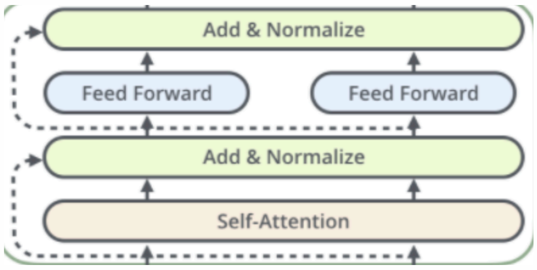
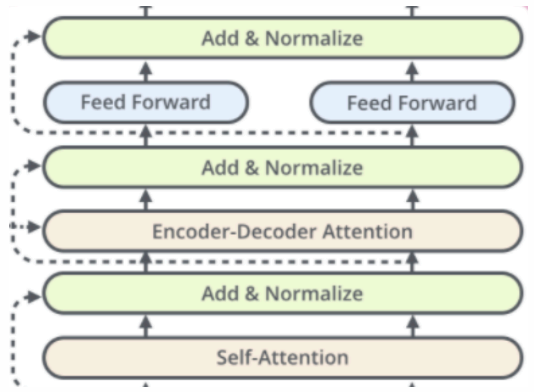
Q1: Model

- **Model**

  First, an input sequence will be mapped into a sequence of embeddings and pass into the encoder. The encoder comprises a stack of blocks. For each block, it follows the order of a self-attention layer, normalization layer, feed-forward network and normalization layer. After each normalization layer, a residual skip connection is used for each subcomponent's input to its output. The architecture of each encoder block is as follows.



  The decoder is similar to the structure of the encoder. The difference is that there is standard attention mechanism after normalization layer following the self-attention layer, and it attends to the output of the encoder. The output of final decoder block will be fed into a Linear layer with a softmax output. The architecture of each decoder block is as follows.



  Finally, the concatenation of outputs will become the result of summarization.

- **Preprocessing**

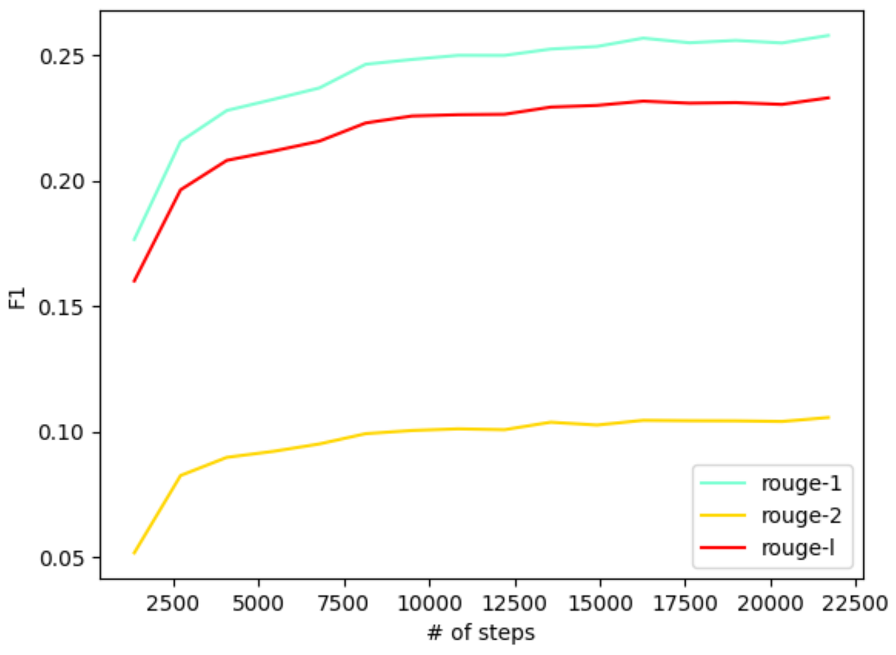
I used **T5Tokenizer** as my tokenizer. To encode input text and titles, I added truncation and set the max length to be 1024 for input sentences and 128 for titles. Also, padding was added and the padding strategy is "max_length".

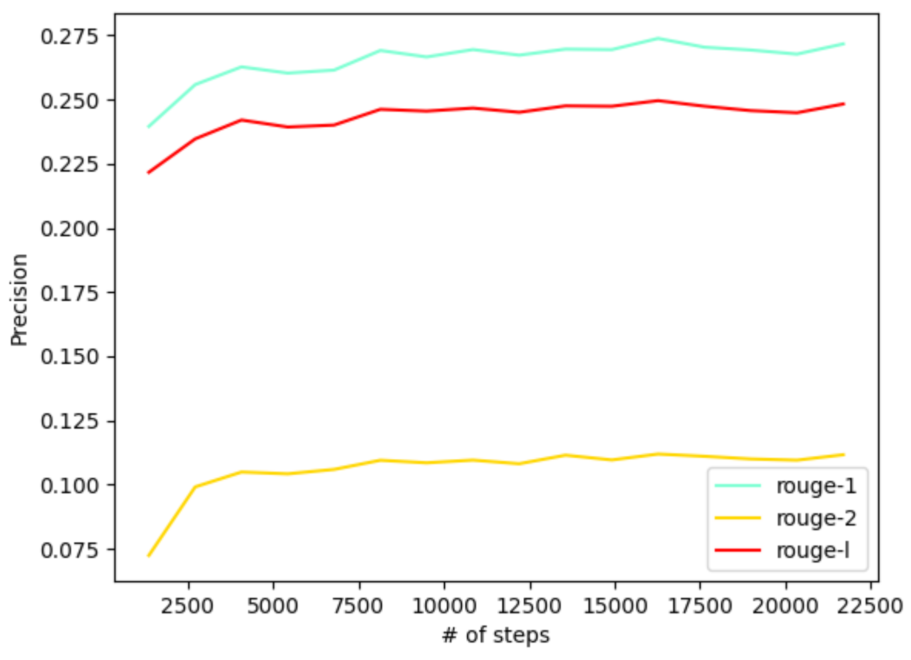
Q2: Training

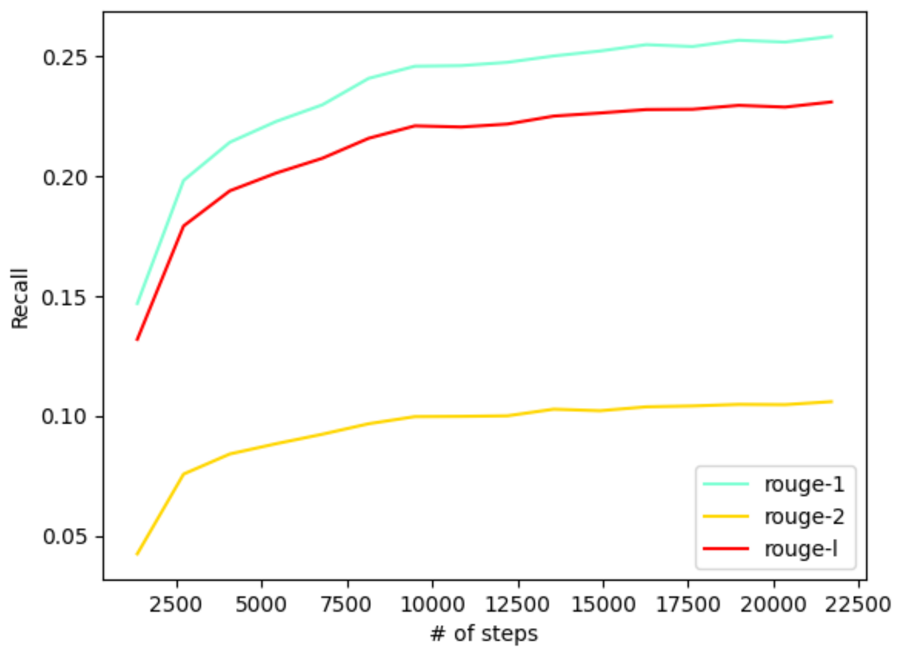
- **Hyperparameters**
  - input max length: 1024
  - target max length: 128
  - gardient accumulation steps: 16 (batch size: 1)
  - training epoch: 20
  - learning rate: 5×10^{-5} (with learning scheduler)
  - weight decay: 0.01
  - number of beams: 5

  Most of the above hyperparameters are the default hyperparameters from the documentation of huggingface.co. Learning rate, gradient accumulation steps and number of beams were decided by trial and error (from a list of candidates).

- **Learning Curve**







Q3: Generation Strategies

- **Strategies**
  - Greedy

    Greedy strategy selects the word with the highest probability to be the next word. That is, $w_t = \underset{w}{argmax} P(w|w_1, w2, \ldots, w_{t-1})$ for each t.
  - Beam Search

    Assume B is a hyperparameter indicating the number of beams. Beam Search strategy selects B hypotheses with highest probability at each time steps t. Eventually, it will select the one with the highest overall probability among the B hypotheses. When B is small, the output will become more ungrammatical and unnatural. In contrast, when B is large, the complexity of computation will be expensive but the output will be more reasonable.

  - Top-k sampling

    Top-k sampling strategy is a kind of sampling strategy but limits the sampling pool to the top-k probable words. That is, we can only sample from the top-k words via distribution. When k is large, the output will be more diverse but risky. Otherwise, when k is small, the output is safer and more generic.

  - Top-p sampling

    Top-p sampling strategy is similar to top-k sampling. The difference is that top-p sampling strategy samples from the smallest set of words, whose accumulation of probability exceeds p.

  - Temperature

    Temperature will slightly modify the probability distribution of each word. The new distribution is as follows. τ is the value of temperature. Higher temperature makes the distribution more uniform while lower temperature makes the distribution more spiky.

$$P(w_t) = \frac{e^{s_w/\tau}}{\sum_{w' \in V} e^{s_{w'}/\tau}}$$

- **Hyperparameters**
  - Settings:
    - min_length: 15
    - max_length: 128
    - repeatition_penalty: 2.5
    - length_penalty: 1.0

strategy \ score(F1*100)	**rouge-1**	**rouge-2**	**rouge-l**
Greedy	24.80	8.82	21.36
Beam Search num_beams=3	25.78	10.39	23.25
Beam Search num_beams=5	25.78	10.57	23.30
Beam Search num_beams=10	25.76	10.48	23.28
Top-k k=20	20.63	6.41	17.73
Top-k k=50	18.81	5.77	16.48
Top-p p=0.92	16.20	4.86	14.30
Top-k + Top-p k=50 p=0.92	20.20	6.35	17.44
Top-k + Top-p k=20 p=0.92	21.61	7.02	18.59
Top-k + Top-p +temperature k=20 p=0.92 τ=0.7	23.29	8.01	20.04

The final strategy that I used is **Beam Search** with num_beams=5.

Bonus: Applied RL on Summarization

- **Algorithm**

  The alogrithm I used is **Policy Gradient** and the reward function is simply $r = 10 \times avg(Rouge\text{-}N)$. Hence, the loss function is original_loss $\times$ r. The hyperparemters are as follows.

  - Learning Rate: 5×10^{-5}
  - Input Max Length: 1024
  - Target Max Length: 128
- **Compare to Supervised Learning**

The Rouge score of RL is as follows.

rouge-1	rouge-2	rouge-l
25.91	10.55	23.40

It's a little improvement on both rouge-1 and rouge-l score but the rouge-2 score is slightly lower than that of supervised learning. And I found that some of the output in SL might be a little bit nonsense, while that of RL is more reasonable. For example, there is a SL result "生肖運勢天天看!12生肖運勢如何嗎?" which is weird and that of RL is "生肖運勢天天看!12生肖運勢最不佳 預防爛桃花纏身敗運" which looks more reasonable.