COMP5423 Group Project Instruction

Topic	Retrieval-Augmented Generation (RAG)
Group Size	Up to 4 students per Group
Due Date	**November 30, 2025 (HK time, 23:59)**
TA	Runyang You, Xin Zhang
Contact	`{runyang.you, xin404.zhang}@connect.polyu.hk`

Objective

This project aims to develop a Retrieval-Augmented Generation (RAG) system using a subset of the HotpotQA dataset, which contains multi-hop, Wikipedia-based questions along with their corresponding answers. The objective is to build an end-to-end question answering system capable of handling complex queries that require reasoning across multiple documents. By integrating a retrieval module with a generative large language model, the system will improve both the factual accuracy and explainability of answers through evidence-grounded response generation. The HotpotQA dataset will serve as the training and evaluation foundation, with a focus on understanding the application of RAG techniques to multi-hop, explainable question-answering tasks. The project will optimize both retrieval and generation components to enhance overall system performance for real-world RAG applications.

Project Resource

In this project, we will develop and evaluate an RAG system based on a sampled subset of HotpotQA. Table 1 shows the statistics of the subset, which is called **HQ-small**. The `Train` and `Validation` splits with relevant documents could be used in developing and self evaluation, while the `Test` split is for the final performance evaluation by TAs. You are required to submit your answer prediction and the indices of the retrieved supporting document for the `Test` split (refer to sec 4). **HQ-small** is released at `https://huggingface.co/datasets/izhx/COMP5423-25Fall-HQ-small` (The test split will be released 15 days before the submission deadline).

Split Name	Count	Data Format
Train	12,000	{“id”: str, “text”: str (query text), “answer”: str, “supporting_ids”: list[str]}
Validation	1500	{“id”: str, “text”: str (query text), “answer”: str, “supporting_ids”: list[str]}
Test	1052	{“id”: str, “text”: str (query text)}
Collection	144,718	{“id”: str, “text”: str (document)}

Table 1: Overview of the HotpotQA subset (HQ-small) for this project.

In Lab 2 (Oct 22) & 3 (Nov 12), we will talk about the implementation of the retrieval part and the whole RAG system. You can draw inspiration from the lecture content and the accompanying code, which are released in the subject github `https://github.com/polyunlp/COMP5423-25Fall`. The official evaluation scripts for retrieval and question answering are also provided in this repo.

Task Overview

The entire project is divided into four major components as summarized in Table 2.

Component	Weight (%)
System Implementation	40%
Written Report	20%
Demo Video	20%
System Performance	20%

Table 2: Overall project component weighting.

Detailed Requirements

1.0 System Implementation (40%)

The implementation should include a retrieval module, a generation module, and a simple user interface.

1.1 Retrieval Module

Students are required to implement **at least one** retrieval method selected from the following categories:

- Sparse retrieval (e.g., TF-IDF, BM25, or predicted term weights from neural models)
- Static embedding retrieval (e.g., Word2Vec, GloVe, model2vec)
- Dense retrieval (encoder-based models) (e.g., E5, BGE, GTE)
- Dense retrieval with instruction (LLM-based models) (e.g., Qwen3-Embedding, E5-Mistral)
- Multi-vector retrieval (e.g., ColBERT)

If multiple retrieval methods are implemented, you may either combine them into a *hybrid retrieval system* (i.e., a combination of the above methods) or simply include them as separate options within your system interface.

Grading rule. The retrieval component will be evaluated based on the diversity of implemented methods. Implementing **a single** retrieval method will earn **at least half** the grade from the retrieval module; implementing all methods guarantees full marks for this part.

1.2 Generation Module

This component focuses on integrating a generative model to produce grounded answers based on retrieved evidence. You **must** complete the *Basic Single-Turn RAG* component, and you may optionally implement *Feature A* and/or *Feature B* to earn additional credit. Implementing both optional features will result in full marks for this section. The evaluation of whether optional features have been implemented will be determined by the written report, the demo video, and the code. The generator model(s) must be selected from the following: **Qwen2.5-0.5B-Instruct**, **Qwen2.5-1.5B-Instruct**, **Qwen2.5-3B-Instruct**, and / or **Qwen2.5-7B-Instruct**.

Basic Single-Turn RAG Implement a simple RAG pipeline: feed the retrieved passages together with the user query into the LLM to generate an answer. You should consider the design of the prompt template—how to order and place the instruction, the question, and the retrieved passages. Think about how to extract the answer cleanly and how to instruct the LLM to respond in a consistent, easily parseable format.

Feature A: Multi-Turn Search Extend your system to support multi-turn interaction and context-aware retrieval. For example:

> Q: “Where was Barack Obama born?” A: Hawaii, USA.
> Q2: “What about his wife, where was she born?” A2: Chicago, Illinois, USA.

You need to design a mechanism to maintain dialogue memory (e.g., tracking entities or conversation history) and reformulate follow-up queries into self-contained queries before retrieval. Consider how to manage long contexts—such as pruning earlier retrieved passages—to keep prompts efficient. A well-designed system should correctly link references (e.g., “his wife”) and retrieve the appropriate evidence for each turn.

Feature B: Agentic Workflow Go beyond direct question answering by introducing intermediate workflow or steps that improve result quality. For instance, you might include:

- **Query rewriting and planning**: Decompose complex queries into sub-queries and perform parallel retrieval.
- **Self-checking:** verify the generated answer against retrieved evidence or run a secondary check to detect hallucination.
- **Explicit reasoning steps:** implement a simple ReAct-style plan–act–reflect loop or chain-of-thought prompting that guides retrieval and synthesis.

You are fully welcome to design more advanced solutions. However, the most effective designs should be those that are mindful of the trade-off between computational cost and performance.

1.3 User Interface

The user interface can be implemented as either a **terminal-based** or a **web-based** application. Its goal is to enable users to interact with the retrieval-augmented generation (RAG) system intuitively and to display both results and reasoning processes if available.

Basic Requirement The interface should allow users to input a query, display the retrieved documents, and present the final generated answer from the LLM. A clean, readable layout will be positively considered.

Bonus Score A bonus score will be awarded if the interface additionally shows the system's intermediate reasoning or workflow results—such as generated sub-questions, intermediate retrieval outputs, or self-checking steps. This requires the corresponding components (*Feature B: Agentic Workflow*) to be implemented in the *Generation Module*.

2.0 Written Report (20%)

The report must be at least **6 pages** (excluding cover page), using **12-point font size** and **single line spacing**. Submit the final version in **PDF format**. TAs will release a Microsoft Word template at the github repo `https://github.com/polyunlp/COMP5423-25Fall`.

Content Requirements

- **Cover Page:** Include the project title, group number, names of all group members, and corresponding student IDs.
- **Team Contribution:** Clearly specify the **role** and contribution percentage of each member. Example: Student A (30%), Student B (25%), Student C (25%), Student D (20%).
- **System Design Methodology:** Describe the overall design strategy and key components of your system, including how retrieval and generation modules are integrated.
- **System Flowchart:** Provide a labeled diagram that visualizes the workflow or data pipeline of your system, from query input to final answer generation.
- **Result Analysis:** Analyze and discuss experimental outcomes. Highlight differences under varying configurations (e.g., different retrieval methods, prompts, or generation settings). Include relevant tables or figures to support your analysis.
- **User Interface Design:** Present screenshots or diagrams of your UI. Explain how users interact with the system and how the interface displays retrieved evidence and generated answers.
- **Additional Exploration (Optional):** Share any extra attempts, such as experiments with prompt tuning, reasoning, or performance optimization.
- **Electronic Signatures:** At the end of the report, include electronic signatures of all group members to confirm the authenticity and accuracy of the information provided.

3.0 Demo Video (20%)

The demo video should clearly demonstrate your system in action and explain its key components. It should be ≤ **5** minutes and submitted in `.mp4` format.

Content Requirements

- Demonstrate the system's real-time response capability and highlight key aspects of your solution, including the main challenges you addressed and any advanced features implemented beyond the basic requirements.
- Provide an overview of your code structure, explaining which components handle data preprocessing, retrieval, and answer generation. The goal is to show that you understand the logic and workflow of your implementation.

4.0 System Performance (20%)

System performance will be evaluated using your submitted `test_prediction.jsonl` file. The TAs will compare your predictions against the hidden reference file to determine your system's overall ranking among all groups. One demo submission file is at `https://github.com/polyunlp/COMP5423-25Fall/blob/main/group_project/result-demo.jsonl` . Each line in `test_prediction.jsonl` must be a valid JSON object in the following format:

```
{"id": <str>, "question": <str>, "answer": <str>, "retrieved_docs": [[id_1, score_1],
                                [id_2, score_2], ..., [id_10, score_10]]}
```

The `question` and `answer` fields should be plain text strings, and the `retrieved_dcs` field should list **10** retrieved pairs of document index and score corresponding to the passages used for generation.

Evaluation Metrics. Performance will be computed using the official evaluation script (`metrics_calculation.py`), which calculates two complementary metrics:

- **Answer Accuracy :** measured by *Exact Match (EM)* between the predicted and reference answers.
- **Retrieval Quality :** measured by *nDCG@10* over the retrieved documents.

It worth to note that both metrics contribute proportionally to the final score, which determines your relative ranking among all teams.

Scoring Scheme. Your final score (out of 20%) will be assigned according to your ranking percentile as follows:

Top 10% (inclusive)	→	100% of total score
10% – 20%	→	90%
20% – 30%	→	80%
30% – 40%	→	70%
40% – 50%	→	60%
50% – 60%	→	50%
60% – 70%	→	40%
70% – 80%	→	30%
80% – 90%	→	20%
Below 90%	→	10%
Blank submission	→	0%

Example. If there are 40 groups in total and your system ranks 16th overall, your percentile falls within the 30%–40% range. Therefore, you will receive $20 \times 70\% = 14$ points for this component.

Submission

All materials should be packaged into a single compressed file (`.zip`) and submitted via **Blackboard**. The submission must contain the following components:

1. Output File. Include your prediction file `test_predict.jsonl`, formatted exactly as specified in the *System Performance* section.

2. Code Directory. Include a directory named `code/`, which should contain:

- All source code files necessary to reproduce your system.
- A `README.md` that clearly describes:
  - The structure of the code base.
  - Step-by-step instructions for running your system and reproducing `test_predict.jsonl`.
  - Environment setup instructions, specifying the required packages and their versions. You may provide this information through a `requirements.txt` file, an `environment.yml` file, or a detailed dependency list within the README itself.
- Appropriate annotations or docstrings at the beginning of each major code file, briefly explaining its functionality and any important notes for the reader.

3. Others. Include the **project report** in PDF format (`Group-X-Report.pdf`) and the **demo video** in MP4 format (`Group-X-Video.mp4`).

Note. Submissions that fail to follow the directory structure or file format may not be evaluated correctly. Please test your packaged submission locally before uploading to ensure that it can be executed end-to-end. To avoid submitting excessively large files, you do **not** need to include model weight files directly in your package. If necessary, provide access to them via a shared cloud link (e.g., Google Drive, OneDrive, or HuggingFace model hub). Make sure the link has public read permission.

Please note that project material submissions are required to be made on a group basis, i.e., any group member can submit on behalf of the entire group. Even if you wish to work on the project individually, it is still necessary to register, as failure to do so may result in restrictions on submitting through Blackboard.

Reference

Papers

1. Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing.
2. Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint 2023.
3. Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. The Eleventh International Conference on Learning Representations

Tools & Models

1. BM25s `https://github.com/xhluca/bm25s`
2. faiss `https://github.com/facebookresearch/faiss`
3. Milvus `https://github.com/milvus-io/milvus`
4. model2vec `https://github.com/MinishLab/model2vec`
5. FlagEmbedding `https://github.com/FlagOpen/FlagEmbedding`
6. GTE `https://huggingface.co/Alibaba-NLP/gte-modernbert-base`
7. E5-mistral `https://huggingface.co/intfloat/e5-mistral-7b-instruct`
8. Qwen3-Embedding `https://github.com/QwenLM/Qwen3-Embedding`
9. GTE-ColBERT `https://huggingface.co/lightonai/GTE-ModernColBERT-v1`
10. Ollama `https://ollama.com`
11. LangChain `https://github.com/langchain-ai/langchain`
12. LlamaIndex `https://github.com/run-llama/llama_index`
13. RAGFlow `https://github.com/infiniflow/ragflow`
14. LightRAG `https://github.com/HKUDS/LightRAG`

Free GPU or API Resources

1. Google Colab or Kaggle Kernel
2. OpenRouter API `https://openrouter.ai`
3. SiliconFlow API `https://siliconflow.cn`

Project instruction version 1.0