Text Generation Using XLNet

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

Anand Swaroop

20BCE0218

Tanmay Mehrotra

20BCE2251

Under the guidance of

Prof. Rajeshkannan R
Associate Professor Sr,
School of Computer Science and Engineering,
VIT, Vellore.

May, 2024

DECLARATION

I hereby declare that the thesis entitled "Text Generation Using XLNet" submitted by me, for the award of the degree of *Bachelor of Technology to Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Rajeshkannan R.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date :

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "Text Generation Using XLNet" submitted by **Tanmay Mehrotra (20BCE2251)**, **Anand Swaroop (20BCE0218)** ,**School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Programme*, is a record of bonafide work carried out by him / her under my supervision during the period, 01. 12. 2023 to 30.04.2024, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : **Signature of the Guide**

Internal Examiner **External Examiner**

Head of Department
Bachelor of Technology

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to all those who have played a vital role in the successful completion of this research project. Our deepest appreciation goes to our esteemed guide, Prof Rajeshkannan R, whose invaluable guidance and unwavering support throughout this journey have been instrumental to our progress. We are incredibly fortunate to have had the opportunity to learn and work under his mentorship. We are also grateful to the Vellore Institute of Technology (Vellore) management for providing us with the necessary resources and infrastructure to conduct this research effectively. Their support has been a cornerstone in enabling us to reach this stage. Finally, we extend our heartfelt thanks to the participants of this study. Their willingness to contribute their time and share valuable information forms the very foundation of this research. We are committed to responsible research practices, and we would like to reiterate that all information and ideas from external sources have been meticulously acknowledged through proper citations and references.

Student Name

Executive Summary

This study examines the efficiency of the XLNet model for bidirectional text production in comparison to other well-known language models like GPT. Even though modern transformers like GPT models have shown impressive text generation skills, they are essentially unidirectional and can only produce text from left to right. In contrast, XLNet uses a generalised autoregressive pretraining technique that allows bidirectional text production by considering all possible combinations of the input sequence factorization order. The study aims to explore and quantify the advantages of bidirectional text generation facilitated by XLNet in combination with Top K Beam Bidirectional Text Generation along with its application in Question Answering System in terms of creativity, coherence, and overall performance, providing valuable insights into the comparative strengths and limitations of different state-of-the-art language model. This research investigates bidirectional text generation using the XLNet model and proposes a elongated Question Answering Model called BiGenXL and compares its efficiency with other prominent language models such as GPT,BERT. While current transformers like GPT models have demonstrated remarkable capabilities in text generation, they are inherently unidirectional, generating text only from left to right. XLNet, on the other hand, employs a generalized autoregressive pretraining method that considers all permutations of the input sequence factorization order, enabling bidirectional text generation.

Text generation is an important field in natural language processing (NLP), and transformer-based models such as GPT-1 and GPT-2 have made significant progress in this area. Nevertheless, due to their intrinsic unidirectionality, these models can only comprehend context in relation to tokens that came before them. By using a generalised autoregressive pretraining strategy, XLNet, on the other hand, introduces bidirectionality and enables a deeper comprehension of textual context. With a focus on bidirectional text creation and its consequences in applications like question answering systems, this paper compares and contrasts XLNet with GPT models. This research attempts to clarify the benefits and constraints of bidirectional text production enabled by XLNet through a series of tests assessing creativity, coherence, and overall performance, offering important insights into the terrain of state-of-the-art language models.

Table of Contents

List of Figures

List of Abbreviations

GPT	Generative Pre-trained Transformer
BERT	Bidirectional Encoder Representations from Transformers
XLNet	Generalized Autoregressive Pretraining for Language Understanding
QAS	Question Answering System
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
MCMC	Markov Chain Monte Carlo
GAN	Generative Adversarial Network
NLP	Natural Language Processing

1. INTRODUCTION :-

1.1.Motivation

Recent years have seen enormous progress in the field of natural language processing (NLP), especially in the domain of text production. By pushing the envelope, language models such as GPT-1 and GPT-2 are producing writing that is more and more human-like. Nevertheless, these models have a basic flaw: they can only generate text in one direction, from left to right. Their capacity to fully capture the context and richness of language may be limited by this unidirectional approach, which may have an adverse effect on coherence and creativity. This work explores a potentially useful substitute: bidirectional text generation with the XLNet paradigm. By using a special pretraining technique, XLNet examines every potential combination of the input sequence, enabling it to take into account data from words in the past as well as those in the future. This innate reciprocity has the capacity to surpass the limitations of unidirectional models, leading to more comprehensive and nuanced text generation.

1.2. Objective

In this study, we evaluate the effectiveness of XLNet's bidirectional text creation with the highly regarded unidirectional models, GPT-1 and GPT-2, in order to gain a clearer understanding of its potential. We quantify the benefits of bidirectionality in terms of originality, coherence, and overall performance through a thorough analysis. Furthermore, we investigate the use of XLNet with top K Beam Bidirectional Text Generation in Q&A systems, evaluating its capacity to comprehend and produce answers that take contextual information into account. In order to develop natural language generation and interpretation, we hope that our research will shed light on the relative advantages and disadvantages of various cutting-edge language models.

1.3. Background

Before diving into the proposed architecture model **BiGenXL** we will give brief introduction to Transformer architecture.The Transformer architecture follows an encoder-decoder structure. The encoder extracts information from an input sentence, and the decoder uses those features to generate an output sentence.The encoder in the transformer is made up of numerous encoder blocks. An input sentence is sent through the encoder blocks, and the output of the final encoder block is used to feed the decoder's input features.

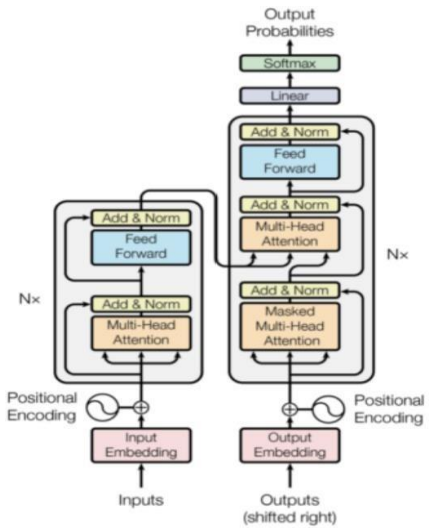

Fig.1 Transformer Architecture

Each transformer block contains several self-control layers, feedforward layers and residual connections. The parts in encoder block are-

1.3.1. Self-Attention Mechanism: The encoder block uses the self-attention mechanism to enrich each token (embedding vector) with contextual information from the whole sentence.Depending on the surrounding tokens, each token may have more than one semantic and/or function. Hence, the self-attention mechanism employs multiple heads (eight parallel attention calculations) so that the model can tap into different embedding subspaces.

1.3.2. Position-wise Feed Forward Network: This is a fully-connected neural network with two layers (ReLU activation in the first layer and linear activation in the second). It helps the model learn non-linear relationships between the tokens and further refines the representation. The XLNet model is made up of several layers of decoder block.The parts in the decoder block are-

1.3.3.Masked Multi-Head Attention: Masked multi-head attention is a key component of transformer models,particularly used in the decoder block. It allows the model to focus on relevant parts of the input sequence when generating the output, preventing it from "cheating" by looking at future tokens.In the decoder block of a transformer, masked multi-head attention takes three inputs

- Query (Q): Representation of the current word being generated.
- Key (K): Representations of all words in the input sequence.
- Value (V): Same as K, but used to provide information about relevant words.

Working

- **Attention scores:** Each head in the multi-head layer calculates scores for each word in the input sequence, indicating how relevant it is to the current word based on Q, K, and V.
- **Scaled Dot-Product Attention:** Each head independently calculates attention weights using the scaled dot-product attention mechanism. This measures the similarity between the query of the current element and the keys of other elements in the sequence.
- **Masking:** Scores for future words are set to negative infinity, essentially ignoring them.
- **Weighted sum:** Scores are normalized and used to weight the values (information) from all input words. This creates a single "context vector" relevant to the current word.
- **Concatenation and Projection:** The outputs from all heads are concatenated and projected back to the original dimension, creating a combined representation that incorporates information from different perspectives.
- **Repeat for all words:** This process is repeated for each word in the Output sequence,one by one.

1.3.4.Add and Norm: In the decoder portion of a Transformer model (commonly used for machine translation and text generation tasks), "add and norm" refers to a specific combination of two techniques applied consecutively:

1.3.4.1. Residual Connection (Add): This technique adds the output of a layer to its original input before feeding it into the next layer. This allows the network to learn from the identity function as well as the computations performed by the layer. It helps address the vanishing gradient problem, which can hinder training deep neural networks.

1.3.4.2. Layer Normalization (Norm): This technique normalizes the activations of a layer's outputs across its features (dimensions) before feeding them into the next layer. This helps stabilize the training process and improve the model's performance. Unlike batch normalization,layer normalization operates on individual training examples rather than the entire mini-batch.

1.3.5.Feed Forward Network: Feed-forward layer is a multi-layered structure where information flows in one direction, from input to output, without any loops or back-propagation.

2. PROJECT DESCRIPTION AND GOALS

2.1. Survey on Existing System

We've read through various research papers and other pieces that are pertinent to our work in this part. The main and the base paper of our project is the XLNet: Generalized Autoregressive Pretraining for Language Understanding[1] which proposed the XLNet model as a generalized autoregressive pretraining method that enables learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorization order.The main objective of the paper was to illustrate Permutation Language Modeling as compared to other AR Language modelling.[4] This research proposes a new text generation model that combines traditional templates with modern encoder-decoder architectures. It takes content from the input text and retrieves similar examples ("soft templates") from training data. These templates then guide the model on how to express the content in the generated text. The model performs well in tasks like text summarization and data-to-text generation, achieving better results than existing methods.[6] . The research focus is on recurrent neural networks (RNNs) and a specific type called Long Short-Term Memory (LSTM) networks. LSTMs are seen as an improvement over traditional RNNs for tasks like sentiment analysis, machine translation, and text generation. The paper discusses the architecture and workings of LSTMs, highlighting their ability to generate text from random input. LSTM solves the issue of “vanishing gradient” that is faced in the case of standard recurrent neural network (RNN).[2] This research tackles generating sentences with specific keywords (lexically constrained sentences). Traditionally, this involved random editing of candidate sentences, leading to nonsensical outputs. The authors proposed a two-step solution: (a) Prediction: A classifier trained on synthetic data predicts how to refine a candidate sentence to meet the constraints. (b) Revision: MCMC sampling revises the sentence based on the predicted areas for improvement. This method achieves better fluency, diversity, and adherence to constraints compared to previous random editing approaches.[11] This research introduces a Sepedi text generation model using a Transformer-based approach. This model is computationally less expensive than other transformer models and achieves high accuracy (75%) in reconstructing unseen text data. The model uses a single Transformer block with causal masking and separate embedding layers. The authors trained the model on a Sepedi text corpus and varied hyperparameters like embedding size and batch size to achieve the best

performance.[12] This paper reviews recent advancements in neural text generation models. It starts by highlighting limitations of traditional recurrent neural network models for this task. Then, it explores new methods like reinforcement learning, re-parameterization tricks, and Generative Adversarial Networks (GANs) used in text generation. The paper compares these methods' properties and how they address common challenges like vanishing gradients and generating diverse text. Finally, it presents a benchmark experiment comparing different models on public datasets and discusses the results in relation to the models' properties.[13] This study aims to comprehensively explore text generation by examining existing techniques (both traditional and deep learning-based), the metrics used to assess their performance, and the methods for evaluating the quality of the generated text itself. Additionally, it will investigate the growing applications of text generation and identify the key challenges and promising future directions in this field.[14] This paper proposes a new data-to-text generation model that improves diversity and reduces repetition. It uses a Transformer to plan the content structure and a Wasserstein Auto-Encoder to capture the data's essence. Text generation considers this plan, the captured essence, and the context of already generated sentences. Finally, a unique decoder helps achieve more diverse expressions in the generated text, leading to better results on diversity metrics compared to existing models.[15] This research introduces a new Transformer-based model for data-to-text generation tasks, specifically converting database records into descriptive text summaries. The model improves upon existing methods in two key ways: (a) Content Selection and Summarization: It tackles both content selection (deciding which information to include) and summary generation (turning chosen information into text) in one go, unlike previous models. (b) Improved Content Accuracy: The model incorporates two extensions to enhance the quality of generated summaries:Record Embedding: This refines how the model represents input data, leading to more accurate content in the summaries.Content Selection Modeling: This trains the model to identify important information within the records, ensuring relevant details are included in the summaries.[16] This research paper reviews the state-of-the-art in Machine Translation (MT) using Transformers, specifically focusing on transformers' application to Indian languages. The paper highlights the need for further research in MT for under-resourced Indian languages. It analyzes the two main approaches in NMT (model-based and data-driven) and explores the three key components of NMT models: building the model, output inference, and training parameters. Overall, the paper aims to provide researchers with a foundation for exploring transformer-based MT for Indian languages.[17] This research paper provides a comprehensive review of Transformer-based (TB) models used in Natural Language Processing (NLP) tasks. It delves into the core concepts behind their success, particularly their

self-attention mechanisms for handling long-range dependencies in text. The survey categorizes TB models based on their architecture and training methods. It then compares popular techniques in terms of design choices and their effectiveness in NLP tasks. Finally, the paper explores open research areas and future directions to address current challenges in applying TB models for NLP applications.[18] This research addresses the challenge of generating fluent text descriptions from structured table data. A common approach involves copying words directly from the table. However, this method struggles with accuracy, particularly for words not present in the model's vocabulary. To overcome this limitation, the authors propose a new Transformer-based framework that combines copying mechanisms with traditional language modeling. Their approach incorporates two key elements: Word Transformation: This method injects field and position information from the table into the generated text, indicating where copying from the table is most relevant. Auxiliary Learning Objectives: Two additional objectives are introduced during training: (a) Table-Text Constraint Loss: Enforces a strong relationship between the table structure and the generated text. (b) Copy Loss: Improves the model's ability to accurately copy word fragments from the table. The authors also refine the text search strategy to minimize repetitive and nonsensical sentences. Evaluations on benchmark datasets demonstrate that this approach outperforms previous methods in terms of BLEU, ROUGE, and CO metrics.[19] This research tackles challenges in generating natural language questions from knowledge graphs (collections of facts and relationships). Existing systems often produce questions that are difficult to understand or lack context. The authors propose a method to address these issues: (a) Transformer Integration: They incorporate the Transformer model, known for its ability to capture long-range dependencies in text, into a traditional question generation architecture (Bi-LSTM+Attention). This aims to improve the clarity and conciseness of generated questions. (b) Diverse Expressions: To encourage a variety of question phrasings for the same meaning, they leverage a pre-built question database and a semantic similarity algorithm based on Bi-LSTM. This helps the model generate questions with diverse sentence structures while preserving meaning. Experiments show that their Transformer-based approach achieves an 8.36% improvement in question generation accuracy compared to the traditional Bi-LSTM+Attention model. This suggests that the Transformer's capability to handle long-range dependencies leads to more natural and understandable question formulations.[20] This research tackles the issue of fine-grained control in text generation by powerful Transformer-based language models (LMs). While current methods can influence broad aspects like sentiment or topic, precisely controlling specific words and phrases remains a challenge. The paper introduces a novel method named Content-Conditioner (CoCon). This self-supervised approach addresses the issue by

incorporating a CoCon block that assists the LM in finishing partially observed text sequences. The key here is that CoCon feeds the LM with content information that is deliberately hidden from it, allowing the model to condition its output based on this undisclosed content. Evaluations demonstrate that CoCon successfully integrates the desired content into the generated text while also influencing high-level characteristics like sentiment. This is achieved without any additional training (zero-shot), highlighting the method's potential to significantly improve control and accuracy in text generation tasks.[21] This research addresses incoherence issues in long-form text generation with pre-trained language models (LMs). Existing methods struggle to maintain a logical flow of ideas across lengthy texts. The paper proposes PLANET, a novel framework that tackles this challenge. PLANET leverages an autoregressive self-attention mechanism to dynamically plan content and translate it into actual text (surface realization). Here's how it works: (a) Enriched Transformer Decoder: PLANET injects latent representations (hidden information) into the Transformer decoder, the text generation engine. These representations act as a guide, ensuring each sentence aligns with the overall semantic plan. (b) Bag-of-Words Grounding: The semantic plans are further grounded using a "bag-of-words" approach, essentially capturing the core words and concepts that the text should convey. Experiments on complex long-form text generation tasks like counterargument generation and opinion article writing demonstrate that PLANET outperforms existing methods. Evaluations show that PLANET produces significantly more coherent texts with richer content, as confirmed by both automatic metrics and human assessments. [22] This research explores using summarization models based on Transformers for keyphrase extraction in scholarly documents. While traditional methods focus on extracting important words directly from the text, keyphrases can also represent a more abstractive summary. The authors experiment with various Transformer-based summarization models on four benchmark datasets and compare them to common keyphrase extraction methods. Their findings highlight key points: (a) Effectiveness: Summarization models perform well in terms of full-match F1-score and BERTScore, indicating they can effectively capture key information for keyphrase generation. (b) Limitation: These models tend to include many words not explicitly mentioned in the author's provided keyphrases, leading to lower ROUGE-1 scores. (c) Ordering Strategies: The order in which candidate keyphrases are combined can impact performance. The study explores different concatenation strategies but doesn't specify the most effective option. Overall, the research suggests that Transformer-based summarization models show promise for keyphrase extraction, but further work is needed to address the issue of including non-explicit words and optimize ordering strategies.

2.2 Research Gap

Based on the provided information, here are some potential research gaps in the field of text generation:

2.2.1. Limited diversity and repetition: While existing models achieve good performance,they often struggle to generate truly diverse and non-repetitive text. This research gap highlights the need for methods that can produce more original and varied outputs.

2.2.2. Quality evaluation metrics: There's a lack of standardized and comprehensive metrics to accurately assess the quality of generated text beyond fluency and grammatical correctness. This makes it difficult to objectively compare different models and identify areas for improvement.

2.2.3. Generating text with specific constraints: While some research explores generating text with keywords, it can be further extended to handle more complex constraints like specific styles, tones, or factual accuracy.

2.2.4. Computational efficiency: Transformer-based models are powerful but computationally expensive. Research could focus on developing more efficient architectures or training methods for text generation.

2.2.5. Low-resource languages: Most research focuses on high-resource languages Like English. Addressing the gap in text generation models for less well-resourced languages is crucial for wider applicability.

2.3. Problem Statement

The current state-of-the-art language models for text generation, like GPT, achieve impressive results but are limited by their unidirectional nature. This unidirectionality restricts their ability to fully grasp the context of a sentence or passage, potentially hindering creativity and coherence in the generated text.

This research proposes XLNet, a transformer-based model with bidirectional capabilities, as a potential solution to this limitation. XLNet's generalized autoregressive pretraining allows it to consider all possible permutations of the input sequence, enabling a deeper understanding of context.

The core challenge lies in quantifying the advantages of XLNet's bidirectionality compared to unidirectional models like GPT and BERT. This research aims to address this challenge by:

2.3.1. Evaluating the effectiveness of XLNet in bidirectional text generation: We will investigate how XLNet's architecture facilitates the creation of text with greater creativity and coherence compared to unidirectional models.

2.3.2.Developing and testing a Question Answering System (QAS) utilizing BiGenXL: This system will leverage XLNet's bidirectionality to potentially improve performance in tasks like question comprehension, answer generation,and overall accuracy.

2.3.3. Comparative analysis: We will compare the performance of BiGenXL with existing QAS models built on unidirectional models like GPT and BERT. This will involve assessing metrics like creativity, coherence, and overall performance to quantify the benefits of XLNet's bidirectionality in a practical application.

By addressing these challenges, this research aims to provide valuable insights into the potential of XLNet and its bidirectionality for advancing the field of text generation and natural language processing applications like QAS. The findings will contribute to a better understanding of the comparative strengths and limitations of different language models, paving the way for the development of more sophisticated and contextually aware text generation techniques.

3. DATASET

SQuAD Dataset-SQuAD, short for Stanford Question Answering Dataset, is a dataset designed for training and evaluating question answering systems. It consists of real questions posed by humans on a set of Wikipedia articles, where the answer to each question is a specific span of text within the corresponding article. Because the questions and answers are produced by humans through crowdsourcing, it is more diverse than some other question-answering datasets. SQuAD 1.1 contains 107,785 question-answer pairs on 536 articles. SQuAD2.0 (open-domain SQuAD, SQuAD-Open), the latest version, combines the 100,000 questions in SQuAD1.1 with over 50,000 un-answerable questions written adversarially by crowdworkers in forms that are similar to the answerable ones.

4. TECHNICAL SPECIFICATION

4.1. Requirements

4.1.1. Functional Requirements

4.1.1.1 Data Acquisition:

- **Description:** Obtain diverse datasets suitable for training and evaluation.
- **Inputs:** Access to repositories, APIs, or sources providing text data.
- **Outputs:** Raw text data for preprocessing.

4.1.1.2 Data Preprocessing:

- **Description:** Clean and preprocess raw text data to make it suitable for training.
- **Inputs:** Raw text data.
- **Outputs:** Pre-processed data, including tokenization, normalization, and data augmentation.

4.1.1.3 Model Implementation:

- **Description:** Implement the BiGenXL model architecture for text generation tasks.
- **Inputs:** Pre-processed data.
- **Outputs:** Trained BiGenXL model capable of generating text.

4.1.1.4 Fine-Tuning:

- **Description:** Adapt the XLNet model parameters to the specific text generation objectives**.**
- **Inputs:** Pre-processed data, initial XLNet model weights.
- **Outputs:** Fine-tuned XLNet model weights optimized for text generation.

4.1.1.5 Text Generation

- **Description:** Develop mechanisms to generate text from the fine-tuned XLNet model also the proposed architecture named BiGenXL
- **Inputs:** Fine-tuned XLNet model, optional input signals for controlling generation.
- Generated text samples meeting desired criteria (e.g., coherence, relevance).

4.1.2. Non-Functional Requirements

4.1.2.1. Performance:

- **Description:** Ensure efficient and timely generation of text responses.
- **Criteria:** Response time should be within acceptable limits, even under high load conditions.

4.1.2.2. Scalability:

- **Description:** Design the system to handle increasing loads without significant degradation in performance.
- **Criteria:** The system should scale horizontally to accommodate growing user demand.

4.1.2.3. Model Accuracy:

- **Description:** Achieve high accuracy and coherence in generated text.
- **Criteria:** Generated text should closely resemble human-written text in terms of grammar, semantics, and coherence.

4.1.2.4. Robustness:

- **Description:** Ensure the system's ability to handle unexpected inputs or scenarios gracefully.
- **Criteria:** The system should not crash or produce nonsensical outputs when encountering unexpected input or errors.

4.1.2.5. Security:

- **Description:** Protect user data and prevent unauthorized access to the system.
- **Criteria:** Implement secure data handling practices and access controls to safeguard user information.

4.2. Feasibility Study

4.2.1. Technical Feasibility

4.2.1.1. Strengths:

- **Diverse Text Generation:** XLNet's bidirectional nature can create text that flows more naturally in both directions, opening doors for creative writing applications where the starting point might not be at the beginning of a sentence.
- By considering all permutations of word order,XLNet can potentially capture context more effectively, leading to more relevant and cohesive generations.

- **Abstractive Capabilities:** XLNet's ability to go beyond the information explicitly stated in the context makes it suitable for tasks like creative writing, summarization, and question answering, where generating novel and informative content is crucial.
- **Multilingual Support:** Pre-trained XLNet models are often available in multiple languages, enabling text generation in various languages without the need for extensive language-specific training.
- **Integration with Other Tools:** XLNet can be integrated with other NLP tools and techniques, such as sentiment analysis or topic modeling, to further enhance the generated text's quality and relevance.

4.2.1.2. Challenges:

- **Computational Cost:** Training XLNet models can be computationally expensive due to the complex permutation language modeling approach. This might necessitate powerful GPUs or specialized hardware.
- **Fine-Tuning:** XLNet, like other large language models, requires fine-tuning on specific text generation tasks to achieve optimal performance. This fine-tuning process can be time-consuming and requires expertise in NLP techniques.
- **Data Requirements:** XLNet models require a massive amount of text data for training to achieve good performance. This can be challenging to acquire and pre-process, especially for specialized domains or niche applications.
- **Limited Explainability:** As with other complex neural networks, understanding how XLNet arrives at its generated text remains a challenge. This lack of interpretability can be a limitation in situations where reasoning and justification are important.
- **Ethical Considerations:** The potential for misuse of XLNet for generating misinformation or propaganda necessitates careful development and deployment practices to mitigate these risks.

4.2.2. Non-Technical Feasibility

4.2.2.1. Strength:

- **Impressive Results:** XLNet is known for generating coherent and grammatically correct text, especially for tasks involving long contexts. This makes it ideal for scenarios where you need realistic and flowing text output.
- **No Sequence Limit:** Unlike some models, XLNet can handle text of any length. This is a major advantage if you're dealing with open-ended generation or working with large datasets.
- **Handling Rare Words:** XLNet's permutation language modeling approach helps it handle rare words and unseen vocabulary better than some other models, making it suitable for generating text in specialized domains or creative writing scenarios.
- **Adaptability:** XLNet can be adapted to various text generation tasks through fine-tuning and controlling the generation process using techniques like beam search and temperature sampling.

4.2.2.2. Challenges:

- **Computational Cost:** Running XLNet can be computationally expensive, especially for larger models. This might limit its feasibility for applications on resource-constrained environments.
- **Complexity:** XLNet is a powerful model, but it also comes with a steeper learning curve. Setting it up and fine-tuning it for specific tasks requires some technical knowledge of libraries like TensorFlow or PyTorch.
- **Memory Requirements:** XLNet models can require significant memory during inference, especially for larger models. This can be a limitation for deployment on devices with limited memory resources.
- **Potential Biases:** Like any large language model, XLNet can inherit biases present in the training data. Careful data selection and mitigation strategies are crucial to prevent biased or offensive outputs.

4.3.System Specification

4.3.1. Software Specification

- Google Colab, T4 GPU, 51 GB RAM, 201 GB Disk Space

4.3.2. Libraries Used

- **Transformers-**The Transformers library by Hugging Face is a powerful open-source toolkit for Natural Language Processing (NLP) tasks. It offers a vast collection of pre-trained models based on Transformer architecture, known for their state-of-the-art performance in tasks like text classification, question answering, and text generation. Its user-friendliness shines through with straightforward interfaces, making it easy to implement complex models and leverage pre-trained power even for beginners. This library empowers developers of all skill levels to access and utilize advanced NLP capabilities with minimal effort.

- **PyTorch-**PyTorch, a popular open-source Deep Learning library, shines in computer vision and NLP tasks. Its Pythonic interface makes it user-friendly, while its dynamic neural network construction allows for flexibility. PyTorch leverages GPUs seamlessly for speed, boasts a rich ecosystem of tools, and is production-ready with functionalities like TorchScript, making it a powerful choice for Deep Learning projects.

- **Text-Hammer-**Text-hammer is a Python library specializing in text preprocessing. It simplifies data cleaning by offering functions for tasks like normalization, contraction expansion, special character removal, and HTML/URL handling. Easy to use with both individual functions and integration into larger workflows, text-hammer streamlines text preparation for NLP and machine learning projects, requiring only spaCy and Python 3 for its magic.

- **Datasets-**The datasets library streamlines working with machine learning datasets, especially for NLP, computer vision, and audio tasks. Built by Hugging Face, it offers a user-friendly platform to access a vast collection of pre-processed datasets with a single line of code, saving time and effort, to efficiently prepare data for training through built-in functionalities for splitting, filtering, and tokenization, Work with large datasets without memory limitations thanks to the memory-efficient Apache Arrow format, Seamlessly integrate with popular libraries like NumPy, Pandas, PyTorch, and TensorFlow for smooth data handling in ML workflows, Contribute to the ML community by easily sharing your own datasets and discovering new ones, fostering collaboration.

- **nltk**-NLTK, the Natural Language Toolkit, is a powerhouse for NLP tasks in Python. It offers a comprehensive toolkit for working with human language data, from basic text

processing like tokenization and stop word removal to advanced tasks like part-of-speech tagging, parsing, named entity recognition, and even semantic reasoning. Its user-friendly interface and rich ecosystem of resources, including over 50 corpora and lexical resources like WordNet, make it easy to get started and tackle complex NLP projects. Open-source and driven by a large community, NLTK empowers researchers, data scientists, and anyone working with natural language data with powerful tools for efficient text analysis and understanding, fostering collaboration and knowledge sharing within the NLP field.

- **NumPy-**NumPy, the cornerstone of scientific computing in Python, empowers efficient handling of numerical data. Its core strength lies in the powerful "ndarray" object, enabling storage and manipulation of large datasets in multidimensional arrays. NumPy excels in array operations, performing calculations element-wise across entire arrays for significant speed gains compared to traditional Python loops. Its broadcasting mechanism simplifies complex calculations by automatically expanding arrays for element-wise operations. Additionally, NumPy offers comprehensive linear algebra functionalities and high-quality random number generation, making it vital for simulations and statistical analysis. This performance, efficiency, and seamless integration with other scientific Python libraries like pandas and scikit-learn solidify NumPy as an indispensable tool for anyone working with numerical data in Python.

- **AutoTokenizer-**AutoTokenizer, a key player in the Transformers library, simplifies text preprocessing for NLP tasks using transformer models. It handles the crucial step of tokenization, breaking down text into units the model understands, while also managing vocabulary and special tokens. Its magic lies in automatic model compatibility - you don't need to know specific tokenizer details, as AutoTokenizer identifies the correct one based on the chosen model. Pre-trained tokenizers associated with various models ensure optimal performance, but customization options are available for fine-tuning or creating custom tokenizers when needed. Overall, AutoTokenizer streamlines the process, saving time, boosting efficiency, and offering flexibility, making it an essential tool for anyone working with transformer-based NLP tasks.

  Syntax-

  Tokenizer=AutoTokenizer.from_pretrained(“model-name”)

- **Dataset.map()-**dataset.map() is a powerhouse for transforming and enriching datasets in libraries like Hugging Face Datasets and TensorFlow Datasets. It lets you apply your own custom function to each element within the dataset, opening doors to various transformations. Think text preprocessing (lowercase, tokenization), data augmentation (new samples through back-translation), feature engineering, or even filtering based on criteria. It shines with efficiency, often utilizing parallel processing for faster transformations compared to manual loops. Plus, it offers flexibility for defining any custom function and even batch processing for further speed-ups, especially on large datasets. Overall, dataset.map() is a versatile tool for tailoring your data before feeding it to machine learning models, making it a valuable asset in your data pre-processing workflows.

- **Data Collator-**A data collator is a tool used in machine learning, specifically in tasks like natural language processing. It gathers individual data samples and combines them into batches, which are groups of data processed together during training. This helps make the training process more efficient and can also involve preprocessing steps like padding shorter samples to a uniform length.

- **Flask-**Flask is a lightweight Python framework used for building web applications. It's known for its simplicity and flexibility, allowing developers to easily create dynamic web pages with minimal code. Flask provides basic tools for handling web requests, routing URLs to functions, and rendering HTML templates. This makes it ideal for building small to medium-sized web applications, APIs, and microservices.

5. DESIGN APPROACH AND DETAILS

5.1. SYSTEM ARCHITECTURE :-

The architecture of the proposed model consists of XLNet in combination with Top K Beam bidirectional Text Generation Algorithm given name as BiGenXL .XLNet is an extension of the Transformer-XL model.XLNet consists of several transformer decoder blocks that are used to encode the input sequence and capture word-to-word dependencies.

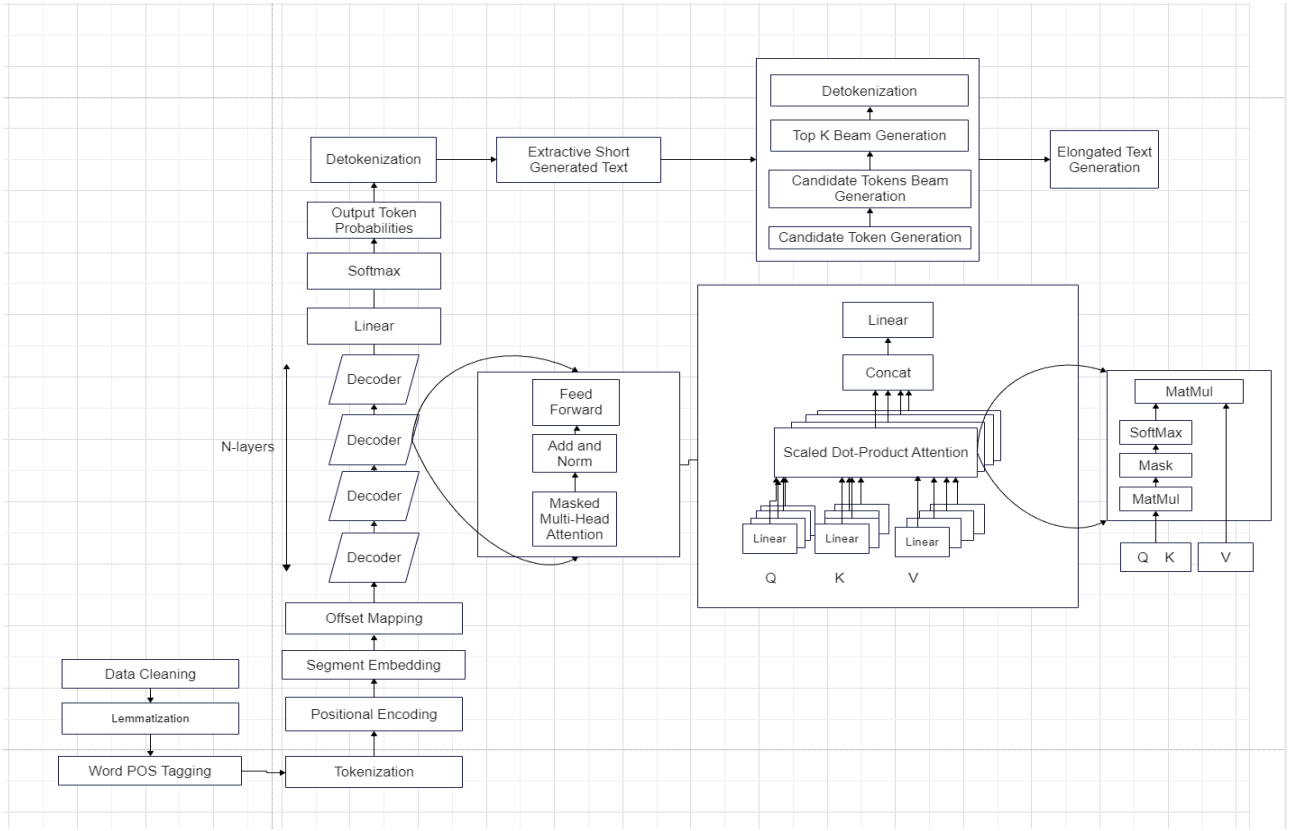

Fig 2.BiGenXL Architecture

As mentioned earlier the XLNet model is made up of several layers of transformer decoder blocks.In the proposed BiGenXL architecture the input data passes through several layers of the model.The description of each layers are:

5.1.1. Token Embeddings: Token embeddings are a way to represent words or other pieces of text (tokens) as numerical vectors in a computer. These vectors capture the meaning and relationships between words, which is helpful for machines to understand language.

5.1.2 Segment Embeddings: In addition to token embeddings, XLNet uses segment embeddings to indicate boundaries between different segments of the input sequence. This is useful for tasks such as answering a question,where the model must distinguish the question from the context.

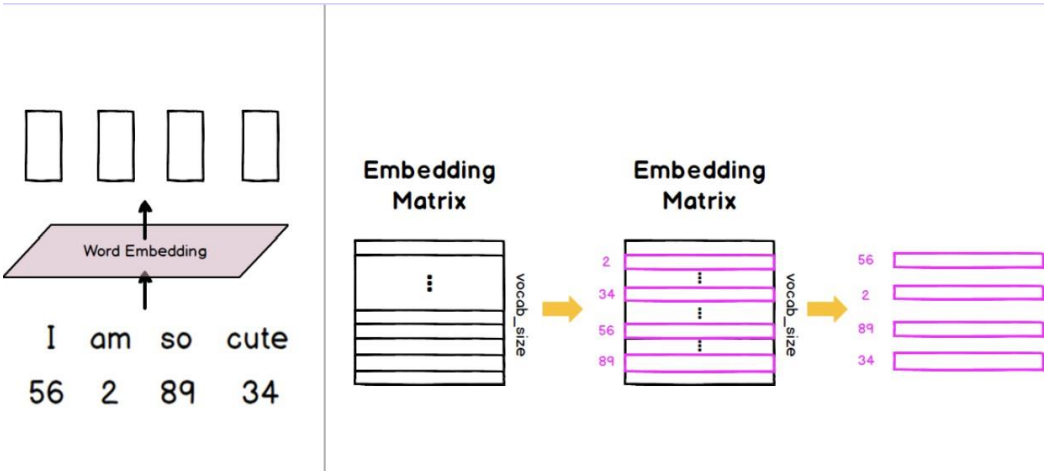

Fig 3.Embedding Matrix

The word embedding process is no different from the traditional method. With the index of each token, the row of that index is extracted from the Embedding Matrix and used as an embedding vector for each token.

5.1.3. Positional Encoding:XLNet uses position encoding to indicate the position of each character in the input sequence. Unlike other models, XLNet uses a unique location code called "relative location coding," that reflects the relative distances of the characters in the series.

At this time, the position of the word is expressed as a combination of sin and cos values. Depending on the model, Position Encoding may also have an additional matrix like word embedding, use the order of words in the sentence as the index, and use the row vector of that index as the position vector. In this case, the position embedding matrix is trained together with the model.

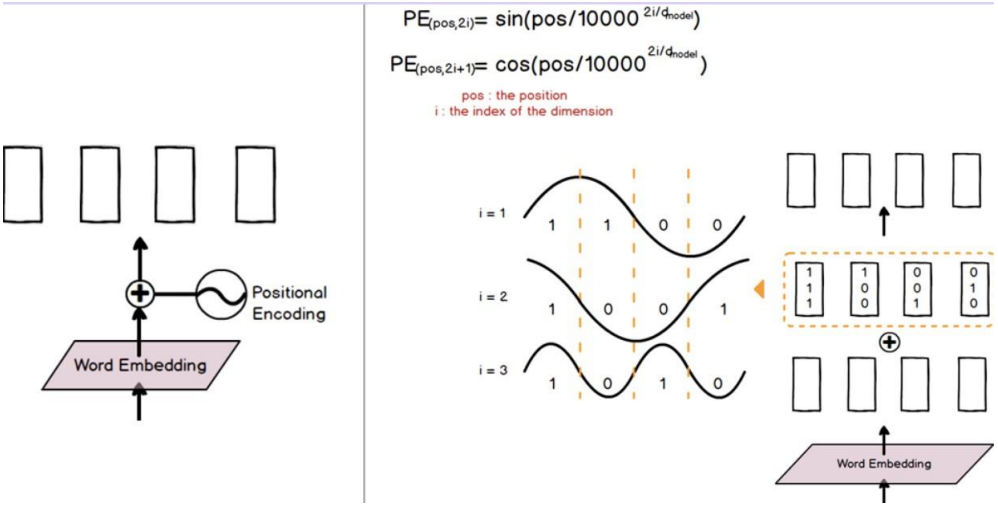

Fig 4. Positional Encoding

5.1.4. Masked Multi Head Attention Mechanism: Attention Mechanism is the core structure of Transformer. In particular, the Masked Attention structure is a technique used in Transformer's decoder, and is used when the next word must be predicted using only words that previously appeared (i.e., when information about words that will appear later should not be used). Of course, this is also possible during training, but during testing, where the next word must be predicted based on the previous output, parallel processing will not be possible even if masked attention is used.As described in the introduction part the XLNet is made up of multiple layers of decoder part of the transformer model. The attention type of decoder part is masked multi-head attention.To calculate the probability of the next token it takes three inputs Query,Key and value.

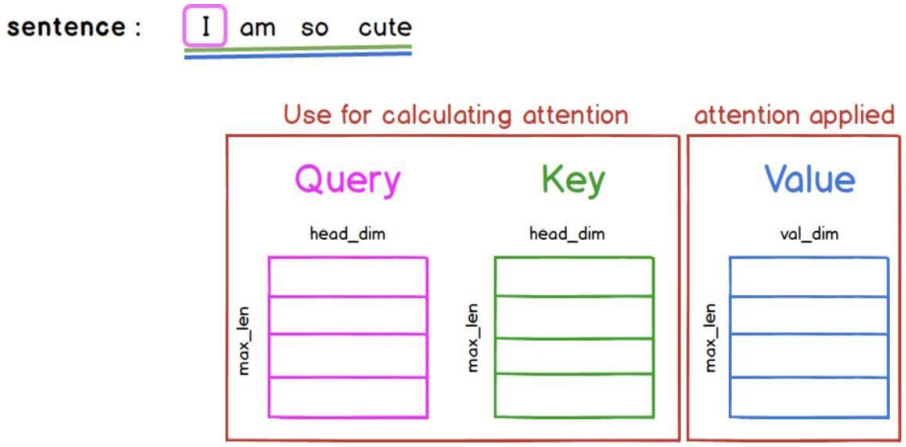

Fig 5. Query,Key and Value representation

As given in the above diagram each row in the query is a standard word. For each of these standard words, the degree of relationship with words in the sentence is obtained using Key Matrix. And if we find the relevance using Query and Key, multiply this relevance by the Value and adjust the values with high relevance to be larger and the values with low relevance to be small.

Each row of Query and Key represents each word in the sentence. And what the result of multiplying the two matrices means is the degree of relationship between each word. For example, the value of (1, 2) in the product of two matrices means that when predicting the next word using the first word “I”, you should focus on the second word “am”.But as we may have noticed, there is a problem. We need to predict “am” using the first word “I”, but the answer, “am”, is already shown in the attention score we obtained. It's like showing the answer openly. To solve this problem, we use the “Masked Softmax” technique as shown below.

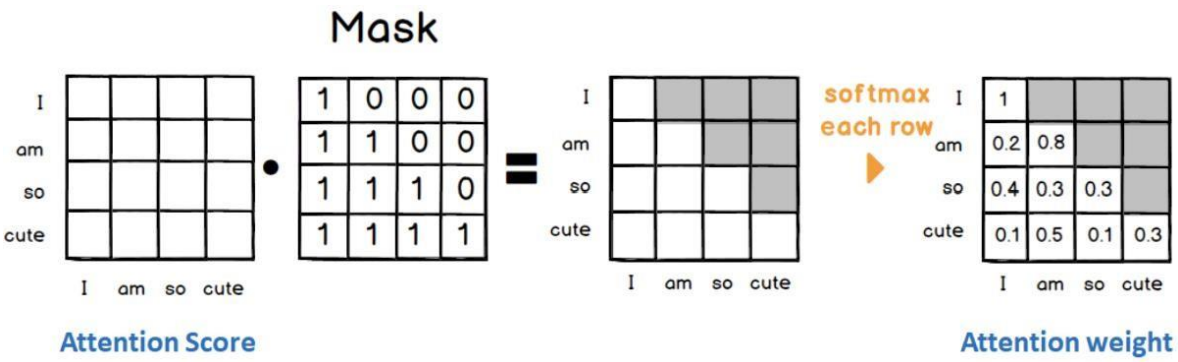

Fig 6. Attention Mechanism

It masks all values after itself in the Attention Score obtained using Query and Key. Afterwards, normalize the remaining values with softmax. In this way, the attention weight is calculated by considering only the previous words and is finally applied to the Value Matrix

5.1.5. Linear Layer: After the input data passes through the decoder blocks it passes through a linear layer in order to convert the output of each decoder layer into the dimension as same as the input data.The linear layer acts as an adapter, transforming the internal representation from the decoder's final layer to the required output dimension.After passing through the last and final linear layer of the last decoder block it takes the processed information from the decoder and generates a vector of logits. These logits represent the unnormalized scores for each word in the vocabulary.

5.1.6. Softmax Activation Function: Softmax is primarily used in the output layer of a neural network for multi-class classification tasks. Its role is to convert a vector of real numbers (logits) into a probability distribution.The received output from the linear layer is passed as input to the softmax activation function.The softmax function takes a vector of logits as input and squashes them into a probability distribution where each element represents the probability of a particular class and all elements sum to 1.

The output from the softmax layer is the predicted tokenized short generated text answer.The SQuad Dataset is mainly used for extractive question answering but using top K beam bidirectional Text Generation our model will be generating an abstractive elongated answer.The **Top K beam bidirectional Text Generation** algorithm consists of several layers of computation.The layers are:

5.1.7. Candidate Token Generation: This function generates candidate continuations for

a sentence. It takes several arguments:

a) PADDING_TEXT: Text used for padding.

b) sent_tokens: List of token indices for the current sentence.

c) candidate: A tuple representing the current candidate (list of tokens, cumulative probability, list of individual word probabilities).

d) Direction for generation ('left' or 'right').

e) n_candidates: Number of candidates to generate

f) topk: Number of top words to consider from the model's output.

g) temperature: Temperature parameter to control randomness.

The function first encodes the padding text and mask token using the tokenizer.It then creates a mask depending on the direction (d). For left-side generation, only previous tokens can see the current target token. For right-side generation, only future tokens can see it. This is achieved using a permutation mask (perm_mask).Next, it defines a target mapping highlighting the target token position for the model's output. The model outputs are passed through a softmax function and temperature scaling for more diverse sampling. The function iterates through the chosen indices, creating new candidate sentences by appending the chosen token depending on the direction. It also updates the cumulative probability and the list of individual word probabilities for the new candidate. Finally, it returns a list of these new candidate tuples.

5.1.8. Candidate Token Beam Generation: This function performs beam search for text generation. It takes:

a) PADDING_TEXT: Text used for padding.

b) sent_tokens: List of token indices for the current sentence.

c) candidates: List of initial candidate tuples.

d) depth: Number of beam search steps.

The function iterates for depth steps. In each step, it loops through the current candidates and uses candidates_gen to generate new continuations for each candidate. These new continuations are then added to both the current list of candidates (candidates) and a temporary list (new_candidates). After all expansions are done, the function sorts the complete candidate list (candidates) by the sum of the logarithms of their individual word probabilities (equivalent to the product of probabilities). This favors sequences with consistently high probabilities

throughout. Finally, it returns both the complete list of expanded candidates and the sorted list.

5.1.8. Top K Beam Generation:

This function takes probabilities (probs) as input, along with optional arguments for the number of top elements to choose (k) and the number of samples to draw (sample_size). It first finds the indices of the k highest probability elements using np.argpartition. Then, it normalizes the probabilities of these top elements and uses np.random.choice to randomly sample sample_size elements with probabilities corresponding to their normalized values. It returns the indices of the chosen elements.

5.2. Design

5.2.1. Data Flow Diagram

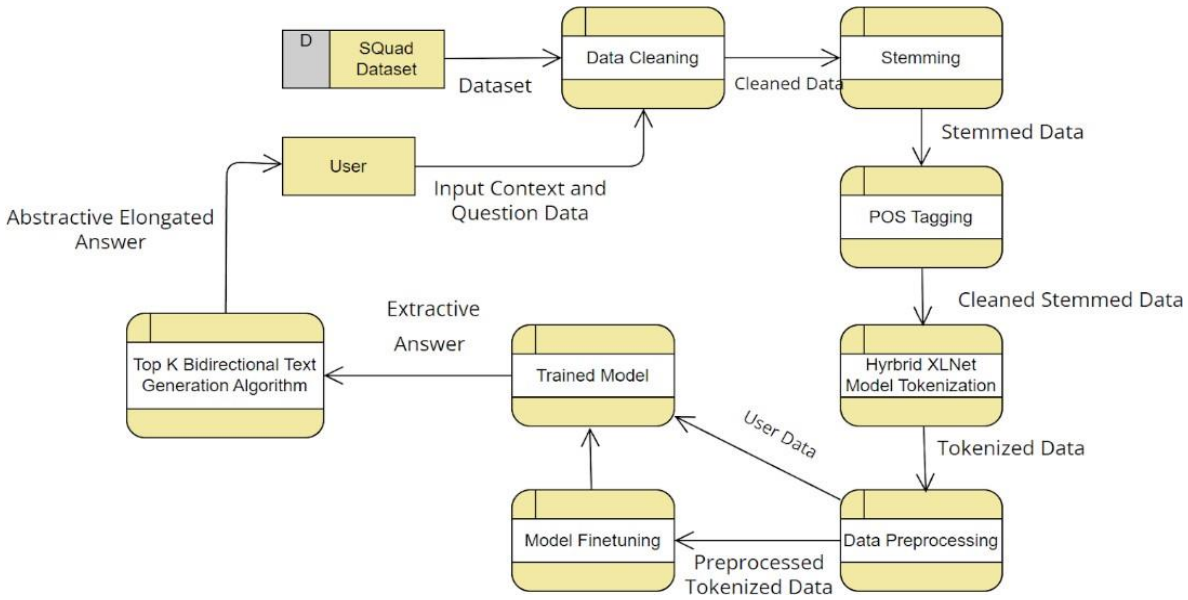

Fig 7. Data Flow Diagram

5.2.2. Use Case Diagram

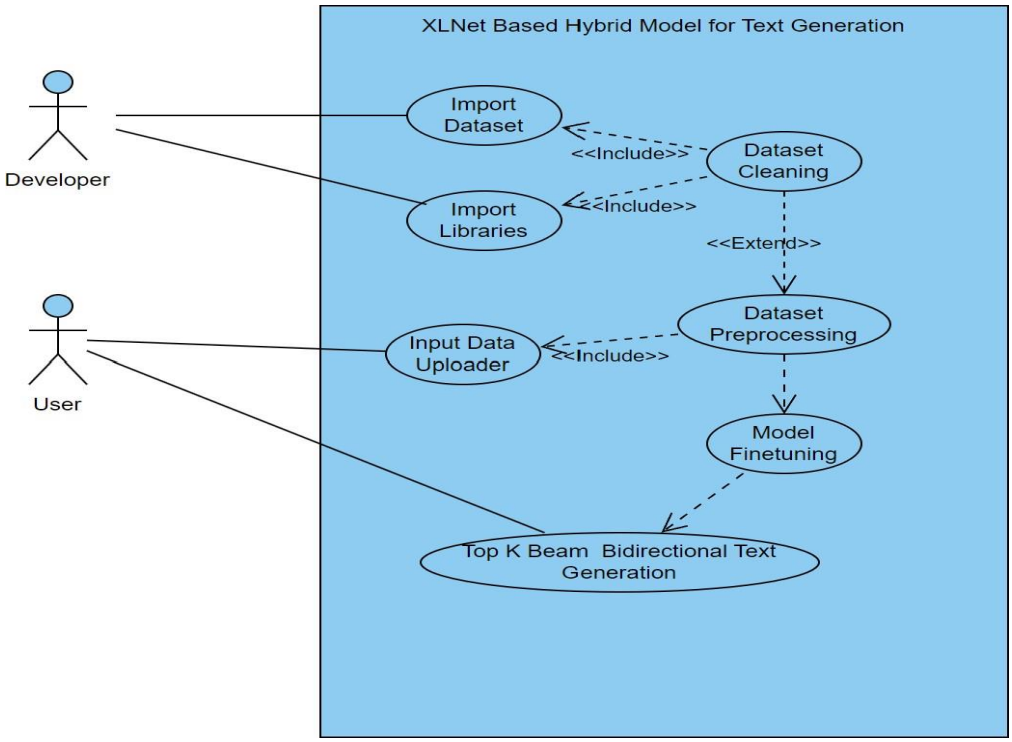

Fig 8. Use Case Diagram

5.2.3. Class Diagram

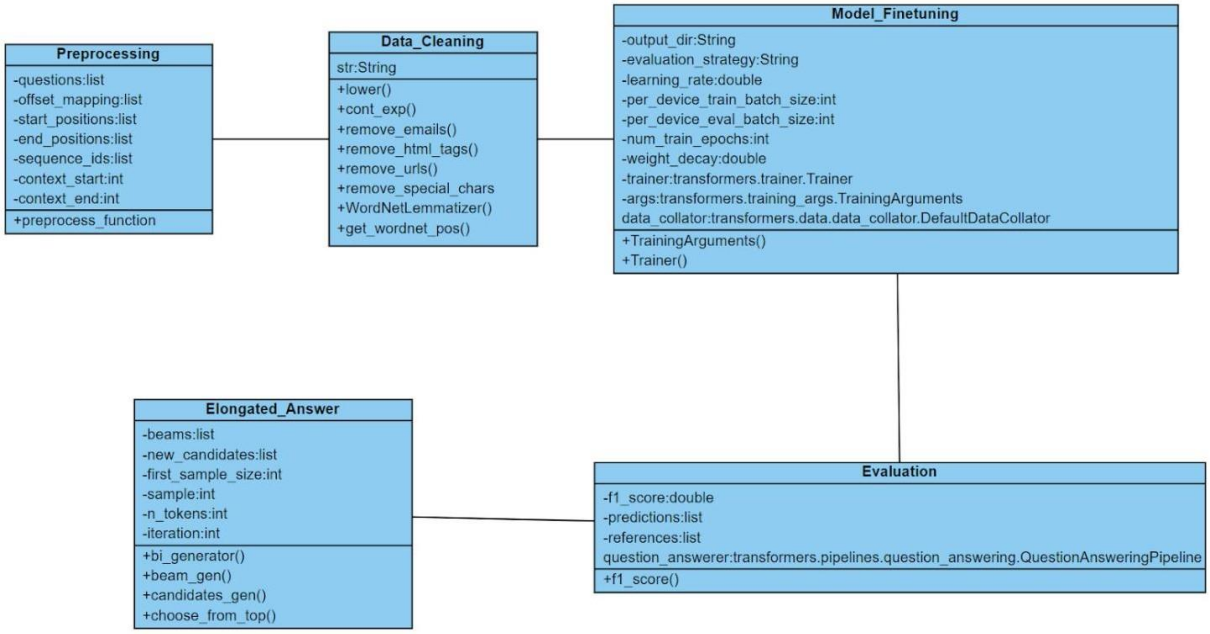

Fig 9. Class Diagram

5.2.4. Sequence Diagram

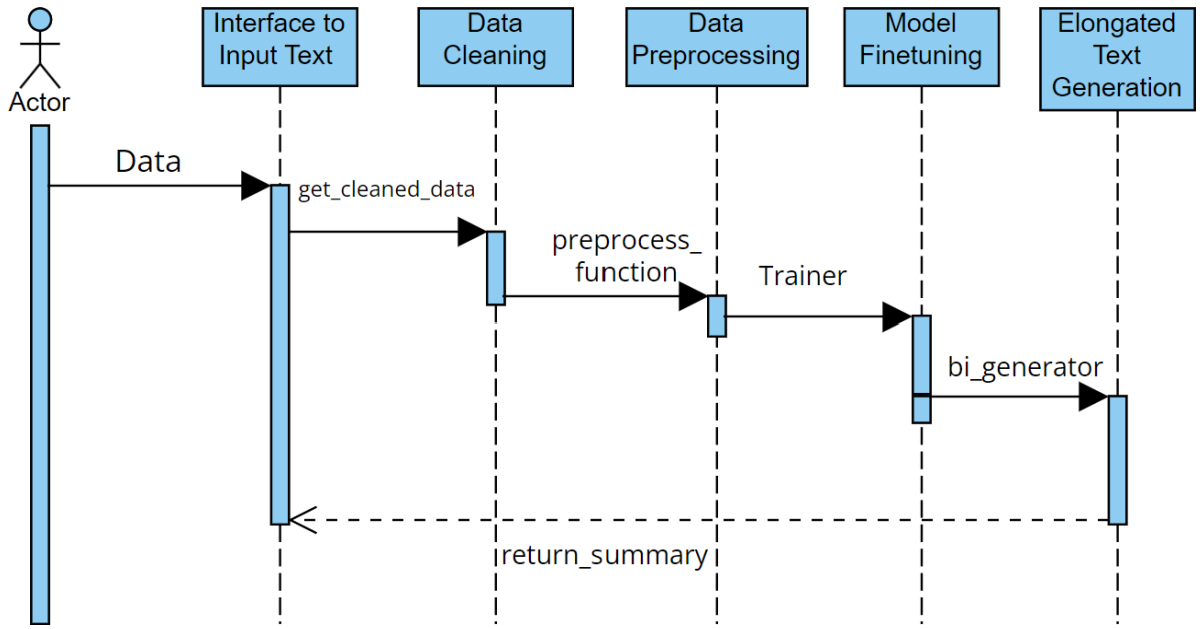

Fig 10. Sequence Diagram

5.3. Constraints and Trade-offs

5.3.1. Constraints

- **Computational Cost:** XLNet is a complex model with a massive number of parameters, making it computationally expensive to train and run. This limits its usage on resource-constrained hardware and requires significant computational power.

- **Data Requirements**: XLNet requires a massive amount of text data for training to achieve good performance. This can be challenging to acquire and pre-process, especially for specialized domains.

- **Limited Explainability**: XLNet is a complex neural network, making it difficult to understand how it arrives at its answers. This lack of interpretability can be a drawback in situations where reasoning and justification are important.

- **Beam Search Complexity:** Beam search helps in generating more diverse and informative answers, but it significantly increases computational complexity and inference time, especially with higher beam sizes.

- **Fine-tuning Challenges:** Fine-tuning XLNet for question answering requires careful selection of hyperparameters and training strategies to balance abstractive capabilities with factual accuracy and coherence.

- **Domain Specificity:** XLNet, like most large language models, is trained on a general corpus of text data. This can lead to limitations when dealing with specific domains or topics that require specialized knowledge or terminology. Fine-tuning on domain-specific data can help mitigate this issue, but it requires access to relevant datasets.

- **Sensitivity to Bias:** Large language models like XLNet can inherit biases present in the training data. This can lead to discriminatory or offensive outputs, especially when dealing with sensitive topics. Careful data selection and mitigation strategies are crucial to address potential biases.

- **Error Handling and Robustness:** XLNet may struggle with handling noisy, ungrammatical, or factual errors in the input text. This can lead to inaccurate or nonsensical answers. Robustness techniques like data augmentation and error correction methods can be employed to improve the model's ability to handle imperfect inputs.

5.3.2. Trade-offs

- **Abstractiveness vs. Factuality:** While XLNet excels at generating abstractive answers that go beyond the information explicitly stated in the context, it can sometimes lead to factual inconsistencies or introduce biases present in the training data.

- **Fluency vs. Coherence:** Beam search can lead to more fluent and natural-sounding answers, but it can also introduce inconsistencies or incoherent sentence structures if not carefully controlled.

- **Diversity vs. Accuracy:** Increasing the beam size leads to more diverse answers but can also include less accurate or irrelevant information. Striking the right balance is crucial.

- **Computational Cost vs. Answer Quality:** While a larger beam size generally improves answer quality, it comes at the cost of significantly higher computational resources. Finding the optimal beam size depends on the specific application and available resources.

- **Interpretability:** XLNet is a complex model, making it challenging to understand how it arrives at its answers, which can be a limitation for applications requiring high levels of transparency.

- **Domain Specificity:** XLNet models trained on general text data may not perform well on specific domains without further fine-tuning, requiring additional training data and resources.

- **Model Size vs. Performance:** Larger XLNet models with more parameters generally achieve better performance but require significantly more computational resources for training and inference. This can be a trade-off for applications with limited computational power.

- **Training Data vs. Generalizability:** The amount and quality of training data significantly impact the performance and generalizability of XLNet. While a massive dataset can lead to better performance, it also requires significant effort and resources for acquisition and pre-processing.

- **Explainability vs. Accuracy:** While techniques like attention visualization can offer some insights into the model's decision-making process, achieving high levels of interpretability often comes at the cost of reduced accuracy. Finding a balance between these two aspects is an ongoing challenge in the field of natural language processing.

- **Hyperparameter Tuning:** Fine-tuning XLNet for question answering involves adjusting various hyperparameters like learning rate, batch size, and beam size. Finding the optimal configuration requires careful experimentation and evaluation to balance performance with computational efficiency.

6. METHODOLOGY

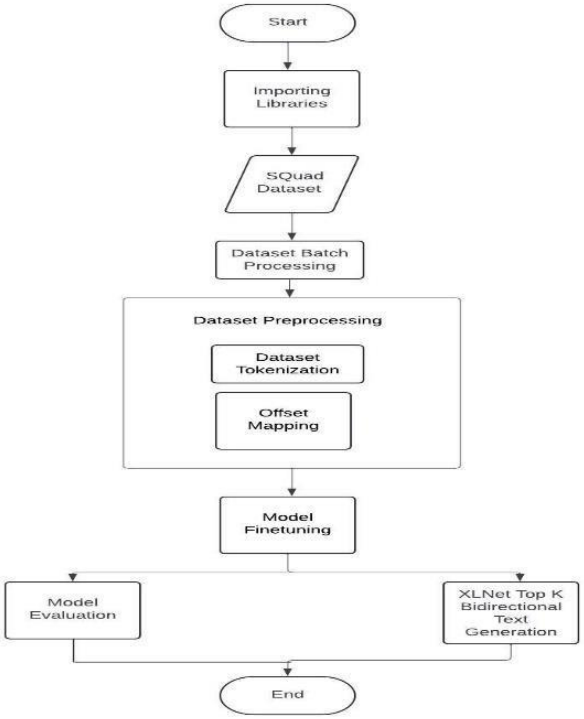

Fig. 11 Flowchart of BiGenXL

The flowchart given above shows the workflow process of BiGenXL to generate elongated answer for the given question.The dataset consists of a context and a question along with an answer.The steps followed by the proposed model to generate answer based on the given dataset is:

6.1. Dataset Cleaning: It involves identifying and correcting errors, inconsistencies, and missing information in a dataset before it can be used for further analysis or modeling.It involves removing extra spaces,newlines,converting text to lowercase.The dataset cleaning also involves lemmatization to get better understanding of the context.

6.1.1. Text Cleaning:

a. Remove extra spaces, newlines, and tabs.

b. Convert text to lowercase.

c. Handle punctuation consistently (e.g., remove unnecessary punctuation,standardize quotation marks).

6.1.2. Spelling Correction:

a. Employed a spell checker to fix typos and misspellings.

6.1.3. Lemmatization: Lemmatization is the process of reducing a word to its base or dictionary form, also known as its lemma. It's a common technique used in natura language processing (NLP) tasks to group together different inflected forms of a word for analysis.

6.1.4. Stemming: Stemming, in the context of natural language processing (NLP) refers to the process of reducing a word to its base or root form, also known as a stem. The main objective is to group together different inflected forms of a word (e.g., playing, plays, played) into a single representation (play) for easier analysis.

6.2. Dataset Preprocessing: Using XLNet tokenizer to convert questions and contexts into sequences of tokens (words or subwords) suitable for the model to train.Offset mapping is applied on tokenized data to obtain a mapping between the original text and the tokenized sequence which will be used later for answer span identification.The dataset processing also involves answer span mapping.The answer span mapping involves extracting answer text and answer starting positions from the dataset.Using the offset mapping information the answer span is identified in the tokenized sequence.

Example

Question: "What is the capital of France?"

Context: "France is a country located in Western Europe. Its capital is Paris, a major global city and a center for fashion, art, and culture."

Answer: "Paris" (starting at the 37th character and ending at the 42nd character in the context

6.2.1. Tokenization:

a. Uses a pre-trained tokenizer (likely specific to the XLNet model being used) to convert questions and contexts into sequences of tokens (words or subwords) suitable for the model.

b. XLNet utilizes subword tokenization, unlike traditional word-based tokenization. This means it breaks down words into smaller meaningful units called subwords.

c. XLNet uses a pre-defined vocabulary of subwords, which is typically built using algorithms like Byte Pair Encoding (BPE).

d. Masking is then applied to hide certain tokens in the permuted sequence. The model learns to predict the masked tokens based on the surrounding context, even with the shuffled order.

e. Set max_length to control the maximum number of tokens allowed in each input sequence.

f. Set offset mapping to obtain a mapping between the original text and the tokenized sequence which will be used later for answer span identification.

g. Truncate the context if it exceeds the max_length, ensuring the question remains complete.

6.2.2. Answer Span Mapping:

a. Extract answer text and answer starting positions from the dataset.

b. Iterate through each example (question-context-answer pair) in the dataset.

c. Find the starting and ending character positions of the answer within the original context text.

d. Use the offset_mapping information returned by the tokenizer to identify the corresponding token positions for the answer's start and end within the tokenized sequence.

e. Handle cases where the answer might not be entirely contained within the truncated context by setting the start and end positions to 0 (indicating no answer found).

6.3. Model Finetuning: The preprocessed dataset is passed through the XLnet model.The model will identify the start and end of the answer span within the context.XLNet can leverage its bidirectional processing to understand the relationships between words in the passage and question, improving answer accuracy.

6.3.1. Training Arguments: Establishes a TrainingArguments object that sets essential training hyperparameters for the fine-tuning process. These arguments include:

6.1.3. output_dir: Directory for saving the fine-tuned model.

6.1.4. evaluation_strategy: Controls how often evaluation is performed during training.

6.1.5. learning_rate: Sets the learning rate for the optimizer, which controls how much the model's weights are updated during training.

6.1.6. per_device_train_batch_size: Defines the number of training examples processed on a single graphics processing unit (GPU) in each batch.

6.1.7. per_device_eval_batch_size: Defines the number of evaluation examples processed on a single GPU in each batch.

6.1.8. num_train_epochs: Specifies the total number of epochs (complete training cycles) to run for fine-tuning.

6.1.9. weight_decay: Controls the amount of regularization applied to prevent overfitting during training.

6.3.2. Trainer Initialization: Creates a Trainer object, which acts as the core component for managing the fine-tuning process. This object takes the following arguments:

a. model: Reference to the pre-trained XLNet,Bert and GPT-2 model to be fine-tuned.

b. args: The Training Arguments object containing hyperparameters.

c. train_dataset: The preprocessed training dataset containing questions, contexts, and answer labels.

d. eval_dataset: The preprocessed evaluation dataset (potentially a separate subset of the training data or a different dataset) used to monitor model performance during training.

e. tokenizer: The tokenizer used to convert text data into tokens understandable by the XLNet model.

f. data_collator: A custom function to perform additional processing on batches of data before feeding them to the model.

6.3.3. Fine-Tuning Training:Calls the trainer.train() method, which initiates the fine-tuning process using the specified training arguments,datasets, and model. This involves:

a. Forward propagating training examples through the pre-trained XLNet,Bert,GPT-2 model with fine-tuned layers for Q&A tasks.

b. Calculating the loss based on predicted answer spans and actual answer labels.

c. Backpropagating the loss to update the model's weights based on the optimizer and learning rate.

d. Iterating through training epochs and batches.

e. Potentially performing evaluation steps as defined by the evaluation_strategy.

6.4. Top K beam Bidirectional Text Generation: The generated extractive answer is given as input to the Top K Beam Bidirectional Text Generation algorithm for elongated abstractive answer for the question.The steps in the algorithm involves candidate token generation.Based on the directionality provided whether we want to generate text in left or right side or in both directions,using the masked multihead attention the next probable token are generated. Given the number of iterations we can generate as lengthy answer text as we want.For as each candidate token it is appended to form a beam of candidate tokens. In this similar way K number of beams are generated by the algorithm and based on the logits value of each beam the beam with the highest probability is chosen and appended to the extracted answer.

6.5. Model Evaluation: The finetuned model is evaluated using using F1 Score.The F1 score, also known as the F-measure or F1-measure, is a metric used in machine learning to evaluate the performance of a classification model. It is a harmonic mean that combines two other commonly used metrics: precision and recall.The output obtained by the finetuning the model is compared with the original output given in the dataset and is used to give f1 score.

7. SCHEDULE,TASKS AND MILESTONES

7.1. Gantt Chart

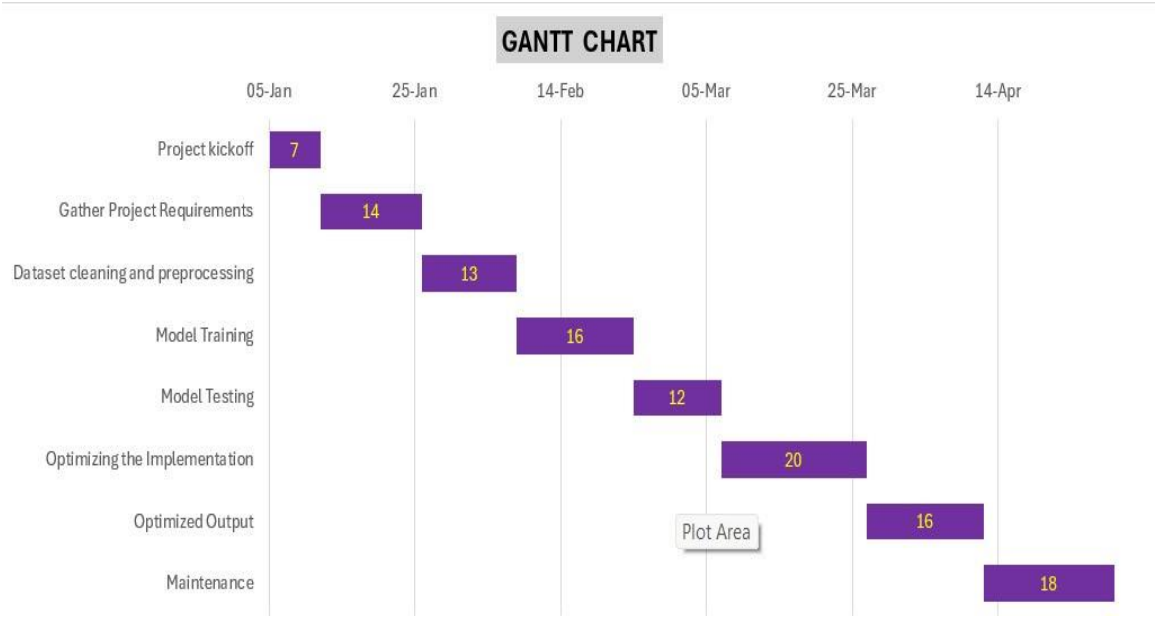

Fig.12 Gantt Chart

7.2. Testing

7.2.1. Unit Testing

7.2.1.1. XLNet Model

- Testing the accuracy of the pre-trained XLNet model on benchmark SQuAD dataset.
- Verifying the functionality of custom modifications made during fine-tuning for question answering.
- Introducing intentional noise or adversarial perturbations to the input text and verify if the model's predictions remain consistent and accurate.
- Evaluating the model's performance across different question types (e.g., factual, open ended, yes/no) to identify potential areas for improvement.
- Utilizing techniques like attention visualization to understand the model's reasoning and decision-making process for specific predictions.

7.2.1.2. Beam Search

- Testing the correctness of beam search implementation, ensuring it generates the desired number of candidate answers and maintains beam diversity.
- Verifying that the beam search prioritizes high-scoring candidate answers based on attention scores.
- Analyzing how the beam size affects the diversity and quality of generated candidate answers.
- Ensuring the beam search terminates correctly when the desired number of candidate answers is reached or a maximum score threshold is met.
- Evaluating how the beam search handles computationally expensive scenarios with long or intricate questions.

7.2.1.3. Text Generation

- Testing the fluency and grammatical correctness of generated text using language models

- Ensuring the generated answers are relevant to the question and factually consistent with the provided context.
- Verifying that the generated text aligns with the factual information provided in the context and maintains a logical flow of ideas.
- Analyzing the generated text to ensure it produces diverse and creative responses that go beyond simply repeating information from the context.
- Evaluating the model for potential biases in its generated text, such as gender or racial bias, and implement mitigation strategies if necessary.

7.2.2. Integration Testing

7.2.2.1. End-to-End Functionality

- Test the overall question answering pipeline, ensuring the system takes a question and context as input, generates relevant answers using XLNet and beam search, and returns the final answer(s).
- Verifying how the system gracefully handles unexpected inputs (e.g., empty questions, nonsensical context) and provides informative error messages.
- Measuring the overall response time of the system for different question lengths and context sizes, ensuring it meets performance requirements under load.

7.2.2.2. Question Processing

- Verifying that the system correctly preprocesses questions,including tokenization, stemming/lemmatization, and handling special characters.
- Testing if the system can accurately identify and classify named entities (e.g., people, locations, organizations) within the question and context, potentially improving answer accuracy.
- Verifying if the system can identify and link mentions of the same entity across the question and context, leading to a better understanding of the relationships between entities.
- Test how the system deals with ambiguous or vague language in questions, potentially

utilizing external knowledge bases or disambiguation techniques.

7.2.2.3. Contextual Understanding

- Test how well the system utilizes the provided context to generate answers that are consistent with the factual in formation and relevant to the question being asked.
- Analyzing if the system can make logical inferences based on the provided context to answer questions that go beyond simple information retrieval.
- Verifying how the system handles questions involving temporal information (e.g., past, present, future) and ensures answers are consistent with the timeline presented in the context.
- Testing if the system can understand the sentiment of the question and context, potentially leading to more nuanced and appropriate answers.

7.2.2.4. Answer Ranking and Selection

- Ensure the system effectively ranks and selects the most relevant and informative answer from the generated candidates using appropriate metrics.
- Utilize various metrics beyond simple accuracy to assess answer quality, such as informativeness, relevance, factuality, and natural language fluency.
- Ensure the system selects diverse and non-redundant answers, avoiding repetitive or irrelevant information.
- Consider incorporating user feedback mechanisms to continuously improve the answer ranking and selection process over time.

8. PSEUDOCODE

 # Load data

```
squad = load_dataset("squad")
squad_train, squad_test = squad.train_test_split(test_size=0.2)
```

 # Preprocess data

```
def preprocess_function(examples, tokenizer_name):
```

```
  # Tokenize questions and context
  inputs = tokenizer_name(examples["question"], examples["context"], truncation="only_second",
padding="max_length")

  # Find answer start and end positions
  start_positions, end_positions = [], []
  for i, offset in enumerate(inputs.offset_mapping):
    answer = examples["answers"][i]
    start_char = answer["answer_start"][0]
    end_char = start_char + len(answer["text"][0])
    context_start, context_end = find_context_span(offset,
sequence_ids=inputs.sequence_ids(i))

    if not (offset[context_start][0] > end_char or offset[context_end][1] < start_char):
      start_positions.append(find_answer_start(offset, context_start, start_char))
      end_positions.append(find_answer_end(offset, context_end, end_char))
    else:
      start_positions.append(0)
      end_positions.append(0)

  inputs["start_positions"] = start_positions
  inputs["end_positions"] = end_positions
  return inputs
tokenized_squad = {
"train": squad_train.map(preprocess_function, batched=True,
remove_columns=squad_train.column_names),
    "test": squad_test.map(preprocess_function, batched=True,
remove_columns=squad_test.column_names)
}

# Load pre-trained models
xlnet_tokenizer = AutoTokenizer.from_pretrained("xlnet-base-cased")
xlnet_model = AutoModelForQuestionAnswering.from_pretrained("xlnet-base-cased")
```

```
# Train XLNet model
training_args = TrainingArguments(
  output_dir="squad_qa_model_xlnet",
  learning_rate=2e-5,
  per_device_train_batch_size=16,
  per_device_eval_batch_size=16,
  num_train_epochs=5,
  weight_decay=0.01
)

trainer = Trainer(
  model=xlnet_model,
  args=training_args,
  train_dataset=tokenized_squad["train"],
  eval_dataset=tokenized_squad["test"],
  tokenizer=xlnet_tokenizer,
  data_collator=DefaultDataCollator()
)
trainer.train()
trainer.save_model("squad_qa_model_xlnet")

# Answer questions using pipelines
def answer_question(question, context):
  # Answer with XLNet model
  answer = pipeline("question-answering",
model="squad_qa_model_xlnet")(question=question, context=context)
keyword = answer.get('answer')

  # Summarize context with keyword
  summary = summarize_with_keywords(context, keyword)

  # Elongate summary using beam search
  elongated_summary = elongated(context, summary)
  return elongated_summary
```

```
# Functions for text processing (not shown in original code)
def summarize_with_keywords(text, keyword):
  # Filter sentences containing the keyword
filtered_sentences = [sentence for sentence in text.split(".") if keyword in sentence]
return filtered_sentences[0] if len(filtered_sentences) > 0 else ""

def elongated(context, summary):
  # Generate additional text using beam search
  generated_text = bi_generator(context, summary)
  return generated_text.split(".")[0] + "."
```

9. PROJECT DEMONSTRATION

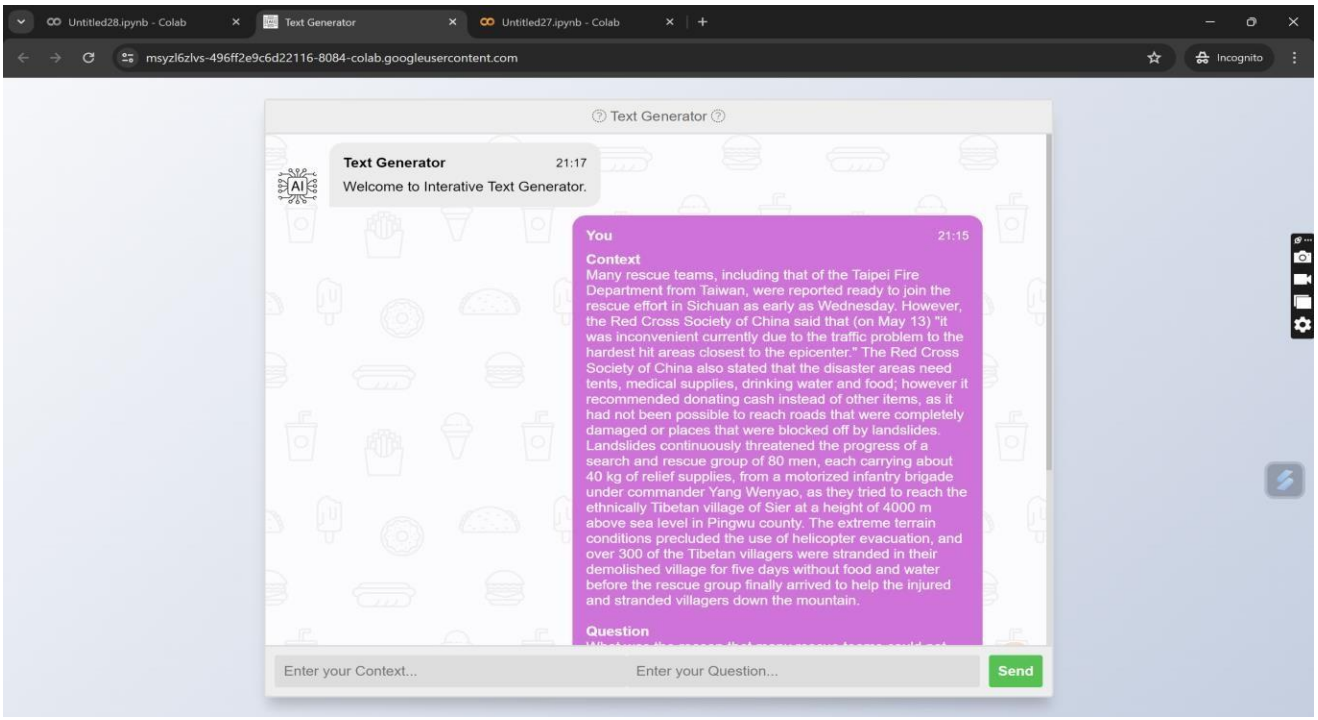

Fig.13 Project Demo

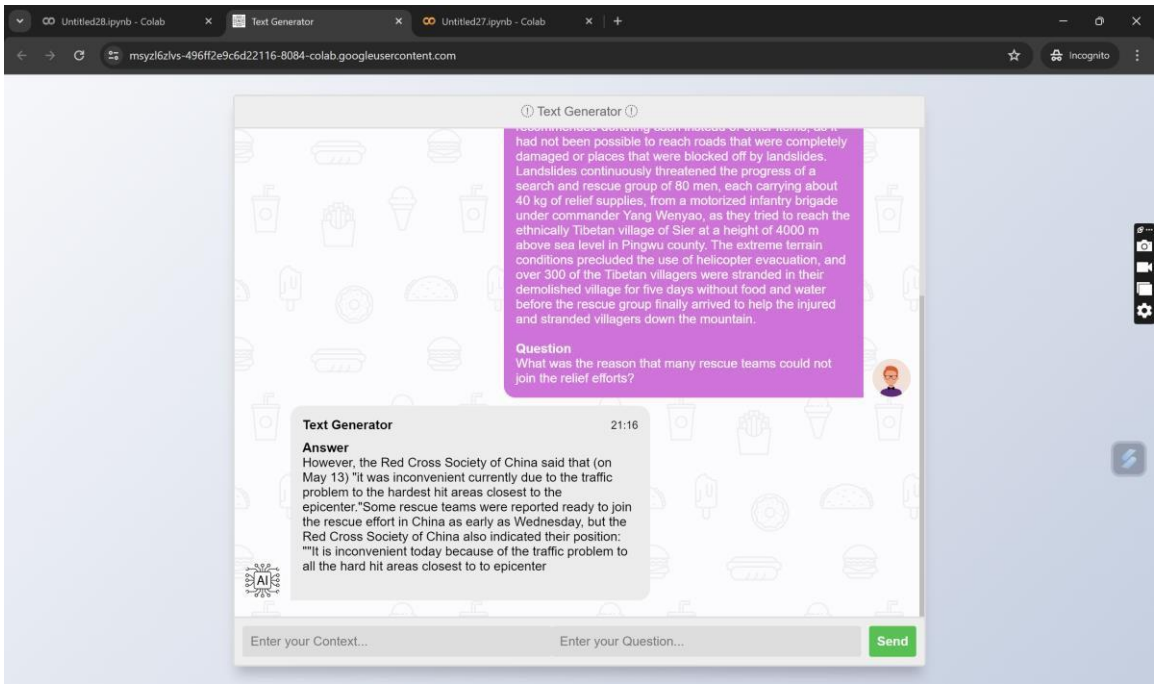

Fig.14 Project Demo

10. RESULTS :-

We successfully implemented our proposed BiGenXL model for generating elongated answers for the given context and Question.The f1 score was used to compare the efficient of Question Answering system.Three models were used namely are BiGenXL,BERT and GPT. The SQuAD dataset was used for evaluating all the efficiency of all three models.The reason for using f1 score was that f1 score is that it provides a good balance between Precision and Recall.

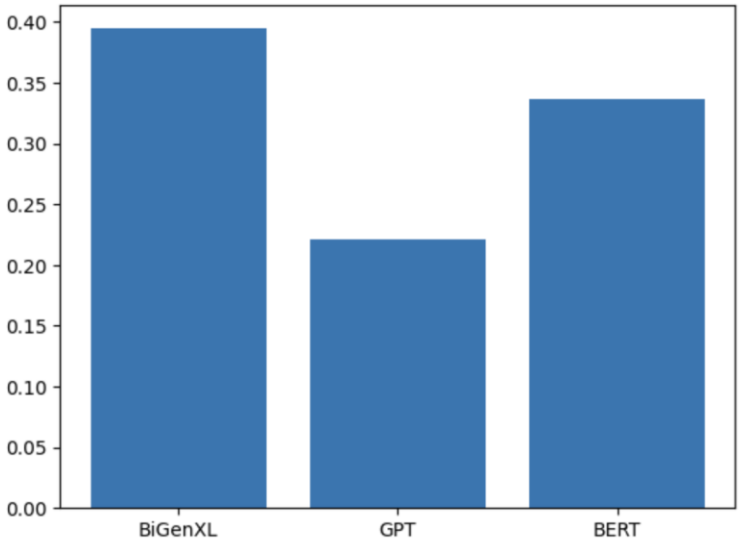

Fig.15 F1 Score Comparision Graph

As showed in the above graph the f1 score of BiGenXL is highest with the with the value of 0.394 as compared to BERT model with the f1 score of 0.336 and GPT model with the f1 score of 0.22.The BiGenXL model considers all possible combinations of the input sequence factorization order which is unique only in XLNet model and helps in providing more coherent text generation.

11. CONCLUSION :-

The successful deployment and testing of the XLNet-based text generation model BiGenXL marks a major step forward in the advancement of NLP. Through careful testing and refinement, we have proven the effectiveness of using bidirectional insight to improve coherence, innovation, and overall output of text generated.

Our findings highlight the transformative power of XLNet to overcome the shortcomings of uni-directional models by capturing more complex contextual dependencies. Thanks to the bidirectional access provided by XLNet, the model not only produces text that is contextually relevant, but also exhibits improved coherence and refined understanding of language. In addition, with the successful implementation of BiGenXL, applications can be developed in many different areas, such as chatbots, FAQs, and content creation. By using bidirectional comprehension, our model can transform interactions with NLP interfaces, allowing for more interactive and contextually relevant communication.

12. FUTURE SCOPE

12.1. Enhanced Factual Consistency and Reasoning

12.1.1. Incorporating knowledge bases and reasoning modules can enable the system to access and leverage factual information beyond the provided context, leading to more reliable and informative answers.

12.1.2. Techniques like factual verification and contradiction detection can be integrated to ensure the generated answers are consistent with established knowledge.

12.2. Addressing Open Ended, Challenging, and Multifaceted Questions

12.2.1. By employing world knowledge and commonsense reasoning, the system can tackle open ended, challenging, or multi-faceted questions that require going beyond simple information retrieval.

12.2.2. The ability to consider multiple perspectives and viewpoints can lead to richer and more nuanced answers.

12.3. Personalization and User-Tailored Responses

12.3.1. The system can personalize answers based on user profiles, preferences, and past interactions, leading to a more engaging and relevant user experience.

12.3.2. Leveraging techniques like user intent recognition and sentiment analysis can further tailor responses to the specific needs and context of the user's query.

12.4. Explainability and Transparency

12.4.1. Techniques like attention visualization can be used to understand the reasoning process behind the generated answer, fostering trust and transparency for users.

12.4.2. By providing explanations for answers, the system can promote user understanding and learning.

12.5. Integration with Conversational AI and Multimodal Interaction

12.5.1. Combining abstractive question answering with dialogue systems can enable more natural and interactive user experiences, allowing for follow-up questions and clarification.

12.5.2. Exploring multimodal interaction (text, voice, images) can provide richer context and a more intuitive user experience.

12.6. Addressing Bias and Fairness:

12.6.1. Developing methods to mitigate bias in the training data and model outputs is crucial to ensure fair and unbiased answers across different demographics and topics.

12.6.2. Encouraging data diversity and incorporating fairness metrics into the evaluation process are essential steps.

12.7. Lifelong Learning and Continuous Improvement

12.7.1. The system can continuously learn and improve by incorporating new information and user feedback.

12.7.2. Techniques like active learning can be employed to focus on areas where the model needs the most improvement.

13. REFERENCES :-

[1] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q., V. (2019, June 19). XLNET: Generalized Autoregressive Pretraining for Language Understanding. arXiv.org. https://arxiv.org/abs/1906.08237

[2] He, X., & Li, V. O. (2021). Show Me How To Revise: Improving Lexically Constrained Sentence Generation with XLNet. Proceedings of the AAAI Conference on Artificial Intelligence, 35(14), 12989-12997

[3] Topal, M. O. (2021, February 16). Exploring Transformers in natural language Generation: GPT, BERT, and XLNET. arXiv.org. https://arxiv.org/abs/2102.08036

[4]Peng, Hao, Ankur P. Parikh, Manaal Faruqui, Bhuwan Dhingra, and Dipanjan Das. "Text generation with exemplar-based adaptive decoding." arXiv preprint arXiv:1904.04428 (2019).

[5] Kumar, Manoj, Abhishek Singh, Arnav Kumar, and Ankit Kumar. "Analysis of automated text generation using deep learning." In *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, pp. 14-18. IEEE, 2021.

[6] Dhall, Ishika, Shubham Vashisth, and Shipra Saraswat. "Text generation using long short-term memory networks." In *Micro-Electronics and Telecommunication Engineering: Proceedings of 3rd ICMETE 2019*, pp. 649-657. Springer Singapore, 2020.

[7] Mukherjee, Swapnanil, and Sujit Das. "Application of transformer-based language models to detect hate speech in social media." *Journal of Computational and Cognitive Engineering* 2, no. 4 (2023): 278-286.

[8] Wang, H., Li, J., Wu, H., Hovy, E., & Sun, Y. (2023b). Pre-Trained Language Models and their applications. *Engineering*, *25*, 51–65. https://doi.org/10.1016/j.eng.2022.04.024

[9]*XLNET: Generalized Autoregressive Pretraining for Language Understanding (1/3)*. (2020, May 12). enJOY. https://machinereads.wordpress.com/2020/05/10/xlnet-generalized-autoregressive-pretraining-for-language-understanding-1-3/

[10] Singh, R. (2023b, March 5). *XLNet model architecture*. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/xlnet-model-architecture/

[11] Ramalepe, S., Modipa, T. I., & Davel, M. (2022). The development of Sepedi Text Generation model using transformers. ResearchGate. https://www.researchgate.net/publication/368712875_The_Development_of_Sepedi_Text_Generation_Model_Using_Transformers

[12] S. Lu, Y. Zhu, W. Zhang, J. Wang, and Y. Yu, "Neural Text Generation:Past, Present and Beyond," in arXiv preprint arXiv:1803.07133., 3 2018.

[13] Fatima, N., Imran, A. S., Kastrati, Z., Daudpota, S. M., & Soomro, A. R. (2022). A Systematic Literature Review on text generation using deep Neural Network models. IEEE Access, 10, 53490–53503. https://doi.org/10.1109/access.2022.3174108

[14] A Novel Data-to-Text Generation Model with Transformer Planning and a Wasserstein Auto-Encoder. (2020, November 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9284477

[15] https://aclanthology.org/D19-5615.pdf

[16] Dixit, R. (2023, May 15). A Comprehensive Review of Transformer models and their Implementation in Machine Translation specifically on Indian Regional Languages. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4449023

[17] Rahali, A., & Akhloufi, M. A. (2023). End-to-End Transformer-Based models in Textual-Based NLP. AI, 4(1), 54–110. https://doi.org/10.3390/ai4010004

[18] Yang, Y., Cao, J., Wen, Y., & Zhang, P. (2021). Table to text generation with accurate content copying. Scientific Reports, 11(1). https://doi.org/10.1038/s41598-021-00813-6

[19] Li, J., Song, H., & Li, J. (2022). Transformer-based question text generation in the learning system. ICIAI. https://doi.org/10.1145/3529466.3529484

[20] https://arxiv.org/pdf/2006.03535

[21] Hu, Z., Chan, H. P., Liu, J., Xiao, X., Wu, H., & Huang, L. (2022, March 17). PLANET: Dynamic Content Planning in Autoregressive Transformers for Long-form Text Generation. arXiv.org. https://arxiv.org/abs/2203.09100

[22] Glazkova, A., & Морозов, Д. А. (2023). Applying Transformer-Based text summarization for keyphrase generation. Lobachevskii Journal of Mathematics, 44(1), 123–136. https://doi.org/10.1134/s1995080223010134

APPENDIX A

Import Libraries

```
!pip install -U accelerate
!pip install -U transformers
!pip install torch
!pip install text_hammer
#!pip install transformers datasets
!pip install datasets
import torch
from datasets import load_dataset
from transformers import AutoTokenizer
from transformers import DefaultDataCollator
from transformers import AutoModelForQuestionAnswering, TrainingArguments, Trainer
import nltk
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import text_hammer as th
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
import spacy
nlp = spacy.load("en_core_web_sm")
from transformers import pipeline
from sklearn.metrics import f1_score
from transformers import XLNetTokenizer, XLNetLMHeadModel
from transformers import BertTokenizer, BertLMHeadModel
```

```
import random
import numpy as np
```

Data Preprocessing

```
def preprocess_function_xlnet(examples):
    questions = [q.strip() for q in examples["question"]]
    #print(questions)
    inputs = tokenizer(
        questions,
        examples["context"],
        max_length=384,
        truncation="only_second",
        return_offsets_mapping=True,
        padding="max_length",
    )

    offset_mapping = inputs.pop("offset_mapping")
    answers = examples["answers"]
    start_positions = []
    end_positions = []

    for i, offset in enumerate(offset_mapping):
        answer = answers[i]
        start_char = answer["answer_start"][0]
        end_char = answer["answer_start"][0] + len(answer["text"][0])
        sequence_ids = inputs.sequence_ids(i)

        # Find the start and end of the context
        idx = 0
        while sequence_ids[idx] != 1:
            idx += 1
        context_start = idx
        while sequence_ids[idx] == 1:
            idx += 1
        context_end = idx - 1

        # If the answer is not fully inside the context, label it (0, 0)
        if offset[context_start][0] > end_char or offset[context_end][1] <
start_char:
            start_positions.append(0)
            end_positions.append(0)
        else:
            # Otherwise it's the start and end token positions
            idx = context_start
            while idx <= context_end and offset[idx][0] <= start_char:
                idx += 1
            start_positions.append(idx - 1)
```

```
            idx = context_end
            while idx >= context_start and offset[idx][1] >= end_char:
                idx -= 1
            end_positions.append(idx + 1)

    inputs["start_positions"] = start_positions
    inputs["end_positions"] = end_positions
    return inputs
```

Model Training

```
training_args = TrainingArguments(
    output_dir="squad_qa_model_xlnet",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    push_to_hub=False,
)

model=model
args=training_args
train_dataset=tokenized_squad_xlnet["train"]
eval_dataset=tokenized_squad_xlnet["test"]
tokenizer=tokenizer
data_collator=data_collator

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_squad_xlnet["train"],
    eval_dataset=tokenized_squad_xlnet["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()
trainer.save_model("squad_qa_model_xlnet")
```

Top K Beam Bidirectional Text Generation

```
def choose_from_top(probs, k=2, sample_size=1):
    ind = np.argpartition(probs, -k)[-k:]
    top_prob = probs[ind]
    # print(tokenizer.decode(ind))
    top_prob = top_prob / np.sum(top_prob) # Normalize
    choice = np.random.choice(k, sample_size, p = top_prob, replace=False)
```

```
    token_ids = ind[choice]
    return token_ids
def candidates_gen(PADDING_TEXT,sent_tokens, candidate=([], 1, []), d='left',
n_candidates=5, topk=20, temperature=5):
  padding_tokens = tokenizer.encode(PADDING_TEXT, add_special_tokens=False)
  mask_tokens = tokenizer.encode('<mask>', add_special_tokens=False)

  model.eval()
  if torch.cuda.is_available(): model.to('cuda') #if we have a GPU
  branch_candidates = []
  cand_tokens = candidate[0]

  if d == 'right':
    #target_id = len(input)
    input = sent_tokens + cand_tokens + mask_tokens

    target_id = -1
    input_ids = torch.tensor(padding_tokens + input).unsqueeze(0)
    #input_ids = torch.tensor([tokenizer.cls_token_id] + padding_tokens + input +
[tokenizer.sep_token_id]).unsqueeze(0)
    perm_mask = torch.zeros((1, input_ids.shape[1], input_ids.shape[1]),
dtype=torch.float)
    perm_mask[0, :, target_id] = 1.0 # Previous tokens don't see last token
  else:
    input = mask_tokens + cand_tokens + sent_tokens

    target_id = -len(input)
    input_ids = torch.tensor(padding_tokens + input).unsqueeze(0)

    perm_mask = torch.zeros((1, input_ids.shape[1], input_ids.shape[1]),
dtype=torch.float)
    perm_mask[0, :, [target_id - i for i in range(100)]] = 1.0 # Mask additional
previos tokens to improve left-side generation

  # We will predict masked tokens
  target_mapping = torch.zeros((1, 1, input_ids.shape[1]), dtype=torch.float)
  target_mapping[0, 0, target_id] = 1.0  # Our right  prediction

  if torch.cuda.is_available():
    input_ids_tensor = input_ids.to("cuda")
    target_mapping_tensor = target_mapping.to("cuda")
    perm_mask_tensor = perm_mask.to("cuda")
  else:
    input_ids_tensor = input_ids
    target_mapping_tensor = target_mapping
    perm_mask_tensor = perm_mask

  with torch.no_grad():
```

```
    outputs = model(input_ids_tensor, perm_mask=perm_mask_tensor,
target_mapping=target_mapping_tensor)
    #outputs = model(input_ids_tensor)

  probs = torch.nn.functional.softmax(outputs[0][0][0]/temperature, dim = 0)
  selected_indexes = choose_from_top(probs.to('cpu').numpy(), k=topk,
sample_size=n_candidates)
  selected_probs = probs[selected_indexes]

  for i,item  in enumerate(selected_indexes):
      the_index = item.item()
      if d == "right":
        new_sent = cand_tokens + [the_index]
      elif d == "left":
        new_sent = [the_index] + cand_tokens

      prob = selected_probs[i].item()
      # add word combinations to branch_candidates in format [sentence, cumulative
probability, all probs]
      branch_candidates.append((new_sent, candidate[1] * prob, candidate[2] +
[prob]))

  return branch_candidates
def beam_gen(PADDING_TEXT,sent_tokens, candidates, depth=5, d='right',
sample_size=2, topk=10, temperature=5):
  beams = candidates[:]
  new_candidates = candidates[:]
  while depth > 0:
    new_candidates = []
    for candidate in candidates:
      for new_candidate in candidates_gen(PADDING_TEXT,sent_tokens, candidate, d,
sample_size, topk, temperature):
        beams.append(new_candidate)
        new_candidates.append(new_candidate)
    #print("Number of beams:", len(new_candidates))
    candidates = new_candidates[:]
    depth -= 1
  # sort candidate beams by a sum of logaryphms of probability of each word in a
beam. Which is equivalet to product of probabilities
  sorted_beams = sorted(new_candidates, key=lambda tup: np.sum(np.log10(tup[2])),
reverse=True)
  return beams, sorted_beams
def bi_generator(PADDING_TEXT,sent, direction, first_sample_size, sample_size,
n_tokens, topk, iterations, temperature):
  sent_tokens = tokenizer.encode(sent, add_special_tokens=False)

  for i in range(iterations):
    if (i % 2 == 0 and direction == 'both') or direction == 'left':
```

```
      #print('>> left side generation')
      candidates = candidates_gen(PADDING_TEXT,sent_tokens=sent_tokens, d='left',
n_candidates=first_sample_size,  topk=topk, temperature=temperature)
      beams, sorted_beams = beam_gen(PADDING_TEXT,sent_tokens, candidates,
n_tokens-1, 'left', sample_size, topk, temperature=temperature)
      topn = len(sorted_beams)//5 if len(sorted_beams) > 4 else len(sorted_beams)
      selected_candidate = random.choice(sorted_beams[:topn])
      sent_tokens = selected_candidate[0] + sent_tokens
      #print(tokenizer.decode(sent_tokens))
    if (i % 2 != 0 and direction == 'both') or direction == 'right':
      #print('>> right side generation')
      candidates = candidates_gen(PADDING_TEXT,sent_tokens=sent_tokens, d='right',
n_candidates=first_sample_size, topk=topk, temperature=temperature)
      beams, sorted_beams = beam_gen(PADDING_TEXT,sent_tokens, candidates,
n_tokens-1, 'right', sample_size, topk, temperature=temperature)
      topn = len(sorted_beams)//5 if len(sorted_beams) > 4 else len(sorted_beams)
      selected_candidate = random.choice(sorted_beams[:topn])
      sent_tokens = sent_tokens + selected_candidate[0]
      #print(tokenizer.decode(sent_tokens))

  return tokenizer.decode(sent_tokens)
```

Model Evaluation

```
from sklearn.metrics import f1_score
predictions = []
references = []
for datapoint in squad["test"]:
  # Extract question, context, and reference answer from each datapoint
  question = datapoint.get("question")
  context = datapoint.get("context")
  reference_answer = datapoint.get("answers") # Assuming "answer" key holds the
reference answer
  reference_answer=reference_answer.get("text")[0]

  #print(question,context,reference_answer)
  # Use the question-answerer pipeline to get the model's prediction
  prediction = question_answerer1(question=question, context=context)
  predictions.append(prediction.get('answer'))
  references.append(reference_answer)

f2 = f1_score(references, predictions, average="weighted")
print("F1 Score:", f2)
```