Version: 1.0

RAG_against_the

Will you answer my questions?

Summary

Retrieval Augemented Generation, that's it.
That's the goal of this project .
Made in collaboration with @ldevelle, @pcamaren, @crfernan

#LLM

#AI

Intellectual Property Disclaimer

All content presented in this training module, including but not limited to texts, images, graphics, and other materials, is protected by intellectual property rights held by Association 42.

Terms of Use:

- **Personal use:** You are permitted to use the contents of this module solely for personal purpose. Any commercial use, reproduction, distribution, modification, or public display is strictly prohibited without prior written permission from Association 42.
- **Respect for Integrity:** You must not alter, transform, or adapt the content in any way that could harm its integrity.

Protection of Rights:

Any violation of these terms constitutes an infringement of intellectual property rights and may result in legal action. We reserve the right to take all necessary measures to protect our rights, including but not limited to claims for damages.

For any questions regarding the use of the content or to obtain authorization, please contact: `legal@42.fr`

Contents

Chapter 1

Foreword

The **birthday paradox** is a classic problem in probability theory that demonstrates how unexpected events can occur more frequently than expected. It shows that even when the probability of an event is very low, it can still happen if there are enough opportunities.

In order to understand it better, let's make a guess:

If we take a classroom of 23 students, what is the probability that at least two of them have the same birthday?

This problem is a veridical paradox, which regroups a great lists of problems that seem to be false but are in fact, true.
Enough suspense, the probability is: 50% ! Who would have guessed ? Here is the formula :

$$1 - \left(\frac{364}{365}\right)^{n(n-1)/2}$$

With n being the number of students. The probability is 50% when $n = 23$.

The most surprising is yet to come. If we take a classroom of 70 students, the probability is 99.9% that at least two of them have the same birthday.

"Hey good piece of trivia, but where am I going with this?" - you may ask.
In cryptography, we now find a type of attack that takes advantage of the birthday paradox to find collisions in a hash function. It has been called: **the birthday attack**.

Now that you know that our instincts can be wrong and that maths leads you to bruteforce, let's move on to the project.

Chapter 2

Common Instructions

2.1 General Rules

- You must use **Python 3.10** for this project.
- All classes must use `pydantic` for validation and type safety.
- Your project must adhere to the **flake8** coding standard. Bonus files are also subject to this standard.
- Your functions must handle exceptions gracefully to avoid crashes. Use try-except blocks to manage potential errors. If your program crashes due to unhandled exceptions during the review, it will be considered non-functional.
- All resources (e.g., file handles, network connections) must be properly managed to prevent leaks.

2.2 Additional Guidelines

- You can use the libraries you want. We recommend `dspy`, `fire`, `tqdm`, `langchain`, `bm25s`, `chromadb`, `chonkie` packages.
- You can use the following models:
  - `ollama_chat/qwen3:0.6b` (default)
  - Feel free to use other models (using the names from the HuggingFace hub) during the beta and let us know!
- We will use `uv` as a project and package manager.
- Your system must provide a CLI interface using Python Fire.
- Progress bars should be implemented for long-running operations using `tqdm`.

2.3 Context

2.3.1 Overview

A new project is often linked to new techniques and new skills: we've seen **function calling** in *call_me_maybe* and we will continue our exploration into the world of AI in this project. The main topic we're going to approach is **RAG**. But before seeing what **RAG** is about in its substance, let's focus on what it does! To do so, let's take some height.

When you want to work with a model in AI, the first thing to come in mind is to train it. You want the model to be able to use **language, reasoning, structure** and in order to do so you'll feed it a huge amount of data. Once you've done the training, the model "remembers" what you've fed it but it "knows" only the data that you've given to it. If you want to have fresher knowledge, you'll have to retrain it. And it takes a long time.

Training is a *technique*, and **RAG** is another one. Instead of feeding the model data, the RAG will give the model access to an external source of data, and that source is of *your choice*.
The techniques can be combined: the model has to be trained on the key concepts such as we've seen before (language, reasoning, structure) to have the basis but for the knowledge, it can combine the trained data and the external connection.

2.3.2 What is RAG ?

Now that we know where we're at, you might ask yourself (if you haven't looked it up yet!) what is **RAG**? To understand it, we'll break it down into its three key concepts:

- **Retrieving**: Since the model is not trained on your specific data, it needs to search the database to *retrieve* the most useful snippets. First, the data has to be prepared. Then the model needs to understand your question. Once that's done, it matches the query with the database to choose the best results and finally pulls out the most relevant pieces of information. This involves **indexing**, **query encoding**, **similarity search**, **ranking**, and **retrieving**.

- **Augmenting**: Once the AI has retrieved the information, it can combine it with what it already "knows." This is called **augmenting**, since the AI is expanding its abilities by adding new information. Starting from the retrieved results, you can clean and filter them to remove irrelevant snippets (to avoid potential **noise**), insert them into the **context window**, and then combine them with the **trained knowledge** (the real augmentation step!).

- **Generating**: Now that you have retrieved the information and augmented it, the AI can finally generate an answer! Whether it's writing text, explaining a concept, or producing code snippets, this is the visible outcome of RAG. To do so, the AI reads the **context window**, understands the task at hand, blends the knowledge, and generates the output. Modern RAG systems often refine while writing, adjusting phrasing on the fly to maintain coherence and match the tone requested in the query.

Now that everything is clear, let's move forward !

2.4 Mandatory part

Yes, yes, we know you know. Since you've read the summary carefully, you already understand that you'll be coding a **Retrieval-Augmented Generation (RAG)**.

Your system should demonstrate the ability to:

- Build an indexed knowledge base from the repository files (using both docs and code files).
- Retrieve and rank the most relevant pieces of information.
- Pass them to the LLM within context limitations.
- Generate structured JSON output as described in the output section.
- Implement intelligent chunking strategies for different file types.
- Provide a comprehensive CLI interface for all operations.
- Include evaluation metrics and performance analysis.

Don't panic ! Start by measuring your error using the simplest possible approach.
Advance to complex methods once your error measurement is improving.

2.4.1 Performances

Your system must respect some minimal performances that are listed as follow:

- Indexing time: 5 minutes maximum
- Retrieval time: 1 minute for 1 question
- Question answering time : 1.8 seconds for 1 question

BETA : If you do not meet the expected performances, talk about it on discord

2.5 Data Models

Your program must implement the following Pydantic models for type-safe data handling. These models ensure data integrity and provide automatic validation throughout the pipeline.

The `MinimalSource` model represents a minimal source of information:

```
class MinimalSource(BaseModel):
    file_path: str
    first_character_index: int
    last_character_index: int
```

The `UnansweredQuestion` and `AnsweredQuestion` model represents an unanswered question and an answered question:

```
class UnansweredQuestion(BaseModel):
    question_id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    question: str

class AnsweredQuestion(UnansweredQuestion):
    sources: List[MinimalSource]
    answer: str
```

The `RagDataset` model represents a dataset of RAG questions:

```
class RagDataset(BaseModel):
    rag_questions: List[AnsweredQuestion, UnansweredQuestion]
```

The `MinimalSearchResults` and `MinimalAnswer` model represents the search results and an answer:

```
class MinimalSearchResults(BaseModel):
    question_id: str
    retrieved_sources: List[MinimalSource]

class MinimalAnswer(MinimalSearchResults):
    answer: str
```

The `StudentSearchResults` and `StudentSearchResultsAndAnswer` model represents a search results and a search results and an answer:

```
class StudentSearchResults(BaseModel):
    search_results: List[MinimalSearchResults]
    k: int

class StudentSearchResultsAndAnswer(StudentSearchResults):
    search_results: List[MinimalAnswer]
```

The provided models are a foundation. You can expand them by adding new models or extra fields (for example in the search results model) if your implementation requires it.

2.6 Smart Chunking Strategy

Your system must implement different strategies:

Python Code Chunking

Here are some hints about how to implement this strategy:

- Use Abstract Syntax Tree (AST) parsing to understand code structure
- Keep functions and classes together as logical units
- Split on scope boundaries (function/class definitions)
- Preserve code integrity between chunks while respecting size limits
- Have a minimal tokenization to preserve code structure for the retrieval system

Documentation Chunking

- Split by headers to maintain semantic structure (e.g. in Markdown, split on #, ##, etc.)
- Add overlap between chunks for context preservation
- Handle large sections with sentence-aware splitting
- Full tokenization with stemming and stopword removal

The maximum chunk size is 2000 characters and it has to be configurable through a variable.

2.7 Retrieving Method

Your program must implement at least one basic retrieving method:

- TF-IDF
- BM25

Performance Target: Your retrieval method implementation should achieve at least **BM25:75% recall@5** on English questions when evaluated against the provided test set. **TF-IDF:65%recall@5** on English questions when evaluated against the provided test set.

2.7.1 Command-Line Interface

Your program must provide a comprehensive command-line interface using Python Fire.

Required Commands:

- **ingest**: Index the repository

```
uv run python -m src index
```

- **search**: Search the indexed repository

```
uv run python -m src search "OpenAI compatible server" --k 10
with k being a configurable parameter representing the number
```

- **search_dataset**: Search dataset and search results

```
uv run python -m src search_dataset \
data/datasets/UnansweredQuestions/Dataset_2025-09-21_valid.json
```

- **evaluate**: Evaluate search results

```
uv run python -m src measure_recall_at_k_on_dataset \
data/output/search_results/Dataset_2025-09-21.json \
data/datasets/AnsweredQuestions/Dataset_2025-09-21_valid.json
```

- **generate**: Generate answers for dataset

```
uv run python -m src answer_dataset \
data/output/search_results/Dataset_2025-09-21_valid.json
```

- **answer**: Answer single query with context

```
uv run python -m src answer "How to configure OpenAI server?" --k 10
```

Think of these commands as a base template: you can add extra commands or customize them with flags according to your project's needs.

2.8 Input

Ingestion Options:

- **Full Repository**: Index all files in the vLLM repository
- **Selective Ingestion**: Only process files mentioned in questions.tsv (recommended for testing)

For each query, your system must retrieve relevant chunks of the repository and generate an evidence-based response in the same form as the output.

> Linked to the different chunking strategies, you can create different indexes for the different types of files.

2.9 Output

Your program must output a comprehensive JSON file containing detailed results and metadata:
Your system must output JSON files that conform to the provided Pydantic models:

- **For search operations**: Use `StudentSearchResults` model with:
  - `search_results`: List of `MinimalSearchResults` containing question_id and retrieved_sources
  - `k`: Number of results requested
- **For answer generation**: Use `StudentSearchResultsAndAnswer` model with:
  - `search_results`: List of `MinimalAnswer` containing question_id, retrieved_sources, and answer
  - `k`: Number of results requested
- **Source information**: Each `MinimalSource` contains:
  - `file_path`: Full path to the source file
  - `first_character_index`: Starting character position
  - `last_character_index`: Ending character position

Output Format

Your system must provide detailed output. The output must respect the minimal basis of the provided models but can be enhanced as follows:

```
"search_results": [
    {
        "question_id": "q1",
        "retrieved_sources": [
            {
                "file_path": "docs/serving/openai_compatible_server.md",
                "first_character_index": 9867,
                "last_character_index": 10100
            },
            {
                "file_path": "vllm/entrypoints/openai/api_server.py",
                "first_character_index": 267,
                "last_character_index": 400
            }
        ]
    }
],
"k": 10
```

For answers, the output should follow the `StudentSearchResultsAndAnswer` model:

```
"search_results": [
    {
        "question_id": "q1",
        "retrieved_sources": [
            {
                "file_path": "docs/serving/openai_compatible_server.md",
                "first_character_index": 9867,
                "last_character_index": 10100
            },
            {
                "file_path": "vllm/entrypoints/openai/api_server.py",
                "first_character_index": 267,
                "last_character_index": 400
            }
        ],
        "answer": "To configure the OpenAI compatible server in vLLM..."
    }
],
"k": 10
```

2.10 Evaluation

The evaluation of the program is performed using a **recall@k** metric that measures the effectiveness of the retrieval component.

Recall@k Calculation

The recall@k for a given question is calculated by checking how much the retrieved sources overlap with the correct sources.

A source is considered "found" if there is at least 5% overlap between the retrieved source and any correct source.

If there are multiple sources in the question, their retrieval score for that question is

$$\frac{\text{number\_found}}{\text{total\_sources}}$$

.

Dataset-Level Evaluation

For the recall on the dataset, it is the average of the recall over all questions.

2.11 Optional part

Your system can implement an advanced retrieving strategy:

- Embedding-based retrieval with semantic similarity
- Hybrid approaches combining lexical and semantic search
- Query expansion and refinement techniques

Remember, this is not just about building a search engine – you're creating a comprehensive RAG system that intelligently processes, indexes, and retrieves information to augment language model capabilities. The birthday paradox taught us that our instincts can be wrong; let your implementation prove that systematic approaches and solid engineering can build something truly effective!

Chapter 3

Beta

Thank you for participating in the beta of this project. We are looking forward to your feedback.