



GRADUATE CERTIFICATE INTELLIGENT REASONING SYSTEM (IRS)

ParliGPT

28-APR-2024

Team Members

YONG TING RUI

E1339479

Table of Contents

Table of Figures.....	2
Table of Tables	2
1 Executive Summary.....	3
1.1 Introduction.....	3
1.2 Key Findings.....	3
1.3 Recommendation	3
1.4 Conclusion	3
2 Introduction	4
2.1 Background & Problem Statement.....	4
2.2 Analysis of Existing Solutions.....	4
2.3 Project Objectives.....	5
2.4 Measure of Effectiveness.....	6
3 Project Solution.....	7
3.1 System Design.....	7
3.2 RAG Architecture	8
3.2.1 Hybrid Retrieval Approach – Fused Retriever.....	9
3.2.2 Retrieve and Rerank Method.....	10
3.2.3 Contextual Filtering & Compression	10
3.2.4 Response Generation	11
3.3 Data Management.....	11
3.3.1 Document Processing & Chunking	11
3.3.2 Vector Database for Document Chunks.....	11
3.3.3 Text Preprocessing for NLP	12
3.3.4 Relational Database for Word Tokens.....	12
3.4 Application Workflow	13
4 System Implementation.....	14
4.1 Sparse Retrieval – BM25+.....	14
4.2 Dense Passage Retrieval	14
4.3 Document Reranking	15
4.4 Prompt Engineering – Contextual Filtering & Compression	15
5 Conclusion.....	17
5.1 Challenges.....	17
5.2 Future Works	17
6 References.....	18
7 Appendix A: Project Proposal	19
8 Appendix B: Techniques and Skills.....	24

9	Appendix C: Installation and User Guide	25
9.1	Prerequisites	25
9.2	Web Scraper Setup (Optional)	25
9.3	Data Extraction Setup (Optional)	26
9.4	Application Setup	27
10	Appendix D: Individual Reports	29

Table of Figures

Figure 1: ParliaGPT System Design	7
Figure 2: ParliaGPT's RAG Architecture Design	9
Figure 3: Demonstration of Contextual Compression & Filtering	10
Figure 4: Data Management Strategy	11
Figure 5: Example of Datapoints Stored in Vector Database (ChromaDB)	12
Figure 6: Example of Word Tokens Stored in DuckDB	13
Figure 7: Prompt Engineering Excerpt for Contextual Compression & Filtering	16

Table of Tables

Table 1: Competitor Solutions' Strengths and Weakness Analysis	5
Table 2: Backend Sub-Modules and Elaboration	8

1 Executive Summary

1.1 Introduction

ParliaGPT is an FAQ Bot that answers questions and provides relevant contextual pages using parliamentary debate documents as contexts. This report details on the development and implementation process, capabilities, market analysis and future developments of ParliaGPT.

1.2 Key Findings

- **Trends and Challenge:** There is a growing importance for the general public to stay relevant in politics and current affairs in today's fast paced and uncertain world. Obtaining reliable information on current affairs from official sources remains a challenge as the process is tedious due to the length of these documents.
- **Market Competitors:** Analysis of the strengths and weaknesses of existing solutions that are available in the market.
- **Technologies Involved:** Leveraging on Large Language Models to generate natural language responses to user queries; using information retrieval methods to provide transparency in the data used for natural language responses.

1.3 Recommendation

- **Inference Enhancement:** Accelerating response generation by implementing light-weight techniques to reduce computation workloads with LLMs, as well as using response caching to reduce inferencing when not needed.
- **Feedback Mechanism:** Incorporate voting mechanisms for users to feedback on responses generated for further product enhancement and training of models.

1.4 Conclusion

ParliaGPT represents a new way for the general public to obtain and access information from government sources, offering users an intuitive and transparent access to a large database of documents with natural language responses.

2 Introduction

2.1 Background & Problem Statement

In today's fast-paced and uncertain world, keeping up to date with current affairs is especially crucial, as information travels at lightning speed and events unfold rapidly. By staying relevant on current affairs, individuals can better adapt and respond to changes in their environment, allowing them to navigate the increasingly complex landscape with clarity and confidence.

At present, the main source of reliable information to consume current affairs comes from mainstream media. However, it summarizes on the policies and legislations that would come into effect, omitting discussion / decision-making processes related to the policies and legislations. For additional information, individuals would have to turn to watching the parliamentary debate videos, or reading the parliamentary debate minutes.

Based on current available means, obtaining specific information from parliamentary debates is highly tedious and extremely time consuming, as it entails being highly informed about the contents of each parliamentary debate to find the appropriate minutes and videos for reference.

2.2 Analysis of Existing Solutions

An exploration was done to seek out some of the existing solutions that are available to the general public, and we look at what are the current strengths and weakness to identify opportunities for our product as a competitor.

Solution	Strengths	Weaknesses
ChatGPT	<ul style="list-style-type: none"> Allows users to make natural language queries Provides users with responses in natural language 	<ul style="list-style-type: none"> Unable to provide information on events / data that it has not been trained on
Search Engines (Google, Bing, Yahoo, Duckduckgo, etc)	<ul style="list-style-type: none"> Allows users to search and obtain extensive results from public domain 	<ul style="list-style-type: none"> Users have to deep dive into search results to check if the information provided is desired Information may come from alternative sources instead of trusted source

Table 1: Competitor Solutions' Strengths and Weakness Analysis

2.3 Project Objectives

With the problem statement defined and an understanding of the existing solutions in the market, we created the following objective statement:

"How can the process of finding information from parliamentary debate documents be streamlined, so that users can quickly obtain a response, and more information from trusted sources for future deep dive?"

The objective statement serves to guide the features of our product, ParliaGPT, and to differentiate it from its' competitors. ParliaGPT is designed to be a parliamentary debate FAQ Bot for use by general members of the public that have a keen interest in current affairs. In essence, it will contain two main features:

- A retriever module that retrieves relevant parliamentary debate documents and pages from user queries
- A text generator that synthesizes a summary from the relevant parliamentary debate pages that had been retrieved

Users will be able to access ParliaGPT through the frontend web-interface without the need for any login credentials, as the solution is intended to serve users with finding only information from official reports of parliamentary debate minutes.

2.4 Measure of Effectiveness

Based on the objective statement that had been defined in the earlier section. We had identified the following as metrics to measure the effectiveness of ParliaGPT:

- User adoption – This metric measures how often FAQ Bot is used for obtaining information from parliamentary documents. Available metrics include site visit count, query execution count, etc. Logging can also be introduced to identify the queries that had been executed by users for further refinement and future enhancements
- Retrieval Relevance – Documents retrieved and shown to users must contain information that is relevant to users' query. Should users ask irrelevant questions, the FAQ Bot should recognize it to be an irrelevant question and not retrieve any documents
- Response Efficiency – The FAQ Bot must be able to respond to user queries quickly. Efficiency of the FAQ Bot comes in 2 areas:
 - Retrieval efficiency: Techniques used to retrieve and extract relevant content from documents
 - Generation efficiency: Speed at which the LLM generates a response, derived from balancing LLM size with hardware

3 Project Solution

3.1 System Design

The system design diagram in the Figure below illustrates the components that will be used for the development of ParliaGPT.

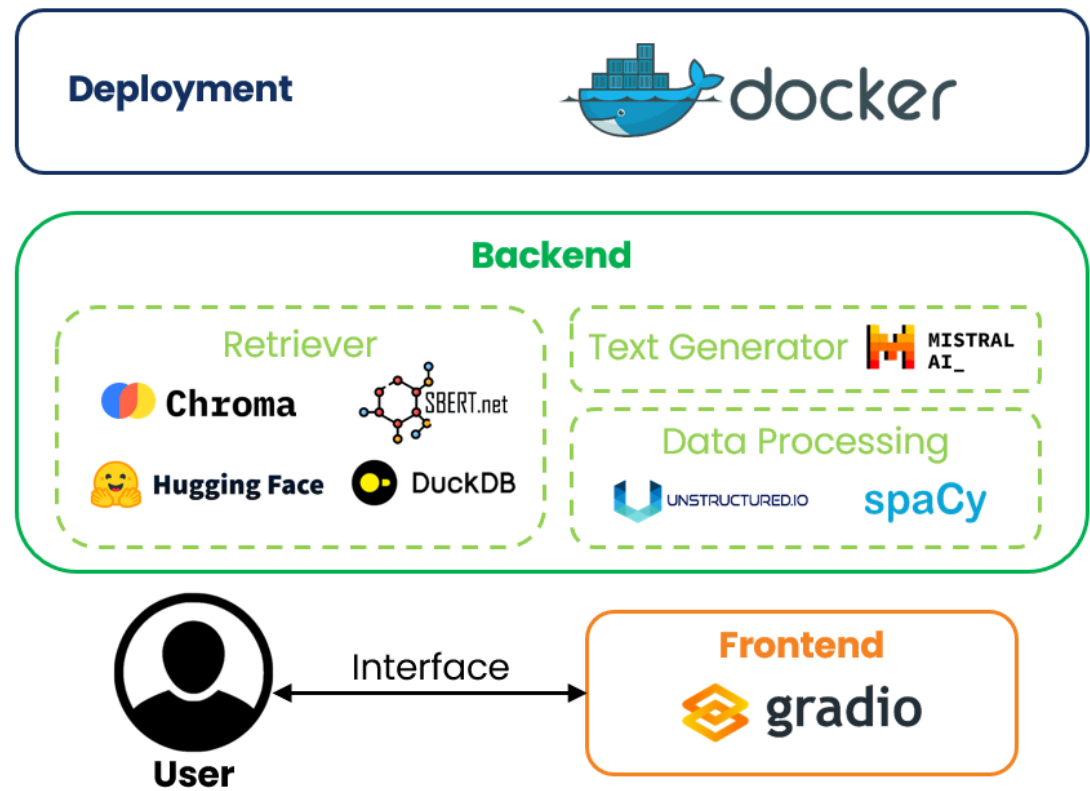


Figure 1: ParliaGPT System Design

As ParliaGPT is purposefully built as an FAQ Bot for querying only parliamentary debate documents as its ground truth, it has built as a standalone application.

ParliaGPT leverages on gradio for its frontend user interface, as it allows rapid prototyping to let users test out and to provide feedback on ParliaGPT’s features, as well as to advise on future desired features for further frontend enhancement.

For ParliaGPT’s backend, it has been divided into 3 sub-modules with its respective elaboration in the Table below.

Sub-Module	Elaboration
Data Processing	<ul style="list-style-type: none"> The data processing submodule focuses on processing raw parliamentary debate documents (unstructured data) into structured data for ingestion into databases. The submodule operates through the use of 2 components, UnstructuredIO and spaCy. <ul style="list-style-type: none"> UnstructuredIO is an open-source Python library that contains components for ingesting and pre-processing images and text documents (ie: PDFs, HTML, Word documents, etc). spaCy is a Python library for advanced natural language processing tasks. It is utilized for performing post-processing of extracted texts to carry out sparse retrieval.
Retriever	<ul style="list-style-type: none"> The retriever submodule focuses on the retrieval of documents that are relevant to the user query. In ParliaGPT, a hybrid retrieval & rerank approach is used for document retrieval, requiring both a vector database and a relational database to operate. <ul style="list-style-type: none"> ChromaDB is an open-source vector database that allows documents to be embedded with custom embedding models. DuckDB is an in-process OLAP database that allows for fast execution of SQL queries to retrieve data. HuggingFace is a centralized repository for machine learning (ML) models and Large Language Models (LLMs). SentenceTransformers is a Python framework for sentence and text embeddings, used for performing dense passage retrieval and re-ranking.
Text Generator	<ul style="list-style-type: none"> The text generator submodule focuses on the generation of natural language response from retrieved documents. <ul style="list-style-type: none"> Mistral AI is an artificial intelligence company that develops LLMs for commercial uses. The company is known for its high performing open source LLMs Mistral-7B and Mixtral-8x7B.

Table 2: Backend Sub-Modules and Elaboration

For deployment, the application shall leverage on docker to be built into a docker image. This facilitates with the deployment of the application as a standalone container that can be loaded either on-premises, or deployed onto a server with an exposed port for access by users.

3.2 RAG Architecture

As indicated during the analysis of existing solutions, a key challenge with generating natural language responses from LLMs is that it is unable to respond to questions that are not

contained within its' training data. In short, LLMs would not be able to answer questions from recent parliamentary debate minutes as the information is too new and was not part of the LLMs training data. [1]

To combat this challenge, Retrieval-Augmented Generation (RAG) is utilized so that LLMs are able to reference knowledge from outside of its training data to generate a response. In ParliaGPT, the Figure below illustrates the RAG architecture developed to reference relevant parliamentary debate documents for generating response to user queries.

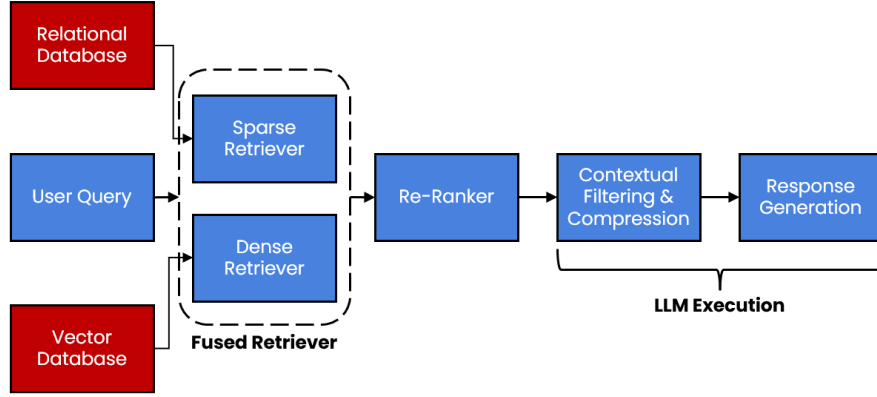


Figure 2: ParliaGPT's RAG Architecture Design

3.2.1 Hybrid Retrieval Approach – Fused Retriever

ParliaGPT's RAG architecture uses a hybrid retrieval approach for the retrieval of document chunks from user queries. Hybrid retrieval refers to the use of both sparse retrieval techniques (i.e.: TF-IDF, BM25, etc) and dense passage retrieval techniques (i.e.: word embeddings) to enhance document retrieval performance, as compared to using either one of the approaches on its own. [2] To aggregate the impact of both approaches, a weighted combination of the similarity scores from both approaches can be used as the ranking function, as represented by the following:

$$S_{hybrid}(D, Q) = \alpha Sim(v_{dense}(D), v_{dense}(Q)) + (1 - \alpha) Sim(v_{sparse}(D), v_{sparse}(Q))$$

where:

- D represents the document and Q represents the user's query
- v_{dense} and v_{sparse} represent the dense and sparse embedding functions respectively
- Sim represents the cosine similarity measuring the angle between the vector embeddings
- α represents the hyperparameter (weight) to be tuned by the user

3.2.2 Retrieve and Rerank Method

Although retrieval is able to extract document chunks that are similar to the query, the most relevant documents may not be ordered at the top for response generation. To prioritize the most relevant information, document chunks retrieved from hybrid retrieval would be re-ranked using a reranker to identify document chunks that are both similar and relevant to the users' query. [3]

3.2.3 Contextual Filtering & Compression

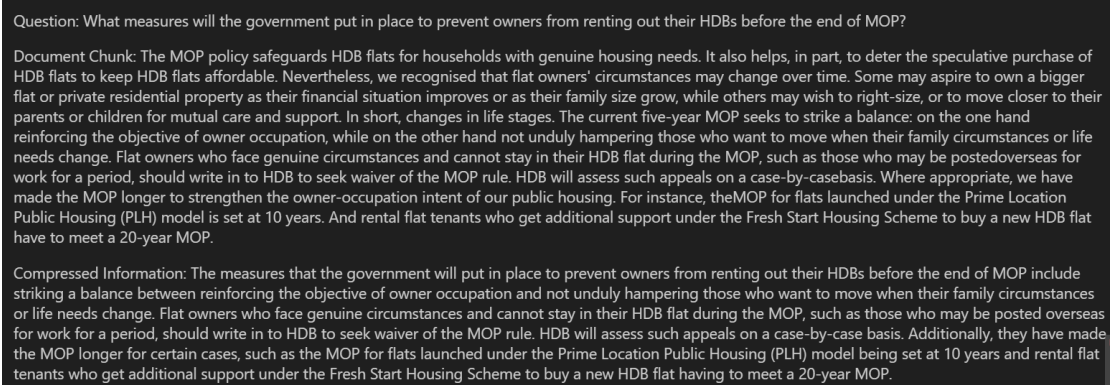
Although retrieval performance can be enhanced with the retrieve and rerank method, it does not eliminate the irrelevant contents found within document chunks – stems from the strategy of chunking documents by chunk length. Ideally, contents that are passed into LLM should be clean, to avoid LLM hallucination and using irrelevant contents in response generation.

To ensure that only contents that are relevant to the query in each document chunk are used for response generation, we apply a technique known as contextual filtering & compression.

Contextual compression & filtering is a two-step process which involves:

- Compressing the reranked documents into relevant text segments
- Filtering to identify reranked documents that are useful for generating a response to the query

The purpose of this approach recognizes that documents stored in the database may not be entirely useful for responding to queries; instead, only snippets of the document are useful for generating answers. [4] The approach uses Large Language Models (LLMs) extract these document snippets via prompt engineering, in which a set of instructions and examples are passed into the LLM as a prompt, guiding the LLM to fulfilling the desired task.



Question: What measures will the government put in place to prevent owners from renting out their HDBs before the end of MOP?

Document Chunk: The MOP policy safeguards HDB flats for households with genuine housing needs. It also helps, in part, to deter the speculative purchase of HDB flats to keep HDB flats affordable. Nevertheless, we recognised that flat owners' circumstances may change over time. Some may aspire to own a bigger flat or private residential property as their financial situation improves or as their family size grow, while others may wish to right-size, or to move closer to their parents or children for mutual care and support. In short, changes in life stages. The current five-year MOP seeks to strike a balance: on the one hand reinforcing the objective of owner occupation, while on the other hand not unduly hampering those who want to move when their family circumstances or life needs change. Flat owners who face genuine circumstances and cannot stay in their HDB flat during the MOP, such as those who may be posted overseas for work for a period, should write in to HDB to seek waiver of the MOP rule. HDB will assess such appeals on a case-by-case basis. Where appropriate, we have made the MOP longer to strengthen the owner-occupation intent of our public housing. For instance, the MOP for flats launched under the Prime Location Public Housing (PLH) model is set at 10 years. And rental flat tenants who get additional support under the Fresh Start Housing Scheme to buy a new HDB flat have to meet a 20-year MOP.

Compressed Information: The measures that the government will put in place to prevent owners from renting out their HDBs before the end of MOP include striking a balance between reinforcing the objective of owner occupation and not unduly hampering those who want to move when their family circumstances or life needs change. Flat owners who face genuine circumstances and cannot stay in their HDB flat during the MOP, such as those who may be posted overseas for work for a period, should write in to HDB to seek waiver of the MOP rule. HDB will assess such appeals on a case-by-case basis. Additionally, they have made the MOP longer for certain cases, such as the MOP for flats launched under the Prime Location Public Housing (PLH) model being set at 10 years and rental flat tenants who get additional support under the Fresh Start Housing Scheme to buy a new HDB flat having to meet a 20-year MOP.

Figure 3: Demonstration of Contextual Compression & Filtering

3.2.4 Response Generation

With relevant contents in each document chunk extracted, they are fed into a LLM for response generation. The LLM is then instructed to synthesize a response to answer the users' query by referencing only relevant contents extracted from the previous steps, before returning the generated response to the end-user via the frontend user interface.

3.3 Data Management

The data source for ParliaGPT comes from official reports of parliamentary debate minutes that has published in the public domain in this [link](#). To automate the retrieval of documents from public domain, we use Selenium scripts to scrape the domain. Following which, we convert the documents into a data source for RAG by processing the data and store it into a vector database and a relational database for hybrid retrieval.

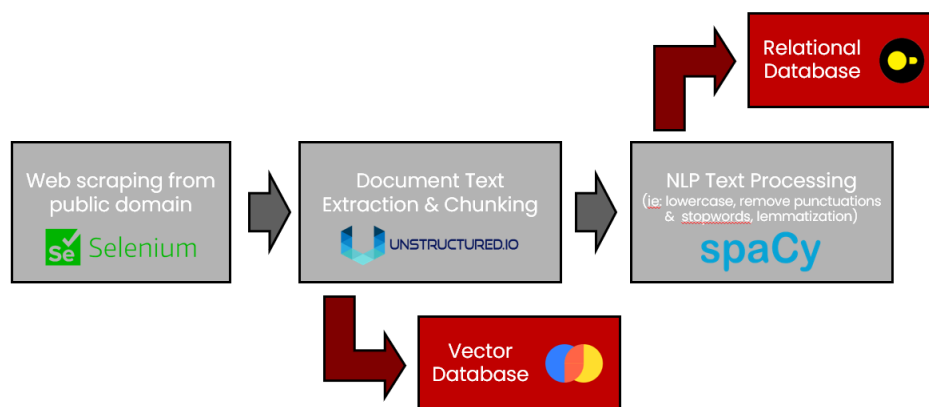


Figure 4: Data Management Strategy

3.3.1 Document Processing & Chunking

To process the unstructured data (i.e.: PDF documents), we use UnstructuredIO's partition pdf feature, which extracts document elements in accordance with its layout information. The documents are then chunked into a length of not more than 2000 characters (approximately 400 tokens) and an overlap length of 200 characters. The chunk length has been strategically chosen in view of the limited context window of 512 tokens for both the selected document embedding and reranker models.

3.3.2 Vector Database for Document Chunks

To perform dense passage retrieval on document chunks (elaborated in Section 4.2),

ChromaDB was used as the vector database for storage of text embeddings and its metadata. Text embeddings are dense vector representations of text in the vector space; it is derived from the development of transformer-based architectures such as GTE (General Text Embedding) and BERT (Bi-directional Encoder Representations from Transformers), allowing words in text to be converted into word vectors and pooled together to create dense vector representations.

```
{'ids': ['id_100', 'id_20', 'id_50'],
'embeddings': None,
'metadatas': [{'filename': '20230109.pdf', 'page_number': '29'},
{'filename': '20230109.pdf', 'page_number': '7'},
{'filename': '20230109.pdf', 'page_number': '15'}],
'documents': ['So, that is the supply side in terms of physical capacity. A second
"The passing of SGT1 Edward Go is the first SCDF fatality in a firefighting operat
"The MOP policy safeguards HDB flats for households with genuine housing needs. It
'uris': None,
'data': None}
```

Figure 5: Example of Datapoints Stored in Vector Database (ChromaDB)

In ParliaGPT, document chunks are embedded using GTE model, converting each document chunk into a 512-dimension dense vector before stored it in a ChromaDB collection.

3.3.3 Text Preprocessing for NLP

Before sparse retrieval can be applied to document chunks, text preprocessing has to be applied to convert document chunks into word tokens. This involves the following steps, which will be facilitated through the use of spaCy package:

- Remove all HTML tags (if present)
- Remove all punctuation marks
- Convert all words into lowercase
- Convert document chunks into token lists – tokenization
- Remove stop words
- Convert all words into its lemma – lemmatization

3.3.4 Relational Database for Word Tokens

To apply sparse retrieval on word tokens, we use DuckDB as the relational database for storage of word tokens and its associated document chunks. This also allows for indexing of word token lists, for ease of retrieving specific data points upon query.

text varchar	filename varchar	page_number varchar	id varchar	lemmas varchar
The passing of SGT...	20230109.pdf	7	id_20	passing sgt1 edward scdf fatality firefighting opera...
ESG has worked wit...	20230109.pdf	9, 10	id_30	esg work goldbell year familiar goldbell plan goldbe...
Mr Speaker: Mr Leo...	20230109.pdf	12	id_40	mr speaker mr leon perera mr leon perera aljunie tha...
The MOP policy saf...	20230109.pdf	15	id_50	mop policy safeguard hdb flat household genuine hous...
ENFORCEMENT AGAIN...	20230109.pdf	18	id_60	enforcement business raise price gst increase excuse...
Mr Speaker, Sir, o...	20230109.pdf	20, 21	id_70	mr speaker sir current measure control number travel...
To be honest, COVI...	20230109.pdf	23, 24	id_80	honest vaccination new norm like endemic disease lik...
Mr Speaker: Ms Ng ...	20230109.pdf	27	id_90	mr speaker ms ng ling ling ms ng ling ling ang mo ki...
So, that is the su...	20230109.pdf	29	id_100	supply term physical capacity second element manpowe...

Figure 6: Example of Word Tokens Stored in DuckDB

3.4 Application Workflow

The workflow of ParliaGPT is as follows:

1. Users will be able to interface with ParliaGPT through the gradio frontend and type in their queries related to parliamentary debate documents
2. Retriever module operates on the backend to retrieve a set of top K* documents that are relevant to the users' query with two possible outcomes:
 - a. Retriever successfully retrieves a set of documents for re-ranking
 - b. No documents were retrieved by the retriever, terminating the workflow
3. The retrieved set of documents will be processed and re-ranked to find top N* documents that are most similar and relevant to the user query
4. The top N documents would be passed into a prompt-engineered LLM for contextual compression & filtering, which extracts relevant sub-chunks from retrieved document chunks with two possible outcomes:
 - a. LLM identifies that the document chunk is not relevant to query, and returns a fixed input – defined with prompt engineering
 - b. LLM identifies and extracts the relevant sub-chunks from the document chunk
5. Responses from the prompt-engineered LLM are then utilized to filter out document chunks that have been deemed not relevant to the query, to yield top P* relevant sub-chunks.
6. LLM generates a natural language response using top P* relevant sub-chunks, which is then returned to the user; document pages from top P* relevant sub-chunks are also retrieved and returned to the user as an image for reference. There are again two possible outcomes:
 - a. LLM identifies that there are no relevant sub-chunks (i.e.: top P = 0) and terminates the workflow
 - b. LLM references the knowledge provided by the relevant sub-chunks to synthesize a response for the users' query

* - where $K > N > P$

4 System Implementation

4.1 Sparse Retrieval – BM25+

BM25+ (Best Matching 25+) is a ranking function used in information retrieval to estimate the relevance of documents to a given search query. [5] As a successor to term frequency – inverse document frequency (TF-IDF), it represents documents as sparse vectors, using the bag-of-words retrieval function to rank documents based on the query terms appearing in each document. However, BM25+ is more complex as it incorporates additional parameters to factor the effect of document length and term saturation in its scoring function.

The formula of BM25+ is as follows:

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \left[\frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} + \delta \right]$$

where:

- $f(q_i, D)$ represents the number of times that the keyword q_i occurs in document D ,
- $|D|$ represents the number of words in document D (document length)
- $avgdl$ represents the average document length within the corpus which documents were retrieved from
- k_1 , b and δ are free parameters, and
- $IDF(q_i)$ represents the inverse document frequency weight of the query term q_i

The formula for $IDF(q_i)$ is as follows:

$$IDF(q_i) = \ln \left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1 \right)$$

where:

- N represents the total number of documents in the corpus
- $n(q_i)$ represents the number of documents containing q_i

In ParliaGPT, sparse retrieval is performed by first applying NLP on the user's query to identify the lemma tokens, the query lemmas are then compared with document lemmas in the relational database to obtain a score, which represents the proximity of the query to the document. Higher scores indicate a higher degree of similarity between the document and query, vice versa.

4.2 Dense Passage Retrieval

Dense Passage Retrieval (DPR) is another technique used in information retrieval to efficiently retrieve relevant documents from a text corpus. Unlike sparse retrieval which relies on sparse representations of text, DPR compares the dense vector representations of the document and the search query using a similarity measure (i.e.: Euclidean Distance, Cosine Similarity, Manhattan Distance, etc). As such, even if two sentences do not share keywords, they can be semantically similar if their dense vector representations are closely related. [6]

In ParliaGPT, DPR is carried out by retrieving the ChromaDB collection and passing both the query and document embedding model (i.e.: GTE model) as arguments into its query function. For measurement of similarity between the search query and each document chunk, cosine similarity was pre-selected as the similarity measure during the creation of the ChromaDB collection.

4.3 Document Reranking

Document reranking is an information retrieval technique used for refining the ranking of document chunks retrieved by an initial retrieval pipeline. It involves reevaluating and reordering of the retrieved document chunks to enhance the relevance of top-most similar document chunks.

In ParliaGPT, document reranking is performed with cross encoder models (i.e.: BGE Reranker). Cross-encoders are a type of neural network architecture that is primarily used for text / sentence pair classification; it accepts a pair of inputs and generates a single output that indicates relevance between the input pair.

Input pairs are created by pairing the search query with each retrieved document chunk, before it is encoded by the cross-encoder model to generate a single representation for reranking the retrieved document chunks. Although the retrieval performance of cross-encoders is higher than text embedding (bi-encoder) models, cross-encoders are not used as an initial retrieval pipeline due to its lower retrieval speed.

To accelerate retrieval, a retrieve and rerank pipeline was implemented by first performing DPR (with GTE model) and BM25+ to quickly obtain an initial set of similar document chunks, before applying the more accurate, but slower BGE Reranker model to rerank the retrieved document chunks.

4.4 Prompt Engineering – Contextual Filtering & Compression

As previously mentioned in section 3.2.3, Contextual Filtering and Compression is a technique used to reduce data size while preserving the essential contextual information. In ParliaGPT, this is carried out by applying prompt engineering to guide an LLM to perform the task. As the outputs from the prompt engineered LLM are required for response generation, three-shot prompting was applied for the model to learn the desired response outputs:

- If the document chunk has contents that are relevant to the search query, extract the contents. Otherwise,
- If the document chunk is completely irrelevant to the search query, return a negative flag value

An excerpt of the prompt is as shown in the Figure below.

```
#####  
question = "What are the benefits of eating fruits?"  
  
context = "Apples contain key nutrients, including fiber and antioxidants.  
They may offer health benefits, including lowering blood sugar levels and benefitting heart health.  
Apples are among the world's most popular fruits.  
They grow on the apple tree (Malus domestica), originally from Central Asia.  
Apples are high in fiber, vitamin C, and various antioxidants. They are also very filling, considering their low calorie count.  
Studies show that eating apples can have multiple benefits for your health.  
Usually eaten raw, apples can also be used in various recipes, juices, and drinks. Various types abound, with a variety of colors and sizes."  
  
answer = "Apples contain key nutrients, including fiber and antioxidants.  
They may offer health benefits, including lowering blood sugar levels and benefitting heart health.  
Studies show that eating apples can have multiple benefits for your health."  
#####
```

Figure 7: Prompt Engineering Excerpt for Contextual Compression & Filtering

5 Conclusion

5.1 Challenges

The following are some of the challenges that were encountered during the development of ParliaGPT:

- **Complex query inputs:** Complex queries are questions that contain multiple other questions within them. Due to the complexity, LLMs are unable to solve them directly as multiple intermediary steps are required prior to reaching resolution.
- **Slow response times:** Due to the requirement to make multiple LLM calls in the Contextual Compression & Filtering pipeline, significant time may be required to process a query (subjected to response generation length) which may lead to a poor user experience.

5.2 Future Works

As a result, from the development of ParliaGPT, these are some reflections on the areas that can be improved to further enhance the prototype:

- **Sub-Query Engine:** A sub-query engine is designed to split up a complex query into multiple simpler queries for processing, before synthesizing the responses from the simple queries together to answer the complex query posed by the user.
- **Voting Mechanism:** A voting mechanism would allow users to feedback on the quality of responses generated by the LLM, this allows data scientists to deep dive into the response model's shortcoming and possibly retrain it for better performance in the future.
- **Response Caching:** A response cache allows the prototype to store past queries made by users. For commonly asked questions, the response cache helps to cut down on generation time, giving users faster responses to their queries.

6 References

- [1] S. K. S. T. Z. B. M. A. Scott Barnett, "Seven Failure Points When Engineering a Retrieval Augmented Generation System," 2024 .
- [2] R. M. Priyanka Mandikal, "Sparse Meets Dense: A Hybrid Approach to Enhance Scientific Document Retrieval," 2024.
- [3] G. R. M. F. M. C. A. R. N. P. C. A. G. Michael Glass, Re2G: Retrieve, Rerank, Generate, 2022.
- [4] A. B. S. H. Nicholas Harris, Enhancing Embedding Performance through Large Language Model-based Text Enrichment and Rewriting, 2024.
- [5] A. A. G. P. W. K. S. V. Arian Askari, Injecting the BM25 Score as Text Improves BERT-Based Re-rankers, 2023.
- [6] B. O. M. L. W. E. C.-t. Y. Vladimir Karpukhin, Dense Passage Retrieval for Open-Domain Question Answering, 2020.
- [7] Y. Z. L. Z. D. L. P. X. R. G. Guangwei Xu, "Hybrid Retrieval and Multi-stage Text Ranking Solution at TREC 2022 Deep Learning Track," 2022.

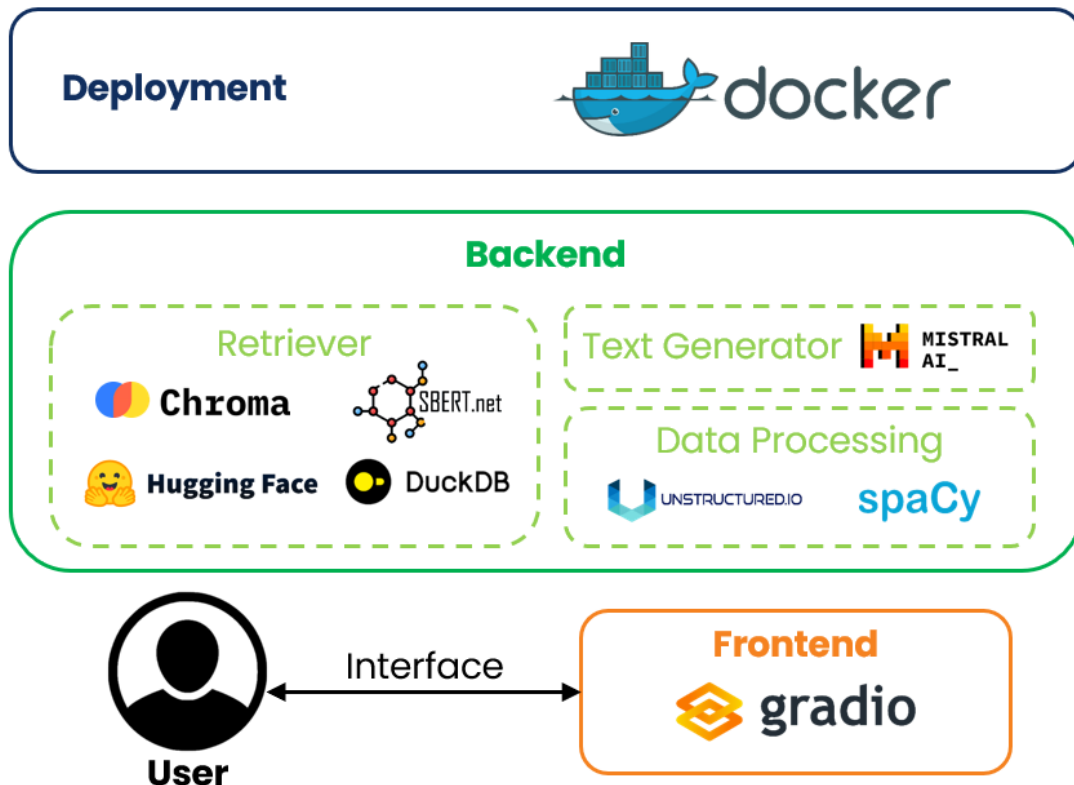
7 Appendix A: Project Proposal

Date of Proposal: 28 April 2024
Project Title: ParliGPT
Group ID (As Enrolled in Canvas Class Groups): Group Members (name, Student ID): YONG TING RUI E1339479
Sponsor/Client: <i>(Company Name, Address and Contact Name, Email, if any)</i> Institute of Systems Science (ISS) at 25 Heng Mui Keng Terrace, Singapore NATIONAL UNIVERSITY OF SINGAPORE (NUS)
Background/Aims/Objectives: <p>Parliamentary debates are a useful form of content that allows members of the public to keep up-to-date on politics and the country's developments. However, these debates often very long and information dense for consumption by members of the general public. Currently there are 2 ways for users to consume the information: 1) To watch the entire video or 2) To read the parliamentary debate minutes. Using either approach, the process of obtaining desired information from parliamentary debates is tedious and time-consuming</p> <p>Objective Statement: How can the process of finding information from parliamentary debate documents be streamlined, so that users can quickly obtain a response, and more information for future deep dive?</p> <p>Market Research For the proposed solution, we have identified the following competitors:</p> <ul style="list-style-type: none">• ChatGPT-<ul style="list-style-type: none">○ Strengths: Allow users to make natural language queries, and provide natural language responses back to its' users○ Weakness: Unable to provide information on events/data it has not been trained on• Search Engines (Google, Duckduckgo, Bing, etc)-<ul style="list-style-type: none">○ Strengths: Allow users to search and obtain extensive results from the public domain

- Weakness: Users have to deep dive into search results for more information, and information may come alternative sources (ie: news articles) instead from the root source (ie: parliamentary debate documents)

Project Descriptions:

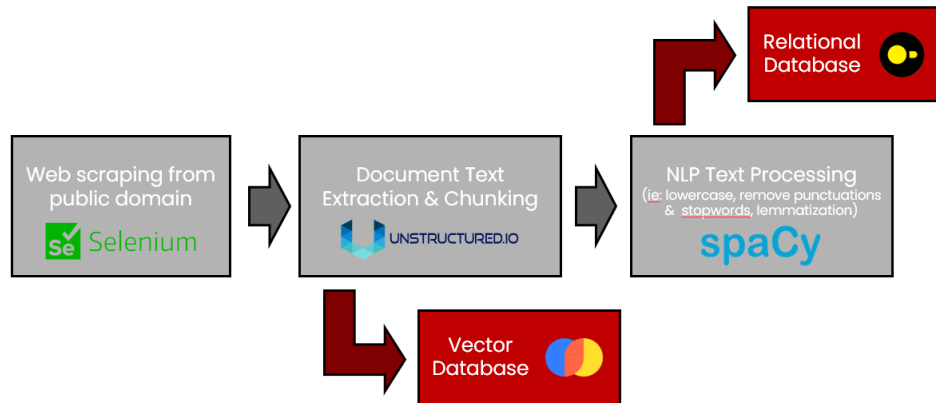
System Overview



The following system overview illustrates the components that will be used for building ParliaGPT. The frontend user interface will be powered by gradio for rapid prototyping. The build-up of RAG capability for ParliaGPT has been divided into 3 modules:

- The data processing module uses unstructuredio to ingest and extract text from PDF documents scraped from the web. Spacy will be utilized for its NLP capabilities to perform text-preprocessing.
- The retriever module will utilize a vector database (ChromaDB) and a relational database (DuckDB), open-source models will be sourced through SentenceTransformers and HuggingFace.
- The text generator module will utilize Mistral AI to generate natural language responses back to the user.

Data Collection

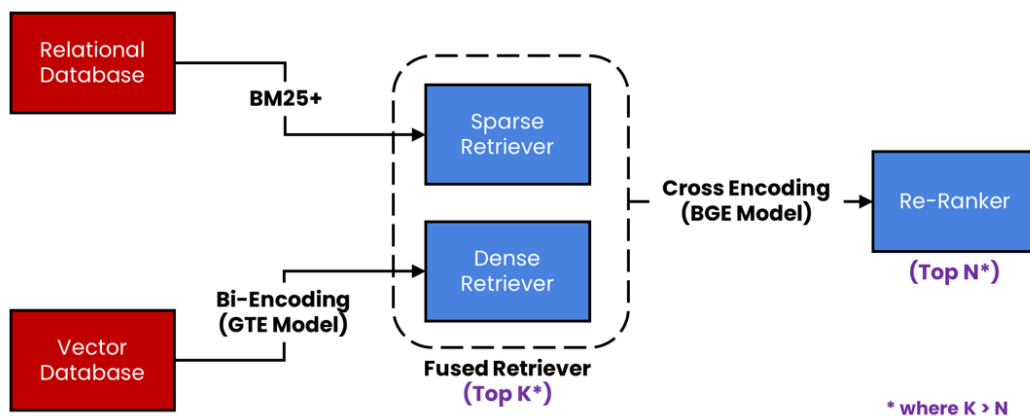


For gathering data from the public domain, one would be required to know when are the parliamentary debate dates. As the information is not easily obtainable as clean data, the approach is to use Selenium to automate data scraping across a user-defined timespan (eg: 4 months, 1 year, etc). Selenium will first be used to check if the target webpage is valid, if it is valid and comprises of the desired elements, the webpage will be saved into PDFs for subsequent processing.

The downloaded PDFs will be partitioned by unstructured, which extracts the PDF information in accordance to its' layout and chunks it into appropriate sizes for embedding into a vector database for dense retrieval.

For sparse retrieval which uses a keyword-based approach, spaCy is used to perform NLP on the extracted text chunks, converting it into chunk lemmas and storing it in a relational database, which will be used during sparse retrieval.

Fused Retriever + Reranker Pipeline



1. Fused Retriever

Fused retrievers are hybrid retrieval models that use the combination of both sparse vectors and dense vectors to perform document search. The strategy has benefits over the use of only either method as each retriever type has its own benefits:

- Sparse retrievers perform retrieval using keyword-based approach, it is useful for capturing domain based keywords or acronyms that an encoding model has not been trained or fine-tuned on.
- Dense retrievers perform retrieval by embedding the semantic features of text into vectors via an encoder model.

Combining the 2 methods together into a hybrid strategy allows us to leverage on the benefits of both methods. In addition, an algorithm will also have to be utilized

2. Reranker

Rerankers are cross-encoder models that accept both the user query and document as a text pair to generate a single similarity score as output. It is more accurate than retrievers as unlike bi-encoders which compresses the textual information into a pair of vectors for similarity comparison, rerankers receive the raw textual information of both the document and the query for computing similarity.

However, the higher accuracy of cross-encoder models comes with a performance trade-off (longer processing time), due to the need for on-the-fly computation of both query and document texts. To speed up computation, rerankers are only used for ranking documents after the fused retriever had retrieved an initial set of documents.

Contextual Compression & Filtering

Contextual compression & filtering is a two-step process which involves:

- Compressing the reranked documents into relevant text segments
- Filtering to identify reranked documents that are useful for generating a response to the query

The purpose of this approach recognizes that documents stored in the database may not be entirely useful for responding to queries. Instead, only snippets of the document are useful for generating answers. Therefore, the process uses Large Language Models (LLMs) to compress and filter documents.

This is achieved through the use of prompt engineering in LLMs, in which a set of instructions and examples that are passed into the LLM as a prompt, guiding the LLM in fulfilling the desired task.

Hardware Requirements

GPU with at least 8GB memory

8 vCPU, 16GB RAM

8 Appendix B: Techniques and Skills

Module Course	Knowledge and Skills
Machine Reasoning	Natural Language Processing of Text Approximate String Matching Relational Database (DuckDB) Vector Database (ChromaDB)
Reasoning Systems	Hybrid Reasoning System Similarity-Based Reasoning
Cognitive Systems	Generative Language Models Content Retrieval & QA with LLM

9 Appendix C: Installation and User Guide

As there are 3 separate parts to the application, the installation guide will be broken up into 3 the following 3 subparts:

- Web Scraper (Optional)
- Data Extraction (Optional)
- Prototype Application

9.1 Prerequisites

- Docker, if running application as a container
- Internet access for downloading of open-source models
- Linux (Ubuntu) is HIGHLY recommended if setting up data extraction program

9.2 Web Scraper Setup (Optional)

The web scraper setup guide serves to create a python virtual environment that is dedicated to the scraping the PARL website for parliamentary debate minutes documents.

Note: The web scraper program requires google chrome as a prerequisite.

If google chrome has not been setup, follow these steps:

Windows

1. Download google chrome from https://www.google.com/intl/en_sg/chrome/
2. Launch the installer and complete the installation process

Linux

1. Download the google chrome .deb file
 - `wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb`
2. Install the downloaded Google Chrome .deb file
 - `sudo dpkg -i google-chrome-stable_current_amd64.deb`

Otherwise, continue from here:

1. Create a Python 3.10 virtual environment using either venv or conda with the following commands:

Venv

- `python3 -m venv <myenvname>`

Conda

- `conda create --name <myenvname> python=3.10`

Note: Conda is recommended as compared to venv

2. Activate virtual environment and run the requirements.txt file in **webmine** folder

Venv

- `venv <myenvname_directory>/bin/activate`
- `pip install -r requirements.txt`

Conda

- `conda activate <myenvname >`
- `pip install -r requirements.txt`

3. Open the file webmine.py in **SystemCode** folder and edit the *START_DATE* and *END_DATE* parameters, which dictates the time period to check for parliamentary debate documents

```
#####  
# To edit in the format of YEAR, MONTH, DAY #  
#####  
START_DATE = date(2023, 2, 21)  
END_DATE = date(2024, 4, 19)  
#####
```

4. Once complete, save file and run the webmine.py file in **SystemCode** folder. Ensure that the virtual environment has been activated.
 - `python <folder_directory>/webmine.py`

9.3 Data Extraction Setup (Optional)

The data extraction setup guide serves to create a python virtual environment that is dedicated to the extraction of text and images from the scraped PDF files from PARL website.

Note: The data extraction program is highly recommended to be setup on Linux, due to compatibility issues with detectron2 package on Windows.

1. On Linux (Ubuntu), install the following linux dependency packages:
 - `sudo apt-get install -y libmagic-dev`
 - `sudo apt-get install libpoppler-dev`
 - `sudo apt install tesseract-ocr`
 - `sudo apt install libtesseract-dev`
2. Create a Python 3.10 virtual environment using either venv or conda with the following commands:
Venv
 - `python3 -m venv <myenvname>`**Conda**
 - `conda create --name <myenvname> python=3.10`**Note: Conda is recommended as compared to venv**
3. Activate virtual environment and run the requirements.txt file in ***process_data*** folder
Venv
 - `venv <myenvname_directory>/bin/activate`
 - `pip install -r requirements.txt`**Conda**
 - `conda activate <myenvname>`
 - `pip install -r requirements.txt`
4. Once complete, run the process_data.py file in ***SystemCode*** folder. Ensure that the virtual environment has been activated.

9.4 Application Setup

The application setup guide serves to create a python virtual environment for serving the ParliaGPT prototype application. There are 2 ways for serving the application:

- Running it locally using localhost
- Running it as a container via Docker

Regardless of the setup, the deployment steps for both methods are relatively similar via the following:

1. Download the chroma.sqlite3 file from the following [link](#) and add it into the **vdb** folder with the following directory: **SystemCode/data/vdb**
2. Create a Python 3.10 virtual environment using either venv or conda with the following commands:

Venv

- python3 -m venv <myenvname>

Conda

- conda create --name <myenvname> python=3.10

Note: Conda is recommended as compared to venv

3. Activate virtual environment and run the requirements.txt file in **app** folder

Venv

- venv <myenvname_directory>/bin/activate
- pip install -r requirements.txt

Conda

- conda activate <myenvname >
- pip install -r requirements.txt

4. Run **download_models.py** in **SystemCode** folder

- python download_models.py

5. Execute the following in accordance with your desired deployment type

Local

- python app.py
- Access via provided weblink in terminal

Docker

- docker build . -t parliagpt:latest
- docker run -dp 7868:7868 --gpus all --name parliagpt_app parliagpt
- Access via http://localhost:7860

10 Appendix D: Individual Reports

Name: Yong Ting Rui	Matriculation Number: E1339479
Personal Contribution: <ul style="list-style-type: none">• Development of Selenium script to automate data extraction from the web• Implemented data extraction pipeline for extracting texts and page images from PDF documents• Designed RAG pipeline for creating knowledge from isolated data sources• Implemented NLP pipeline for sparse retrieval• Implemented Hybrid Retrieval algorithm and pipeline• Implemented Retrieve + Rerank pipeline• Implemented Contextual Filtering & Compression pipeline with LLM• Development of Gradio Application for Prototype	
What learnt is most useful for you: <ul style="list-style-type: none">• Combining retrieval techniques to yield a blended / more performant results that takes into account both similar keywords and semantic similarity of texts• Applying prompt engineering to guide LLMs in performing a desired task• Leveraging on LLM to perform tasks beyond generating output (ie: Extracting snippets of important texts from long passages)• Using Selenium for webscraping purposes• Development of simple UIs for prototypes	
Application of knowledge and skills in workplace: <p>From this mini-project, I have learnt about the plethora of RAG techniques and strategies that are currently available to perform tasks of different complexities in real world situations. Although LLMs are extremely large and difficult to train with enthusiast hardware, the availability of model repositories meant that it is easy to download and utilize open-source models that had been created, trained or even fine-tuned by the community for development use.</p> <p>The mini-project has also taught me on how to utilize Generative AI beyond generating text content by asking questions, as these models possess NLP capabilities to perform functions (ie: summarization and extraction) that would previously require humans to perform, and the potential to incorporate Generative AI into processing pipelines using human-in-the-loop as a guard rail. It also taught me on how Generative AI can be used in domain-specific contexts, allowing my workplace and other organisations to leverage on Generative AI.</p>	