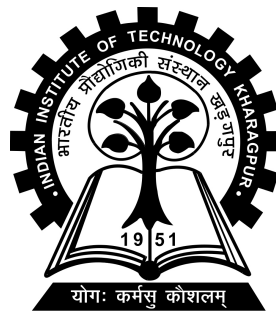


Privacy Enhanced Conversational AI using Large Language Models

Project Part-I (MA57011) report submitted to
Indian Institute of Technology Kharagpur
in partial fulfilment for the award of the degree of
Integrated M.Sc.
in
Mathematics and Computing

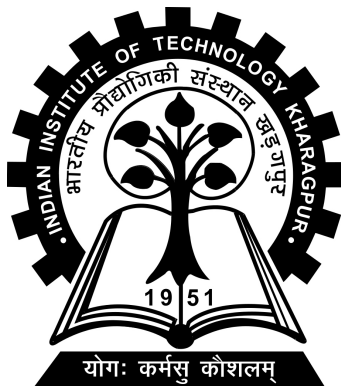
by
Keerti P.Charantimath
(19MA20059)

Under the supervision of
Professor Hari Shankar Mahato



Department of Mathematics
Indian Institute of Technology Kharagpur
Autumn Semester, 2023-24
November 27, 2023

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
KHARAGPUR - 721302, INDIA



CERTIFICATE

This is to certify that the project report entitled “Privacy Enhanced Conversational AI using Large Language Models” submitted by Keerti P.Charantimath (Roll No. 19MA20059) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Integrated M.Sc. in Mathematics and Computing is a record of bona fide work carried out by him under my supervision and guidance during Autumn Semester, 2023-24.

Date: November 27, 2023
Place: Kharagpur

Professor Hari Shankar Mahato
Department of Mathematics
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

DECLARATION

I certify that

- (a) The work contained in this report has been done by me under the guidance of my supervisor.
- (b) The work has not been submitted to any other Institute for any degree or diploma.
- (c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
- (d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: November 27, 2023

Place: Kharagpur

(Keerti P.Charantimath)

(19MA20059)

Acknowledgements

Many amazing people played a big part in making this project a success. I'm really grateful for their help, support, and the good advice they gave me. They've been a huge inspiration and their guidance meant a lot.

I am grateful for the guidance and support provided by Prof. Hari Shankar Mahato, my mentor. This project wouldn't have been achievable without his invaluable help. His encouragement to explore extensively, delve into various research papers, and experiment with diverse ideas has been instrumental.

I wish to express my sincere gratitude to Prof. Adrijit Goswami, the Head of the Department of Mathematics, for providing the necessary facilities and infrastructure. Additionally, I would like to thank Prof. Mahato and Prof. Dipankar Ghosh, our faculty advisors, for their tremendous support and guidance.

Finally, my heartfelt thanks to my family members and friends for their constant love and support.

Contents

Certificate	i
Declaration	ii
Acknowledgements	iii
Contents	iv
Abstract	v
1 Introduction	1
2 Large Language Models	3
2.1 Generative AI	3
2.2 Word to Vector Encoding (Word2Vec)	4
2.3 Sequence to Sequence (Seq2Seq) Model	5
2.3.1 Encoder	5
2.3.2 Decoder	6
2.4 Attention	7
2.5 Transformers	8
2.5.1 Encoder-Decoder Transformers	10
2.5.2 Encoder-only Transformers	10
2.5.3 Decoder-only Transformers	11
3 Implementation and Results	12
3.1 Implementation of LLMs on local system	12
3.2 Fine-tuning	13
3.2.1 Selection of an LLM model	13
3.2.2 Ingestion of custom data	14
3.3 Running LLM on custom files	15
4 Future Work	17
Bibliography	18

Abstract

In today's digital era, our reliance on smartphones and computers has become indispensable. We constantly strive to simplify our lives through technology. Despite the brilliance of technology, conducting transactions online often leaves us with lingering doubts. We wonder about the security of our transactions, whether we can trust the people we're dealing with, and if we'll receive our orders as expected. Additionally, managing tasks across multiple apps or websites can be inconvenient. The remedy for these concerns involves combining two technologies: chatbots and blockchain. Combining the security of blockchain technology and the versatility of chatbots to do various tasks by mere text prompts would potentially be the route to be able to increase the usecase of chatbots exponentially. This will ensure that the chatbot works with utmost privacy due to the inclusion of blockchain technology with the Large Language Models.

Chapter 1

Introduction

Large Language Models (LLMs) are a specific class of artificial intelligence models that are trained on massive amounts of text data [2]. They utilize a technique called deep learning, specifically using architectures like Transformers [12], to process and generate human-like text.

The term “large” in LLMs refers to the sheer size of these models in terms of both the data they are trained on and the number of parameters within the model itself. For instance, models like GPT-3 have billions of parameters, which are the internal variables that the model uses to understand and generate text.

The training process involves feeding the model huge amounts of text data from various sources such as books, articles, websites, and more. During this process, the model learns the patterns, structures, and relationships within the text data, allowing it to understand language at a sophisticated level.

Once trained, these LLMs can perform a wide array of natural language processing (NLP) tasks, including text generation, text completion, translation, summarizing, question answering, and text classification.

Blockchain, on the other hand, is an innovation that is reshaping conventional systems and redefining trust in the digital era. At its core, blockchain represents a decentralized ledger where data is spread across multiple participants, causing a shift in how data is stored, shared, and validated across a network of nodes without the need for intermediaries.

The blockchain technology operates as a series of interconnected information blocks. These blocks are cryptographically linked, creating an indestructible chain that ensures transparency, security, and verifiability of data.

Combining the two powerful technologies would ensure that we achieve privacy and security along with the wide capabilities of chatbots. As mentioned in [4], Deloitte has come up with a chatbot that saves all the warranty cards digitally, reducing the hassle of maintaining hard copies and the worry of losing them. They created this chatbot with the security of blockchain technology ensuring that private data is not lost.

This fusion would allow us to develop the following technologies that make our daily lives easier along with providing privacy and security.

1. **Increasing the capabilities of voice-activated chatbots:** Blockchain's security can enhance voice-activated bots like Apple's Siri, enabling tasks involving sensitive information like credit card details or OTP access. This would simplify tasks such as ordering food or booking tickets through simple voice commands. Additionally, integrating these bots with UPI services could potentially enable voice-command payments.
2. **Chatbots with tailored functionalities to provide financial advice:** Current technology allows for chatbots, as evidenced in [1], to provide financial insights based on training data and trends like spending, income, and investments. Ongoing research in enhancing LLMs could lead us to create chatbots that provide us with financial advice based on our expenditure information or chatbots that could advise which stocks are best to invest in on the basis of our income and savings details.
3. **Chatbots to guide through ERP or Databases:** Blockchain-powered chatbots could enable us to swiftly access specific details within vast ERPs or databases. This is particularly beneficial as sensitive and private data, such as employee records, company codes, or even personal files/folders, can be securely accessed without tediously navigating the complex systems manually.

Chapter 2

Large Language Models

2.1 Generative AI

Generative AI [Fig. 2.1] refers to a category of artificial intelligence that focuses on generating new content or data that resembles what humans might create. This technology involves machines creating original content, such as images, text, music, and even videos, based on patterns learned from large datasets. Large Language Models (LLMs) like GPT-3 (Generative

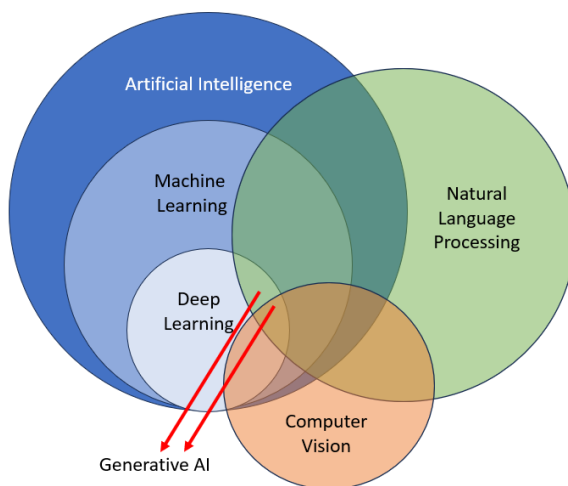


FIGURE 2.1: Hierarchy of Artificial Intelligence

Pre-trained Transformer 3) are a significant part of generative AI. They fall under the category of text-based generative AI. These models are designed to understand language patterns and generate text that mimics human writing. They accomplish this by being trained on vast amounts of text data, and learning the structure, context, and relationships between words.

LLMs can generate human-like text in response to prompts or questions, complete sentences, create stories, translate languages, summarize content, and more. Their generative capabilities stem from their ability to understand the context provided in the input and generate relevant, coherent text based on the learned patterns in the training data.

2.2 Word to Vector Encoding (Word2Vec)

Word to Vector Encoding is a technique used in natural language processing (NLP) to convert words or phrases into numerical vectors. It's based on the idea that words with similar meanings tend to occur in similar contexts within a large corpus of text.

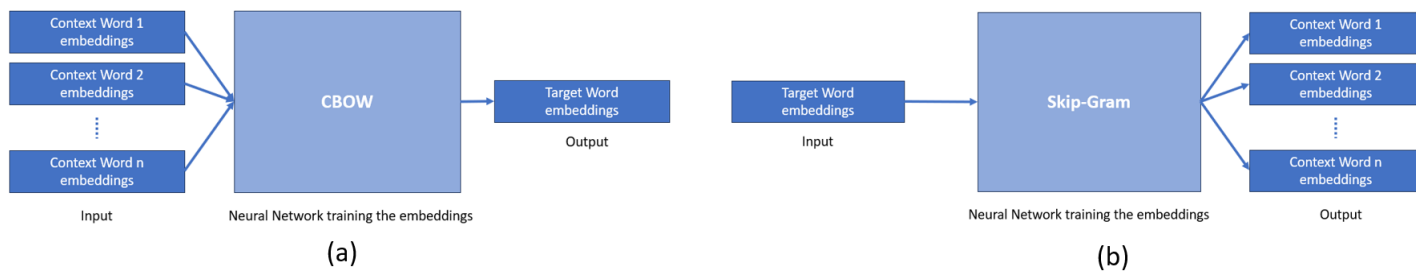


FIGURE 2.2: Word2Vec Models

The popular Word2Vec algorithm [7] has two main models:

1. **Continuous-Bag-of-Words (CBOW):** This model [Fig. 2.2(a)] predicts a target word based on the context words surrounding it. It learns to predict a word from its neighboring words in a sentence.
2. **Skip-gram:** In contrast, the skip-gram model predicts [Fig. 2.2(b)] context words given a target word. It tries to predict the context words within a certain window given a single input word.

Both CBOW and skip-gram are basically neural networks that aim to create word embeddings, which are numerical representations of words in a high-dimensional space. These embeddings are created using **training data consisting of around 3 million words**. Each word has more than **100 embeddings** associated with it. These embeddings capture semantic relationships between words, allowing mathematical operations between word vectors to reveal similarities between words. Word2Vec encoding has been widely used in various NLP tasks such as word similarity calculations, language modeling, text classification, and more, due to its ability to capture semantic meanings and relationships between words.

2.3 Sequence to Sequence (Seq2Seq) Model

The Sequence-to-Sequence (Seq2Seq) model [11] is a neural network architecture used for tasks involving sequences, such as language translation, text summarization, and question answering.

The Seq2Seq model consists of two main components: an encoder and a decoder. It is one of the most basic encoder-decoder models used in natural language processing (NLP).

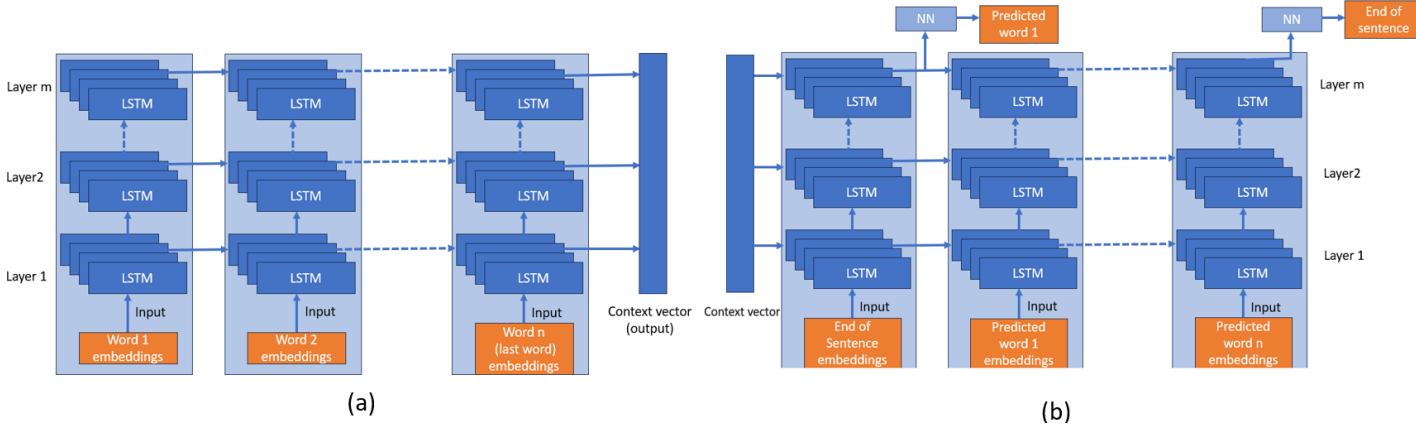


FIGURE 2.3: (a)Seq2Seq Encoder, (b)Seq2Seq Decoder

2.3.1 Encoder

The Seq2Seq Encoder [Fig. 2.3(a)] works in the following manner while translating a sentence from language A to language B.

1. The encoder sequentially takes the language A embeddings of the words of the sentence to be translated as input.
2. Embedding corresponding to each word is passed as input to **about 1000 LSTMs** (Long Short-Term Memory Models) of Layer 1.
3. The output of LSTMs of Layer 1 is taken as input to Layer 2 and this proceeds till the last layer i.e the **4th layer**. Each layer has the same number of LSTMs and each layer has a different set of LSTMs.
4. The output of these LSTMs is taken as input to the corresponding LSTM while processing the next word. The whole set of LSTMs used for all the words is the same.
5. This is repeated until the last word of the sentence to be translated is processed. At the end, we get a context vector as an output. It contains the information of all the words of the sentence to be translated.

2.3.2 Decoder

The Seq2Seq Decoder [Fig. 2.3(b)] works in the following manner while translating a sentence from language A to language B.

1. The decoder takes the context vector created by the encoder along with **EOS (End of Sentence)** embeddings in language B as input.
2. The input is processed by LSTMs in the decoder similar to the encoder. Then, using these outputs of the LSTMs of the 4th layer, it predicts the next word in language B using a Neural Network (NN).
3. This predicted word is input in the next round to get the prediction of the word that comes after it. This is repeated until EOS is reached or a maximum limit is hit. The Neural Network used for predicting each word is same. (During training, the decoder is initiated with the actual words that have to be predicted irrespective of what the model is predicting.)

It is to be noted that the number of layers in the decoder is the same as that of the encoder i.e.

4. Also, the number of LSTMs per layer in the encoder and decoder are the same. However, the set of LSTMs used in the decoder is different from the set of LSTMS used in the encoder. Like in the encoder, even in the decoder, the set of LSTMs used for each word is the same.

Seq2Seq can handle tasks where an input sequence needs to be transformed into an output sequence of varying lengths. It also was one of the first models that could process whole sentences at once, in contrast to previous models like RNN (Recurrent Neural Networks) and LSTM which only have the capability to process word by word. It is safe to say that Seq2Seq has revolutionized the processing of language and has been highly successful in tasks such as sentence translation and text summarization.

While Seq2Seq works well for short-length sequences, but handling long sequences can be challenging due to computational limitations and memory constraints. The model might struggle to maintain dependencies and relationships between distant tokens in lengthy sequences and might lose out on important details of a sentence. Say we need to translate a long sentence starting with “*Don’t*” and if the context vector fails to capture this, then the translated sentence will have the opposite meaning altogether. This can be overcome by introducing “Attention”.

2.4 Attention

In Large Language Models (LLMs), Attention refers to a crucial mechanism that allows the model to understand the relationships between different words in a sequence of text. It enables the model to focus on specific parts of the input text while processing and generating output.

The Attention mechanism in LLMs involves calculating attention scores, which are similarity scores between the word being processed and all the words that would potentially have an influence on it. These scores determine how much attention should be given to each input by the word being processed, capturing the context and semantic relationships effectively.

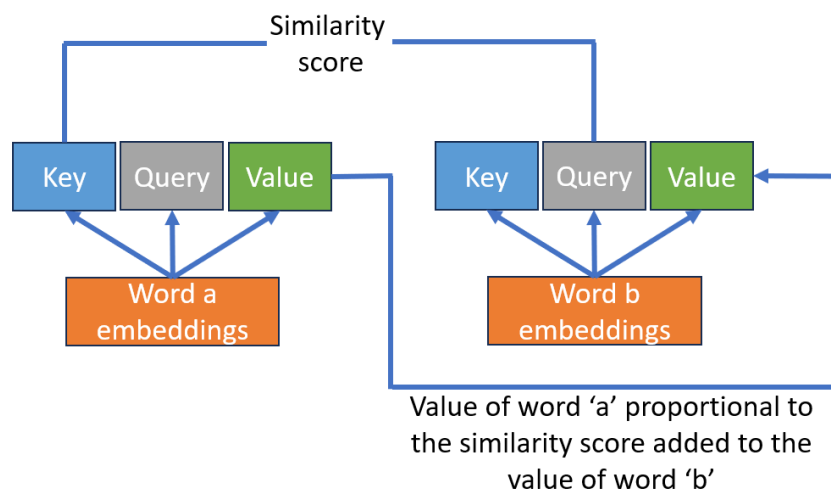


FIGURE 2.4: Attention of *Word a* calculated with respect to *Word b*

In an attention mechanism, there are three primary components [Fig. 2.4]:

1. **Query:** The query is a representation of the word or token for which we want to calculate the relevance or similarity with other words in the sequence. It serves as the point of reference for computing attention scores.
2. **Key:** The key is another representation derived from the input sequence. It signifies the importance or relevance of each word in the sequence concerning the query. Keys help determine how much attention each word should give to the query.
3. **Value:** The value represents the actual information or content associated with each word in the sequence. Values are used in conjunction with attention scores to compute weighted sums, producing the context or output of the attention mechanism. In case of a word in input sentence, attention is calculated with all of the input words including itself and weighted proportions according to respective attention scores of their values are added to the value of word being processed. For words in the output sequence, the process remains

the same. The only difference would be that attention would be calculated between all input words, output words predicted so far and itself.

During the attention computation process, the query is compared or matched with keys to measure the similarity or relevance between the query and each word in the sequence. This comparison generates attention scores that reflect how much focus or attention each word should receive concerning the query. These attention scores are then used to weight the values (the actual content or information associated with each word), producing a context vector or representation that emphasizes relevant information in the input sequence concerning the query.

This attention mechanism is fundamental in LLMs like GPT (Generative Pre-trained Transformer) models as it enables them to understand the context of the input text at different levels of granularity, capturing dependencies between words and generating more coherent and contextually relevant text.

2.5 Transformers

Transformers are a type of neural network architecture that has revolutionized natural language processing (NLP) tasks, especially in Large Language Models (LLMs). They excel in understanding and generating human-like text.

Key components of transformer architectures, as proposed in [12] include :

1. **Self-Attention Mechanism:** This is a critical component allowing the model to weigh the significance of different words in a sequence. It calculates attention scores between all word pairs in a sequence to understand the relationships and dependencies between them. In case of a word in input sentence, attention is calculated with all of the input words including itself. In case of a word in output sentence, self-attention is calculated with all the output words generated so far (which also includes the word being processed). Using only the words seen so far for calculation attention values is known as “*masked attention*”.
2. **Encoder-Decoder Attention Mechanism:** In addition to the self-attention values, the decoder has another set of attention values known as the encoder-decoder attention. This is the attention calculated between all input words and the output being processed. This helps us calculate which words in the input are close to the words being processed not just in terms of the meaning but also in terms of the syntactic and grammatical rules.
3. **Multi-Head Attention:** Transformers use multiple attention heads to capture diverse patterns and relationships within the input data. Basically, in place of a single attention

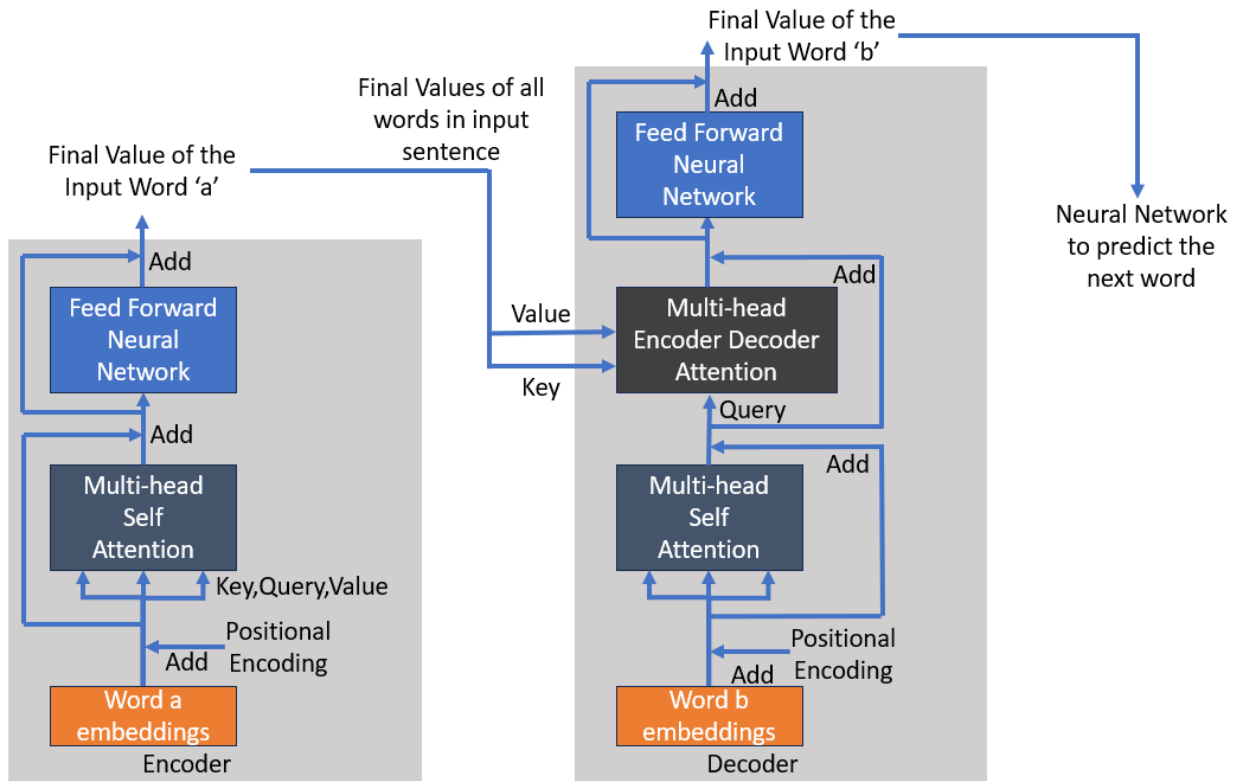


FIGURE 2.5: Transformer architecture

value, multiple attention values for a single pair of words are calculated and used. This allows the model to focus on different aspects of the same word simultaneously.

4. **Positional Encoding:** To maintain the sequence order of words, positional encoding is added to the input embeddings. Positional encoding is done generally using an alternating sequence of sine and cosine waves of different magnitudes. The values of these waves corresponding to the position of the word in the sentence make up for its positional encoding. It helps the model understand the positions of words in a sentence or sequence.
5. **Feedforward Neural Networks:** Transformers consist of feedforward neural networks (also just known as neural networks) that process information from the attention layers, helping in feature extraction and learning complex patterns.
6. **Encoder-Decoder Architecture:** Some transformer variants include both encoder and decoder components, where the encoder processes the input data and the decoder generates the output sequence based on the encoded information.

Fig. 2.5 shows the steps involved in the creation of a final value in the encoder as well as the decoder, and the generation of output. As the processing of each word in the input sentence can be done independently, transformers can run on parallel processing. If we calculate the key, query, and values for each word of the input, the whole input can be processed at the same time and hence transformers are very fast in processing.

Transformers have significantly improved language understanding and generation due to their ability to handle long-range dependencies and capture contextual information effectively. They allow for parallelization, enabling faster training and inference compared to previous sequential models. This architecture has been pivotal in the development of powerful language models like GPT-3, BERT, and others, enabling them to excel in various NLP tasks.

2.5.1 Encoder-Decoder Transformers

There are a few developed LLMs that specifically follow variations of the encoder-decoder architecture based on the use case they were built for. One such is the **Text-to-Text Transfer Transformer** [9] also known as **T5**. T5 unifies various natural language processing tasks by converting them into a text generation or transformation task. It follows a single “text-in, text-out” approach, meaning both input and output are in textual format. This model is explicitly designed based on an encoder-decoder architecture. It has shown versatility in various language understanding and generation tasks.

Another encoder-decoder model-based LLM is **BART - Bidirectional and Auto-Regressive Transformer** [6]. BART is a denoising autoencoder model that uses a sequence-to-sequence architecture, incorporating both encoder and decoder transformers. It’s effective in tasks like text generation, language understanding, and summarization.

2.5.2 Encoder-only Transformers

Encoder-only Transformers refer to a specific architecture within transformer-based models that comprises solely the encoder component without the inclusion of a decoder. Unlike traditional encoder-decoder architectures used in sequence-to-sequence tasks, these models focus solely on encoding input sequences and don’t generate output sequences directly.

Encoder-only transformers, like **BERT - Bidirectional Encoder Representations from Transformers** [5] and its variants, specialize in a range of natural language understanding (NLU) tasks. These models excel in tasks that involve understanding and processing textual information such as text classification, named entity recognition (names of persons, organizations, dates, etc.), answer extraction (extracting the portion of the training data that contains the answer of the question posed), text similarity, paraphrase identification, etc.

2.5.3 Decoder-only Transformers

Decoder-only transformers, in contrast to encoder-decoder architectures, focus solely on the generation of output sequences based on context vectors or representations provided to them. These models are primarily used for autoregressive generation tasks, where the goal is to generate sequential output one step at a time.

Decoder-only models use only the decoder to process the input sentence and the same decoder is used to predict the output word by word. It uses *masked-self attention* in both cases, i.e., the self-attention values are calculated with only the words seen so far. For the input string, this means the attention is calculated only with the words before the current word including itself. For the output string, the attention values are calculated with the input string words and the words generated so far including itself. These transformers start with an initial input, often a special token (e.g., *start*), to kickstart the generation process.

Generative Pre-trained Transformer Series (GPT-3, GPT-2) [3] [8], are examples of LLM that use the decoder-only architecture. These models use autoregressive language modeling to generate coherent and contextually relevant text based on given prompts or context. These models specialize in text generation, text completion, summarization, question answering, and language translation.

Chapter 3

Implementation and Results

3.1 Implementation of LLMs on local system

The objective of MTP-1 was to intensively study the architecture and principles behind the building of LLMs. On the execution front, my first aim was to locally execute and interact with pre-trained Large Language Models (LLMs) using Python libraries without using APIs for the online version LLMs.

To achieve the same, these steps were followed:

1. **Environment Setup:** Installed all necessary Python packages such as transformers, torch, etc. to work with LLMs.
2. **Selecting a Pre-trained Model:** Chose some of the latest pre-trained LLMs that worked well on personal files.
3. **Initializing the Model:** Loaded the pre-trained LLMs using the chosen library and initialized it for inference and fine-tuning.
4. **Tokenization and Input Handling:** Tokenized input text or sequences according to the model's requirements. This basically means that I wrote the code that prepared the input data in a format suitable for the chosen LLM.
5. **Inference and Generation:** Used the model to perform inference and generate text as a response to the basic greeting *'hi'*.
6. **Handling Outputs and Stopping Criterion:** Wrote the code in such a way that the model took inputs from the terminal and gave outputs in the terminal itself until *'exit'* prompt is input.

7. **Logging:** While running, made sure that any prompt information such as loading time, evaluation time, and total response time was displayed along with the output of the input prompt in the terminal.
8. **Testing and Validation:** Tested the script with sample inputs like ‘*hi*’ and other general knowledge questions to validate the functionality and correctness of the LLM’s output

This outlines the process and structure of the script I designed to run pre-trained LLMs locally in Python, focusing on the fundamental steps to use these models for text-related tasks.

3.2 Fine-tuning

I aimed to create an LLM that could read, understand and answer questions based on custom personal files. To achieve this, the LLM model with the best comprehensive capacities had to be selected, ingestion of data from the files had to be conducted and the best strategies/algorithms for the same had to be discovered and implemented.

3.2.1 Selection of an LLM model

An empirical investigation was conducted to compare the performance of the contemporary LLMs, Llama-2, Vicuna-7, and Guanaco-7 [Table 3.1], across CPUs and GPUs. The study focused on evaluating their response accuracy and response time for the same input ‘*hi*’, aiming to discover the respective advantages and disadvantages of each model.

TABLE 3.1: Comparative performances of different LLMs

LLM	CPU/GPU	Response Time (in sec)
Vicuna-7	CPU	84.983
	T4-GPU	crashed
	V100-GPU	crashed
	A100-GPU	1.74
Llama-2	CPU	86.923
	T4-GPU	3.85
	V100-GPU	3.26
	A100-GPU	3.19
Guanaco-7	CPU	78.18
	T4-GPU	crashed
	V100-GPU	crashed
	A100-GPU	5.19

It was observed that Vicuna-7 and Guanaco-7 which are built on the Llama model work very fast and give good responses to greetings like ‘*hi*’. Their downside is that they work well mostly

on CPUs only and are fine-tuned to maximize performance specifically on conversational tasks. This involves training on dialog datasets to optimize conversational metrics. Llama-2, on the other hand, is trained more generally for text, not specialized for dialog. In our case, we require the model to deeply understand the context of data it is been fine-tuned to. Although Llama-2 is a bit slower than the other two models, it works very well with GPUs and can understand the context of the data better.

Given our use case's emphasis on the model's contextual understanding of custom data rather than its conversational abilities, I proceeded with **Llama-2**. Additionally, this model showcases rapid response times when paired with GPUs, thereby outweighing the advantages offered by other models in this specific scenario.

3.2.2 Ingestion of custom data

The algorithm to ingest data from custom documents [Fig. 3.1] accepts the data on text, PDF (with selectable text) and CSV format. The model I developed didn't have OCR capabilities, so pictures or PDFs with embedded pictures of text cannot be used currently.

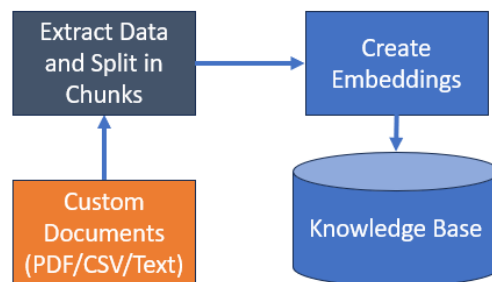


FIGURE 3.1: Ingestion algorithm

The data from the required custom documents is then extracted one document at a time and divided into various chunks. The various chunking strategies [10] considered were:

1. **Fixed-size Chunking:** This standard and direct method for chunking involves determining the token (words along with symbols like fullstop) count in each chunk, with the possibility of overlap. Overlapping chunks maintain semantic context. Fixed-sized chunking, typically preferred, is computationally efficient and straightforward, not relying on complex NLP libraries like other methods.
2. **Recursive Chunking:** Recursive chunking hierarchically breaks down input text by iteratively using separators. If the initial splitting doesn't create desired-sized chunks, it repeats with different separators until the size or structure is met. Though chunks may not be precisely the same size, they aim for similarity.

3. **Sentence Splitting:** This kind of chunking creates chunks in such a way that each chunk has one complete sentence. Generally, no overlapping is considered, hence losing out of semantic context.

The chunking strategy that I chose was **Recursive Chunking**. Recursive chunking iteratively adjusts its splitting criteria, accommodating diverse text structures or varying semantic contexts. This adaptability allows it to handle texts with irregular patterns or complexities more effectively than fixed chunking. Adding overlapping to it would also allow the model to learn semantic context. Smaller chunks grasp finer semantic details and larger chunks like 512 or 1024 tokens maintain extensive context. As overlapping was used to grasp the semantic details, the **overlapping size was chosen to be 20 tokens** (words along with special characters are collectively known as tokens). On the other hand, we required the chunks to grasp the context, so the chosen **chunk size was 1000 tokens**. Both these parameters were chosen through trial and error.

Once chunks are created, they are used to create their respective embeddings. The algorithms that were considered for embedding creation were - the Instructor algorithm, the E5 algorithm, and the MiniLM Algorithm. The Instructor algorithm takes the most time but gives very accurate results. The E5 algorithm is faster than the Instructor algorithm but does not produce as accurate results. The MiniLM Algorithm is the fastest but gives the least accurate results. As the requirement of the project is to create a model that has a high contextual understanding and thus gives the most accurate outputs, I went with the **Instructor algorithm**. The created embeddings are then saved in a database along with their respective chunks of text.

3.3 Running LLM on custom files

Once the database is created after ingesting the custom documents, we can run queries on the LLM that are specifically related to those custom documents.

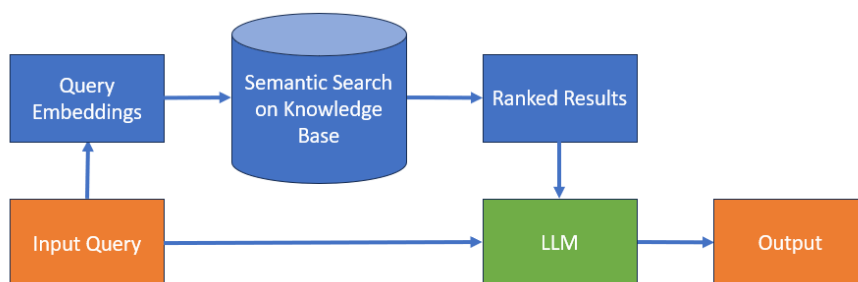


FIGURE 3.2: Algorithm to run an LLM on custom files

As seen in Fig. 3.2, the algorithm to run LLM on custom files works in the following steps:

1. Once a query/ question is input, embeddings are created for the question using the same algorithm as used during the ingestion (i.e. Instructor algorithm in our case).
2. These embeddings are then used to run a semantic search on the database where the ingested embeddings and chunks are stored.
3. Based on the semanticity of the query embeddings, a bunch of ranked results that are close to the query embeddings are input to the LLM model (which in our case is LLama-2).
4. The LLM generates output based on the input from the query and the semantic search on the ingestion database.

As an example, I have trained my model on my resume. The queries give very accurate outputs with minimal hallucination as seen in Fig. 3.3.

```
> Question:
Keerti belongs to which country

> Answer:
Based on the provided context, Keerti P. Charantimath belongs to India.

> Question:
What are her education details

> Answer:
As an assistant, I can provide you with Keerti P. Charantimath's education details based on the provided context.
Here are the details of Keerti's education:
* Degree: Integrated M.Sc with B.Sc Hons.
* Institution: Indian Institute of Technology Kharagpur
* Year of completion: 2024
* CGPA/Percentage: 9.23/10.00

> Question:
where did keerti do her last internship

> Answer:
Based on the provided context, Keerti P. Charantimath did her last internship at Microsoft as a Software Engineering Intern.

> Question:
Where all has keerti studied?

> Answer:
Based on the provided context, Keerti P. Charantimath has studied at the following institutions:
* Indian Institute of Technology Kharagpur (IIT Kharagpur)
* MGVM's PU College, Belgaum (for her Class 12 board exams)
* KLE's International School, Belgaum (for her Class 10 board exams)
She has also participated in various cultural and art competitions at IIT Kharagpur, including the Inter-IIT Cultural Meet,
```

FIGURE 3.3: Results produced on training the model on my resume

Chapter 4

Future Work

As a part of my MTP-2, my intention is to research extensively on blockchain technology, delving into its theoretical intricacies and potential applications. On the implementation front, I aim to integrate blockchain seamlessly into the existing codebase of the LLM, thereby transforming it into a secure and privacy-centric chatbot.

At the same time, I also aim to make the LLM part of these chatbots even better. I want to improve how well they work, how accurate they are, and how good they are at their job. To do this, I would use the newest and best ways to tweak and make them even smarter. The main goal is to make these language models really good, staying up-to-date with all the latest improvements in this area and making them really good at understanding and using language in different ways.

This comprehensive plan encompasses two pivotal objectives: firstly, integrating the security attributes of blockchain into the LLM-based chatbot system; and secondly, refining the language model's precision and performance. This amalgamation of blockchain technology with advanced LLM methodologies aims to culminate in the development of a resilient, privacy-centric chatbot. This bot will boast enhanced language comprehension capabilities, meeting the evolving requisites of secure and effective conversational AI systems.

The potential of this chatbot extends to its ability to be trained using private local databases and large-scale ERP systems. It can serve as a robust tool, adept at guiding us through complex and intricate details within these systems.

Bibliography

- [1] M. S. I. Bhuiyan, A. Razzak, M. S. Ferdous, M. J. M. Chowdhury, M. A. Hoque, and S. Tarkoma. Bonik: A blockchain empowered chatbot for financial transactions. *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1079–1088, 2020.
- [2] M. U. Bilal, A. Shah, A. Muneer, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, M. A. Al-Garadi, J. Wu, and S. Mirjalili. Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects. *techRxiv*, 2023.
- [3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.
- [4] S. T. ChangSheng. How will blockchain and chatbots be our everyday future. *Medium*, 2018.
- [5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.
- [6] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. rahman Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*, 2019.
- [7] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.
- [8] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

-
- [9] C. Raffel, N. M. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2019.
 - [10] R. Schwaber-Cohen. Chunking strategies for llm applications. *Pinecone*.
 - [11] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *ArXiv*, abs/1409.3215, 2014.
 - [12] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017.