# Assignment 3 – Natural Language Processing 2024 – 2025 (JM2050)

For this assignment, you will be using the open-source RAG Me Up repository which can be found here: https://github.com/AI-Commandos/RAGMeUp. This repository contains a RAG framework written on top of Langchain as a server and web UI written in Scala.

## Main Goal

The main goal of the assignment will be to modify **one** of the components of the RAG pipeline that is used by default by the RAG Me Up repository and make a pull-request of your work on Github.

You are forced to use at least this course's material as the data for the RAG pipeline, in order to create a virtual study assistant for yourself.

## Guidelines

The following guidelines should be taken into consideration:

- You will have to create a Github account (for yourself or your team) if you don't have one already an fork the repository.
- The diagram shown under https://github.com/AI-Commandos/RAGMeUp?tab=readme-ov-file#how-does-rag-me-up-work gives you an idea of what the current components of the default RAG Me Up framework look like. You have two choices for this assignment:
    1. Replace an existing component with something else. For example, you can replace the current reranker component with a ColBERT implementation.
    2. Add a new component to the RAG flow. For example, you can add Text2SQL as a retrieval mechanism.
- The deliverable should be a pull-request from your fork to the RAG Me Up repository. A link to your PR will suffice. In addition you have to create a (brief) report of what you did and how you implemented your component. This can either be uploaded to Canvas or written in markdown as your readme.md in your fork.
- You may showcase use of other data (types) if this is required to showcase your code's novelty but you have to at least make use of the course materials and make sure your PR will work with that data.
- Unless required otherwise for your PR, you have to make use of BM25 and Milvus (not Postgres). Loading the data into a (local) Milvus DB is something RAG Me Up will do automatically for you. The resulting DB files should be uploaded to Canvas.

## Background and Inspiration

The RAG Me Up repository is written in Python on top of Langchain. With the theory obtained in the course, you should be able to understand each of the components in the framework and you will notice most are implemented on top of or using Langchain directly.

As described, you are tasked with creating a novel component in the RAG chain or modify an existing one in a novel way. While a bit arbitrary, it stands without question that some components are (far) more complex than others and it is up to you to choose a suitable level of complexity to tackle. To help you along the way, here are some considerations to take into account or yield inspiration from:

- While the document loaders and chunking are integral and important aspects of the RAG pipeline, simply adding support for a new document type or a different chunker is in no way novel. This changes of course when you try to support a new type of data

altogether, for example tabular or structured data, for which other loaders and chunkers are required.

- Some hot fields in RAG that could be interesting:
    - Support for late interaction retrieval and/or reranking (through eg. ColBERT)
    - GraphRAG
    - Entity extraction and retrieval
    - Text2SQL
    - Support for filter queries (like: give me all action movies of the last year with a rating of at least 8.0).
    - (Automatic) evaluation of RAG pipelines, hallucinations, etc.
    - Agentic RAG
    - Moving from Langchain to Langgraph
    - Multi-agent support
    - Multi-modality (eg. Through vision transformers or OCR)
    - Data leakage/prompt injection and guardrailing

The complexity of the task chosen will be factored into the evaluation and grading of your work so make sure to choose something you deem challenging enough but small enough to chew. If you have ideas or doubts, feel free to reach out at erik@futureclub.nl.

### Instructions

Since this assignment has to be run on Google Colab and given the fact that the RAG framework uses a server/client architecture, you have to run a server on Google Colab. This not a straightforward and simple task but there is a way to achieve just this using a reverse proxy via ngrok. You will have to follow these steps to make this work, which you have executed in the tutorial during the lectures as well.

1. You have to sign up for Google Colab. You can use any e-mail address you like, it doesn't have to be a personal or gmail account. You can sign up here https://colab.research.google.com/
2. You have to create an account at ngrok via https://ngrok.com/. There are paid plans but a free account will suffice. (Tip: Use Google login with the same account you used for Colab).
3. Once signed up, on the left menu you will be able to find "Your authtoken". You will need this later on.
4. Clone and use this notebook if you haven't done this in the tutorial already: https://colab.research.google.com/drive/1pA3ZMetOvImXYv7jSb3Q6cHFXa7pRp9-?usp=sharing
5. When running this notebook, make sure you select a GPU backend, otherwise you gain nothing in terms of speed versus running locally.
6. In the notebook in cell 4 you will need to place you ngrok auth token.
7. If you want to use gated LLMs like LLaMa 3.1 Instruct (which is the default), you will need to create a Huggingface account at https://huggingface.co/ and request access to the specific model(s). This usually takes a few minutes, hours tops. The default LLaMa 3.1 8B Instruct model can be found here: https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
8. To tell your notebook who you are and whether you can access a Huggingface model, you will need a Huggingface access token. You can create one by logging into Huggingface and clicking on your profile icon at the top right menu, then navigating to "Access tokens".

9. When you run the cells in the notebook, there will be 1 interactive cell asking you for your Huggingface access token before the rest of the notebook will be executed. Be sure to enter your token there.
10. When you fully run the notebook, the LLM of your choice will be downloaded and an ngrok reverse proxy will be started. In the output of your last cell, you will see something like this: * Tunnel URL: NgrokTunnel: "https://xyz-12-34-56-78.ngrok-free.app" -> "http://localhost:5000". You can now use the .ngrok-free.app URL to use in your RAG Me Up application.conf UI to connect to your server run on Google Colab.
11. You can now clone the RAG Me Up repository locally too and start the server from the ui/scala folder by executing the command `sbt run`. For this command to work you need to have a JDK 17 or newer installed and SBT (https://www.scala-sbt.org/).
12. Make sure you modify the application.conf file in ui/scala/conf to point to the right ngrok URL.
13. If you do not plan on modifying the UI but only the server, you can also skip installing JDK 17 (https://openjdk.org/projects/jdk/17/) and SBT and download a release of the UI. You will still need the **JRE** 17 (note: if you have a JDK, this includes the JRE by default) or newer but not a JDK and SBT. The latest pre-built release can be found here: https://github.com/AI-Commandos/RAGMeUp/releases/tag/scala-ui
    a. Do make sure you modify the application.conf to point to ngrok. You can run the pre-build binary by opening a shell and navigating to the **root folder** of the project. Then execute the following command: `bin/ragmeup -Dapplication.conf=conf/application.conf`
    b. If you are using the JDK with SBT, make sure the environment variable JAVA_HOME is set. This should be done automatically when installing a JDK but if not, add it manually and make sure it points to the root folder of your JDK, ie. `C:\Program Files\Java\jdk-17`