NLP Text Summarization Report

Omer Zilcer 318247350

Yehuda Shargal 318621968

Shahar Levi 318664018

Omer Dahary 315752154

Contents

1. Reading Material Summary

1.1. Extractive vs. Abstractive

1.1.1. Extractive Summarization

Extractive summarization is a method for generating a summary of a body of text by selecting and combining the most significant sentences or phrases from the original text. It aims to retain the most important information from the text and present it in a concise and meaningful way. It is a relatively straightforward and simple method that can be used to quickly generate summaries of large amounts of text.

The extractive method is considered easier due to its straightforward approach of selecting and combining rather than creating whole new sentences. The resulting summary is generally more accurate, as it is based on the actual content of the text and less prone to errors. It also retains the original style and tone of the text, making it easier to understand and retain the most important information.

However, this method also has some limitations. One of the main disadvantages is that it is limited by the content of the original text and may not provide a complete or comprehensive summary of the text. Also, the resulting summary may be repetitive or redundant, especially if the original text contains multiple sentences that convey the same information. This can make the summary less engaging and harder to understand.

1.1.2. Abstractive Summarization

Abstractive summarization is a method of generating a summary of a body of text by creating new sentences that capture the meaning of the original text. The method requires an understanding of the content of the text in order to generate a new condensed version that captures the most important information.

Abstractive summarization offers greater flexibility in generating a summary, because it creates sentences that capture the main ideas of the original text. This allows for a more comprehensive representation of the text, rather than being limited by the exact sentences that are in the original text as extractive summarization does.

The process of abstractive summarization is also more complex and requires way more computational power. It requires a deeper understanding of the content of the text, and the generation of new sentences, which increases the likelihood of errors and inaccuracies in the summary. It is incrementally harder to teach the computer how to understand the actual meaning of text and teach it how to choose which parts of it are important. And because of this the generated sentences may not accurately reflect the content of the original text, leading to potential misunderstandings or misrepresentations of the information.

1.2. Evaluation Methods

1.2.1. Introduction

When building machine learning models, we need some form of metric to measure the goodness of the model. The goodness of the model could have multiple interpretations, but generally when speaking of it in a Machine Learning context we are talking of the measure of a model's performance on new instances that weren't a part of the training data. But in the scope of our project and in the subject of NLP, we would be comparing the given human-made text summaries with the models output text summaries of the same paragraphs, instead of new models.

Evaluating summaries typically involves having human judges assess various quality metrics such as coherence, conciseness, grammaticality, readability, and content. However, conducting a thorough manual evaluation of summaries over a number of linguistic and content-related questions on a large scale would be a huge task requiring thousands of hours of human effort. This is costly and impractical to do on a regular basis.

1.2.2. ROUGE

Rouge (Recall-Oriented Understudy for Gisting Evaluation) is an automated evaluation method for text summarization which gives a score from 0 to 1 for each candidate (machine created summary from a model's output) by comparing it to a reference, human-made, summary of a text, based on the overlapping co-occurrences of word or words between them.

Different ROUGE methods use different ways of words comparison between the reference and the candidate: overlapping n-grams, word sequences, or words between the two. Almost all ROUGE methods calculate precision and recall by their own metric, and only then determine the F-measure, which is the harmonic mean of the precious two.

1.2.2.1. ROUGE-N

This method compares the overlapping occurrences of n-grams. n-gram is a continuous sequence of exactly n words taken from a given text. N is a parameter. To make the assessment, first, we need to extract all the n-grams from both the reference summary and the candidate summary. Then we need to count the sum of occurrences of overlapping n-grams between the two. Eventually we compute the score of the assessment as:

$$\frac{\sum_{S\in\{ReferemceSummaries\}}\ \sum_{gram_n\in S} Count_{match}(gram_n)}{\sum_{S\in\{ReferenceSummaries\}}\ \sum_{gram_n\in S} Count(gram_n)}$$

In our case, we only have one reference summary. So we divide the number of overlapping n-grams of the two summaries by the number of n-grams in the reference summary. This assessment is also called recall-oriented because we make the comparison against the reference summary (opposite to the precision-oriented

approach which compares the relevant measured items over the generated summary occurrences).

1.2.2.2. ROUGE-L

ROUGE-L takes what is called LCS (Longest common Subsequence) which is the longest common sequence of words which show in the same order in both candidate and reference but aren't necessarily appearing adjacent one after the other (denoted by $LCS(X,Y)$).

Then divide it by the number of words in the reference summary to get the recall LCS (R_{lcs}). Or divide it by the length of the candidate to get the precision LCS (P_{lcs})

$$R_{lcs} = \frac{LCS(X,Y)}{m}$$
$$P_{lcs} = \frac{LCS(X,Y)}{n}$$

where m, n are the lengths of X, Y.

There are two kinds of LCS evaluators:

Sentence-Level LCS: Calculating LCS between two pieces of text as if each of them is one sentence.

Summary-Level LCS: Calculating LCS of multiple sentences of text. First you calculate the LCS of each pair of sentences between the candidate and the reference. Then sum the unions of those LCS pairs.

1.2.2.3. ROUGE-S

Rouge-S is a method that measures co-occurrence of Skip-Bigrams, which is any pair of words in their sentence order, allowing for arbitrary gaps in between them. This allows for matching of non-contiguous word sequences, which is useful when the order of words in the summary is not important. The recall and precision of this method are as:

$$R_{skip2} = \frac{SKIP2(X,Y)}{C(m,2)}$$
$$P_{skip2} = \frac{SKIP2(X,Y)}{C(n,2)}$$

Where $SKIP2(X,Y)$ is the number of co-occurring skip-bigrams in X and Y (summary/candidate and reference). And C(n,2) is the number of bi-grams with the sentence length of n.

1.2.2.4. ROUGE-SU

ROUGE-SU extends ROUGE-S by also including unigrams (single words) in the matching process. This means that the ROUGE-SU score considers not only the skip-bigrams that appear in both the candidate and reference summaries, but also the unigrams that appear in both. This helps to account for cases where important words may be skipped in the summary, but still appear elsewhere in it.

1.2.3. Pyramid

Automatic summarization has traditionally been viewed as a sentence selection problem, and systems were evaluated using metrics such as precision and recall. Just like we discussed about ROUGE. However, this approach had several undesirable aspects that were revealed over time. For example, using a single human model for comparison was not recommended due to human variation, and multiple human gold standards would provide better ground for comparison. When asked to write a summary of a text, people do not produce an extract of sentences from the original text. Therefore, the exact match of system sentences with human model sentences is not possible. Additionally, it's important to consider how sentences are similar in meaning and how much detail should be analyzed.

The Pyramid Method has been proposed as a unified framework for addressing these issues. It uses a manual procedure to identify important information in a text, called Summary Content Units (SCUs), which are semantically motivated, substantial units that emerge from annotation of a collection of human summaries. SCUs are similar in spirit to the automatically identified elementary discourse units, the manually marked information nuggets and factoids. A pyramid represents the opinions of multiple human summary writers each of whom has written a model summary for the input set of documents.

1.3. Abstractive summarization by Transformers

Abstractive summarization, as explained before, is the task of generating a concise and informative summary of a longer document, while also including information that may not be explicitly written in the original text, but retain the original meaning. Transformers are a type of neural network architecture that has been widely used to achieve abstractive summarization.

To perform abstractive summarization with transformers, the model is usually trained on a large corpus of text, using a technique known as supervised learning. During training, the model is presented with pairs of input-output examples, where the input is a longer text and the output is a summary of that text.

The transformer model learns to encode the input document into a series of vectors, known as the document's "contextualized" representation. It then uses this representation to generate the summary, one word at a time. At each step, the model predicts the most likely next word, based on the context it has already encoded and the output it has generated so far. This process is repeated until the model generates an entire summary.

One advantage of using transformers is their ability to capture long-range abstract dependencies and contextual information, which is essential for generating high-quality summaries. However, because transformers are a type of neural network, they require a large amount of training data and proportionately large computational resources.

2. The Project Assignment

2.1. Research Question / Introduction

The task we had to accomplish is training a Natural Language Processing transformer to create a model for text summarization using the texts of The Rabbi Kook and evaluate the results of the model we trained using different evaluation methods.
The transformer we used are MT5-small, MT-5base and mBART-50 and the evaluation method we used is Rouge.

2.2. Text Summarization Transformer to Model

2.2.1. MT5

The T5 model can be utilized for solving NLP problems by converting the problem into a text-to-text format. In order to train the model, both input and target sequences are required. One variant of the T5 model is the mT5, which is a multilingual model. It has been pre-trained on the mC4 dataset, which consists of data from various domains and encompasses 101 languages, including Hebrew. The Hebrew language accounts for 1.06% of the mC4 dataset.

Transformer models like T5 and mT5 are trained to perform a specific type of task called conditional text generation, where the model is trained to generate a new output sequence that is conditioned on the input. In the case of summarization, the input sequence might be a longer piece of text, and the output sequence is a shorter summary of that text. The model learns to identify the most important information in the input text and generate a summary that captures this information in a concise way.

Transformers work by using self-attention mechanisms to weigh the importance of different words in a sentence or sequence, allowing the model to understand the context and relationships between them. During training, the transformer model is fed large amounts of text data and learns to predict the correct output for a given input by adjusting the weights of its neurons through backpropagation. This process involves minimizing the difference between the model's predicted output and the correct output, using a loss function. To achieve this, the model is trained using a supervised learning approach, where it is given pairs of long text passages and their corresponding summaries as training examples.

At inference time, the trained transformer model can generate summaries for new, unseen text inputs. This is done by tokenizing the input text into a sequence of words or subwords, which is then fed into the model. The model uses its learned weights to generate a summary output that captures the most important

information in the input text. This output can then be further processed, such as by removing any unnecessary words or correcting grammar, to produce a final summary.

2.2.2. mBART-50

mBART-50 is a multilingual machine translation model that supports 50 languages. It is created by fine-tuning a pre-trained model on multiple languages simultaneously. The model is trained using a denoising pretraining objective where the source documents are noised using shuffling and in-filling schemes, and the model is tasked to reconstruct the original text. . The mBART-50 been pre-trained on the ML50 dataset that contains monolingual documents in multiple languages. There are 204,380 distinct Hebrew sentences in the ML50. The language id symbol is used as the initial token to predict the sentence.

At the training stage, a special language id ('he' for Hebrew) token is used as a parameter for both sources and targets languages.

During inference, the transformer model that has been trained has the ability to create summaries for text inputs that have not been seen before. To achieve this, the input text is first tokenized into a sequence of words or subwords, which is then provided as input to the model.

2.2.3. Differences between MT5 and mBART50

Some of the differences between MT5 and mBART50 include:

- MT5 is trained on the mC4 corpus which is a large dataset which includes 101 languages, while mBART50 was trained on Common Crawl and Wikipedia data in less number on languages.
- Although mBART50 was trained on a smaller number of languages it is found to be more effective in translating in some cases.
- mBART50 is said to be trained on 204,380 distinct Hebrew sentences, while the Hebrew is 1.06% of the MT5 dataset (mC4), which of itself consists of 6.6B pages. Those different types of data are hard to compare directly without further public knowledge of the models.

2.3. Model Training

2.3.1. Creating the dataset

In order to train the model, we had to create a dataset that would contain paragraphs from Rabbi Kook's writings, with the corresponding summary attached to each paragraph. We extracted the paragraphs from files א-ח, and from books of "אורות" for which there are summaries in the file "שביבי אורות".

First, we used a python script using the docx library that takes a docx file and extracts the paragraphs from it, so that each paragraph is in a separate txt file. We repeated this for the paragraphs and for the summaries.

During the extraction, we noticed that some of the paragraphs contained more than one paragraph and therefore a situation of inconsistency between the number of paragraph files and summary files occurred. Therefore, we merged the multiple paragraphs to one paragraph for consistency. Similarly, in some files, two paragraphs were merged into one, unlike the summary where there were separate paragraphs. So, we decided to cut them into two different paragraphs. Additionally, we had to delete subheadings and spaces so they wouldn't be mistaken for paragraphs.

Afterwards, when we had files for each paragraph and corresponding files for its summary, we used a python script to merge them into a single json file.

We decided to "clean" the dataset as best we could. we deleted the paragraph number in each paragraph and in its summary, deleted unnecessary tabs that originally appeared in the summary file, and deleted unnecessary comments in the summaries such as the appearance of "[המשך]" in some of the summaries.

When we uploaded our dataset to the Hugging Face website, we had about 3500 paragraphs and summaries ready to train the model.

link to the dataset:

https://huggingface.co/datasets/thaomer/ravkuk_summerize_dataset

2.3.2. Training the model

During the project, we tried experimenting with several different models, but at first, we chose to experiment with the mt5-small model, because our computer only had 8GB of GPU memory, and it was the only model that was small enough so it didn't cause out of memory errors.

In this model we also tried to reduce and utilize the amount of memory, so we used gradient accumulation optimization, which is a method used to increase the size of the batch used for training, without the need to use a large amount of GPU memory. Typically, during model training, its parameters are updated based on the average gradient of all samples in the batch. In contrast, in the gradient accumulation method, the gradient of a large number of mini batches is accumulated and averaged before the parameters are updated. For example, when we use a batch size that is too large to be loaded into the gpu memory, we divide the batch into several smaller mini batches of a size that can be loaded into the memory, run each of them on the model, and only at the end do they all update the model weights.

We also tried to reduce the amount of memory, by setting the hyperparameter fp16=true. Fp16 signifies that the model's weights are represented by 16-bit floating point numbers, instead of the default 32-bit. However, this change resulted in divergence in the first epoch.

When this model also encountered out of memory issues due to database changes, we tried to check other infrastructure options. We looked at AWS because one of us has a background in their systems after using them in a distributed systems programming course. In his student account it was not allowed to use GPU(limit=0), and even after he opened a personal account and had a chat correspondence with a representative from AWS it was not possible.
Therefore, we chose to use google colab.
In google colab there are two options for using GPU's, the standard one that allocates 15GB of GPU memory which we used to train the smaller mt5-small model and the premium option that allocates 40GB of GPU memory and which we then used to train the larger models.

The mt5-small model consumed about 13GB of GPU memory, in contrast, the mt5-base and mbart-50 models required around 30GB of GPU memory and this varied from run to run.

In terms of run times, the runtime of each training ranged from 30 minutes for certain runs on the mt5-small to about an hour and a half for certain runs on the larger mt5-base and mbart-50 models.

For computational and economic reasons, we chose to focus initially on training the mt5-small model. We performed various optimizations that contributed to improving the accuracy of the model, but starting at a certain stage the improvement was minimal.

We started training the mt5-small model with the standard values we saw used in running examples in Hugging Face:

- learning_rate=5e-4
- batch_size=4
- validation size=20%
- gradient_accumulation=2
- weight_decay=0.01
- optimizer=AdamW
- lr_scheduler=linear

Learning rate is a hyperparameter that determines how fast the model learns from the training data. In each iteration of the training process, the model weights are updated according to the gradient of the loss function according to the weights times the step size. If the step size is too large, the optimization algorithm may miss the optimal weights and diverge, thus resulting in poor accuracy. Conversely, a step size that is too small makes the optimization process long and may also cause the optimization to get stuck at a local minimum rather than a global one.

Batch size is a hyperparameter that refers to the number of samples that the model processes in each iteration of the training phase. In the training phase, the data is divided into batches, and the model updates its parameters based on the average gradient for each batch. The batch size is an important hyperparameter that can affect the performance of the model in the training phase. A large size can lead to accurate gradient estimation and faster convergence, but it also requires more memory and computational capabilities. On the other hand, a small size can lead to slower convergence but allows for more frequent updates and can also contribute to generalization.

Weight decay is a hyperparameter that helps to prevent overfitting in models by adding a penalty term to the loss function. This penalty term encourages the model to have smaller weights, which can reduce the complexity of the model and prevent it from fitting the noise in the training data.

2.3.3. Model results and optimization

2.3.3.1 experimenting with learning rate and batch size

At first, we tried to experiment only with the learning rate and batch size, and after setting them with fitting values we tried to experiment with the other hyperparameters:

2.3.3.1.1 initial values of learning rate and batch size

For the standard values we got the following results:

model=mt5-small BS=4 lr=5e-4:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	3.681	2.92293	0.1113	0.0334	0.1056	0.1054	18.9701
1	3.2478	2.76506	0.123	0.0438	0.117	0.117	18.9787
2	2.9972	2.65029	0.1462	0.0539	0.1395	0.1391	18.973

Training loss represents the difference of the representative vectors of the real paragraph summaries versus what the model predicted, for the training data.

validation loss represents the same for the validation data that did not participate during training.

The different rouge scores indicate the similarity between the real summaries and the summaries produced by the model, in terms of the evaluation criteria from the corresponding rouge metric. As the number is lower and closer to 0, then the similarity is smaller.

gen len represents the average length of the summary length predicted by the model.

2.3.3.1.2 experimenting with different learning rates

We tried increasing the step size to 5e-3, and as a result, we got worse results:

model=mt5-small BS=4 lr=5e-3:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	5.6462	5.415173	0.0052	0	0.0052	0.0052	19
1	5.3325	5.210734	0.0095	0	0.0094	0.0094	19
2	4.8188	4.717731	0.0077	0	0.0078	0.0078	19
3	4.4205	4.467945	0.0136	0	0.0136	0.0136	19

We tried more values for the learning rate in order to find the optimal value:

model=mt5-small BS=4 lr=5e-5:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	4.5096	3.340521	0.0799	0.018	0.0745	0.0744	18.4253
1	3.9632	3.10924	0.1105	0.0311	0.1028	0.1025	18.8165
2	3.789	3.016706	0.1148	0.0331	0.1072	0.1065	18.8521
3	3.7107	2.954438	0.1172	0.0336	0.109	0.1085	18.9004

model=mt5-small BS=4 lr=1e-3:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	3.3857	2.915842	0.1173	0.0345	0.1097	0.11	18.9787
1	3.0639	2.772439	0.1445	0.053	0.1388	0.1386	18.9815
2	2.7103	2.665697	0.1346	0.0492	0.13	0.1302	18.9801
3	2.5274	2.604588	0.1468	0.0548	0.1409	0.1406	18.9801

2.3.3.1.3 experimenting with batch size

In the end, we found a good combination of the two hyperparameters mentioned above when we decided to multiply the batch size by 2. We read about a heuristic claiming that the two hyperparameters above are correlated, and if one doubles the batch size by 2, then he must multiply the learning rate by the root of 2, so we multiplied it as well. We thought that by increasing the batch size we would be able to improve the runtime performance and might improve the training results.

model=mt5-small BS=8 lr=1e-3*sqrt(2):

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	3.722	3.08175	0.1136	0.0321	0.1049	0.1049	18.9744
2	3.0545	2.845023	0.1301	0.0445	0.1233	0.1234	18.9744
3	2.8413	2.707774	0.1343	0.0469	0.1278	0.128	18.99
4	2.5248	2.669475	0.1419	0.0494	0.1341	0.1342	18.973
5	2.4187	2.627167	0.148	0.0561	0.1405	0.1405	18.9872
6	2.1362	2.594006	0.1516	0.0573	0.1441	0.144	18.9858
7	2.0174	2.588678	0.1479	0.0543	0.1422	0.1421	18.9872
8	1.8484	2.632994	0.1494	0.0554	0.1425	0.1425	18.9772

We tried to reduce the batch-size to 2 and we also changed the learning rate accordingly.

model=mt5-small BS=2 lr=1e-3/sqrt(2):

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	3.3483	2.96643	0.1109	0.0262	0.1023	0.1022	18.9716
1	2.9805	2.799987	0.1364	0.0442	0.1291	0.1288	18.9986
2	2.7694	2.697395	0.1406	0.0472	0.1329	0.1331	18.9872
3	2.5415	2.647351	0.1457	0.0503	0.1385	0.1384	18.9844

We got roughly the same results, but for batch size of 8 the runtime was significantly shorter, so we decided to stop the run and stick with lr=1e-3*sqrt(2) and BS=8.

2.3.3.2 experimenting with weight decay

After determining the value of the two hyperparameters, we started experimenting with changing other hyperparameters. One of these values is weight decay. The standard value of this value is 0.01 and we decided to try setting it to 0.

model=mt5-small lr=1e-3*sqrt(2) BS=8 weight_decay=0:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	3.5366	2.99068	0.1135	0.0266	0.1057	0.1057	18.8378
2	3.2096	2.810269	0.1218	0.0395	0.1161	0.1158	18.9573
3	2.7978	2.724437	0.1376	0.0478	0.1308	0.1308	18.9772
4	2.547	2.660583	0.134	0.0462	0.1257	0.1258	18.9744
5	2.4047	2.618089	0.1391	0.0524	0.1337	0.134	18.9758
6	2.2401	2.61175	0.1467	0.0527	0.1415	0.1414	18.9758
7	2.0772	2.611927	0.147	0.0548	0.1408	0.1406	18.9772

There was no improvement in the model results, so we decided to return it to the standard value of 0.01.

2.3.3.3 experimenting with warmup ratio

Next, we tried to experiment with the warmup ratio hyperparameter. This is a hyperparameter that controls the rate at which the learning rate increases in the initial phase of training until reaching the initial learning rate value.

Since we took a model checkpoint that was trained on a different dataset and we are now running retraining with a different dataset, we thought that by not changing the weights aggressively right in the beginning, the model will become accustomed to the new dataset and will learn better.

we got the following results which are the best results so far.
model=mt5-small lr=1e-3*sqrt(2) BS=8 warmup_ratio=first_epoch:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	3.797	3.003924	0.1107	0.0311	0.1036	0.1038	18.9801
2	3.1263	2.835973	0.1318	0.0487	0.1254	0.1256	18.9872
3	2.9149	2.695484	0.1322	0.0453	0.1252	0.1253	18.9772
4	2.6009	2.626936	0.139	0.0485	0.133	0.1336	18.9701
5	2.3127	2.583289	0.1454	0.0541	0.1388	0.1393	18.9915
6	2.1352	2.558371	0.1484	0.0577	0.1413	0.1418	18.9815
7	1.9894	2.578928	0.1571	0.0613	0.1507	0.151	18.9886
8	1.7983	2.558214	0.1537	0.0608	0.1483	0.1488	18.9886
9	1.6909	2.616386	0.156	0.0623	0.1497	0.1504	18.9801

2.3.3.4 experimenting with gradient accumulation

We tried to see if different gradient accumulation values impact accuracy.

model=mt5-small lr=1e-3*sqrt(2) BS=8 warmup_ratio=first_epoch gradient_accumulation=1:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	6.2594	3.145413	0.0986	0.0206	0.0922	0.0917	18.9488
2	3.3151	2.886459	0.1237	0.0402	0.1173	0.117	18.9189
3	2.8591	2.769665	0.1356	0.0465	0.131	0.1307	18.9772
4	2.5488	2.703102	0.1426	0.0531	0.1372	0.1368	18.9815
5	2.2762	2.659359	0.1438	0.0524	0.138	0.1382	18.9758
6	2.0393	2.648589	0.1432	0.0544	0.1382	0.1382	18.9744
7	1.8233	2.666296	0.1481	0.0583	0.1434	0.1435	18.9744

model=mt5-small lr=1e-3*sqrt(2) BS=8 warmup_ratio=first_epoch gradient_accumulation=4:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	7.5253	3.117632	0.1035	0.0292	0.0955	0.0959	18.9616
2	3.4148	2.876252	0.1208	0.0355	0.1138	0.1137	18.9758
3	2.9963	2.746762	0.1259	0.0482	0.1226	0.1228	18.9744
4	2.7171	2.677763	0.1354	0.0512	0.129	0.1292	18.963
5	2.5022	2.636155	0.1471	0.0568	0.1414	0.1413	18.9687
6	2.3161	2.619492	0.1433	0.0567	0.1386	0.1389	18.9701
7	2.1372	2.60457	0.1494	0.0589	0.1433	0.1433	18.9801

model=mt5-small lr=1e-3*sqrt(2) BS=8 warmup_ratio=first_epoch gradient_accumulation=8:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	8.62	3.195321	0.0933	0.0209	0.0862	0.0865	18.9659
2	3.5606	2.900435	0.106	0.0245	0.0979	0.0979	18.9844
3	3.1149	2.779139	0.1172	0.031	0.1098	0.1099	18.9844
4	2.8524	2.707489	0.1377	0.047	0.1304	0.1303	18.99
5	2.6542	2.635717	0.133	0.0433	0.1263	0.1262	18.9915
6	2.4969	2.616893	0.141	0.0471	0.134	0.1341	18.9929
7	2.404	2.608723	0.1432	0.0503	0.1359	0.1354	18.9972

The difference between 1 and 2 was not particularly significant, but when we entered 4 and 8 the accuracy of the model decreased, so we chose to stay with gradient accumulation=2.

2.3.3.5 changing the validation data ratio

Because we thought that the number of examples in the training data was too small and might have prevented the model from learning, we decided to increase the training data size at the expense of the validation data size. We reduced the validation size from 20% to 10%, and got the following results:

model=mt5-small lr=1e-3*sqrt(2) BS=8 warmup_ratio=first_epoch training validation=0.1:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	6.3333	3.074067	0.1038	0.0279	0.099	0.0986	18.8153
1	3.2841	2.814539	0.1302	0.0435	0.1249	0.1246	18.9517
2	2.8629	2.708767	0.1324	0.0434	0.127	0.1264	18.9574
3	2.5655	2.584338	0.1437	0.0514	0.1376	0.1374	18.9631
4	2.3279	2.560894	0.1464	0.0508	0.1399	0.1394	18.9631
5	2.1346	2.53717	0.1476	0.0544	0.1422	0.1416	18.9602
6	1.9774	2.531805	0.1507	0.0545	0.1451	0.1445	18.9574

We went back to experimenting with the warmup ratio and changed it to 30 percent of all the steps and got:

model=mt5-small lr=1e-3*sqrt(2) BS=8 warmup_ratio=0.3 training validation=0.1:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	7.5743	3.044461	0.1157	0.0351	0.1101	0.1098	18.9574
1	3.4548	2.875929	0.1169	0.0342	0.1104	0.1101	18.9801
2	3.0904	2.763636	0.1346	0.0448	0.1283	0.1278	18.9801
3	2.7423	2.655377	0.1495	0.058	0.1447	0.1445	18.9801
4	2.4527	2.57652	0.1586	0.0576	0.1512	0.1504	18.9688
5	2.2118	2.554793	0.1605	0.062	0.1524	0.1523	18.9773
6	2.0126	2.545896	0.1571	0.0605	0.1511	0.1508	18.9773
7	1.8677	2.559095	0.1683	0.0666	0.1607	0.1602	18.9773

This is the best result we got from all the different runs.

2.3.3.6 changing the lr scheduler

Finally, we tried to change the lr scheduler to cosine instead of linear. The lr scheduler is responsible for changing the learning rate according to the advancement of the training progress.

model=mt5-small lr=1e-3 * sqrt(2) BS=8 warmup_ratio=0.3 training validation=0.1 lr_schedule=cosine:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	7.2002	3.006621	0.1161	0.0273	0.1077	0.1076	18.9233
1	3.4528	2.823484	0.1279	0.0327	0.1217	0.1215	18.9688
2	3.0918	2.71368	0.1278	0.0392	0.1219	0.1218	18.9631
3	2.7613	2.604368	0.1381	0.0438	0.1326	0.132	18.9744
4	2.4509	2.539791	0.14	0.0464	0.1347	0.1342	18.9688
5	2.1858	2.489903	0.1475	0.0503	0.141	0.1407	18.9744
6	1.97	2.475735	0.1466	0.0514	0.1392	0.1392	18.9744
7	1.8701	2.491151	0.1489	0.0524	0.1404	0.1402	18.9744

We also tried to combine this together with changing the warmup ratio:
model=mt5-small lr=1e-3 * sqrt(2) BS=8 warmup_ratio=0.1 training validation=0.1 reschedule=cosine:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	6.286	3.067253	0.1105	0.0301	0.1025	0.1027	18.9091
1	3.29	2.823812	0.1309	0.0472	0.123	0.123	18.8949
2	2.8709	2.725399	0.1461	0.0585	0.1377	0.1377	18.9432
3	2.5576	2.659745	0.1417	0.0531	0.1348	0.1347	18.9432
4	2.3058	2.595429	0.1529	0.0641	0.1466	0.1465	18.9517
5	2.0827	2.594278	0.1515	0.0623	0.1453	0.1449	18.9403
6	1.8869	2.57992	0.1547	0.0642	0.1476	0.1477	18.9233
7	1.7458	2.615192	0.1589	0.0653	0.1524	0.1519	18.9233

It didn't improve the accuracy, and still for the values:
lr=1e-3 * sqrt(2) BS=8 warmup_ratio=0.3 training validation=0.1
We got the best results.

2.3.4 conclusions regarding training the mt5-small model

1) The most influential hyperparameter on accuracy was the learning rate.
The other hyperparameters also have an effect, but in a relatively negligible manner.

2) The batch size did not impact accuracy, so we used larger batch size for faster learning time.

3) It is possible to continue optimizing and improving the performance of the mt5-small model, but the results show the improvement will likely be minimal.

4) Apparently, the small size of the dataset affected the accuracy of the model, it could have been improved given a larger number of examples.

5) We saw that approximately after 7 epochs the model usually converged so running more epochs did not improve overall results.

2.3.5 training the mt5-base model

After training the mt5-small model, we wanted to try training more complex models with a larger number of parameters and see their accuracy.
We trained the mt5-base model and the mbart-50 model.

For mt5-base, we used the most optimal configuration we found for mt5-small, and got the following results:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	5.899	2.879679	0.1223	0.0431	0.1168	0.1166	18.8523
1	3.263	3.083246	0.0745	0.0174	0.071	0.0704	18.9659
2	3.217	2.696372	0.124	0.0409	0.1179	0.1178	19
3	2.7902	2.637841	0.1281	0.0469	0.1237	0.1236	18.9659
4	2.371	2.468977	0.1497	0.0547	0.1436	0.1432	18.9886
5	1.8459	2.382818	0.1619	0.0589	0.1549	0.1545	18.9801
6	1.3731	2.443076	0.1563	0.0618	0.1493	0.1491	18.9858
7	1.0625	2.540544	0.1587	0.0667	0.1542	0.1539	18.9858

We tried experimenting with the hyperparameters configuration, but we did not achieve major improvements over mt5-small.

Due to high economic and computational cost, we chose to focus on the mbart50 model because it is an entirely different model from mt5.

2.3.6 training the mbart50 model

2.3.6.1 training the mbart50 with initial hyperparameters values

We tried the following arguments lr=1e-3*sqrt(2) BS=8 warmup_ratio=0.3 training validation=0.1 And we got the following results:

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	6.8179	5.598524	0.0011	0	0.0011	0.0011	15.3807
1	5.1468	4.732496	0.0465	0.001	0.046	0.0461	27.3949
2	4.5573	4.658832	0.0307	0.0002	0.0299	0.03	69.4631
3	4.1419	4.25315	0.0385	0.0006	0.0364	0.0366	45.5568
4	3.7458	4.19133	0.0178	0	0.0173	0.0174	26.8835
5	3.3105	4.204408	0.0452	0.0004	0.0418	0.0417	35.517
6	2.8078	4.359331	0.009	0	0.0088	0.0088	30.4261

The results were quite bad and even worse than the results of the previous models.

2.3.6.2 experimenting with learning rate

We tried experimenting with the warmup and batch size:

learning_rate=5.6e-4 BS=8 epochs=5 no warmup validation=0.1

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	6.5792	5.230215	0.0032	0	0.0032	0.0031	16
1	5.0224	4.784769	0.0078	0	0.0078	0.0077	33.4261
2	4.5835	4.515709	0.0317	0	0.0311	0.0311	37.858

learning_rate=5.6e-5 BS=8 epochs=5 no warmup validation=0.1

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	3.0906	2.408861	0.1739	0.0605	0.1636	0.1636	35.3409
1	1.869	2.32393	0.1683	0.0625	0.1586	0.1589	35.5767
2	1.3025	2.337524	0.1881	0.0768	0.1754	0.1758	36.2642
3	0.9093	2.423138	0.1948	0.0852	0.182	0.1825	36.0369
4	0.7752	2.583419	0.2049	0.0952	0.1924	0.1932	35.5312

The results started improving but they were not relatively better than the other models.

2.3.6.3 mbart50 best hyperparameter values

After similar several attempts and experimenting with the hyperparameters, we got significantly better results than the other models.

learning_rate=5.6e-5, weight_decay=0.01, gradient accumulation =2, batch_size = 8, num_train_epochs = 10, test_size=0.1

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	3.1169	2.360387	0.1878	0.0725	0.1737	0.1737	33.7784
1	1.8945	2.252172	0.1897	0.0765	0.1776	0.1776	34.2074
2	1.3083	2.288599	0.2001	0.0927	0.1895	0.1892	35.4432
3	0.8693	2.372662	0.2243	0.1123	0.2122	0.2117	31.4943
4	0.5507	2.505911	0.2577	0.1527	0.2463	0.2466	34.3693
5	0.3385	2.603164	0.2703	0.1672	0.2593	0.2584	33.5994
6	0.2031	2.651797	0.2912	0.1932	0.2812	0.281	34.1676
7	0.1272	2.704045	0.2891	0.1895	0.2799	0.2796	34.6761
8	0.0842	2.751488	0.2978	0.198	0.2888	0.2887	34.1932
9	0.0605	2.776234	0.2928	0.1926	0.2815	0.2816	34.5028

2.3.7 summary of the training process

In this project we were assigned training a model for text summarization. We chose to train 3 different models and compare their performance:

1) Mt5-small
2) Mt5-base
3) Mbart-50

In the beginning of the training process, we chose to experiment more with the Mt5-small model due to memory and runtime issues. We started training it with standard arguments found in the Hugging Face site. Afterwards, we mainly tried to get the optimal values of the learning rate hyperparameter and corresponding batch size. Next, we tried experimenting with other hyperparameters such as weight decay, gradient accumulation and warmup ratio. We noticed that the other hyperparameters did not contribute much to the model performance.

After getting the best values for the Mt5-small model, We tried them on the Mt5-base model and adjusting them to the new model, but did not notice major improvement in the model performance.

We decided to try training the Mbart50 model, because we thought its performance might be better than the Mt5-models.

In the first runs the model accuracy was worse than the previous models, but after some trials of different hyperparameter values, it performed much better than the previous Mt5 models, as can be seen later in the models' evaluation section.

2.4. Using the models online

The three models are available to use in hugging face online:

MT5-small: https://huggingface.co/thaomer/le-fine-tune-mt5-small
MT5-base: https://huggingface.co/thaomer/le-fine-tune-mt5-base
Mbart: https://huggingface.co/thaomer/le-fine-tune-mbart-large-50

To use them you need to copy your desired paragraph to summarize and click on the "Compute" button to get the summary.

2.5. Evaluating the Results

The method we chose to evaluate our model results is Rouge. In particular, 3 different Rouge variations: Rouge-1, Rouge-2 and Rouge-L. We took some result summaries from the model we trained and compared them with their human-made reference summary, through these Rouge evaluators using Python scripts and compared the findings.

2.5.1. Summary Examples

The following are paragraphs of Rabbi Kook that the models were not trained on, together with the output summaries that our models produced for them, as examples for their performances.

2.5.1.1.

Paragraph	העולם עלה, נתבסם ונתמתק בכללו – זאת היא הנחה כוללת מוסכמת מהחכמה של התולדה, ומקורה ביסוד רזי תורה. בדורות האחרונים התירו וציוו חכמי קודש להודיע סתרי תורה, לבאר בגילוי ובפרסום, מפני שהעולם נתבסם ונמתק ממרירות הדינים שהיו בימים הראשונים, על-כן יש כח, ויש צורך ממילא והכרח, לקבל אור רזי תורה. והצורך הזה כשאינו מתמלא מצדם של ישראל, משתלם הוא ברעם ורעש, בהרס וירידה. מקום כינוס כל התוכן הרעיוני שבעולם, כל המובן הרוחני שלו, שהוא תמצית כל מה שהוא חושב ומרגיש, הוא הרעיון האלהי, מקור הדעת והאמונה, מקור המוסר והחיים בנשמה. הציור האלהי המתפשט בהמון רב, הוא תולדה נאמנה ממצב המדע והמוסר הכללי, האנושיות והאומה. העסקנות בחכמת האלהות הוא מכון כח החיים לכל כשרון ומעשה. את פעולותיו הוא מגלה אפילו במרחקים, אפילו בשדרות שלפי המבט החיצוני אין להם שום יחש אליו. ונברכו בך כל משפחות האדמה, אפילו ספינות הבאות מגליא לאספמיא אינם מתברכות אלא בשביל ישראל."
MT5-small Summary	העולם עלה, אך בזמן שהעולם עלה, הוא מתגלה
MT5-base Summary	רזי התורה נתעלו בישראל בזמן שהעולם נתבסם, ולכן ה
Mbart Summary	רזי תורה הנם תולדת העולם, וגילוים נעשה על-ידי ישראל, על-ידי העסקנות בחכמת האלוהות.

2.5.1.2.

Paragraph	לא רק האגדה לבדה תואר מאור הרעיון הבהיר של תורת ארץ ישראל, כי-אם גם ההלכה, יסודות הסברות, נתוח הפסקים, שרשי השיטות, ומובנם הכללי הנעוץ בעמקי החיים הרוחניים והמעשיים. ולא רק התורה במובנה הלימודי המצוי בד'-אמות של הלכה, כי-אם הארת החיים כולם, הכל תלוי בעושר הרעיון. מעומק התחיה הרוחנית המוכנת לתורת ארץ ישראל, הגבולים וקירות הברזל שבין ענין לענין, בין מקצע למקצע, ילכו ויקטנו. העולם הרוחני כולו יהיה מוסקר באויר חיי נשמות של ארץ חיים בסקירה אחת, הוד החיים וחמדת הרזים, ברק הפלפול והתעודדות התחיה של כנסת ישראל על אדמת הקודש, ברור הלכות והתרחבות החזיון והשירה, חפץ השקידה המצויינת וחשק ההתפתחות הגופנית. וכל אלה וביוצא בהם, שבחוץ-לארץ ובנוהג שבעולם לפני ימי התחיה העומדים אחר כתלנו היו נדונים כרחוקים זה-מזה וסותרים זה-את-זה, הנם עומדים לנו עתה להיות מאוגדים בחוברת של קשר ואגודת אמת, וכל אחת מסייעת את חברתה, להרחבתה והעמקתה, להתפשטותה ושכלולה.
MT5-small Summary	רק האגדה לבדה תואר מאור הרעיון הבהיר של תו
MT5-base Summary	רק בארץ ישראל תואר הרעיון הבהיר של תורת ארץ ישראל
Mbart Summary	תורת ארץ ישראל מאגדת את המקצועות השונים שבתורה ובחיים לאגודה אחת, ובהירותה מאירה אותם

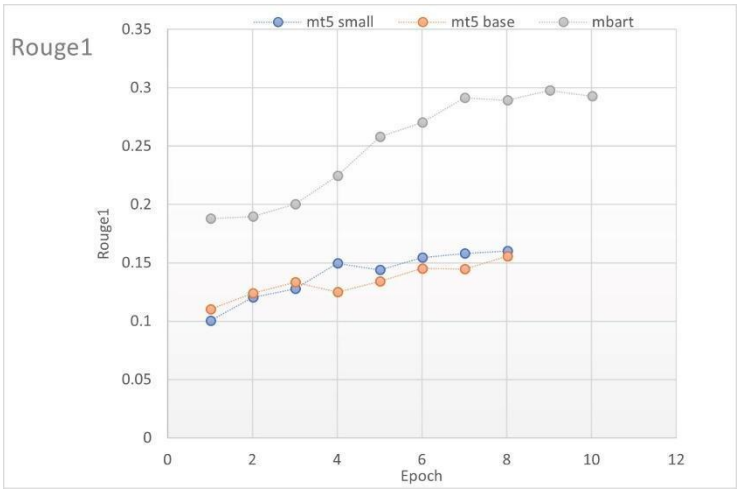
2.5.1.3.

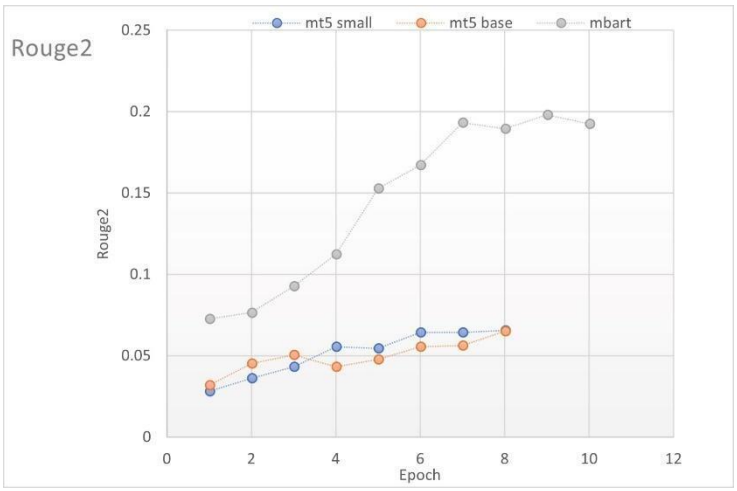
Paragraph	הלימוד האפל שאין בו כל רוח חיים, מביא להטעות את הבריות, ולחשוב שאין שום יחש בין האורה העליונה של הדעות הצחות והמדעים האלהיים והעיונים המוסריים, הכבירים והרחבים, וכל ההגיונות הרוחניות, אל השכל המעשי שבהלכות וארחות החיים. ועוד יש שמביא לחשוב שאלה העיונים הנם סותרים אלה לאלה. אבל כל-זה בא מחסרון הרגל ונסיון, מתוך העזיבה של ההגיון הקדוש שהוא כל-כך נדרש לנו כאויר לנשימה ממש, ומתוך כך נעשים הענינים זרים והרעיונות בלתי מתאימות. אבל עם ההרגל הטוב, תשוב האורה הכללית האלהית להראות את פעולתה על כל, גם על הפרטים המעשים היותר מדוייקים וקצובים. "ונותנת לו מלכות וממשלה וחיקור דין".
MT5-small Summary	הלימוד האפל מביא להטעות את הבריות, אך גם בהרגל
MT5-base Summary	הלימוד האפל מפתח את טהרת הנפש, אך בהרגל הרגל
Mbart Summary	הלימוד האפל מדכא את הרוח, ומתוך כך גם המחשבות והדעות זרות, ותפקיד הנסיון להראות את פעולתה על הכל.

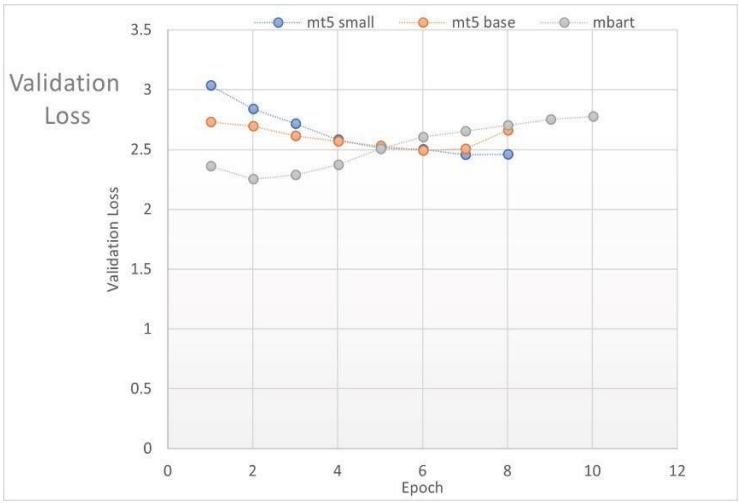
As we can see in the examples above, MT5-small produced the shortest and less coherent summary.
MT5-base was still not very incoherent in Hebrew but managed to produce a sentence from the body of the original paragraph.
Mbart was the most successful at constructing sentences in Hebrew that make sense, and more able to somewhat summarize ideas that it extracted from the original paragraph in a different way.

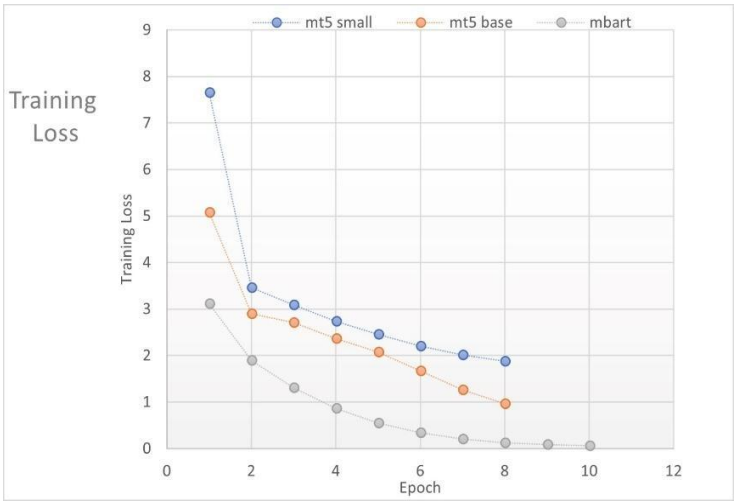
2.5.2. Models evaluation comparison

The following are charts comparing the final form of the 3 different models by different parameters presented in their final training.









3. References

- https://aclanthology.org/W04-1013.pdf
- https://repository.upenn.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1771&context=cis_papers
- https://huggingface.co/docs/transformers/model_doc/mt5
- https://huggingface.co/docs/transformers/model_doc/mbart