



# Researchers' AI Chatbot

Team: Yingrong Zhang, Warit Boonmasiri, Aiden Blishen Cuneo,  
Mohammad Alhajjeleh, Jada Gamis

## Contents

1. Abstract.....	2
2. Introduction.....	2
3. Background .....	3
4. Methodology.....	3
4.1 Language filtering – Langdetect.....	3
4.2 Reduce Dimensionality – PCA .....	4
4.3 Semantic Clustering – K-means .....	4
4.4 t-SNE Visualisation.....	4
4.5 Data Representation via Embeddings – Sentence Transformers .....	4
4.6 LLM Model – Google/Gemma-7b-it.....	4
4.7 Retrieval-Augmented Generation (RAG) .....	5
4.8 FAISS - Facebook AI Similarity Search.....	5
4.9 Chatbot Application UI – Streamlit.....	5
5. Experimental Results.....	6
6. Dataset.....	6
7. Workflow of the code and the experimental results.....	8
7.1 Workflow of the code from eda.ipynb .....	8
7.2 Workflow of the code from app.py .....	12
8. Conclusion .....	17
9. Group Member.....	18
10. AI Declaration .....	18
11. Reference .....	18

## 1. Abstract

Unprecedented levels of information are being made available in today's digital landscape. While it democratises access to valuable knowledge, its overwhelming amount exceeds our capacity to process and apply it efficiently. This issue proves particularly relevant in academic research, where open-source archives amass hundreds of publications daily. Our project addresses the problem of the inability to keep up with an influx of information across various fields. It seeks to use natural language processing (NLP) techniques introduced in this course to visualise and organise scholarly literature from the computer science field. In particular, the implementation of this solution uses explicit automated clustering, topic modelling, and transformers, which are applied to a corpus of papers sourced from the open-source archive platform arXiv.

## 2. Introduction

The foundation of our project was referenced from Kaggle, and applies k-means clustering to a corpus of papers, grouping them based on semantic similarity. This unsupervised approach will allow us to identify coherent topic groups, specifically through vectorized representations using Term Frequency Inverse Document Frequency (TF-IDF) and word embedding.

Following this process, the group will then feed the clustered data into a transformer-based model, allowing it to perform tasks such as classification, topic extraction and summarisation. Clustering the papers beforehand as a pre-processing step for transformer-based tasks improves performance and contextual coherence. The integration of transformer architectures and standard k-means clustering ultimately serves as a scalable method for managing and interpreting these extensive document collections. This includes, but is not limited to tasks such as identifying groups of papers written on the same topics, sifting through datasets with category or keywords, and identifying thematic trends across disciplines. Furthermore, our project also includes a visualisation of the corpus through a 2D scatter plot- on which papers with similar topics will be grouped under a single label, and plotted within close proximity to each other. In the case that users want to further narrow their scope, it includes a search functionality that specifically outputs papers containing the search term, and can display basic information such as author, journal, and abstract upon hovering over the plot points.

Overall, our solution satisfies the stated problem through simplifying the search for related publications and providing an in-depth map to the landscape of certain research topics. The conversion from an unordered repository of papers to a navigable map makes this knowledge to be both engaging and accessible, allowing users to accelerate their research, and conduct informed decision-making.

### 3. Background

Though the group had a clear technical direction, the project presented several significant challenges, of which time constraints were one of the most pressing issues. However, the largest hurdle in the progress of our assignment was the decision to either implement a question-and-answer generation using an LLM or to take a retrieval-augmented generation (RAG) approach.

Initially, the group explored the idea of fine-tuning a large language model (LLM) to generate question-answer pairs for each document, to provide users with a conversational interface. However, there were several limitations to this approach. Firstly, due to hallucinations and formatting inconsistencies, the group was unable to generate usable question-answer pairs. While we tried to address this using few-shot prompting, the models would constantly copy the template examples instead of generating accurate question-answer pairings of the same essence. Furthermore, our chosen dataset was not structured in a compatible way for question-answer tasks, which on a general basis created an incongruence with the report aim and data. These issues, paired with a strict deadline prompted us to ultimately choose a RAG-based solution.

Our decision to run transformer-based models, exceptionally those capable of understanding and generating text at a large scale, placed a heavy load on our systems. The group encountered slow runtimes and inconsistent results, which made testing and deployment more time-consuming and complex. These constraints required us to make careful trade-offs and prioritise developing a purely functional solution rather than implementing more advanced features.

### 4. Methodology

#### 4.1 Language filtering – Langdetect

The articles were written in a range of different languages, which had the potential to disrupt our model's workflow. To combat this, we opted to apply Langdetect for each summary to filter out non-English documents and eliminate documents written in the wrong format or with a corrupted summary. After implementing this, there was a higher accuracy rate during the embedding and clustering stages. These techniques were employed to combat the common AI issue of inaccurate data preprocessing, which we ensured was successful prior to conducting any NLP modelling tasks.

## 4.2 Reduce Dimensionality – PCA

Due to the high dimensionality of our dataset vectors, we applied principal component analysis (PCA) to reduce them. This operation helps to remove noise from the data and speeds up the clustering process. PCA transforms the original set of variables into a smaller set of new, uncorrelated variables called principal components, while keeping most of the information from the original dataset. This allows us to preserve the essential structure of the data while reducing the number of dimensions.

## 4.3 Semantic Clustering – K-means

We decided to use K-means clustering to group summaries into semantically similar clusters for NLP. Each cluster will act as one knowledge base. When we use RAG, instead of working with the whole dataset, we focus on a particular cluster with more specific information about a certain area. Using only one cluster like this ensures that the model only uses necessary information, while also increasing computation efficiency and reducing the time and memory taken for Q&A (question-and-answer).

## 4.4 t-SNE Visualisation

To evaluate the clustering, we applied t-distributed Stochastic Neighbour Embedding (t-SNE) to reduce the PCA output into 2D for plotting. After doing so, we can visualise the data presented in 2 dimensions. One key point is that we still have to use PCA to reduce the overall dimensions, remove the noise first, and then feed the result data from PCA into t-SNE for processing and visualisation.

## 4.5 Data Representation via Embeddings – Sentence Transformers

We used SentenceTransformer (all-MiniLM-L6-v2) to convert each paper's summary into a 384-dimensional sense vector embedding. This tool captures the semantic meaning of the text, which means that similar documents will have similar vectors, and in addition, the distance between documents is linked to the difference in their meaning.

<https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

As we can see from Hugging Face, all-MiniLM-L8-V2 is a sentence-transformers model: It maps sentences and paragraphs to a 384-dimensional dense vector space and can be used for clustering or semantic search.

## 4.6 LLM Model – Google/Gemma-7b-it

<https://huggingface.co/google/gemma-7b-it>

For our LLM Model, we picked Google/Gemma-7b-it. Gemma is a lightweight, state-of-the-art open model from Google. It is used to create the Gemini models, which are available in English, with pre-trained variants, open weight, and instruction-tuned variants.

Gemma models are suitable for various text generation tasks, including answering questions, summarising, and reasoning. Google/Gemma-7b-it is relatively small, which makes it feasible for our use case, which is to deploy it on our personal computer.

(Note: It still needs a computer with a powerful GPU. For our group, when we deployed it on a regular computer, we could not run the model. We could only run the model on one of our groupmates' high-performance computers with an NVIDIA GPU containing 24GB of VRAM. However, our model should be able to run on a device with a GPU that has at least 20GB of VRAM.)

## 4.7 Retrieval-Augmented Generation (RAG)

We used RAG techniques and combined dense retrieval with semantic retrieval and LLM-based generation. Instead of purely relying on the prompt itself, RAG first retrieves the relevant cluster context (top-k summaries from our chosen cluster) and then feeds this context along with the user query to our language model Google/Gemma-7b-it. This added context provides the model with specific, relevant and up to date information that a regular LLM would not normally have access to. This allows the model to answer questions with much higher precision.

## 4.8 FAISS - Facebook AI Similarity Search

<https://ai.meta.com/tools/faiss/>

Our project needed to quickly retrieve the top K most semantically similar summaries, which are based on a user query vector, so we decided to make use of the FAISS library. It is a high-performance library used for efficient similarity search in high-dimensional vector spaces. It can enable real-time document retrieval for RAG.

## 4.9 Chatbot Application UI – Streamlit

After looking for a suitable UI framework, the group ultimately chose to use Streamlit as it was easy to employ, with Python being a familiar language amongst our team members. It also possesses a cloud version, so we can not only run it on our computer, but also make it a cloud version for further distribution. The group primarily chose Streamlit due to its capability of having an interactive UI for LLM workflows, in combination with Python's ability to make use of data science tools such as Pandas, FAISS and Hugging Face models, which were needed for our project. So, after considering the options, we implemented Streamlit as our UI tool for the AI chatbot, as it was the obvious choice.

## 5. Experimental Results

The code is mainly within these two notebooks

1. eda


---

 `eda.ipynb`

---

2. app including RAG & UI

---

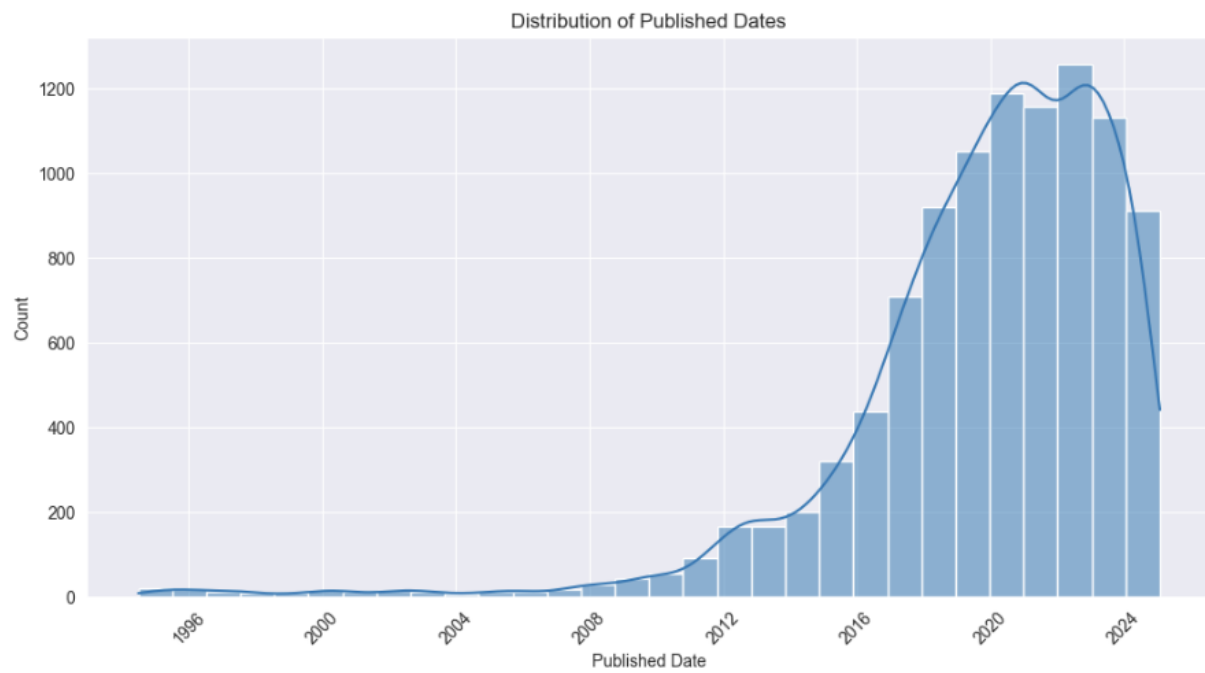
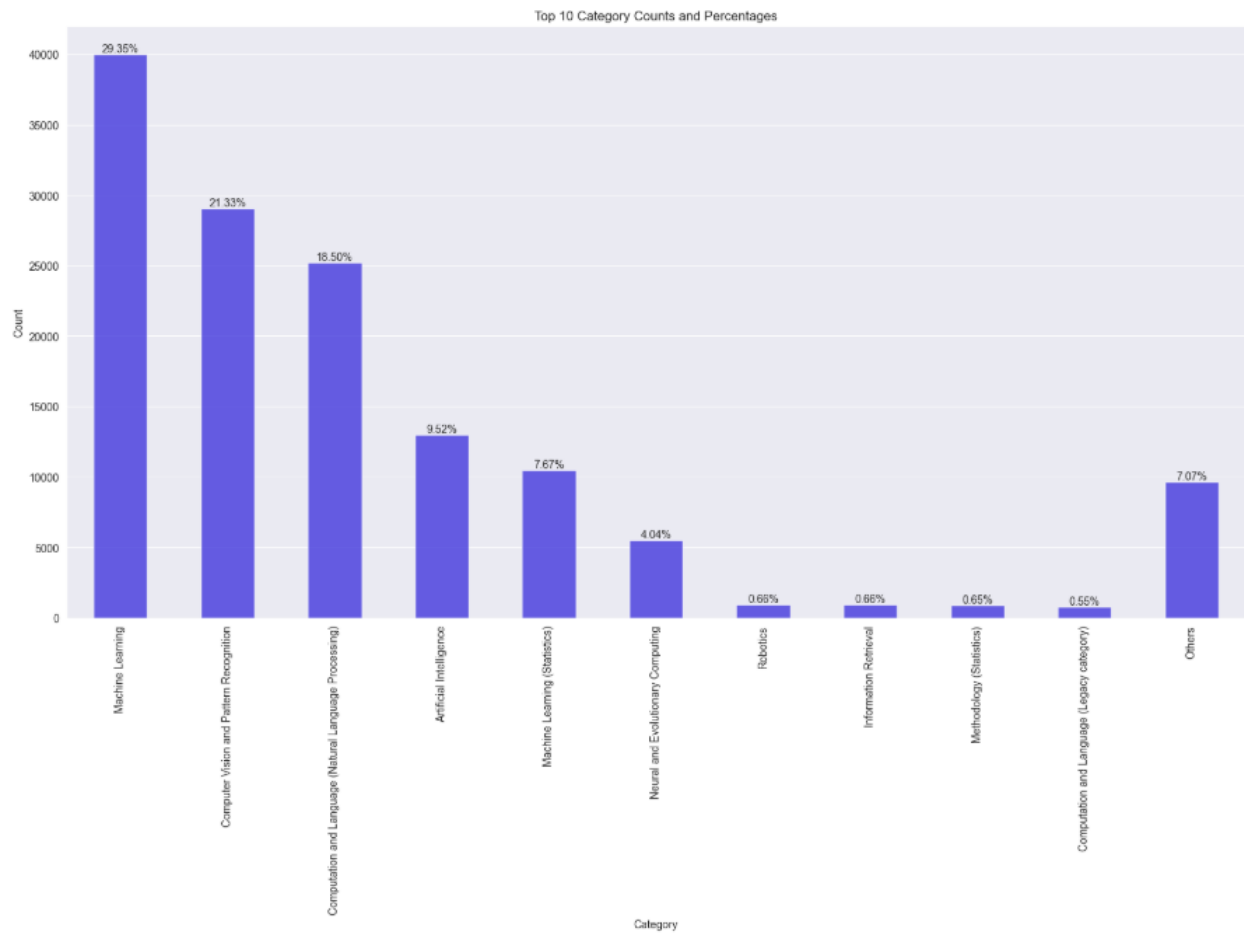
 `app.py`

---

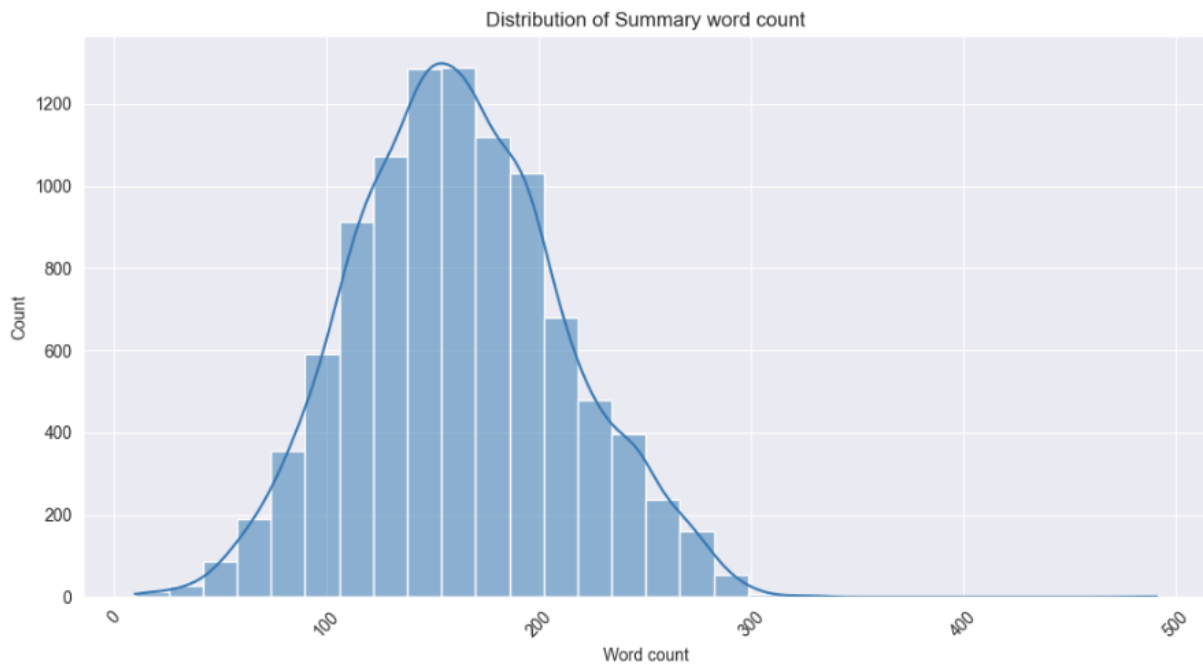
## 6. Dataset

We chose the arXiv Scientific Research Papers Dataset, which we collected from Kaggle. This dataset contains over 100,000 articles on various topics, such as Artificial Intelligence, Machine Learning, Computer Science, and more.

- Data Source: [Kaggle – arXiv Scientific Papers](#)
- Fields Used: Title, summary, abstract\_summary, categories, published\_date
- Size: Filtered subset of ~ 10,000 summaries
- Data distribution of category, published date, and word count as below.







## 7. Workflow of the code and the experimental results

We divided our workflow into two phases:

- (A) Preprocessing & Clustering – eda.ipynb
- (B) Retrieval-Augmented Generation & Chatbot UI – app.py

### 7.1 Workflow of the code from eda.ipynb

#### Preprocessing & Clustering – eda.ipynb

##### Step 1: Language Detecting (Filtering)

We use Langdetect to remove non-English or incorrectly formatted text data.

- We only detect the first 50 words.
- We also detected those poorly formatted texts.

As a result, we figured out that there are only a number of English papers, 10000.

```
: print(f'In summary, there are {languages_dict}')
```

```
In summary, there are {'en': 10000}
```

## Step 2: NLP clean up (Optional)

We applied NLP processing techniques using the Spacy Library to improve the quality of our text inputs for embedding. We included the following methods:

1. **Removing stopwords**

We removed very commonly used words such as “*the*”, “*a*”, and “*with*”, using Spacy’s built-in STOP\_WORDS list.

2. **Lemmatisation**

Reducing the words to their base forms, such as *running*-> *run*, ensures a better semantic representation by using *lemma\_*.

3. **Filtering non-informative or noisy tokens**

We also added preprocessing using regex or Spacy to further clean the summaries. We removed special characters, non-alphabetical tokens, and rare symbols.

## Step 3: PCA-Dimensionality Reduction

We reduced the number of dimensions from 4096 to 2830 using PCA, but this was still too high.

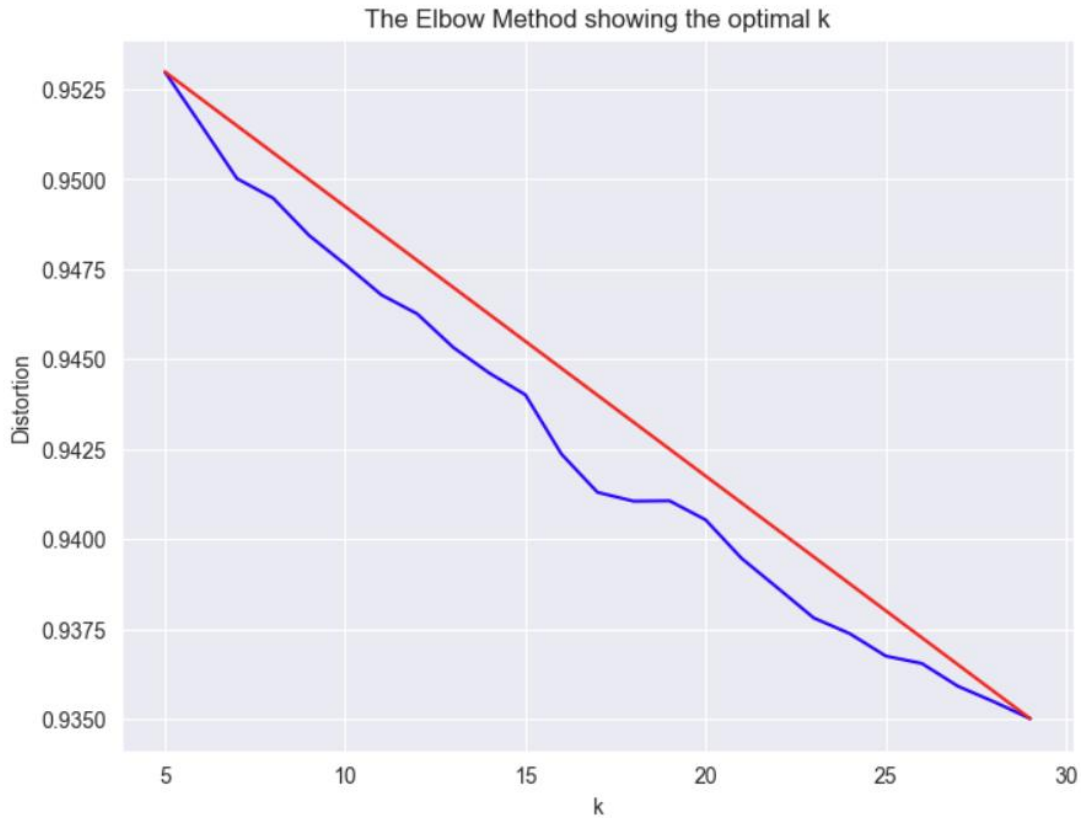
```
print(f'Reduced dimensions from {X.shape} to {X_reduced.shape}')
```

```
Reduced dimensions from (10000, 4096) to (10000, 2830)
```

## Step 4: K-means Clustering

We applied k-means clustering to group similar research paper summaries into 20 semantic clusters. The plot below, which shows the Elbow Method showing the optimal k, shows that we chose k=20. The `.fit_predict()` method assigns a cluster ID to each summary, which is stored in a new column, `cluster_id`.

- Model we used: KMEANS
- Distance calculation: Euclidean

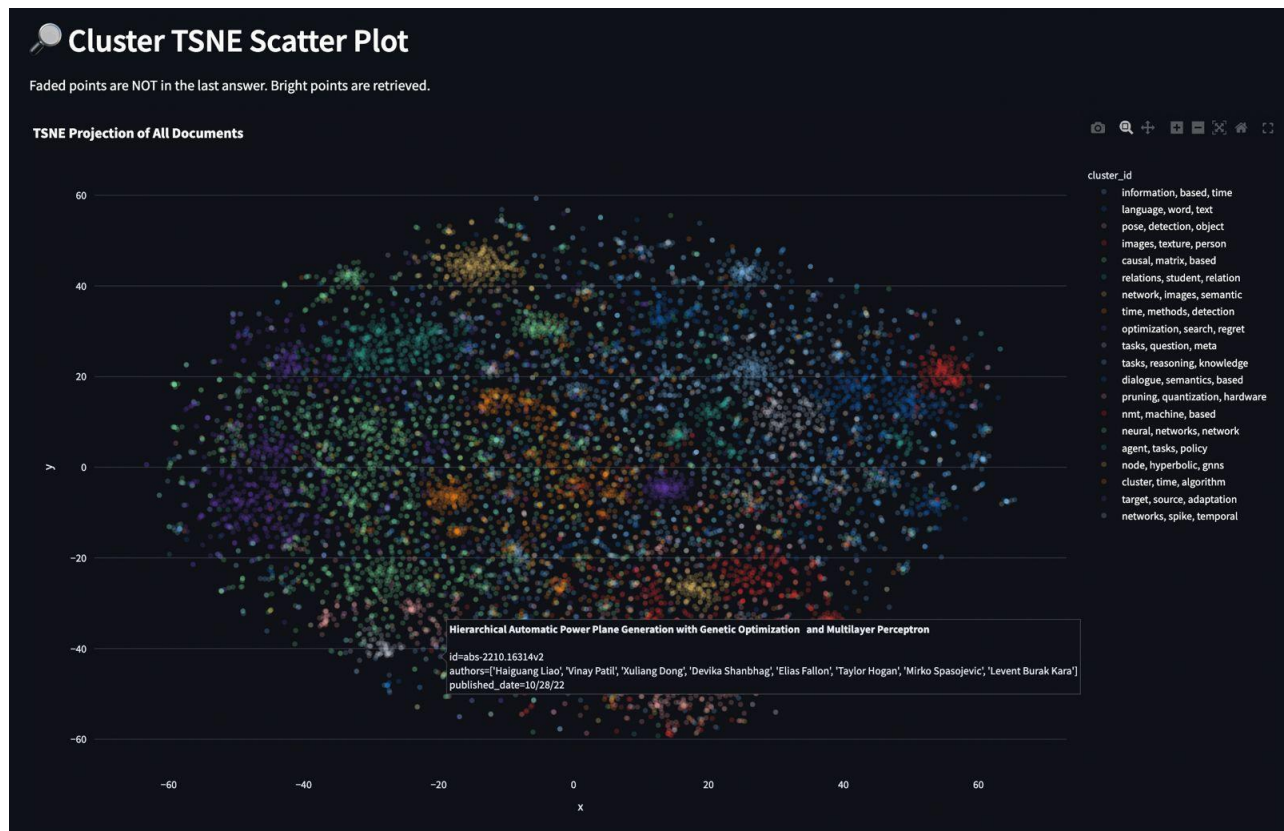


### Step 5: Dimension Reduction into 2D – t-SNE

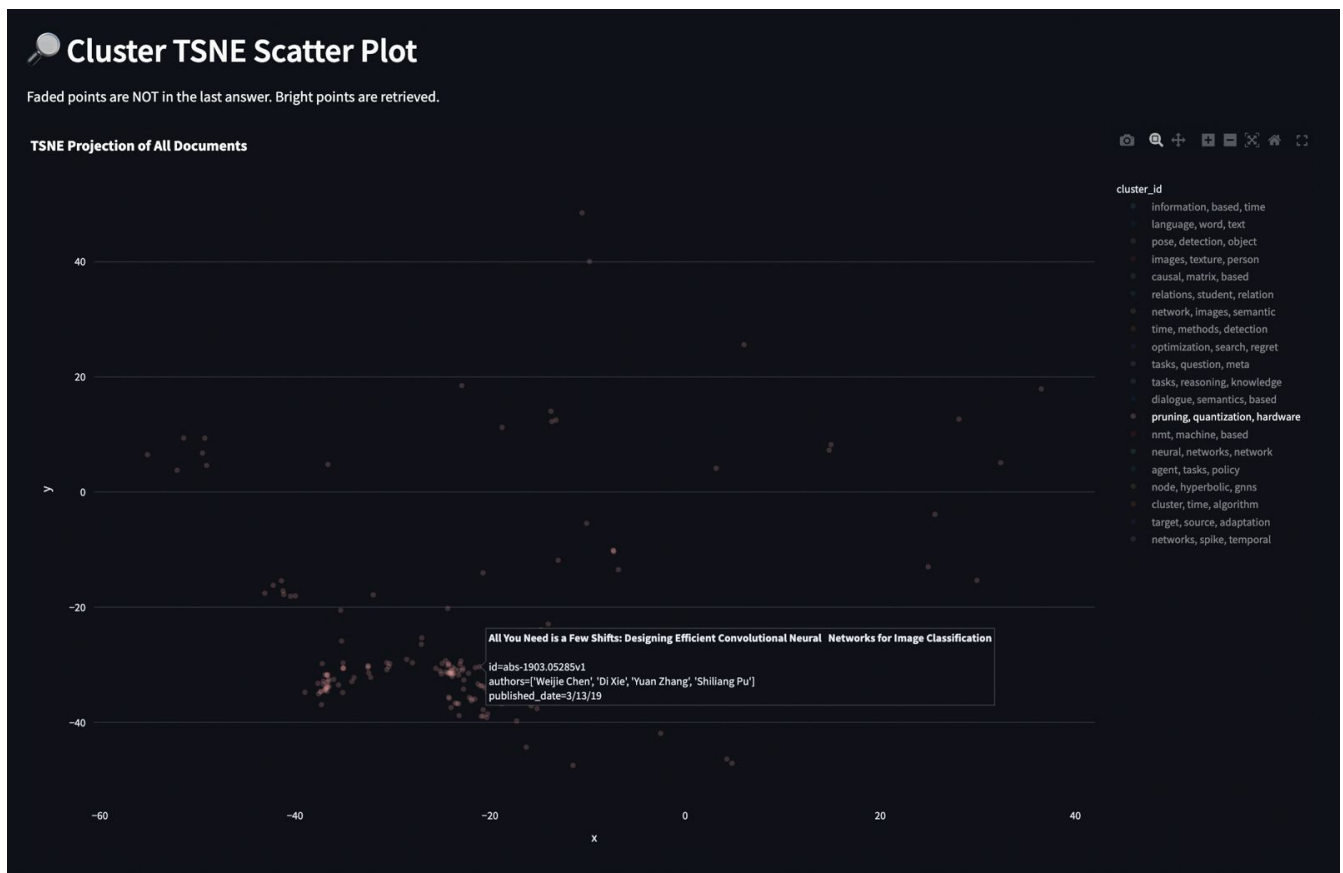
To visualise the semantic structure of the clustered summaries, we used t-distributed Stochastic Neighbourhood Embedding (t-SNE) from the sklearn *manifold* library to reduce the high-dimensional data into 2 dimensions. As humans, we can only interpret data in low dimensions, such as 2D or 3D. For this reason, t-SNE helps us visualise our results clearly.

### Step 6: Visualisation

Our team designed an interactive UI for data visualisation. When hovering your mouse over a particular dot, it will give you the main information about this paper, as each dot represents one individual paper. Each colour represents one cluster, so you can see clearly how the papers are distributed among the clusters in our dataset.



Furthermore, if you click on a specific cluster on the right-hand side of the UI, it will automatically filter the related papers. Using this feature, our Interactive UI users can focus on whichever cluster they like, and even zoom in on it or find out more information about it.



## 7.2 Workflow of the code from app.py

### Retrieval-Augmented Generation & UI

In this part of the code's workflow, we build our chatbot system, combining RAG techniques with an interactive Streamlit user interface.

#### Step 1: Load Required Models (Embedding & LLM)

- **SentenceTransformer("all-MiniLM-L6-v")**  
 Firstly, we load *the SentenceTransformer model* for later convert the text into dense semantic vectors.
- **google/gemma-7b-it**  
 Then, Gemma-7 b, a large language model from Hugging Face, is loaded for response generation.
- **@st.cache\_resource**

We wrapped it up with `@st.cache_resource` to avoid it being on every app run.

## Step 2: Interactive Cluster Selection (Streamlit UI)

This step allowed users to select a precomputed cluster from a dropdown.

Then, the scoped knowledge base for semantic searching is defined.

- **Precomputed K-means cluster**

We clustered our dataset and saved the results into a CSV file. Each row has a *cluster\_id* assigned from K-means.

- **st.selectbox UI**

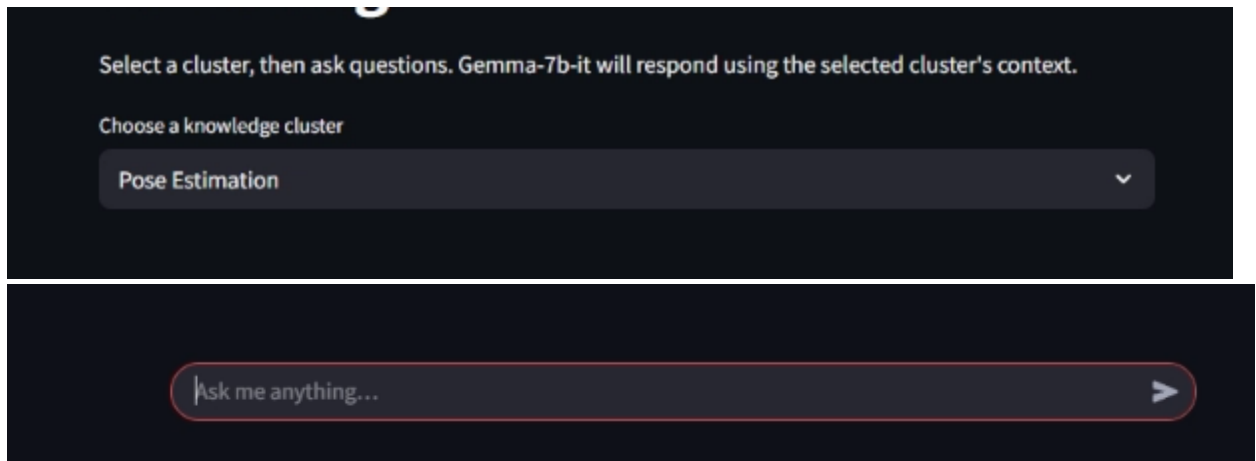
The *selectbox* is a Streamlit dropdown UI element where the user selects which cluster to check.



## Step3: Load Cluster Data & Create FAISS Index

- **Subdf (subset of DataFrame)**

It represents the paper in the selected cluster, where we build a smaller knowledge base within the cluster we need. Filters the dataset to include only entries from the selected clusters. In this case, embedding will only be created for this cluster instead of the whole dataset.



- **Embedding Creation: SentenceTransformer**  
Summaries from subdf are encoded into vectors using SentenceTransformer
- **FAISS Index (faiss.IndexFlatIP):**  
Afterwards, a FAISS index is created only from this subset. In this case, we made a scoped semantic search base, which is more efficient and relevant in searching the data in the related cluster instead of the whole dataset. Then, embedding is added to a FAISS index for the similarity search for fast similarity.

#### Step 4: User Enters a Query (UI)

- **Text input box via st.chat or st.text\_input**
- `user_q := st.chat_input("Ask me anything...")`  
User asks a question.

#### Step 5: Semantic Retrieval – Core RAG Step

- **Query Embedding**  
`q_emb = retriever_model.encode([prompt], normalize_embeddings=True)`

The user's question is converted into a 384-dimensional embedding using SentenceTransformer.

- **FAISS Search (k=20)**  
`_, I = index.search(q_emb, k)`

It tells the FAISS to take this query vector and find the index's top 20 most similar vectors.

```
index = faiss.IndexFlatIP(dim)
index.add(embeddings)
```

The index contains only embeddings from one cluster. Users then use the inner product (IP = Inner Product) to rank similarity.

- **Fetch Search**

```
idxs = fetch_context(subdf, retriever_model, index, user_q)
```

Retrieved indices are used to match relevant text passages or summaries from subdf.

## RAG Chatbot with Fast Cluster Switching

Select a cluster, then ask questions. Gemma-7b-it will respond using the selected cluster's context.

Choose a knowledge cluster

Pose Estimation

can you tell me what you know about pose estimation from your knowledge base

**References:**

- 'Self-supervision on Unlabelled OR Data for Multi-person 2D/3D Human Pose Estimation', 7/16/20
- 'Bottom-up approaches for multi-person pose estimation and its applications: A brief review', 12/22/21
- 'When Regression Meets Manifold Learning for Object Recognition and Pose Estimation', 5/16/18
- 'SPGNet: Spatial Projection Guided 3D Human Pose Estimation in Low Dimensional Space', 6/4/22
- 'Detection of pose orientation across single and multiple axes in case of 3D face images', 9/18/13
- 'End-to-end learning of keypoint detection and matching for relative pose estimation', 4/2/21
- '3D Face Pose and Animation Tracking via Eigen-Decomposition based Bayesian Approach', 8/29/19
- 'L6DNet: Light 6 DoF Network for Robust and Precise Object Pose Estimation with Small Datasets', 2/3/20
- 'Fast and Robust Multi-Person 3D Pose Estimation from Multiple Views', 1/14/19
- 'Pano2CAD: Room Layout From A Single Panorama Image', 9/29/16
- 'A simple yet effective baseline for 3d human pose estimation', 5/8/17
- 'DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model', 5/10/16
- 'A Global to Local Double Embedding Method for Multi-person Pose Estimation', 2/15/21
- '3D Pose Regression using Convolutional Neural Networks', 8/18/17
- 'Invariant Representation Learning for Infant Pose Estimation with Small Data', 10/13/20
- 'A generalizable approach for multi-view 3D human pose regression', 4/27/18
- '3D Human Pose Estimation = 2D Pose Estimation + Matching', 12/20/16
- 'Back to Optimization: Diffusion-based Zero-Shot 3D Human Pose Estimation', 7/7/23
- 'Learning to Refine Human Pose Estimation', 4/21/18
- 'Segmentation-driven 6D Object Pose Estimation', 12/6/18

Ask me anything...

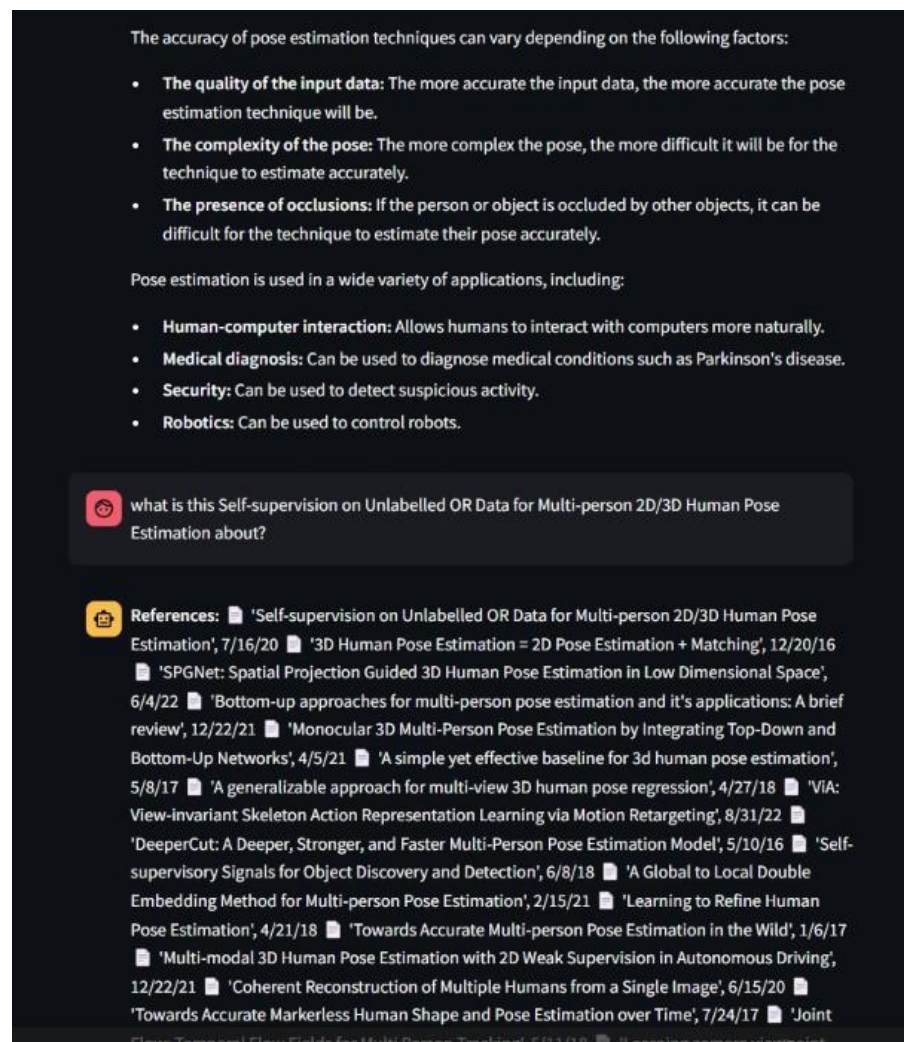


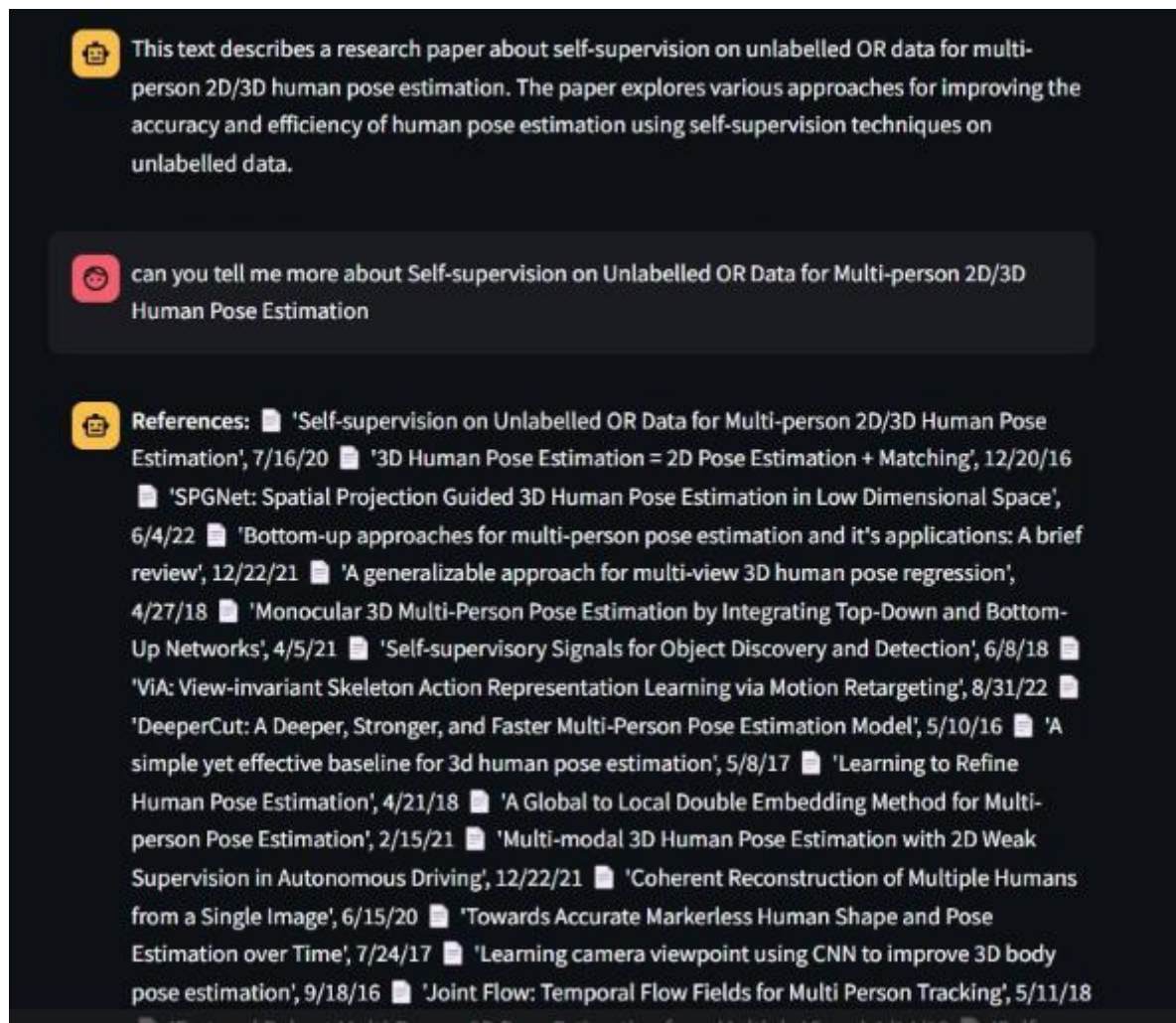
## Step 6: Tokenise the prompt and Generate Answer

- **build\_prompt(...)**  
Create a custom prompt using retrieved context(which is the top-k relevant context passages retrived from FAISS), then combine with the user's current query, and format a single prompt that is then sent to the LLM model(GEMMA-7b-it).
- **tokenized = tokenizer(...)**  
In this part, the prompt is tokenised.
- **out = llm\_model.generate(...)**  
Afterwards, it returns a structured answer.

## Step 7: Display chat in Steamlit

- `st.chat_message("assistant").markdown(formatted_answer)`





## 8. Conclusion

The group has successfully designed and implemented an interactive research exploration tool that combines RAG with techniques such as visualisation and semantic clustering. By taking advantage of pre-trained models such as SentenceTransformer for embeddings and Gemma-7B-IT for natural language generation, we built an AI-powered chatbot with an architecture that supports a variety of tasks such as efficient data filtering, scoped knowledge retrieval, and dynamic prompt construction. Overall, the group has developed a key demonstration of techniques introduced in the course- particularly vector embeddings, clustering, RAG-based retrieval, and language model inference.

## 9. Group Member

Members	Email
<b>Warit Boonmasiri</b>	warit.boonmasiri@student.uts.edu.au
<b>Mohammed Alhajjeh</b>	mohammad.alhajjeh@student.uts.edu.au
<b>Jada Gamis</b>	jada.i.gamis@student.uts.edu.au
<b>Yingrong Zhang</b>	Yingrong.zhang@student.uts.edu.au
<b>Aiden Blishen Cuneo</b>	aiden.r.blishencuneo@student.uts.edu.au

## 10. AI Declaration

We used generative AI to assist us in developing this project. After completing the first version of our RAG implementation, we gave ChatGPT the Python script to see if we could make improvements.

## 11. Reference

Google. (2024). Gemini 1.5 [Multimodal AI model]. <https://gemini.google>

OpenAI. (2025). ChatGPT (March 14 version) [Large language model]. <https://openai.com>