



TARUMT

NLP- PARAPHRASING MACHINE

USING OPEN PRETRAINED TRANSFORMER

TEO SHI HAN



1 Introduction	2
1.1 The Benefit of Paraphrasing Tools	2
1.2 Approaches in developing paraphrasing machine	2
1.3 OPT as a solution	3
2 Research Background	4
2.1 Understanding OPT	4
2.1.1 Defining Open Pre-trained Transformer (OPT)	4
2.1.2 Understanding Language Model and Large Language Model (LLM)	4
2.2 Application of LLM in paraphrasing	5
2.3 Transfer learning methods in LLM	6
2.3.1 Soft Prompt	6
2.3.2 LoRA (Low-Rank Adaptation of Large Language Models)	7
2.3.3 Others	7
3 Methodology	8
3.1 Process management- Extreme programming methodology with MoSCoW	8
3.2 Soft Prompt Architecture	11
3.2.1 Data architecture	11
3.2.2 Model Architecture	14
3.3 Checkpointing	16
3.4 Callback system	17
3.5 Tools and libraries	19
4 Result	22
4.1 Training Curves	22
4.2 Evaluation	23
4.2.1 Result	23
4.2.2 Introduction to BART, BLEU and ROUGE score	23
5 Discussion and conclusion	24
5.1 System Limitation	24
5.2 Future improvements	24
5.3 Conclusion	26

1 Introduction

1.1 The Benefit of Paraphrasing Tools

Paraphrasing is a technique that is widely used and applied in industry. According to Jolly et al. (2020), the paraphrasing tool can be used to generate natural language and has potential to solve the problem of insufficient training data for Task Oriented Dialog System (TOD). The TOD is an emerging topic that a lot of researchers have proposed to solve various problems. It is used in online shopping to help customers in completing various purchase-related tasks, searching products and answering questions (Yan et al. 2017). The paraphrasing tool can provide data for TOD to train a better model. Apart from that, researchers also proposed that the paraphrasing tool can help in developing a paraphrase-driven question answering model (Fader et al. 2013), sentence fusion summarization model (Barzilay and McKeown, 2005) and improve the statistical machine translation using the Monolingually-Derived Paraphrases (Marton et al. 2009). In view of the benefit it can provide, developing a paraphrasing tool has become a good direction to work on. Hence, in this project, the problem identified is to develop a paraphrasing model to generate a paraphrased text from the source text.

1.2 Approaches in developing paraphrasing machine

There are several techniques used to create paraphrasing machines. According to Zhou, Jianing, and Suma (2021), there are 4 approaches when coming to developing the paraphrasing model. Such approaches included the Rule-based approach that required significant manual efforts to develop the paraphrasing rules and Thesaurus-Based approach that generated paraphrased text using substituting words with synonyms. The third method is the statistical machine translation, SMT-Based Approach that uses the machine translation mechanism to generate the Paraphrasing tool. The Fourth method is the neural network approach. It is tightly related to the Seq2Seq framework for paraphrasing. However, with the emerging trend and release of large language models and transformers architecture, a very new approach has been created. According to Palivela (2021), the paraphrase generation technique can be classified into two big categories, the Controlled Paraphrase Generation Methods and fine-tuning Large language models. The Controlled Paraphrase Generation Methods includes the LSTM that generation is controlled by the syntactic trees. Retriever-editor based approaches use the embedding distance of the source to predict the paraphrasing text. The second approach is trying to modify the weight of the pre-trained LLM for downstream tasks. Witteveen and Andrews (2019) has tried to fine tune the GPT2 large language model provided by OpenAI to perform the paraphrasing task that gains the average BLEU score of 0.5762 when tested on the Paraphraghs dataset. This is an application of the transfer learning to modify the existing model to perform various downstream tasks which paraphrasing is one of it.

1.3 OPT as a solution

The Rule-based approach and Thesaurus-Based approach is quite old and have underlying limitations in a sense, hence it is given a lower priority when it comes to deciding the development approach. It seems that the large language model approach appears to be a novel approach despite the fact that the neural network has held a strong position in the field. Hence the final decision is to develop a paraphrase machine using the large language model. The Open Pre-trained Transformer(OPT) released in May 2022 by Meta has been chosen for this task since it is a new release of the Large Language Model compared to others such as GPT2, GPT3 and BERT. There are only a few projects published to publicity hence the model still required a lot of exploration to test its usefulness. Hence, it is valuable and worthwhile for us to explore the way to develop a paraphrasing machine using OPT. In conclusion, the project proposed is to develop a paraphrasing tool using OPT through transfer learning techniques.

2 Research Background

The research background is divided into 3 sections, which include understanding what is an open pretrained transformer in detail, current researches and studies that focus on training the LLM to perform the paraphrasing task and transfer learning techniques that can be used to transfer the domain of the LLM into a paraphrasing purpose.

2.1 Understanding OPT

2.1.1 Defining Open Pre-trained Transformer (OPT)

Pre-trained Transformer (OPT) is actually a suite of **decoder only** pre-trained **transformers** ranging from **125M to 175B parameters**, which aimed to fully and responsibly share with interested researchers (Zhang et al., 2022). The transformers, or transformer model is actually a deep learning model that uses mechanisms of self-attention to enhance and was proposed in the provision of increased performance (less training time) compared to RNN because it can provide more parallelization (Vaswani et al., 2017). It is a concept proposed by Google research which is focused on the area of Natural Language Processing. The term decoder gives a hint about the architecture of the model. It used the architecture of “decoder-only Transformer models”, also known as 'auto-regressive models', which is designed to improve the zero-shot generalization, and eliminate the necessity of encoding by leveraging the attention mechanism (Golea, 2022). According to (Wang et al., 2022), it is proven that the causal decoder-only pretrained with full language modeling performs best if evaluated immediately after pretraining and can be used to produce excellent generative models and high performance models after **tuning** for other tasks (Wang et al., 2022). OPT is not the only member of “decoder-only Transformer models”, there are several popular models that belong under this category which are **CTRL, GPT, GPT-2, GPT-4** and Transformer XL. It is different from Encoder models like ALBERT, BERT, DistilBERT, ELECTRA and RoBERTa. The decoder-only architecture allows OPT to achieve good **Generalized Zero-Shot Learning**, meaning that when the model encounters unseen data, the stranger data, it still has the capability to classify it to a correct classification, or correct prediction, which is considerably a very challenging task (Pourpanah et al., 2022). According to Zhang et al. (2022), the OPT is a **Large language model (LLM)** which uses the pre-training Corpus originated from RoBERTa, The Pile and PushShift.io Reddit.

2.1.2 Understanding Language Model and Large Language Model (LLM)

Language model is the representation of probability distribution over the sequences of words (Jurafsky and Martin 2021). To understand it in a simple way, if a sequence of words is input into the language model, it can return us with the probability of the sequence. The language model is already proposed to be applied in 1990 to perform Speech Recognition using cache-based methodology (Kuhn and De Mori 1990). The basic idea of **automatic speech recognition (ASR)** is to match the voice, or acoustic input to words in vocabulary. The matched words are the most probable words corresponding to the voice. The language model can be used to improve the accuracy of the speech recognition by providing the probability of words based on the previous words that form the text sequence. According to Luong, Kayser and Manning (2015), deep neural language models can be used to improve **machine translation**. When the neural machine translation is being carried out, it needs a source sentence to maximize the conditional probability which can be presented in the conditional probability formula $p(\text{translation} \mid \text{source sentence})$. The $p(\text{translations})$ is provided by the language model and the $p(\text{source sentence} \mid \text{translation})$ is provided by the translation model using Bayes's rule

(Gulcehre et al., 2017). **Large Language Model (LLM)** refers to those language models that are large enough, which have billions of parameters. According to (Luitse and Denkena 2021), the emergence of the LLM trend has started by Google after it introduced the Transformer architecture in 2017 to make the training of enormous text corpus become feasible. To further understand the capabilities of LLM, it is worth understanding the basic knowledge of parameters in deep learning models. If the input layer has 4 neurons and the hidden layer has 2 neurons, then the parameter can be calculated by (4×2) weights + 2 bias which equals 10 total parameters. According to Zhang et al. (2022), it is stated that the OPT model has parameters ranging from 125 Millions to 175 Billions, hence we can imagine how large it is. There must be a very large number of neurons. According to Shahsavari et al. (2017), the increased number of neurons could increase the performance of the deep learning model but will cause the problem of overfitting. However, the Large Language model (LLM) has an unexpected phenomenon that it actually won't face the overfitting issue. Several hypotheses have been proposed to explain this situation. According to Nakkiran et al. (2021), it is suggested that the overfitting problem did exist if parameters increased at a certain level, however, if the parameter kept increasing, the issue of overfitting would reduce.

2.2 Application of LLM in paraphrasing

Training the LLM to perform the paraphrasing tasks is a topic that has been discussed for some time. Witteveen and Andrews (2019) have tried an attempt to fine tune the GPT-2 to perform the paraphrasing task. The datasets used included MSR Paraphrase Identification dataset with 4000 examples of original text and paraphrased text pairs, online news articles and paraphrase paragraphs from entertainment, news and food articles available online. The training technique used is called **Fine-Tuning** based on preprocessed supervised dataset. The data is formatted in the following format of "original text >>>> paraphrased text" where the ">>>>" token is used as a separator. The model is trained in small epochs to prevent overfitting on the dataset. The USE score and ROUGE-L score is used to evaluate the paraphrasing result. Hegde and Patil (2020) also tried to train GPT2 for paraphrasing using fine tuning techniques through the HuggingFace Transformers Library for sentence reconstruct tasks. The model was trained on a Quora Question Pair dataset with 140k actual examples. Palivela (2021) has done slightly different work compared to previous projects. The Text-To-Text Transfer Transformer (T5) model is fine-tuned into two approaches, which is paraphrase identification and paraphrase generation. The Unified system architecture is used to fine tune the model.

The hyper-parameter is also fine-tuned based on heuristic along the way of fine-tuning the model. The dataset used for the task are PARANMT-50M, Quora duplicate questions pair and Microsoft Paraphrase Research Database (MSRP). They claimed that the T5 model gives state-of-the-art results while at the same time preserving the flexibility to train on any downstream task. The T5 model used have 225M parameters, 12-layers and 2075 feed-forwards hidden states, 768-hidden layers, and 12-heads. The new OPT model hasn't had much research in fine-tuning it as a paraphrasing tool, hence, in this project our main objective is to fine tune the OPT model to produce a paraphrasing tool.

2.3 Transfer learning methods in LLM

2.3.1 Soft Prompt

There are many approaches to transfer learning that have potential to train the OPT model into the paraphrase machine. Lester, Al-Rfou and Constant (2021) suggested a method called soft prompts as a robust domain transfer method. In Soft Prompt methodology, the entire pretrained model is frozen, meaning that the weights of the entire model will not be modified. It only allows k numbers of tuneable tokens to be prepended to the input text as Illustrated in the diagram below.

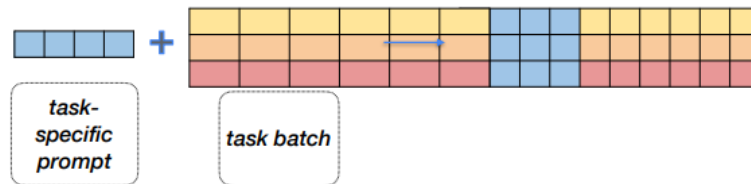


Figure 1 : Prepending the tuneable tokens into the input text (Iyyer, 2020)

To explain using mathematical annotation, let's denote the paraphrased text as Y , original text input as X and the tuneable tokens as T , then to predict a correct Y , the formula used is $\text{Pr}_{\theta}(Y | [P; X])$, where the θ is the model parameters (weights and bias). These tuneable tokens are the tokens embedding, where defined as $P = \{p_1, p_2, \dots, p_n\}$. The embedding is in the form of matrix, and can be concatenated with the input text that flows through the encoder-decoder architecture to produce the output text which in our case is the paraphrased text. The architecture of the soft prompt tuning can be illustrated in the diagram below.

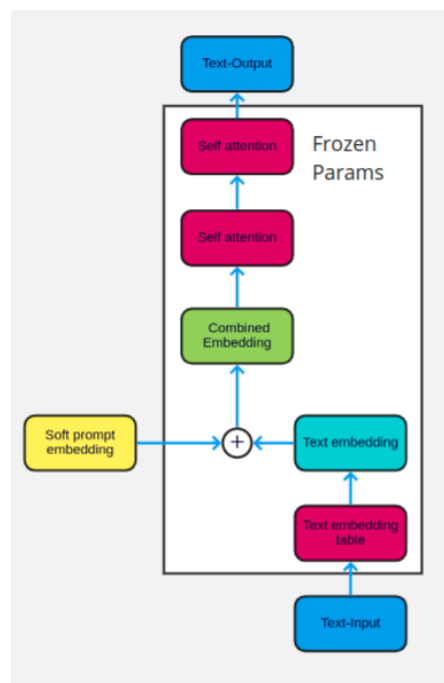


Figure 2 : The architecture of soft prompt tuning (Iyyer, 2020)

2.3.2 LoRA (Low-Rank Adaptation of Large Language Models)

LoRA was a transfer learning methodology proposed by Hu et al. (2021). The focus in the research is to reduce the number of trainable parameters of the large language model for downstream tasks, hence reducing the hardware requirements such as the GPU memory. The model is tested to have higher training throughput despite the fewer available parameters. There are 4 advantages when it comes to using the lora model. Firstly, the domain transfer is easy, by changing the A and B matrix in diagram below.

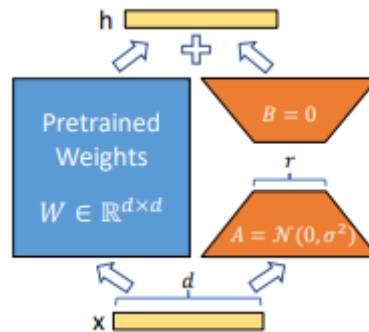


Figure 3 : LoRA reparametrization for the pretrained model (Hu et al. 2021)

Secondly, it can lower the hardware barrier by 3 times when using adaptive optimizers because only small low-rank metrics are needed to be optimized. Thirdly, it has no inference latency because the design is simple, just merging the trained matrix and the weight. Fourthly, it can be easily integrated with other methods such as prefix tuning. However, LoRA also has its limitation that when it comes to batch input different tasks in training, it is not that straightforward because there are many different A and B.

2.3.3 Others

Fine-Tuning (FT) is an approach that loads entire pre-trained weights and biases when doing the transfer learning. All the weights and biases will be altered based on input during the gradient descent. Some advanced Fine-Tuning may not train all the layers by freezing some layers. Li and Liang(2021) has done the Fine tuning on the GPT2 model for text generation with a freezing layer. Zaken, Ravfogel, and Goldberg (2021) suggested that train only the bias and freeze all the weight. The method is called **Bias-terms Fine-tuning (BitFit)**, however, the method does not seem to outperform the majority baseline. **Adapter tuning** was also proposed by Houlsby et al. (2019), injecting the adapter layers between the attention module, linking with the residual network.

3 Methodology

The method that is chosen by us to create a paraphrase machine is the Soft Prompt Tuning approach, using the opt model with 1.3 billion parameters. The model used is hosted by HuggingFace and the opt variation is OPTForCausalLM that is equivalent to an opt model with a causal language modeling head, that can be used to generate next text based on probability. The methodology is divided into 5 sections, describing the process management, soft prompt architecture, checkpointing methodology, callback system and introducing some tools and libraries.

3.1 Process management- Extreme programming methodology with MoSCoW

The software development model used is Personal Extreme programming, one of the variations for the agile software development methodology that is designed particularly for autonomous developers who work on software development individually. According to Marthasari, Suharso & Ardiansyah (2018), the MoSCoW technique can be implemented together with PXP in order to prioritize the more significant requirements. The procedure of extreme programming is illustrated in the diagram below.

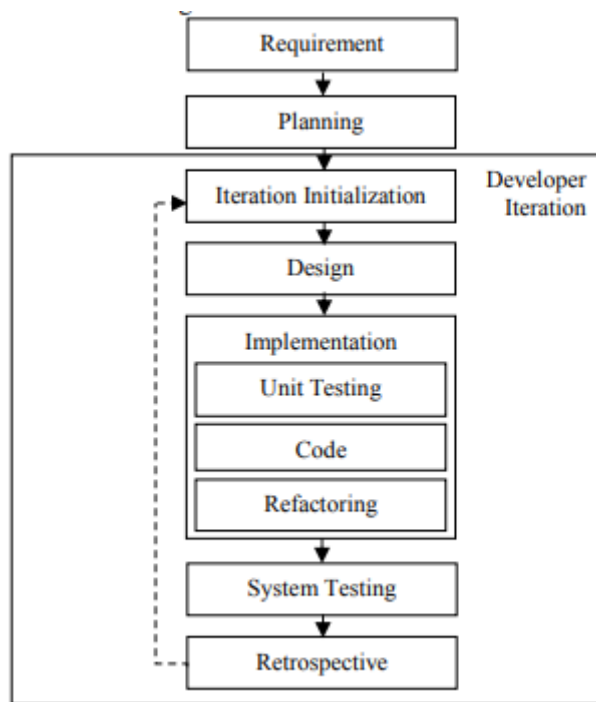


FIG 1. STAGES OF PXP METHODOLOGY [6].

Figure 4 : Extreme programming procedure (Marthasari, Suharso & Ardiansyah 2018)

Requirements

The requirements must be collected and converted into functional specifications. According to Cohn (2015), the requirements are gathered using the user story cards in which it follows the format of s "As [role] I can [action] so [purpose]".

Planning

According to Dzhurov, Krasteva & Ilieva (2009), the focus on the planning phase is to convert the requirements into a list of tasks. It usually involves estimation on time for each task. By incorporating the MosCoW methodology, the task can be categorized into three categories which is “Must Have”, “Should Have” and “Could Have” (Kravchenko, Bogdanova & Shevgunov 2022).

Iteration Initialization

The iteration can be initiated by selecting the tasks that will be focused in the iteration, and the iteration might lead to either the release-candidate, which means the software nearly can be released that might have bugs or a released version of the product.

Design

In the design phase, the tasks are modeling the system modules and classes that will be going to be used in the first iteration.

Implementation

The implementation involves unit testing that tests on each component of the software to ensure that it is working, doing code generation and code factoring. It must ensure that the code should compile without any errors and pass the unit successfully. Another criteria is that the design must fulfill the requirements specified before only allowed to be exited from the implementation phase.

System test

After the implementation is done, it must pass to the client for testing purposes. For each user story and requirements specified before must pass a user acceptance test or criteria.

Retrospective

It is a review of the performance of development. It includes the comparison between estimated time to complete the task and the actual time. If there is any delay, state the reasons to improve the future plan. The retrospective spot for space of improvement in project planning. The Retrospective could start another round of the iteration.

Requirements

1. As a developer I can load the data so that I can use it to train the model.
2. As a user I can transform the data so that it can be used when training the model.
3. As a developer I can combine multiple data sources easily so that handling of data is easier.
4. As a developer I can assemble the Model structure so that it can suit my need for paraphrasing.
5. As a developer I can save the Model regularly while training so that it won't lose if any failure occurs.
6. As a developer I can visualize the training progress to understand the training statistics.
7. As a developer I have generated the paraphrased version of text given input to fulfill the project objective.
8. As a developer I can compare the trained model with others so that we can get an idea about how our model performs compared to others.
9. As a developer I can visualize the evaluation matrix for better understanding.

Planning

	Use Stories	Priority	Point
Iteration 1	load the data into the platform	Must have	1
	Preprocess the data	Must have	1
	Combining the multiple data sources.	Should have	1
	Velocity		3
Iteration 2	Create the model architecture	Must have	1
	Training and checkpointing	Must have	1
	Visualizing the training matrix	Should have	1
	Velocity		3
Iteration 3	Generate paraphrase text using trained model	Must have	1
	Evaluate the model using evaluation matrix	Must have	1
	Visualize the evaluation matrix	Should have	1
	Velocity		3

3.2 Soft Prompt Architecture

In order to understand better about how the soft prompt works, the UML diagram is used to clarify the entire training architecture used when it comes to soft prompt.

3.2.1 Data architecture

The data used to train the model is organized using the LightningDataModule. The benefit of using the LightningDataModule is to allow us to combine the data from various sources, which in our case is the PARANMT 50M dataset and the PARABANK dataset. Since the PARANMT 50M and PARABANK dataset is a benchmark dataset, it reduces the effort needed to preprocess the data. The basic data preprocessing in this project is to formulate a prompt that follows the format of source </s> target and perform the text tokenization for embedding purposes that serve as input token. The DataCombiner class is used to combine the data streamed from the two data modules and perform data mixing using probabilities specification. The Detailed explanation of the attributes and methods was stated in the section below.

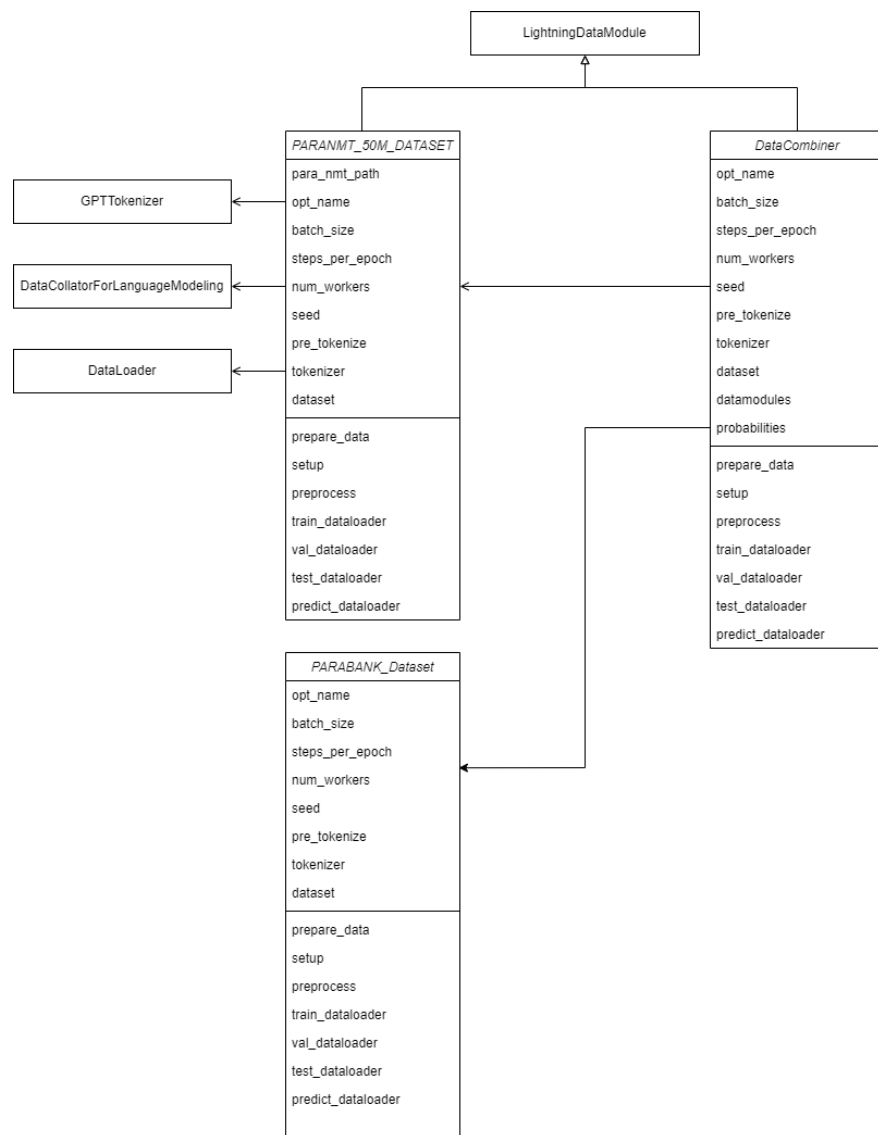


Figure 5 : Data Architecture for soft prompt tuning

Class : ParaNMT_50M_DATASET DataModule & PARABANK_Dataset

Attributes	Description
para_nmt_path	Use in load_dataset() function to get the dataset instance.
opt_name	To get the tokenizer model.
batch_size	Specify how many samples will be passed out for each iteration. This attribute is used in the DataLoader() constructor.
steps_per_epoch	To specify the number of batches required to be run to complete one epoch of training.
num_workers	It tells the pytorch DataLoader about how many sub-processes to use for data loading, related with parallelism.
seed	To be able to reproduce a similar result.
pre_tokenize	To specify whether the text needed to be tokenized.
tokenizer	Store the tokenizer instance created from HuggingFace transformers.
dataset	Store the dataset instance created by load_dataset() provided by the hugging face. This attribute is used in the DataLoader() constructor.
Methods	Description
prepare_data	Use for download tasks, for instance, download the tokenizer class for caching purpose, faster for later processing.
setup	Called after prepare_data. Data preprocessing can be done in this method.
preprocess	This function will be invoked to preprocess or transform the input batches into the shape of what we want.
train_dataloader	This method will be used when the training process is carried out, starting from the Trainer runs the fit() method.
val_dataloader	Generate the validation data loader, invoked when the trainer invokes fit() and validate() method.
test_dataloader	Invoked when the Trainer object invokes the test() method.
predict_dataloader	Invoked when the Trainer object invokes the predict() method.

Class : DataCombiner

Attributes	Description
datamodules	This is a list data structure that stores all the data module classes that originated from various sources.
dataset	This is to store the interleaved version of the dataset, loaded from the multiple data module.
probabilities	This is to define the proportion of the dataset from various sources to be used in forming a batch. The sum of the probabilities should be 1.
Methods	Description
setup	The setup instantiates all the data modules, and then put into the list data structure, and then interleaved and put into the dataset attributes.

3.2.2 Model Architecture

The diagram below illustrates how the model is assembled using the LightningModule and HuggingFace Transformer library.

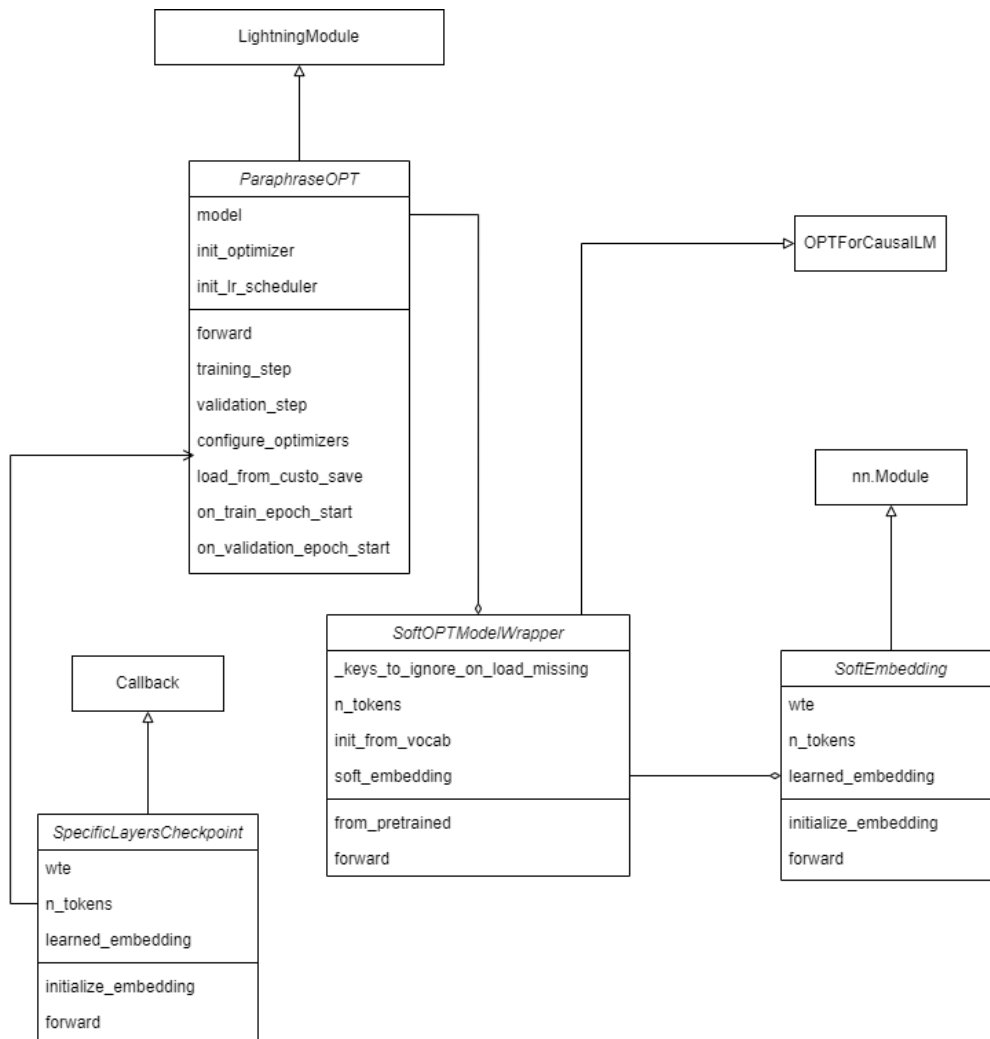


Figure 6 :Model Architecture for soft prompt tuning

Let's review the *SoftOPTModelWrapper*. This class is used to create a SoftEmbedding layer at the OPTForCausalModel which is not available in the original OPT provided by Huggingfae. We can imagine that the SoftOPTModelWrapper creates a new model architecture.

The *SoftEmbedding* class is used to create a SoftEmbedding layer proposed by Lester, Al-Rfou and Constant (2021). SoftEmbedding is actually the concatenation between token embedding and prompt that is used to create the decoder output. It inherited from the nn.Module so that it can be merged or combined with the all the neural network, including HuggingFace Transformers. It is also trainable as we can see that the forward function exists hence it is known as learnable token embedding.

SoftOPTModelWrapper is just used to add on the model, there must be a model used when training in action. For this purpose, we use the *ParaphraseOPT* for as a learnable model that includes validation step and optimization techniques.

The normal checkpoint system is just saving the entire training state, which is large. The ***SpecificLayersCheckpoint*** saves only the embedding layer when training, resulting in considerably smaller data size. The reason why it inherits the Callback class is because we can simply inject the saving action into the Trainer object. The SpecificLayersCheckpoint does checkpointing when the training epoch is completed.

3.3 Checkpointing

According to Lakshmanan (2019), the checkpointing design pattern in machine learning is an alternative way to training, evaluating and saving the model. A checkpoint is a record of the entire state of the trained model, including the weights, learning rate, bias etc. This makes it possible to resume from any checkpoint made for a model.

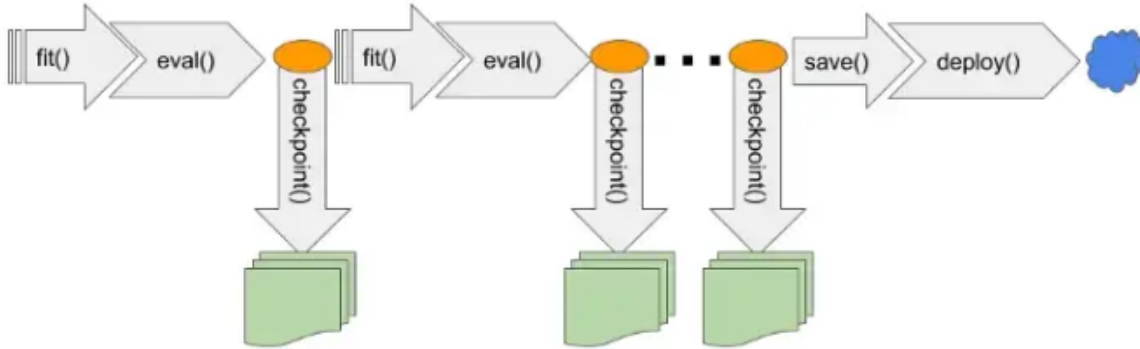


Figure 7 : Checkpointing in machine learning (Lakshmanan 2019).

The reason why checkpointing is good is because it provides resiliency by enabling us to restore to the previous successful state if there is any failure in the training process. It also can restore back if the model is overfitting. According to Eisenman et.al (2022), the Meta company used the Check-N-Run system that applied the incremental checkpointing to checkpoint the modified version of the model to solve the bandwidth issue. There are several conditions that must be fulfilled when doing the checkpointing which is accuracy, frequency, write bandwidth and storage capacity. In this project, the Pytorch checkpointing mechanism is applied through using the Model checkpoint class.

3.4 Callback system

According to Risto (2020), the callback means to inject code into the existing program. It is a way to pass the code into a function or class to invoke it when it is necessary. It can be used to improve the code base by enabling change of program working without interfering much on the source code written that will result in unmaintainable codebase.

There are reasons why callback is important in machine learning. Data Scientists tend to adjust if they find out that the model they trained does not perform well. The adjustment usually involves adding of the code. But it is difficult to know which change will give a better result. So, the callback comes to resolve this problem by enabling adding the new code without changing the source code. This provides an easy way to test and get insight from the outcome. The diagram below shows the classic training loop without callback mechanism.

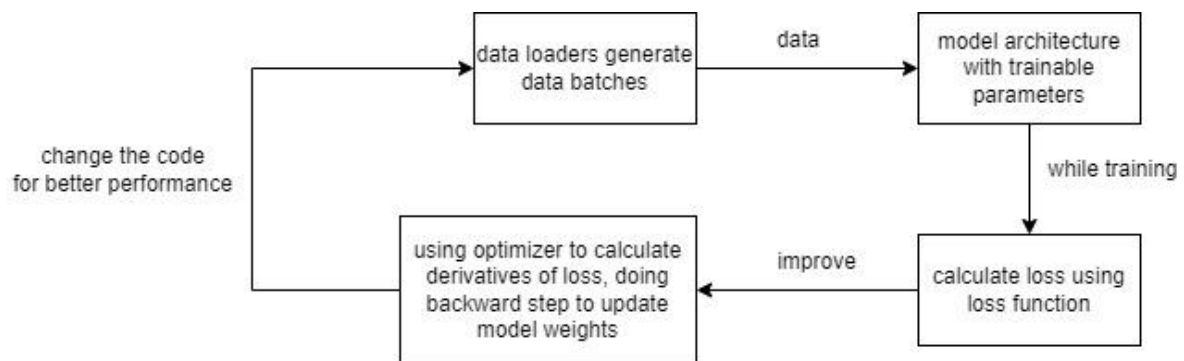


Figure 8 : The classic training loop

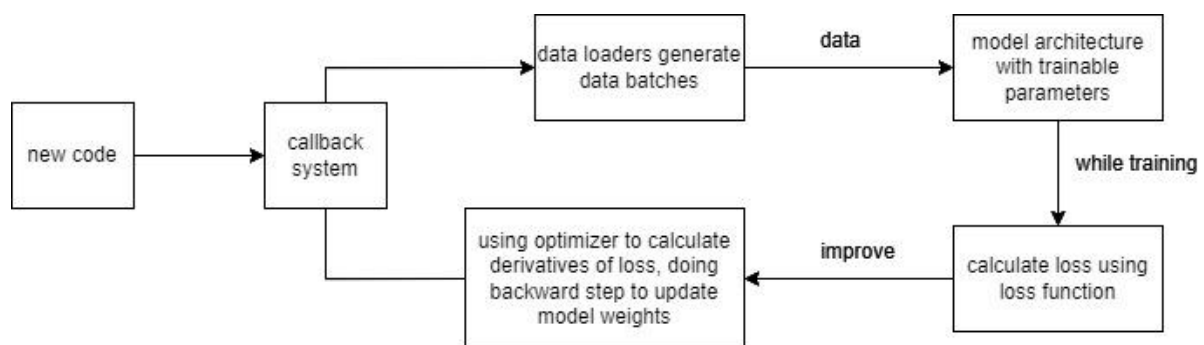


Figure 9 : The training loop implemented with the callback system

Pytorch lightning provides the callback class to support the callback system. The user can choose to add new code at different states of training. The table below illustrates some states that we can add to the code.

state	description
on_train_epoch_end	Called when the train epoch ends.
on_fit_start	Called when fit begins.
on_train_batch_start	Called when the train batch begins.
on_validation_epoch_end	Called when the val epoch ends.

3.5 Tools and libraries

Pytorch LightningDataModule

Pytorch data module is a class that wraps all the data processing steps which are implemented by us. The powerful advantage is the class is reusable, hence, can eliminate the redundant data processing step. The 5 processes that can be handled by the pytorch data module are illustrated in the diagram below.

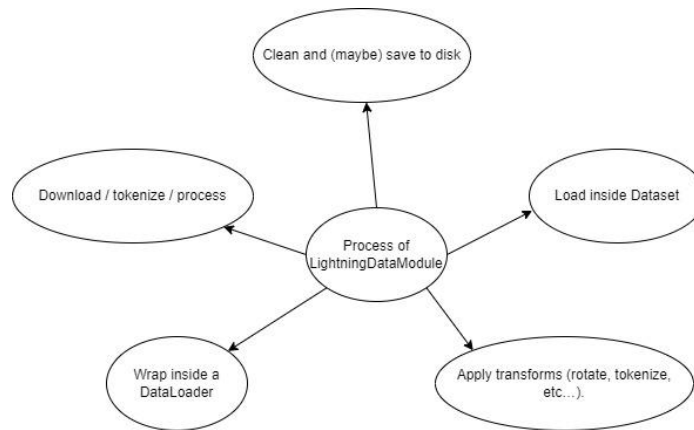


Figure 10 : The Process encapsulated by Pytorch LightningDataModule

The DataLoader and Dataset class would be discussed in the flowing sections. Pytorch LightningDataModule can specify the data splitting techniques, data transformation techniques, data normalization techniques and tokenization techniques. For simplicity, it combined the DataLoaders that are commonly used in machine learning tasks. Such data loaders involve DataLoaders for training, testing, predicting and validation. The specific methods and attributes applied in these tasks are illustrated in the diagram below.

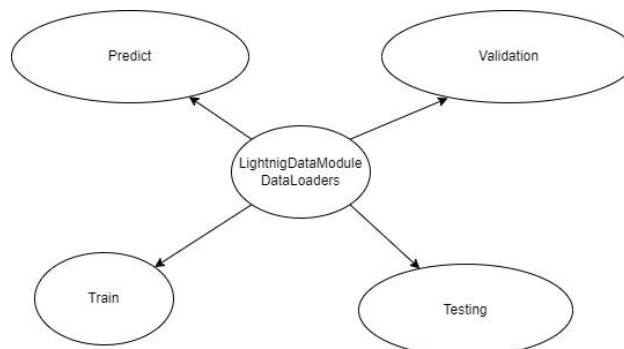


Figure 11 : DataLoaders purpose handled by Pytorch LightningDataModule

Pytorch Dataset and DataLoader Class

The Pytorch Dataset class provides a way to use the pre-loaded datasets, which are accessible remotely and openly. It also enables us to use our own dataset in the machine learning process. The DataLoader wraps an Iterable interface to the Dataset, so that it can be accessed using the looping to stream the data including loop forward and loop backward.

HuggingFace Datasets

The hugging face Datasets library is for ease of access and sharing of the datasets specialty for the nlp tasks. Hugging face also provides the data processing methods for quickly preparing the data for training in deep learning models. It is the way to download the datasets hosted on the HuggingFaceHub. It can be easily integrated to work with the pytorch library. If the data is too big, it also provides the data **streaming mode** that eliminates the need of loading the entire data by **streaming only part of the data** using the Iterable interface.

HuggingFace Tokenizer

Apart from splitting the string into subwords, the hugging face PretrainedTokenizer can do the job more than that. Such applications include BPE and SentencePiece, which add new tokens to the vocabulary. It also can manage the special tokens like beginning-of-sentence.

HuggingFace DataCollatorForLanguageModelling

The problem that DataCollator is trying to solve is combining multiple datasets, after that, performing a batch that contains the mixture of the data. Note that the data must have the same type to ensure proper functioning.

Optuna Library

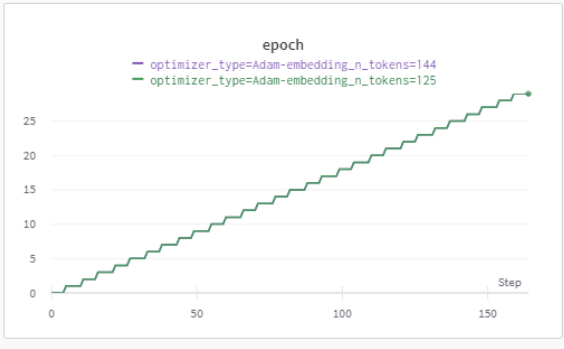
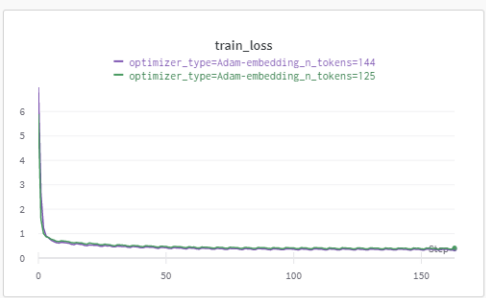
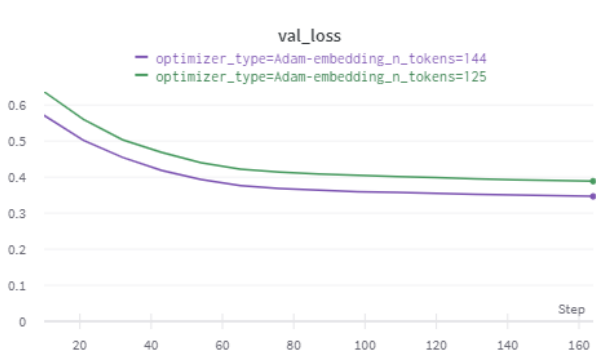
The Optuna is used to optimize the hyperparameter of the machine learning model automatically. It is one of the ways to find the best hyperparameter such as Grid Search, Bayesian, Random Search and Evolutionary Algorithms. According to Sion (2021), the Optuna is a lightweight and versatile library, which makes it simple to install when doing the machine learning task. Takuya Et.al (2019) also mentioned that the Optuna is designed specifically to allow users to construct the parameter search space dynamically. The pruning strategies of the Optuna is also designed to follow the principle of efficiency.

4 Result

The trained model using soft prompt by us has capability to paraphrase text, change the text without changing a meaning. The achievement is that it is successfully to transfer the domain of the model into the paraphrase task. Before doing training, the baseline OPT model will just continue our typed text and the length is dependent on our setting. However, after training, the model no longer continues to generate the text, and in the worst case, it just returns the original text.

4.1 Training Curves

The training progress was visualized in real time when training using the wandb.

 <p>Steps and Epoch</p>	<p>The visualization of the epoch and stem, it looks like the step because the epoch increases one for n number of steps. In our case, the total number of epochs is 30 and the step is 8000.</p>
 <p>Training loss</p>	<p>The loss means the model error. The diagram depicts that as the training progresses, the model error tends to decrease and the highest decrease happens at the beginning of the training. The training loss measures how well the model is fit to the training data. The diagram also illustrates the Adam optimization metrics.</p>
 <p>Validation loss</p>	<p>The model tends to fit to the new training data as the training is in progress, however, experiences stunted improvement after steps 60.</p>

The final model trained by us has an embedding token of 144 and loss value of 0.347.

4.2 Evaluation

4.2.1 Result

The evaluation is carried out by generating the 500 sample prediction from the model and the final product file contains two columns which are source and target. The prediction sample was saved in the csv file format. To calculate the BART score, ROUGE score and BLEU score, the Pytorch TorchMetrics library was used. The result obtained was illustrated in the diagram below.

bartscore	-1.766973
bleuscore	0.137793
rouge1_fmeasure	0.915261
rouge1_precision	0.879762
rouge1_recall	0.958095
rouge2_fmeasure	0.833050
rouge2_precision	0.801667
rouge2_recall	0.871429
rouge1_fmeasure	0.915261
rouge1_precision	0.879762
rouge1_recall	0.958095
rouge1sum_fmeasure	0.915261
rouge1sum_precision	0.879762
rouge1sum_recall	0.958095

Figure 12 : The evaluation result of the Soft Prompt Tuning Paraphrasing Model

The overall score is quite high indicating that there is no problem in the semantic structure of the paraphrased text and there is no diversion of the meaning after the text has been paraphrased.

4.2.2 Introduction to BART, BLEU and ROUGE score

The BARTScore is a evaluation matrix that was proposed by Liu et al. (2021) that is used to evaluate the generated Text as Text Generation. The higher BARTScore means the generated text is better. The paper proposed 7 perspective the BARTScore can used to inference, which is Informativeness (INFO), Relevance (REL), Fluency (FLU), Coherence (COH), Factuality (FAC), Semantic Coverage (COV), Adequacy (ADE). Since the log function is used, the number would be negative.

BLEU score, also known as Bilingual Evaluation Understudy Score is a metric that is originally designed to measure the performance of the translation model. It was proposed by Papineni, Roukos, Ward, and Zhu (2002). The BLEU score works by using the N-gram statistics. Two sets of N-gram will be generated, one from source and one from target. The value is formed by counting the number matching N-gram between the two sets of n-gram.

ROUGE score is a metric that is used to measure the similarity between two sentences, that is popularly used in the text summarization application. It is also known as Recall-Oriented Understudy for Gisting Evaluation. The ROUGE also uses the N-Gram as a basic source of statistics value. For example, to calculate the ROUGE recall, divide the overlapping between source text and target text n-gram with the total n-gram of the source text.

5 Discussion and conclusion

5.1 System Limitation

Weak paraphrasing capability

The first limitation identified in our system is that it seems to have difficulties generating the paraphrased text for some input especially without the keywords in the learned embedding token. Most of the time if the sentence without the tokens in learned embedding is inputted, the model will just return the original text, as illustrated in the diagram below.

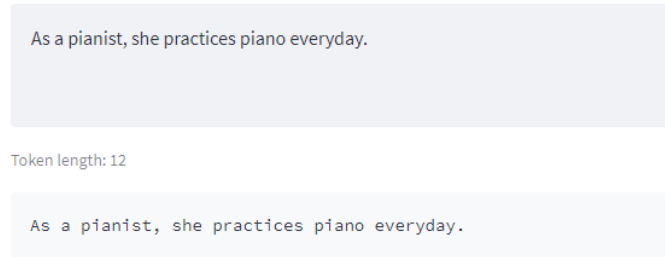


Figure 13 : The System failed to provide paraphrased text

Invalid token in word embedding

Apart from that, the learned embedding appears to have some tokens that contain the invalid characters such as eccentric characters, newline, punctuation and invalid characters. This might affect the prediction on the run time and might degrade the quality of the generated text. One major cause identified is the text corpus provided by the dataset is not clean enough hence there are unfavorable elements in there despite the dataset being a benchmark dataset. The 144 learned tokens in the embedding layer are listed in the section below.

['mur', 'ÛÛ', ' Mang', 'sha', ' Silver', 'ze', 'HI', 'aby', ' Nanto', ' forward', 'é', 'ub', 'g', 'ood', 'ty', 'acht', ';;;;', 'special', 'cool', ' ♦', ' RE', 'ial', 'town', 'ces', '"', 'se', ' board', '//////////', ' realistic', ' Licensed', ' prescribing', ' Rome', 'lig', ' proportion', ' open', ' endif', 'burgh', 'awa', 'Supporters', ' fro', ' Today', ' Public', ' Prom', ' Marshall', ' destroyer', ' m', 'eries', ' ', ' Sig', ' Par', 'itching', ' need', 'pers', ' U', ' missed', ' gathering', 'roy', 'Reviewer', ' seriously', 'igh', 'age', ' Berg', 'TA', 'enge', ' pick', ' mileage', ' he', ' play', ' sufficient', 'or', ' filmed', 'ne', ' key', 'stone', 'ova', 'elt', ' 1000', ' sol', ' Que', ' Blues', 'everal', ' fer', 'hi', 'iru', 'Grand', ' vide', 'ysc', ' Constitutional', 'handed', ' Talk', ' Wave', 'cia', ' Prin', ' Brom', 'land', ' swing', 'BS', ' kind', 'ter', 'ee', 'lesh', ' distinction', ' fig', ' environmentalists', 'usalem', 'alg', 'ère', ' bars', 'imir', 'uce', 'gha', 'ute', ' bank', 'des', ' train', ' Opening', ' working', ' interactive', ' no', ' managing', 'Bul', 'ists', ' ', 'bo', ' Bows', ' monthly', ' infl', 'aper', 'om', 'Force', ' prod', ' All', ' foot', ' clients', 'otor', 'eless', ' Kad', 'obs', ' north', ' preferred', ' domestic', ' code', ' doping', ' side']

Minimal changes

Sometimes the paraphrasing text generated by our model has done only minimal work such as replacing the word with synonym, and another part of the sentence remains unchanged. It can't change the syntactic structure of the sentence in some of the input.

5.2 Future improvements

Use a larger model

Diagram below illustrates the variation of the OPT model arranged according to the number of parameters. The 1.3B is the third smallest model according to the listing. Lester, Al-Rfou and Constant (2021) suggested that the soft prompt methodology could give better results if a larger model is in use. It is worthwhile to try training the 175B model or larger model to test the result, and

compare the difference with the smaller one if the hardware requirements are sufficient to cater the large model. Anyway, The 175B version of the OPT model required a request in order to make it accessible.

Model	#L	#H	d_{model}	LR	Batch
125M	12	12	768	$6.0e-4$	0.5M
350M	24	16	1024	$3.0e-4$	0.5M
1.3B	24	32	2048	$2.0e-4$	1M
2.7B	32	32	2560	$1.6e-4$	1M
6.7B	32	32	4096	$1.2e-4$	2M
13B	40	40	5120	$1.0e-4$	4M
30B	48	56	7168	$1.0e-4$	4M
66B	64	72	9216	$0.8e-4$	2M
175B	96	96	12288	$1.2e-4$	2M

Figure 14 : Variation of OPT model by parameter size

Improve word embedding quality

The learnable token embedding has included some unfavorable tokens. This might degrade the quality of the generated paraphrase text. So, in future, it is worthwhile to find a method to control the tokens that was included in the embedding space to eliminate the unfavorable tokens. Unanue, Borzeshi and Piccardi (2017) has tried an attempt to do such specialized word embedding by providing the medical dataset. Yildiz and Tezgider (2021) suggested that the word embedding quality can be improved through optimizing hyperparameters. Such accounted parameters that can be used to control the word embedding included word count, vector size, window size, negative sample, and iteration number. It is worthwhile to use these metrics to control the quality of the word embedding.

Add grammar and spelling correction capability

It is observed that to some extent, the model would try to generate “grammatically correct” sentences if the sentence with grammatical error is given as input. It could also identify the spelling error in the sentence and return a corrected version. For example, consider the diagram below. The model seems to naturally have this capability even though it was not trained specifically for this purpose, though the grammatical correction capability might not be strong enough. This gives an opportunity to create a model that has two capabilities, grammar checking and paraphrasing. Alikaniotis and Raheja (2019) have found that the Large Language Model like GPT2 and BERT is actually good at doing the grammatical correction task. The mixed functionality of the paraphrase and grammar correction might be a good field to be researched on.

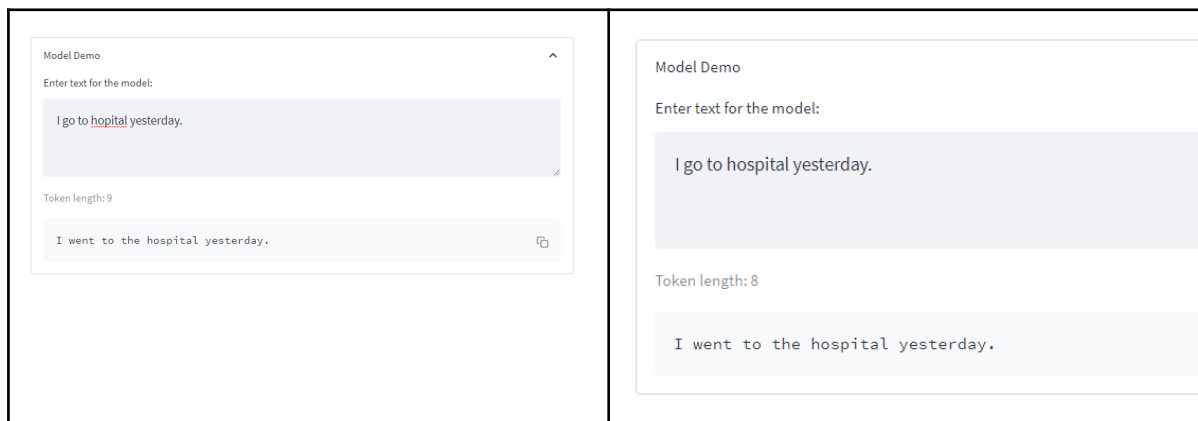


Figure 12 : The grammar correction and spelling correction capability of the model

5.3 Conclusion

In conclusion, this project at least fulfills the goal of creating a paraphrasing machine using OPT. The major problems of the system are weak paraphrasing capability, invalid token in word embedding and minimal changes applied to the source sentence. 3 future improvements and explorations have been identified which are improving the word embedding quality, adding the grammar and spelling correction capability as well as using a larger model to train the paraphrase tool.

Reference

1. Jolly, S., Falke, T., Tirkaz, C. and Sorokin, D., 2020, December. Data-efficient paraphrase generation to bootstrap intent classification and slot labeling for new features in task-oriented dialog systems. In *Proceedings of the 28th International Conference on Computational Linguistics: Industry Track* (pp. 10-20).
2. Yan, Z., Duan, N., Chen, P., Zhou, M., Zhou, J. and Li, Z., 2017, February. Building task-oriented dialogue systems for online shopping. In *Thirty-first AAAI conference on artificial intelligence*.
3. Liao, K., Liu, Q., Wei, Z., Peng, B., Chen, Q., Sun, W. and Huang, X., 2020. Task-oriented dialogue system for automatic disease diagnosis via hierarchical reinforcement learning.
4. Fader, A., Zettlemoyer, L. and Etzioni, O., 2013, August. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1608-1618).
5. Marton, Y., Callison-Burch, C. and Resnik, P., 2009, August. Improved statistical machine translation using monolingually-derived paraphrases. In *Proceedings of the 2009 conference on empirical methods in natural language processing* (pp. 381-390).
6. Zhou, Jianing, and Suma Bhat. "Paraphrase generation: A survey of the state of the art." *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021.
7. Witteveen, S. and Andrews, M., 2019. Paraphrasing with large language models. arXiv preprint arXiv:1911.09661.
8. You, K., Kou, Z., Long, M. and Wang, J., 2020. Co-tuning for transfer learning. *Advances in Neural Information Processing Systems*, 33, pp.17236-17246.
9. Bakker, M. A., Chadwick, M. J., Sheahan, H. R., Tessler, M. H., Campbell-Gillingham, L., Balaguer, J., ... & Summerfield, C. 2022. Fine-tuning language models to find agreement among humans with diverse preferences. arXiv preprint arXiv:2211.15006.
10. Le, H., Vial, L., Frej, J., Segonne, V., Coavoux, M., Lecouteux, B., Allauzen, A., Crabbé, B., Besacier, L. and Schwab, D., 2019. Flaubert: Unsupervised language model pre-training for french. arXiv preprint arXiv:1912.05372.
11. Hegde, C. and Patil, S., 2020. Unsupervised paraphrase generation using pre-trained language models. arXiv preprint arXiv:2006.05477.
12. Chen, Z., Yuan, H. and Ren, J., 2022. Zero-shot domain paraphrase with unaligned pre-trained language models. *Complex & Intelligent Systems*, pp.1-14.
13. Palivela, H., 2021. Optimization of paraphrase generation and identification using language models in natural language processing. *International Journal of Information Management Data Insights*, 1(2), p.100025.

14. Lester, B., Al-Rfou, R. and Constant, N., 2021. The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691.
15. Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P., Sridhar, A., Wang, T., Zettlemoyer, L. 2022. 'OPT: Open Pre-trained Transformer Language Models'.
16. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. 'Attention is all you need'. Advances in neural information processing systems, 30.
17. Goled, S., The rise of decoder-only Transformer models'.
18. Wang, T., Roberts, A., Hesslow, D., Scao, T.L., Chung, H.W., Beltagy, I., Launay, J. and Raffel, C., 2022. 'What Language Model Architecture and Pre Training Objective Work Best for Zero-Shot Generalization?'.
19. Pourpanah, F., Abdar, M., Luo, Y., Zhou, X., Wang, R., Lim, C.P., Wang, X.Z. and Wu, Q.J., 2022. 'A review of generalized zero-shot learning methods'. IEEE transactions on pattern analysis and machine intelligence.
20. Jurafsky, D., Martin, J.H. 2021. 'N-gram Language Models'. Speech and Language Processing (3rd ed.).
21. Kuhn, R., De Mori, R., 1990. 'A cache-based natural language model for speech recognition'. IEEE transactions on pattern analysis and machine intelligence, 12(6), pp.570-583.
22. Luong, M.T., Kayser, M. and Manning, C.D., 2015. 'Deep neural language models for machine translation'. In Proceedings of the Nineteenth Conference on Computational Natural Language Learning (pp. 305-309).
23. Gulcehre, C., Firat, O., Xu, K., Cho, K. and Bengio, Y., 2017. 'On integrating a language model into neural machine translation'. Computer Speech & Language, 45, pp.137-148.
24. Luitse, D. and Denkena, W., 2021. 'The great transformer: Examining the role of large language models in the political economy of AI. Big Data & Society', 8(2), p.20539517211047734.
25. Shahsavari, M., Boulet, P., Shahbahrani, A. and Hamdioui, S., 2017. 'December. Impact of increasing number of neurons on performance of neuromorphic architecture'. In 2017 19th International symposium on computer architecture and digital systems (CADSD) (pp. 1-6). IEEE.
26. Zhou, Z.H., 2021. 'Why over-parameterization of deep neural networks does not overfit?'. Science China Information Sciences, 64(1).

27. Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B. and Sutskever, I., 2021. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12), p.124003.
28. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. and Chen, W., 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
29. Li, X.L. and Liang, P., 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.
30. Zaken, E.B., Ravfogel, S. and Goldberg, Y., 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.
31. Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H. and Neubig, G., 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.
32. Papineni, K., Roukos, S., Ward, T. and Zhu, W.J., 2002, July. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics* (pp. 311-318).
33. Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019.
34. Optuna: A Next-generation Hyperparameter Optimization Framework. In *KDD*.
35. Lakshmanan, L 2017, *ML Design Pattern #2: Checkpoints*
36. Eisenman, A., Matam, K.K., Ingram, S., Mudigere, D., Krishnamoorthi, R., Nair, K., Smelyanskiy, M. and Annavaram, M., 2022. {Check-N-Run}: a Checkpointing System for Training Deep Learning Recommendation Models. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (pp. 929-943).
37. https://pytorch-lightning.readthedocs.io/en/stable/api/pytorch_lightning.callbacks.ModelCheckpoint.html
38. Lakshmanan, L 2017, *Callbacks in neural networks*.
39. Unanue, I.J., Borzeshi, E.Z. and Piccardi, M., 2017. Recurrent neural networks with specialized word embeddings for health-domain named-entity recognition. *Journal of biomedical informatics*, 76, pp.102-109.
40. Yildiz, B. and Tezgider, M., 2021. Improving word embedding quality with innovative automated approaches to hyperparameters. *Concurrency and Computation: Practice and Experience*, 33(18), p.e6091.

41. Alikaniotis, D. and Raheja, V., 2019. The unreasonable effectiveness of transformer language models in grammatical error correction. arXiv preprint arXiv:1906.01733.

