LLM Prompt Recovery Approaches: from Reinforcement Learning to LLM Finetuning

Lian Gan
University of California San Diego
lgan@ucsd.edu

Jessica Song
University of California San Diego
jwsong@ucsd.edu

Zhirui (Raymond) Xia
University of California San Diego
z4xia@ucsd.edu

Boqi (Bobby) Zhu
University of California San Diego
b2zhu@ucsd.edu

Abstract

In this paper, we explore the emerging field of testing and studying logic and explainability within large language models (LLM). To do so, We participate in a Kaggle competition named LLM Prompt Recovery [8]. In the challenge, participants are given an original text and a text that's rewritten by Gemma, Google's newly introduced family of open text generation models. The challenge is to recover the prompt that's used to rewrite the original text into the rewritten text. Because the competition doesn't provide the dataset, we tested with multiple existing dataset for this challenge and also generated our own. Moreover, we explore multiple avenues of models towards solving this problem, including building a prompting baseline model using GPT2, combining BERT encoder with GPT2 decoder to generate texts, and using reinforcement learning to guide the agent to produce prompts. We use the similarity metric that [8] specified, the novel Sharpened Cosine Similarity (SCS) metric, which applies a third-degree exponent to standard cosine similarity measures, to penalize loosely related responses. Top contestants in the competition tend to get a SCS of 0.65. Ultimately, we combined 4 datasets on Kaggle and generated our own prompts in order to generate a large and diverse training dataset for our purposes. We also tried four different approaches to this problem: reinforcement learning with PPO, an EncoderDecoder model, a BERT infilling model, and purely finetuning large language models such as Gemma and GPT-2. Although faced with a lot of constraints in terms of computational resources, we eventually arrived at a 0.56857 mean SCS on test set through finetuning Gemma. We introduce our data, all 4 appraoches models and their corresponding performance in further detail in the paper.

1 Introduction

The advent of LLMs has revolutionized various aspects of Natural Language Processing (NLP), from text generation to language translation and beyond. Among the myriad applications, text rewriting has emerged as a significant area of interest, wherein LLMs are prompted to reformulate text according to specified stylistic or content-based directives. Despite the broad adoption and success of LLMs in text rewriting tasks, the art of prompt engineering the process of designing prompts to elicit desired responses from LLMs remains an area ripe for exploration and optimization.

In this context, the LLM Prompt Recovery competition [8] serves as a novel arena for researchers and practitioners to delve into the intricacies of prompt engineering for text rewriting. The challenge at the heart of the competition is to reverse-engineer the prompts used by Gemm to rewrite a

Preprint. Under review.

set of texts. Through this exploration, we set to learn the intricacies of fine-tuning LLMs, examine LLM's ability to logically reason through context, experiment with combining pre-trained networks together, and contributes to the broader field of NLP by shedding light on effective prompting strategies.

2 Related Work

We did comprehensive research in order to find other models that have completed the task of recovering a rewriting prompt from LLM input and output; however, we were not able to find any as this problem is extremely specific and new. The only information on this particular task that we can find is the discussion posts of this Kaggle competition.

Therefore, we switched focus and grasped as much information as we can from the Kaggle competition discussion board. Through research, although we were not exposed to other people's model architectures, we were able to refer to others' training datasets.

This task lacks a designated dataset, necessitating the creation of our own. Nonetheless, the Kaggle discussion board features dozens of training datasets generated by other participants. Since we lacked the access to computational resources and time to generate large enough datasets, we used [3], [4], [5], and [1] in the generation of our dataset. The common methodology for creating such datasets begins with selecting random original texts, followed by generating prompts with tools like GPT-4, Llama-2, and PalLM 2. These prompts, along with the original texts, are then fed into another, more compact language model (LLM) like Gemma-7B, which is manageable on a single A100 GPU, to produce rewritten texts. The four datasets selected for this study all utilize GPT-4 for prompt generation and Gemma-7B for text rewriting. This approach ensures that the datasets exhibit high linguistic quality and coherence, leveraging GPT-4's adeptness at crafting contextually rich prompts. Gemma-7B, optimized for operation on an A100 GPU, further enriches this process by generating rewritten texts with nuanced variations, rendering these datasets essential for tasks demanding textual diversity and innovation. This strategy, which combines GPT-4's prompt generation with Gemma-7B's text rewriting capabilities, strikes an optimal balance between creativity and computational feasibility, forming a solid basis for our experimental work. Each dataset contains thousands of original and rewritten text pairs, encompassing a broad spectrum of linguistic structures, themes, and levels of complexity. Such diversity is vital for training models to comprehend and produce varied textual styles and contents, thereby improving their versatility and effectiveness in diverse applications.

3 Methods

3.1 Data

Beside using dataset posted by other users, we also generate our own datasets. The foundational dataset for our study originates from Meta Kaggle, which includes a diverse collection of text sources such as literature excerpts, online forum discussions, social media posts, and news articles. This dataset provides a wide range of linguistic styles and contexts, making it an ideal base for our text rewriting experiments.

To generate prompts for text rewriting, we utilized OpenAI's GPT-4, a cutting-edge language model. We began by providing GPT-4 with a set of example prompts to establish a baseline context. We then instructed GPT-4 to generate a broad array of prompts, with an emphasis on creativity and diversity. This method enabled us to quickly produce thousands of unique prompts, significantly expanding the potential for our text rewriting tasks. We categorize original text into three parts, literature, news article and forum posts. For each category, we generate different set of prompts. Thus we can avoid some situation that come rewrite prompt is not appropriate for specific text.

For the rewriting process, we employed Gemma-7B on local computer. Each piece of original text from the Meta Kaggle dataset was paired with a randomly selected prompt from our GPT-4 generated collection. The input for Gemma-7B was formatted into:"<start_of_turn>prompt:"'text"'<end_of_turn>Your response should always be in format:'Here's result:rewritten text'n<start_of_turn>model". By adding additional prompt helped standardize the output and avoid variations that could hint at the model's intervention during inference.

The final step involved compiling the results into a CSV file, organized into three columns: Original_text, Prompt, and Rewritten_text. This structured dataset encapsulates our comprehensive efforts in text rewriting, providing a solid foundation for further analysis and exploration in the field of natural language processing.

In the end we merged four dataset posted by other users and our own dataset. In some situation, the first sentence may include hint about prompt, such as "Sure, here is the rewritten text using Shakespeare's style. We tried our best to clean all first sentences that may provide extra information. After dropping some extremely short and long texts, and non-sense rewritten text, we finally get 8673 data.

3.2 Reinforcement Learning

Another attempt is to use Proximal Policy Optimization (PPO) to fine-tune a large language model to generate prompts related to the input and output [6]. PPO is a on-policy model free algorithm that uses a surrogate clipping objective function to ensure stable learning and efficiency. The pipeline starts with processing the original texts and rewritten texts into a single tokenized sequence, and then feed it into the language model, which in this case we selected GPT-2 as it is a small language model within our computational budget, and utilzied PPOTrainer from Huggingface to perform PPO update. The training loop starts with generating a response from the tokenized input, and use T5 sentence transformers to embed both the generated text and the target (the real rewrite prompt), and calculate the Sharpen Cosine Similarity. This score acts as the reward for the agent, and perform policy gradient based on this signal. The input sequence is formatted like this:

```
"{original_text}. Rewrite this narrative to: {rewritten_text}.
Provide a prompt for this task:"
```

3.3 EncoderDecoder

In this model, we set to use an encoder and a decoder together. Our primary idea is that we wish to encode the input and output, build an architecture to capture the difference between them, and translate that difference into human language with a decoder.

In order to do so, we utilize BERT's uncased base encoder to encode the inputs. To combine input and output into one string, we utilize the format

```
"[CLS] {original_text} [SEP]
Rewrite the previous text into {rewritten_text}
using this prompt:"
```

to input a combination of the original text and rewritten text into the encoder

For the training process, we implemented two different approaches. The first is the EncoderDecoder library from Huggingface. We combined BERT and GPT2 in an EncoderDecoderModel using the function `from_encoder_decoder_pretrained`, then used the model's own loss function to perform backpropagation. On the other hand, since we wanted to add our own structure, we also implemented a modified BERT model of our own. In this model, we initialize the decoder with a modified configuration to enable cross-attention with the encoder's outputs. Then, we assert that the hidden sizes of the encoder and decoder match to ensure compatibility.

In the network in between encoder and decoder, we add a couple of layers. Because the last maxpooling layer of BERT resulted in backpropagation issues (i.e. nonexistent gradient), we implemented a structure such that we get the hidden embeddings in BERT a before the last maxpooling layer. Therefore, the first layer in the in-between network is a adaptive max pooling layer with the the encoder's input size. Then, we add three fully connected linear layers in between, then pass the output to the encoder.

In the forward function, we handle both teacher-forcing and autoregressive generation modes, using the encoder's output as the cross-attention context for the decoder. To do so, the model directly uses these decoder input ids provided as the input to the decoder, along with a prepended start token. This approach ensures that during training, the decoder is exposed to the ground truth tokens at each timestep instead of its own predictions from the previous timestep, which is characteristic of teacher

forcing. Specifically, the start tokens are created and prepended to the decoder input ids to simulate the beginning of a new sequence, and then the decoder generates the output based on this sequence along with the encoder's hidden states. This method is used to help the model learn more efficiently by always providing the correct input at each step of the sequence generation.

The model dynamically generates sequences token by token when not using teacher forcing, with a provision to customize the maximum generation length, which we configured as 100 to generate a shorter prompt, aiming for a better cosine similarity.

3.4 Bert Model Text Infilling

Using the Bert model to do text filling for recovering prompts is another attempt we have tried. As an encoder model and able to understand the context from both ends, Bert is expected to understand the meaning of the original text well and capable of filling in missing text, which in our problem is the prompt. We used a bert-base-uncased tokenizer and model. Here is the format of input we fed into Bert:

```
f"{original} The task is to rewrite this narrative with the
given blanks: {masks} {rewrite}"
```

The number of masks in the input is determined by the number of words in our true prompts. It is a known number in the training set. The tokenizer would convert the input sentences above and prompt (our target) to IDs. After that, we initialized a new list that was the same length as the input IDs, and then we only assign the prompt's IDs back when its index is the same as the masks' index in the input IDs. In this case, we would have two ID lists. One is target IDs that have all -100 padding IDs except the element where it should be compared with predicted IDs. Another one is the input IDs, which have all input IDs except the middle part that needs to be inferred by the Bert Model. After getting these masks filled in, we embedded these predicted prompt and true prompt using t5, and use sharpen cosine similarity to calculate our loss to do our backpropagation. In the test part, since the true length of the prompt is unknown, we would use the distribution of the training prompt to randomly get a length for the predicted prompt.

3.5 Generative Pre-trained LLM Fine-tuning and Prompting

In this method we are using GPT-2 Large and Gemma-2B, which works as a baseline method. Both models are generative language model that uses deep learning to produce human-like text. This model has been widely recognized for its ability to generate coherent and contextually relevant text based on a given prompt.

The GPT-2 [2] Large architecture, developed by OpenAI, is a notable example of the Transformer model, which revolutionized natural language processing with its reliance on self-attention mechanisms. With 774 million parameters, GPT-2 Large excels at generating coherent and contextually relevant text by predicting the next word in a sequence, showcasing the power of deep learning in understanding and producing language. Gemma [7], by Google DeepMind, while based on the transformative principles of the Transformer architecture as well, introduces enhancements tailored to its two versions with 2 billion and 7 billion parameters. Key architectural innovations in Gemma include the implementation of Multi-Query Attention and RoPE (Rotary Position Embedding), which refine its attention mechanism for improved processing of long sequences. Furthermore, Gemma's training on a massive corpus of text, alongside sophisticated fine-tuning for tasks such as dialogue, instruction-following, and safety, represents a leap forward in creating versatile and responsible AI models. Both architectures embody the cutting-edge in AI research, with GPT-2 laying the groundwork for generative models and Gemma building on this foundation to push the boundaries of language model capabilities and applications.

Before finetuning the model, we reconstructed our dataset to include the original text, the rewritten text, and the corresponding prompts. Each entry in our dataset is formatted as follows:

```
Original text: [Original Text] <special] token> Rewritten text:
[Rewritten Text] <special token> The prompt used to generate rewritten
text given original text is: [Prompt]
```

where <special token> is a unique delimiter used to separate the different components of each data entry. This format is specifically designed to train the model to understand and generate rewritten text based on the original text and a given prompt.

Then we use Hugging Face Transformer Library to finetune GPT-2 and Gemma-2B. The training process uses cross entropy Loss. This loss function is suitable for the next-token prediction tasks that are typical of language models.

In the inferece phase, we feed

```
Original text: [Original Text] <special] token> Rewritten text:
[Rewritten Text] <special token> The prompt used to generate rewritten
text given original text is:
```

to the model. Here, we remove the prompt, so the model will predict the sentence, which should be prompt, after the special token.

4 Results

4.1 Result of Reinforcement Learning with PPO

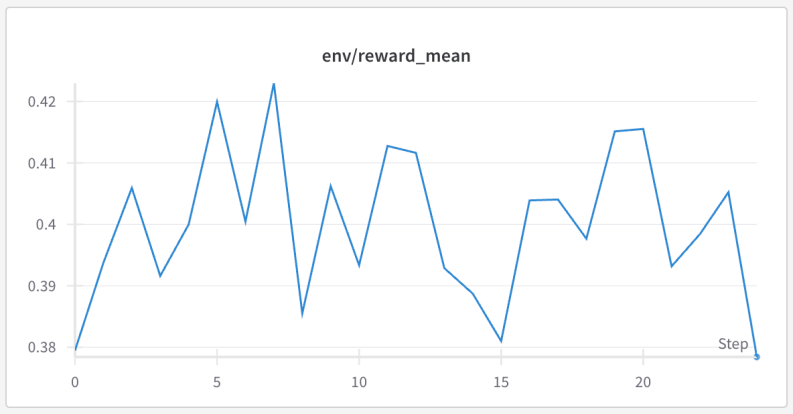

Figure 1: Average Reward VS. Steps

4.2 Result of Bert infilling Approach

Model	Train Size	Epoch	Average SCS
Bert-Base-Uncased	1000	1	0.457
Bert-Base-Uncased	2000	6	0.514
Bert-Base-Uncased	3000	15	0.512

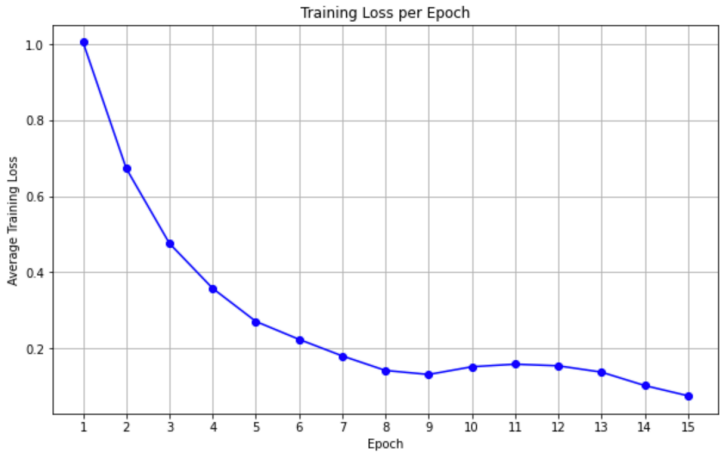

Figure 2: Bert Model: Epoches vs Loss

Some example generations include:

rewrite prompt	**target**	**score**
change it into an opening scene for a fantasy epic.	Translate the essence of this text into a horror story narrative.	0.4915778934955597
style it as a declaration of independence for a colony on mars.	Transform the text into a trivia quiz, testing knowledge on facts or details mentioned in the story.	0.4184628129005432
add drama to this text to amplify its impact :	Write it as a heartfelt reunion at a train station.	0.4298127889633178
make it a dialogue between two historical figures.	Write like Stephen King: Adopt King's knack for storytelling, blending the ordinary with the supernatural in a compelling narrative.	0.4816368520259857
adapt this text as a script for a noir detective in a sci - fi robot setting.	Create a palindrome version of the text, if possible.	0.5257990956306458

4.3 Result of Encoder-Decoder Approach

Model	Train Size	Epoch	Average SCS
Bert-Base-Uncased	3000	1	0.505
Bert-Base-Uncased	3000	3	0.522
Bert-Base-Uncased	3000	6	0.521

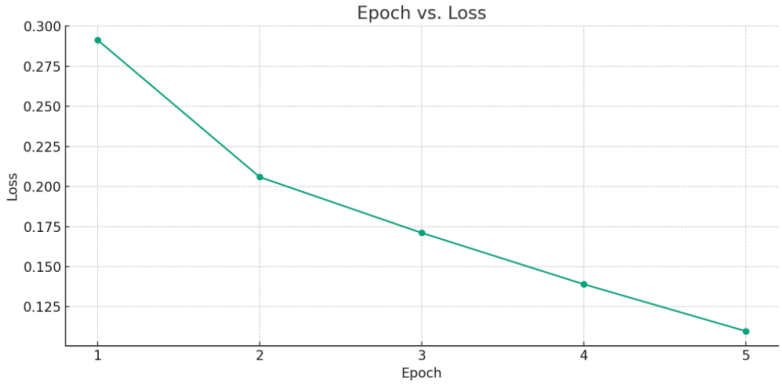

Figure 3: Encoder-Decoder: Epoches vs Loss

Some example generations include:

rewrite prompt	target	score
translate this information into an elevator pitch for a series of for a.	Transform this text into a captivating poem:	0.4908713102340698
rewrite it as a series of a series of post - to a series of a series	Turn it into a heartfelt plea for environmental conservation.	0.4120125770568847
rewrite it as a series of a series of post - to a series of a series	Recast it as a fantasy novel prophecy.	0.4743748605251312
adapt this text as a script for a scientist in a futuristic ai setting.	Adapt this text as a script for a noir detective in a Victorian gentleman setting.	0.628739058971405
rewrite this as a series of for a series of for a.	Rewrite it as a lively square dance caller's instructions.	0.4622969031333923

4.4 Result of LLM Fine-tuning and Prompting

Model	Train Size	Epoch	Average SCS
GPT-2 Large	1000	10	N/A
Gemma-2B	8000	1	0.43726459665102185
Gemma-2B	8000	5	0.5055176896574609
Gemma-2B	8000	10	0.46803958659396744
Gemma-2B	8000	15	0.485734378012063
Gemma-2B	1000	5	0.568576277086664

Table 1: Comparison of Model Performances Based on Training Size and Epochs

Above table shows overall performance, sharpen cosine similarity score on test set for models, with different training size and epochs. We decide leaving GPT-2 result blank, since GPT-2 gives completely irrelevant result. GPT-2 keeps writing its own story, which is not a prompt at all. The mean SCS reach peak after only trained for a few epochs on Gemme-2B. And when Gemma-2B is trained on a small train set, it gives best performance through our experiemnt.

4.4.1 GPT-2 Large

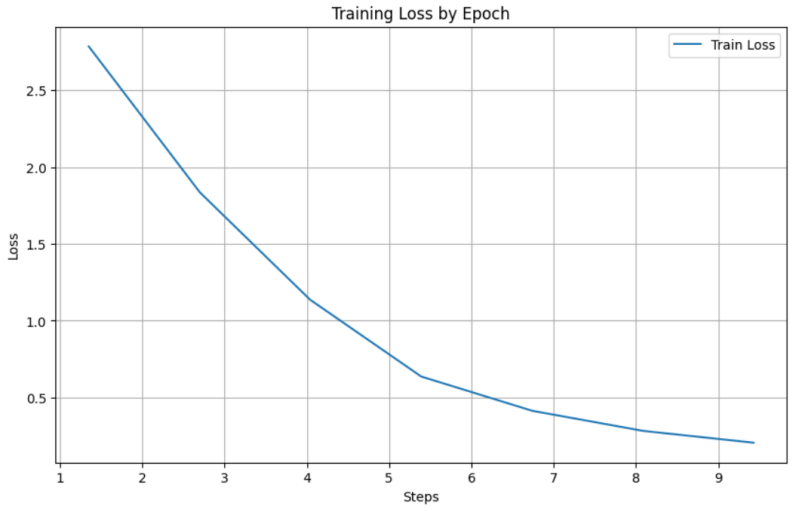

Figure 4: Training Loss of GPT-2 Large

The loss graph for GPT-2 while training shows a good trend for training loss. However, GPT-2 still doesn't know how to do the task. Here's one exmaple, when we feed an original text and a rewritten text in the style of a ancient Rome comedy into fine-tuned GPT-2.

```
True Prompt: Rewrite this text in the style of a ancient Rome comedy.

GPT-2 output:My Dear Dr.-Commander Smith:""My esteemed peers and comrades
here at Bael'Skithunse Radio Tower prepare your daily breakfast this afternoon"
```

So we decide to not focusing on fine-tuning GPT-2 anymore.

4.4.2 Gemma-2B

Figure 5: Training Loss by Steps

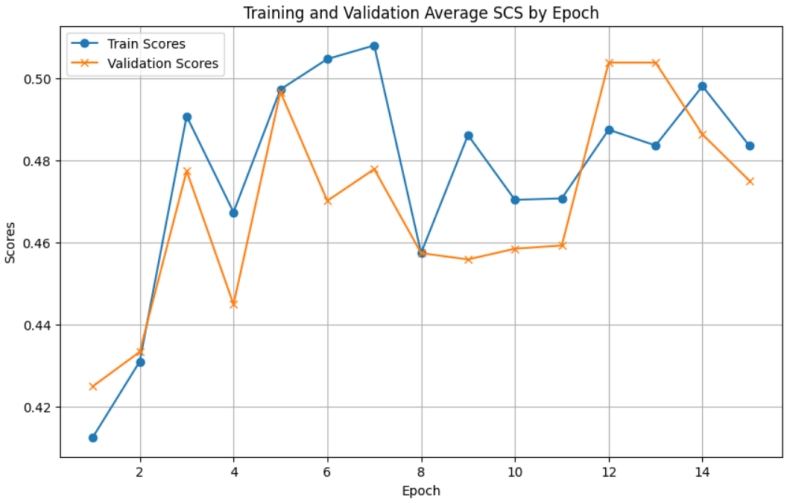

Figure 6: Training and Validation Average SCS VS. Epoch

For Gemma-2B, the training loss also shows a good trend, notice that 476 steps is 1 epoch, since we are using huggingface training library. The loss, cross-entropo loss, is calucalted on entire training text, which doesn't mat our requirement. We only care about the performance on <prompt> part. So we also test mean SCS while training. The mean SCS graph indicate that, our model start to learning

in first a few epochs. However, the model starts overfitting and learning content in original text and rewritten text. Below is some examples shows prediction by models.

```
True Prompt: Rewrite this text in the style of a ancient Rome comedy.

Gemma-2B without fine-tuning: "This script converts words into simple
sentences using NLTK's pretrained model (BERT-base-uncased)." (SCS:0.3903)

Gemma-2B fine-tuned for 1 epoch on whole train set: "1950s housewife: Craft
the essay with the detached, haus-proud style of a 1950s housewife, full of
picket fences." (SCS:0.3958)


Gemma-2B fine-tuned for 5 epoch on whole train set: "1920s Rome: Describe
the setting as if it were a historical epic tale." (SCS:0.6166)

Gemma-2B fine-tuned for 5 epoch on only 12% portion of train set: "Imagine
this text was a ancient Rome in the world of medieval knight, how would
it be written?" (SCS:0.6817)
```

5 Discussion

In the following section, we analyze our results for all four approaches separately and discuss potential improvements in the future.

5.1 Reinforcement Learning with PPO

From Figure 1, we can see that the performance of the model does not improve over time. Even though RL is a common approach for fine-tuning language models, it might not be suitable for this specific task, especially for small language models like GPT-2. The limitation is quite obvious. The reason for that is the improvement of the model heavily relies on the reward it is given at every steps. Despite the reward we gave matches the metric used for the competition, it fails to provide signals for generating a semantically related "prompts". With SCS as the only reward, the model tends to generate response that continues the flow of the rewritten text instead of generating a real prompt. Without a clear reward signal for generating a prompt, the model couldn't find a direction to improve, which leads to this poor result. For future explorations, we can improve this approach by first choosing a large language model with more tunable parameters such as Gemma-2b/7b. What we see from tuning Gemma directly is that larger language model tends to perform better than the smaller ones even with the same strategy. Anther improvement we can make will be choosing better data in general. The generation process of Transformers relies significantly on the preceding token, which is essential to the self-attention layers.

5.2 Bert Model Text Infilling

For the Bert infilling model, our generated result looks like real prompts. However, the generated prompts don't align with the expected prompt. We suspected there was overfitting in our training process, which cause model learns the details and noise in the training data to the extent that it negatively impacts the model's performance on new data. In the future, we might need to find a larger and more diverse dataset to train our model. Another thing we can do is to create augmented training examples specifically designed to improve infilling skills. By programmatically removing parts of sentences or paragraphs and training the model to predict the removed content. Another thing we can do is to replace our current Bert model with a large Bert model. We hope by doing these, our model can be more robust.

5.3 EncoderDecoder

In the EncoderDecoder model, our outputs also look like real prompts that makes no difference with the target prompts in terms of semantic structures and wordings. However, when compared with

the actual prompts, we again see a discrepency between the content described in the original target versus our generated results. However, in Figure 3, we can see that the loss is dropping well in our training process. In order to solve this problem, we tried a lot of different approaches, including training on different datasets, changing the loss between cross entropy loss, cosine embedding loss, and other losses that capture the semantic meanings. Moreover, we also tried increasing the number of training epochs and rows of training data. However, all of these approaches were to no avail.

Primarily, we believe that this method may be failing to learn because of the simplicity of the structure. We added the middle layers in an attempt to capture the semantic difference using a complicated neural network structure. However, in order to maintain a relatively original meaning in embeddings, and also due to our limitations in computational resources, we only added a couple of fully connected layers. In order to capture the semantic differences, we might have to implement a structure more closely to that of transformers. A couple of linear layers may only have added noise to our prediction instead of helping with extracting the difference between the input and output texts.

Moreover, a couple of potential improvements to this label include using a more sophisticated decoder such as Gemma-7B, training the network on a more diverse dataset, and training with more epochs given more computational resources.

5.4 LLM Fine-tuning and Prompting

While GPT-2 gives completely irrelevant result, Gemma-2B is able to predict some prompt relevant to true prompt. To account this phenomenon, Gemma incorporates architectural improvements and optimizations over GPT-2, such as Multi-Query Attention and RoPE (Rotary Position Embedding). These enhancements allow for more nuanced understanding and generation of text, enabling Gemma to maintain context over longer sequences and more effectively mirror the structure and style of the input prompts. We also have two important discovery. First, while training, the content in original texts and rewritten texts are also being trained, instead of just prompt. It results in continuously decreasing in training loss but mean SCS on validation set starts to fluctuate after a few epochs. Thus, next work in the future can be coming up with a method that train the model on prompt only given texts. Second, fine-tuned model gives better result on small training set. This may also indiciate the problem that model is overfitting on whole training texts. It may also causes by the quality of our training data set. Our train set is highly depending on automation by using other LLM. Some original text are not appropriate to be rewritten by some prompt, but they are still included in the train set. Thus, to make a better performance model, we need a higher quality of training set that supervised by human while generating such set. And it also tells us that training text is especially crucial on fine-tuning LLM.

5.5 Author's Contribution

Lian Gan: I generated training data by using Gemma-7B and GPT-4, and combined our data with 4 other datasets, constructing a diverse and large enough training dataset for the following approaches. I also fine-tuned GPT-2 Large and Gemma-2B to solve prompt recovery tasks.

Bobby Zhu: I implemented the Bert-infilling model while collaborating with Zhirui Xia. I also helped debug and train the encoder-decoder model and the Bert-infilling model with Zhirui and Jessica. Also, I documented results on the Bert infilling model, results, discussion, and presentation.

Zhirui Xia: I built the reinforcement learning approach. I also wrote baseline for the BERT infilling model, wrote training process for EncoderDecoder model with the Huggingface package, engineered the data loading process in both BERT Infilling and EncoderDecoder, and actively debugged and came up with new approaches to each problem as we work on each model in parallel.

Jessica Song: I built the EncoderDecoder model without using the Huggingface package, implementing a modified BERT model, structure of the EncoderDecoder model and the forward function. I also trained the EncoderDecoder model with different datasets and researched on losses that capture semantic meanings and experimented with them. I also coordinated between teammates and organized the presentation.

bibliography

References

[1] URL: https://www.kaggle.com/datasets/gali1eo/llm-recovery-pair?select=gemma_v1_7b-it.csv.

[2] Rewon Child David Luan Dario Amodei Ilya Sutskever Alec Radford Jeffrey Wu. "Language Models are Unsupervised Multitask Learners. Semantic". In: (2019). URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[3] Newton Baba. URL: https://www.kaggle.com/datasets/newtonbaba12345/llm-prompt-recovery-data-gemini-and-gemma/data?select=gemma_data_set_prompt_recover_1.csv.

[4] Newton Baba. URL: https://www.kaggle.com/datasets/newtonbaba12345/llm-prompt-recovery-data-gemini-and-gemma/data?select=gemma_data_set_prompt_recover_2.csv.

[5] Nicholas Broad. URL: https://www.kaggle.com/datasets/nbroad/gemma-rewrite-nbroad/data?select=nbroad-v2.csv.

[6] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[7] Gemma Team et al. *Gemma: Open Models Based on Gemini Research and Technology*. 2024. arXiv: 2403.08295 [cs.CL].

[8] Sohier Dane Ashley Chow Will Lifferth Paul Mooney. *LLM Prompt Recovery*. 2024. URL: https://kaggle.com/competitions/llm-prompt-recovery.