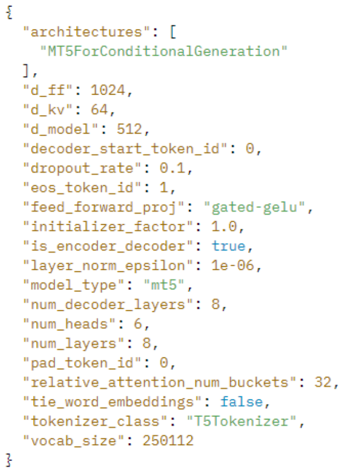
Q1 Model

◆ Model

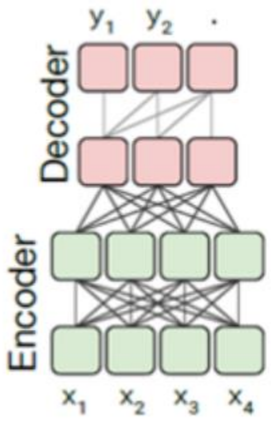
➢ Describe the model architecture and how it works on text summarization.

BERT 主要做 NLU 任務，對於 NLG 有點力不從心，但是 T5 用一種大一統的思想把 NLU 和 NLG 任務一起解決了，這一種大一統的思想就是：把所有的 NLP 任務都轉化成了文字到文字 Text2Text 格式的任務。 而 MT5 就是 Mutilingual 基於 T5 的架構。

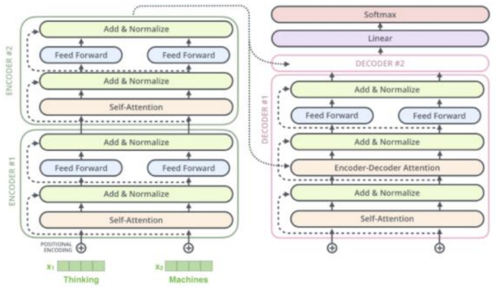
T5 是基於 transformer - Encoder-Decoder 架構如下圖



```
{
  "architectures": [
    "MT5ForConditionalGeneration"
  ],
  "d_ff": 1024,
  "d_kv": 64,
  "d_model": 512,
  "decoder_start_token_id": 0,
  "dropout_rate": 0.1,
  "eos_token_id": 1,
  "feed_forward_proj": "gated-gelu",
  "initializer_factor": 1.0,
  "is_encoder_decoder": true,
  "layer_norm_epsilon": 1e-06,
  "model_type": "mt5",
  "num_decoder_layers": 8,
  "num_heads": 6,
  "num_layers": 8,
  "pad_token_id": 0,
  "relative_attention_num_buckets": 32,
  "tie_word_embeddings": false,
  "tokenizer_class": "T5Tokenizer",
  "vocab_size": 250112
}
```



其整體架構如下:



MT5 可以做許多不同種類的任務，像翻譯任務，分類任務，回歸任務，而 summarization 任務就是 seq2seq 任務。

◆ Preprocessing

- ➢ Describe your preprocessing (e.g. tokenization, data cleaning and etc.)
- ➢

將 input content size (maintext)設為 1024，當 maintext 大於 1024 時，刪掉超過長度的字，不到 1024 的則做 padding，在將字轉為 vocab 裡的 id。

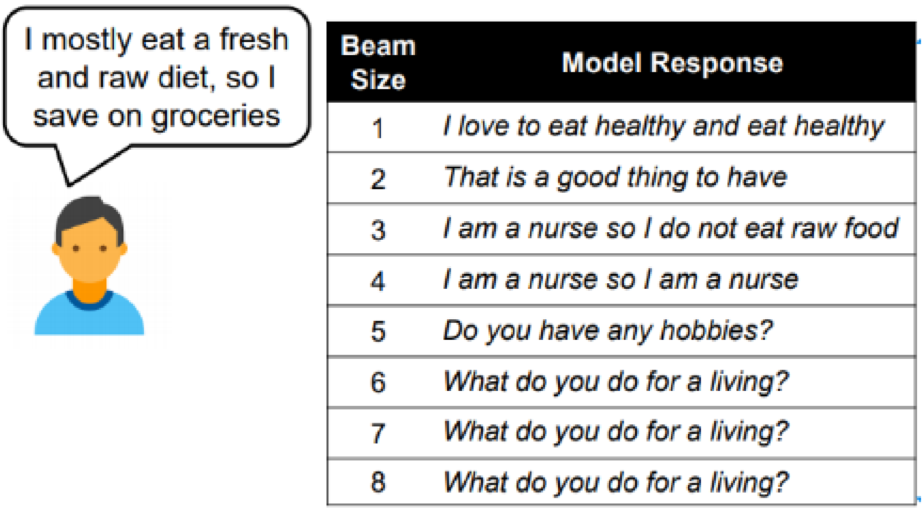
mT5 涵蓋了 101 種語言，總詞表(vocab)有 25 萬，而且它採用的 T5.結構的 Softmax 還不共享引數，這就導致了 Embedding 層佔用了相當多的引數量，比如 mT5 small 的參數量為 3 億，其中 Embedding 相關的就佔了 2.5 億，而大部分的參數我們都用不到，因此要精簡 Embedding ，其原理就是需要在兩個 Embedding 矩陣中刪除不需要的行就行了，關鍵在於如何決定要保留的 token，以及如何得到一個精簡後的 sentencepiece 模型。決定要保留的 token，簡單來想就是把中文的 token 保留下來，但是也不只是中文，英文的也要保留一部分。

Q2 Training

◆ Hyperparameter

- ➢ Describe your hyperparameter you use and how you decide it.

當選擇小的 beam size 可以使產生的文字較符合主題，但是文法會比較不通順。
當選擇大的 beam size 可以使產生的句子文法較通順且回答較正確，但是較通用且無相關性。


I mostly eat a fresh and raw diet, so I save on groceries

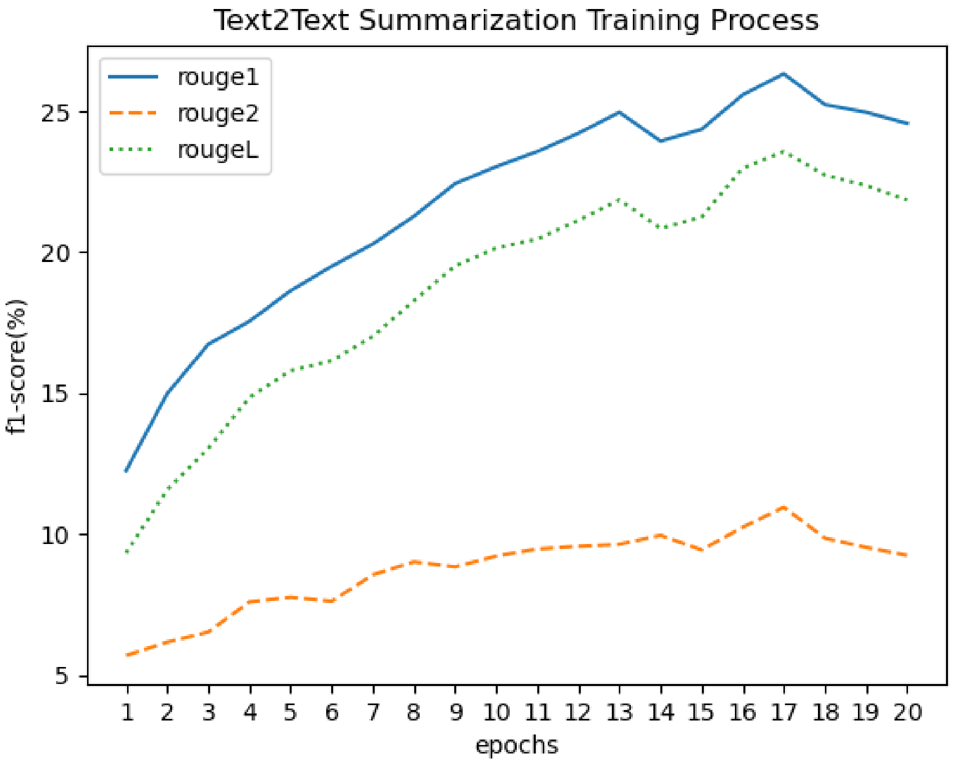
Beam Size	Model Response
1	*I love to eat healthy and eat healthy*
2	*That is a good thing to have*
3	*I am a nurse so I do not eat raw food*
4	*I am a nurse so I am a nurse*
5	*Do you have any hobbies?*
6	*What do you do for a living?*
7	*What do you do for a living?*
8	*What do you do for a living?*

經過多次的測試，我最後使用 beam size = 7，並搭配 No_repeat_ngram_size=3 來防止一直產生相同的詞。

```
max_source_length=2048
max_target_length=128
learning_rate=3e-5
train_batch_size=1
gradient_accumulation_steps=16
eval_batch_size=6
```

◆ Learning Curves

➢ Plot the learning curves (ROUGE versus training steps)

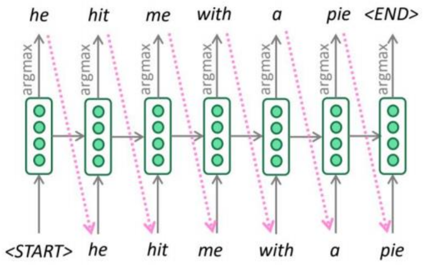


Q3 Generation Strategies

◆ strategies

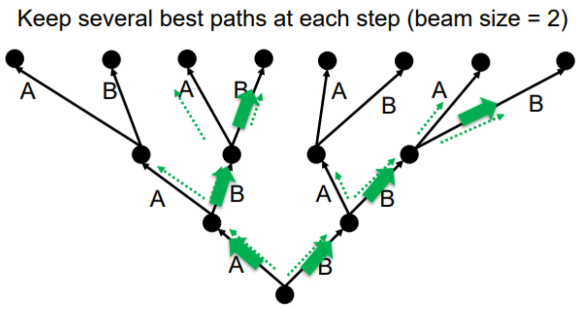
- ➢ Describe the detail of the following generation strategies:
  - Greedy

    選擇最大機率的字(argmax)



  - Beam Search

    選擇 k 個最大機率的 sequence 並且找出最佳(最大機率)的一句 sequence。



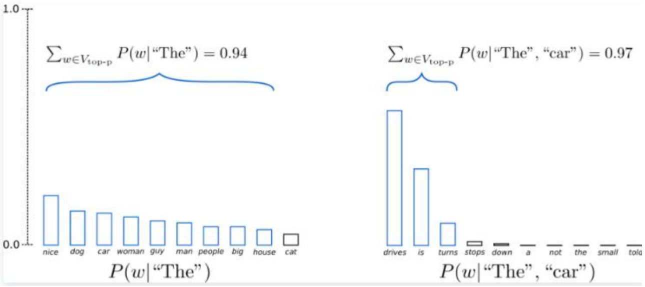
  - Top-k Sampling

    將挑選出 K 個最有可能的下一個單詞，並且僅在這 K 個下一個單詞之間重新為它們分配概率。

- Top-p Sampling

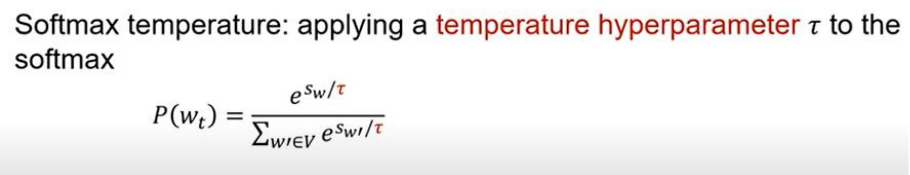
  在 Top-p 採樣中，不是僅從最可能的 K 個單詞中採樣，而是從其累積概率超過一個閥值 的最小可能單詞集中進行選擇，然後將這組單詞重新分配概率。這樣，單詞集合的大小（也就是集合中單詞的數量）可以根據下一個單詞的概率分布動態地增加或減少。

  如下所示:



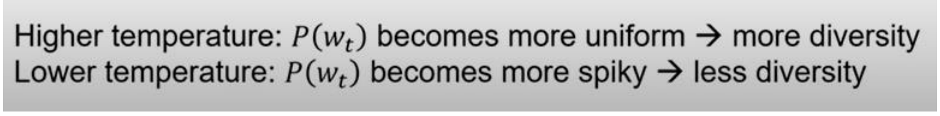
- Temperature (不是一種 strategy，而是一種附加的參數可以搭配不同的 Generation Strategies 使用)

  Ex：



Softmax temperature: applying a temperature hyperparameter τ to the softmax

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



Higher temperature: $P(w_t)$ becomes more uniform → more diversity
Lower temperature: $P(w_t)$ becomes more spiky → less diversity

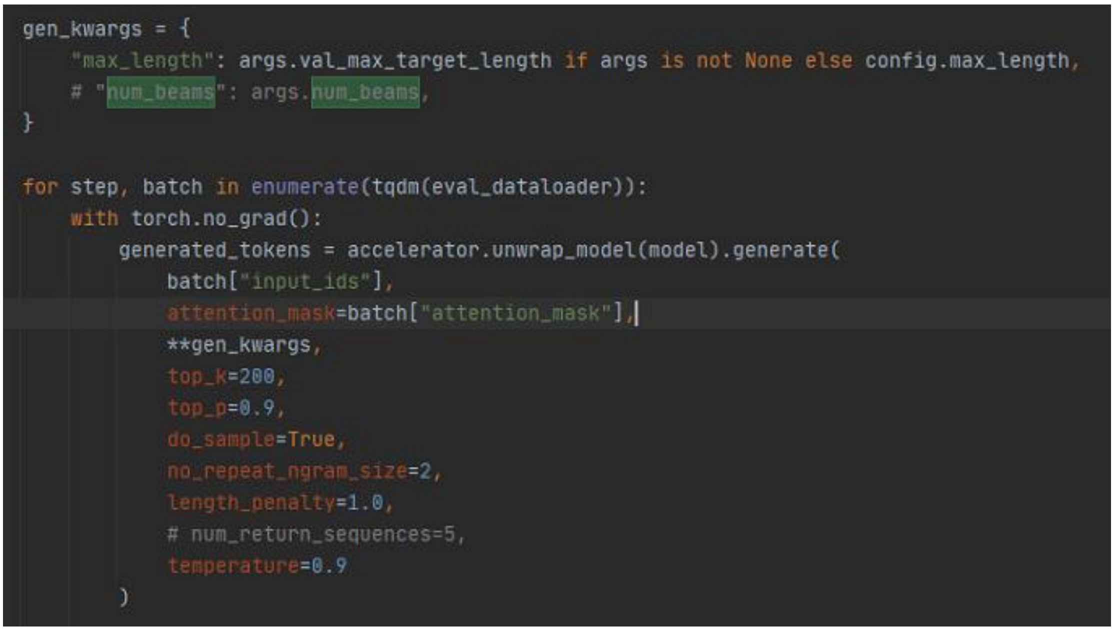
◆ Hyperparameters

- Try at least 2 settings of each strategies and compare the result.
  - Top_k (200) + top_p (0.9) + temperature(0.9)

Epoch=20

Input size : 2048

Output size : 128



```
gen_kwargs = {
    "max_length": args.val_max_target_length if args is not None else config.max_length,
    # "num_beams": args.num_beams,
}

for step, batch in enumerate(tqdm(eval_dataloader)):
    with torch.no_grad():
        generated_tokens = accelerator.unwrap_model(model).generate(
            batch["input_ids"],
            attention_mask=batch["attention_mask"],
            **gen_kwargs,
            top_k=200,
            top_p=0.9,
            do_sample=True,
            no_repeat_ngram_size=2,
            length_penalty=1.0,
            # num_return_sequences=5,
            temperature=0.9
        )
```

Result:

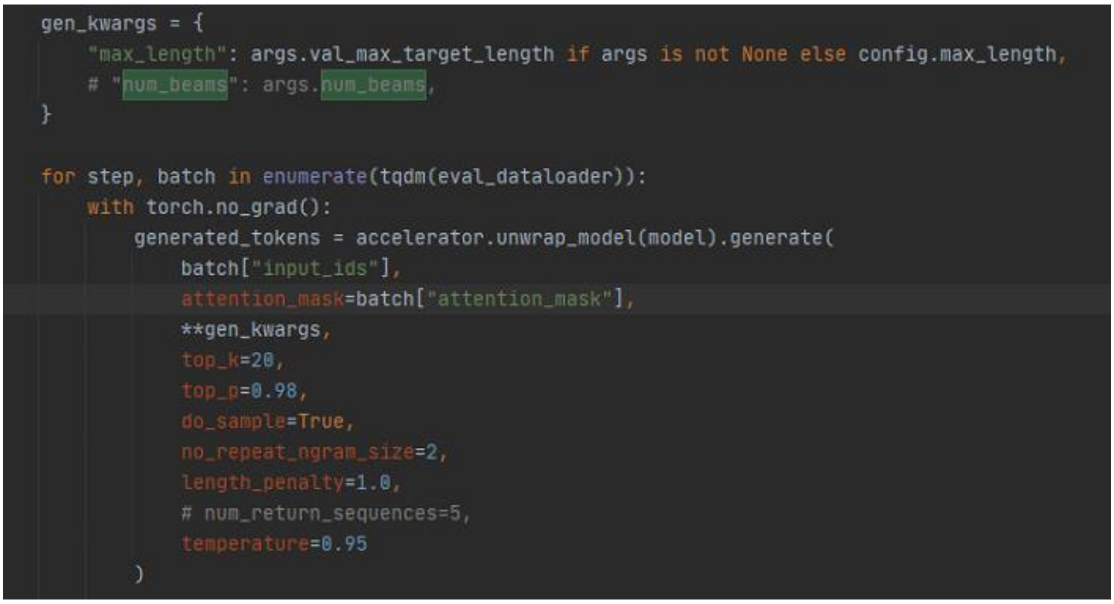
```
{
   "rouge-1": {
      "f": 0.233834156220925,
      "p": 0.23770516832925132,
      "r": 0.24138568730958734
   },
   "rouge-2": {
      "f": 0.08343446488783317,
      "p": 0.08486256536017532,
      "r": 0.08652563689559623
   },
   "rouge-l": {
      "f": 0.20825253250619444,
      "p": 0.2131501573446441,
      "r": 0.2131291228248939
   }
}
```

- Top_k (20) + top_p (0.98) + temperature(0.95)

Epoch=20

Input size : 2048

Output size : 128



```
gen_kwargs = {
    "max_length": args.val_max_target_length if args is not None else config.max_length,
    # "num_beams": args.num_beams,
}

for step, batch in enumerate(tqdm(eval_dataloader)):
    with torch.no_grad():
        generated_tokens = accelerator.unwrap_model(model).generate(
            batch["input_ids"],
            attention_mask=batch["attention_mask"],
            **gen_kwargs,
            top_k=20,
            top_p=0.98,
            do_sample=True,
            no_repeat_ngram_size=2,
            length_penalty=1.0,
            # num_return_sequences=5,
            temperature=0.95
        )
```

Result:

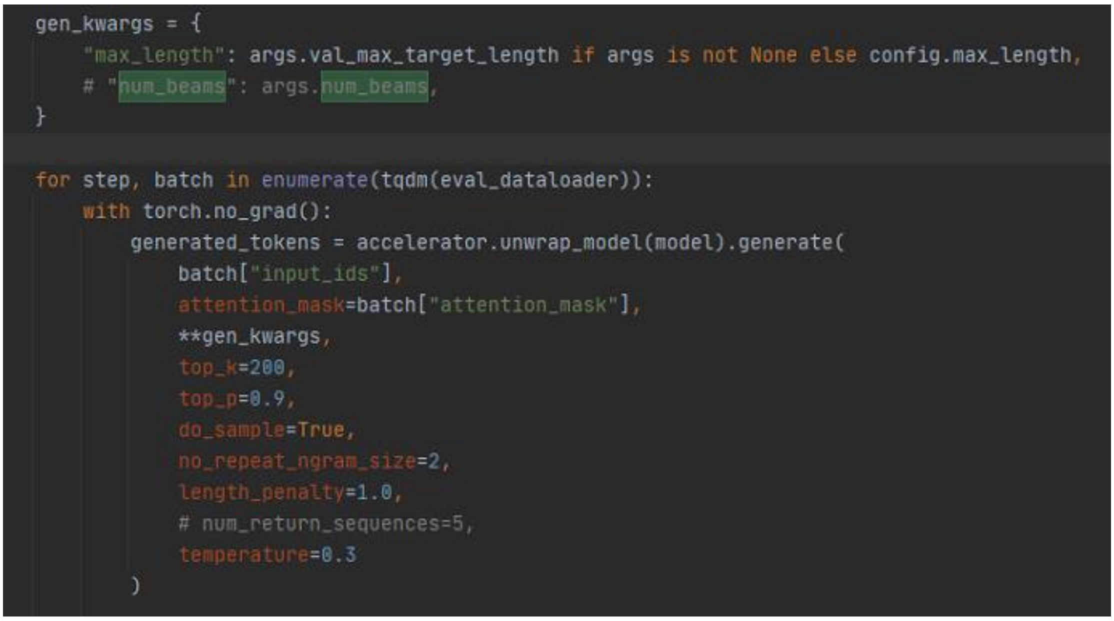
```
{
  "rouge-1": {
    "f": 0.23535839680203713,
    "p": 0.2394467326381215,
    "r": 0.24292915867524267
  },
  "rouge-2": {
    "f": 0.08377209643883991,
    "p": 0.08490877035974002,
    "r": 0.08699999718388637
  },
  "rouge-l": {
    "f": 0.20798106454380164,
    "p": 0.21321483321684892,
    "r": 0.2125747874588796
  }
}
```

- Top_k (200) + top_p (0.9) + temperature(0.3)

Epoch=20

Input size : 2048

Output size : 128



```python
gen_kwargs = {
    "max_length": args.val_max_target_length if args is not None else config.max_length,
    # "num_beams": args.num_beams,
}

for step, batch in enumerate(tqdm(eval_dataloader)):
    with torch.no_grad():
        generated_tokens = accelerator.unwrap_model(model).generate(
            batch["input_ids"],
            attention_mask=batch["attention_mask"],
            **gen_kwargs,
            top_k=200,
            top_p=0.9,
            do_sample=True,
            no_repeat_ngram_size=2,
            length_penalty=1.0,
            # num_return_sequences=5,
            temperature=0.3
        )
```

Result:

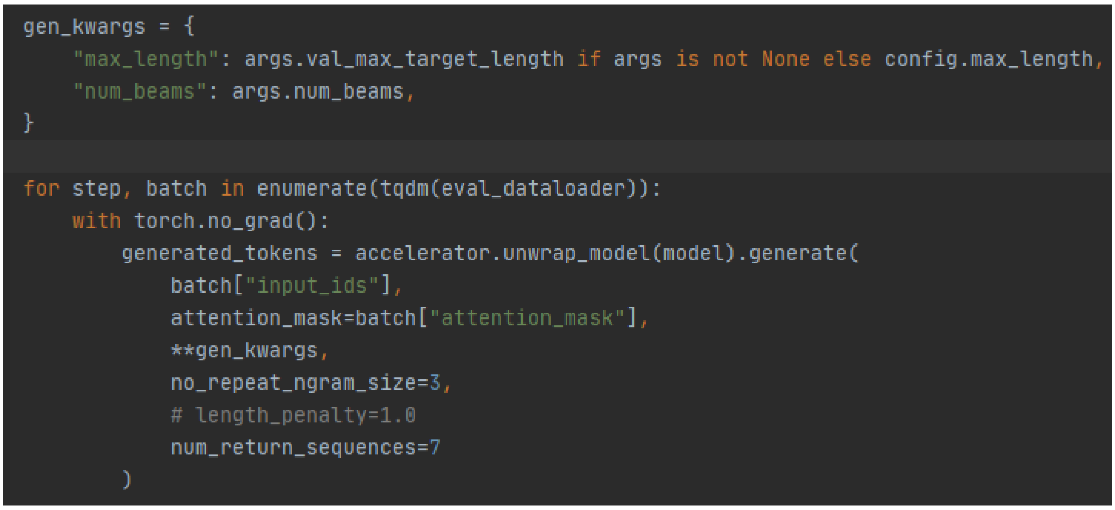
```
{
   "rouge-1": {
      "f": 0.2595305394176496,
      "p": 0.2692356949705075,
      "r": 0.2620393624302309
   },
   "rouge-2": {
      "f": 0.09864152097394568,
      "p": 0.1025161004487499,
      "r": 0.0998890773058013
   },
   "rouge-l": {
      "f": 0.2304447916811395,
      "p": 0.241074326424151,
      "r": 0.23025836914758732
   }
}
```

- Beams_number (7) + ngram(3)

Epoch=20

Input size = 2048

Output size = 128



```
gen_kwargs = {
    "max_length": args.val_max_target_length if args is not None else config.max_length,
    "num_beams": args.num_beams,
}

for step, batch in enumerate(tqdm(eval_dataloader)):
    with torch.no_grad():
        generated_tokens = accelerator.unwrap_model(model).generate(
            batch["input_ids"],
            attention_mask=batch["attention_mask"],
            **gen_kwargs,
            no_repeat_ngram_size=3,
            # length_penalty=1.0
            num_return_sequences=7
        )
```

Result:

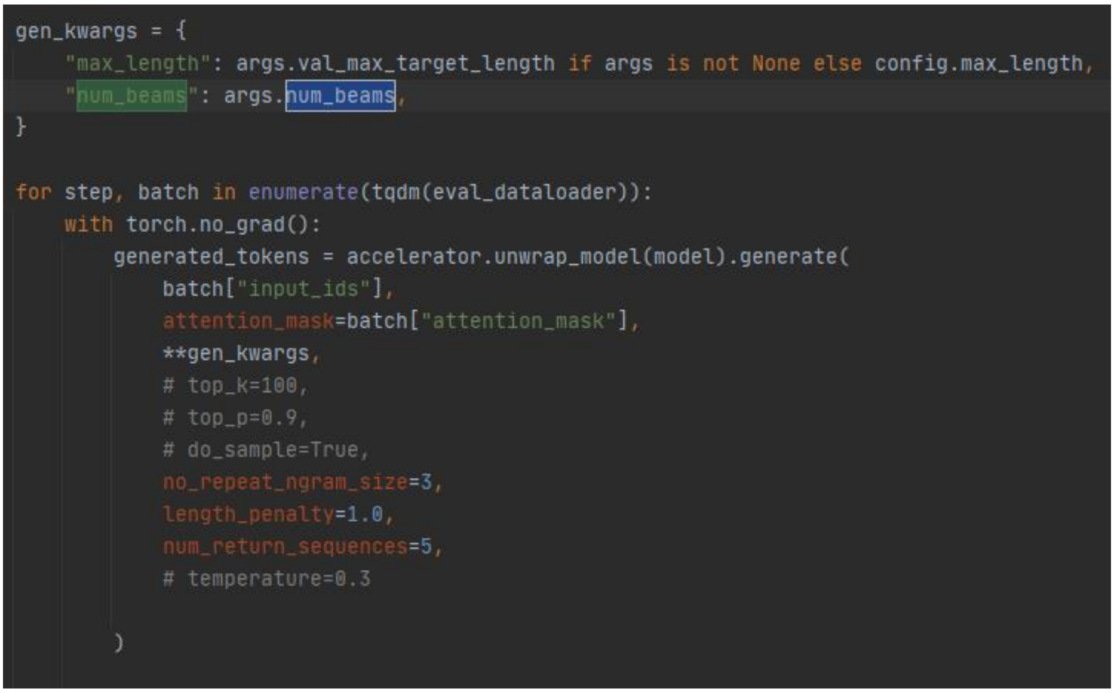
```
{
  "rouge-1": {
    "f": 0.2783196093622653,
    "p": 0.2854129922686215,
    "r": 0.2843418305349766
  },
  "rouge-2": {
    "f": 0.1164916516785287,
    "p": 0.11983329519416126,
    "r": 0.11913228357826665
  },
  "rouge-l": {
    "f": 0.24909060026246263,
    "p": 0.2572848723185751,
    "r": 0.2522693397227175
  }
}
```

- Beams_number (5) + ngram(3)

Epoch=20

Input size = 2048

Output size = 128



```python
gen_kwargs = {
    "max_length": args.val_max_target_length if args is not None else config.max_length,
    "num_beams": args.num_beams,
}

for step, batch in enumerate(tqdm(eval_dataloader)):
    with torch.no_grad():
        generated_tokens = accelerator.unwrap_model(model).generate(
            batch["input_ids"],
            attention_mask=batch["attention_mask"],
            **gen_kwargs,
            # top_k=100,
            # top_p=0.9,
            # do_sample=True,
            no_repeat_ngram_size=3,
            length_penalty=1.0,
            num_return_sequences=5,
            # temperature=0.3

        )
```

Result:

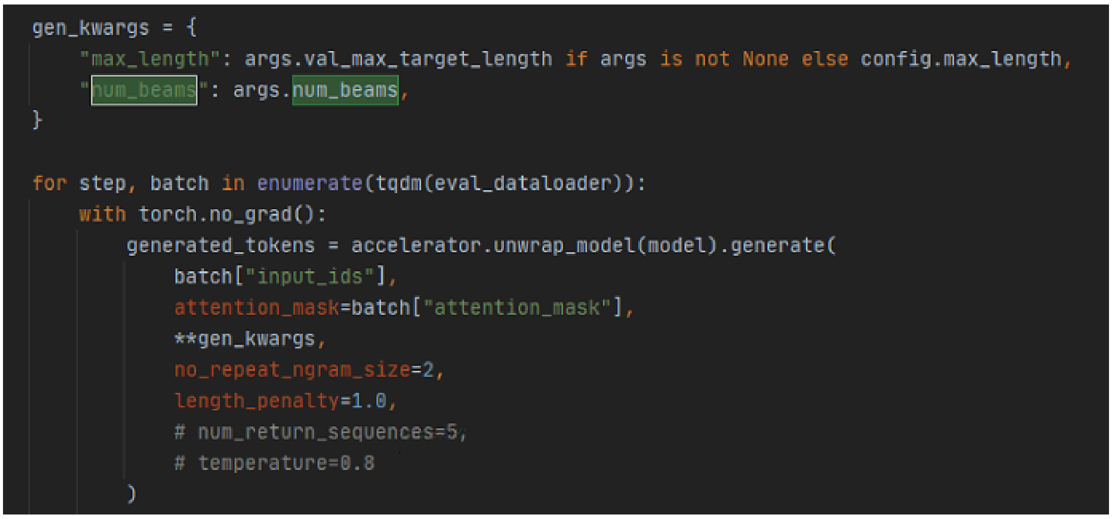
```
{
  "rouge-1": {
    "f": 0.2765556796885529,
    "p": 0.2840055329476194,
    "r": 0.2822207147691371
  },
  "rouge-2": {
    "f": 0.11531495092447706,
    "p": 0.11870296027625428,
    "r": 0.11783350736967611
  },
  "rouge-l": {
    "f": 0.2481989948995724,
    "p": 0.2569447884469682,
    "r": 0.2508616852042255
  }
}
```

- Beams_number (1) == use greedy + ngram(2)

Epoch=20

Input size = 2048

Output size = 128



```python
gen_kwargs = {
    "max_length": args.val_max_target_length if args is not None else config.max_length,
    "num_beams": args.num_beams,
}

for step, batch in enumerate(tqdm(eval_dataloader)):
    with torch.no_grad():
        generated_tokens = accelerator.unwrap_model(model).generate(
            batch["input_ids"],
            attention_mask=batch["attention_mask"],
            **gen_kwargs,
            no_repeat_ngram_size=2,
            length_penalty=1.0,
            # num_return_sequences=5,
            # temperature=0.8
        )
```

Result:

```
{
   "rouge-1": {
      "f": 0.25963228071846917,
      "p": 0.26944781520616207,
      "r": 0.2619231585166096
   },
   "rouge-2": {
      "f": 0.09855605665930152,
      "p": 0.10231209321021209,
      "r": 0.09995070765347569
   },
   "rouge-l": {
      "f": 0.2304594506771392,
      "p": 0.24124622876238538,
      "r": 0.23018201278218925
   }
}
```

- What is your final generation strategy? (you can combine any of them)

  - Beam search (num=7)
  - Length penalty =1.0 target 長度限制
  - No_repeat_ngram_size=3 target 相同詞彙出現限制
  - num_return_sequences=7 predict 返回最佳前五句

```
{
  "rouge-1": {
    "f": 0.2783196093622653,
    "p": 0.2854129922686215,
    "r": 0.2843418305349766
  },
  "rouge-2": {
    "f": 0.1164916516785287,
    "p": 0.11983329519416126,
    "r": 0.11913228357826665
  },
  "rouge-l": {
    "f": 0.24909060026246263,
    "p": 0.2572848723185751,
    "r": 0.2522693397227175
  }
}
```