

# Customized Transformer Model for Efficient Extraction of Information from Textbooks

John Tawiah

---

Master's thesis  
2024



Tomas Bata University in Zlín  
Faculty of Applied Informatics

---

Tomas Bata University in Zlín  
Faculty of Applied Informatics  
Department of Informatics and Artificial Intelligence

Academic year: 2023/2024

# ASSIGNMENT OF DIPLOMA THESIS

(project, art work, art performance)

Name and surname: **John Akowuah Tawiah**  
Personal number: **A22715**  
Study programme: **N0613A140023 Information Technologies**  
Specialization: **Software Engineering**  
Type of Study: **Full-time**  
Work topic: **Přizpůsobený transformer model pro efektivní získávání informací z učebnic**  
Work topic in English: **Customized Transformer Model for Efficient Extraction of Information from Text-books**

## Theses guidelines

1. Prepare a literature survey on the topic of LLM or transformer, in general, using your own knowledge base that can efficiently extract valuable insights and information from educational books.
2. Gather requirements for model functionality to support learning and research.
3. Select an appropriate solution and create a functional model that meets the requirements, especially for summarizing a complex text.
4. Test the functionality of the proposed solution.
5. Discuss the possibilities of further development and modifications.

Form processing of diploma thesis: **printed/electronic**  
Language of elaboration: **English**

Recommended resources:

1. KUHAIL, Mohammad Amin, et al. Interacting with educational chatbots: A systematic review. *Education and Information Technologies*, 2023, 28.1: 973-1018.
2. SARRION, Eric. Exploring the power of ChatGPT: applications, techniques, and implications. Springer, [2023], ISBN 9781484295281.
3. HWANG, Gwo-Jen; CHANG, Ching-Yi. A review of opportunities and challenges of chatbots in education. *Interactive Learning Environments*, 2023, 31.7: 4099-4112.
4. KOOLI, Chokri. Chatbots in education and research: A critical examination of ethical implications and solutions. *Sustainability*, 2023, 15.7: 5614.
5. OPENAI, INC. *Documentation of OpenAI products*. Online. 2023. Dostupné z: <https://platform.openai.com/docs>. [cit. 2023-11-12].
6. JUSTIN, Paul; UENO, Akiko a DENNIS, Charles. ChatGPT and consumers: Benefits, Pitfalls and Future Research Agenda. Online. *International Journal of Consumer Studies*. 2023, roč. 47, č. 4, s. 1213-1225. Dostupné z: <https://doi.org/https://doi.org/10.1111/ijcs.12928>. [cit. 2023-11-12].
7. META PLATFORMS, INC. *Llama 2*. Online. 2023. Dostupné z: <https://ai.meta.com/llama/>. [cit. 2023-11-12].
8. LANGCHAIN, INC. *Documentation LangChain*. Online. 2023. Dostupné z: <https://python.langchain.com/docs>. [cit. 2023-11-12].

Supervisors of diploma thesis: **doc. Ing. Michal Pluháček, Ph.D.**  
Department of Informatics and Artificial Intelligence

Date of assignment of diploma thesis: **November 5, 2023**  
Submission deadline of diploma thesis: **May 13, 2024**



**doc. Ing. Jiří Vojtěšek, Ph.D. m.p.**  
Dean

**prof. Mgr. Roman Jašek, Ph.D., DBA m.p.**  
Head of Department

In Zlín January 5, 2024

Name of the student: John Akowuah Tawiah

Thesis topic: Custom Transformer Model for Effective Text Extraction From Textbooks

Name of the student:

Thesis topic:

I hereby declare that:

- I understand that by submitting my Master's thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Master's Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Master's Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlín, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Master's Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlín has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Master's Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlín with a view to the fact that Tomas Bata University in Zlín must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Master's Thesis include the use of software provided by Tomas Bata University in Zlín or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Master's Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Master's Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

I herewith declare that:

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.

In Zlín;

dated:

.....

Student's Signature

## **ABSTRAKT**

Se vznikem a popularitou velkých jazykových modelů (LLM) bylo získávání informací vždy náročné. Tyto LLM jsou však trénovány na textovém korpusu internetu a trénovací data mají datum uzávěrky; proto jsou ve většině případů data při vydání modelu zastaralá. Tato práce by zkoumala možnost výzkumu řešení, které uživatelům umožní přístup ke stručným informacím z rozsáhlého korpusu, jako je výuková učebnice. Navrhovaným řešením je implementace aplikace Retrieval Augmentation Generation umožňující uživatelům pracovat s LLM pomocí jejich soukromých dat. Práce bude také zkoumat možnost použití tohoto přístupu lokálně namísto používání rozsáhlých a těžkopádných modelů online. Výsledky ukázaly, že tento přístup je proveditelný a funguje dobře pro různé učebnice bez ohledu na jejich velikost, a také výstupní výsledky jsou dobré. Tato práce poskytuje cenné poznatky o tom, jak architektura funguje, a poskytuje správné nástroje pro implementaci tohoto řešení.

Klíčová slova: LLM, RAG, textový korpus

## **ABSTRACT**

With the emergence and popularity of Large Language Models (LLM), obtaining information has always been challenging. However, these LLMs are trained on the text corpus of the Internet, and the training data has a cut-off date; hence, in most cases, the data is out of date when the model is released. This thesis would investigate the possibility of researching a solution that allows users to access concise information from a large corpus like an educational textbook. The proposed solution is to implement a Retrieval Augmentation Generation application enabling users to work with an LLM using their private data. The thesis will also investigate the possibility of using this approach locally instead of using vast and cumbersome models online. The results demonstrated that this approach is feasible and works well for different textbooks irrespective of their size, and output results are also good. This thesis

provides valuable insights into how architecture works and provides the right tools to implement this solution.

Keywords: LLM, RAG, text corpus,

## **ACKNOWLEDGEMENTS**

I would like to express my deepest gratitude to my thesis advisor, Assoc. Prof. Michal Pluháček, PhD, for their unwavering guidance, support, and invaluable insights throughout the entire process of researching and writing this thesis. Their expertise and encouragement have been instrumental in shaping this work.

I am also very grateful to Prof. Ing Roman Šenkeřík, PhD, for suggesting this topic for my thesis and believing in me to see it through.

I am also grateful to all the staff and teachers, Ing Adam Viktorin, PhD, and Prof. Ing. Zuzana Komínková Oplatková, for their insightful courses and seminars, which gave me the technical know-how to approach this thesis.

Special thanks are due to the Faculty of Informatic (FAI) and all the staff for providing the necessary resources and environment conducive to academic inquiry.

I extend my heartfelt appreciation to my family for their endless love, encouragement, and understanding during this demanding period. Their unwavering belief in my abilities has been a constant source of strength.

To my friends and colleagues who provided moral support and encouragement, I am profoundly grateful.

Lastly, I acknowledge the countless researchers, scholars, and authors whose work laid the foundation for this study. Their contributions have been indispensable in shaping my understanding of the subject matter.

This thesis would not have been possible without the support and contributions of all those mentioned above. Thank you for being part of this significant milestone in my academic journey.

## CONTENTS

<b>INTRODUCTION</b> .....	10
<b>THEORY</b> .....	12
<b>I TEXT SUMMARIZATION</b> .....	13
<b>1.1 NATURAL LANGUAGE PROCESSING</b> .....	13
1.1.1 RECURRENT NEURAL NETWORK.....	13
<b>1.2 PORTABLE DOCUMENT FORMAT (PDF)</b> .....	14
1.2.1 TEXT EXTRACTION.....	15
<b>1.3 TRADITIONAL METHODS FOR TEXT EXTRACTION</b> .....	15
<b>II SEQUENCE MODELING TECHNIQUES</b> .....	17
<b>2.1 RECURRENT NEURAL NETWORK</b> .....	17
2.1.1 LONG SHORT-TERM MEMORY (LSTM).....	18
<b>III HISTORY OF TRANSFORMER MODEL</b> .....	19
<b>3.1 MACHINE LEARNING ALGORITHMS</b> .....	20
<b>3.2 WHAT IS THE TRANSFORMER MODEL</b> .....	20
3.2.1 COMPONENTS OF THE TRANSFORMER MODEL.....	21
<b>IV EMERGENCE OF LARGE LANGUAGE MODELS</b> .....	24
<b>4.1 IMPACT OF LARGE LANGUAGE MODELS</b> .....	25
<b>4.2 POPULAR LARGE LANGUAGE MODELS</b> .....	25
4.2.1 GENERATIVE PRETRAINED TRANSFORMER (GPT).....	26
4.2.2 FLAN T5 .....	26
4.2.3 BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS	
27	
4.2.4 LAMDA.....	27
4.2.5 PALM .....	27
<b>4.3 IMPORTANCE OF DOMAIN-SPECIFIC CUSTOMIZATION</b> .....	28
<b>V VECTOR DATABASE</b> .....	29
<b>ANALYSIS</b> .....	31
<b>VI IMPLEMENTATION ARCHITECTURE</b> .....	32
<b>6.1 RETRIEVAL AUGMENTED GENERATION</b> .....	32
6.1.1 COMPONENTS OF THE RAG ARCHITECTURE .....	33
<b>VII TOOLS AND SYSTEMS FOR IMPLEMENTATION</b> .....	35
<b>7.1 PYTHON</b> .....	35
<b>7.2 LANGCHAIN</b> .....	35



<b>7.3</b>	<b>THE HUGGING FACE PLATFORM .....</b>	<b>35</b>
<b>7.4</b>	<b>VECTOR EMBEDDINGS.....</b>	<b>37</b>
<b>7.5</b>	<b>STREAMLIT .....</b>	<b>37</b>
<b>7.6</b>	<b>LANCE DB .....</b>	<b>38</b>
<b>7.7</b>	<b>HUGGING FACE PIPELINE .....</b>	<b>38</b>
<b>VIII</b>	<b>LLM MODELS .....</b>	<b>40</b>
<b>8.1</b>	<b>SUMMARIZATION .....</b>	<b>40</b>
<b>8.2</b>	<b>TEXT GENERATION.....</b>	<b>41</b>
<b>8.3</b>	<b>TEXT-TO-TEXT GENERATION .....</b>	<b>41</b>
<b>IX</b>	<b>IMPLEMENTATION OVERVIEW .....</b>	<b>43</b>
<b>9.1</b>	<b>EXTRACTING TEXT FROM PDF DOCUMENTS .....</b>	<b>43</b>
<b>9.2</b>	<b>VECTOR EMBEDDING MODEL.....</b>	<b>43</b>
<b>9.3</b>	<b>CAPTURE EMBEDDINGS IN A VECTOR STORE .....</b>	<b>44</b>
<b>9.4</b>	<b>USER QUERY AND INPUTS.....</b>	<b>44</b>
<b>9.5</b>	<b>MODEL PIPELINE .....</b>	<b>45</b>
<b>9.6</b>	<b>QUERY RETRIEVAL.....</b>	<b>45</b>
<b>9.7</b>	<b>RETRIEVING OUTPUT FROM LLM .....</b>	<b>46</b>
<b>X</b>	<b>RESULTS .....</b>	<b>47</b>
<b>10.1</b>	<b>TEST CASE 1 .....</b>	<b>47</b>
<b>10.2</b>	<b>TEST CASE 2 .....</b>	<b>48</b>
<b>10.3</b>	<b>TEST CASE 3 .....</b>	<b>49</b>
	<b>CONCLUSION.....</b>	<b>52</b>
	<b>BIBLIOGRAPHY .....</b>	<b>53</b>
	<b>LIST OF ABBREVIATIONS.....</b>	<b>56</b>
	<b>LIST OF FIGURES .....</b>	<b>57</b>
	<b>LIST OF TABLES .....</b>	<b>58</b>
	<b>APPENDICES .....</b>	<b>59</b>

## INTRODUCTION

Textbooks remain essential sources of instructional material in the modern period, as digital information has wholly transformed access to knowledge. However, textbook information's sheer volume and diversity are severely needed to improve adequate comprehension and extract essential ideas. Conventional techniques for extracting information from textbooks mainly rely on manual work, which can be tedious, error-prone, and time-consuming.

Current automated methods, including keyword or rule-based systems, could better handle textbook material's complex structures and different formats. Thus, there is an urgent need for sophisticated Natural Language Processing (NLP) systems that can quickly, accurately, and adaptively extract relevant information from textbooks automatically.

This thesis will explore the meaning and significance of a custom transformer model. In the last couple of years, there have been advancements in research on Machine Learning, especially with Natural Language Processing (NLP) and Recurrent Neural Networks, which had been initially used to process and analyze large chunks of unstructured data. As most data available to users and businesses in general are large amounts of unstructured, text-heavy data, with NLP, we can, to some extent, analyze and process so-called unstructured data [1, 2].

Using NLPs, we have built valuable systems like chatbots and voice assistants. NLPs work by adequately pre-processing the unstructured data and turning it into a format a Machine Learning model could understand and generate associations[3]. NLP gave rise to the Transformer architecture, which gained momentum rapidly when the paper based on the multi-head attention mechanism “Attention Is All You Need” was published in 2017 [4].

The Transformer is a Large Language Model (LLM), a Deep Learning model trained on an immense amount of data, making it capable of understanding and generating natural language to perform various tasks. Transformers have rapidly become the dominant architecture for NLU, surpassing alternative neural models such as convolutional and recurrent neural networks in performing tasks that help with natural language understanding and natural language generation [5].

The Transformer model gained popularity with the release of OpenAI's Generative Pre-Trained Transformer (GPT), which at that time could do tasks we never imagined could be done. With the release of GPT-3.5, which was trained on data scraped from the Internet up to April 2023. GPT-3.5 made it easy to quickly gain any information readily available online

without going through multiple websites and pages hunting for said information [6]. However, it was limited to its training data alone; hence, the model was unavailable for newer or custom information. The inability to grasp domain-specific nuances hindered the accuracy and relevance of the generated content within these areas.

Hence, my contribution to the body of work before me was focused on creating and evaluating a bespoke language model (LLM) transformer tailored to a particular domain. We will discuss the limitations of these generic large language models and solutions to mitigate them. The principal aim is to investigate the feasibility of allowing users to query a large language model using their custom data, making extracting information from textbooks and articles easy.

We would also investigate the various large language models available to determine which are suited to the task at hand. Domain-specific customization is critical since it can improve performance, accuracy, and relevance in certain activities, leading to breakthroughs in various fields, including legal, medical, financial, and others.

## **I. THEORY**

## 1 TEXT SUMMARIZATION

A summary is generally a shortened form of a long paragraph or sentence. Text summarization compresses the source text into a diminished version, conserving its information content and overall meaning[7]. Most data produced in our day-to-day lives are usually unstructured; to extract insights, we need to summarize the data and gain the valuable information we seek. Since manual text summarization is time-consuming and expensive, hence the automation of text summarization[2].

In this era of big data, automating this process has become more urgent than ever, and much research has gone into it. Text summarization automation has historically been the focus of the machine learning fields of natural language processing (NLP) and recurrent neural networks (RNN). These disciplines have made great strides in automating text summarization.

### 1.1 Natural Language Processing

Natural Language Processing is an aspect of Artificial Intelligence that helps computers understand, interpret, and utilize human language. NLP makes it possible for computers and humans to converse using human language. NLP disciplines, till recently, have been the go-to area for handling text summarization tasks.[2, 3]. Natural Language Processing uses various pre-processing processes to clean and normalize text data, such as removing punctuation, converting text to lowercase, and tokenizing sentences into individual words or phrases.

Additionally, stemming and lemmatization help reduce words to their base forms to improve analysis accuracy. Once pre-processed, the text is transformed into numerical representations through word embeddings or vectorization, enabling machine learning models to understand and work with the data. These processed representations are then fed into NLP algorithms, which utilize statistical methods, neural networks, or rule-based approaches to perform tasks such as named entity recognition, part-of-speech tagging, and syntactic parsing, ultimately enabling computers to comprehend and generate human language.[2]

#### 1.1.1 Recurrent Neural Network

Natural Language Processing had issues with the memory of previous tokens, and with Recurrent Neural Network, we found a solution to this problem. RNN, a class of supervised

machine learning models with more than one feedback loop[8], is also artificial intelligence designed to aid automatic text summarization. It effectively handled sequential data by retaining memory of past inputs.

The RNN's architecture allows it to process various sequences of differing lengths. At each step, an RNN takes an input vector and an internal hidden state vector representing information from previous steps. This structure enables it to capture context and patterns over time, making it suitable for language modelling tasks.

Nevertheless, understanding long-term dependencies is challenging for conventional RNNs due to vanishing or expanding gradient issues. In response, several variations were created, such as the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM), which use gating mechanisms to control information flow and more accurately record long-range relationships, increasing their efficacy for a variety of sequential activities [8, 9].

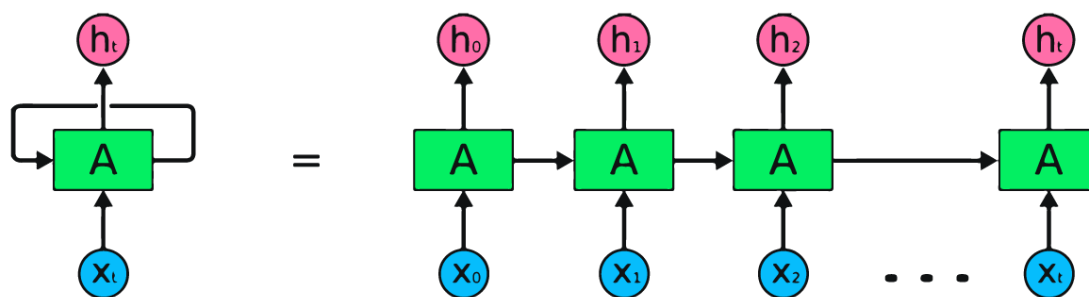


Figure 1: Recurrent Neural Network

## 1.2 Portable Document Format (PDF)

The Portable Document Format, popularly known as PDF, was initially developed by Adobe Inc. in Inc. 90[10]. PDFs are one of the most used file formats for distributing text and image information on printed paper, including eBooks, forms, and paper slips. They allow users to view, print, copy, and share documents easily, saving time and paper.

PDFs are a necessary tool as most of the textbooks available are in PDF format, so understanding how a PDF works would benefit this study. To create a PDF, you need a PDF editor, which allows you to create content that can be parsed in PDF format. You also need a PDF reader to read the data from a PDF. For the scope of this research, we will focus on the PDF reader and some tools we can use to extract data from a PDF.

### 1.2.1 Text Extraction

Textual data is necessary to develop a custom large language model, and textual information is usually stored in a Portable Document Format (PDF). Hence, we need to understand how to extract all the required information from the PDF without losing the data's integrity.

Its widespread availability across multiple digital platforms and domains makes it a preferred medium for large language model (LLM) evaluation and training. Textual data provides a rich tapestry of linguistic nuances and semantic complexity, making it an excellent substrate for investigating the subtleties of human language and cognition. Textual data can range from social media snippets to academic essays to textbooks [7, 9]. Because of its organized structure and computational ease of use, many analytical methods may be used, enabling scholars and professionals to find hidden trends and extract insightful information from a large text corpus[9].

Furthermore, by transforming static documents into machine-readable formats, text extraction from PDFs expedites workflows in document processing. It makes it possible for automated text-processing activities like document classification, summarization, and translation. Text extraction techniques make it possible for PDF material to be seamlessly integrated into a variety of apps and systems, increasing productivity and decreasing human labour [7, 9]

## 1.3 Traditional Methods for Text Extraction

Conventional text extraction methods include a range of approaches, such as PDF parsing libraries, regex-based parsing, and optical character recognition (OCR), used to extract text content from documents. We will briefly look at what these so-called tools do, but this study will focus on PDF parsing libraries that enable us to extract information from PDFs adequately.

- **Optical Character Recognition (OCR)** algorithms examine how characters are represented visually to identify and translate them into text that machines can read. Character segmentation, feature extraction, and classification are steps in this process [7, 11]. Over time, advances in OCR technology have made it possible to extract text from complex and damaged documents accurately. It's commonly employed in

digitizing printed documents, archival tasks, and making scanned documents searchable[7].

- **Regex-Based Parsing** - Regular expressions, or regex, are used in regex-based parsing to recognize and extract text patterns from documents. This method matches text sequences or structures inside the document using predetermined patterns or rules [7]. You can modify regex patterns to capture other elements, including phone numbers, email addresses, dates, or keywords. Text extracted from documents with complicated formatting or irregular layouts may be complex for regex-based parsing, notwithstanding its effectiveness in extracting organized data or specific information formats.

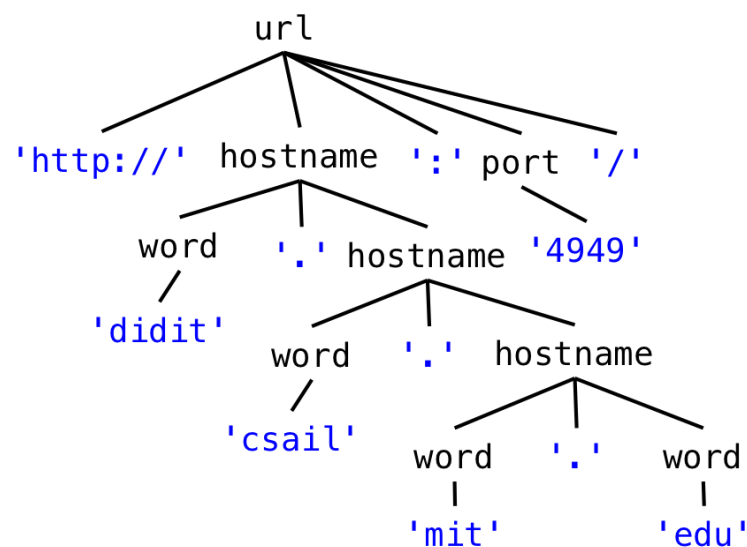


Figure 2: Regex Based Parsing

- **PDF Parsing Libraries**—These are generally software tools designed to read PDFs and help parse the PDF's content into other formats. Such a format would enable us to feed our large language model with the necessary data from the PDF. In recent times, the Langchain community has provided excellent tools that can be used to adequately parse the information from the PDF into any machine-readable format.



## 2 SEQUENCE MODELING TECHNIQUES

There were prior studies in this area before establishing Large Language Models, which are sequence modelling techniques. Sequence modelling started with Natural Language Processing and was made better with Recurrent Neural Networks. Sequence models are the ability of a model to interpret, make predictions about or generate any sequential data[8].

### 2.1 Recurrent Neural Network

Recurrent Neural Networks (RNNs) (RNNs) are particularly suited for sequence modelling. They process sequential data by maintaining an internal state. This mechanism helps RNN models remember the past, and decisions are influenced by what they did in the past by learning from prior inputs[8].

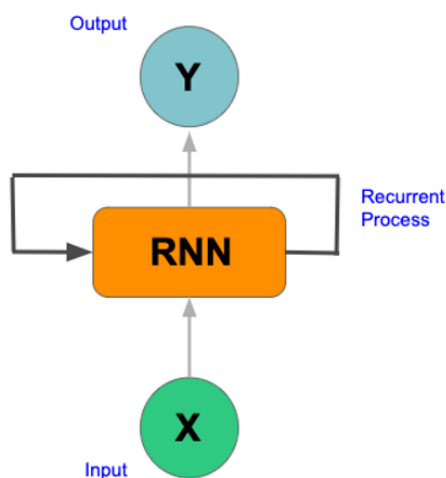


Figure 3: Single RNN cell.

RNNs are called recurrent because they perform the same task on every element in the sequence, which is usually a text stream. Due to its cyclical nature, RNN models have a short-term memory; hence, if the sequence is long enough, you will have a hard time carrying information from earlier time steps to later ones; this phenomenon is known as the Vanishing Gradient. On the other hand, if gradients rush to large values ( $>1$ ), they get larger and eventually blow up and crash the model. This is the so-called Exploding Gradient[12]

### 2.1.1 Long Short-Term Memory (LSTM)

LSTM is an improved version of the regular RNN. It was designed to help RNN models capture long-term dependencies in sequential data. The core idea is to help the model easily remember information for long periods and the default behaviours[12].

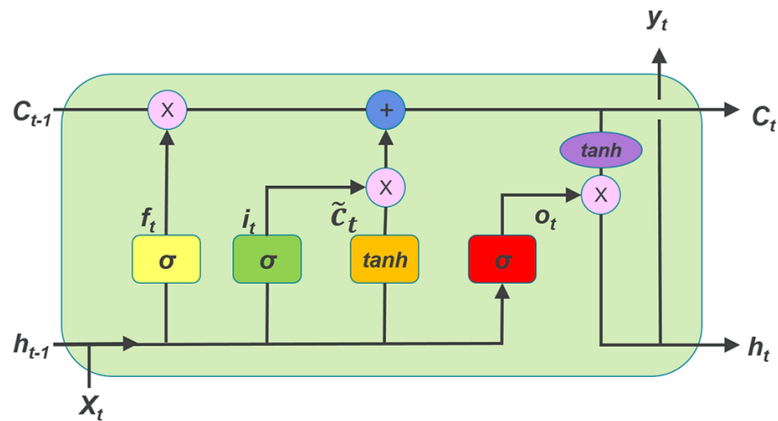


Figure 4: Long Short-Term Memory Architecture.

With LSTM, the model's cell state runs through the architecture like a conveyor belt with small, minor linear interactions. Hence, information flows along easily and remains unchanged. It uses carefully regulated gates to remove and add information to the cell state.

### 3 HISTORY OF TRANSFORMER MODEL

The discipline of Natural Language Processing (NLP), which has long been of interest in artificial intelligence, is where transformer models have their roots [3, 13]. Natural language processing (NLP) enables tasks like question answering, sentiment analysis, and translation by allowing computers and humans to communicate through natural language.

To analyze and comprehend text, early methods of NLP mainly depended on rule-based systems, in which linguistic rules were manually created. Unfortunately, the subtleties and complexity of natural language were too much for these algorithms to manage, which resulted in only patchy success[3, 14].

The development of statistical techniques in NLP in the latter half of the 20th century brought about a dramatic change. Using methods like Probabilistic Context-Free Grammars (PCFGs) and Hidden Markov Models (HMMs), researchers were able to statistically model language and obtain better results in tasks like machine translation and speech detection [3, 14].

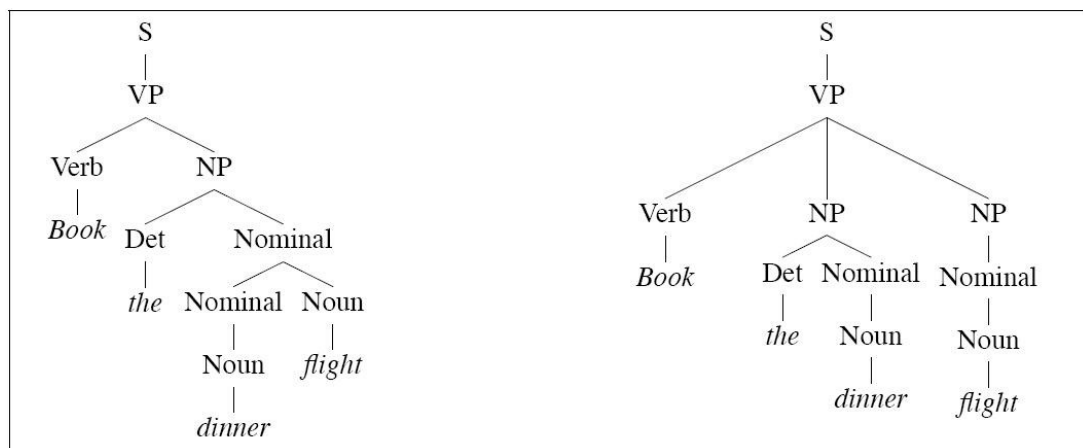


Figure 5: An example of Probabilistic Context-Free Grammar

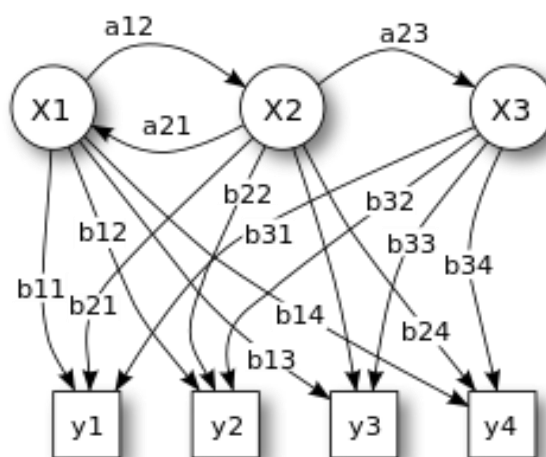


Figure 6: An example of the Hidden Markov Model

### 3.1 Machine Learning Algorithms

The discipline of NLP was significantly expanded with the development of machine learning methods, especially neural networks. Recurrent Neural Networks (RNNs) have become highly effective models for sequence modelling applications, opening new avenues for more complex language creation and comprehension. We as humans do not start thinking from scratch every second, and our understanding of sentences or paragraphs is based on our understanding of previous words.[15] Traditional Neural Networks cannot do this; thus, it has been a major shortcoming. RNNs were developed to address this issue[16].

RNNs are a family of artificial neural networks that are especially helpful for tasks like time series analysis, speech recognition, natural language processing (NLP), and more since they are made to simulate sequence data efficiently. Recurrent neural networks (RNNs) are unique among neural network types because of their capacity to process inputs sequentially while preserving internal state or memory[3, 15].

### 3.2 What Is the Transformer Model

In machine learning and natural language processing (NLP), the Transformer model, first presented by Vaswani et al. in 2017, marks a paradigm change. Since then, this

groundbreaking architecture has been the basis for numerous cutting-edge models in various fields, such as sentiment analysis, text generation, language translation, and more [4]. The transformer model has given rise to modern AI tools like ChatGPT and many pre-trained and fine-tuned Large Language Models. Some of the Transformer's salient features are self-attention mechanisms, positional encodings, multi-head attention, feedforward neural networks, and an encoder-decoder architecture [4]. The Transformer model has formed the basis for many cutting-edge NLP models because it can efficiently capture local and global relationships while processing whole sequences in parallel. It has dramatically advanced the science of natural language processing (NLP) by demonstrating extraordinary performance in tasks like question answering, text summarization, and machine translation [4].

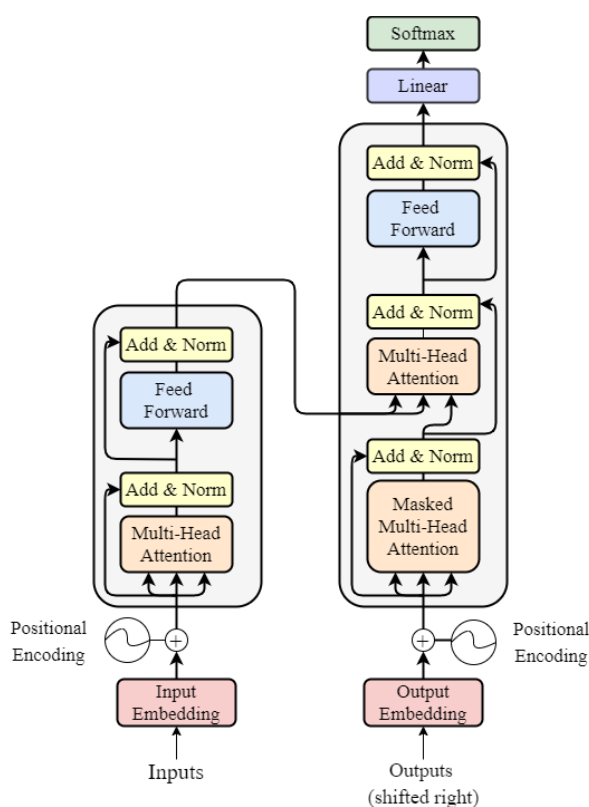


Figure 7: Transformer Model Architecture

### 3.2.1 Components of the Transformer Model

- Self-Attention Mechanism** - The self-attention mechanism is the core of the Transformer architecture. It is a crucial component that allows each word in the input sequence to pay attention to and consider every other word. This allows for the efficient capture of local and global dependencies [4, 14, 17]. By allocating weights

to distinct words according to their relevance and importance, this system provides a thorough knowledge and depiction of the contextual interactions between words at different granularities, ranging from terms that are close to terms that are far away. The self-attention mechanism is essential to boosting the model's ability to identify complicated patterns and correlations in textual material and improve its overall performance in various scenarios by enabling such sophisticated and nuanced information processing [4].

- **Positional Encoding** - The Transformer model uses positional encoding to include positional information into input sequences. Appending sine and cosine functions with varying frequencies and phases to input embeddings compensates for the inherent order understanding deficit in parallel processing [4]. This enables the model to recognize the sequential nature of a sequence and differentiate between points within it. The Transformer can analyze sequences efficiently in tasks like natural language processing because of positional encoding[17–19].
- **Encoder and Decoder** - The encoder-decoder architecture is a critical component of sequence-to-sequence models like the Transformer, whose popularity has increased dramatically in recent years. The encoder in this design is in charge of carefully processing the input sequence to provide contextual representations that contain essential information [4]. As such, the decoder uses these representations to construct the output sequence carefully, guaranteeing a precise and consistent outcome. Moreover, this architecture is widely applied to tasks such as question answering, text summarization, and machine translation, allowing models to manage and produce sequences of different lengths with impressive efficiency [4, 20].
- **Multi-Head Attention** - A critical feature of transformer models is multi-head attention, which allows the model to focus on several parts of the input sequence simultaneously, improving the model's capacity to identify complex correlations in the data [4]. The self-attention mechanism is applied repeatedly in parallel, increasing computational efficiency while simultaneously allowing the model to extract more specific and subtle information from the input. The model may provide richer representations of the input by using several attention heads, making it easier to comprehend and evaluate the underlying patterns and dependencies in the data.

- **Feedforward Neural Networks**—Besides the self-attention layers, feedforward neural networks (FFNs) (see Feed Forward Neural Network) are also part of the Transformer design. These FFNs are essential in adding additional nonlinearities and transformations to the representations that the model learns.

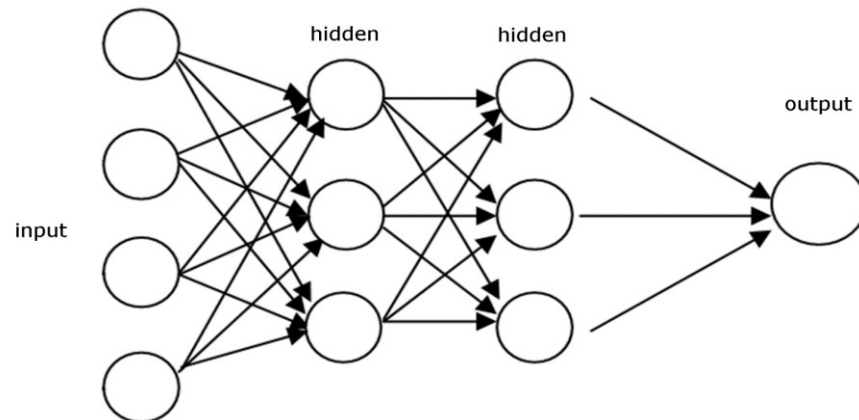


Figure 8: Feed Forward Neural Network

## 4 EMERGENCE OF LARGE LANGUAGE MODELS

The Transformer Model forms the base architecture for training Large Language Models [20, 21]. A notable development in natural language processing (NLP) is the advent of Large Language Models (LLMs). These Transformer-based models are trained on copious volumes of textual input to acquire rich linguistic representations. During pre-training, LLMs use unsupervised learning objectives, including language modelling or masked language modelling, like GPT (Generative Pre-trained Transformer) and BERT (Bidirectional Encoder Representations from Transformers). After undergoing pre-training, LLMs can be optimized for downstream tasks, leading to cutting-edge results on various sequencing tasks [17, 20].

With the advent of large-scale pre-training in natural language processing (NLP), models such as the GPT (Generative Pre-trained Transformer) series and BERT (Bidirectional Encoder Representations from Transformers) series proved to be remarkably effective [22, 23]. These models acquired sophisticated language representations through pre-training, identifying complex patterns and semantic correlations in the data. The models gained a profound understanding of language during this pre-training phase, which helped them perform well in various downstream NLP tasks, such as text generation, question answering, and language understanding. The GPT series and BERT's success demonstrated the value of extensive pre-training in enhancing LLM and opening the door to a new era of language generation and processing [14, 20].

Significant progress has been made in several natural language processing (NLP) problems thanks to large language models, or LLMs. Because LLMs can recognize complex linguistic patterns and semantic linkages, they perform better in language comprehension tasks, including text categorization, named entity recognition, and sentiment analysis [14, 17, 21]. Text sequencing activities such as story generation, dialogue systems, and content production are greatly aided by LLMs such as GPT, which generate coherent and contextually appropriate text. LLMs also help with information retrieval and document summarization jobs by efficiently condensing vast amounts of text into summaries.



## 4.1 Impact of Large Language Models

LLMs have had immense applications since their introduction in the last couple of years. Here are some critical areas in which they have impacted.

- **Advancement of Research** - Advancements in model architectures, training methodologies, and transfer learning have propelled research in Large Language Models (LLMs). Beyond NLP, LLMs have stimulated research into computer vision and audio processing. Bias and fairness are two ethical issues that are actively addressed. Transfer learning makes it possible to fine-tune LLMs for tasks, while new architectures like XLNet and RoBERTa increase performance and efficiency[24, 25]. Efficient scaling is made possible by training methods like gradient accumulation and distributed training. LLM research keeps advancing AI, influencing the direction of future multidisciplinary applications and natural language processing [13].
- **Model Architecture**—Large Language Models (LLMs) have inspired new model architectures to improve performance, scalability, and efficiency. By introducing novel attention mechanisms, parameter sharing, and task formulations, variants such as XLNet, RoBERTa, and T5 push the frontiers of natural language processing[22].
- **Applications beyond NLP** - The use of LLMs in domains other than the typical natural language processing (NLP) has attracted attention, including computer vision, audio processing, and reinforcement learning. By utilizing their strong representation learning skills, LLMs are modified to handle a variety of modalities in addition to text [6, 17, 23]. Cross-modal pre-training strategies are created to facilitate knowledge transfer between different domains and allow LLMs to learn joint representations across several modalities. In computer vision, for example, LLMs can be trained to comprehend images and the accompanying captions, allowing them to perform jobs like labelling pictures and answering questions visually. LLMs can also evaluate audio signals and the accompanying transcripts or descriptions in audio processing [2, 25].

## 4.2 Popular Large Language Models

The following models have revolutionized several language tasks and made a substantial contribution to the field of natural language processing: The following models have

revolutionized several language tasks and made a significant contribution to the field of natural language processing:

#### 4.2.1 Generative Pretrained Transformer (GPT)

GPT is a sizable language model that OpenAI created. The initial version was made available in 2018. It can produce logical, contextually relevant phrases by guessing the word that will come after a given sequence of words. It has been trained on a wide variety of internet text. With 175 billion parameters, GPT-3 was the first model to generate highly realistic, human-like text and code based on subtle instructions. It was released in 2020[6].

OpenAI released ChatGPT in late 2022; it was improved using RLHF and built on top of GPT 3.5. ChatGPT's innovative capacity to produce human-like outputs in response to natural language cues was impressive. OpenAI's most powerful model, GPT-4, was released in April 2023 and may be used directly by API or as part of the ChatGPT service. GPT-4 significantly outperforms[6].

#### 4.2.2 FLAN T5

Flan T5 is an open-source, sequence-to-sequence, large language model that can be used commercially. Google published this model, which has been fine-tuned for many tasks. This architecture uses an encoder-decoder structure from the “Attention is All You Need” paper released in 2017. T5 was trained with an extensive 750 GB corpus of text known as the Colossal Clean Crawled Corpus (C4).

Flan T5 is mainly used for chat and dialogue summarization and text classification. It can be easily downloaded from the Hugging Face code repository.

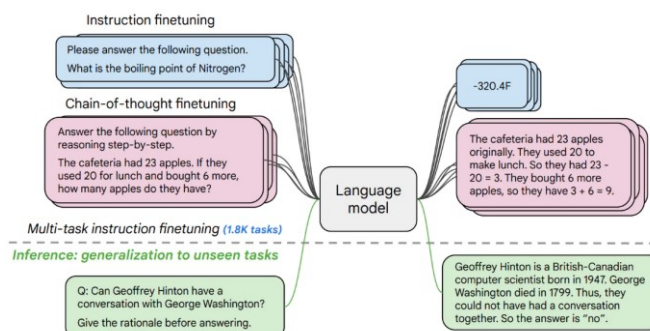


Figure 8: Flan T5 model.

### 4.2.3 Bidirectional Encoder Representations from Transformers

Google created the model known as BERT or Bidirectional Encoder Representations from Transformers. BERT examines both sides of a word, unlike GPT, which only looks at the context to the left of a word. Using a bidirectional approach, BERT can better comprehend a word's context, enhancing its ability to understand and generate language. BERT has been instrumental in several NLP tasks, including question-answering and linguistic inference. It has been an essential component of the Google search engine for several years[3, 6].

### 4.2.4 LaMDA

Google created a sizable language model called LaMDA. It was an early concept that could have free-flowing discussions on any subject, carry over the context of a debate, and consider information from earlier exchanges. Although LaMDA was not made available to the public, its ability to communicate with users like a human grabbed attention[3]. Blake Lemoine, a Google engineer who collaborated extensively with LaMDA, thought the system was aware. However, the majority of AI professionals and Google themselves refuted his assertion.

### 4.2.5 PaLM

Google's AI intelligence chatbot, Bard, is powered by the transformer-based Pathways Language Model (PaLM), which has 540 billion parameters. This model is trained on several TPU 4 Pods, Google's custom machine-learning hardware, and is intended to handle reasoning tasks such as coding, math, categorization, and question-answer activities. The PaLM paradigm divides complex jobs into smaller, more manageable subtasks [3].

The Pathways research project at Google aimed to create a master model suitable for various applications, giving rise to the moniker PaLM. PaLM has been refined through several versions. While Sec-PaLM is made for cybersecurity deployment and helps with faster threat analysis, Med-PaLM 2 is suited for the life sciences and medical data[3, 6].

The models discussed here are primarily generic models trained on data accessible on the Internet and that only. For the focus of my study, we will discuss how to extend the capability of large language models to enable users to make queries about their custom data or data inaccessible on the Internet.

### 4.3 Importance of Domain-Specific Customization

The domain-specific customization of Large Language Models (LLMs) is necessary since specialized areas have specific linguistic requirements. Despite their proficiency in recognizing broad language patterns, generic Large Language Models (LLM) often need help comprehending context, semantic nuances, and domain-specific jargon. Language usage is highly context-dependent and specializes in fields like academia, law, health, and finance, necessitating specialist solutions for accurate text generation and interpretation.

Domain-specific customization helps language managers (LLMs) better understand domain-specific conventions, interpret complex jargon, and produce content appropriate for the context by enabling them to adapt to the different language environments of different domains [22]. Researchers may give pre-trained LLM structures the domain knowledge required to succeed in particular tasks by fine-tuning them on domain-specific datasets. This results in more effective information extraction from domain-specific data.

## 5 VECTOR DATABASE

A vector database stores high-dimensional data that traditional DBMS cannot characterize [26]. Also known as vector similarity search databases or vector stores, vector databases are specialist data management systems that efficiently store, retrieve, and query high-dimensional vectors. Applications in machine learning, artificial intelligence, computer vision, and natural language processing are especially well-suited for these databases[27].

Vector databases are essential in building a custom LLM as they help users store their text corpus as an embedding in the database. The database can then be queried using a similarity search to retrieve the relevant information. Here is the basic structure of a vector database.

- **Data Model** - Vector databases employ an efficient representation of high-dimensional vectors that support a variety of data kinds, including textual embeddings, numerical values, and categorical attributes. As building blocks, vectors encode data points in a high-dimensional space, offering the choice of dense or sparse representations. [26, 27] Large-scale datasets from the machine learning, computer vision, and natural language processing fields can be efficiently stored, retrieved, and analyzed because of this model's flexible modification and optimization capabilities, which are helpful for applications like recommendation systems and similarity search.[27]
- **Indexing Structures** - Utilizing specialized algorithms, vector databases' indexing structure effectively organizes and retrieves vectors based on similarity. Graph-based indexes for approximate nearest neighbour search, hashing algorithms like locality-sensitive hashing (LSH), and tree-based structures like k-d trees and ball trees are common approaches. These structures facilitate activities like nearest neighbour searches and similarity searches in high-dimensional spaces, optimizing query processing by reducing search space and enabling quick retrieval of vectors like a given query[28, 29].
- **Query Processing** - Finding similar vectors to a given query vector is a necessary step in the querying process in vector databases. Using metrics like cosine similarity and Euclidean distance, this procedure assesses how far or similar the query vector is to the stored vectors. Using sophisticated methods such as approximate closest neighbour search, query processing is optimized to retrieve vectors under a given

similarity threshold effectively. To improve speed and scalability for activities like nearest neighbour queries and similarity search in high-dimensional spaces, query optimization procedures can involve index selection and query pruning [26, 27].

## **II. ANALYSIS**

## 6 IMPLEMENTATION ARCHITECTURE

In this section, we will discuss the approach to achieving the desired results. We will discuss the general architecture of the approach as well as the tools and other procedures used to achieve the desired results.

### 6.1 Retrieval Augmented Generation

Large language Models take input from sequential data, and this data is usually limited depending on the model available. These models have a finite number of tokens they can take, and any amount above the limit would break the model. Retrieval Augmented Generation, commonly known as RAG, is an Artificial Intelligence framework for retrieving facts from an external knowledge base to aid Large Language Models in attaining the most accurate, up-to-date information and give users insight into the LLM generative processes[30].

RAG is currently the best-known tool for grounding LLMs on the latest and most verifiable information and lowering the cost of constantly retraining and updating them. Fine-tuning a model is another alternative to consider, and it involves adjusting the weight of a model, but that can only work for a specific dataset, and any new data needed would imply retraining the whole model all over again. However, retraining the model is [31]unnecessary with a RAG implementation [31]. We would go with an RAG implementation for my contribution to this body of work. Here is the general architecture of the Retrieval Augmented Generation implementation, which we will discuss briefly.



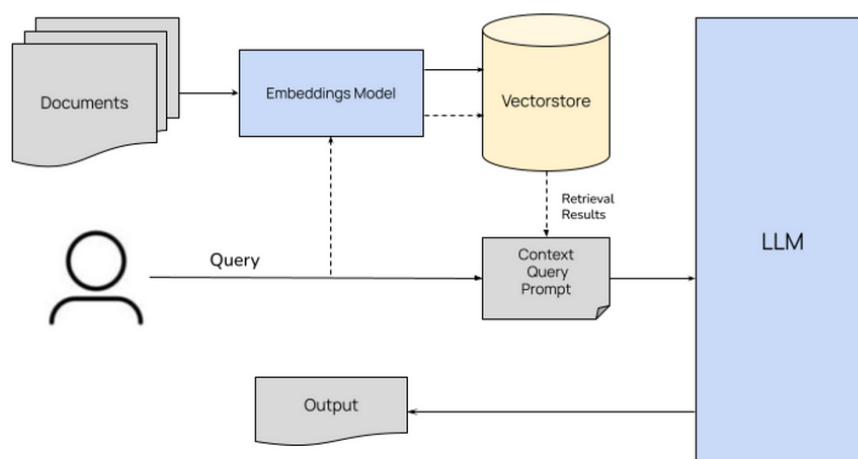


Figure 9: RAG Architecture

The RAG Architecture helps to fix our issue with the context window of large language models by first extracting all the text data from our document into a computer-readable format, then creating vector embeddings from the data generated using an encoder-only Transformer model like BERT, and then storing these embeddings in a vector database.

The user makes a query, and the vector store uses the query to perform a similarity search to retrieve the relevant vector embeddings. These retrieved embeddings, known as the Context Query Prompt, are fed to the large language model, and the user queries to retrieve an output from the large language model. We will discuss in detail the various components of the general architecture of the RAG implementation.

### 6.1.1 Components of the RAG Architecture

- **Documents** refer to our data source, which we will use for our RAG implementation. This data source would be a text document in PDF format. These documents are the knowledge foundation the RAG system would draw to generate responses [31, 32]. Here, we would use various publicly available text extraction tools to extract the data we need from user documents.
- **Embedding Model**—Encoder-only Transformer models like BERT return vector embeddings as output. This embedding model transforms the text from each document into a series of numerical vectors, effectively creating a high-dimensional representation of the papers' content. This vector embedding encodes the text

corpus's semantic meaning, allowing the system to understand and manipulate the information from the documents at a conceptual level [32].

- **Vector Store** - As discussed earlier, vector stores are necessary to execute the RAGs. The vector embeddings are stored in the vector database, which is designed to handle vector embeddings. Vector stores act like databases and provide methods to manipulate the data in the database like other regular databases. Vector stores are optimized for similarity searches, allowing them to quickly sift through millions of vectors to find those that closely match an output [32]. Vector stores store the vector embeddings, and the original text corpus is used to generate the embeddings, making it easy to query for information from the vector store, which then returns the vector embeddings like the user query.
- **Query**—When a user inputs a query, the system converts it into vector form, utilizing the same embedding model used to process the documents. This query vector encapsulates the user's intent and the semantic nuances of their request.
- **Retrieval** – The vector store compares the query vector with the entire corpus of document vectors. Using similarity metrics, a subset of vectors that are semantically closest to the query vector is retrieved. These correspond to the document's most likely to contain relevant information[32].
- **Context Query Prompt**—The retrieval results are amalgamated with the original query to create a context-rich query prompt. This amalgamation enriches the initial query with specific insights from the retrieved documents, equipping the system with a nuanced understanding of what the user is seeking[31, 32].
- **Large Language Model** – Large Language Models like GPT, Llama, and Flan T5 are trained on vast amounts of textual data, enabling them to understand natural language and generate coherent, contextually appropriate responses. Hence, with a shortened corpus of vector embeddings, we feed the LLM with the data from the context prompt, and the LLM produces the relevant output needed.
- **Output**—This is the final step in the Retrieval Augmented Generation architecture. At this point, the relevant information is returned to the user based on the query provided by the user.

## 7 TOOLS AND SYSTEMS FOR IMPLEMENTATION

Various tools are available to build a custom Transformer model to extract data for textbooks. We will look at some of the tools and systems we would use for the implementation.

### 7.1 Python

Python is a high-level, general-purpose programming language that is generally great for statistical analysis. Due to this, it is mainly used for Machine Learning and Artificial Intelligence solutions. Python has a large community and a vast ecosystem that enables efficient development, making it a preferred language for implementing machine learning algorithms and models.

### 7.2 Langchain

Langchain was launched in October 2022 and has gained many improvements from contributors on GitHub. It contains many tools that help integrate external systems and provides utility tools that make working with large language models relatively easy.

Langchain is an open-source tool quite popular for utility processes such as text extraction and other data pre-processing processes necessary for RAG implementation. From Langchain, we will use the PyPDFLoader and the RecursiveCharacterTextSplitter to extract data from user documents.

- **PyPDFLoader** - PyPDFLoader, with the help of the pypdf package, takes a path to a pdf as a parameter and loads the pdf into an array, where each context contains page content and metadata.
- **RecursiveCharacterTextSplitter**—The RecursiveCharacterTextSplitter is a helper utility that splits the loaded data into smaller chunks. It is recommended for generic text. A list of characters parameterizes the text splitter. It then tries to separate the characters in order until the chunks are small enough.

### 7.3 The Hugging Face Platform

For our pick on the best LLMs for the contribution, we would use the HuggingFace platform, an online repository of the most popular Large Language Models that provide datasets to use with these models. HuggingFace is the GitHub of the ML world.

HuggingFace provides a collaborative platform with tools that empower anyone to create, train and deploy ML models using open-source code[5]. The models are pre-trained; hence, you usually do not have to start from scratch; load a pre-trained model and fine-tune it to your specific task. These are some core components of HuggingFace.

- **Transformers Library** - This contains a comprehensive suite of state-of-the-art machine learning models. It consists of an extensive collection of pre-trained models optimized for tasks such as ‘summarization’, ‘text classification’, and ‘text generation’. Most NLP tasks have been abstracted into a ‘pipeline’ function.
- **Model Hub**—The model hub is a platform with thousands of models and datasets available at your fingertips. It is an innovative feature that allows users to share and discover models the community contributes, promoting a collaborative approach to NLP development[5, 33].

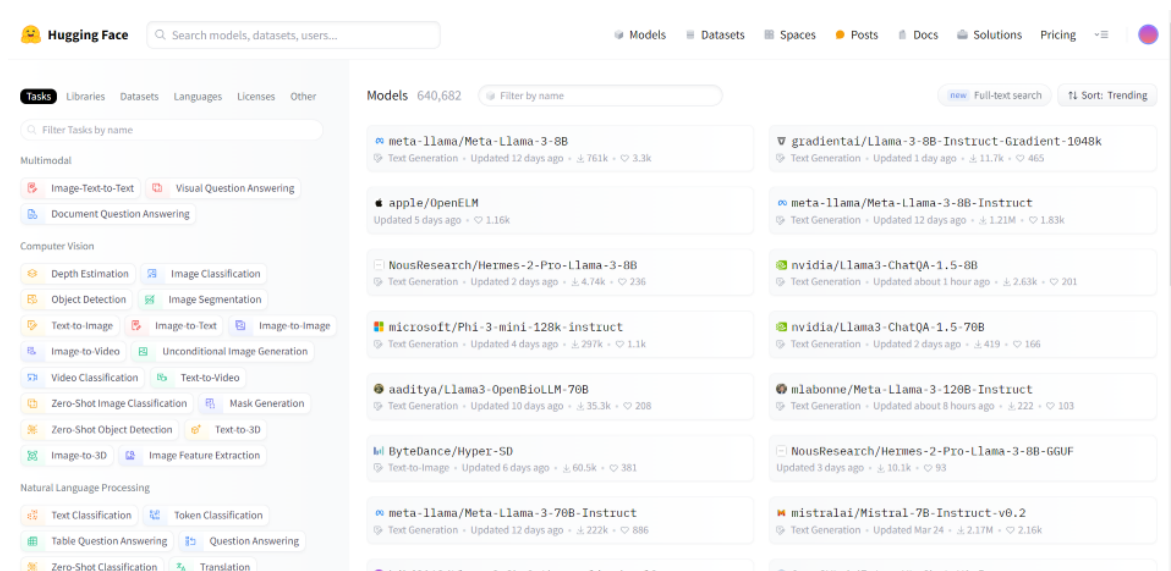


Figure 10: HuggingFace Model Hub

- **Tokenizers**—Tokenizers convert sequence text into a format that machine learning models can understand, which is essential to processing different languages and text structures. HuggingFace provides a wide range of tokenizers that facilitate the conversion of tokens into vector representation for LLM inputs and handle the truncation and padding of uniform sequence lengths[5].

## 7.4 Vector Embeddings

Vector Embeddings are a way to convert words, sentences and other data into numbers that capture their meaning and relationships. They represent different data types as points in a multidimensional space, where similar data points are clustered close together. These processes help machines understand and process this data more effectively. Word and sentence embedding are the most used types of vector embeddings [26];

- **Word Embedding**—Word embedding represents words as vectors in a continuous vector space, where the geometric relationship between vectors captures semantic similarities between words. Some popular word embedding models are Word2Vec and fastText [27].
- **Sentence Embedding**—Unlike word embedding, which represents individual words, sentence embeddings capture the semantic meaning of the entire text [26].

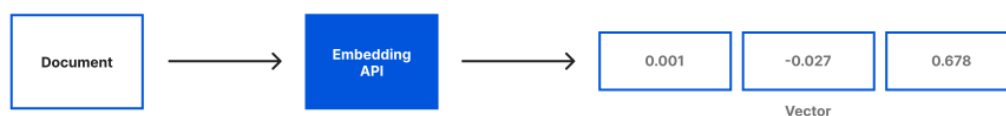


Figure 11: Vector Embeddings

Encoder-only Transformer models like BERT are usually used to generate vector embeddings. For our body of work, we will be using **sentence\_transformers**, which is an excellent library for generating vector embeddings for any text body. It also comes with different pre-trained encoder-only models that you can select from to use for each case.

## 7.5 StreamLit

Streamlit is a free, open-source framework for rapidly building and sharing beautiful machine learning and data science web applications. It is a Python-based library specifically designed for machine learning engineers. Streamlit is easy to use to build attractive user interfaces, especially for people without front-end knowledge.

Hence, streamlit will be used to build the user interface for our application, which will take user input and user documents to perform the RAG implementation. Streamlit provides easy-to-use User Interface (UI) components that are easy to set up and relatively easy to use.

## 7.6 Lance DB

LanceDB emerges as an exemplary choice for our vector store solution, catering specifically to the demands of Artificial Intelligence (AI) applications. As an open-source vector store, LanceDB excels in its ability to effectively store, manage, query, and retrieve vector embeddings, which is essential for numerous AI tasks. Its architecture is meticulously crafted to accommodate the intricacies of vector data, ensuring optimal performance across a spectrum of use cases.

One of LanceDB's standout features is its support for disk-based indexing and storage, a crucial facet that underpins its scalability. This architecture enables LanceDB to handle massive volumes of vector data effortlessly without imposing exorbitant resource demands.

## 7.7 Hugging face pipeline

Running machine learning algorithms involves pre-processing, feature extraction, model fitting, and validation stages. Pipelines are the creation of independent and reusable modules that can be easily pieced together to create an entire workflow.

The HuggingFace pipeline is a great and easy way to use models for inference. These pipelines provide a simple API by abstracting most of the complex processes you need. The pipeline is a wrapper around other available pipelines. Due to its high abstraction, the pipeline takes a lot of parameters, and for the focus of this thesis, I would talk about the ones used;

- **Task type**—This is the first parameter needed for the pipeline, and it is used to determine the kind of task that the Large Language Model will perform. There are many available task types, but the focus will be **summarization, text generation, and text2text generation**. These task types will determine the sub-pipeline the main pipeline extends to deliver the best results.
- **Model**—The model parameter determines the LLM to use for the task. For the best result, it is advised to use an LLM that matches the right task type.

- **Min Length** – The min length parameter determines the minimum number of output tokens to be returned as an output.
- **Max Length** - The max length parameter determines the maximum number of output tokens to be returned.
- **Model Kwarg**s—This parameter is a dictionary of sub-arguments that can be used to configure the pipeline. It takes **temperature** as a critical argument, ranging from 0 to 1. This determines how strict the results are and can give the model leeway to be creative, which can sometimes not be desired. Another key argument is the **device**. This determines if the model would be run using a CPU or a GPU.

## 8 LLM MODELS

Selecting a model for our approach would be crucial as it would determine the quality of the results. When building pipelines for Large Language Models using the HuggingFace pipeline utility, we first need to choose the kind of sequencing task we are trying to achieve. We will briefly discuss the three main sequencing tasks necessary for the approach in the RAG implementation.

### 8.1 Summarization

Generally, summarization creates a short version of the document or data provided while capturing all the relevant information. The goal of text summarization is to extract the most essential information from a text document and present it in a concise and understandable form.

With the self-attention mechanism, Transformers gain a contextual understanding of the text corpus, which allows them to adequately identify the most critical information in the data provided[9]. HuggingFace provides many fine-tuned and pre-trained large language models that are great for handling summarization and can be easily used with the pipeline utility. Here are some models provided for text summarization on the HuggingFace Platform.

- **Facebook/bart-large-cnn**—The BART model is pre-trained in English and fine-tuned to CNN Daily Mail, a carefully curated dataset of over 300,000 unique news articles written by CNN and Daily Mail journalists. This is one of the most popular models on HuggingFace [5].
- **Google/pegasus-large** - The Pegasus is a sequence-to-sequence model using an encoder-decoder architecture similar to BART. Pegasus is pre-trained jointly on two self-supervised objective functions. In Masked Language Modelling (MLM), encoder input tokens are randomly replaced by a mask token and must be predicted by the encoder. The other is the Gap Sentence Generation (GSG); here also, the whole encoder input sentences are replaced by a second mask token and fed to the decoder, but which uses a casual mask to hide the future words.



## 8.2 Text Generation

Unlike summarization, which tries to extract the most important information from an original document, the Large Language Model produces an entire text corpus with text generation. This output could answer a question, a classification task, or even some sentiment analysis by providing text that reflects some emotions or opinions [5].

When given the proper context, a Large Language Model with text generation will generate an entire corpus of text that fits the parameters provided; the key to the success of Text Generation models largely depends on the quality of the prompt and the context provided. Here are some text generation models provided by HuggingFace.

- **Meta-llama/Meta-Llama-3-8B**—The Llama 3 model released by the Meta company is a family of large language models, a collection of pre-trained and fine-tuned generative text models in 8B and 70B parameter sizes. The Llama 3 instruction-tuned models are optimized for dialogue use cases and outperform many available open-source LLM models on common industry benchmarks[34].
- **Apple/OpenELM-3B-Instruct**—OpenELM, a family of Open-Efficient Language Models, uses a layer-wise scaling strategy to allocate parameters within each layer of the Transformer model efficiently, enhancing accuracy [5, 33].

## 8.3 Text-To-Text Generation

Text-to-text generation is a type of text generation that involves taking an input as text and returning text as an output. This text generation model works excellently for tasks like speech-to-text and language translation but can also handle summarization and text generation quite well.

Text-to-text generation leverages large language models, such as GPT (Generative Pre-trained Transformer) models, which have been pre-trained on vast amounts of text data. HuggingFace also provides lots of pre-trained and fine-tuned LLMs that we can use to handle this task.

- **Google/flan-t5-base** – The T5 (Text-to-Text Transfer Transformer) is a series of Large Language Models developed by Google AI. T5 is an encoder-decoder model pre-trained on the Colossal Clean Crawled Corpus(C4), which generally contains

code and textual data scraped from the Internet. It also belongs to a family of models with varying parameter sizes[35].

Model	Parameters	# Layers	$d_{\text{model}}$	$d_{\text{ff}}$	$d_{\text{kv}}$	# heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
XL	3B	24	1024	16384	128	32
XXL	11B	24	1024	65536	128	128

Table 1: Google T5 family of models

From the table above, we understand the various parameters the Google AI team uses to train the T5 family of models.

- **# Layers** explain the number of encoders as well as the number of decoders used for training the model
- **$d_{\text{ff}}$**  It also explains the dimension of the feedforward network within each encoder and decoder layer.
- **$d_{\text{kv}}$**  describes the key and value vectors used in the self-attention mechanism.
- **$d_{\text{model}}$**  refers to the dimension of the embedding vectors.
- **# heads** describe the number of heads in each attention block.

## 9 IMPLEMENTATION OVERVIEW

This section of the thesis will discuss the general implementation used to achieve the results. As discussed in the section about the RAG architecture, the first procedure to examine is how to extract text from the user documents they will provide, and for that, we will use tools from the Langchain community.

### 9.1 Extracting text from PDF documents

We created a file processing function to extract text from user documents and take a path to a PDF file. We use the PyPDFLoader utility from HuggingFace to load the PDF into memory and then split it into the number of pages contained within it.

Then, we use the RecursiveCharacterTextSplitter to split the data in the pages into shorter sentences. The `file_processing` function returns an array of texts extracted from the user PDF document.

```
def file_preprocessing(filepath):  
    loader = PyPDFLoader(filepath)  
    pages = loader.load_and_split()  
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000,  
chunk_overlap=150)  
    return text_splitter.split_documents(pages)
```

### 9.2 Vector Embedding Model

After successfully extracting the text corpus from the user documents, we would generate vector embeddings from the data. This would be a high-dimensional representation of data in vectors that captures the semantic meaning of the text corpus.

```
def get_embeddings():  
    modelPath = "sentence-transformers/all-mpnet-base-v2"  
    model_kwargs = {'device': 'cpu'}  
    encode_kwargs = {'normalize_embeddings': False}  
  
    return HuggingFaceEmbeddings(
```

```
model_name=modelPath,  
model_kwargs=model_kwargs,  
encode_kwargs=encode_kwargs,  
)
```

For our vector embedding, we define a `get_embeddings` function that would use an encoder-only Transformer model to generate vector embeddings on our text corpus. Langchain provides the `HuggingFaceEmbeddings` utility that takes a few parameters to create the vector embeddings. The most important parameter here is the vector model we decided to use. `Sentence_transformers` provide a wide range of models we can use to generate the embeddings.

### 9.3 Capture Embeddings in a Vector Store

After embedding is generated, we must store the vector embeddings in a vector database. At the time of writing this thesis, there is a wide range of capable vector stores available, and for this thesis, we will be using LanceDB, which is a very competent vector store.

```
db = lancedb.connect("/tmp/lancedb")
```

LanceDB works in memory, and initializing a database is straightforward,

```
db.create_table("my_table", data={{  
    "vector": your_array_of_vector_embeddings  
    "text": your_text_corpus_parsed_into_a_string  
    "id": an_ID_for_unique, }},  
mode="overwrite", )
```

We can then store our vector embeddings into the vector database.

### 9.4 User Query and Inputs

For this thesis, we will build a user interface that allows users to add input and upload a PDF file, which we will use to retrieve augmented generation. This user interface would also expose configuration to the hyperparameters, such as the temperature, minimum token length, and maximum token length, and pick models that would be an excellent option for each use case.

With this user interface, we would use the Streamlit library, which contains many easy-to-use UI components. With streamlit, I would be building a user-friendly interface that would be intuitive. With Streamlit, user queries and documents can be easily captured.

## 9.5 Model Pipeline

As discussed earlier, HuggingFace provides an excellent pipeline tool that can be used intuitively and easily. We create a transformer function to handle custom configurations and take external parameters like the task type, model, and temperature.

```
def transformer():  
    chain = pipeline(  
        'task type',  
        model = reference to HuggingFace model,  
        min_length = min token length  
        max_new_tokens = max token length  
    )  
    return HuggingFacePipeline(  
        pipeline=chain,  
        model_kwargs={"temperature": temperature for model creativity },  
    )
```

## 9.6 Query Retrieval

Now, with the model and our vector store in place, we can adequately retrieve the information we need from the vector store using a query input from the user. Hence, the user queries will be used on the vector store, which performs a similarity search on the vector representation and returns a set of vectors with the closest matching.

For Query Retrieval, Langchain provides a great tool for retrieving embedding from the vector store.

```
db = LanceDB.from_documents(docs, embeddings)  
retriever = db.as_retriever(search_kwargs={"k": 5})  
docs = retriever.get_relevant_documents(user prompt)
```

This creates a memory reference that can be used to retrieve the embeddings. This retriever then gets the vector embeddings based on the user prompt. This prompt can then be passed to the Large Language Model for the sequence-to-sequence modelling.

## 9.7 Retrieving Output from LLM

After getting the suitable vector embeddings from the query, we can feed the LLM with these vector embeddings as context and the user prompt. This would then keep the context window needed for the LLM in check.

```
llm = transformer()  
qa = RetrievalQA.from_chain_type(llm=llm, chain_type="refine",  
retriever=retriever, return_source_documents=False)  
result = qa.invoke({"query": user prompt})  
return result["result"]
```

Here, we use the transformer function defined earlier with all the configurations. Then, we use another tool from Langchain called RetrievalQA; with this, we can easily retrieve the LLM output from our prompt.

## 10 RESULTS

When everything is put carefully together, the RAG implementation can adequately be tested for the results on different temperatures, models, and document lengths.

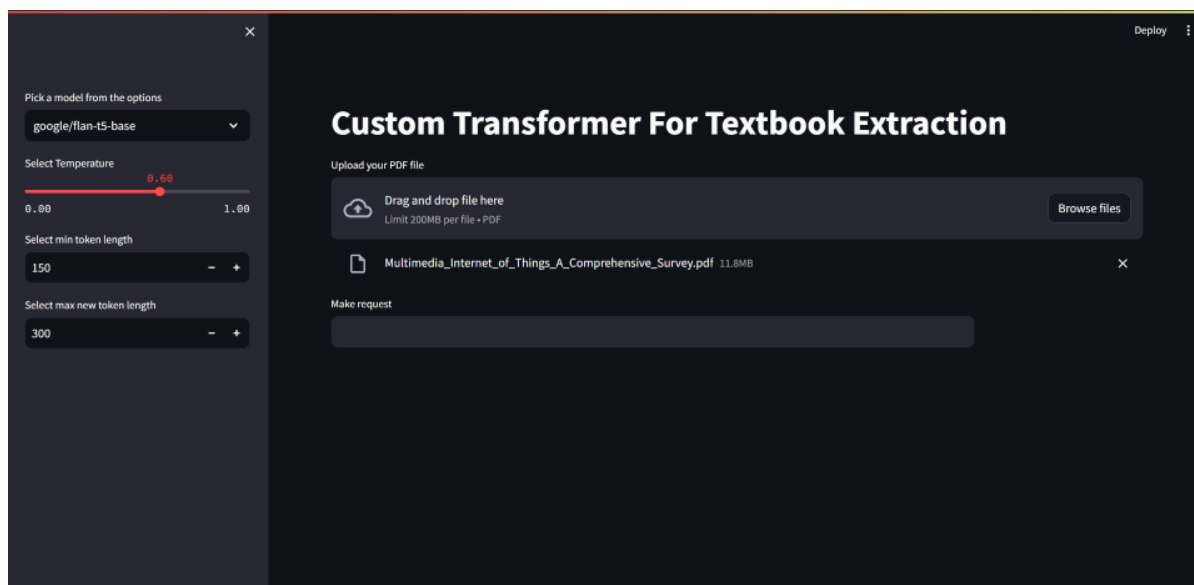


Figure 12: User Interface for RAG Implementation

We built a user interface that allows a user to access some LLM hyperparameters, input for a user query, and upload files. Hyperparameters here are passed the Transformer pipeline in the implementation above. Users can upload any PDF file with lots of text content and submit a query with the 'Make request' input, and an adequate output will be returned to the user.

### 10.1 Test Case 1

For this test case, I will use an article about agile methodologies. The model will be the Google Flan T5 LLM. This model is large, so results from the application take some time.

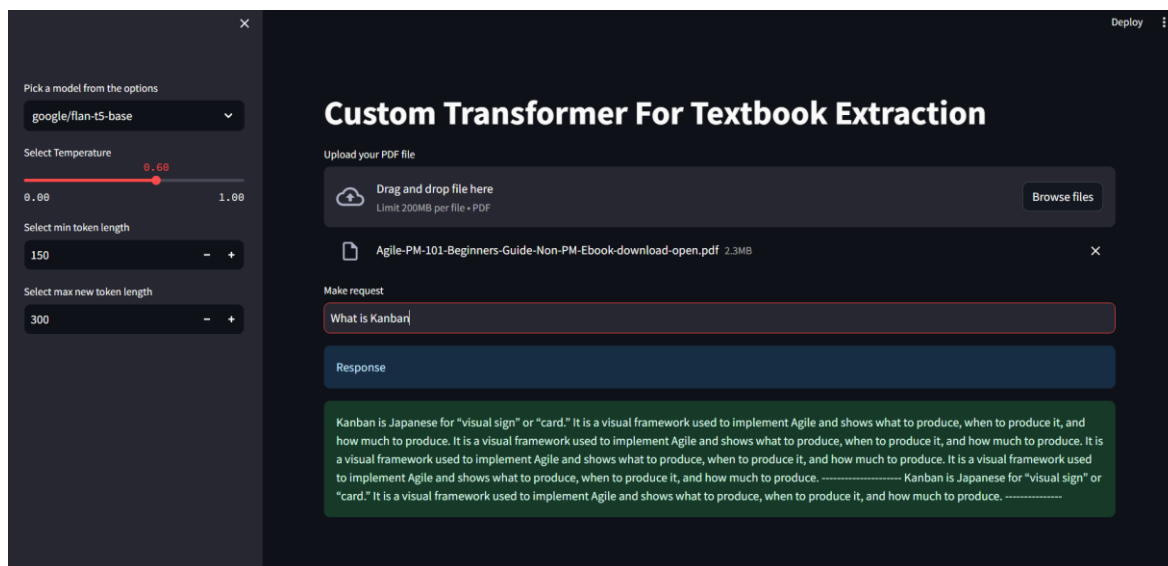


Figure 13: Test case with Google Flan T5

From the image above, we realize that the temperature for the model is set to 0.6, with a min token count of 150 and a max token count of 300. We uploaded the Agile Guide and asked it, "What is Kanban?" which is stated in the document. Results output was "Kanban is Japanese for "visual sign" or "card". It is a visual framework used to implement Agile and shows what to produce when to produce, and how much to produce.". It is a framework used to implement Agile and shows what to produce when to produce it.....". This is a bit accurate, and some text is repeated several times.

## 10.2 Test case 2

For this test case, we reduced the temperature for the model, and the results were similar. This result is observed because the text content in vector representation returned from the vector store doesn't give the LLM much room to be creative; hence, the results are usually the same, irrespective of the temperature.



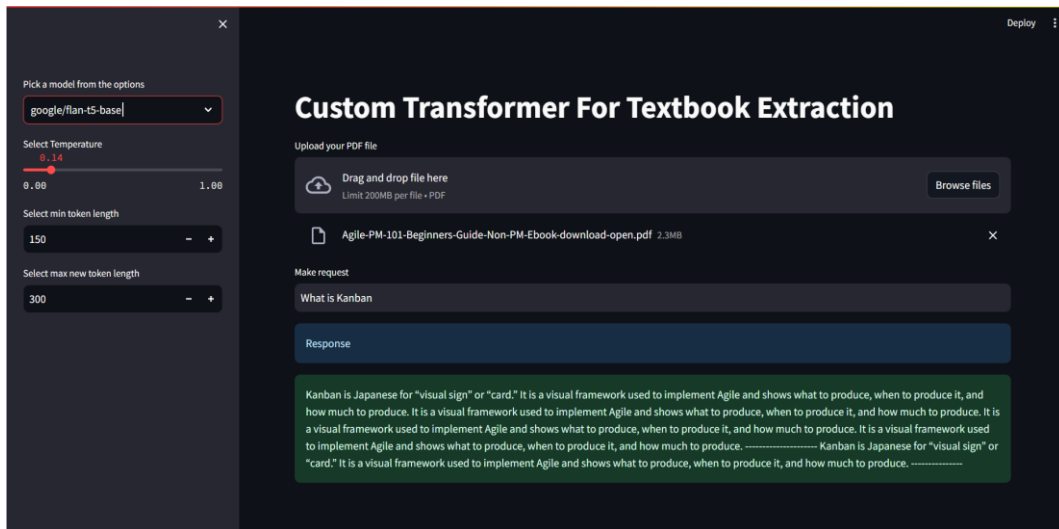


Figure 14 : Test case 2, Reduced Temperature

The image above shows that results do not change from the first instance with reduced temperatures.

### 10.3 Test case 3

For this test case, we will use the **Lamini Flan T5 238M** model. This smaller model produces faster results. The temperature worked well for this model and gave varying results.

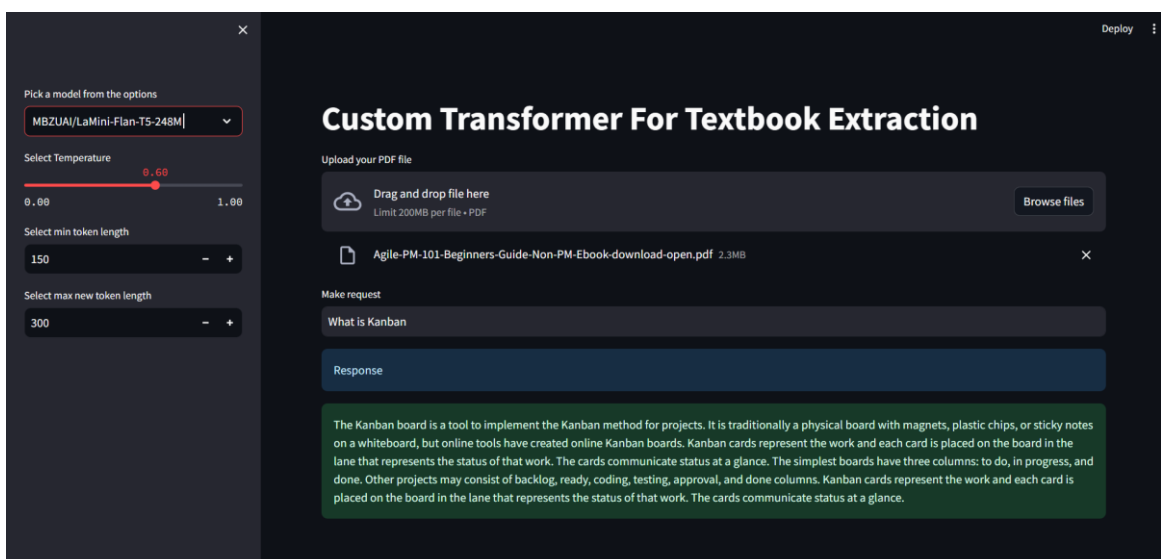
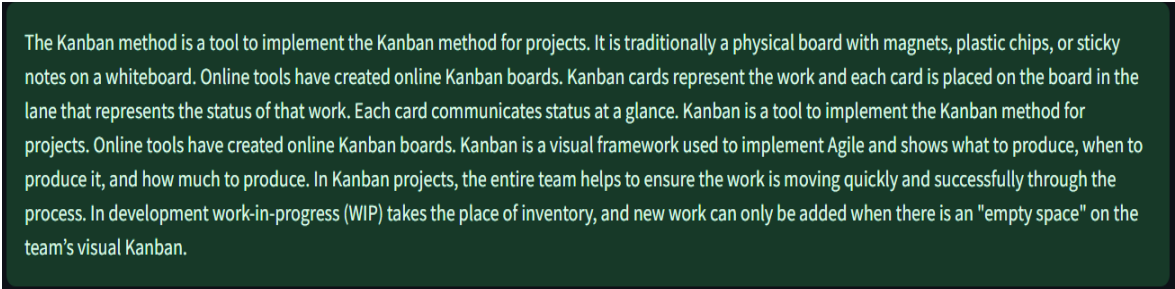


Figure 15: Test case 3 results.

When the temperature is set to **0.24**, the results for the prompt were, **“The Kanban method is a tool to implement the Kanban method for projects. It is traditionally a physical board with magnets, plastic chips or sticky notes on a whiteboard. Online tools have created online Kanban boards. Kanban cards represent the work, and each card is placed on the board in the lane that represents the status of that work. Each card communicates status at a glance. Online tools have created Kanban boards. Kanban is a tool that can be used to implement the Kanban method in projects. Kanban is a visual framework used to implement Agile and show what to produce, when, and how much to produce. In development, work-in-progress (WIP) takes the place of inventory, and new work can only be added when there is a 'space' on the team visual board.”**



The Kanban method is a tool to implement the Kanban method for projects. It is traditionally a physical board with magnets, plastic chips, or sticky notes on a whiteboard. Online tools have created online Kanban boards. Kanban cards represent the work and each card is placed on the board in the lane that represents the status of that work. Each card communicates status at a glance. Kanban is a tool to implement the Kanban method for projects. Online tools have created online Kanban boards. Kanban is a visual framework used to implement Agile and shows what to produce, when to produce it, and how much to produce. In Kanban projects, the entire team helps to ensure the work is moving quickly and successfully through the process. In development work-in-progress (WIP) takes the place of inventory, and new work can only be added when there is an "empty space" on the team's visual Kanban.

Figure 16: Test results from Temperature 0.24

When the temperature was set to **0.85**, the results for the prompt were, **“The Kanban board is a tool to implement the Kanban method for projects. It is traditionally a physical board with magnets, plastic chips, or sticky notes on a whiteboard, but online tools have created online Kanban boards. Kanban cards represent the work, and each card is placed on the board in the lane that represents the status of that work. The board should remain clear and easy to read, with the columns marked by phase with no timeframes associated. The board should remain clear and easy to read, and incorporating bells and whistles into the Kanban board buries the important information. The disadvantages of Kanban include confusion, inaccuracies, and miscommunication.”**

The Kanban board is a tool to implement the Kanban method for projects. It is traditionally a physical board with magnets, plastic chips, or sticky notes on a whiteboard, but online tools have created online Kanban boards. Kanban cards represent the work and each card is placed on the board in the lane that represents the status of that work. The board should remain clear and easy to read, with the columns marked by phase, with no timeframes associated. The board should remain clear and easy to read, and incorporating bells and whistles to the Kanban board just buries the important information. The disadvantages of Kanban include confusion, inaccuracies, and miscommunication.

Figure 17: Test results for Temperature 0.85

## CONCLUSION

Based on the research conducted in this thesis, we can adequately conclude that extracting data from textbooks using a custom Transformer model is feasible and yields promising results. Our experiments and evaluations have demonstrated the effectiveness and efficiency of the developed Transformer in accurately extracting information from various textbook sources.

With the ability to run Large Language Models locally, it is remarkable to use these models for querying tasks on custom data without relying on the large online-run LLMs. Local deployment ensures data privacy and security while enabling faster response times, making it ideal for sensitive or time-critical applications. And that, coupled with the ability to add your custom information to the LLM as input to retrieve meaningful output, is genuinely groundbreaking.

A custom Transformer model that works in the educational context can transform how we approach studying in general by streamlining the arduous tasks of going through several pages of a book to find something significant. This can efficiently save educators valuable time and empower learners with faster pertinent information. Ultimately, incorporating customized transformers into educational frameworks enables individualized learning, lowers access hurdles, and creates a more dynamic and inclusive learning environment ready to accommodate students' changing demands in the digital era.

In conclusion, developing and utilizing a custom transformer for educational text extraction marks a significant advancement in educational technology. Our research showcases its potential to streamline knowledge acquisition, improve learning outcomes, and foster inclusivity. Continuing innovation in natural language processing is vital to enhance these tools further and address evolving educational needs. By leveraging custom transformers, we can empower learners and educators, paving the way for a more efficient and effective educational landscape globally.

**BIBLIOGRAPHY**

- [1] ALEXANDER S. GILLIS. *Natural Language Processing (NLP)* [online]. únor 2024 [vid. 2024-05-03]. Dostupné z: <https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP#:~:text=NLP%20uses%20many%20different%20techniques,way%20a%20computer%20can%20understand.>
- [2] COLLOBERT, Ronan, Jason WESTON, Jweston@google.COM, Michael KARLEN, Koray KAVUKCUOGLU a Pavel KUKSA. *Natural Language Processing (Almost) from Scratch*. 2011.
- [3] KEITH D. FOOTE. A Brief History of Natural Language Processing. *dataversity.net* [online]. 6. červenec 2023 [vid. 2024-04-06]. Dostupné z: <https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/>
- [4] VASWANI, Ashish, Google BRAIN, Noam SHAZEER, Niki PARMAR, Jakob USZKOREIT, Llion JONES, Aidan N GOMEZ, Łukasz KAISER a Illia POLOSUKHIN. *Attention Is All You Need*. 2017.
- [5] WOLF, Thomas, Lysandre DEBUT, Victor SANH, Julien CHAUMOND, Clement DELANGUE, Anthony MOI, Pierric CISTAC, Tim RAULT, Rémi LOUF, Morgan FUNTOWICZ, Joe DAVISON, Sam SHLEIFER, Patrick VON PLATEN, Clara MA, Yacine JERNITE, Julien PLU, Canwen XU, Teven Le SCAO, Sylvain GUGGER, Mariama DRAME, Quentin LHOEST a Alexander M. RUSH. *HuggingFace's Transformers: State-of-the-art Natural Language Processing* [online]. 2019. Dostupné z: <http://arxiv.org/abs/1910.03771>
- [6] WU, Tianyu, Shizhu HE, Jingping LIU, Siqi SUN, Kang LIU, Qing Long HAN a Yang TANG. A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development. *IEEE/CAA Journal of Automatica Sinica* [online]. 2023, **10**(5), 1122–1136. ISSN 23299274. Dostupné z: doi:10.1109/JAS.2023.123618
- [7] KIYANI, Farzad a Oguzhan TAS. A survey automatic text summarization. *Pressacademia* [online]. 2017, **5**(1), 205–213. ISSN 2146-7943. Dostupné z: doi:10.17261/pressacademia.2017.591
- [8] SALEHINEJAD, Hojjat, Sharan SANKAR, Joseph BARFETT, Errol COLAK a Shahrokh VALAEE. *Recent Advances in Recurrent Neural Networks* [online]. 2017. Dostupné z: <http://arxiv.org/abs/1801.01078>
- [9] GALLO, Matej. *Text Summarization by Machine Learning*. nedatováno.
- [10] ÇAKIR, Ahmet. *Usability and accessibility of portable document format* [online]. B.m.: Taylor and Francis Ltd. 2. duben 2016. ISSN 13623001. Dostupné z: doi:10.1080/0144929X.2016.1159049
- [11] DEHRU, Virender, Pradeep Kumar TIWARI, Gaurav AGGARWAL, Bhavya JOSHI a Pawan KARTIK. Text Summarization Techniques and Applications. *IOP Conference Series: Materials Science and Engineering* [online]. 2021, **1099**(1), 012042. ISSN 1757-8981. Dostupné z: doi:10.1088/1757-899x/1099/1/012042
- [12] NAGESH SINGH CHAUHAN. Introduction to RNN and LSTM. *The AI Dream* [online]. 29. listopad 2020 [vid. 2024-05-04]. Dostupné z: <https://www.theaidream.com/post/introduction-to-rnn-and-lstm>
- [13] KIM, Young Jin a Hany Hassan AWADALLA. *FastFormers: Highly Efficient Transformer Models for Natural Language Understanding* [online]. 2020. Dostupné z: <http://arxiv.org/abs/2010.13382>
- [14] WOLF, Thomas, Lysandre DEBUT, Victor SANH, Julien CHAUMOND, Clement DELANGUE, Anthony MOI, Pierric CISTAC, Tim RAULT, Rémi LOUF, Morgan

- FUNTOWICZ, Joe DAVISON, Sam SHLEIFER, Patrick VON PLATEN, Clara MA, Yacine JERNITE, Julien PLU, Canwen XU, Teven LE SCAO, Sylvain GUGGER, Mariama DRAME, Quentin LHOEST a Alexander M RUSH. *Transformers: State-of-the-Art Natural Language Processing* [online]. nedatováno. Dostupné z: <https://github.com/huggingface/>
- [15] MEDSKER, LR, DC LC JAIN a Boca RATON LONDON NEW YORK WASHINGTON. *RECURRENT NEURAL NETWORKS Edited by Design and Applications*. 2001.
- [16] LEWIS, Mike, Yinhan LIU, Naman GOYAL, Marjan GHAZVININEJAD, Abdelrahman MOHAMED, Omer LEVY, Ves STOYANOV a Luke ZETTLEMOYER. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension [online]. 2019. Dostupné z: <http://arxiv.org/abs/1910.13461>
- [17] WEN, Qingsong, Tian ZHOU, Chaoli ZHANG, Weiqi CHEN, Ziqing MA, Junchi YAN a Liang SUN. Transformers in Time Series: A Survey [online]. 2022. Dostupné z: <http://arxiv.org/abs/2202.07125>
- [18] KALYAN, Katikapalli Subramanyam, Ajit RAJASEKHARAN a Sivanesan SANGEETHA. AMMUS: A Survey of Transformer-based Pretrained Models in Natural Language Processing [online]. 2021. Dostupné z: <http://arxiv.org/abs/2108.05542>
- [19] BHATTAMISHRA, Satwik, Arkil PATEL, Phil BLUNSOM a Varun KANADE. Understanding In-Context Learning in Transformers and LLMs by Learning to Learn Discrete Functions [online]. 2023. Dostupné z: <http://arxiv.org/abs/2310.03016>
- [20] GILLIOZ, Anthony, Jacky CASAS, Elena MUGELLINI a Omar Abou KHALED. Overview of the Transformer-based Models for NLP Tasks. In: *Proceedings of the 2020 Federated Conference on Computer Science and Information Systems, FedCSIS 2020* [online]. B.m.: Institute of Electrical and Electronics Engineers Inc., 2020, s. 179–183. ISBN 9788395541674. Dostupné z: doi:10.15439/2020F20
- [21] INTELLIGENCE, Artificial a Jennifer D' SOUZA. *A Review of Transformer Models* [online]. nedatováno. Dostupné z: <https://orkg.org/comparison/R609226/>
- [22] USMAN HADI, Muhammad, qasem AL TASHI, Rizwan QURESHI, Abbas SHAH, amgad MUNEER, Muhammad IRFAN, Anas ZAFAR, Muhammad BILAL SHAIKH, Naveed AKHTAR, Jia WU, Seyedali MIRJALILI, Qasem AL-TASHI, Amgad MUNEER a Mohammed ALI AL-GARADI. Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects Large Language Models: A Comprehensive Survey of Applications, Challenges, Limitations, and Future Prospects [online]. nedatováno. Dostupné z: doi:10.36227/techrxiv.23589741.v4
- [23] TOPSAKAL, Oguzhan a Tahir Cetin AKINCI. *Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast* [online]. nedatováno. Dostupné z: <http://as-proceeding.com/:Konya,Turkeyhttps://www.icaens.com/>
- [24] ZHANG, Mengli, Gang ZHOU, Wanting YU, Ningbo HUANG a Wenfen LIU. *A Comprehensive Survey of Abstractive Text Summarization Based on Deep Learning* [online]. B.m.: Hindawi Limited. 2022. ISSN 16875273. Dostupné z: doi:10.1155/2022/7132226
- [25] JURAFSKY, Daniel a James H MARTIN. *Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition Third Edition draft*. nedatováno.

- [26] HAN, Yikun, Chunjiang LIU a Pengfei WANG. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge [online]. 2023. Dostupné z: <http://arxiv.org/abs/2310.11703>
- [27] JING, Zhi, Yongye SU, Yikun HAN, Bo YUAN, Haiyun XU, Chunjiang LIU, Kehai CHEN a Min ZHANG. When Large Language Models Meet Vector Databases: A Survey [online]. 2024. Dostupné z: <http://arxiv.org/abs/2402.01763>
- [28] GAO, Yunfan, Yun XIONG, Xinyu GAO, Kangxiang JIA, Jinliu PAN, Yuxi BI, Yi DAI, Jiawei SUN, Meng WANG a Haofen WANG. Retrieval-Augmented Generation for Large Language Models: A Survey [online]. 2023. Dostupné z: <http://arxiv.org/abs/2312.10997>
- [29] LEWIS, Patrick, Ethan PEREZ, Aleksandra PIKTUS, Fabio PETRONI, Vladimir KARPUKHIN, Naman GOYAL, Heinrich KÜTTLER, Mike LEWIS, Wen-Tau YIH, Tim ROCKTÄSCHEL, Sebastian RIEDEL a Douwe KIELA. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* [online]. nedatováno. Dostupné z: <https://github.com/huggingface/transformers/blob/master/>
- [30] LIU, Shangqing, Yu CHEN, Xiaofei XIE, Jingkai SIOW a Yang LIU. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN [online]. 2020. Dostupné z: <http://arxiv.org/abs/2006.05405>
- [31] MARTINEAU K. What is Retrieval -Augmented Generation (RAG). *IBM* [online]. 2023 [vid. 2024-05-06]. Dostupné z: <https://research.ibm.com/blog/retrieval-augmented-generation-RAG>
- [32] ALI GLEN. Mastering Retrieval Augmented Generation(RAG) Architecture. *Stackademic* [online]. 8. duben 2024 [vid. 2024-05-06]. Dostupné z: <https://blog.stackademic.com/mastering-retrieval-augmented-generation-rag-architecture-unleash-the-power-of-large-language-a1d2be5f348c>
- [33] JIANG, Wenxin, Nicholas SYNOVIC, Matt HYATT, Taylor R. SCHORLEMMER, Rohan SETHI, Yung Hsiang LU, George K. THIRUVATHUKAL a James C. DAVIS. An Empirical Study of Pre-Trained Model Reuse in the Hugging Face Deep Learning Model Registry. In: *Proceedings - International Conference on Software Engineering* [online]. B.m.: IEEE Computer Society, 2023, s. 2463–2475. ISBN 9781665457019. Dostupné z: doi:10.1109/ICSE48619.2023.00206
- [34] HUGGING FACE. Meta Llama- Meta Llama 8B. *Hugging Face* [online]. duben 2024 [vid. 2024-05-07]. Dostupné z: <https://huggingface.co/meta-llama/Meta-Llama-3-8B>
- [35] CHUNG, Hyung Won, Le HOU, Shayne LONGPRE, Barret ZOPH, Yi TAY, William FEDUS, Yunxuan LI, Xuezhi WANG, Mostafa DEGHANI, Siddhartha BRAHMA, Albert WEBSON, Shixiang Shane GU, Zhuyun DAI, Mirac SUZGUN, Xinyun CHEN, Aakanksha CHOWDHERY, Alex CASTRO-ROS, Marie PELLAT, Kevin ROBINSON, Dasha VALTER, Sharan NARANG, Gaurav MISHRA, Adams YU, Vincent ZHAO, Yanping HUANG, Andrew DAI, Hongkun YU, Slav PETROV, Ed H. CHI, Jeff DEAN, Jacob DEVLIN, Adam ROBERTS, Denny ZHOU, Quoc V. LE a Jason WEI. Scaling Instruction-Finetuned Language Models [online]. 2022. Dostupné z: <http://arxiv.org/abs/2210.11416>

**LIST OF ABBREVIATIONS**

NLP – Natural Language Processing

NLU – Natural Language Understanding

AI – Artificial Intelligence

LLM – Large Language Model

RNN – Recurrent Neural Network

CNN – Convolutional Neural Network

LSTM – Long Short-Term Memory

PCFG – Probabilistic Context-Free Grammar

HMM – Hidden Markov Model

GRU – Gated Recurrent Units

FFN – Feed Forward Network

API – Application Programming Interface

MLM – Masked Language Modelling

GSG – Gap Sentence Generation

CPU – Central Processing Unit

GPU – Graphical Processing Unit



**LIST OF FIGURES**

Figure 1: Recurrent Neural Network .....	14
Figure 2: Regex Based Parsing .....	16
Figure 3: Single RNN cell.....	17
Figure 4: Long Short-Term Memory Architecture. ....	18
Figure 5: An example of Probabilistic Context-Free Grammar.....	19
Figure 6: An example of the Hidden Markov Model.....	20
Figure 7: Transformer Model Architecture.....	21
Figure 8: Flan T5 model.....	26
Figure 9: RAG Architecture.....	33
Figure 10: HuggingFace Model Hub .....	36
Figure 11: Vector Embeddings .....	37
Figure 12: User Interface for RAG Implementation.....	47
Figure 13: Test case with Google Flan T5 .....	48
Figure 15 : Test case 2, Reduced Temperature.....	49
Figure 16: Test case 3 results.....	49
Figure 17: Test results from Temperature 0.24.....	50
Figure 18: Test results for Temperature 0.85.....	51

## LIST OF TABLES

Table 1: Google T5 family of models.....	42
--	----

## APPENDICES

[https://python.langchain.com/docs/modules/data\\_connection/document\\_loaders/pdf/](https://python.langchain.com/docs/modules/data_connection/document_loaders/pdf/)

<https://www.python.org/>

<https://huggingface.co/models>

<https://lancedb.github.io/lancedb/>

<https://streamlit.io/>

<https://github.com/JohnTawiah19/Lamini-test>