Query Rewriting

Sahar Khazali

1

1. Introduction

Conversational systems, including customer service bots, search engines, and virtual assistants, are integral to digital interactions. Their effectiveness depends largely on their ability to accurately understand and process user inputs. A key aspect of enhancing these systems is improving query rewriting—a process that not only corrects user input errors but also optimizes and refines queries to better capture the user's intent [2].

Query rewriting is crucial for managing common issues like typos, grammatical inconsistencies, and incomplete queries. By refining and sometimes expanding on the original inputs, query rewriting helps systems more accurately interpret and respond to users, improving overall system accuracy and user experience. This report outlines a systematic approach to selecting and deploying advanced models for query rewriting, aiming to elevate the performance of modern conversational interfaces. In the following sections, we will explore the challenges, data preparation, model evaluation, and practical benefits of the proposed solution [6].

2. Background

Natural language understanding presents significant challenges for computational systems, as human communication often involves slang, idioms, acronyms, and context-dependent expressions. These complexities make it difficult for conventional models to accurately decode user intent without advanced linguistic processing techniques. The field of query rewriting has evolved from simple rule-based systems to sophisticated machine learning frameworks, particularly with the advent of deep learning and Transformer-based models. These advancements have greatly enhanced systems' ability to understand and manipulate language, making them more adaptable and effective [4].

Modern query rewriting techniques, particularly those using AI models like T5, have set a new standard in managing ambiguities and implicit meanings in conversational queries. A key enhancement is the ability to process queries independently of their conversational context, improving system performance and scalability. However, the continuous evolution of language and the demand for real-time processing present ongoing challenges, necessitating further research and development to keep pace with emerging linguistic patterns and application needs [5].

2.1. Sequence-to-Sequence Tasks: The Role of Transformers

Sequence-to-sequence (Seq2Seq) tasks involve converting an input sequence into an output sequence, making them essential for applications like machine translation, text summarization, and conversational AI. Transformers have revolutionized this field by addressing limitations of earlier models like RNNs and LSTMs, particularly in handling long-range dependencies and parallel processing of data. Transformers introduced a self-attention mechanism that allows models to weigh the importance of different words in a sequence, significantly

improving performance in Seq2Seq tasks. Variations of transformers, such as BERT, GPT, and T5, have been tailored for specific tasks, with BERT excelling in understanding context, GPT in text generation, and T5 in versatile text-to-text tasks. A popular transformer-based model for Seq2Seq tasks is T5, which is designed to perform a range of text-to-text tasks, and BART, which uses both autoencoding and autoregressive properties to generate coherent and contextually appropriate text [1].

3. Data and Data Preprocessing

3.1. Data Overview

For the purpose of enhancing our query rewriting model, we utilize a dataset composed of pairs of original and disfluent queries. These pairs are stored in a JSON format, where each entry is uniquely identified by an ID and contains both an 'original' query, which represents the clean, intended user input, and a 'disfluent' version, which includes typical real-world noise such as repetitions, interruptions, or misstarts often seen in conversational speech. An example from the dataset is as follows:

'5a5918ff3e1742001a15cf7e':

'original': 'What do unstable isotope studies indicate?',

'disfluent': 'What do petrologists no what do unstable isotope studies indicate?'

3.2. Normalization

The text data undergoes normalization to standardize it. This includes converting all text to lowercase to reduce the complexity the model needs to handle and stripping out unnecessary punctuation that does not serve a functional purpose in understanding the query's intent.

3.3. Data Cleaning and Preprocessing

In the initial phase of data preprocessing, we focused on cleaning and refining the text data to ensure its suitability for subsequent modeling tasks. The first step involved the removal of emojis from the text. This was accomplished using the following function:

```
def remove_emojis(text):
    emoji_pattern = re.compile(
        "["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F700-\U0001F77F"  # alchemical symbols
        u"\U0001F780-\U0001F7FF"  # Geometric Shapes Extended
        u"\U0001F800-\U0001F8FF"  # Supplemental Arrows-C
        u"\U0001F900-\U0001F9FF"  # Supplemental Symbols and Pictographs
        u"\U0001FA00-\U0001FA6F"  # Chess Symbols
        u"\U0001FA70-\U0001FAFF"  # Symbols and Pictographs Extended-A
        u"\U00002702-\U000027B0"  # Dingbats
        u"\U000024C2-\U0001F251"
        "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

Subsequently, we proceeded to clean the text by removing any ASCII or punctuation marks, ensuring that the text was normalized and stripped of any non-alphanumeric characters. The following function was utilized for this purpose:

```
def clean_text(text):
    # Normalize to ASCII
    text = text.encode('ascii', 'ignore').decode('ascii')
    # Remove specific Unicode patterns
    text = re.sub(r'\\u[0-9A-Fa-f]{4}', '', text)
    # Remove non-alphanumeric characters (except spaces and punctuation)
    text = re.sub(r'[^a-zA-Z0-9\s.,!?]', '', text)
    return text
```

After cleaning the text, we identified and removed any rows where the text fields were empty. This was crucial to avoid processing errors during model training. The problematic rows were identified and filtered as follows:

```
problematic_rows = df[(df['disfluent'].str.strip() == '')

| (df['original'].str.strip() == '')]
```

To gain insights into the distribution of text lengths, we plotted the distribution of text lengths for both original and disfluent texts. This analysis helped us determine the average text length and identify the maximum length, which would be important for setting sequence lengths in later model training stages. During this analysis, we identified

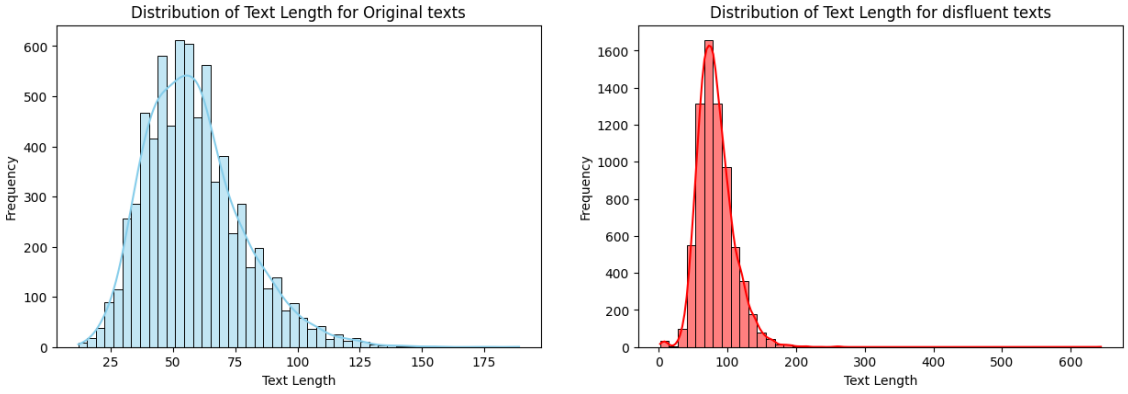

Figure 1. lext length of original and disfluent texts.

an outlier: a particularly long text that did not conform to the typical length distribution:

```
['QuestiontAnswer What is the dispensary subject to

in a majority of countries?...
```

Additionally, some texts were found to be filled with irrelevant content, such as repeated instances of the word "VALUE!". These were also removed from the dataset.

Next, we plotted various metrics such as absolute length difference (*abs_len_diff*), mean length, and longest substring ratio to further understand the dataset's characteristics.

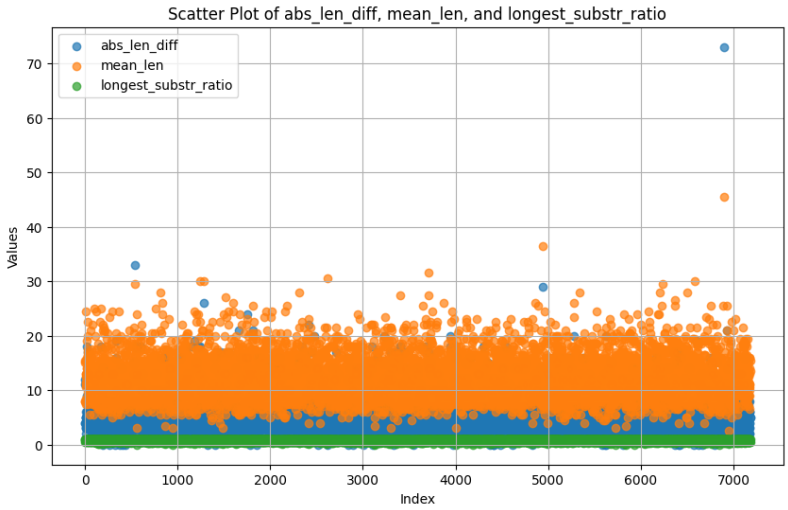

Figure 2. scatter plot for length ratio, mean and length difference of original and disfluent texts.

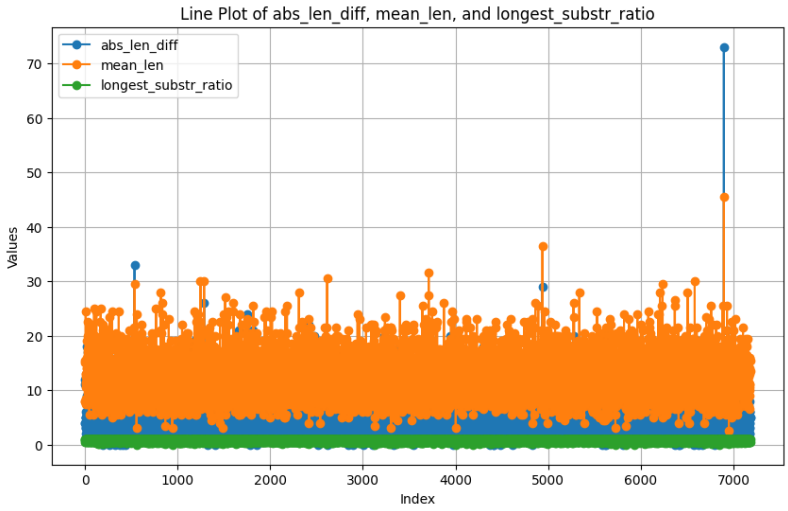

Figure 3. length ratio, mean and length difference of original and disfluent texts.

Following this, we computed and plotted the distance and cosine similarity between original and disfluent texts.

Upon reviewing texts with a cosine similarity of less than 0.1, we discovered a set of rows with nearly identical or meaningless content, such as:

```
["Question na", "question na", "no question", "No question", "No question.",
"Question NA", "na", "no question na"]
```

These rows were subsequently removed to enhance the quality of the data. The preprocessing steps described above laid the foundation for the subsequent stages of model development and evaluation.

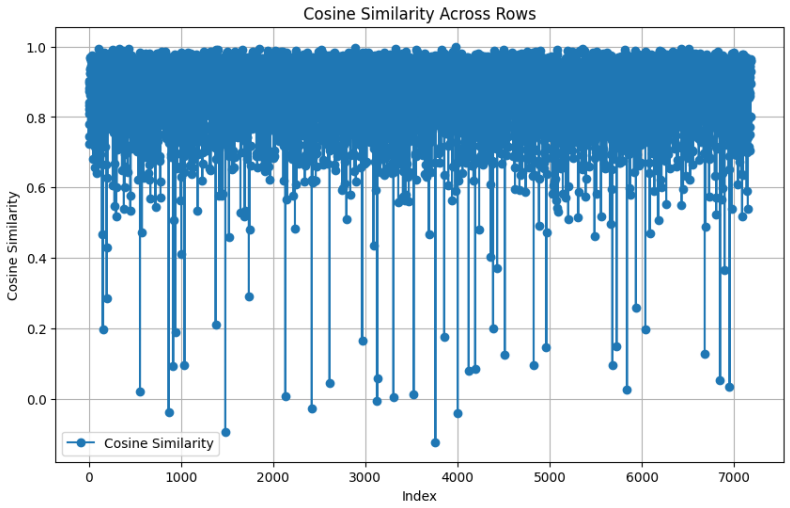

Figure 4. Cosine similarity between original and disfluent texts

4. Evaluation of Options

4.1. Candidate Models

To enhance query rewriting, we evaluated several top-performing large language models (LLMs). These include:

Pegasus: Specializing in abstractive summarization, Pegasus is adept at generating coherent summaries by masking and reconstructing key sentences, making it useful for refining disfluent queries.

T5 (Text-to-Text Transfer Transformer): T5's unified text-to-text format is ideal for transforming noisy queries into fluent ones, making it highly flexible for various NLP tasks.

Flan-T5: A T5 variant with instruction tuning, Flan-T5 excels at tasks with sparse examples, improving its adaptability to diverse query data.

BART: Known for text generation, BART's strengths in text correction align well with query rewriting objectives.

GPT-2: Although primarily a text generator, GPT-2's ability to create coherent text makes it suitable for improving incomplete or grammatically incorrect queries.

We used Optuna, an open-source framework, for hyperparameter tuning, ensuring optimal model performance tailored to our dataset.

4.2. Evaluation Metrics

To assess model performance in query rewriting, we used several metrics:

Accuracy: Evaluates how often the rewritten query matches the target query exactly, indicating precision.

BLEU: Measures how closely rewritten queries match the intended phrasing and structure of the original queries.

GLEU: A BLEU variant tailored for shorter texts, providing a more detailed evaluation of linguistic quality.

These metrics were applied to a development set to monitor performance, focusing on model confidence and avoiding overfitting.

4.3. Testing Environment

The models were tested in a high-performance computing setup with:

CUDA Version 12: Ensures compatibility with modern deep learning frameworks and efficient GPU utilization.

NVIDIA T4 GPUs with High RAM: These GPUs support large models and datasets, enabling faster computation and more efficient training.

5. Model Selection

After evaluating several large language models, Flan-T5 [3] was selected for our query rewriting task due to its superior performance in our evaluations. It outperformed models like Pegasus, T5, BART, and GPT-2 across metrics such as accuracy, BLEU, and GLEU, demonstrating its ability to effectively transform disfluent queries into fluent forms.

Flan-T5's flexible training configurations make it particularly suited to our needs. Key features include:

Fine-tuning with and without exemplars: Allows the model to generalize across various query styles by learning from both direct examples and task instructions. Chain-of-thought training: Enhances the model's ability to reason step-by-step, improving its handling of complex queries. Few-shot learning: Enables the model to perform well even with limited training data, making it adaptable to diverse and sparse real-world query examples.

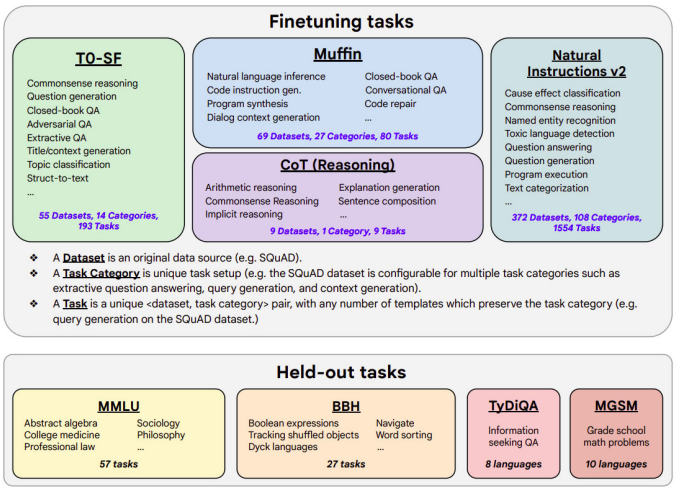

Figure 5. Examples of tasks performed using Flan-T5 for fine tunning

We used Hugging Face's TrainingArguments to optimize training speed and model performance. Data preparation involved a custom TextDataset class for tokenization, ensuring compatibility with Flan-T5. The model and tokenizer were initialized using the

pre-trained flan-t5-base configuration, which balances computational efficiency with task complexity. This setup is designed to maximize the effectiveness of our query rewriting system in both benchmark tests and real-world applications.

6. Train

For model training, we allocated 15% of our training data for evaluation, with the remaining 85% used for training. Key hyperparameters, including the number of epochs, batch size, weight decay, and learning rate, were optimized using Optuna to ensure the model's performance was fine-tuned to our specific task.

The training process was constrained by memory limitations, which influenced our choices, particularly in reducing batch size. Initially, we observed that the validation loss was higher than the training loss, indicating potential overfitting. To address this, we reduced the learning rate and increased the weight decay, which helped to stabilize training and improve generalization.

The training process was conducted using the following `TrainingArguments`:

```
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=3,              # number of training epochs
    per_device_train_batch_size=4,   # batch size for training
    per_device_eval_batch_size=4,    # batch size for evaluation
    warmup_steps=500,                # number of warmup steps for scheduler
    weight_decay=0.04,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    logging_steps=10,                # log every 10 steps
    evaluation_strategy="epoch",     # evaluate at the end of each epoch
    learning_rate=6.602358273531259e-05, # learning rate
)
```

After training, we compared the accuracy of various models, with Flan-T5 demonstrating the best results. The evaluation metrics for Flan-T5 after training are summarized as follows:

```
{'eval_loss': 0.17827652394771576,
 'eval_runtime': 28.4012,
 'eval_samples_per_second': 25.175,
 'eval_steps_per_second': 6.303,
 'epoch': 3.0}
```

These results indicate that Flan-T5 not only performed well during training but also maintained high accuracy and efficiency during evaluation, making it the optimal choice for our query rewriting task.

7. Result

The performance of our query rewriting model was evaluated using several metrics, including ROUGE and BLEU scores. Below are the key findings:

Metric	Precision (Low)	Precision (Mid)	Precision (High)	Recall (Low)	Recall (Mid)	Recall (High)	F-Measure (Low)	F-Measure (Mid)	F-Measure (High)
ROUGE-1	0.956385	0.963460	0.969847	0.948229	0.955644	0.962761	0.950216	0.957330	0.963771
ROUGE-2	0.920788	0.931283	0.940752	0.913335	0.923626	0.933575	0.914987	0.925091	0.934606
ROUGE-L	0.948259	0.956490	0.963346	0.940738	0.948531	0.956202	0.942557	0.950265	0.957448
ROUGE-Lsum	0.948895	0.956255	0.963541	0.940578	0.948554	0.956587	0.942819	0.950168	0.957832

Figure 6. Rouge performance

7.1. ROUGE Scores

The ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores provide a detailed assessment of the model's performance across different dimensions, including Precision, Recall, and F-Measure, at varying confidence levels (Low, Mid, and High). The ROUGE scores are as follows:

These results indicate that the model achieved high precision, recall, and F-measure across all ROUGE metrics, reflecting its strong ability to rewrite queries effectively while retaining the original meaning.

7.2. BLEU Score

The BLEU (Bilingual Evaluation Understudy) score for the model was 89.56, further demonstrating the model's capability to generate outputs that closely match the reference texts in terms of fluency and accuracy.

7.3. Overfitting

During training, we observed that the validation loss was higher than the training loss, indicating potential overfitting. To address this, we reduced the learning rate and increased the weight decay, which helped to stabilize training and improve generalization. These adjustments were crucial in achieving the high performance observed in the ROUGE and BLEU scores, ensuring that the model did not merely memorize the training data but learned to generalize well to new inputs.

7.4. Overconfidence

Overconfidence in a model can be inferred when there is a significant discrepancy between training and validation/test performance, where the model performs exceptionally well on training data but poorly on unseen data. In our evaluation, we found that the validation and test performance closely matched the training performance, without any significant drop-off. This consistency across datasets suggests that the model is not overconfident and has learned to generalize well beyond the training data, supporting the robustness of our query rewriting model.

References

[1] Sequence-to-sequence tasks: The role of transformers, 2024. Discussion on the impact of Transformers on Sequence-to-Sequence tasks, covering models such as BERT, GPT, T5, and BART in applications like machine translation, text summarization, and conversational AI.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam

McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. This paper discusses advancements in language models, including those that could be adapted for query rewriting.

[3] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Sebastien Jean, Ed H. Chi, Zoubin Ghahramani, and Quoc V. Le. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[4] Jianfeng Gao, Michel Galley, and Lihong Li. Neural approaches to conversational ai. *Foundations and Trends in Information Retrieval*, 2019.

[5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson, 3rd edition, 2019. This textbook provides foundational concepts in natural language processing and could be cited particularly for the theoretical background on conversational systems and query understanding.

[6] Gonçalo Raposo, Rui Ribeiro, Bruno Martins, and Luísa Coheur. Question rewriting? assessing its importance for conversational question answering. *INESC-ID, Instituto Superior Técnico, Universidade de Lisboa*, 2023. INESC-ID, Instituto Superior Técnico, Universidade de Lisboa.