

# HOMWORK 5<sup>\*</sup>

## MMR AND AMGPT<sup>†</sup>

24-880 AI AGENTS FOR ENGINEERS  
<https://cmuagents.github.io>

OUT: February 22th, 2026  
DUE: March 1st, 2026  
TAs: Peter Pak  
CAs: Hardik Choudhary

### Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
  - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template for the submissions which can be written in  $\text{\LaTeX}$  (via [Overleaf](#)) or directly on the PDF document. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
  - **Programming:** Gradescope's Autograder will be used to evaluate the coding portions of this assignment. In order to best obtain the points for this assignment, please follow the code upload instructions. If you do not follow the instructions correctly the autograder may not work properly.

| Question                                  | Points |
|-------------------------------------------|--------|
| Maximum Marginal Relevance (Written)      | 15     |
| Maximum Marginal Relevance (Programming)  | 10     |
| Additive Manufacturing GPT (Written)      | 70     |
| Programming Scripts Upload                | 0      |
| Collaboration and Generative AI Questions | 5      |
| Total:                                    | 100    |

---

<sup>\*</sup>Adapted from CMU 10-623 Generative AI Homework Template

<sup>†</sup>Compiled on Sunday 22<sup>nd</sup> February, 2026 at 20:32

## Introduction

In this homework assignment we'll take a look into the application of Retrieval Augmented Generation (RAG) within the work AMGPT along with some additional concepts.

At a high-level, this homework will cover the following aspects.

1. Explore Maximum Marginal Relevance (MMR) and why it is useful.
2. Simple implementation example of MMR.
3. Run through a simplified implementation of AMGPT and related concepts.

## Local Environment

For this homework we'll be using `uv` as the project and package manager. If you haven't install `uv`, you can follow the installation instructions here <https://docs.astral.sh/uv/>. You can explore all of the available commands using `uv --help` but in short, here are a couple of useful commands that you'll often use:

|                                            |                                                                        |
|--------------------------------------------|------------------------------------------------------------------------|
| <code>source .venv/bin/activate</code>     | (Linux and Mac) Activates python environment.                          |
| <code>source .venv/Scripts/Activate</code> | (Windows) Activates python environment.                                |
| <code>uv sync</code>                       | Installs / synchronizes projects dependencies.                         |
| <code>uv add &lt;package&gt;</code>        | Installs <code>pip</code> package and adds it to project dependencies. |
| <code>uv remove &lt;package&gt;</code>     | Removes <code>pip</code> package dependency from project.              |
| <code>uv run &lt;script.py&gt;</code>      | Runs python script inside of <code>uv</code> environment.              |

1. Navigate to folder project directory and synchronize dependencies with:

```
uv sync
```

2. Activate the environment

Mac and Linux:

```
source .venv/bin/activate
```

Windows:

```
source .venv/Scripts/activate
```

3. Test `uv` works and run `main.py` file.

```
uv run main.py
```

# 1 Maximum Marginal Relevance (Written) (15 points)

Conventional information retrieval approaches often rank documents based on their relevance to the given query and utilize this ranking to inform selection of the most suitable result [Buckley \[1985\]](#), [Kupiec et al. \[1995\]](#), [Luhn \[1958\]](#), [Salton \[1988\]](#). This approach excels in obtaining matching results within cases where the archives of relevant documents are limited and few. However, in scenarios where there is an excess of potentially relevant documents (including redundant and duplicate results) the narrow approach of selection based solely on query relevant becomes insufficient for selecting the most suitable result [Goldstein and Carbonell \[1998\]](#). Rather, an approach that considers both the relevancy of the document to the query along with its novelty would address the previous presented issues of redundancy [Goldstein and Carbonell \[1998\]](#). The authors name this approach as Maximum Marginal Relevance (MMR) as it strives to maximize both the novelty and marginal relevance to the initial query through a user tunable parameter  $\lambda$ .

$$\text{MMR}(D_i) = \lambda \cdot \text{sim}_1(D_i, Q) - (1 - \lambda) \cdot \text{sim}_2(D_i, D_j) \quad (1)$$

The implementation (Eq. 1) utilizes the parameter  $\lambda$  to assign weight to two different similarity metrics ( $\text{sim}_1$  and  $\text{sim}_2$ ): that of the document's ( $D_i$ ) relevance to the query ( $Q$ ) and that of the document's relevance to other documents ( $D_j$ ). [Goldstein and Carbonell \[1998\]](#) found that setting  $\lambda$  to 0.3 in this user defined sampling strategy emphasized focus towards the provided query then to the most important parts of the information space. When utilized within RAG, this approach has been utilized to increase recall for the most relevant chunks that are associated with the input query (Fig. 2) [Wang et al. \[2025\]](#).

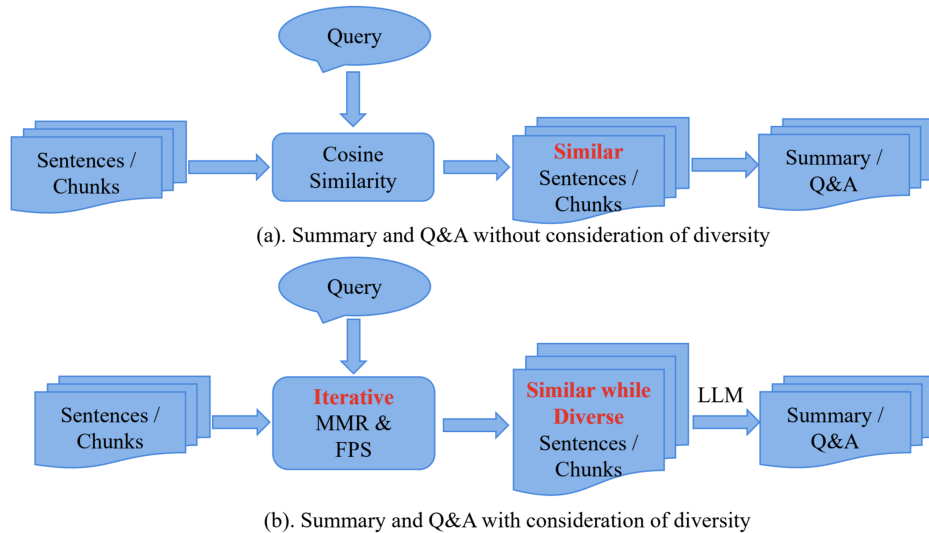


Figure 1: From [Wang et al. \[2025\]](#), “For both QA and summarization tasks, the initial dataset is divided into sentences or chunks, and corresponding embeddings are extracted. In a traditional pipeline, query embeddings are generated and used to select relevant materials to LLMs for downstream tasks. In contrast, methods like MMR and FPS incorporate diversity in a greedy manner when selecting relevant sentences. This approach increases the recall of the correct answer within the chosen sentences or chunks”

- 1.1. (5 points) What benefit does the Maximum Marginal Relevance (MMR) strategy provide over conventional information retrieval (IR)?

- 1.2. (3 points) According to Carbonell and Goldstein [1997], what should  $\lambda$  be set to in order to just use the standard relevance ranked list?
- ☐ 0
  - ☐ 0.3
  - ☐ 0.5
  - ☐ 1
- 1.3. (3 points) According to Carbonell and Goldstein [1997], what should  $\lambda$  be set to in order to maximize diversity?
- ☐ 0
  - ☐ 0.3
  - ☐ 0.5
  - ☐ 1
- 1.4. (4 points) According to Carbonell and Goldstein [1997], what approach did they find most useful when setting  $\lambda$
- ☐ Start at  $\lambda = 0$  and move closer towards  $\lambda = 0.3$
  - ☐ Start at  $\lambda = 1$  and move closer towards  $\lambda = 0.7$
  - ☐ Start at  $\lambda = 0.3$  and move closer towards  $\lambda = 0.7$
  - ☐ Start at  $\lambda = 0.7$  and move closer towards  $\lambda = 0.3$

## 2 Maximum Marginal Relevance (Programming) (10 points)

In this section, you will implement Maximal Marginal Relevance (MMR) for use in RAG systems. MMR helps select documents that are both relevant to the query **and** diverse from each other, reducing redundancy in the retrieved context.

Recall the MMR formula from the previous section:

$$\text{MMR}(D_i) = \lambda \cdot \text{sim}(D_i, Q) - (1 - \lambda) \cdot \max_{D_j \in S} \text{sim}(D_i, D_j) \quad (2)$$

where  $S$  is the set of already-selected documents.

**File to edit:** `programming/mmr_rag.py`

### 2.1. (2 points) Cosine Similarity

Implement the `compute_cosine_similarity(vec1, vec2)` function that computes the cosine similarity between two vectors:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \quad (3)$$

#### Requirements:

- Use `np.dot()` for the dot product
- Use `np.linalg.norm()` for vector magnitude
- Return 0.0 if either vector has zero magnitude

### 2.2. (2 points) MMR Score Computation

Implement the `compute_mmr_score()` function that computes the MMR score for a candidate document given:

- The candidate document embedding
- The query embedding
- List of already-selected document embeddings
- The  $\lambda$  parameter

#### Requirements:

- Use `compute_cosine_similarity()` for similarity calculations
- When no documents have been selected, the diversity penalty should be 0
- Apply the MMR formula correctly

**2.3. (2 points) MMR Document Selection**

Implement the `select_documents_mmr()` function that iteratively selects  $k$  documents using the MMR criterion.

**Algorithm:**

1. Initialize empty sets for selected documents
2. Repeat  $k$  times:
  - (a) Compute MMR score for each remaining document
  - (b) Select the document with the highest MMR score
  - (c) Add it to the selected set
3. Return the list of selected document indices

**2.4. (2 points) MMR Reranking**

Implement the `mmr_rerank()` function that reranks documents using a modified MMR formula that incorporates initial retrieval scores:

$$\text{MMR}(D_i) = \lambda \cdot \text{score}(D_i) - (1 - \lambda) \cdot \max_{D_j \in S} \text{sim}(D_i, D_j) \quad (4)$$

**Requirements:**

- Normalize initial scores to  $[0, 1]$  range before applying MMR
- Return a list of (index, mmr\_score) tuples

**2.5. (2 points) Testing Your Implementation**

Run the tests locally to verify your implementation:

```
uv run -m pytest tests/test_mmr_rag.py -v
```

Your implementation should pass all 5 tests (2.1 through 2.5).



### 3 Additive Manufacturing GPT (Written) (70 points)

Answer all questions below. Keep your responses concise, technically correct, and clearly written. This section is worth **70 points**.

General-purpose large language models (LLMs) can produce fluent answers, but in technical domains they often fall back on high-level summaries and may miss process-specific details (or hallucinate) when asked for actionable guidance. This is especially problematic in metal additive manufacturing, where correct answers depend on precise context: alloy system, process window, defects, and post-processing.

Chandrasekhar et al. introduce *AMGPT*, a domain-focused assistant that uses Retrieval-Augmented Generation (RAG) to ground an open-source LLM in a curated additive manufacturing literature corpus. Rather than training a new model from scratch, AMGPT retrieves relevant passages (via embeddings and similarity search over indexed documents) and injects them into the prompt so the generator can produce evidence-based, context-aware responses. In this written section, you will reason about the design decisions in such a RAG pipeline: retrieval, embeddings, prompting, and evaluation, and connect them to practical tradeoffs like specificity vs. coverage and accuracy vs. compute.

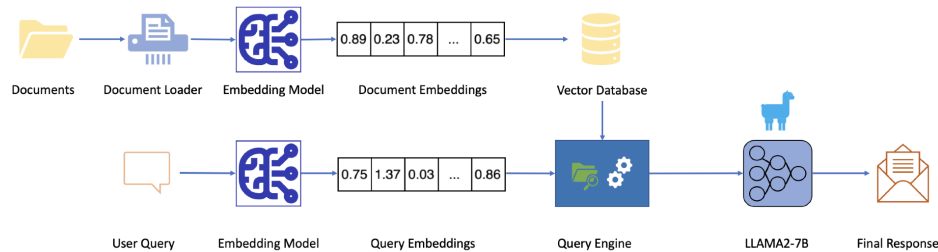


Figure 2: From [Chandrasekhar et al. \[2024\]](#), “Illustration of a Retrieval-Augmented Generation (RAG) workflow. Documents are loaded and processed into chunks, which are then embedded using an embedding model, creating vectors stored in a database. The query engine utilizes these vectors to match user queries against document chunks, and retrieves the most relevant content. Finally, the retrieved information is enhanced by the LLaMA2-7B language model to generate comprehensive and contextual responses.”

**Retrieval-Augmented Generation (RAG)**

- 3.1. (5 points) Define Retrieval-Augmented Generation (RAG). Explain how it combines an off-the-shelf LLM with external knowledge sources. What fundamental shortcomings of large language models does RAG seek to address?

- 3.2. (5 points) Explain why RAG is considered especially useful for specialized engineering fields such as additive manufacturing. Write an example of a domain-specific engineering query to illustrate how RAG improves response quality compared to a general-purpose LLM.

**Text Embeddings and Semantic Search**

- 3.3. (5 points) What is a text embedding? Explain how semantic search using embeddings differs from traditional keyword-based search.

- 3.4. (5 points) Describe the key characteristics of the embedding model used by the authors of AMGPT (e.g., dimensionality, context handling). Why did the authors choose this particular model?

**Fine-Tuning vs. RAG and Generation Parameters**

- 3.5. (8 points) Discuss why AMGPT prefers to use a pre-trained LLM combined with RAG rather than training or fine-tuning a model from scratch. Consider computational cost, task-specific adaptability, and hallucination.

3.6. (8 points) Define the following LLM decoding / generation parameters and explain their influence on output text. For each, state the typical range of values and which setting is most appropriate for **factual knowledge generation** versus **creative exploration**.

- Temperature
- Top- $k$
- Top- $p$  (nucleus sampling)
- Max tokens

**System Architecture of AMGPT**

3.7. (12 points) Describe the complete end-to-end architecture of the AMGPT system. Your answer should include:

- Preparation and preprocessing of the document corpus
- Embedding model and vector database
- Query encoding, similarity search, and retrieval
- Method used for instructing the LLM on steps to be taken
- Role of LlamaIndex and Hugging Face
- User interaction through the Streamlit interface

Clearly explain how data flows through the system from user query to final response.



**Embedding Models and Infrastructure**

- 3.8. (8 points) What is an embedding model and how does it differ from a generative language model? Explain the role of embedding models in enabling semantic retrieval within a RAG pipeline. Describe how the authors of AMGPT integrated their chosen embedding model into the system's retrieval workflow.

- 3.9. (7 points) Define what a vector store index is and explain its role within a RAG system. How does a vector store index enable efficient retrieval compared to brute-force search over raw documents? Describe how the authors of AMGPT utilized a vector store index in their implementation.

- 3.10. (7 points) The company Groq employs Language Processing Units (LPUs) for large language model inference. Define what an LPU is and explain how its architecture differs fundamentally from that of a GPU. What are the primary performance benefits of LPUs over GPUs for LLM inference workloads?

## 4 Programming Scripts Upload (0 points)

To upload your code to Gradescope, run the following command to generate a zip file:

```
uv run programming/zip_programming.py
```

**Not using the command above may cause Autograder to not work properly!**

Upload the generated zip file to the appropriate programming slot on Gradescope.

4.1. Make sure the following files are included when uploading to Gradescope

- `mmr_rag.py`

## 5 Collaboration and Generative AI Questions (5 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 5.1. (2 points) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

- 5.2. (3 points) Did you use generative AI for parts of this assignment? If so, include the input prompts along with links to the conversation.

## References

- Chris Buckley. Implementation of the SMART Information Retrieval System. Technical Report, Cornell University, USA, April 1985.
- Jaime G. Carbonell and Jade Goldstein. The Use of MMR and Diversity-Based Reranking in Document Reranking and Summarization. July 1997. doi: 10.1184/R1/6610814.v1. URL [https://kilthub.cmu.edu/articles/journal\\_contribution/The\\_Use\\_of\\_MMR\\_and\\_Diversity-Based\\_Reranking\\_in\\_Document\\_Reranking\\_and\\_Summarization/6610814/1](https://kilthub.cmu.edu/articles/journal_contribution/The_Use_of_MMR_and_Diversity-Based_Reranking_in_Document_Reranking_and_Summarization/6610814/1).
- Achuth Chandrasekhar, Jonathan Chan, Francis Ogoke, Olabode Ajenifujah, and Amir Barati Farimani. AMGPT: a Large Language Model for Contextual Querying in Additive Manufacturing, May 2024. URL <http://arxiv.org/abs/2406.00031>. arXiv:2406.00031.
- Jade Goldstein and Jaime Carbonell. Summarization: (1) using MMR for diversity - based reranking and (2) evaluating summaries. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998*, TIPSTER '98, pages 181–195, USA, October 1998. Association for Computational Linguistics. doi: 10.3115/1119089.1119120. URL <https://dl.acm.org/doi/10.3115/1119089.1119120>.
- Julian Kupiec, Jan Pedersen, and Francine Chen. A trainable document summarizer. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '95*, pages 68–73, Seattle, Washington, United States, 1995. ACM Press. ISBN 978-0-89791-714-8. doi: 10.1145/215206.215333. URL <http://portal.acm.org/citation.cfm?doid=215206.215333>.
- H. P. Luhn. The Automatic Creation of Literature Abstracts. *IBM Journal of Research and Development*, 2(2):159–165, April 1958. ISSN 0018-8646, 0018-8646. doi: 10.1147/rd.22.0159. URL <http://ieeexplore.ieee.org/document/5392672/>.
- Gerald Salton. *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., USA, March 1988.
- Zhichao Wang, Bin Bi, Yanqi Luo, Sitaram Asur, and Claire Na Cheng. Diversity Enhances an LLM's Performance in RAG and Long-context Task, April 2025. URL <http://arxiv.org/abs/2502.09017>. arXiv:2502.09017 [cs].