

# Privacy Enhanced Conversational AI using Large Language Models

Master's Thesis Project report submitted to

Indian Institute of Technology Kharagpur

in fulfilment for the award of the degree of

Integrated M.Sc.

in

Mathematics and Computing

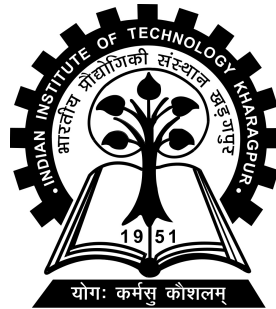
by

Keerti Panchakshari Charantimath

(19MA20059)

Under the supervision of

Professor Hari Shankar Mahato



Department of Mathematics

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

May 1, 2024

## 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: May 1, 2024

Place: Kharagpur

(Keerti Panchakshari 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 support.

# Contents

Declaration	i
Acknowledgements	ii
Contents	iii
Abstract	iv
<b>1 Introduction</b>	<b>1</b>
<b>2 Large Language Models</b>	<b>3</b>
2.1 Generative AI . . . . .	3
2.2 Word to Vector Encoding (Word2Vec) . . . . .	4
2.3 Sequence to Sequence (Seq2Seq) Model . . . . .	5
2.4 Attention . . . . .	6
2.5 Transformers . . . . .	7
<b>3 InterPlanetary File System</b>	<b>10</b>
3.1 Principles and Components of IPFS . . . . .	11
3.2 Use Cases of IPFS . . . . .	11
3.3 Challenges and Limitations of IPFS . . . . .	12
<b>4 Cryptographic Methods</b>	<b>13</b>
4.1 Symmetric Key Encryption . . . . .	14
<b>5 Implementation and Results</b>	<b>16</b>
5.1 Implementation of LLMs on local system . . . . .	16
5.2 Fine-tuning of LLM model . . . . .	17
5.2.1 Selection of an LLM model . . . . .	17
5.2.2 Ingestion of custom data . . . . .	18
5.3 Encryption and Uploading to IPFS . . . . .	19
5.3.1 Selection of encryption algorithm . . . . .	19
5.3.2 Encryption and Uploading of the Knowledge Base . . . . .	20
5.4 Running LLM on custom files . . . . .	20
5.4.1 Downloading and decryption of the database . . . . .	20
5.4.2 Running LLM on custom Database . . . . .	21
5.5 Results . . . . .	22
<b>6 Conclusion</b>	<b>24</b>
Bibliography	25

# *Abstract*

In this era of ubiquitous data sharing, privacy and security concerns are paramount, particularly when it comes to personal data. This thesis proposes a novel approach to address these concerns through the development of a Large Language Model (LLM) based chatbot model designed for on-the-go assistance, leveraging custom data training, IPFS deployment, and robust cryptography for encryption.

The proposed chatbot model serves as a personalized assistant capable of handling user queries while ensuring the confidentiality and integrity of sensitive information. Unlike conventional chatbots, this model prioritizes privacy by training on custom data, allowing users to retain control over their personal information. By leveraging IPFS (InterPlanetary File System), the model becomes easily accessible from anywhere, ensuring seamless deployment across various platforms and devices.

Furthermore, to fortify data protection, the model integrates advanced cryptographic techniques for encryption. This encryption mechanism safeguards personal data from unauthorized access, mitigating risks associated with data breaches or leaks. Despite the stringent security measures, the model maintains flexibility in extracting nuances from encrypted data, enabling efficient query resolution without compromising privacy.

Overall, the proposed chatbot model represents a significant advancement in on-the-go assistance, offering a balance between personalized service and robust data security. By combining custom data training, IPFS deployment, and cryptography, it provides users with a trusted platform for accessing assistance while safeguarding their personal information in an increasingly digital landscape.

# Chapter 1

## Introduction

Large Language Models (LLMs) are a specific class of artificial intelligence models that are trained on massive amounts of text data [4]. They utilize a technique called deep learning, specifically using architectures like Transformers [13], 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.

On the other hand, The InterPlanetary File System (IPFS) has revolutionized the way data is stored, accessed, and shared in the digital realm. Unlike traditional centralized systems, IPFS operates on a decentralized architecture, where files are addressed based on their content rather than their location.

This innovative approach offers several benefits for chatbot deployment. Firstly, IPFS ensures robustness and reliability by distributing data across a network of nodes, eliminating single points of failure. This means that even if some nodes go offline, the data remains accessible, ensuring uninterrupted access. Furthermore, by distributing data across multiple nodes and

encrypting communications, IPFS mitigates the risks associated with centralized data storage, such as data breaches and unauthorized access. This decentralized approach empowers users to have greater control over their data, aligning with the privacy-centric requirement of modern world.

Combining the two powerful technologies along with industry-standard encryption techniques would ensure that we achieve privacy and security along with the wide capabilities of chatbots. As mentioned in [6], 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 use of blockchain technology which is a peer-to-peer network like IPFS that ensures 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. **Chatbots to guide through ERP or Databases:** IPFS-powered chatbots built using cryptographic techniques and LLMs 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.
2. **Increasing the capabilities of voice-activated chatbots:** IPFS and cryptography can greatly enhance the security of voice-activated bots like Apple's Siri, especially for tasks involving sensitive data such as credit card details or OTP access. By leveraging IPFS for decentralized storage and cryptography for encryption, these bots ensure robust security and privacy. This enables streamlined tasks like food ordering or ticket booking via simple voice commands. Integration with UPI services could also enable secure voice-command payments, offering users a convenient and safe transaction experience.
3. **Chatbots with tailored functionalities to provide financial advice:** Current technology allows for chatbots, as evidenced in [3], 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.

# 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

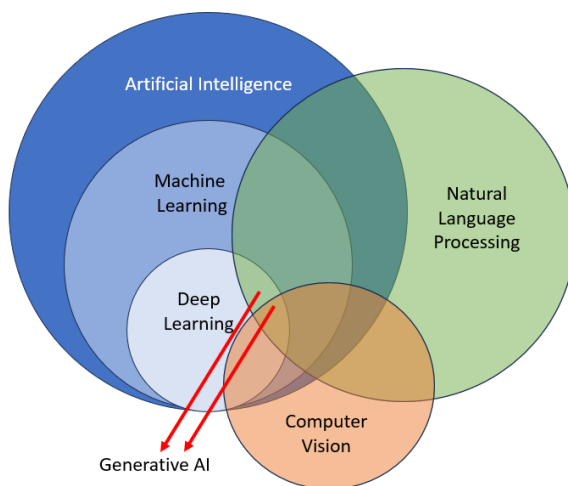


FIGURE 2.1: Hierarchy of Artificial Intelligence

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. This makes it easy to process the words in an algorithmic setup.

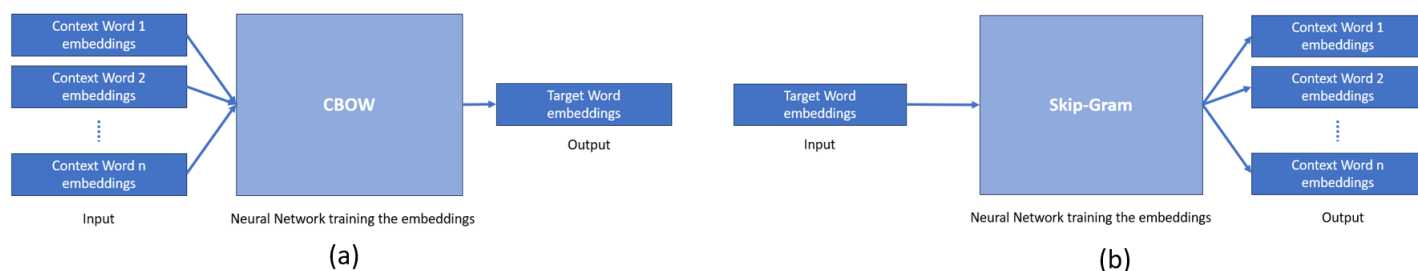


FIGURE 2.2: Word2Vec Models

The popular Word2Vec algorithm [9] 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 which are generally neighboring words in a sentence.
2. **Skip-gram:** In contrast, the skip-gram model predicts [Fig. 2.2(b)] context words within a certain window given a target word.

Both CBOW and skip-gram are neural networks that generate word embeddings which are numerical representations of words in a high-dimensional space. Using training data of around 3 million words, each word is associated with over 100 embeddings. These embeddings capture semantic relationships, enabling mathematical operations between word vectors to uncover similarities between words.

## 2.3 Sequence to Sequence (Seq2Seq) Model

The Sequence-to-Sequence (Seq2Seq) model [12] 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). The Seq2Seq

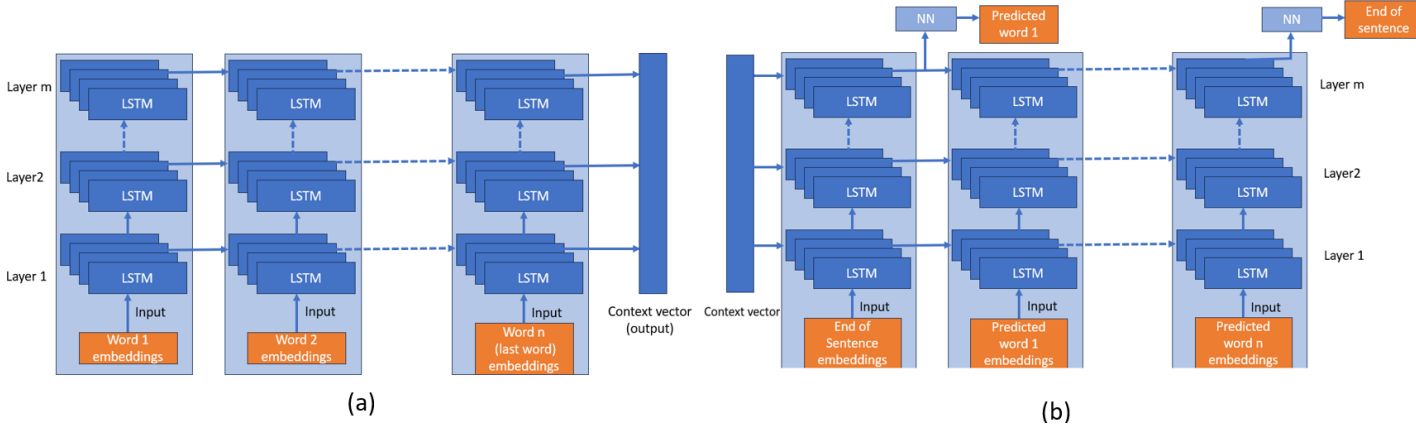


FIGURE 2.3: (a)Seq2Seq Encoder, (b)Seq2Seq Decoder

Encoder and Decoder work in the following manner while translating a sentence from language A to language B:

1. **Encoder:** The encoder [Fig. 2.3(a)] takes word embeddings of the input sentence in language A and passes them through about 1000 LSTMs in each of the  $m$  layers. Each layer has its own set of LSTMs. The output of these LSTMs serves as input to the corresponding LSTM in the next layer. This process continues until all words in the sentence are processed, resulting in a context vector containing information about the entire sentence.
2. **Decoder:** The decoder receives the context vector from the encoder and EOS embeddings in language B as input. LSTMs in the decoder process this input, and using their outputs from the  $m$ th layer, a Neural Network predicts the next word in language B. This prediction process continues iteratively until either EOS is reached or a maximum limit is hit. During training, the decoder is initialized with the actual words to be predicted regardless of the model's predictions.

In the Seq2Seq model, the decoder has the same number of layers and LSTMs per layer as the encoder. However, the decoder uses a different set of LSTMs compared to the encoder. Similar to the encoder, the decoder employs the same set of LSTMs for each word.

Seq2Seq revolutionized language processing by enabling the transformation of input sequences into output sequences of varying lengths. Unlike previous models such as RNN and LSTM, which processed words individually, Seq2Seq can handle entire sentences at once. Its success in tasks like sentence translation and text summarization highlights its pivotal role in advancing natural language processing.

Seq2Seq excels with short sequences but faces challenges with longer ones due to computational and memory constraints. Maintaining dependencies in lengthy sequences can be difficult, potentially leading to the loss of important details. For instance, a mistranslation of a negation like “Don’t” could completely alter the sentence’s meaning. Introducing “Attention” mechanisms helps address these issues by allowing the model to focus on relevant parts of the input sequence, enhancing its performance with longer sequences.

## 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.

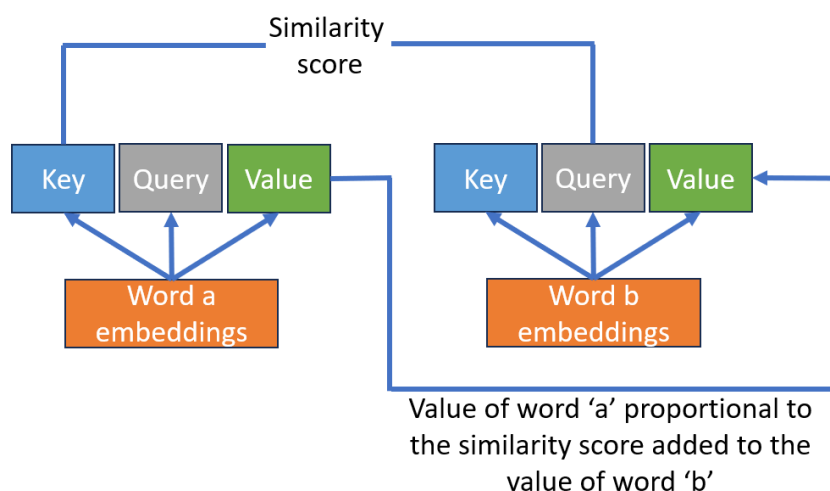


FIGURE 2.4: Attention of *Word a* calculated with respect to *Word b*

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:** Values represent the information or content associated with each word in a sequence, used with attention scores to compute weighted sums and generate the context or output of the attention mechanism. For input words, attention considers all input words, including itself, and adds weighted proportions of their values to the current word being processed. Similarly, for output words, attention is calculated between all input words, output words predicted so far, and the word itself, facilitating the generation of context for each word in the sequence.

During attention computation, the query is compared to keys to measure similarity, generating attention scores reflecting the relevance of each word to the query. These scores weight values, emphasizing relevant information in the input sequence, and produce a context vector highlighting key details.

This attention mechanism is fundamental in LLMs like GPT 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 [13] include :

1. **Self-Attention Mechanism:** This is a crucial mechanism allowing models to weigh the significance of words in a sequence by calculating attention scores between word pairs. This is done through masked attention, where attention values are computed considering only words seen so far. It enables understanding of relationships and dependencies within sequences, aiding tasks like language translation and text generation.
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.

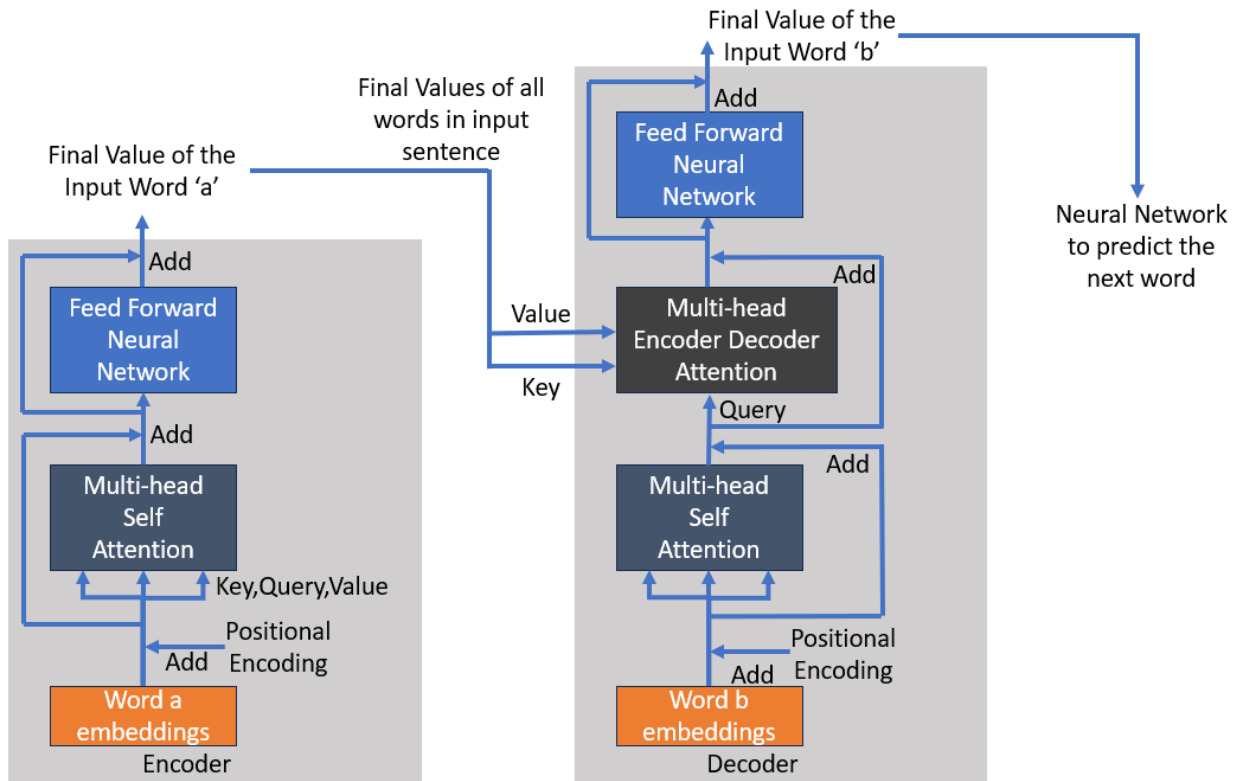


FIGURE 2.5: Transformer architecture

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 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.

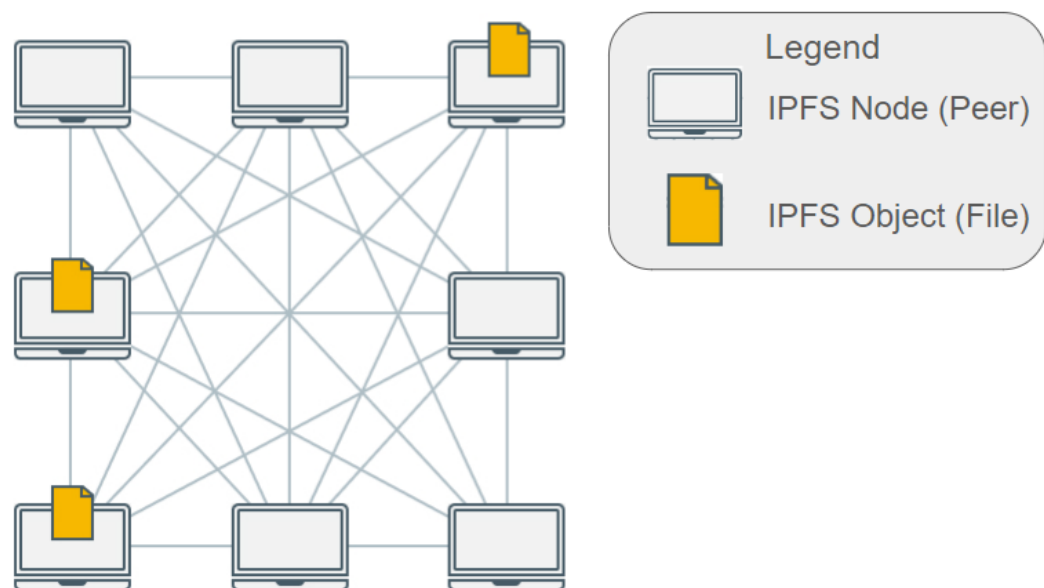
The main three types of transformers are:

1. **Encoder-Decoder Transformers:** The encoder-decoder transformer is a neural architecture in NLP tasks, processing input sequences and generating output sequences. T5 (Text-to-Text Transfer Transformer) [11] and BART (Bidirectional and Auto-Regressive Transformer) [8] are examples. T5 handles both input and output in text format, enabling versatile tasks like translation and summarization. BART, utilizing both encoder and decoder transformers, excels in text generation, language understanding, and summarization.
2. **Encoder-only Transformers:** Encoder-only transformers, like BERT (Bidirectional Encoder Representations from Transformers) [7], focus solely on encoding input sequences without generating output sequences. They excel in various natural language understanding tasks such as text classification, named entity recognition, answer extraction, text similarity, and paraphrase identification.
3. **Decoder-only Transformers:** Decoder-only transformers focus exclusively on generating output sequences based on context vectors, primarily used for autoregressive generation tasks. They employ masked-self attention, calculating attention values with only the words seen so far. Models like GPT-3 and GPT-2 [5] [10] utilize decoder-only architecture for tasks such as text generation, completion, summarization, question answering, and translation.

## Chapter 3

# InterPlanetary File System

InterPlanetary File System (IPFS) [1] is a protocol and network designed to create a content-addressable, peer-to-peer method of storing and sharing hypermedia. It aims to replace the traditional location-addressing of files with content-addressing, thereby enabling a more efficient and decentralized distribution of data across the internet.



Source : <https://aioz.network/blog/what-is-ipfs>

FIGURE 3.1: Block diagram of IPFS

## 3.1 Principles and Components of IPFS

The core principles upon which IPFS is built are:

1. **Content Addressing:** Instead of addressing files by their location on a server (like URLs), IPFS uses content-based addressing. Each piece of content is given a unique cryptographic hash, which serves as its address. This ensures content integrity and facilitates efficient distribution.
2. **Distributed Hash Table (DHT):** IPFS utilizes a DHT to maintain records of peers and the content they are hosting. This allows for content discovery without relying on centralized servers.
3. **Peer-to-Peer Networking:** IPFS employs a peer-to-peer network model [Fig. 3.1], where nodes in the network (peers) collaborate to store and distribute content. Peers request and deliver content directly from each other, reducing reliance on centralized servers.
4. **Versioning and Deduplication:** IPFS supports versioning of files, allowing users to track changes over time. It also deduplicates identical files, ensuring efficient use of storage resources.
5. **Merkle DAG:** IPFS organizes content using a Merkle Directed Acyclic Graph (DAG). This data structure enables efficient verification, retrieval, and traversal of content.

IPFS contains two key components [Fig. 3.1], which are:

1. **IPFS Node:** An instance of IPFS running on a device. Nodes store, retrieve, and distribute content within the IPFS network.
2. **IPFS Object:** The fundamental unit of storage in IPFS. An object can represent a file, directory, or any other type of data. Each object is uniquely identified by its content hash.

## 3.2 Use Cases of IPFS

IPFS offers a multifaceted solution to various digital challenges. Firstly, it revolutionizes web hosting by decentralizing websites and web applications, diminishing reliance on centralized servers and enhancing resistance to censorship. Additionally, it streamlines content distribution, facilitating the efficient dissemination of sizable files like software updates, multimedia content, and scientific datasets through peer-to-peer networking. Furthermore, IPFS serves as



a robust platform for immutable data storage, ensuring the tamper-proof and verifiable retention of crucial information such as cryptographic keys, blockchain data, and historical records. Moreover, it enables seamless offline content sharing by replicating data across devices and synchronizing changes, thus fostering connectivity even in disconnected environments.

### 3.3 Challenges and Limitations of IPFS

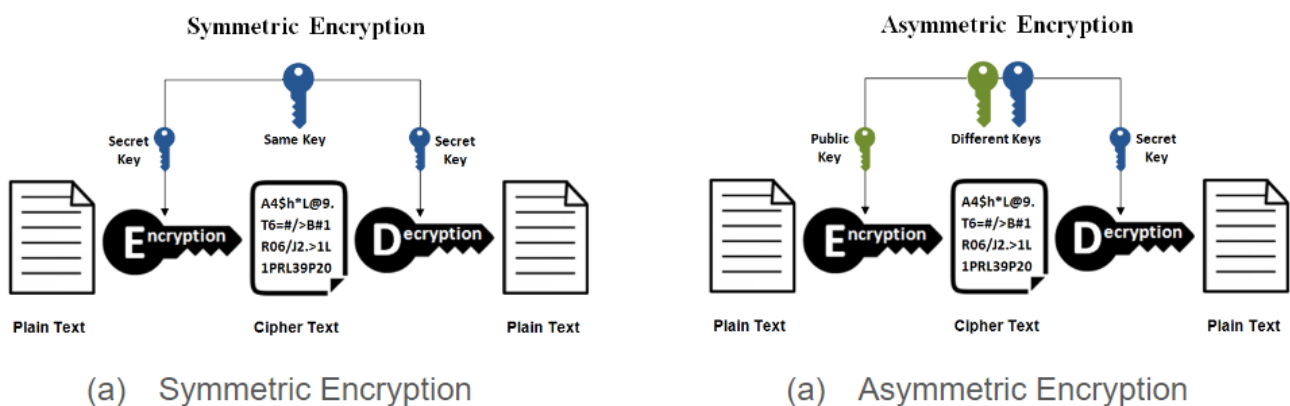
While IPFS employs cryptographic hashing to ensure content integrity during transmission over the network, it faces challenges regarding network scalability, content availability, and content integrity. Network scalability issues, such as network congestion, resource consumption, and performance bottlenecks, are inherent to any peer-to-peer network, including IPFS. Moreover, the availability of content on IPFS relies heavily on the number and reliability of peers hosting the content, potentially leading to slow or incomplete content retrieval in cases of low availability. Furthermore, IPFS encrypts content only during transmission and the content stored on nodes remains in a non-encrypted format, posing a risk of data loss or compromise. Additional measures, such as content validation and reputation systems, may be necessary to mitigate the risk of malicious or compromised content and ensure robust data integrity within the IPFS ecosystem.

To overcome this problem of data being stored in a non-encrypted format in the nodes, we would have to encrypt the data before it is uploaded on the network.

# Chapter 4

## Cryptographic Methods

Cryptography, a fundamental aspect of cybersecurity, encompasses a wide array of techniques and methods aimed at securing communication and data from adversaries. Its core principles include confidentiality, ensuring that unauthorized parties cannot access or understand encrypted data; integrity, guaranteeing data remains unaltered during transmission or storage; authentication, verifying the identity of communication parties; and non-repudiation, preventing parties from denying their actions or transactions. Various cryptographic methods such as symmetric and asymmetric encryption, hash functions, digital signatures, and cryptographic protocols are employed to achieve these goals. These methods play a crucial role in safeguarding digital assets and ensuring privacy and security in an increasingly interconnected world.



Source : <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>

FIGURE 4.1: Symmetric and asymmetric key algorithms

There are three major types of encryption algorithms:

1. **Symmetric key cryptography:** In symmetric encryption [Fig. 4.1(a)], the same key is used for both encryption and decryption. Common symmetric encryption algorithms include AES (Advanced Encryption Standard) and DES (Data Encryption Standard).
2. **Asymmetric key cryptography:** Asymmetric encryption [Fig. 4.1(b)] employs a pair of keys—a public key for encryption and a private key for decryption. RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Curve Cryptography) are popular asymmetric encryption algorithms.
3. **Hash Functions:** Hash functions generate fixed-size hash values from input data. These hash values are unique to the input data and are used for data integrity verification, password hashing, and digital signatures. Common cryptographic hash functions include SHA-256 and MD5.

In my thesis, I have used symmetric-key cryptography, so let's delve into it in depth.

## 4.1 Symmetric Key Encryption

Symmetric key encryption [2], also known as secret key/ private key encryption, utilizes a single key for both encrypting and decrypting data, offering computational efficiency suitable for encrypting extensive data volumes. Algorithms like AES and DES are prominent in symmetric encryption for their speed and security, crucial for safeguarding sensitive information across digital platforms. However, since the same key is used for encryption and decryption, secure key exchange mechanisms are imperative to ensure the confidentiality of communication. Key distribution protocols or physical delivery of keys are commonly employed for securely sharing symmetric keys between communicating parties. Additionally, leveraging public-key cryptography for key exchange or key agreement protocols enhances the security of establishing shared secrets, thereby fortifying the integrity of symmetric key encryption in protecting sensitive data.

Symmetric key cryptography proceeds as follows:

1. In symmetric key encryption, a single secret key is generated or chosen by the sender, ensuring confidentiality between communicating parties.
2. The plaintext, representing the original data, undergoes encryption using the secret key, resulting in ciphertext that appears random and unreadable without the key.
3. Decryption by the recipient involves using the same secret key to reverse the encryption process, recovering the original plaintext from the ciphertext.

Three of the best symmetric key algorithms are:

1. **Advanced Encryption Standard:** AES (Advanced Encryption Standard) stands as a cornerstone in modern cryptography, heralded for its exceptional security and efficiency. Developed as a replacement for the aging Data Encryption Standard (DES), AES employs symmetric key encryption, utilizing the same key for both encryption and decryption processes. What sets AES apart is its robustness against various cryptographic attacks, with its security thoroughly vetted by cryptographers worldwide. Offering flexibility in key sizes—128, 192, or 256 bits—AES allows users to tailor their security requirements according to the sensitivity of their data. Its widespread adoption across industries and applications underscores its reliability and trustworthiness. AES's computational efficiency further enhances its appeal, making it suitable for a diverse range of platforms and environments, from resource-constrained devices to high-performance computing systems. In essence, AES stands as the gold standard in symmetric key encryption, ensuring the confidentiality and integrity of data in an increasingly interconnected digital landscape.
2. **Data Encryption Standard:** DES, developed by IBM in the 1970s, marked a pivotal moment in cryptography as one of the earliest widely adopted symmetric encryption algorithms. Utilizing a 56-bit key, DES served as a benchmark for secure data communication for years. However, its susceptibility to modern brute-force attacks due to its short key length led to its gradual replacement by more robust algorithms like AES. Despite its obsolescence, DES retains historical significance and is still utilized in legacy systems where modern cryptographic standards are impractical. Nevertheless, its legacy continues to influence the evolution of cryptography towards more secure encryption standards.
3. **Triple DES:** Triple DES (3DES) acted as a transitional stage in symmetric key cryptography, aiming to bolster the security of the aging Data Encryption Standard (DES) while ensuring compatibility with existing systems. By employing DES encryption three times sequentially with different keys, 3DES extended the key length to 168 bits, notably enhancing security against brute-force attacks compared to DES. However, its triple encryption process led to slower performance, limiting its suitability for high-speed encryption tasks. Despite these drawbacks, 3DES found widespread adoption as a practical solution for securing legacy systems and maintaining compatibility with DES. While its prominence has diminished with the rise of more efficient encryption algorithms like AES, 3DES persists in certain industries and applications where backward compatibility and security remain paramount.

# Chapter 5

## Implementation and Results

### 5.1 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.

## 5.2 Fine-tuning of LLM model

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.

### 5.2.1 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 5.1], 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 5.1: 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.

### 5.2.2 Ingestion of custom data

The algorithm to ingest data from custom documents [Fig. 5.1] 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.

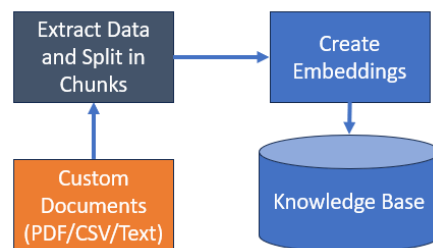


FIGURE 5.1: Ingestion algorithm

The data from the required custom documents is then extracted one document at a time and divided into various chunks. 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.

## 5.3 Encryption and Uploading to IPFS

### 5.3.1 Selection of encryption algorithm

Among AES, DES and Triple DES, I chose AES for the following reasons:

1. **Security:** AES offers superior security compared to DES and 3DES. It has withstood extensive cryptanalysis and is resistant to various attacks, including brute-force attacks.
2. **Efficiency:** AES is computationally efficient, providing strong security with minimal computational overhead. Its optimized implementations make it suitable for a wide range of applications, including resource-constrained devices and high-performance computing systems.
3. **Flexibility:** AES supports multiple key sizes (128, 192, and 256 bits), allowing users to choose the appropriate level of security based on their requirements. This flexibility makes AES adaptable to various security scenarios.
4. **Industry Standard:** AES has become the de facto standard for symmetric encryption and is widely adopted across industries and applications. Its widespread usage ensures compatibility and interoperability among different systems and platforms.

While DES and 3DES have historical significance, AES offers significantly improved security and performance, making it the preferred choice for securing sensitive data in modern computing environments.



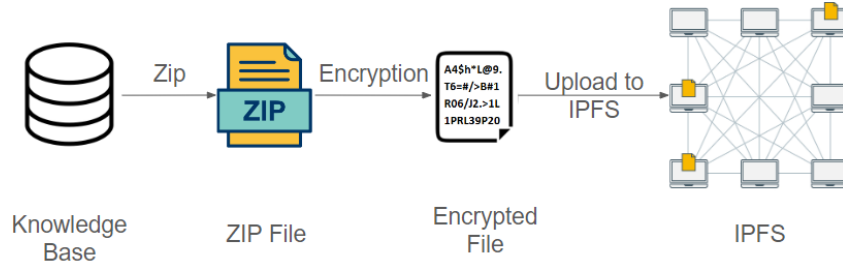


FIGURE 5.2: Steps followed to obtain to upload encrypted database over OPFS

### 5.3.2 Encryption and Uploading of the Knowledge Base

The knowledge base is encrypted using the AES algorithm. The steps followed in encrypting the knowledge base are [Fig. 5.2]:

1. **Zipping** :Initially, the database, comprised of multiple files within a database, is compressed into a single file by zipping it. This zipped file makes it easy to process further steps.
2. **Key Generation**: The encryption key is derived using the PBKDF2 algorithm (Password Based Key Derivation Function 2). The key is generated with a fixed size, utilizing the UserId from the user as a salt and the Password as input for the PBKDF2 function.
3. **Encryption**: The compressed file undergoes encryption using the generated key and AES algorithm to secure its contents.
4. **File Upload**: The encrypted zip file is uploaded over the IPFS network. The hash code of the uploaded file is saved on the local system for reference.

To mitigate potential data breaches, local copies of the database, zip file, and encrypted files are promptly deleted after upload. This ensures that even in the event of a local system breach, access to the stored IPFS file remains improbable.

Notably, neither the UserID, Password, nor the encryption key generated are stored in the local database, enhancing security. Only authorized individuals possessing the UserID and Password can decrypt the file, maintaining strict access control.

## 5.4 Running LLM on custom files

### 5.4.1 Downloading and decryption of the database

As no file of the database is saved on the local system, we need to download the file from the IPFS using the saved hash key and then decrypt it. The steps of the same are below [Fig. 5.3]:

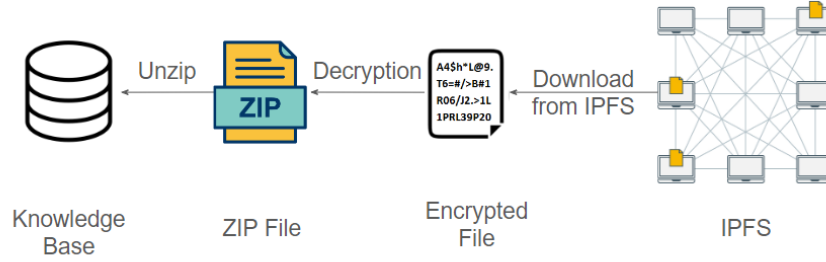


FIGURE 5.3: Steps followed to obtain the database from the file uploaded over IPFS

1. **File Download :** The encrypted zip file is downloaded from the IPFS network using the hash code saved on the local system.
2. **Key Generation:** The description key which is the same as the encryption key is again derived using the PBKDF2 algorithm (Password Based Key Derivation Function 2). As the UserID, Password or encryption key are not saved on the local system, only authorized personnel can create the right decryption key using the UserID and Password.
3. **Decryption:** The encrypted file is decrypted to obtain the compressed file containing the database.
4. **Unzipping:** The compressed file is unzipped to obtain the whole database on which the LLM can be done.

Using IPFS can ensure access to the database from remote/cloud computers making it an easy yet reliable tool to extract required data.

### 5.4.2 Running LLM on custom Database

Once the database is downloaded from IPFS, we can run queries on the LLM that are specifically related to the custom documents contained in the database.

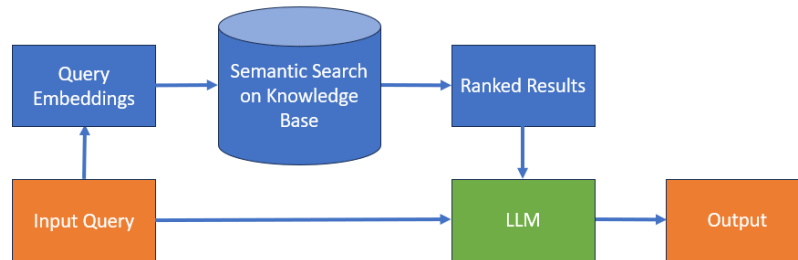


FIGURE 5.4: Algorithm that runs LLM on custom files

As seen in Fig. 5.4, 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.

## 5.5 Results

### LLM Model Accuracy

To test the accuracy of the model developed, I ran the model for 20 times, each time on a different file. Each file had about 500KB of data. In each case, I asked 20 different questions. As a result, I found out that the model gave the accurate answer about **98%** of the times.

When the same experiment was run with 2 random files per run, the results were the same, proving that the model easily understands the distinction between two different topics and does not mix them up.

As an example, I have trained my model on my resume. The queries give very accurate outputs with minimal hallucination as seen in Fig. 5.5.

### Encryption Strength

The encryption used for this project is symmetric key encryption where we do not save the key in any file. The key is generated randomly using a login ID and password that the user enters. The key length is 32 bytes which is 256 bits. So there are  $2^{256}$  possible combinations of keys which is nearly impossible to decode.

Even if any person decodes the key and decypts the file, the file contains data that is in the format of embeddings, which means that chunks of text data are represented as vectors with a huge number of numerical parameters. The embedding algorithm uses various parameters to generate unique embeddings. As we do not upload our code on the IPFS and it is only present locally, no person can guess as to what those series of numbers mean. Hence it is really very hard to decode the sensitive information present in the file, as it is secured twice.

```
> 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 5.5: Results produced on training the model on my resume

# Chapter 6

## Conclusion

In summary, this thesis marks a significant achievement in the development of a sophisticated chatbot powered by Large Language Models (LLMs). Through meticulous implementation of encryption principles, the chatbot boasts enhanced resilience against potential security threats and attacks, ensuring the protection of sensitive data. Furthermore, by leveraging the Inter-Planetary File System (IPFS) for data storage, the chatbot guarantees the accessibility and permanence of information across diverse computing environments, mitigating the risk of data loss and facilitating seamless remote access.

Looking ahead, the implications of this model are profound. Its capability to navigate and comprehend vast repositories of data promises to revolutionize information retrieval processes, offering a more intuitive and efficient alternative to conventional search methods. By enabling users to effortlessly locate specific information within files without relying on intricate file naming conventions, this model streamlines access to information, ultimately enhancing productivity and user experience in accessing and utilizing digital resources.

# Bibliography

- [1] J. Benet. Ipfs - content addressed, versioned, p2p file system. *ArXiv*, abs/1407.3561, 2014.
- [2] R. Bhanuka. Symmetric-key cryptography (private key encryption). *Medium*, 2021.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] S. T. ChangSheng. How will blockchain and chatbots be our everyday future. *Medium*, 2018.
- [7] 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.
- [8] 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.
- [9] 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.

- 
- [10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.
  - [11] 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.
  - [12] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *ArXiv*, abs/1409.3215, 2014.
  - [13] 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.