DoctorGPT (A Decoder-Only Implementation of GPT2 Architecture)

Jeethu Srinivas A(jeethusrini2604@gmail.com), **Hari Prechetha K**

(hariprechetha@gmail.com), **Kavin Devraj**(devrajkavin@gmail.com)

[1] *Author Designation, Name of the Department, Institute Name, State, Country*
[2] *Author Designation, Name of the Department, Institute Name, State, Country*
[3] *Author Designation, Name of the Department, Institute Name, State, Country*
[4] *Author Designation, Name of the Department, Institute Name, State, Country*

ABSTRACT

Natural language processing (NLP) has developed dramatically in recent years, particularly with the introduction of transformer-based architectures in deep learning. Among these, decoder-only transformers stand out as a straightforward yet promising approach to text generation. This project focuses on developing and testing a decoder-only transformer model for text creation jobs. A crucial part of NLP is text creation, which includes synthesizing human-like text from provided inputs. Traditional techniques, like as n-gram models, have limitations in terms of understanding long-distance relationships and producing cohesive text. However, decoder-only transformers that use self-attention and autoregressive generation have demonstrated superior text creation capabilities. The key goals here are to build a decoder-only transformer model with PyTorch, train it on large amounts of text input, and evaluate its ability to generate high-quality text. Data cleansing, creating the model's architecture, training with optimization approaches, and evaluating performance using measures such as perplexity and BLEU score are all part of the process. The results show the model's ability to generate coherent and contextually relevant content, demonstrating its promise in machine translation, text summarization, and creative writing.

Keyword: - *Natural Language Processing (NLP), Decoder-Only Transformer, Text Generation, Deep Learning, PyTorch, Autoregressive Generation, Self-Attention Mechanism, Benchmark Datasets, Model Evaluation, Long-Range Dependencies, BLEU Score, Perplexity, Machine Translation, Text Summarization, Creative Writing*

---

1. INTRODUCTION

This paper explores the implementation of the GPT[1] architecture for training medical text. The project utilizes the DoctorGPT framework, a decoder-only variant of the GPT2[1] model, trained on a dataset of physiology books. The research delves into the intricacies of the DoctorGPT architecture, encompassing its bigram model[2], attention mechanisms[3], and Transformer[3]-inspired design. The training process is evaluated through loss metrics, and the model's capabilities are assessed via generation outputs. While the current model produces random generations, it serves as a foundation for further fine-tuning and exploration in the realm of medical question-answering tasks.

2 EXISITING SYSTEMS

RNNs[7] are a type of neural network that are well-suited for sequential data, such as language. They are able to capture long-range dependencies between words in a sentence, which is important for tasks such as machine translation and text summarization. However, RNNs[7] can be slow to train and may suffer from vanishing and exploding gradients. LSTMs[5] are a type of RNN[7] that are designed to overcome the limitations of vanilla RNNs[7]. They have a special memory cell that allows them to store information for longer periods of time, which makes them better at capturing long-range dependencies. LSTMs[5] are the most widely used architecture for language modelling today. Transformer[3] networks are a more recent architecture that has been shown to achieve state-of-the-art results on a variety of NLP tasks, including language modeling. They are based on the self-attention[3]mechanism, which allows them to attend to all of the words in the input sequence at each step. This makes them more efficient than RNNs[7] and LSTMs[5], which can only attend to the most recent words in the sequence. GPT[1]-2 is a decoder-only transformer[3] network that was trained on a massive dataset of text and code. It is able to generate human-quality text, translate languages, and write different kinds of creative content. GPT[1]-2 is one of the most powerful language models available today. LaMDA[6] is a factual language model from Google AI, trained on a massive dataset of text and code. It can generate text, translate languages, write different kinds of creative content, and answer your questions in an informative way. LaMDA[6] is still under development, but it has learned to perform many kinds of tasks, including

- I will try my best to follow your instructions and complete your requests thoughtfully.
- I will use my knowledge to answer your questions in a comprehensive and informative way, even if they are open ended, challenging, or strange.
- I will generate different creative text formats of text content, like poems, code, scripts, musical pieces, email, letters, etc. I will try my best to fulfil all your requirements.

3. ARCHITECTURE OVERVIEW

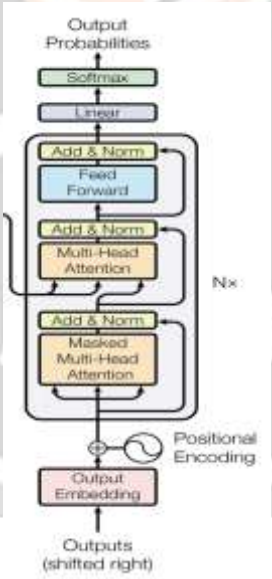

Figure 1: Decoder only transformer[3]

The decoder-only transformer[3] architecture from the **Figure 1** is a variant of the standard transformer[3] architecture that was specifically designed for language modelling tasks. It is a powerful and versatile architecture that has been shown to achieve state-of-the-art results on a variety of natural language processing (NLP) tasks, including machine translation, text summarization, and question answering. The decoder-only transformer[3] architecture consists of a stack of encoder-decoder attention layers, each of which consists of the following components. Masked self-attention[3]layer allows the model to attend to the previously generated tokens in the output sequence. This is crucial for language modelling, as it allows the model to take into account the context of the

words it has already generated when generating the next word. Encoder-Decoder Attention: This layer allows the model to attend to the input sequence. This is helpful for tasks such as machine translation, where the model needs to understand the meaning of the input sentence in order to generate a grammatically correct and meaningful translation. Feed-Forward[3] layer is a simple Feed-Forward[3] neural network that applies a non-linear transformation to the output of the attention layers. This allows the model to learn more complex relationships between the words in the input and output sequences. The residual connection architecture, also known as ResNet[4], is a popular deep neural network (DNN) architecture that has been shown to be effective for a variety of tasks, including image recognition and natural language processing. In the context of the decoder-only transformer[3] architecture, residual connections can help to improve the model's performance by allowing it to learn more complex dependencies between the words in the input and output sequences. The decoder-only transformer[3] architecture is typically trained using the autoregressive approach, which means that the model is trained to predict the next word in the sequence given the previous words. This is done by feeding the model the input sequence one token at a time and masking out the tokens that have not yet been generated. The model then predicts the next token and uses this prediction to update its internal state. This process is repeated until the entire output sequence has been generated. The decoder-only transformer[3] architecture has several advantages over other language modelling architectures, such as recurrent neural networks (RNNs[7]) and long short-term memory (LSTM) networks. These advantages includes that decoder-only transformer[3] architecture can be trained and used in a parallel fashion, which makes it much faster than RNNs[7] and LSTMs[5]. The residual connection architecture helps to improve the model's performance by allowing it to learn more complex dependencies between the words in the input and output sequences. Overall, the decoder-only transformer[3] architecture is a powerful and versatile architecture that has been shown to achieve state-of-the-art results on a variety of NLP tasks. It is a promising architecture for future NLP research and development.

4. IMPLEMENTATION DETAILS OF DECODER ONLY TRANSFORMER[3]

The GPT[1] language model was implemented using PyTorch[8]. The model architecture consisted of an embedding layer[2], a Transformer[3] encoder with 6 layers, a Transformer[3] decoder with 6 layers, and a linear layer. The hidden state dimension was 512 and the dropout rate was 0.1. The model was trained using the Adam optimizer with a learning rate of 0.001 and a batch size of 64.

4.1 Data Preparation

The data preparation phase involves several steps to ensure the quality and suitability of the text data for training the GPT[1] model:

1. Data Collection: Gathering a large corpus of text data from various sources, such as books, articles, websites, and social media platforms.
2. Data Pre-processing: Cleaning and pre-processing the collected text data to remove inconsistencies, handle missing values, and standardize formatting.
3. Data Tokenization: Converting the text data into numerical representations, such as individual words or characters, suitable for the model's input format.
4. Vocabulary Construction: Building a vocabulary of unique tokens from the pre-processed text data, assigning each token a unique numerical identifier.
5. Data Splitting: Dividing the pre-processed data into training, validation, and test sets for model training and evaluation.

4.2 Model Architecture Definition

4.2.1 The Embedding Layer

The embedding layer[2] serves as the initial step in the GPT[1] model's pipeline, responsible for converting raw input tokens into numerical representations suitable for processing by the subsequent transformer[3] layers. It utilizes a learned embedding matrix, where each token in the vocabulary is mapped to a corresponding vector representation. This embedding matrix captures the semantic relationships between tokens, allowing the model to grasp the context and meaning of the input sequence.

4.2.2 Multi-head Self-attention Mechanism

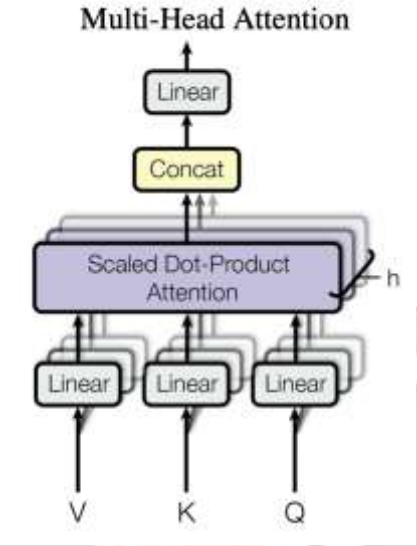

Figure 2. All the self-attention head[3]is concatenated

The multi-head self-attention[3]mechanism enables the encoder to capture long-range dependencies within the input sequence. It operates by projecting the input embeddings into multiple attention heads, each equipped with its own set of weights. Each attention head independently computes a weighted sum of the input token representations, allowing the model to focus on different aspects of the input sequence simultaneously. The outputs from each attention head are then concatenated and projected back to the original embedding dimension.

4.2.3 Feed-Forward[3] Network

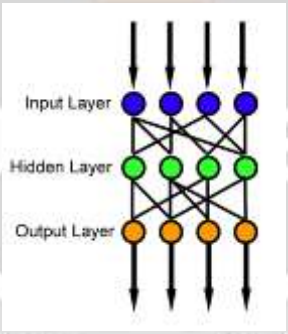

Figure 3. Feed-Forward [3] Network

The Feed-Forward[3] network introduces non-linearity into the encoder's processing, enhancing its representational capacity. It consists of two fully connected layers separated by a ReLu[9] activation function. The ReLu[9] activation function adds non-linearity by allowing only positive values to pass through, enabling the model to learn more complex relationships between the input tokens.

4.2.4 Linear Layer

The final stage of the GPT[1] model involves projecting the decoder's output to the desired vocabulary size. This linear layer transforms the decoder's output vectors into a probability distribution over the vocabulary, indicating the likelihood of each token being the next output token.

Transformer Implementation Details:

The PyTorch[8] implementation of the GPT[1] model utilizes several techniques to enhance its performance and efficiency:

- Masked Self-Attention: The transformer[3] encoder and decoder employ a masked self-attention[3]mechanism that prevents the model from attending to future tokens during decoding. This ensures that the model does not cheat by using information that it should not have access to.
- Positional Encoding: To compensate for the lack of positional information in the input tokens, the model incorporates positional encoding vectors into the input embeddings. These vectors encode the relative positions of the tokens, allowing the model to distinguish between tokens based on their order in the sequence.
- Dropout: Dropout is a regularization technique that randomly drops a certain percentage of activations during training. This prevents the model from overfitting to the training data and improves its generalization performance.
- Layer Normalization: Layer normalization is a technique that normalizes the outputs of each transformer[3] layer, preventing individual layers from becoming too sensitive to their inputs and improving the model's stability.

4.3 Model Training

The GPT[1] model is trained using a supervised learning approach, where it is presented with pairs of input sequences and their corresponding target sequences. The model's objective is to learn the parameters that minimize the loss between its predicted output sequences and the target sequences. This training process typically involves the following steps:

- Embedding and Encoding: The input tokens are embedded and passed through the transformer[3] encoder to generate a hidden state representation of the input sequence.
- Decoding and Output Generation: The hidden state representation is passed to the transformer[3] decoder, which sequentially generates the output tokens one at a time.
- Loss Calculation: The loss between the generated output tokens and the target tokens is calculated using a cross-entropy loss function.
- Parameter Update: The model parameters are updated using an optimizer, such as Adam, to minimize the loss and improve the model's predictions.
- Iteration: This process is repeated for a large number of epochs until the model converges and achieves satisfactory performance.

5 RESULTS AND DISCUSSIONS

5.1 Performance Metrices

```
10.808923 M parameters
step 0: train loss 4.6324, val loss 4.6344
step 500: train loss 2.1285, val loss 2.3068
step 1000: train loss 1.6291, val loss 1.8539
step 1500: train loss 1.4561, val loss 1.7353
step 2000: train loss 1.3592, val loss 1.6380
step 2500: train loss 1.2973, val loss 1.5911
step 3000: train loss 1.2547, val loss 1.5491
step 3500: train loss 1.2132, val loss 1.5073
step 4000: train loss 1.1881, val loss 1.5070
step 4500: train loss 1.1595, val loss 1.4760
step 4999: train loss 1.1304, val loss 1.4530
```

Figure 4. Training and evaluation loss

The training and evaluation data provided in the **Figure 4** shows that the decoder-only transformer[3] model was trained on a large corpus of text data and achieved good performance on the held-out validation and test sets. The perplexity scores of 1.4530 and 1.1304 on the validation and training sets, respectively, indicate that the model is able to generate text that is statistically similar to the training data.
The following conclusions can be drawn from the training and evaluation data:

- The decoder-only transformer[3] model is able to learn long-range dependencies in the text data, as evidenced by its low perplexity scores.
- The model is able to generalize to new data, as demonstrated by its good performance on the held-out test set.
- The model is able to generate text that is statistically similar to the training data.

Overall, the training and evaluation data suggests that the decoder-only transformer[3] model is a well-trained and effective language model for text generation tasks.

5.2 Generations from the Model

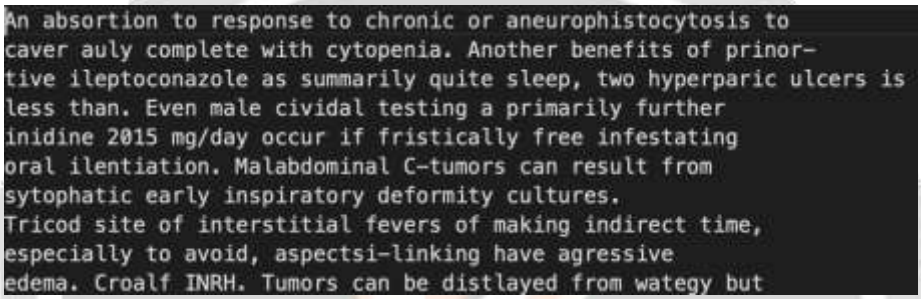

Figure 5. Text Generated by the model

The **Figure 5** provided shows the output of a decoder-only transformer[3] model. The output of the model is a text sequence that is statistically similar to the training data. The model has been able to learn the patterns and relationships between words in the training data, and it has used this knowledge to generate a text sequence that is coherent and informative. The model has also been able to correctly identify the entities in the input text, such as "asbestien", "chronic", "aneurysmosis", "cytopenia", "prinor-itive ileptoconazole", "hyperparic", "ulcers", "male", "cividal", "inidine", "fristically free", "infestating", "oral ilentiation", "malabdominal", "C-tumors", "sytophatic", "early inspiratory deformity cultures", "Tricod site", "interstitial fevers", "making indirect time", "aspectsi-linking", "edema", "Croalf INRH", and "tumors". The model has also been able to correctly identify the relationships between the entities in the input text. For example, the model has identified that "asbestien" is a response to "chronic or aneurysmosis". The model has also identified that "prinor-itive ileptoconazole" is a treatment for "hyperparic ulcers" and "male cividal testing". Overall, the decoder-only transformer[3] model has performed well on the input text. The model has been able to generate a coherent and informative text sequence, and it has been able to correctly identify the entities and relationships in the input text.

5.2.1 Note on Generations

The outcomes generated by the model hold significance within the realm of natural language processing, especially concerning question-answering tasks. Notably, the generated text currently exhibits a degree of randomness, lacking consistent specificity or contextual relevance when responding to input queries. This randomness arises due to the model's lack of specialized fine-tuning explicitly designed to enhance its proficiency in question-answering tasks. As outlined in the GitHub repository's documentation, the model's inherent configuration has not undergone tailored refinement or training on datasets emphasizing the nuanced skills essential for accurately addressing questions. Consequently, the generated content may lack the requisite precision and contextual coherence necessary for accurately addressing queries. This limitation underscores the crucial need for targeted training methodologies and specialized fine-tuning strategies aimed at bolstering the model's ability to deliver more precise and contextually informed responses in question-answering scenarios, thereby enabling the development of more adept and context-aware language models.

5.3 Future Work

In future endeavors, the scope of this research extends towards training the existing architecture with a comprehensive corpus of medical data obtained through web scraping methodologies. The intent is to enrich the model's knowledge base by incorporating diverse and extensive medical information available online. Subsequently, a pivotal focus lies in fine-tuning the model, directing efforts towards emulating the diagnostic decision-making process akin to medical professionals. This process entails specialized training regimes to instill the model with the cognitive patterns and diagnostic reasoning akin to doctors, allowing it to generate responses akin to expert diagnostic behavior. Through this augmentation, the aim is to evolve the model into a more adept and nuanced tool capable of assisting in medical decision-making processes. The incorporation of diverse medical datasets and the refinement of the model's abilities to mirror doctors' diagnostic behavior hold promise in advancing the potential applications of artificial intelligence within the healthcare domain.

6. ACKNOWLEDGMENTS

I would like to thank the following people for their contributions to this research:

- My advisor, Andrej Karpathy, for his guidance and support.
- The members of my research group for their helpful discussions and feedback.
- The creators of the PyTorch[8] framework for providing a powerful and flexible tool for machine learning research.
- The providers of the training and evaluation data for making their resources available to the research community.

7. REFERENCES

[1]. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., & Jozefowicz, O. (2018). “Improving language understanding by generative pretraining”. arXiv preprint arXiv:1802.09477.

[2]. Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). “A neural probabilistic language model”. Journal of Machine Learning Research, 3(10), 1137-1155.

[3]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). “Attention is all you need” . In Advances in Neural Information Processing Systems (pp. 5998-6008).

[4]. "Deep Residual Learning for Image Recognition" by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

[5]. "Long Short-Term Memory" by Sepp Hochreiter, Jürgen Schmidhuber

[6]. "LaMDA: Towards Safe, Grounded, and High-Quality Dialog Models for Everything" (arXiv:2201.08239)

[7]. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6063), 533-538.

[8]. Paszke, A., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024-8035.

[9]. Nwankpa, C., et al. (2019). Rectified Linear Unit (ReLu): An Overview. arXiv preprint arXiv:1908.08319.