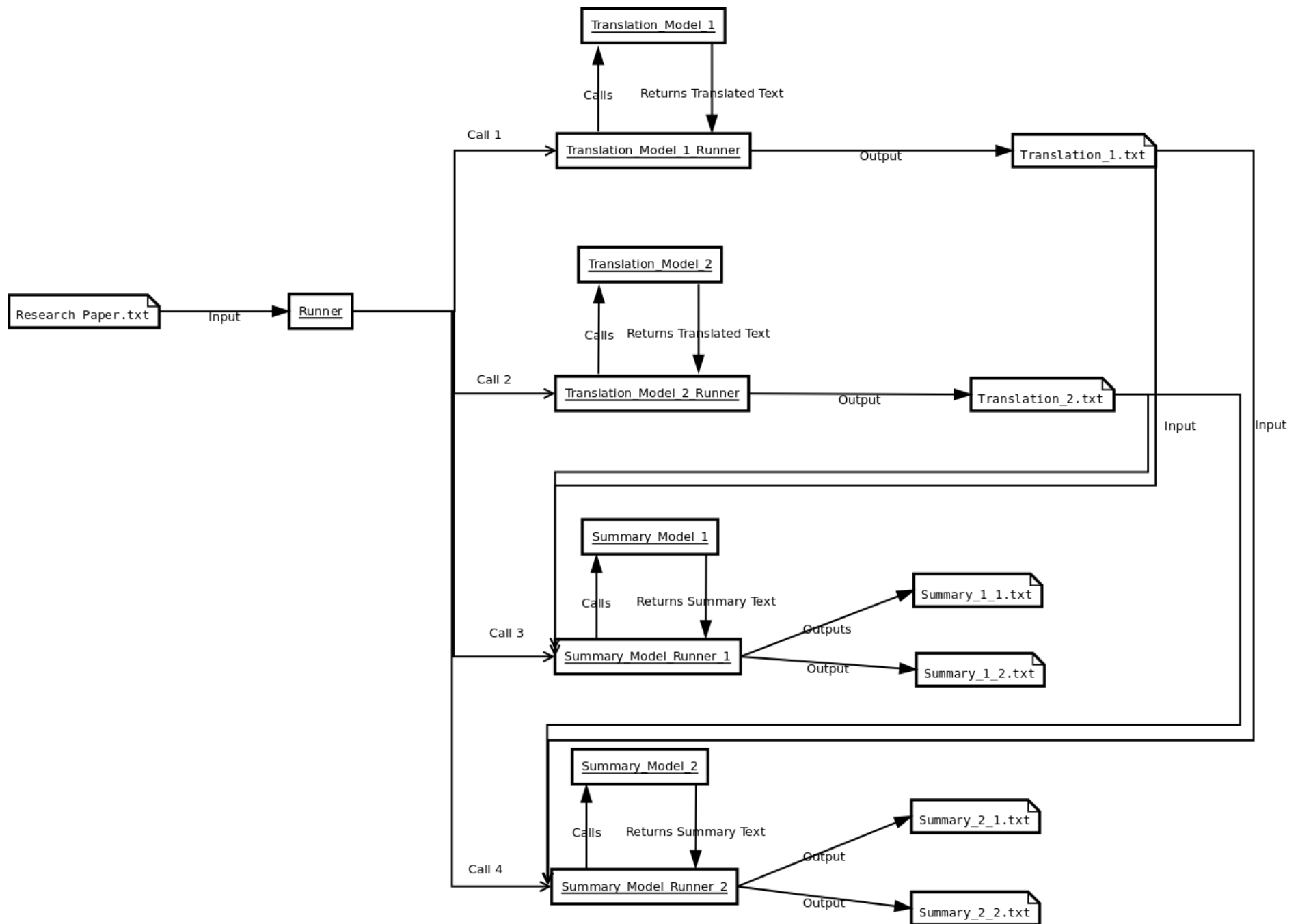
### Summarization Datasets

For Training: [scientific\\_papers](#) subset: **arxiv** - contains papers that are not related to medicine.

For Tuning: [All NeurIPS \(NIPS\) Papers](#)

## Detail Design of Features:







## Analysis:

The project's goal is to make a piece of software that can take a research paper from a different language and translate it to English and then summarize the content to key points within the paper. This required the combination of two major NLP techniques such as machine translation and abstractive summarization. This combination of techniques has been done before but the focus is on research paper summarization and not a general summarization model. Furthermore, the model will be tuned for Computer Science paper summarization using the NIPS dataset.

## Translation:

### Transformer with Self-Attention Model [Oriana]:

A neural network is a model containing at least one hidden layer, which holds a set of neurons performing specific tasks [1].

A deep neural network design, a model with more than one hidden layer, called a **transformer** was created at Google and relies on self-attention processes. It can be thought of as a stack of self-attention layers that includes either a decoder, an encoder, or both [1].

A series of embeddings is transformed into another by a **self-attention** layer [1]. Attention can be any of a vast variety of methods that collect information from a collection of inputs in a dependent manner [1]. An example of a typical attention mechanism is a weighted sum over a collection of inputs, where the weight for each input is determined by a different component. Self-attention refers to the process of paying *attention* to itself rather than another context [1].

Transformers are effective for modeling sequential data because they can be parallelized, therefore, computations can be done concurrently [2]. Additionally, transformers are capable of capturing far-reaching contexts because attention enables each place to access the entirety of the input at each given layer [2]. Comparatively, the information must go through numerous processing steps in RNNs and CNNs in order to travel a long distance, which makes learning more challenging. A Transformer model is similar to a Recurrent Neural Network (RNN) model, but instead of RNN layers, a Transformer uses self-attention layers [2].

I had to develop a subword vocabulary and a BERT tokenizer for this model. A subword tokenizer's key benefit is that it dynamically adjusts between tokenization that is word-based and that is character-based, meaning it can return to word fragments and individual characters for unknown words [3].

According to the International Standardization Organization's (ISO) 639 Language codes "cs" stands for Czech and "en" stands for English [4].

- [1] "Machine Learning Glossary | Google Developers," *Google Developers*, Oct. 2022. [Online]. Available: <https://developers.google.com/machine-learning/glossary>. [Accessed: Nov. 06, 2022]
- [2] M. Daoust, "Neural machine translation with a Transformer and Keras | Text | TensorFlow," *TensorFlow*, Nov. 2022. [Online]. Available: <https://www.tensorflow.org/text/tutorials/transformer>. [Accessed: Nov. 06, 2022]
- [3] M. Daost, "Subword tokenizers | Text | TensorFlow," *TensorFlow*, Jul. 2022. [Online]. Available: [https://www.tensorflow.org/text/guide/subwords\\_tokenizer](https://www.tensorflow.org/text/guide/subwords_tokenizer). [Accessed: Nov. 06, 2022]
- [4] "ISO 639 — Language codes," *ISO*, 2022. [Online]. Available: <https://www.iso.org/iso-639-language-codes.html>. [Accessed: Nov. 06, 2022]

### **RNN with Attention Model [Zachary]:**

The recurrent neural network (RNN) model is based on the attention-based neural machine translation. The primary idea of training the model includes using sequence-to-sequence where the model converts one sequence in a certain language to another sequence in another language. The RNN encoder and decoder map the source language vector to its representation vector in its target sequence. This process works fine for smaller and simpler sentences however, as the length and contextual complexity of the source sequences increases it becomes more difficult for the model to translate. Attention allows for more accurate translations with RNN because of the fact that the decoder has the functionality to look back at previous translations.

### **roBERTa Abstractive summarization Model [Anas]:**

The roBERTa model is based on the BERT model from hugging-face. roBERTa is an acronym for a robustly optimized BERT approach. It employs some enhancements like modifying the key hyperparameters and removing BERT's next sentence pertaining to objectives. roBERTa uses more intricate tokens that go deeper down to characters. This helps in extracting text and reconstructing an abstractive summary for the model. When we say abstractive summary we target a summary that is concise and precise. Through its deeper learning techniques roBERTa leverages more intricate training sessions and a higher accuracy than that of a BERT summarizer. It is solely for these advantageous properties that roBERTa was employed in this project. Although BERT travels through 1M steps in total. The detailed architecture of roBERTa allows it to perform a higher quality grade training in just within 500K steps.

**T5-small Model [Dagar]:**

The T5 model is an encoder-decoder model that has been pre-trained on a huge dataset with supervised and unsupervised learning and can be finetuned downstream for specific tasks. In the case of the project, the weights were initialized to default to get rid of any of the training and were only trained on the science paper dataset. This has led to a significant impact on the summarization of general articles but should have better performance for science paper-related summaries. In addition, the T5-small model converts all NLP problems to Text-to-Text format for better performance.

## Implementation:

### Translation:

#### Transformer with Self-Attention Model [Oriana]:

This model follows and adapts the [Neural machine translation with a Transformer and Keras TensorFlow](#) tutorial. The Tensorflow lesson shows you step-by-step how to build and hone a Transformer model that can convert Portuguese into English by using the [TED Talk HRLR transcripts](#) from the [Tensorflow Dataset Collection](#). However, this model converts Czech into English by using the [WMT 2019 dataset](#) from the Tensorflow Dataset Collection and statmt.org. For the exact dataset paths, look into [wmt.py](#) from the [TensorFlow Datasets Translate](#) Github. Both the original TensorFlow lesson and this notebook use the [Subword Tokenizers](#) TensorFlow tutorial to create a subword vocabulary and build a tokenizer for BERT, exported by using [tf.saved\\_model](#) so it can be used by other notebooks.

< — Brief Overview of Set Up — >

#### Installation:

```
# Google Colab:
# !apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
!pip uninstall -y -q tensorflow keras tensorflow-estimator tensorflow-text
!pip install -q tensorflow_datasets
!pip install -q -U tensorflow-text tensorflow
```

#### Import modules:

```
import logging
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow as tf
import tensorflow_text
```

#### If on Google Colab:

```
# Google Colab:
# !unzip dev.zip
# !unzip training-parallel-nc-v13.zip
```

Download Dataset:

```
config = tfds.translate.wmt.WmtConfig(  
    version="0.0.1",  
    language_pair=("cs", "en"),  
    subsets={  
        tfds.Split.TRAIN: ["newscommentary_v13"],  
        tfds.Split.VALIDATION: ["newstest2018"],  
    },  
)  
builder = tfds.builder("wmt_translate", config=config)
```

```
print(builder.info.splits)  
builder.download_and_prepare()  
datasets = builder.as_dataset(as_supervised=True)  
print('datasets are {}'.format(datasets))
```

```
train_examples = datasets['train']  
val_examples = datasets['validation']
```

If on Google Colab:

```
# Google Colab:  
# !unzip wmt19_translate_cs_en_converter.zip
```

Set up Tokenizer:

```
model_name = 'wmt19_translate_cs_en_converter'  
tokenizers = tf.saved_model.load(model_name)
```

```
# Tokenizer methods  
[item for item in dir(tokenizers.en) if not item.startswith('_')]
```

## RNN with Attention Model [Zachary]:

The datasets used to train the model were gathered from manyThings.org/anki. The data sets were structured as side-by-side sentences, left-side English, and right-side Czech. Every Czech sentence was imported into its own text file, the same as the English sentences. Then the files were stored as arrays in order to create a TensorFlow.data dataset for the model.

The installation libraries:

```
!pip install "tensorflow-text>=2.10"  
!pip install einops
```

And imports :

```
import numpy as np  
import typing  
from typing import Any, Tuple  
import einops  
import matplotlib.pyplot as plt  
import matplotlib.ticker as ticker  
import tensorflow as tf  
import tensorflow_text as tf_text
```

Next creating the TensorFlow dataset from the imported text files. In order to train the model the use of text vectorization for text processing is the next step. Text processing includes removing punctuation from the datasets as well as setting all characters to lowercase for Unicode normalization.

```
text = tf_text.normalize_utf8(text, 'NFKD')  
text = tf.strings.lower(text)  
# Keep space, a to z, and select punctuation.  
text = tf.strings.regex_replace(text, '[^ a-z.?!,\;]', '')  
# Add spaces around punctuation.  
text = tf.strings.regex_replace(text, '[.?!,\;]', r' \0 ')  
# Strip whitespace.  
text = tf.strings.strip(text)
```

```
text = tf.strings.strip(text)
```

Extracting vocabulary from each language is important for the language model. The TensorFlow Keras text vectorization layer handles the vocabulary extraction for both the Czech and English

datasets. The text vectorization function reads one epoch of the training set and initializes the layer based on each data to determine the language vocabulary.

```
context_text_processor.adapt(train_raw.map(lambda context, target:
context))
```

```
# Here are the first 10 words from the vocabulary:
context_text_processor.get_vocabulary()[:10]
```

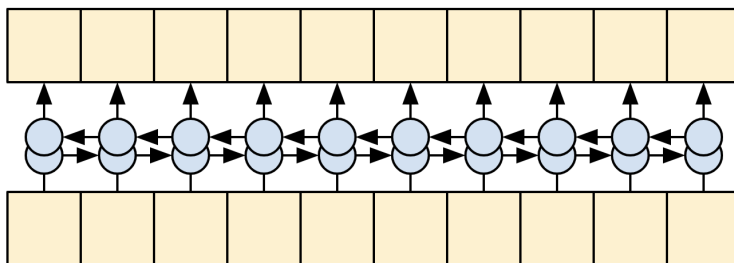
```
target_text_processor = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_vocab_size,
    ragged=True)

target_text_processor.adapt(train_raw.map(lambda context, target: target))
target_text_processor.get_vocabulary()[:10]
```

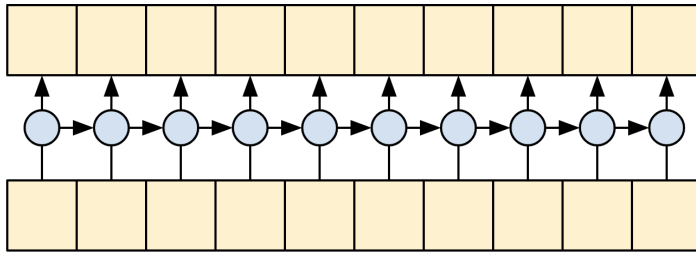
These layers can now convert the batch of strings into a batch of token IDs.

The TensorFlow datasets created previously are converted into 0-padded tensors of token IDs in the process\_text function. In order to use Keras model.fit the input must be the context(English) and the target(Czech) with the target in and target out labels.

The encoder is used to process the context sequence into a sequence of vectors so that the decoder may use that information to predict the output at each timestep. The sequence is constant so the model uses a bidirectional RNN.



The decoder will predict tokens at each target sequence. The decoder uses the RNN to process the target sequence and attention to keep track of each generated prediction. The model while training predicts the next word at each location one word at a time.



Given the context and target tokens, each target token, predicts the next target token. The model components include the RNN, attention setup, encoder, and decoder functionalities. After each section is implemented the model can be trained on the datasets.

```
def __init__(self, units,
              context_text_processor,
              target_text_processor):
    super().__init__()
    # Build the encoder and decoder
    encoder = Encoder(context_text_processor, units)
    decoder = Decoder(target_text_processor, units)

    self.encoder = encoder
    self.decoder = decoder

    def call(self, inputs):
        context, x = inputs
        context = self.encoder(context)
        logits = self.decoder(context, x)
```

An example input after translation:

```
inputs = [
    'Je tu opravdu zima.', # "It's really cold here."
    'Tohle je můj život.', # "This is my life."
    'Jeho pokoj je nepořádek.' # "His room is a mess"
]
```

```
for t in inputs:
    print(model.translate([t])[0].numpy().decode())
```



```
print()  
its really cold  
this is my life doesnt belong to  
his room is overconfident
```

The model works fine on short sentences, but when the input increases, the model loses focus and is unable to provide reasonable predictions. One way this could be improved would be to implement a re-learning of its predicted outputs. The model as is using a teacher-forcing method where each correct token is learned and passed through the model. The model makes predictions and learns the datasets regardless of whether the previous predictions were correct or not. More advanced systems are able to learn from predictions.

## Summarization:

### roBERTa Model [Anas]:

roBERTa is a hugging face module that was built on the architecture of BERT. BERT is a language network architecture that employs text to encode, segment, extract data. It is a part of a hugging-face module. Since roBERTa has more deeply created tokens and a longer pertaining duration we can leverage its intricate structure to develop a more optimized text summarizer. The roBERTa model was implemented relating very closely to the T5-small model, the objective was to implement the training model with the same dataset as with the T5 model so as to compare the summarization behavior between the two models. roBERTa in its name stands for Robust Optimized BERT Approach, unlike BERT that utilizes word piece tokenization we will leverage roBERTa's character level byte pair encoding tokenizers to enhance the number of word pieces used. A dataset was downloaded from the Hugging Face database namely "scientific\_papers". However the design implementation was a bit different,

The dataset was first downloaded and saved and then fed into the trainer.  
The libraries installed were:

```
!pip install transformers  
!pip install datasets  
!pip install rouge_score
```

The libraries imported are:

```
import datasets  
import transformers  
import pandas as pd  
from datasets import Dataset
```

```

from transformers import RobertaTokenizerFast
from transformers import EncoderDecoderModel
from transformers import TrainingArguments

from transformers.trainer_seq2seq import Seq2SeqTrainer
from transformers.training_args_seq2seq import Seq2SeqTrainingArguments
from dataclasses import dataclass, field as dataclassfield
from typing import Optional
from datasets import load_dataset, load_metric
from transformers import RobertaConfig, RobertaModel
from transformers import EncoderDecoderModel
from transformers import RobertaConfig, RobertaModel
from transformers import DataCollatorForSeq2Seq

```

We use RobertaConfig to tokenize our dataset into default biased tokens. One important feature of RobertaConfig is that it helps us create weightless parameters such that there is no distinct bias in our tokens, The same code can be found under the Github repository under the function name weightlessModel()

```

weightless_model(model_checkpoint,tokenizer):
    #make a model that is not pretrained
    config=RobertaConfig()
    model=RobertaModel(config)
    model.init_weights()

    data_collator=DataCollatorForSeq2Seq(tokenizer=tokenizer,model=model)
    return(model,data_collator)

```

This is part of the preprocessing steps that we take, meaning in order for us to feed the data into the model it should be in a way that the model is able to read and learn from it. By not biasing the data we are ensuring a more cleaner and stable learning algorithm that is able to understand word parameters and summarize.

We will not be jumping into other details of preprocessing so as to avoid any instances of duplication.

After we have winder up the preprocess filters we will pass them through our training\_args. Interestingly we've always been curious about what exactly training\_args is? Well, training arguments are tweaks that we make in our dataset so as to compatibilitate it with our training model. The training model we will be using is the Seq2Seq trainer. Training

arguments help set parameters like the batch size of the trainer, CUDA operations, steps required for eval\_process and warmup steps.

```
training_args = Seq2SeqTrainingArguments(  
    output_dir="./",  
    per_device_train_batch_size=batch_size,  
    per_device_eval_batch_size=batch_size,  
    predict_with_generate=True,  
    #evaluate_during_training=True,  
    do_train=True,  
    do_eval=True,  
    logging_steps=2,  
    save_steps=16,  
    eval_steps=500,  
    warmup_steps=500,  
    overwrite_output_dir=True,  
    save_total_limit=1,  
    fp16=True,  
)
```

Here is an example, as we see in the code (Also available in the GIT repo) we have a set of parameters that all relate to our dataset , the way it is to be handled by the trainer.

Finally, lets talk about the trainer. Just like training\_args , imported from Transformers we will be using the Seq2Seq trainer. This trainer allows us to integrate our model with rogue benchmarks.

I would have explained in detail about the rogue benchmarks that we are using but you can find that in the T5-small model explanation.

Both models had been trained to run 5 epochs so as to ensure stability in comparison of their outcomes.

Although summary expectations were set higher , the results exited were not as ambitious as expected. As a research in development this is a great sign as it gives developers an opportunity to explore several windows of improvement, some of which will be eagerly displayed in increment-2.

### **T5-small Model [Dagar]:**

The T5 model was trained on the science paper dataset that was uploaded to the Hugging Face database.

To install the libraries for training using pip:

```
pip install datasets transformers rouge-score nltk
```

The libraries required to train the model are:

- `from datasets import load_dataset, load_metric`
- `from transformers import AutoTokenizer`
- `from transformers import DataCollatorForSeq2Seq`
- `from transformers import AutoModelForSeq2SeqLM, Seq2SeqTrainingArguments, Seq2SeqTrainer`
- `from transformers import AutoModelForSeq2SeqLM, DataCollatorForSeq2Seq, Seq2SeqTrainingArguments, Seq2SeqTrainer`
- `from transformers import create_optimizer, AdamWeightDecay`
- `from transformers import AutoConfig`
- `from transformers import T5Model`
- `import nltk`
- `import numpy as np`

The t5-small tokenizer used was prebuilt and pre-trained as to parse the text from the corpus. The T5-small tokenizer was used as it pairs well with the t5-small summarizer. The tokenizer from the training data takes in the paper and the summary and truncates it to 1024 and 128 bytes for the model.

To initialize the t5-small summarization model to weights, biases, and any other factor from before the model was pre-trained it is important to use:

```
config = AutoConfig.from_pretrained(model_checkpoint)
model = AutoModelForSeq2SeqLM.from_config(config)
model.init_weights()
```

The auto-config generates a t5-small model that has not been trained.

In addition, the T5 model was trained for 5 epochs with the current hyperparameters set to:

```
(
    f"{model_name}-science-papers",
    evaluation_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
```

```

weight_decay=0.01,
save_total_limit=20,
num_train_epochs=my_epochs,
predict_with_generate=True,
fp16=floating_point,
push_to_hub=True,
)

```

The batch\_size was set to 16 to reduce the amount of memory consumption but the runtime increases.

The metric used to compare the models was Recall-Oriented Understudy (ROUGE):

```
metric = load_metric("rouge")
```

The score is measured by comparing the summary generated by the model with a summary written by a human ut ROUGE goes a step further and does:

Recall = Number\_of\_overlapping\_words / Total\_words\_in\_Referece\_summary

To calculate precision which tells how verbose the summary is:

Precision = Number\_of\_overlapping\_words / Total\_words\_in\_System\_summary

Rouge1 measures unigrams

Rouge 2 measures bigrams

Rouge1 measures the longest matching sequence of words by using the LCS algorithm.

Having higher values for all Rouges is always better as it produces generally better summaries.

To install the libraries to run the model using pip:

```
pip install transformers
```

The libraries to import are:

- `import torch`
- `import json`
- `from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config`

- `from transformers import AutoModelForSeq2SeqLM, AutoTokenizer`

To get the custom trained model to use:

```
model =  
AutoModelForSeq2SeqLM.from_pretrained("Dagar/t5-small-science-papers")
```

This line of code gets the model from the online repository that is on Hugging Face:

<https://huggingface.co/Dagar/t5-small-science-papers>

This makes using the model easier as you can just get it from the repo instead of having the model in the git repo.

The tokenizer needs to be initialized as:

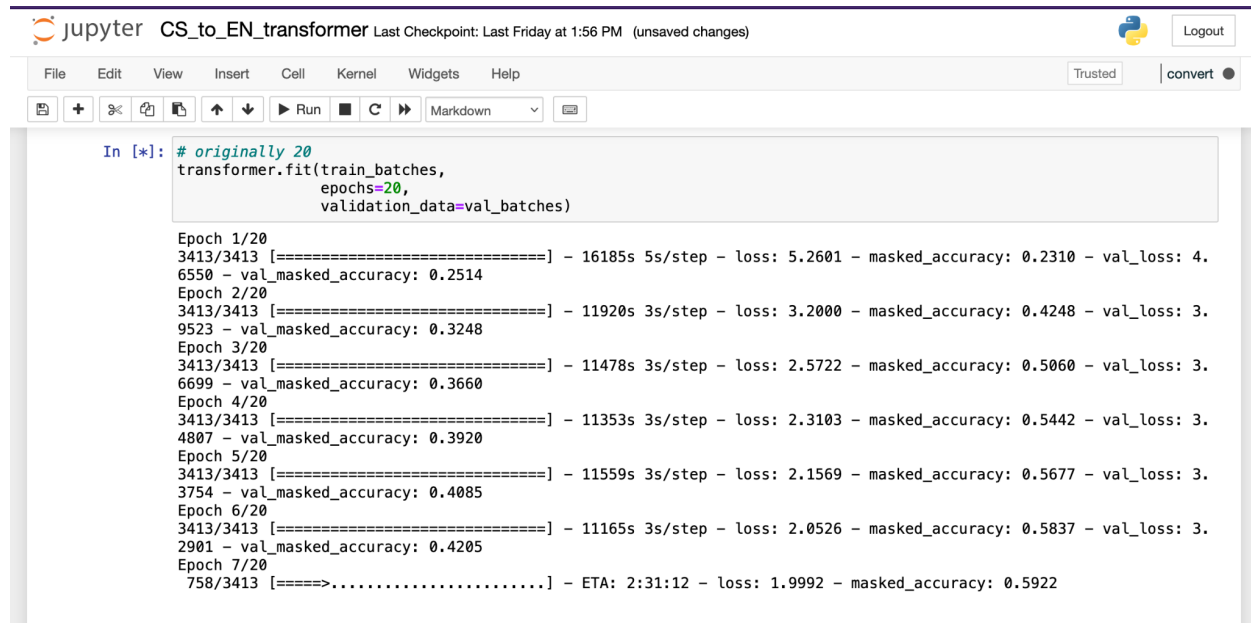
```
tokenizer = AutoTokenizer.from_pretrained('t5-small')
```

The reason it needs to be a t5-small as it was the same tokenizer used to train the model previously.

## Preliminary Results:

### Transformer with Self-Attention Model [Oriana]:

As you can see below, the model is taking a very long time to train since the data set is rather large. There are 20 epochs and each epoch needs about 3 hours to train, however, the accuracy seems to be increasing at a favorable rate.



```
In [*]: # originally 20
transformer.fit(train_batches,
               epochs=20,
               validation_data=val_batches)

Epoch 1/20
3413/3413 [=====] - 16185s 5s/step - loss: 5.2601 - masked_accuracy: 0.2310 - val_loss: 4.6550 - val_masked_accuracy: 0.2514
Epoch 2/20
3413/3413 [=====] - 11920s 3s/step - loss: 3.2000 - masked_accuracy: 0.4248 - val_loss: 3.9523 - val_masked_accuracy: 0.3248
Epoch 3/20
3413/3413 [=====] - 11478s 3s/step - loss: 2.5722 - masked_accuracy: 0.5060 - val_loss: 3.6699 - val_masked_accuracy: 0.3660
Epoch 4/20
3413/3413 [=====] - 11353s 3s/step - loss: 2.3103 - masked_accuracy: 0.5442 - val_loss: 3.4807 - val_masked_accuracy: 0.3920
Epoch 5/20
3413/3413 [=====] - 11559s 3s/step - loss: 2.1569 - masked_accuracy: 0.5677 - val_loss: 3.3754 - val_masked_accuracy: 0.4085
Epoch 6/20
3413/3413 [=====] - 11165s 3s/step - loss: 2.0526 - masked_accuracy: 0.5837 - val_loss: 3.2901 - val_masked_accuracy: 0.4205
Epoch 7/20
758/3413 [=====>.....] - ETA: 2:31:12 - loss: 1.9992 - masked_accuracy: 0.5922
```

### RNN with Attention Model [Zachary]:

With short sentences the model is relatively accurate:



```
[ ] inputs = [
    'Je tu opravdu zima.', # "It's really cold here."
    'Tohle je můj život.', # "This is my life."
    'Jeho pokoj je nepořádek.' # "His room is a mess"
]

%%time
for t in inputs:
    print(model.translate([t])[0].numpy().decode())

print()

its really cold
this is my life doesnt belong to
his room is overconfident

CPU times: user 871 ms, sys: 9.74 ms, total: 881 ms
Wall time: 882 ms
```

However when the input data grows the model becomes less contextually accurate and thus the translation suffers.

### roBERTa Model [Anas]:

The platform had timed out by the time I was able to analyze the rogue outputs. Since the epochs take anywhere 3 to 4 hours to process . I will be displaying the training model results in hopes that they count as preliminary.

```
2944      0.000000
2946      0.000000
2948      0.000100
2950      0.000100
2952      0.000000
2954      0.000000
2956      0.000000
2958      0.000000
2960      0.000000
2962      0.000100
2964      0.000000
2966      0.000000
2968      0.000000
2970      0.000000
2972      0.000000
2974      0.000000

Saving model checkpoint to ./checkpoint-16
Configuration saved in ./checkpoint-16/config.json
Model weights saved in ./checkpoint-16/pytorch_model.bin
/usr/local/lib/python3.7/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:634: FutureWarning:
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
Saving model checkpoint to ./checkpoint-32
Configuration saved in ./checkpoint-32/config.json
Model weights saved in ./checkpoint-32/pytorch_model.bin
Deleting older checkpoint [checkpoint-16] due to args.save_total_limit
/usr/local/lib/python3.7/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:634: FutureWarning:
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
Saving model checkpoint to ./checkpoint-48
Configuration saved in ./checkpoint-48/config.json
Model weights saved in ./checkpoint-48/pytorch_model.bin
Deleting older checkpoint [checkpoint-32] due to args.save_total_limit
/usr/local/lib/python3.7/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:634: FutureWarning:
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
Saving model checkpoint to ./checkpoint-64
Configuration saved in ./checkpoint-64/config.json
Model weights saved in ./checkpoint-64/pytorch_model.bin
Deleting older checkpoint [checkpoint-48] due to args.save_total_limit
```

So by analyzing our data, we see that the more steps we incur the lesser are our training losses.



### T5-small Model [Dagar]:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	RougeL	RougeLsum	Gen Len
1	4.473500	4.372655	9.960400	1.764100	8.621300	9.277900	19.000000
2	4.010400	3.938435	11.400100	2.147400	9.651600	10.660200	19.000000
3	3.823700	3.757975	11.180600	2.122900	9.388100	10.385300	19.000000
4	3.738200	3.673797	11.929800	2.322200	9.907700	11.045000	19.000000
5	3.699400	3.640550	12.356800	2.444900	10.237100	11.420900	19.000000

For the total of 5 epochs run the ROUGE1 score reached 12.35 on the 5th epoch. The ROUGE2 score remains at a 2.44 as the dataset used does not have enough data on research papers to generate consistent bigrams. Furthermore, the idea to increase epochs to more than 5 was discarded to overfit the data. The second dataset chosen for tuning the model should increase the ROUGE2 score. The ROUGE1 score is at a surprising 10.23

For example given the sentence:

“This paper pertains to discussing the Schrodinger equation along with its historical background. The paper starts with an introduction to quantum mechanics and requirement of Schrodinger equation. The paper highlights the need of Schrodinger equation in light of progresses made in quantum mechanics. The paper also highlights the historical background of the development of Schrodinger equation after which the paper also briefly touches the development of the same through classical means. The paper discusses about the Schrodinger equation while presenting the same for a hydrogen atom. The paper ends with conclusion in which the importance of Schrodinger equation is highlighted.”

The summary generated by the model would be:

“in this paper, we study the dynamics of a hamiltonian of the heisenberg equation of motion for atoms. we show that the nonlinear equations of state of two - dimensionality in the case of an atomic system are derived from the generalized eigenstates and the quantum system based on the time evolution equation for the system with arbitrary number of energy and interacting with respect to the energy of this system is characterized by the method of quantum equation with the same time in analytic approach to hilbert space of non”

It is not a good summary but with further tuning the summary should become more accurate towards these types of papers as the hyperparameters have yet to be tuned.

## Project Management:

- Implementation Status Report (**Oriana Borges-Salinas**)

- Work Completed

- Description -

I originally started working on a Chinese-to-English NMT Transformer model but it proved to be difficult. I couldn't get past the preprocessing steps when creating the tokenizer since the top-down subword approach provided by TensorFlow doesn't work for CJK languages because they don't have clear multi-character units. The creator of the tutorial responded to a question asked previously by someone else about this issue, and they answered that the simplest approach to this was to use a pre-trained Chinese segmentation model from the TensorFlow Hub such as [zh\\_segmentation](#) and [bert\\_zh\\_preprocess](#). While I could find one example using the TF BERT segmentation process on Github [here](#) and read about other implementations in WMT conference [papers](#) and from this Stanford [study](#), I didn't want to use the same types of methods they all had taken. However, I couldn't figure out the zh\_segmentation implementation in time so I switched over to Czech since I needed a better understanding of my model and a working piece for the overall product.

I adapted the Portuguese to English Transformer Tensorflow tutorial to fit the Czech dataset of my choosing, and it's currently in the process of training the Czech to English model. Originally I had it running for half a day on Google Collab but it was taking 14 hours for the first epoch, which went blank on me simply because the Wifi went out. Considering the size of the large size of my data set and the lag have I from my computer and cloud, I switched over to Jupyter Notebooks. Each epoch now lasts about three hours and at the time of this writing, it has two hours left for its 7th epoch. I have about 40 hours left, therefore in a little less than 2 days the training will be complete and I can export the model for further use and analysis.

- Responsibility - I am responsible for getting the Neural Machine Translation Transformer using the Self-Attention model working and properly exported, evaluated, and integrated with the project.
- Contributions - I helped Zachary get set up with TensorFlow after switching over from OpenNMT and tried to preprocess the same dataset I used with his tutorial. Since the preprocessing steps are different between each model and the data set from his tutorial isn't originally part of TensorFlow, I converted my sets into text files and modified the first few pre-processing steps in order to better fit the data. It seemed to be

working since I tested it out with Spanish to English. However, when converting the Czech to English text files into NumPy arrays, one of them was off by 1 in terms of shape and they both need to be the same shape in order to use the Dataset.from\_tensor\_slices method. We both decided on using the same language as of this moment until we figured out a way to make the preprocessing work for the same dataset.

- Work to be completed
  - Description - As mentioned above, once my training finishes I will test and evaluate it, then export it. Afterward, I would need to combine my model with the others to achieve the goal of the project: research paper translation and summarization, and then analyze those results.
  - Responsibility (Task, Person) - My responsibility is to help integrate the models with each other and perform the translation-summarization tasks on research papers.
  - Issues / Concerns - My biggest concern is how long it's taking to train each model. It's very long and taxing for my laptop. I am also concerned about getting the same data set to work for both translation models, but that would just take us asking for help and some time to figure out.
- Implementation Status Report (**Zachary Chenausky**)
  - Work Completed
    - Description - Started work originally on OpenNMT translation models with transformers. With a lack of good resources, online a decision was made to switch to TensorFlow RNN with attention translation models. The final project implementation was created using a TensorFlow RNN model with custom datasets.
    - (Task, Person)- Trained RNN model on Czech/English Datasets, found results from different input texts.
    - Contributions - uploaded results to groups GitHub as well as contributed to group meetings and discussions.
  - Work to be completed
    - Description - Potentially improve translation as well as add punctuation to translation output.
    - Responsibility (Task, Person) - Individually I will help the group merge the models into one for the overall research understanding of each model and comparison.
    - Issues / Concerns - One concern would be improving the model could either be an issue of datasets, where I may need to find better and larger dataset resources which could take longer for the model to train on, or that this models implementation is just an outdated model where the transformer translator model would likely excel in the benchmark tests over the RNN model.

- Implementation Status Report (**Anas Mohammed Nayeem**)
  - Work Completed
    - Description - Started work initially on BERT summarizer for an extractive summary, realized expectations were directed towards a more abstractive inclined model rather than an extractive model of summarization. Focused research towards roBERTa optimization and trained a model to implement abstractive and ran a couple of tests to analyze outputs.
    - Responsibility (Task, Person)- The summarizers are of 2 types. A T5 model and a roBERTa model.
    - Contributions - Worked on implementing the roBERTa model from scratch and training it.
  - Work to be completed
    - Description - Integration of translators implemented with roBERTa summarizer. Fine-tuning the system to aim for more accurate output. Record down outputs from the train data.
    - Responsibility (Task, Person) - Aim to integrate both RNN with attention and Transformer Self attention model with roBERTa summarizer.
    - Issues / Concerns - Regarding normalization of datasets. The integration can be a bit tricky though. Essentially we wanna take the output of the translators as our input. But we want to implement that by maintaining as much of a close relationship with the two summarizers as possible.
- Implementation Status Report (**Dagar Rehan**)
  - Work Completed
    - Description - Work on one of the summarization models to train and implement it. Get statistics to compare initial results for the model chosen. Work on the overall design of how to implement the models after training them all.
    - Responsibility (Task, Person) - Train the T5 model for summarization. Implement the T5 model for summarization. Get results using the ROUGE benchmark.
    - Contributions - Trained T5 model from scratch. Made diagrams for the system. Implemented T5 model. Helped with the Background, Related Work, and Design section of the doc.
  - Work to be completed
    - Description - Finish tuning the T5 model to the NIPS dataset for better results regarding research papers.
    - Responsibility (Task, Person) - I will help with merging the modules together and fine-tuning the T5 model for better performance on the NIPS dataset. I will also convert the NIPS dataset to the T5 model format
    - Issues / Concerns - I don't know if we should train our models on some general article to help the model learn better sentence structure. I think fine-tuning the summarization models on general articles and the NIPS dataset would be better. I don't know if you can only tune with the NIPS

dataset or add a general dataset as well. To fine-tune the dataset to NIPS we need to convert the dataset to our own format with each model.

## Increment 2:

### Introduction:

The main objective of the project is to translate and summarize research papers. The project is divided into two parts:

1. Translation of research papers from one language to another.
2. Summarizes the entire research document to give the user the most crucial information.

Using the same training language/sets and number of epochs for the translation and summarizing parts of the project, four deep learning models have been trained (two for translation and two for summarization). There were comparisons made to indicate which models performed well and which did not, as well as the testing for model improvements and/or improvement proposals.

This project is essential to natural language processing and has numerous real-world applications. Two of the most studied areas of computer science, translation and summarization, are used by a diverse array of companies and organizations. We will focus on Czech and English syntax and semantics in this study because different languages have different ways of describing information. Due to the natural language processing components of the project, we put what we learnt in class to use by creating a tool for cross-cultural research. For our team, this is a fantastic learning tool.

We chose Czech to English translation since those two languages are a part of the TensorFlow Dataset collection and have a similar tokenization approach. For example, the top-down subword approach provided by TensorFlow doesn't work for CJK languages because they don't have clear multi-character units.

This is how we distributed project tasks with the team:

Name	Task
Dagar	Summarization - T5-small model
Zachary	Czech to English Translation - RNN model with Attention
Anas	Summarization - Abstractive roBERTa model
Oriana	Czech to English NMT with a Transformer and Keras

## Background:

The project's goal is to make a piece of software that can take a research paper from a different language and translate it to English and then summarize the content to key points within the paper. This required the combination of two major NLP techniques such as machine translation and abstractive summarization. This combination of techniques has been done before but the focus is on research paper summarization and not a general summarization model. Furthermore, the model will be tuned for Computer Science paper summarization using the NIPS dataset.

## Translation:

The task of machine translation involves converting a sentence from one target language to another, which has lately significantly advanced thanks to encoder-decoder attention-based systems like BERT. Rule-based, statistical, hybrid, and neural-based methods can all be used for machine translation.

The rule-based approach, or RBMT, to machine translation, generates sentences using both the source language and the target language's grammatical rules. However, this method has the drawback of heavily relying on dictionaries and extensive editing.

The second approach is models based on statistics. The best part about statistical models is that they can be quickly translated from one language to another without taking context into account.

A hybrid machine translation system, or HMT, combines statistical models and RBMT. This performs better than prior tests conducted by other researchers, but it takes a lot more resources to develop a model like this than it does to use the other methods.

Using neural networks and their capacity to learn when given enough data, neural machine translation is a translation method that can independently pick up linguistic rules and iteratively learn how to connect the source and target languages. The issue with this method is that it requires a sizable dataset for training. NMT was used for this project; it is used by many popular translation sources, such as Google Translate.

## Transformer with Self-Attention Model [Oriana]:

A neural network is a model containing at least one hidden layer, which holds a set of neurons performing specific tasks [1].



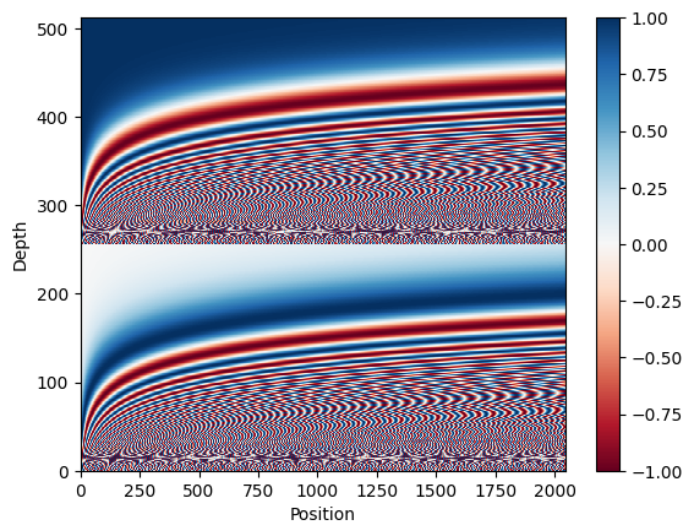
A deep neural network design, a model with more than one hidden layer, called a **transformer** was created at Google and relies on self-attention processes. It can be thought of as a stack of self-attention layers that includes either a decoder, an encoder, or both [1].

A series of embeddings is transformed into another by a **self-attention** layer [1]. Attention can be any of a vast variety of methods that collect information from a collection of inputs in a dependent manner [1]. An example of a typical attention mechanism is a weighted sum over a collection of inputs, where the weight for each input is determined by a different component. Self-attention refers to the process of paying *attention* to itself rather than another context [1].

Transformers are effective for modeling sequential data because they can be parallelized, therefore, computations can be done concurrently [2]. Additionally, transformers are capable of capturing far-reaching contexts because attention enables each place to access the entirety of the input at each given layer [2]. Comparatively, the information must go through numerous processing steps in RNNs and CNNs in order to travel a long distance, which makes learning more challenging. A Transformer model is similar to a Recurrent Neural Network (RNN) model, but instead of RNN layers, a Transformer uses self-attention layers [2].

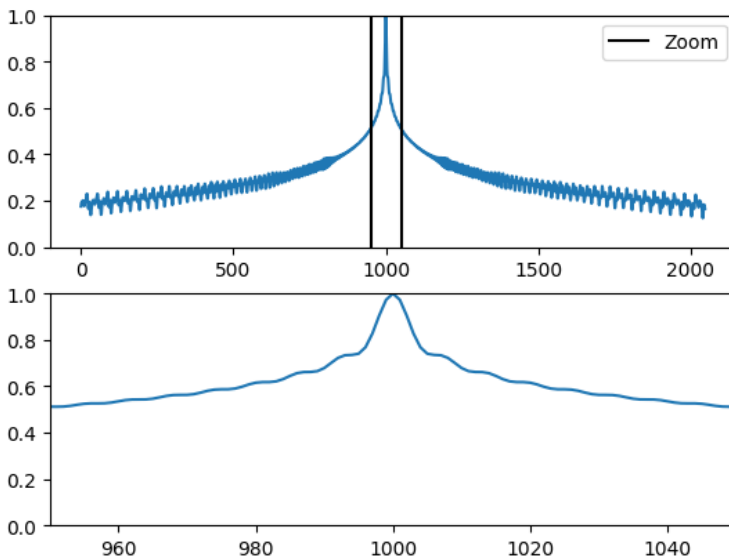
The model's attention layers interpret its input as a collection of unorganized vectors. It must recognize word order, though, or otherwise, the input sequences won't be recognizable from one another [2]. The embedding vectors are given a positional encoding by a Transformer, which is a stack of sines and cosines that vibrate at various frequencies depending on where they are along the embedding vector's depth [2].

Positional Vector Depth graph:



The position encoding vectors are normalized and the vector in position 1000 is compared to the others [2]. This is used to create the position embedding layer.

Dot Product Comparison graph:



I had to develop a subword vocabulary and a BERT tokenizer for this model. A subword tokenizer's key benefit is that it dynamically adjusts between tokenization that is word-based and that is character-based, meaning it can return to word fragments and individual characters for unknown words [3].

According to the International Standardization Organization's (ISO) 639 Language codes "cs" stands for Czech and "en" stands for English [4].

[1] "Machine Learning Glossary | Google Developers," *Google Developers*, Oct. 2022. [Online]. Available: <https://developers.google.com/machine-learning/glossary>. [Accessed: Nov. 06, 2022]

[2] M. Daoust, "Neural machine translation with a Transformer and Keras | Text | TensorFlow," *TensorFlow*, Nov. 2022. [Online]. Available: <https://www.tensorflow.org/text/tutorials/transformer>. [Accessed: Nov. 06, 2022]

[3] M. Daoust, "Subword tokenizers | Text | TensorFlow," *TensorFlow*, Jul. 2022. [Online]. Available: [https://www.tensorflow.org/text/guide/subwords\\_tokenizer](https://www.tensorflow.org/text/guide/subwords_tokenizer). [Accessed: Nov. 06, 2022]

[4] "ISO 639 — Language codes," *ISO*, 2022. [Online]. Available: <https://www.iso.org/iso-639-language-codes.html>. [Accessed: Nov. 06, 2022]

### **RNN with Attention Model [Zachary]:**

The recurrent neural network (RNN) model is based on the attention-based neural machine translation. The primary idea of training the model includes using sequence-to-sequence where

the model converts one sequence in a certain language to another sequence in another language. The RNN encoder and decoder map the source language vector to its representation vector in its target sequence. This process works fine for smaller and simpler sentences however, as the length and contextual complexity of the source sequences increases it becomes more difficult for the model to translate. Attention allows for more accurate translations with RNN because of the fact that the decoder has the functionality to look back at previous translations.

## **Summarization:**

NLP summarization is the task of breaking down large amounts of text into a paragraph or a few sentences to provide key information. As the field of NLP developed, summarization was split into two different subfields Extractive and Abstractive.

Extractive Summarization is the process of extracting key information from the text and using techniques like frequency to determine how relevant that information is. If the frequency is high enough or meets certain criteria then it will be added to the summarization paragraph and displayed to the user. This type of summarization does not involve generating new sentences, instead, sentences that appeared in the original text will appear in the summary as well. This has led to limited use of such a method even though useful it lacks the flexibility to make its predictions.

## **roBERTa Abstractive summarization Model [Anas]:**

The roBERTa model is based on the BERT model from hugging-face. roBERTa is an acronym for a robustly optimized BERT approach. It employs some enhancements like modifying the key hyperparameters and removing BERT's next sentence pertaining to objectives. roBERTa uses more intricate tokens that go deeper down to characters. This helps in extracting text and reconstructing an abstract summary for the model. When we say abstractive summary we target a summary that is concise and precise. Through its deeper learning techniques, roBERTa leverages more intricate training sessions and higher accuracy than that a BERT summarizer. It is solely for these advantageous properties that roBERTa was employed in this project. Although BERT travels through 1M steps in total. The detailed architecture of roBERTa allows it to perform a higher quality grade training within 500K steps.

## **T5-small Model [Dagar]:**

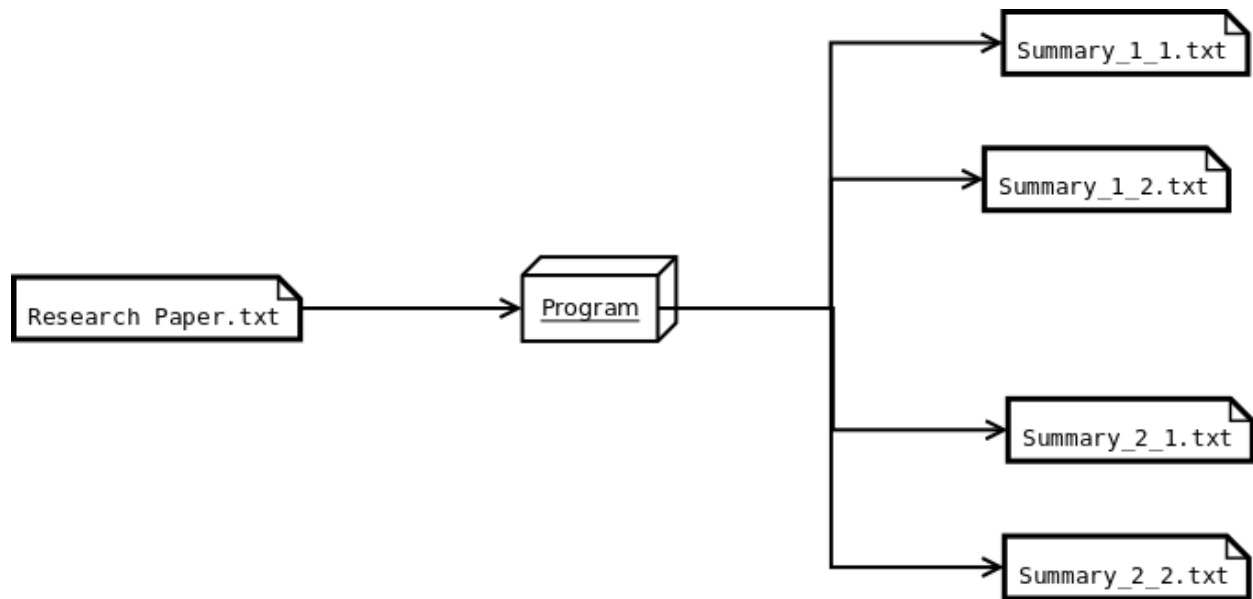
The T5 model is an encoder-decoder model that has been pre-trained on a huge dataset with supervised and unsupervised learning and can be finetuned downstream for specific tasks. In the case of the project, the weights were initialized to default to get rid of any of the training and were only trained on the science paper dataset. This has led to a significant impact on the

summarization of general articles but should have better performance for science paper-related summaries. In addition, the T5-small model converts all NLP problems to Text-to-Text format for better performance.

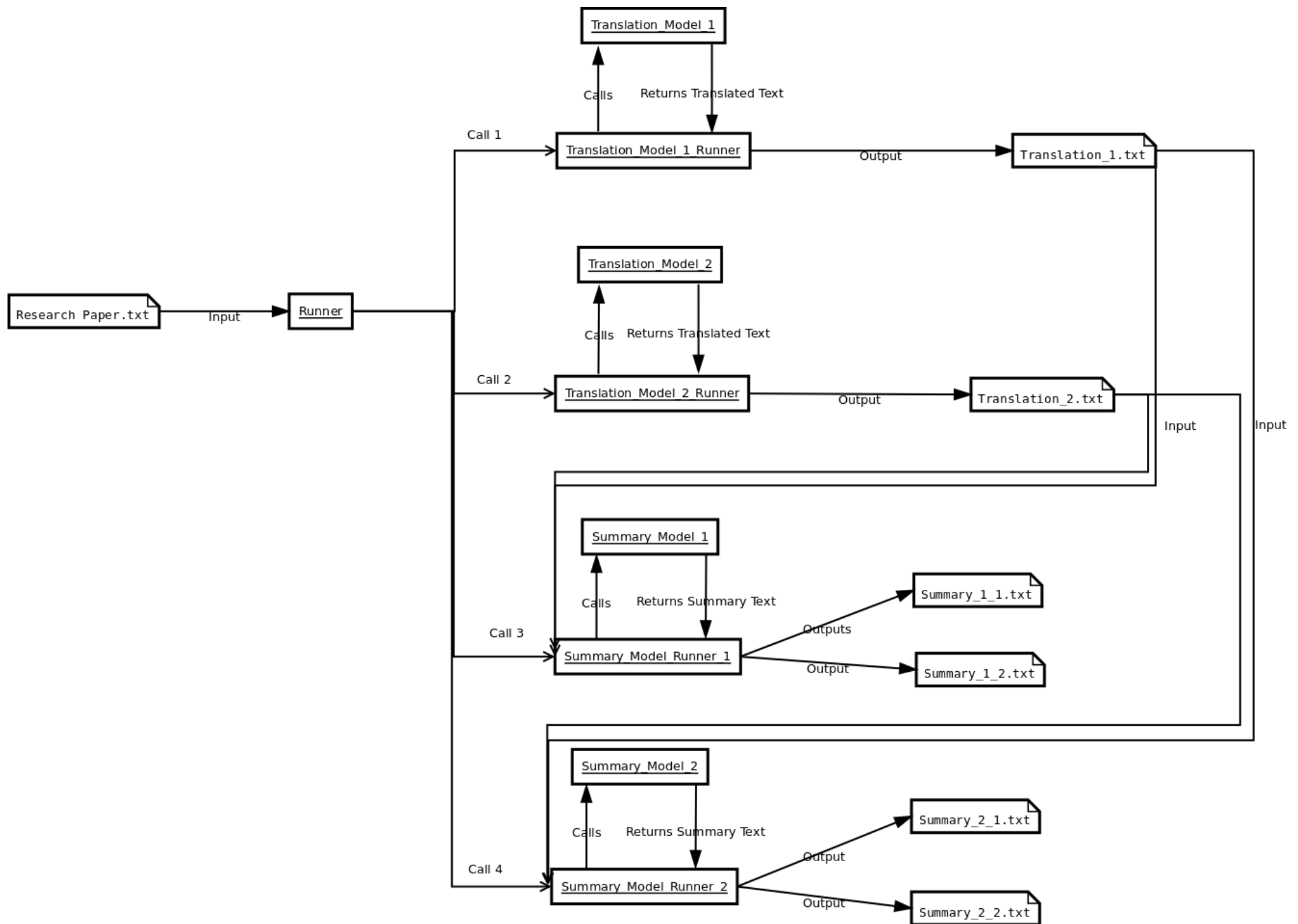
## **Conclusion:**

The combination of translation and summarization has been done before. This project focuses on the translation and summarization of research papers from all different types of fields and shall be fine-tuned for Computer Science related papers from the NIPS dataset.

## Design:



Takes in a paper and outputs 4 different summaries as there are 4 total models used in this project. Two models for translation and two models for summarization. Therefore we get 4 outputs. The intermediate outputs of the translation can also be seen in their respective folders.



The above diagram shows the in-depth flow of the program and any sub-modules that are used by each person are not included to cut the main flow of the program.

## Datasets:

### Translation Datasets

[WMT](#) is the main event for machine translation and research. The conference is held yearly in conjunction with bigger natural language processing conferences [1].

The [wmt19\\_translate](#) dataset is based on the data from statmt.org, versions exist for various years utilizing a combination of several data sources [2].

Czech to English [parallel sentences](#) [3].

Since the preprocessing steps are different between the two translation models and the data set from the RNN model was used by but not a part of TensorFlow, it was decided to use the same language dataset but different preprocessing to each model.

[1]“WMT,” *Machine Translate*, Sep. 2022. [Online]. Available: <https://machinetranslate.org/wmt>. [Accessed: Nov. 06, 2022]

[2]“wmt19\_translate | TensorFlow Datasets,” *TensorFlow*, Oct. 2022. [Online]. Available: [https://www.tensorflow.org/datasets/catalog/wmt19\\_translate](https://www.tensorflow.org/datasets/catalog/wmt19_translate). [Accessed: Nov. 06, 2022]

[3]“Tab-delimited Bilingual Sentence Pairs from the Tatoeba Project (Good for Anki and Similar Flashcard Applications),” *ManyThings*, 2022. [Online]. Available: <http://www.manythings.org/anki/>. [Accessed: Nov. 06, 2022]

### Summarization Datasets

For Training: [scientific\\_papers](#) subset: **arxiv** - contains papers that are not related to medicine. For each model the preprocessing was done in a different way as the models to improve performance.

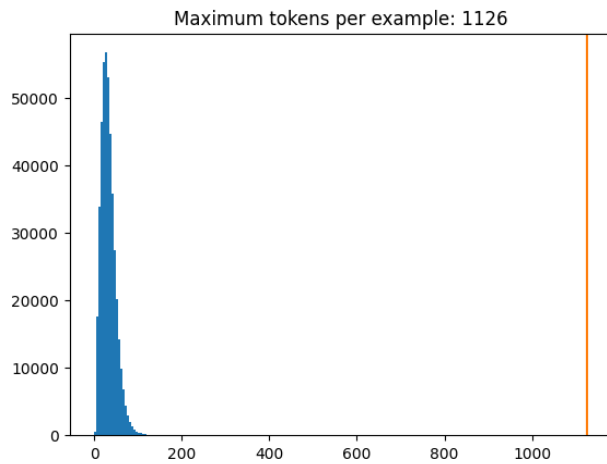
For Tuning: [All NeurIPS \(NIPS\) Papers](#): This dataset was preprocessed so that all the papers that lacked the abstract were removed. In addition, the papers.csv file from the dataset was used as it contains the abstract and papers.



## Analysis of Data:

### Transformer with Self-Attention Model [Oriana]:

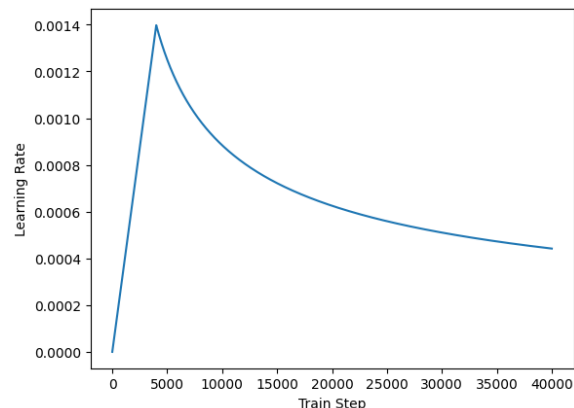
The token distribution per example from the dataset:



### Summary of the Model:

```
Model: "transformer"
Layer (type)          Output Shape          Param #
=====
encoder_1 (Encoder)    multiple              3631872
decoder_1 (Decoder)    multiple              5747968
dense_38 (Dense)       multiple              1005942
=====
Total params: 10,385,782
Trainable params: 10,385,782
Non-trainable params: 0
```

Custom learning rate by using the Adam optimizer:



Time to train all 20 epochs: approximately 3 days.

```
Epoch 1/20
3413/3413 [=====] - 16185s 5s/step - loss: 5.2601 - masked_accuracy: 0.2310 - val_loss: 4.6550 - val_masked_accu
racy: 0.2514
Epoch 2/20
3413/3413 [=====] - 11920s 3s/step - loss: 3.2000 - masked_accuracy: 0.4248 - val_loss: 3.9523 - val_masked_accu
racy: 0.3248
Epoch 3/20
3413/3413 [=====] - 11478s 3s/step - loss: 2.5722 - masked_accuracy: 0.5060 - val_loss: 3.6699 - val_masked_accu
racy: 0.3660
Epoch 4/20
3413/3413 [=====] - 11353s 3s/step - loss: 2.3103 - masked_accuracy: 0.5442 - val_loss: 3.4807 - val_masked_accu
racy: 0.3920
Epoch 5/20
3413/3413 [=====] - 11559s 3s/step - loss: 2.1569 - masked_accuracy: 0.5677 - val_loss: 3.3754 - val_masked_accu
racy: 0.4085
Epoch 6/20
3413/3413 [=====] - 11165s 3s/step - loss: 2.0526 - masked_accuracy: 0.5837 - val_loss: 3.2901 - val_masked_accu
racy: 0.4205
Epoch 7/20
3413/3413 [=====] - 12178s 4s/step - loss: 1.9755 - masked_accuracy: 0.5961 - val_loss: 3.2537 - val_masked_accu
racy: 0.4244
Epoch 8/20
3413/3413 [=====] - 12422s 4s/step - loss: 1.9151 - masked_accuracy: 0.6057 - val_loss: 3.1762 - val_masked_accu
racy: 0.4370
Epoch 9/20
3413/3413 [=====] - 10766s 3s/step - loss: 1.8676 - masked_accuracy: 0.6134 - val_loss: 3.1568 - val_masked_accu
racy: 0.4425
Epoch 10/20
3413/3413 [=====] - 10565s 3s/step - loss: 1.8271 - masked_accuracy: 0.6200 - val_loss: 3.1194 - val_masked_accu
racy: 0.4457
Epoch 11/20
3413/3413 [=====] - 10509s 3s/step - loss: 1.7928 - masked_accuracy: 0.6253 - val_loss: 3.0954 - val_masked_accu
racy: 0.4515
Epoch 12/20
3413/3413 [=====] - 10505s 3s/step - loss: 1.7642 - masked_accuracy: 0.6301 - val_loss: 3.0863 - val_masked_accu
racy: 0.4549
Epoch 13/20
3413/3413 [=====] - 10573s 3s/step - loss: 1.7377 - masked_accuracy: 0.6342 - val_loss: 3.0686 - val_masked_accu
racy: 0.4575
Epoch 14/20
3413/3413 [=====] - 10615s 3s/step - loss: 1.7152 - masked_accuracy: 0.6382 - val_loss: 3.0546 - val_masked_accu
racy: 0.4602
Epoch 15/20
3413/3413 [=====] - 10744s 3s/step - loss: 1.6939 - masked_accuracy: 0.6418 - val_loss: 3.0457 - val_masked_accu
racy: 0.4628
Epoch 16/20
3413/3413 [=====] - 10543s 3s/step - loss: 1.6752 - masked_accuracy: 0.6445 - val_loss: 3.0329 - val_masked_accu
racy: 0.4642
Epoch 17/20
3413/3413 [=====] - 10659s 3s/step - loss: 1.6586 - masked_accuracy: 0.6475 - val_loss: 3.0258 - val_masked_accu
racy: 0.4656
Epoch 18/20
3413/3413 [=====] - 10604s 3s/step - loss: 1.6428 - masked_accuracy: 0.6501 - val_loss: 3.0127 - val_masked_accu
racy: 0.4685
Epoch 19/20
3413/3413 [=====] - 10645s 3s/step - loss: 1.6282 - masked_accuracy: 0.6525 - val_loss: 3.0110 - val_masked_accu
racy: 0.4709
Epoch 20/20
3413/3413 [=====] - 10648s 3s/step - loss: 1.6153 - masked_accuracy: 0.6546 - val_loss: 3.0003 - val_masked_accu
racy: 0.4702
```

As you can see, it took a lot of processing power and time from my computer to train the model. I ran it through Jupyter Notebooks since Google Colab kept crashing and it would have taken about 14 hours for each epoch instead of 3. I would consider this an inefficient model to train with the resources and time I had, but according to our research, a Transformer model is supposed to be one of the better ones being used today for translation models. I wouldn't consider the final accuracy bad, but it's not good either, at about 65%.

### Running Inference Examples:

Example 1:

```
sentence = 'Poslyš, auto se rozbilo a brzdy potřebujou seřídít.'
ground_truth = 'The car broke and the engine needs fixing.'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

Input: : Poslyš, auto se rozbilo a brzdy potřebujou seřídít.  
Prediction : the telephone , cars , and the brakes need to secult .  
Ground truth : The car broke and the engine needs fixing.

Example 2:

```
sentence = 'Koukni, je mi to líto, vážně je.'
ground_truth = 'Look, I am sorry, I really am.'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

Input: : Koukni, je mi to líto, vážně je.  
Prediction : my continuity is a literary .  
Ground truth : Look, I am sorry, I really am.

Example 3:

```
sentence = 'Ale mělo by to být stabilní, dokud nedokončíme vstřikování.'
ground_truth = "But the site should remain stable until we finish the infusion."

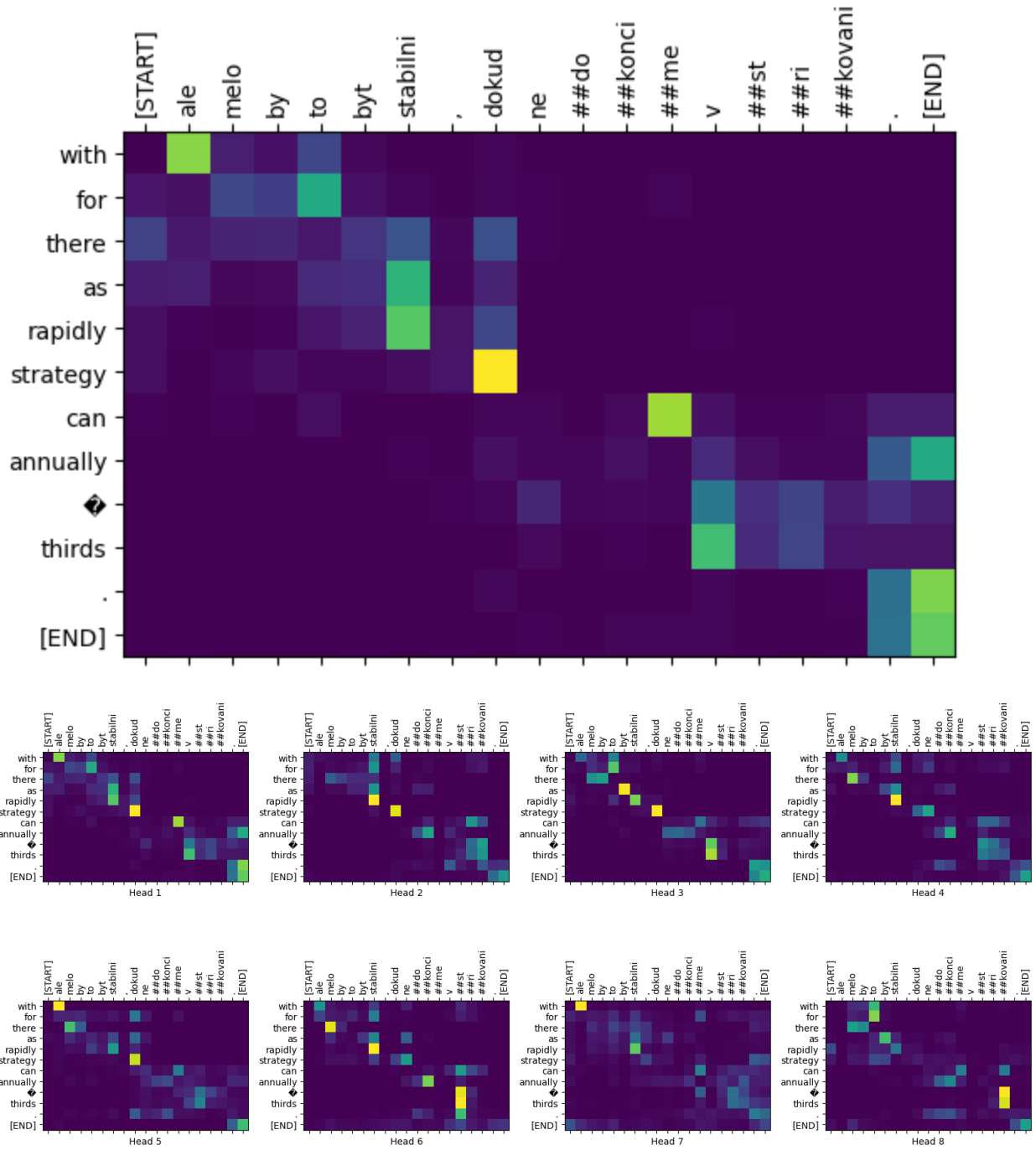
translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

Input: : Ale mělo by to být stabilní, dokud nedokončíme vstřikování.  
Prediction : but it should be stable until we complete the vaccine .  
Ground truth : But the site should remain stable until we finish the infusion.

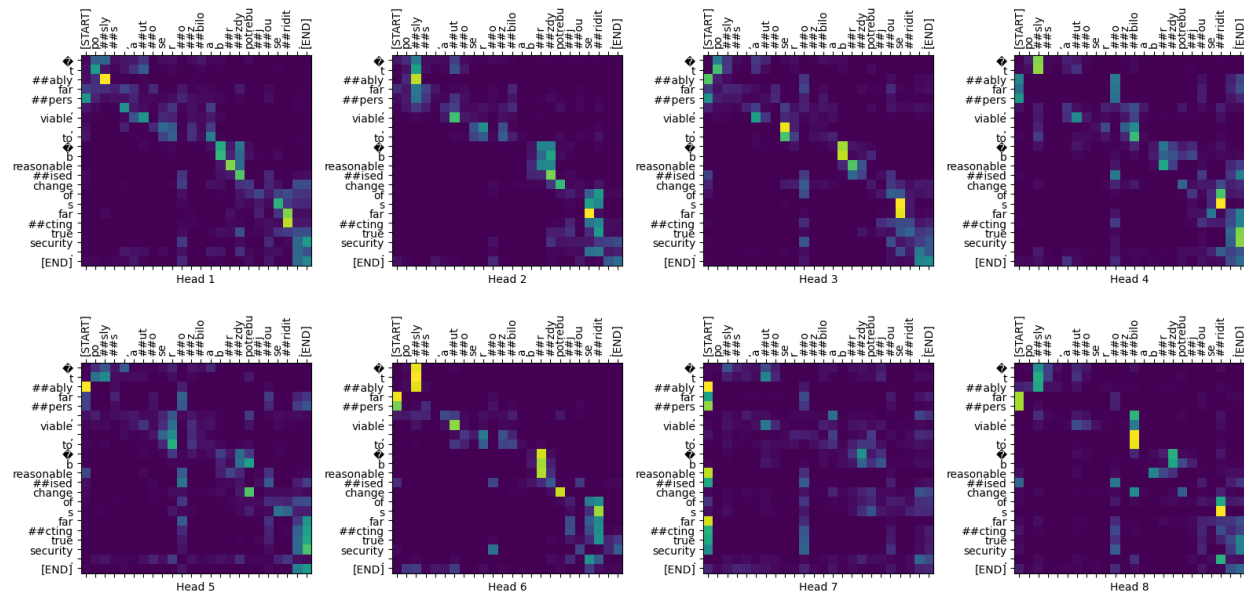
As you can see from the above examples, the predictions are slightly off from the ground truth. It gets the message of each sentence across, but not completely accurate.

The translator class I used while working on the model's notebook returns a dictionary of attention heatmaps that visualizes the model.

Visualized Attention Weights for Token Generation Plots:



The model can handle unfamiliar words, where the model attempts to transliterate them even without a shared vocabulary as seen below:



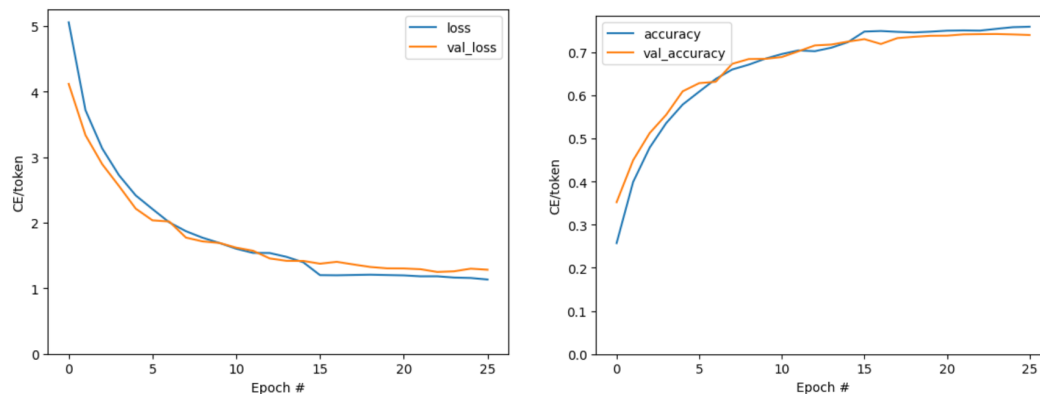
## RNN with Attention Model [Zachary]:

Training the RNN model was on a data set of around 300,000 sentences in both languages. Increasing the datasets could potentially result in more accurate results.

```
Epoch 1/500
100/100 [=====] - 29s 155ms/step - loss: 5.5573 - masked_acc: 0.1926 - masked_loss: 5.5573 - val_loss: 4.7990 - val_mi
Epoch 2/500
100/100 [=====] - 7s 67ms/step - loss: 4.3311 - masked_acc: 0.3034 - masked_loss: 4.3311 - val_loss: 3.8643 - val_mi
Epoch 3/500
100/100 [=====] - 6s 55ms/step - loss: 3.6930 - masked_acc: 0.3765 - masked_loss: 3.6930 - val_loss: 3.3605 - val_mi
Epoch 4/500
100/100 [=====] - 6s 58ms/step - loss: 3.2701 - masked_acc: 0.4326 - masked_loss: 3.2701 - val_loss: 2.9823 - val_mi
Epoch 5/500
100/100 [=====] - 6s 55ms/step - loss: 2.7165 - masked_acc: 0.5017 - masked_loss: 2.7200 - val_loss: 2.7007 - val_mi
Epoch 6/500
100/100 [=====] - 5s 53ms/step - loss: 2.4242 - masked_acc: 0.5436 - masked_loss: 2.4242 - val_loss: 2.5380 - val_mi
Epoch 7/500
100/100 [=====] - 5s 46ms/step - loss: 2.2475 - masked_acc: 0.5705 - masked_loss: 2.2475 - val_loss: 2.3522 - val_mi
Epoch 8/500
100/100 [=====] - 5s 47ms/step - loss: 2.1472 - masked_acc: 0.5918 - masked_loss: 2.1472 - val_loss: 2.1623 - val_mi
Epoch 9/500
100/100 [=====] - 5s 48ms/step - loss: 1.7964 - masked_acc: 0.6364 - masked_loss: 1.7987 - val_loss: 2.0971 - val_mi
Epoch 10/500
100/100 [=====] - 5s 47ms/step - loss: 1.5637 - masked_acc: 0.6705 - masked_loss: 1.5637 - val_loss: 2.0654 - val_mi
Epoch 11/500
100/100 [=====] - 4s 44ms/step - loss: 1.5491 - masked_acc: 0.6714 - masked_loss: 1.5491 - val_loss: 1.9958 - val_mi
Epoch 12/500
100/100 [=====] - 4s 45ms/step - loss: 1.5489 - masked_acc: 0.6723 - masked_loss: 1.5489 - val_loss: 1.9439 - val_mi
Epoch 13/500
100/100 [=====] - 5s 46ms/step - loss: 1.3604 - masked_acc: 0.7029 - masked_loss: 1.3624 - val_loss: 1.9145 - val_mi
Epoch 14/500
100/100 [=====] - 5s 54ms/step - loss: 1.0978 - masked_acc: 0.7422 - masked_loss: 1.0978 - val_loss: 1.9428 - val_mi
Epoch 15/500
100/100 [=====] - 4s 45ms/step - loss: 1.1555 - masked_acc: 0.7285 - masked_loss: 1.1555 - val_loss: 1.8738 - val_mi
Epoch 16/500
100/100 [=====] - 5s 49ms/step - loss: 1.1777 - masked_acc: 0.7277 - masked_loss: 1.1777 - val_loss: 1.9389 - val_mi
Epoch 17/500
100/100 [=====] - 5s 53ms/step - loss: 1.1388 - masked_acc: 0.7343 - masked_loss: 1.1400 - val_loss: 1.8665 - val_mi
Epoch 18/500
100/100 [=====] - 4s 45ms/step - loss: 0.8289 - masked_acc: 0.7915 - masked_loss: 0.8289 - val_loss: 1.8709 - val_mi
Epoch 19/500
100/100 [=====] - 4s 44ms/step - loss: 0.8776 - masked_acc: 0.7820 - masked_loss: 0.8776 - val_loss: 1.8922 - val_mi
```

The resulting model maintained around 70 to 80 percent masked accuracy. The model results in a fairly accurate translator on short translation data including short sentences, but begins to

decline in larger translations. One possible way to improve on this would be to use transformers to increase the models' learning.



Within some of the translation outputs for the RNN model some of the words are tokenized as Unknown (UNK). The text vectorization is a standardization function wrapped in the tensorflow keras text vectorization layer. This handles all vocabulary extraction and conversion of the text into tokens. The text vectorization layer has an 'adapt' method that initializes the layer based on the data in order to determine the vocabulary for each language.

```
max_vocab_size = 5000

context_text_processor = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_vocab_size,
    ragged=True)
```

The unknown tokens are used when the token identified is not in the defined vocabulary or when the token is not the correct type identified. This is often because the words in the translation language are too rare in the training data sets and therefore the model does not have enough information to add to the vocabulary.

```
# Here are the first 10 words from the vocabulary:
context_text_processor.get_vocabulary()[:10]

['', '[UNK]', '[START]', '[END]', ',', 'tom', 'se', 'to', 'je', 'jsem']

target_text_processor = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_vocab_size,
    ragged=True)

target_text_processor.adapt(train_raw.map(lambda context, target: target))
target_text_processor.get_vocabulary()[:10]

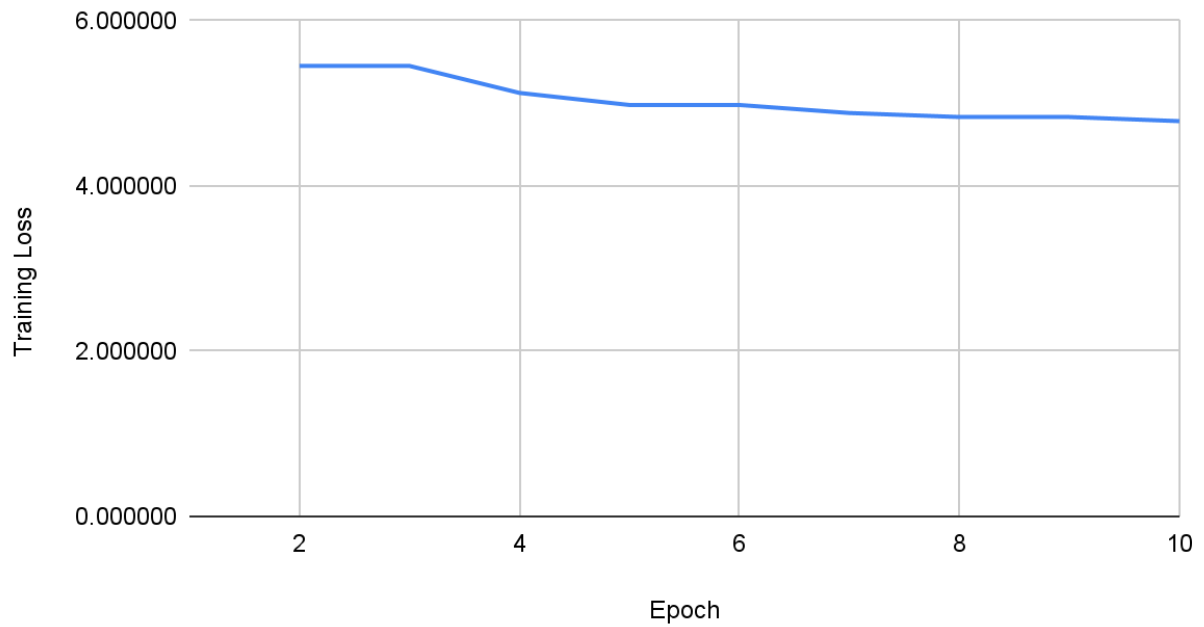
['', '[UNK]', '[START]', '[END]', 'tom', 'i', 'to', 'the', 'you', 'a']
```

### T5-small Model [Dagar]:

With the tuning of the T5 model to the NIPS dataset the results are provided in the below table:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	RougeL	RougeLsum	Gen Len
1	No log	5.185558	13.717200	2.064400	10.218900	12.838000	19.000000
2	5.452200	5.038340	15.621100	2.180800	11.356100	14.305400	19.000000
3	5.452200	4.948597	15.165900	2.330800	11.105200	13.945600	19.000000
4	5.125400	4.885084	15.716000	2.409900	11.495400	14.509900	19.000000
5	4.979400	4.845583	15.550700	2.426700	11.386700	14.323700	19.000000
6	4.979400	4.807345	15.840600	2.425400	11.687800	14.615400	19.000000
7	4.882300	4.787161	15.555400	2.463700	11.340100	14.318300	19.000000
8	4.833800	4.768018	15.478300	2.488800	11.336400	14.203100	19.000000
9	4.833800	4.762086	15.958000	2.566200	11.613900	14.657600	19.000000
10	4.783800	4.756572	15.706600	2.565400	11.467900	14.401700	19.000000

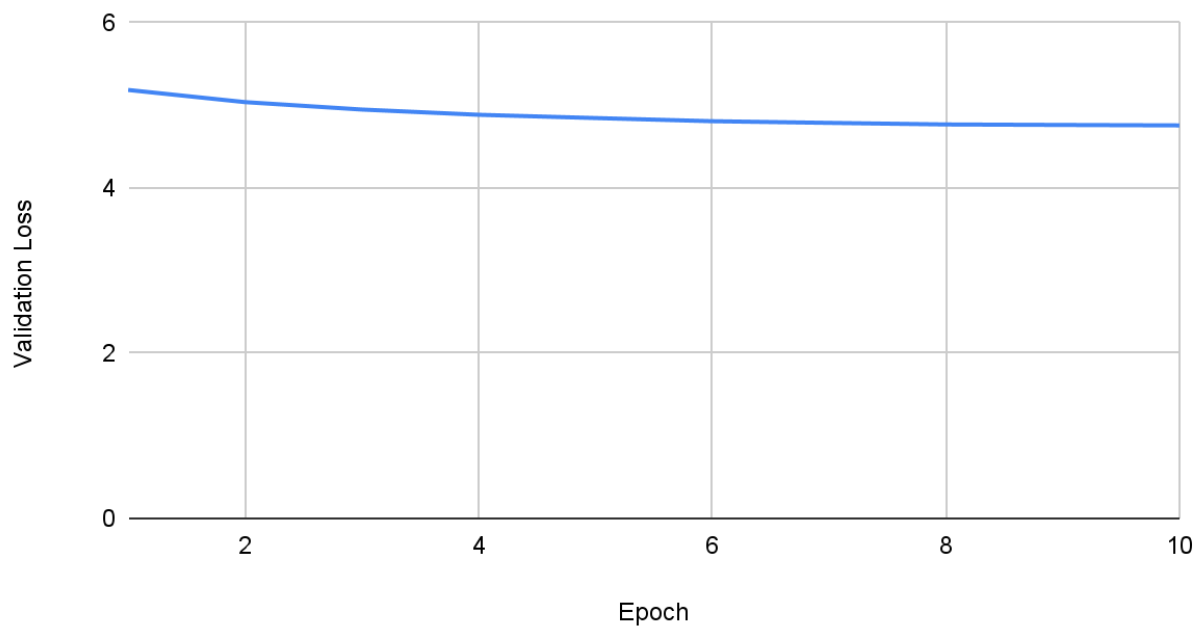
## Training Loss vs. Epoch



The training loss for the first 3 epochs seems to be constant and then begins to decrease slowly and stabilizes in the range of 6.0 and 4.0. If it was trained for more epochs it could be reduced but would lead to detrimental effects because of how small the dataset is.

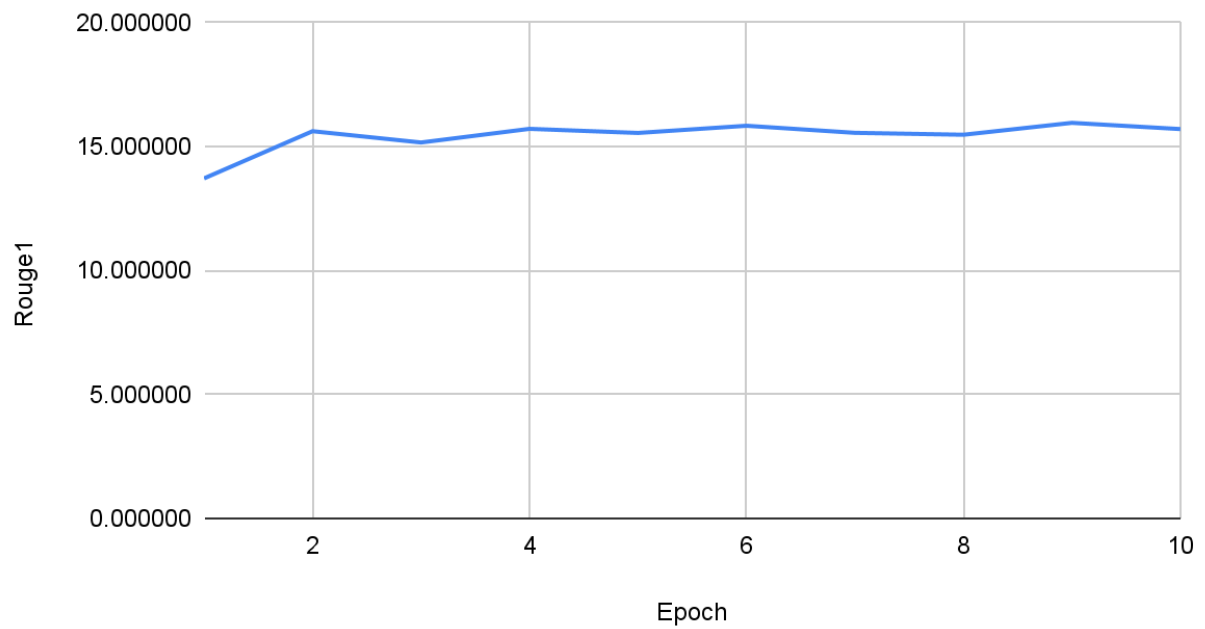


## Validation Loss vs. Epoch



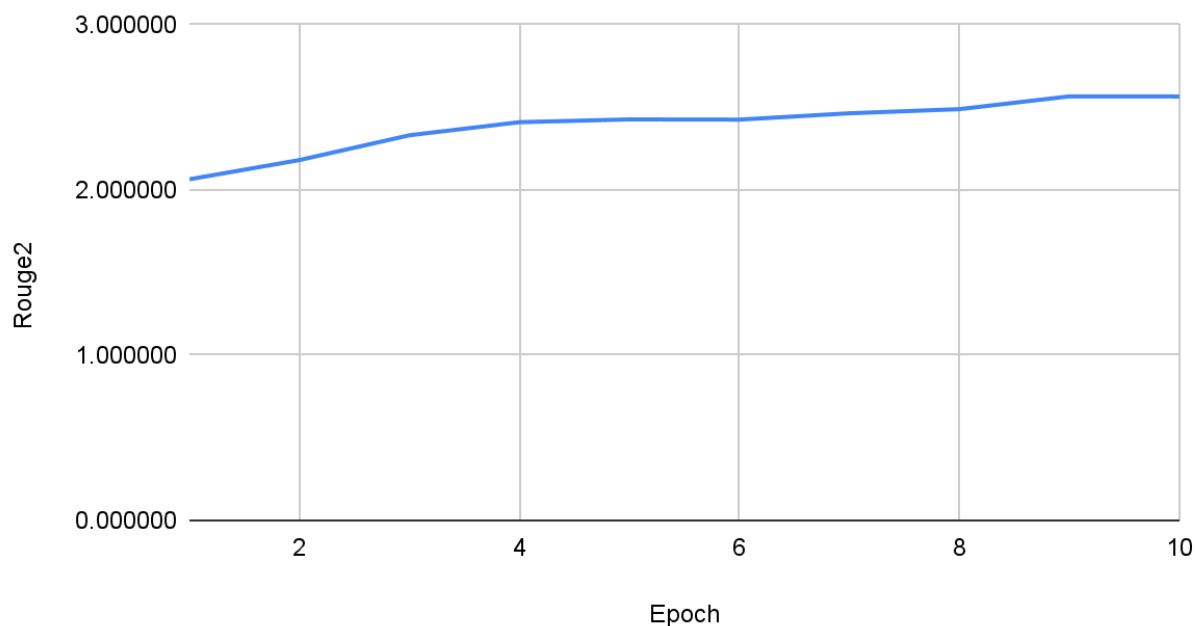
The validation loss plateaus around the value of 5 as well which parallels the Training loss. This is due to the NIPS dataset being very small compared to the training dataset.

## Rouge1 vs. Epoch



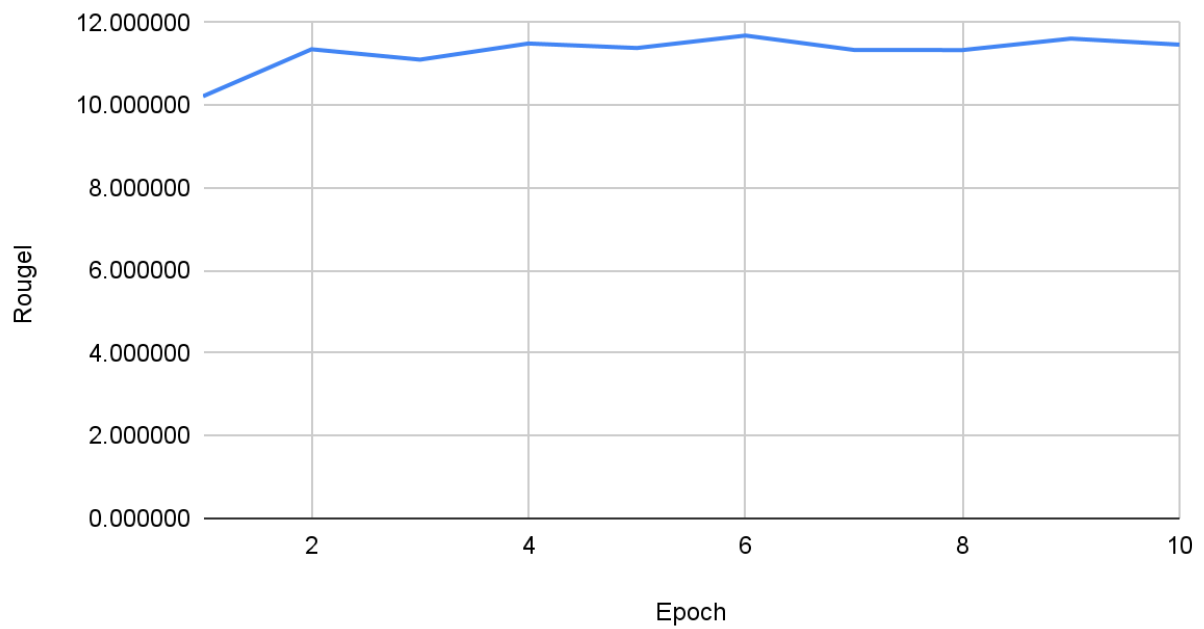
The Rouge1 score shows a good improvement for the 10 epochs which means that the NIPS dataset had a wide variety of word choices which lead to better 1-grams. From epochs 2 to 3 there seems to be some loss of precision related 1-grams but overall the model can recognize more 1-grams then before.

## Rouge2 vs. Epoch



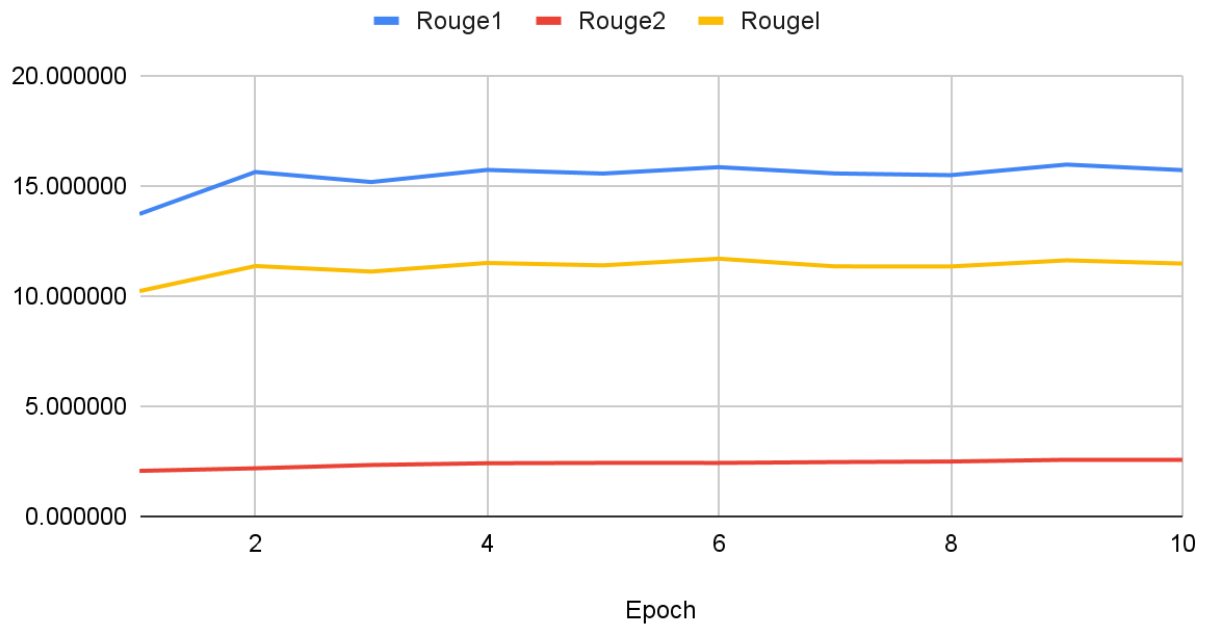
The Rouge2 shows the best improvement as the model has gotten better at recognizing 2-grams which is enough generally for the English language as long as the dependencies of words are close by. In addition, a score of 3 could be reached if the model was trained on a large general-purpose dataset to get a better 2-gram count as science papers to have very far away dependencies.

### Rougel vs. Epoch



The higher order n-grams for Rougel oscillate but never quite reach a score of 12 but being able to generate higher order n-grams is good as the model shows the potential to learn of further dependencies in sentences and be able to generate such sentences in its own use.

## Rouge1, Rouge2 and Rougel



Overall the recognition of 1-grams and n-grams became better but 2-grams could be improved with a larger dataset. When choosing the datasets we choose them based on the problem domain but we ignored general-purpose datasets which has to lead to a bottleneck in the T5 model as it struggles to generate 2 grams.

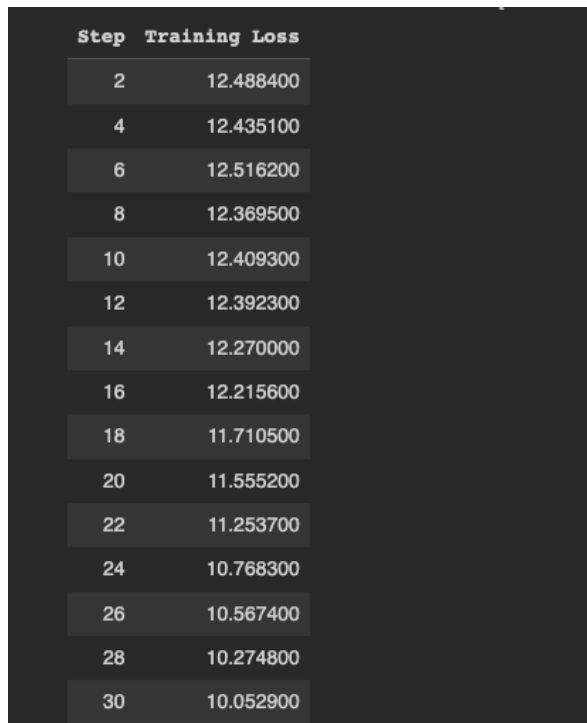
## roBERTa Model [Anas]:

### Analysis:

The roBERTa model tokenizes with a more detailed character level tokenization. What this means is that inevitably we have much more label strings to train with which means a reasonably bigger batch for the prediction steps. This is why even with the same dataset used as in with T5 summarization we see that a lot more steps coupled with training losses had been recorded in training/tuning the model.

Let's take a brief look at our results.

Fig: Reference of loss of training results with an increase in sample steps:



Step	Training Loss
2	12.488400
4	12.435100
6	12.516200
8	12.369500
10	12.409300
12	12.392300
14	12.270000
16	12.215600
18	11.710500
20	11.555200
22	11.253700
24	10.768300
26	10.567400
28	10.274800
30	10.052900

### Advantages:

From the above recordings it is pretty evident that for each step of prediction evaluation our training losses are reduced. If we dive down deeper into the entire training process we notice that the pre-trained roBERTa seq2seq model had run a total of 6446 steps with the 6444th step giving us a training loss of 0.0002 percent and the last step giving us a training loss of 0.00 percent. This is where the advantage of using finer tokens comes into place where we are able to achieve better results in lesser epochs(the number of times the parameters repeat the algorithm) and collectively more results with less effort.

### Disadvantages:

The noticeable challenge was with relaying the hyperparameters into a full batch. It was recorded that using a full batch of "1024" in the seq2seq trainer, while having the training

arguments parameter `evaluate_during_training=False`, overloaded the CUDA drivers. The GPU operated on Colab pro had a maximum capacity of 13.55gigs dedicated to PyTorch, and yet often resulted in errors due to a shortage of device capacity.

Fig. Shows load on CUDA GPUs:

```
[ ] torch.cuda.memory_summary(device=None, abbreviated=False)

'|=====|
| CUDA OOMs: 16 | cudaMalloc retries: 20 |=====| | | |
| Metric | Cur Usage | Peak Usage | Tot Alloc | Tot Freed |=====|
| ry | 13811 MB | 13852 MB | 205759 GB | 205746 GB |=====|
| B | 996 GB | 996 GB |=====|
| from large pool | 13743 MB | 13844 MB | 204763 GB | 204749 GB |=====|
| MB |=====|
| from small pool | 72 MB | 144 MB | 460 MB | 388 MB |=====|
| 270949 KB | 2061 MB | 129727 GB | 129727 GB |=====|
| GB | 1084 GB |=====|
| large pool | 790 | 1183 | 18879 K | 18878 K |=====|
|=====|
| from small pool | 1359 | 1366 | 15857 K | 15856 K |=====|
| 243 | 406 | 731 | 488 |=====|
| 194 |=====|
| ge pool | 105 | 170 | 12292 K | 12292 K |=====|
|=====|
| Oversize GPU segments | 0 | 0 | 0 | 0 |=====|
```

The CUDA cache was relieved and deleted, and yet not much data storage was available. After employing a few different approaches, the best course of action to take was to reduce the batch size while running the trainer.

Fig: Highlighting the use of a lesser batch size:

```
[ ] tokenizer = RobertaTokenizerFast.from_pretrained("roberta-base")
model = EncoderDecoderModel.from_pretrained("./checkpoint-6432")
model.to("cuda")
batch_size = 15

{
  "is_encoder_decoder": false,
  "label2id": {
    "LABEL_0": 0,
    "LABEL_1": 1
  },
  "layer_norm_eps": 1e-05,
  "length_penalty": 1.0,
  "max_length": 20,
  "max_position_embeddings": 514,
  "min_length": 0,
  "model_type": "roberta",
  "no_repeat_ngram_size": 0,
  "num_attention_heads": 12,
  "num_beam_groups": 1,
```

Ultimately the batch size was reduced greatly to just 15. It is to be noted that although this does not affect the ROUGE statistics, it would leave an impact in the summarized text predicted. The most optimum solution to this shortcoming would be to use dedicated GPU hardware to get the required results.

## Results:

Fig: Highlights the cumulative ROGUE scores for the tuned model:

```
[ ] print("ROUGE 1 SCORE: ",rouge.compute(predictions=pred_str, references=label_str, rouge_types=["rouge1"])[0].mid)
print("ROUGE 2 SCORE: ",rouge.compute(predictions=pred_str, references=label_str, rouge_types=["rouge2"])[0].mid)
print("ROUGE L SCORE: ",rouge.compute(predictions=pred_str, references=label_str, rouge_types=["rougeL"])[0].mid)

ROUGE 1 SCORE: Score(precision=0.01004346741221741, recall=0.05198938214563223, fmeasure=0.015292543907103079)
ROUGE 2 SCORE: Score(precision=3.846153846153846e-05, recall=0.00014285714285714284, fmeasure=6.060606060606061e-05)
ROUGE L SCORE: Score(precision=0.009937408309283316, recall=0.051745891608391686, fmeasure=0.015140888053522856)
```

As indicated in the above screenshot, a mean measure was taken to account for the ROGUE scores. The ROGUE-N scores help us understand better the scope precision of the system. It was observed that the highest scores were related to the bigrams in the sequence with both unigrams and LCS (Longest Common Subsequence) taking a hit in relative terms. In conclusion, it was realized that there is much scope for improvement in the suggested model. By feeding more hyperparameters and enhancing a batch size. Neither of these could be tackled in such a small-scale system.



## Implementation:

### Tuning:

#### T5-small Model [Dagar]:

First you need to unzip the dataset:

```
!unzip ./NIPS_archive.zip
```

These are the primary installs needed:

```
!pip install pandas
!pip install datasets
!pip install transformers
!pip install rouge-score nltk
```

These are the imports:

```
import datasets
from datasets import load_dataset
import pandas as pd
from datasets import Dataset, DatasetDict
from sklearn.model_selection import train_test_split
```

Then all the data was read from the papers.csv file:

```
df = pd.read_csv('papers.csv', index_col = "title")
```

Which was preprocessed to remove all the papers that did not have an abstract:

```
for ind in df.index:
    if pd.isnull(df.loc[ind, 'abstract']) or pd.isnull(df.loc[ind,
'full_text']):
        df.drop(ind, inplace = True)
```

Then it was converted into a training and test set:

```
train_set, test_set = train_test_split(df, test_size=0.2,
random_state=0)
```

```
train = Dataset.from_pandas(train_set)
test = Dataset.from_pandas(test_set)
```

```
dataset_tuning = DatasetDict()
dataset_tuning['train'] = train
dataset_tuning['test'] = test
```

```
dataset_tuning
```

The tuning process is the same as the training process but in the

```
def get_model(model_checkpoint, tokenizer):
```

Function instead of using a clean model we use the trained model from the science papers by

```
model =
AutoModelForSeq2SeqLM.from_pretrained("Dagar/t5-small-science-papers")
```

In addition the

```
preprocess_function(examples)
```

Function was modified to handle the NIPS dataset

### roBERTa Model [Anas]:

The Tuning for the roBERTa model was handled very similarly to the T5 model, although the repetition of the same instructions was not intended, it was deemed necessary to account for recording the tasks that were involved.

Fig:Shows the NIPS dataset was uploaded and unzipped.

```
[ ] !unzip NIPS_archive\ \ (1\).zip

Archive:  NIPS_archive (1).zip
  inflating: authors.csv
  inflating: papers.csv
```

Fig: Shows an additional installation of dill for serializing and deserializing objects:

```
pip install dill==0.3.4

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting dill==0.3.4
  Downloading dill-0.3.4-py2.py3-none-any.whl (86 kB)
    | 86 kB 5.7 MB/s
Installing collected packages: dill
```

Fig: Shows the dataframe had been split into two parts. train data and test data.

```
#split the dataframe into sets
train_set, test_set = train_test_split(df, test_size=0.2, random_state=0)

train = Dataset.from_pandas(train_set)
test = Dataset.from_pandas(test_set)

dataset_tuning = DatasetDict()

dataset_tuning['train'] = train
dataset_tuning['test'] = test

dataset_tuning
```

Fig: An additional parameter validation data was required to train the model.

```
val_data=Dataset.from_pandas(df[550000:555000])
```

Fig: Shows that the tokenization process was adjusted to match the new dataset.

```
[ ] tokenizer = RobertaTokenizerFast.from_pretrained("roberta-base")
tokenizer.bos_token = tokenizer.cls_token
tokenizer.eos_token = tokenizer.sep_token

#parameter setting
batch_size=256 #
encoder_max_length=40
decoder_max_length=8

def process_data_to_model_inputs(batch):
    # tokenize the inputs and labels
    inputs = tokenizer(batch["abstract"], padding="max_length", truncation=True, max_length=encoder_max_length)
    outputs = tokenizer(batch["full_text"], padding="max_length", truncation=True, max_length=decoder_max_length)

    batch["input_ids"] = inputs.input_ids
    batch["attention_mask"] = inputs.attention_mask
    batch["decoder_input_ids"] = outputs.input_ids
    batch["decoder_attention_mask"] = outputs.attention_mask
    batch["labels"] = outputs.input_ids.copy()

    # because RoBERTa automatically shifts the labels, the labels correspond exactly to 'decoder_input_ids'.
    # We have to make sure that the PAD token is ignored
    batch["labels"] = [[-100 if token == tokenizer.pad_token_id else token for token in labels] for labels in batch["labels"]]

    return batch

#processing training data
train_data = train_data.map(
    process_data_to_model_inputs,
    batched=True,
    batch_size=batch_size,
    remove_columns=["abstract", "full_text"]
)
train_data.set_format(
    type="torch", columns=["input_ids", "attention_mask", "decoder_input_ids", "decoder_attention_mask", "labels"],
)

#processing validation data
val_data = val_data.map(
    process_data_to_model_inputs,
    batched=True,
    batch_size=batch_size,
    remove_columns=["abstract", "full_text"]
)
val_data.set_format(
    type="torch", columns=["input_ids", "attention_mask", "decoder_input_ids", "decoder_attention_mask", "labels"],
)

Downloading: 100% 899k/899k [00:00<00:00, 12.5MB/s]
```

## Overall:

### Setup [All]:

Installation:

```
#Zachary
!pip install "tensorflow-text>=2.10"
!pip install einops

#Oriana
!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
!pip install -q -U tensorflow-text
!pip install matplotlib

#Anas
!git clone https://github.com/google/seq2seq.git
!pip install -e seq2seq
!pip install dill==0.3.4
!pip install datasets==1.0.2
!rm seq2seq_trainer.py
!wget
https://raw.githubusercontent.com/huggingface/transformers/master/examples/seq2
seq/seq2seq_trainer.py

#Dagar
!pip install transformers
```

Imports:

```
#Oriana
import logging
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow as tf
import tensorflow_text
import shutil

#Zachary
import numpy as np
import typing
```

```

from typing import Any, Tuple
import einops
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import tensorflow as tf
import tensorflow_text as tf_text
import pathlib

#Anas
#[insert here]

#Dagar
import torch
import json
from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

```

The function is used to write to files for all of the models:

```

def write_to_file(filepath, text):
    file1 = open(filepath, "w")
    file1.write(text)
    file1.close()

```

### Transformer with Self-Attention Model [Oriana]:

Unzip tar.gz file:

```
!tar -xzf /content/translator.tar.gz
```

Testing:

```

Transformer = tf.saved_model.load('./translator')
Transformer('Kde je toaleta').numpy()

```

b'where toilets are'

```

Transformer('Ale mělo by to být stabilní, dokud nedokončíme
vstříkování.').numpy()

```

b'but it should be stable until we complete the vaccine .'

```
sentence = 'Ale mělo by to být stabilní, dokud nedokončíme vstřikování.'  
ground_truth = 'But the site should remain stable until we finish the  
infusion.'
```

```
print(f'{"Input":15s}: {sentence}')
```

```
print(f'{"Prediction":15s}:  
{Transformer(sentence).numpy().decode("utf-8")}')  
print(f'{"Ground truth":15s}: {ground_truth}')
```

Input: : Ale mělo by to být stabilní, dokud nedokončíme vstřikování.  
Prediction : but it should be stable until we complete the vaccine .  
Ground truth : But the site should remain stable until we finish the infusion.

```
text = open('papers.txt', 'r')  
lines = text.readlines()  
text.close()  
  
print("Czech Lines:\n")  
for l in lines:  
    print(l)  
  
print("\nEnglish Lines:\n")  
for l in lines:  
    print(Transformer(l).numpy().decode("utf-8"))  
  
print("\nEnglish Lines w/ End Lines:\n")  
for l in lines:  
    print(f'{Transformer(sentence).numpy().decode("utf-8")}\n')  
  
list = ''  
print("\nJoining English Lines:\n")  
for l in lines:  
    string = str(Transformer(l).numpy().decode("utf-8"))  
    list += '\n'  
    list += string  
print(list)  
  
print("\nJoining List:\n")  
joined = ''.join(list)
```

```
print(joined)
```

Czech Lines:

Ale mělo by to být stabilní, dokud nedokončíme vstříkování.

Ale mělo by to být stabilní, dokud nedokončíme vstříkování.

Ale mělo by to být stabilní, dokud nedokončíme vstříkování.

Ale mělo by to být stabilní, dokud nedokončíme vstříkování.

English Lines:

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

English Lines w/ End Lines:

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

Joining English Lines:

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

Joining List:

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

but it should be stable until we complete the vaccine .

Translation Model 1 Runner:

```
Transformer = tf.saved_model.load('/content/translator')
```

```
# First Paramater - the path to the file that needs to be translated
# Second paramter - where the output needs to be written to
# only this function will be called by main()
```

```
def Translation_Model_1_Runner(input_filepath, output_filepath):
```

```
    # read function
```

```
    lines = Translation_Model_1_Read_File(input_filepath)
```

```
    # adding predicted lines to list
```

```
    list = ''
```

```
    for l in lines:
```

```
        strings = str(Transformer(l).numpy().decode("utf-8"))
```

```
        if list:
```

```
            list += '\n'
```

```
            list += strings
```

```
        else:
```

```
            print("Making Predications...")
```

```
            list += strings
```

```
    # combining lines from list
```

```
    translated_text = ''.join(list)
```

```
    # write function
```

```
    write_to_file(output_filepath, translated_text)
```

```
def Translation_Model_1_Read_File(filepath):
```

```
    text = open(filepath, 'r')
```

```
    lines = text.readlines()
```

```
    text.close()
```

```
    return lines
```

**RNN with Attention Model [Zachary]:**



For this second increment the goal has been to put each of our models together. Using the function, 'Translation\_Model\_2\_Runner' My language model is called to translate the input file.

First importing the model

```
[ ] !tar -xzf /content/ZAC_RNN_translator.tar.gz
```

Then saving the model to RNN\_model for translation.

```
def Translation_Model_2_Runner(input_filepath, output_filepath):
    translated_text = "Hello World 2!!!!"

    #do the stuff you need here
    RNN_model = tf.saved_model.load('/content/translator') #make su

    text = Translation_Model_2_Read_File(input_filepath)
    result = ''
    for t in text:
        result += ' ' + RNN_model.translate([t])[0].numpy().decode()

    write_to_file(output_filepath, result)
```

The program will get the input text file, read in each line and pass each word into the translator.

**roBERTa Model [Anas]:**

[insert here]

**T5-small Model [Dagar]:**

The function that sets up and runs the model is called:

```
Summarization_Model_2_Runner
```

And it takes filepath as a parameter

The line below initialize the tuned T5-small model based on the NIPS dataset

```
model =
AutoModelForSeq2SeqLM.from_pretrained("Dagar/t5-small-science-papers-NIPS"
)
```

The tokenizer is set to the default pre-trained model from hugging face:

```
tokenizer = AutoTokenizer.from_pretrained('t5-small')
```

In addition the torch device is set to cpu as there is no training involved:

```
device = torch.device('cpu')
```

The code below preprocesses the data from a file and gets it ready for the tuned T5 model:

```
preprocess_text = text.strip().replace("\n", "")  
t5_prepared_Text = "summarize: "+preprocess_text
```

Then to summarize some parameters need to be set based on the tuning done:

```
summary_ids = model.generate(tokenized_text,  
                             num_beams=4,  
                             no_repeat_ngram_size=2,  
                             min_length=30,  
                             max_length=128,  
                             early_stopping=True)
```

And finally the text is decoded and written to a file

```
summarized_text = tokenizer.decode(summary_ids[0],  
skip_special_tokens=True)  
  
write_to_file(output_filepath, summarized_text)
```

## Runner [All]:

The runner primary job is to call the models using their predefined function names

```
Translation_Model_1_Runner(input_filepath,  
"./Oriana/Translation_1.txt")  
  
Translation_Model_1_Runner(input_filepath,  
"./Zachary/Translation_2.txt")  
  
Summarization_Model_1_Runner("./Oriana/Translation_1.txt",  
"./Anas/Summarization_1_1.txt")  
Summarization_Model_1_Runner("./Zachary/Translation_2.txt",  
"./Anas/Summarization_1_2.txt")
```

```
Summarization_Model_2_Runner("./Oriana/Translation_1.txt",  
"./Dagar/Summarization_2_1.txt")  
Summarization_Model_2_Runner("./Zachary/Translation_2.txt",  
"./Dagar/Summarization_2_2.txt")
```

The setup for each model is done inside the function and the output is also handled by each function. The primary job of the runner is just calling each function with some data that each function needs to work such as the file path where the original paper is or where each translated paper is stored and the output paths.

There is also a cleanup to remove all files after being done:

```
!rm -r Oriana  
!rm -r Zachary  
!rm -r Anas  
!rm -r Dagar
```

## Results:

### *Given the input in Chez*

Anotace: Kooperativní vnímání poskytuje nový způsob, jak překonat omezení snímání u jediného automatizovaného vozidla a potenciálně zvyšuje bezpečnost jízdy. Ke snížení objemu přenosových dat používají stávající řešení mezilehlá data generovaná modely konvolučních neuronových sítí (CNN), jmenovitě mapy funkcí, k dosažení kooperativního vnímání. Mapy funkcí jsou však příliš velké na to, aby mohly být přenášeny současnou technologií V2X. Navrhujeme nový přístup, nazvaný Slim-FCP, který výrazně snižuje velikost přenosových dat. Umožňuje kodéru funkcí kanálů odstranit irelevantní funkce pro lepší kompresní poměr. Kromě toho přijímá inteligentní strategii výběru kanálů, pomocí které jsou pro přenos vybírány pouze reprezentativní kanály map funkcí.

Abychom vyhodnotili efektivitu Slim-FCP, definujeme dále metriku poměru recall-to-bandwidth (RB), abychom kvantitativně změřili, jak se mění vyvolání detekce objektů s ohledem na dostupnou šířku pásma sítě. Výsledky experimentu ukazují, že Slim FCP snižuje velikost přenosových dat o 75% ve srovnání s nejlepším nejmodernějším řešením, s jemnou ztrátou při vyvolání detekce objektů. Indexové termíny - automatizovaná vozidla, kooperativní vnímání, detekce 3D objektů, fúze rysů.

Systém vnímání automatizovaných vozidel (AV) umožňuje vozidlu shromažďovat informace a získávat relevantní znalosti z prostředí, např. detekovat objekty. Kooperativní vnímání na druhé straně umožňuje vozidlům vzájemně sdílet data místního vnímání. Hlavním důvodem pro kooperativní vnímání je maximalizace zorného pole a zorného pole automatizovaných vozidel.

Rozšířené zorné pole automatizovaných vozidel výrazně zlepší svolávací akci při detekci objektů na automatizovaných vozidlech. Hlavní výzvy při dosahování tohoto cíle na propojených a automatizovaných vozidlech (CAV) spočívají v přenosu obrovského množství bohatých dat ze senzorů mezi vozidly. A. Hlavní výzvy Existují dvě technické výzvy, které musíme překonat při navrhování účinného a efektivního řešení kooperativního vnímání. První výzvou je zmenšení mapy funkcí na prostorových a kanálových doménách s menším obětováním na

představení. I když ne všechny funkce jsou pro úlohu detekce objektů smysluplné, lze nepodstatné funkce před přenosem odebrat, aby se ušetřily komunikační prostředky. Navíc, pokud lze přesně identifikovat relevantní prvky a nahradit je referenčním číslem (např. nula), mohou být mapy funkcí dále komprimovány, aby se ušetřila šířka pásma sítě. Poruchy na mapách prvků způsobené odstraněním irelevantních prvků však mohou ovlivnit detekční odvolání modelu detekce objektů, což snižuje výkon kooperativního vnímání. Jinými slovy, odstranění funkcí by nemělo nadměrně obětovat výkon, což nám ponechává velkou výzvu při navrhování správného mechanismu komprese a odstraňování funkcí.

Druhá výzva spočívá v obtížnosti výběru kanálů pro výměnu map mezi vozidly pro kooperativní vnímání. Zatímco F-Cooper snižuje objem transmission data tím, že vysílá pouze podmnožinu map, jak vybrat nejlepší sadu kanálů pro kooperativní vnímání zůstává otevřenou otázkou. Výběr vhodných kanálů může vést k výraznému poklesu detekčního výkonu, protože sémantické informace poskytované přijatými mapami funkcí by mohly být nedostatečné. Objem sémantických informací přenášených různými kanály se liší a identifikace reprezentativních kanálů map funkcí nám pomáhá dosáhnout lepšího výkonu s menšími potřebnými zdroji.

Abychom vyřešili výše uvedené výzvy, navrhujeme pro propojená automatizovaná vozidla odlehčené kooperativní vnímání založené na funkcích (Slim-FCP). Architektura Slim FCP je znázorněna na obr. 1, který obsahuje tři hlavní komponenty: kanálový kodér a dekodér funkcí, irelevantní odstraňovač funkcí a výběr kanálu na mapách prvků. S Slim-FCP budou mapy funkcí vytvořené modelem CNN (Convolutional Neural Network) zakódovány a výrazně komprimovány před jejich sdílením. Zde předpokládáme, že dva CAV využívají stejný detekční model, protože mnoho prací naznačuje, že modely na zařízeních mohou být pravidelně aktualizovány z cloudových nebo okrajových serverů. Zatímco F-Cooper nahrazuje původní surová data fyzickými mapami, aby dosáhl kooperativního vnímání, tvrdíme, že

*The T5 model generates the first summary based on the Transformer with Self-Attention Model was*

*“We present a novel method for generative models of the k-cpc, i.e., the model is based on the data-to-sapsepherical function, which consists of spn, and p-based approaches. We show that our method can be able to learn the compositional data, with the first em-nocs na, or the number of data. The resulting data set is used to derive the same data for the two-dimensional data and the other hand. ”*

*The T5 model generates the first summary based on the RNN with Attention Model was*

“deep deep neural neural networks networks from from to to the the problem problem [ [ ]]  
correlation correlation with with smoking smokingometricmetric information information star  
star from the oisismm information from. 1. 1. [ from [= state stateo] [ with [ correlation [] state  
information [ smoking [ potential potential from state to m]iso [ state] [ gas gas] disc disc [ “

*The roBERTa model generates the first summary based on the Transformer with Self-Attention Model was*

*“anomalys : cooperative perceptions provide a new way to overcome the use of the use of automated vehicles and potentially increases safety of existing solutions . to address the higher - level challenges , we propose automated automated vehicles to affirm affirmative sense of function ( slim - fcp ) .”*

*The roBERTa model generates the first summary based on the RNN with Attention Model was*

*“[UNK] van [UNK] [UNK] the [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] the spider spun the [UNK] to [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] the [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] van [UNK] [UNK] the [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] the recession caused [UNK] to the [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] the recession caused [UNK] to [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] he is studying ”*

## Project Management:

- Implementation Status Report (**Oriana Borges-Salinas**)
  - Work Completed
    - Description - Finished training the Transformer with Self Attention model, exported model, visualized results, and integrated with the overall design.
    - Responsibility (Task, Person)- helping with others' tasks, getting the models to function, and integrating them to the combined notebook.
    - Contributions - Helping with translation models, organizing and updating GitHub, and combining working code.
    - Issues / Concerns - My computer stopped working and I had to send it to the shop for repair. I had to work from a borrowed computer, but the models downloaded from GitHub weren't working. It kept giving me an error that Zach also had. I searched everywhere online but couldn't find anything. Eventually, Dagar and Zach found out that zipping the files was the problem since bytes were being lost. I had to convert the models into tar.gz format and fix the git attributes for our Git Large File Storage. Once I fixed all that, I uploaded our models to GitHub and did some data cleaning/organization before moving on to the integration stage.
- Implementation Status Report (**Zachary Chenausky**)
  - Work Completed
    - Description - Worked on RNN TensorFlow translation model, and exported and saved the model as needed in the group project.
    - Responsibility (Task, Person)- My responsibility was to have a working translation model for the other group members to use and run on their summarization models.
    - Contributions - The RNN TensorFlow model translates Czech to English that can be outputted to a file for testing.
    - Issues / Concerns - Some of the concerns are that because the model does not use transformers the accuracy for larger datasets is decreased. With relatively good accuracy on short sentences, the accuracy decreases when the translation text increases.
- Implementation Status Report (**Anas Mohammed Nayeem**)
  - Work Completed
    - Description - Worked on roBERTa based summarization model to train, tune and implement it, Aim was to ensure the model summarizes the text and is also able to evaluate Rouge metrics to define its usability and precisions. Tuned the model with the NIPS dataset for an enhanced summarization result.
    - Responsibility (Task, Person)- Ensure that the system is able to summarize the text translated. Work on the integration of the project for the roBERTa module and store text in the necessary file.

- Contributions - Built the model, ensured its working and ran a few preliminary test cases. Helped in contributions to the research paper as well as the integrated python files to ensure that the system is up and running.
  - Issues / Concerns - The primary concern recorded was with the inability to scope output to the expectations set. This was primarily because of the lack of GPU beef required to actually run the model to its optimum capacity. Another concern was with running the model for different datasets. The NIPS dataset I initially used was rather outdated. Though it had the same design as the more recent one used. A lot of unpredicted issues had to be challenged to match the labels with prediction tokens.
- Implementation Status Report (**Dagar Rehan**)
  - Work Completed
    - Description - Work on one of the summarization models to train and implement it. Get statistics to compare initial results for the model chosen. Work on the overall design of how to implement the models after training them all. Finish turning the T5 model to the NIPS dataset for better results regarding research papers.
    - Responsibility (Task, Person) - Train the T5 model for summarization. Implement the T5 model for summarization. Get results using the ROUGE benchmark. I will help with merging the modules together and fine-tuning the T5 model for better performance on the NIPS dataset. I will also convert the NIPS dataset to the T5 model format. I also analyzed the results for the ROUGE score for the T% model and came up with the graphs.
    - Contributions - Trained T5 model from scratch. Made diagrams for the system. Implemented T5 model. Helped with the Background, Related Work, and Design section of the doc. Also made graphs to show ROUGE scores for T% model and formatted the combined google notebook to make sure it all worked together.
    - Issues / Concerns - I don't know if we should train our models on some general article to help the model learn better sentence structure. I think fine-tuning the summarization models on general articles and the NIPS dataset would be better. I don't know if you can only tune with the NIPS dataset or add a general dataset as well. To fine-tune the dataset to NIPS we need to convert the dataset to our own format with each model. In addition, the model fails to summarize anything that is not a research paper which can have detrimental effect if it is given input that is not a research paper as the output looks very similar to a research paper.



## Resources:

- Papers:

[A Survey on NLP-based Text Summarization for Summarizing Product Reviews Summarising company announcements](#)

[Automatic Summarizing the News from Inform.kz by Using Natural Language Processing Tools](#)

[Analysis of Learning Approaches for Machine Translation Systems](#)

[Intro to ROUGE](#)

Background:

[1] “Machine Learning Glossary | Google Developers,” *Google Developers*, Oct. 2022. [Online]. Available: <https://developers.google.com/machine-learning/glossary>. [Accessed: Nov. 06, 2022]

[2] M. Daoust, “Neural machine translation with a Transformer and Keras | Text | TensorFlow,” *TensorFlow*, Nov. 2022. [Online]. Available: <https://www.tensorflow.org/text/tutorials/transformer>. [Accessed: Nov. 06, 2022]

[3] M. Daost, “Subword tokenizers | Text | TensorFlow,” *TensorFlow*, Jul. 2022. [Online]. Available: [https://www.tensorflow.org/text/guide/subwords\\_tokenizer](https://www.tensorflow.org/text/guide/subwords_tokenizer). [Accessed: Nov. 06, 2022]

[4] “ISO 639 — Language codes,” *ISO*, 2022. [Online]. Available: <https://www.iso.org/iso-639-language-codes.html>. [Accessed: Nov. 06, 2022]

Datasets:

[1] “WMT,” *Machine Translate*, Sep. 2022. [Online]. Available: <https://machinetranslate.org/wmt>. [Accessed: Nov. 06, 2022]

[2] “wmt19\_translate | TensorFlow Datasets,” *TensorFlow*, Oct. 2022. [Online]. Available: [https://www.tensorflow.org/datasets/catalog/wmt19\\_translate](https://www.tensorflow.org/datasets/catalog/wmt19_translate). [Accessed: Nov. 06, 2022]

[3] “Tab-delimited Bilingual Sentence Pairs from the Tatoeba Project (Good for Anki and Similar Flashcard Applications),” *ManyThings*, 2022. [Online]. Available: <http://www.manythings.org/anki/>. [Accessed: Nov. 06, 2022]

Tensorflow:

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster,

Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens,  
Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker,  
Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas,  
Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke,  
Yuan Yu, and Xiaoqiang Zheng.

TensorFlow: Large-scale machine learning on heterogeneous systems,  
2015. Software is available from [tensorflow.org](https://www.tensorflow.org).