ADLHW2 Report
Student ID: R12922184 Name:鄭星逸

Q1: Model (2%)

Model (1%)
Describe the model architecture and how it works on text summarization.

Answer:
The google/mt5-small model uses the classic transformer-based encoder-decoder architecture. The mt5 uses a text-to-text framework for all tasks, treating both the input and output as text.

For summarization, the input is a long piece of text, and the output is a shorter, summarized version.The mT5-small summarizes text by tokenizing it into subwords, encoding the context and semantics, and then using the decoder to generate a concise summary. The decoder focuses on key information through attention mechanisms

Preprocessing (1%)
Describe your preprocessing (e.g. tokenization, data cleaning and etc.)

Answer:

I use the "MT5Tokenizer" to tokenize both the input text and the target summary. Then, I apply the map() function to process the dataset. After tokenization, the dataset is converted into PyTorch tensor format to ensure compatibility with model training.

Q2: Training (2%)

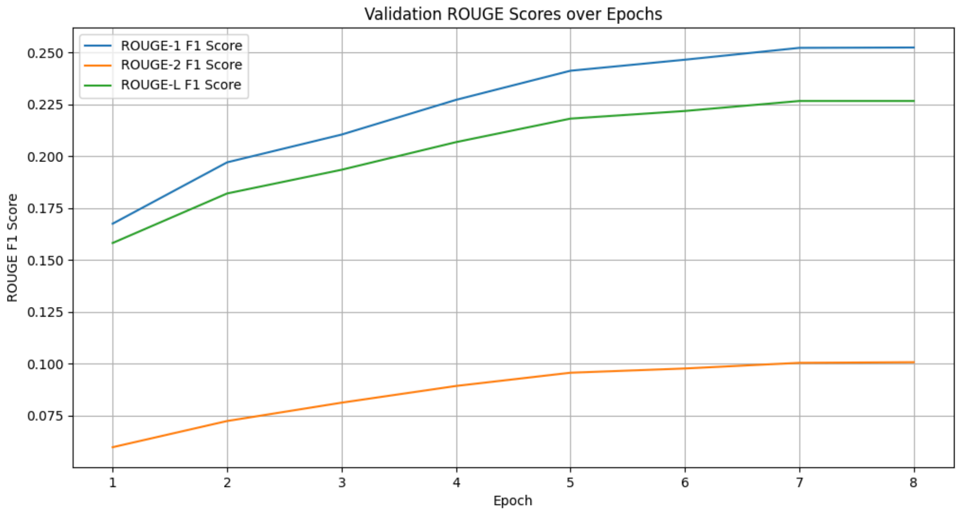
Hyperparameter (1%)
Describe your hyperparameter you use and how you decide it.

Answer:
Hyperparameters:

epoch	8
batch size	8
learning rate	5e-5
max input length	512
max target length	64

Adjust the batch size based on GPU memory usage. Monitor the learning curves to fine-tune the number of epochs and the learning rate. Set the maximum input and target lengths following online recommendations.

Learning Curves (1%)
Plot the learning curves (ROUGE versus training steps)
Answer:



Final validation rouge scores:

ROUGE-1 F1 Score: 0.2523159604390794

ROUGE-2 F1 Score: 0.10077909233388911

ROUGE-L F1 Score: 0.22656482765227043

Q3: Generation Strategies(6%)

Stratgies (2%)

Describe the detail of the following generation strategies:

- *Greedy*
- *Beam Search*
- *Top-k Sampling*
- *Top-p Sampling*
- *Temperature*

Answer:

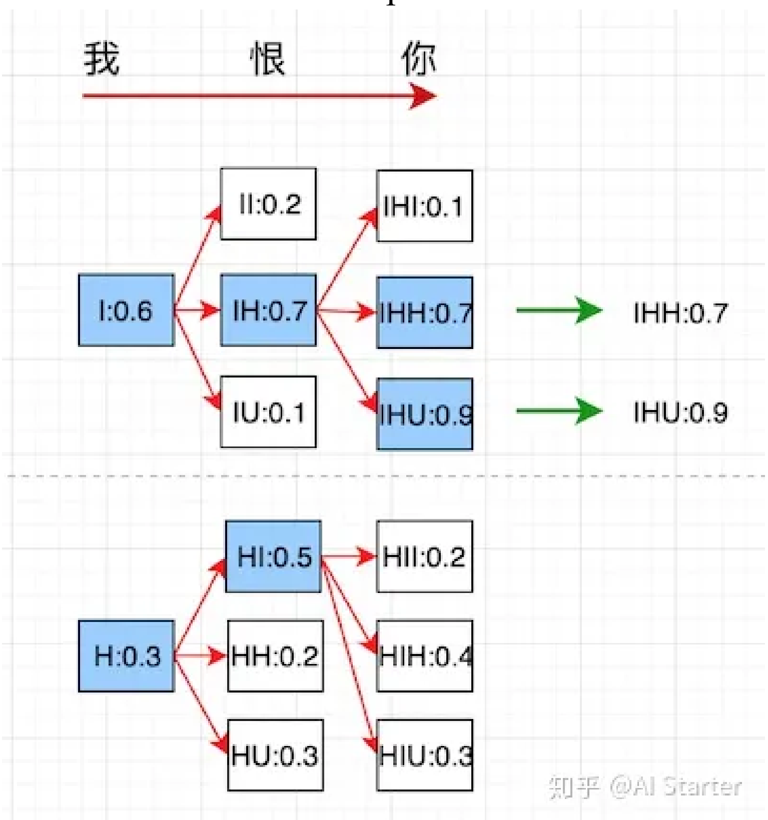
- *Greedy*

Greedy search generates text by selecting the token with the highest probability at each step. It might miss the globally best sequences.

- *Beam Search*

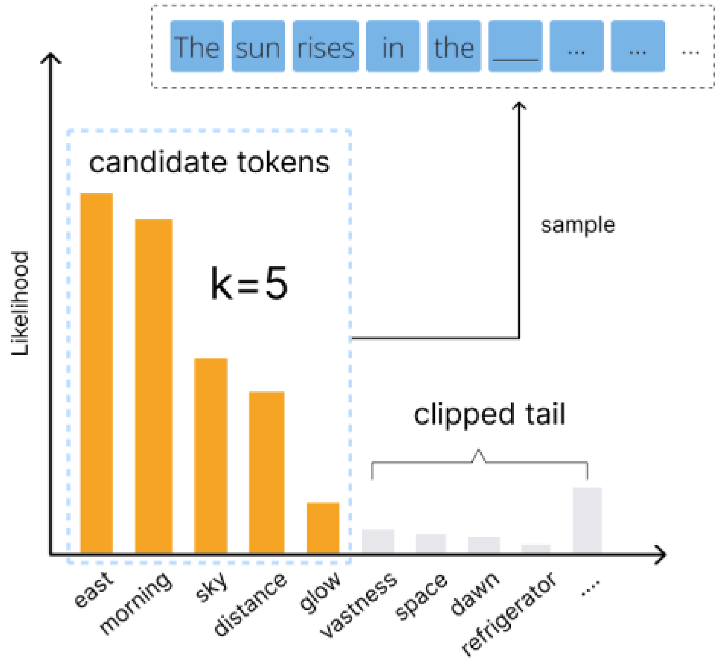
Keeps multiple sequences (beam width) at each step, selecting the best ones, leading to higher-quality results.

It uses a hyperparameter called beam size (k), which determines how many sequences to consider. At the first step, the top k most probable tokens are selected as candidates. At each subsequent step, the algorithm selects the top k sequences based on their cumulative probabilities from all possible combinations, maintaining k candidates throughout. Finally, the best sequence from these k candidates is chosen as the output.



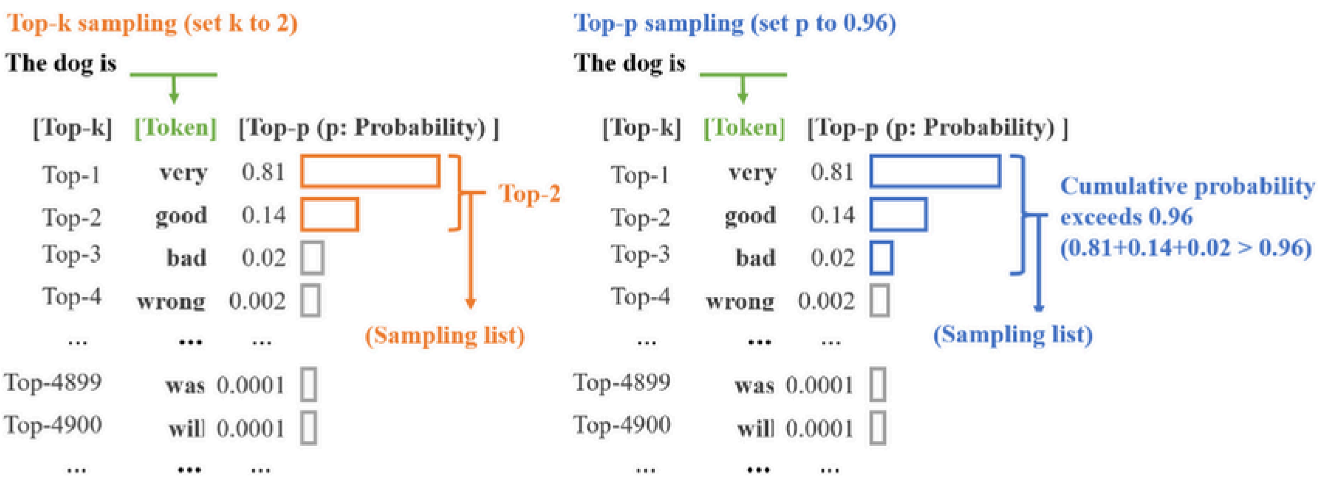
- *Top-k Sampling*

  Top-k sampling selects the next token from the top k most probable tokens. Instead of always choosing the token with the highest probability, it randomly samples from this limited set, introducing variability and creativity in the generation process.



- *Top-p Sampling*

  Top-p sampling dynamically selects a subset of tokens whose cumulative probability is above a threshold p (e.g., 0.9). It allows more flexibility than top-k, as the number of tokens considered varies, ensuring that only the most probable tokens are sampled while still maintaining randomness.



- *Temperature*

  Temperature controls the randomness of predictions by scaling the probabilities before sampling. A low temperature (<1) makes the model more conservative, focusing on high-probability tokens, while a high temperature (>1) makes the model more creative and exploratory by allowing low-probability tokens to be selected more frequently.

Hyperparameters (4%)

- *Try at least 2 settings of each strategies and compare the result.*
- *What is your final generation strategy? (you can combine any of them)*

Answer:

All settings use the same hyperparameters as shown in Q2. The final epoch's validation ROUGE score (F1*100) is used as the performance metric. The results are as follows:

stategies	settings	ROUGE-1	ROUGE-2	ROUGE-L
Greedy	batch_size: 4	25.6	9.5	22.9
	batch_size: 8	24.0	8.7	21.6
Beam Search	num_beams:4	25.2	10.0	22.6
	num_beams:8	25.4	10.3	22.8
Top-k	k:30	21.9	7.3	19.4
	k:50	21.6	7.2	19.2
Top-p	p:0.9	22.2	7.4	19.6
	p:0.8	22.5	7.7	20.0
Temperature	t:0.7	21.7	7.3	19.1
	t:1.1	18.0	5.2	15.7

For Greedy search, a smaller batch size of 4 yields higher ROUGE scores than a batch size of 8.

For Beam Search, increasing the number of beams from 4 to 8 slightly improves the ROUGE scores.

In Top-k Sampling, higher k values lead to lower ROUGE scores. In Top-p Sampling, reducing p from 0.9 to 0.8 increases all ROUGE scores.

For Temperature, a lower temperature of 0.7 results in better ROUGE scores compared to a higher temperature of 1.1.

I ultimately chose beam search with num_beams=8 as the final generation strategy because it has the best overall performance. Here is test results vs. public basline:

	ROUGE-1	ROUGE-2	ROUGE-L
baseline	22.0	8.5	20.5
ours	23.8	9.5	21.5