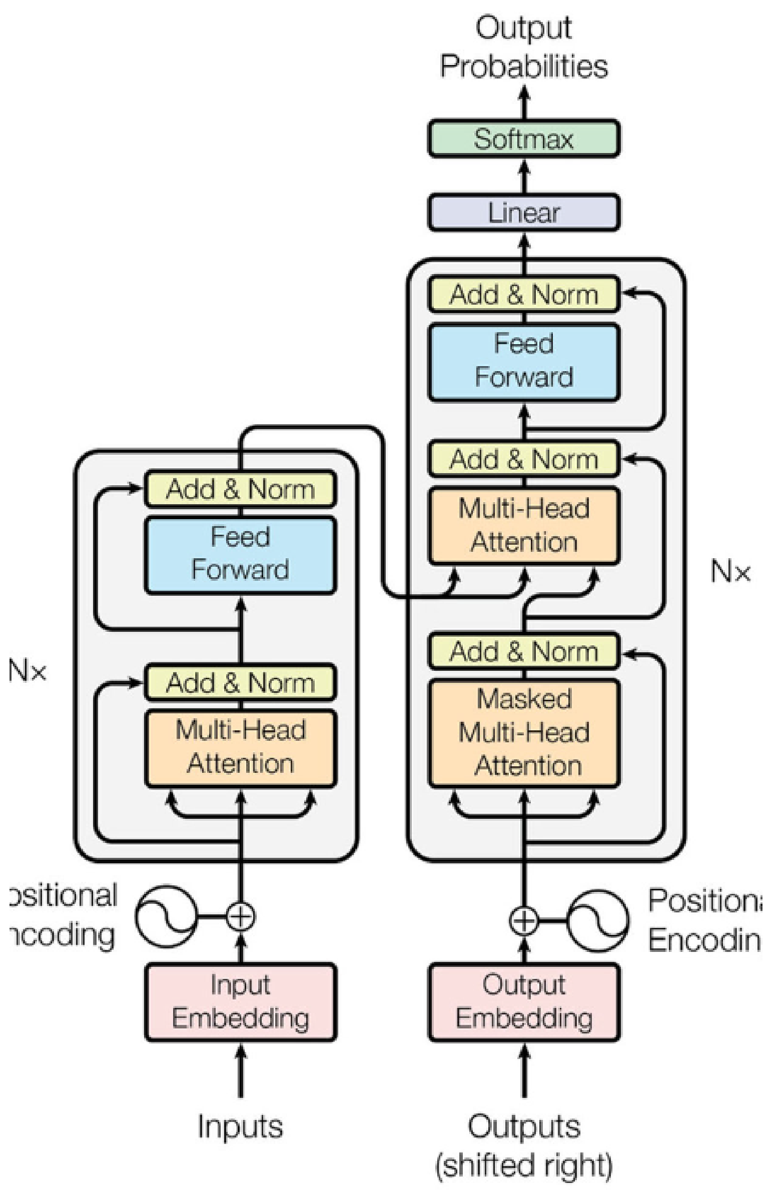
Q1 : Model

- Model (“architectures” : MT5ForConditionalGeneration)
  - T5 model as a transformer Encoder-Decoder frame model, that is trained in an end-to-end manner with text as input and modified text as output. When doing some sequence to sequence tasks like summarization, it will rewrite sentences when necessary rather than just picking up sentences directly from the original text.
  - Encoder
    - Input embedding
    - Adjust key value, attention mask, head mask, hidden states value
    - Layer-by-layer encoding (8 blocks in self.encoder), each block consists of a Self-Attention Layer and Feed Forward Layer.
    - LN and dropout
    - Save Dict
  - Decoder
    - Output embedding
    - Adjust key value, attention mask, head mask, hidden states value
    - Layer-by-layer decoding (8 blocks in self.encoder), each block
    - LN and dropout
    - save Dict
    - Get generated words



- Preprocessing
  - The T5 tokenizer was sentencepiece, consist by byte pair encoding and unigram language model. Splitting the text into sentences, and then cut to subwords, lowering the case of all words and doing truncation and padding. At the same time, will also add a prefix as a new input to put into the tokenizer.

Q2 : Training

- Hyperparameter
  - batch size : 4
    - When batch size gets more big it will present a more stable of the model performance, but when using a task such like summarization, the batch size usually set to 2~4, cause the computing requirements is too big that may cause “CUDA OUT OF MEMORY” when using a big batch size.
  - gradient accumulation : 4

- But when batch size comes too small it may cause poor performance, so we need to set gradient accumulation steps to fix the gradient compute error.

➢ optimizer : Adafactor

- Adafactor abandons the momentum that Adam has, Adafactor purpose is to decrease the usage of the VRAM, cause in the NLP task adaptive learning rate is more important than SGD plus momentum when trying to get better performance.

➢ learning rate : 1e-3

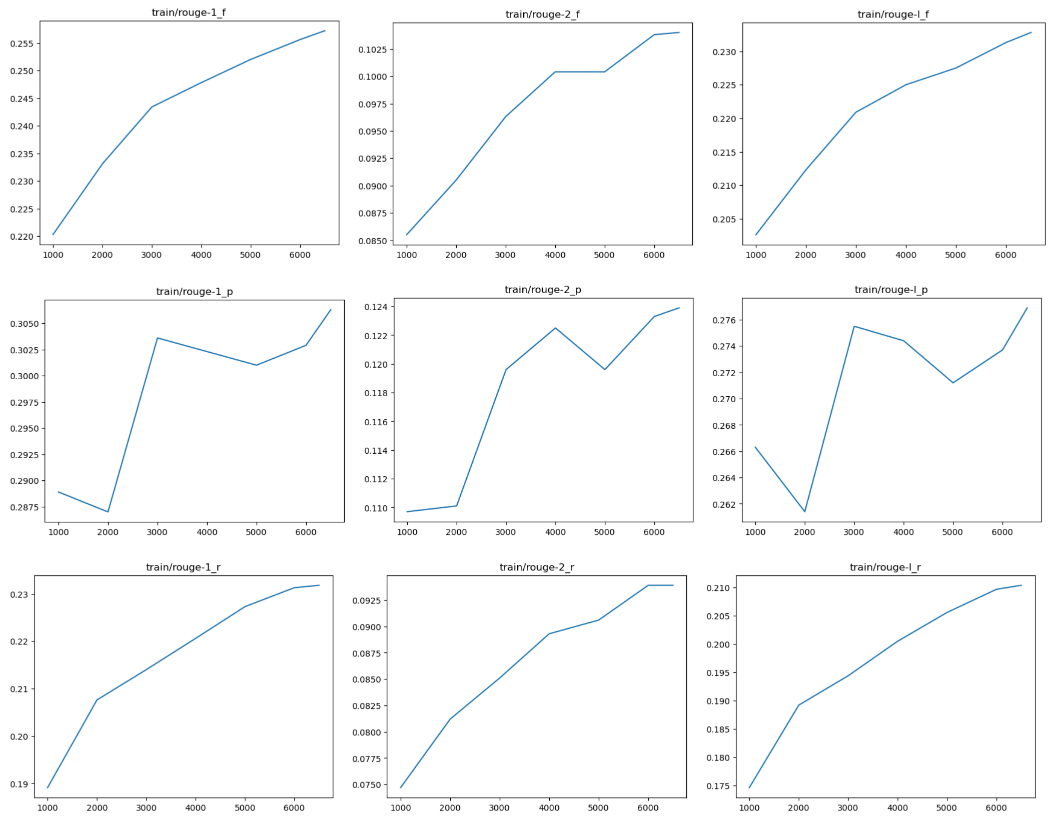
➢ warmup ratio : 0.1

➢ epochs : 5

➢ max source length : 1024

➢ max target length : 256

❖ Learning Curves



Q3 : Generation Strategies

❖ Strategies

➢ Greedy

- Greedy will always choose the biggest $P(w|w_1...w_{t-1})$ in each time step.

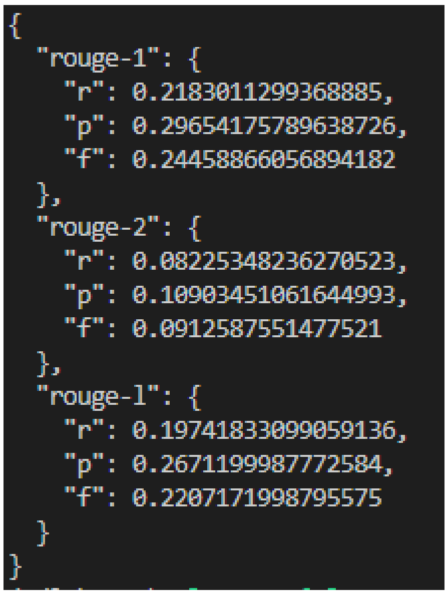
➢ Beam Search

- Beam Search is an optimization of best-first search that reduces its memory requirements, it will choose numbers of num beams for candidate in each time step, when predicting the next time step, use the candidate that prefix time step generated and choose numbers of num beams for this time step, keep doing this until the max length.

➢ Top-k Sampling

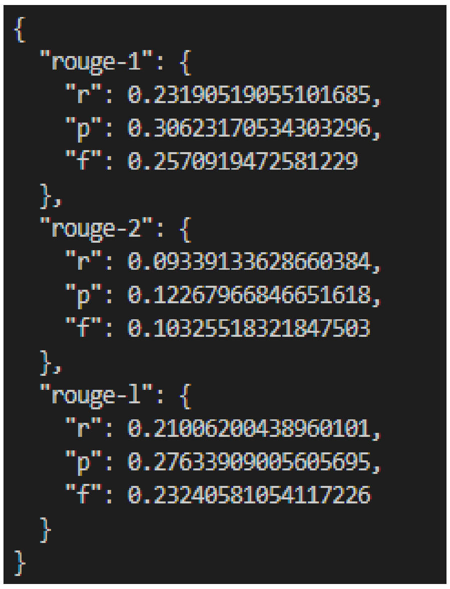
        - Choose the top-k $P(w|w_1...w_{t-1})$ tokens, then accumulate their probability to get P', and then adjust $P(w|w_1...w_{t-1})$ as $P(w|w_1...w_{t-1})$ / P', last we sample one token as output token.
    - ➢ Top-p Sampling
        - adjust the shortcoming that Top-k Sampling have, the tokens it selected will adjust as the change of the distribution of sentence input. ex : the length of the sentences.
    - ➢ Temperature
        - import a temperature t in softmax to change the vocabulary probability distribution, that $t \in [0, 1)$. When t -> 0, will adjust to greedy decoding. When t -> infinite, it will adjust to uniform sampling. We use this to avoid sampling from tail distribution.

- ❖ Hyperparameter
    - ➢ greedy

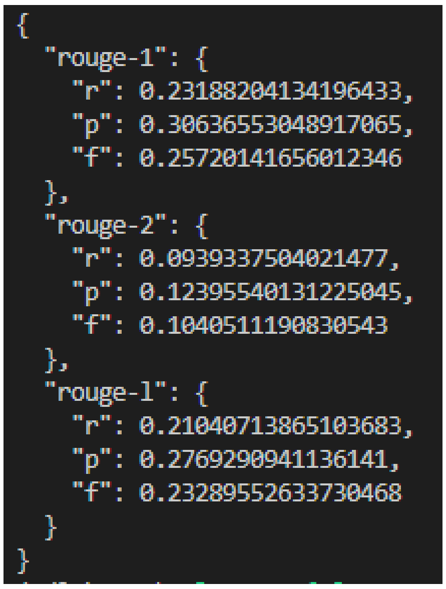


```
{
  "rouge-1": {
    "r": 0.2183011299368885,
    "p": 0.29654175789638726,
    "f": 0.24458866056894182
  },
  "rouge-2": {
    "r": 0.08225348236270523,
    "p": 0.10903451061644993,
    "f": 0.0912587551477521
  },
  "rouge-l": {
    "r": 0.19741833099059136,
    "p": 0.2671199987772584,
    "f": 0.2207171998795575
  }
}
```

    - ➢ beam search 4 v.s. beam search 5

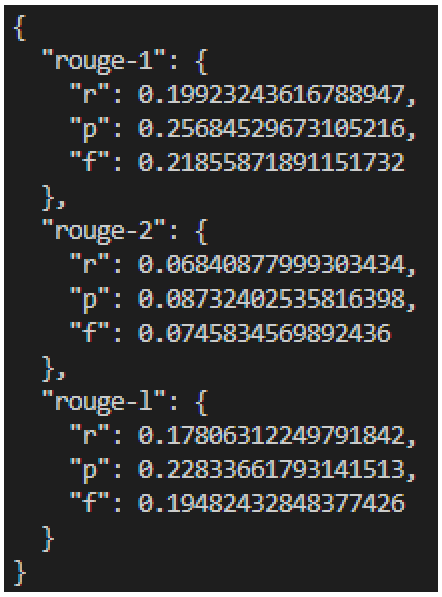


```
{
  "rouge-1": {
    "r": 0.23190519055101685,
    "p": 0.30623170534303296,
    "f": 0.2570919472581229
  },
  "rouge-2": {
    "r": 0.09339133628660384,
    "p": 0.12267966846651618,
    "f": 0.1032551832184750З
  },
  "rouge-l": {
    "r": 0.21006200438960101,
    "p": 0.27633909005605695,
    "f": 0.23240581054117226
  }
}
```

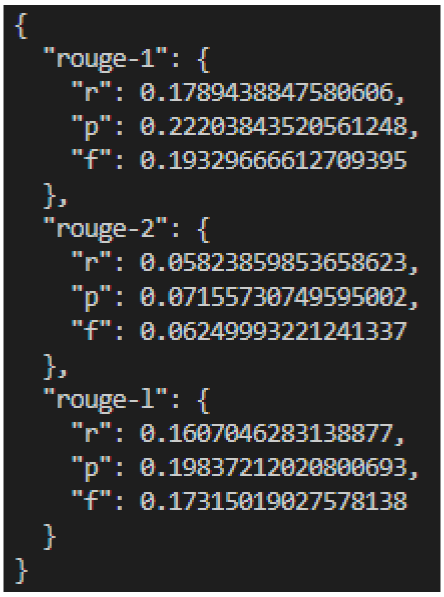


```
{
  "rouge-1": {
    "r": 0.23188204134196433,
    "p": 0.30636553048917065,
    "f": 0.25720141656012346
  },
  "rouge-2": {
    "r": 0.0939337504021477,
    "p": 0.12395540131225045,
    "f": 0.1040511190830543
  },
  "rouge-l": {
    "r": 0.21040713865103683,
    "p": 0.2769290941136141,
    "f": 0.23289552633730468
  }
}
```

- Top-k Sampling 10 v.s 50

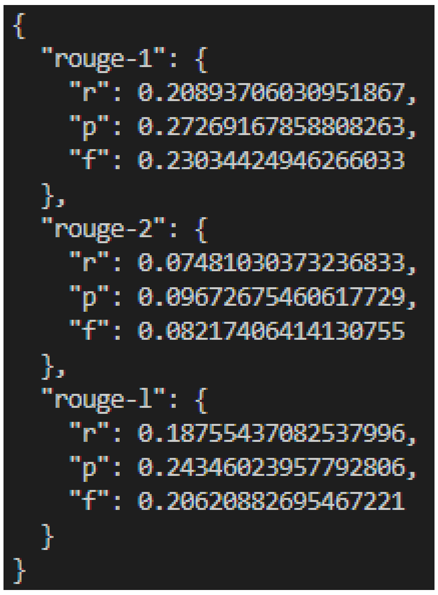


```
{
  "rouge-1": {
    "r": 0.19923243616788947,
    "p": 0.25684529673105216,
    "f": 0.21855871891151732
  },
  "rouge-2": {
    "r": 0.06840877999303434,
    "p": 0.08732402535816398,
    "f": 0.0745834569892436
  },
  "rouge-l": {
    "r": 0.17806312249791842,
    "p": 0.22833661793141513,
    "f": 0.19482432848377426
  }
}
```

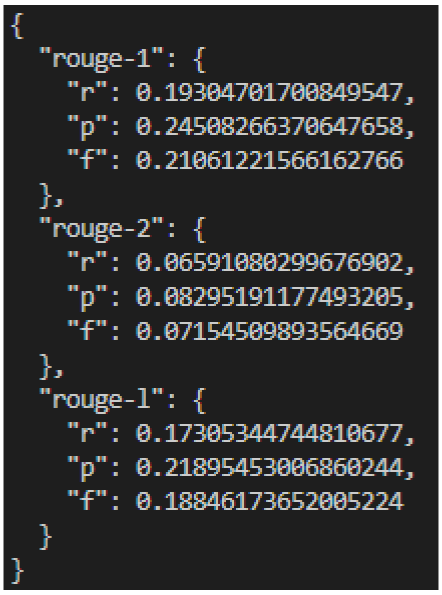


```
{
  "rouge-1": {
    "r": 0.1789438847580606,
    "p": 0.22203843520561248,
    "f": 0.19329666612709395
  },
  "rouge-2": {
    "r": 0.05823859853658623,
    "p": 0.07155730749595002,
    "f": 0.06249993221241337
  },
  "rouge-l": {
    "r": 0.1607046283138877,
    "p": 0.19837212020800693,
    "f": 0.17315019027578138
  }
}
```

- Top-p Sampling 0.6 v.s 0.9

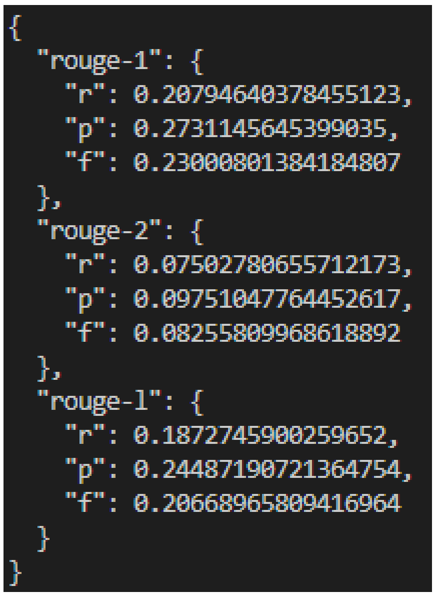


```
{
  "rouge-1": {
    "r": 0.20893706030951867,
    "p": 0.27269167858808263,
    "f": 0.23034424946266033
  },
  "rouge-2": {
    "r": 0.07481030373236833,
    "p": 0.09672675460617729,
    "f": 0.08217406414130755
  },
  "rouge-l": {
    "r": 0.18755437082537996,
    "p": 0.24346023957792806,
    "f": 0.20620882695467221
  }
}
```

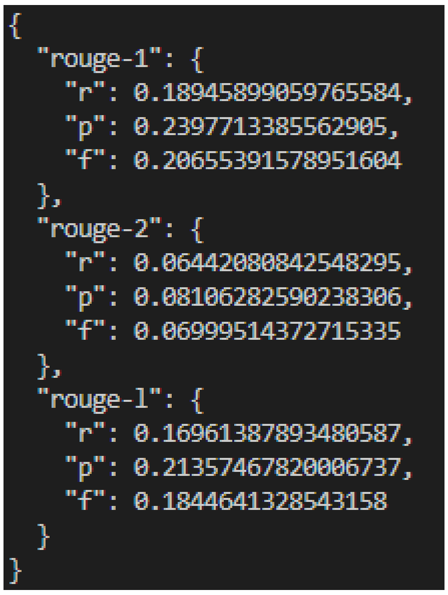


```
{
  "rouge-1": {
    "r": 0.19304701700849547,
    "p": 0.24508266370647658,
    "f": 0.21061221566162766
  },
  "rouge-2": {
    "r": 0.06591080299676902,
    "p": 0.08295191177493205,
    "f": 0.07154509893564669
  },
  "rouge-l": {
    "r": 0.17305344744810677,
    "p": 0.21895453006860244,
    "f": 0.18846173652005224
  }
}
```

- Temperature 0.6 v.s. 0.9



```
{
  "rouge-1": {
    "r": 0.20794640378455123,
    "p": 0.2731145645399035,
    "f": 0.23000801384184807
  },
  "rouge-2": {
    "r": 0.07502780655712173,
    "p": 0.09751047764452617,
    "f": 0.08255809968618892
  },
  "rouge-l": {
    "r": 0.1872745900259652,
    "p": 0.24487190721364754,
    "f": 0.20668965809416964
  }
}
```



```
{
  "rouge-1": {
    "r": 0.18945899059765584,
    "p": 0.2397713385562905,
    "f": 0.20655391578951604
  },
  "rouge-2": {
    "r": 0.06442080842548295,
    "p": 0.08106282590238306,
    "f": 0.06999514372715335
  },
  "rouge-l": {
    "r": 0.16961387893480587,
    "p": 0.21357467820006737,
    "f": 0.1844641328543158
  }
}
```

- ➢ My final strategy
    - Compare all the results above, my final strategy depends on the best performance, that is beam search (num_beams = 5).