H02C8b Information Retrieval and Search Engines: Project (part 1 and 2)

prof. dr. Miryam de Lhoneux ir. Thomas Bauwens* Kushal Tatariya†

April 27, 2025

1 Introduction: RAG

Retrieval-augmented generation (RAG) (Lewis et al., 2020) is a common technique in contemporary natural-language processing (NLP) that incorporates various concepts introduced in this course. It is one of the principal algorithms allowing contemporary chatbots like OpenAI's *ChatGPT* and Google's *Gemini* to stay up-to-date with new information without being continually re-trained. Open-source generative language models (LMs) like *LLaMA* (Touvron et al., 2023) and *Mistral* (Jiang et al., 2023) offer a great opportunity to experiment with RAG, as we will see in this project.

Framework In short, a RAG system consists of two components: an *information retrieval (IR)* subsystem, and an LM subsystem. A user seeking information poses a *query* in natural language to the RAG system, and expects a response in natural language back. First, the IR subsystem searches a *knowledge base* for documents relevant to the query. Second, a *prompt* – a formatted set of instructions – is passed as input to the LM subsystem, containing a.o. the original query and the retrieved documents.

The idea behind RAG is that although LMs have good knowledge of the language, they might not have been trained on information from a specific domain, e.g. private or proprietary data such as the internal knowledge base of a company or government. Hence, its *generation* is *augmented* by supplying it with context from *retrieval* of documents. Though many extensions and variations of RAG have been proposed, you will implement what is colloquially called "naive RAG", formalised below.

Indexing RAG expects access to a knowledge base $\mathcal{D} = \{d_i\}_{i=1}^{N}$ of plain-text documents d_i. An encoding function f_D is employed to convert $\mathcal{D}$ into vectorised representations consisting of H features, such that $f_D(\mathcal{D}) \in \mathbb{R}^{N \times H}$, where the ith row of $f_D(\mathcal{D})$ corresponds to document $d_i \in \mathcal{D}$. The function f_D can be any feature extraction mechanism that converts text into numerical features, such as TF-IDF.

Retrieval During the retrieval step, a user query q is transformed by an encoding function f_Q (naively taken to be $f_Q = f_D$). The encoded query $f_Q(q)$ is then cross-referenced with the encoded corpus $f_D(\mathcal{D})$ to obtain relevancy scores for each document, $\vec{r} = \mathcal{S}(f_D(\mathcal{D}), f_Q(q)) \in \mathbb{R}^N$. Finally, the positions in $\vec{r}$ with largest score are looked up in $\mathcal{D}$ to get a list of relevant documents $\hat{d}_1 \dots \hat{d}_k$.

Generation For the generation step, information from q and $\hat{d}_1 \dots \hat{d}_k$ (for example, the full text from each) is integrated alongside prompt instructions in natural language by some formatter $\mathcal{I}(q, [\hat{d}_1 \dots \hat{d}_k])$. The result is passed to an instruction-trained LM (Ouyang et al., 2022) which generates a response $\hat{y}$ conditioned on the query, relevant document(s), and other prompt content: $\hat{y} = \text{LM}(\mathcal{I}(q, [\hat{d}_1 \dots \hat{d}_k]))$.

2 Problem Statement

In this assignment, you will implement a naive RAG system in Python 3. There are two parts: one to build the core of the RAG system, and one to extend it with advanced features.

*thomas.bauwens@kuleuven.be
†kushaljayesh.tatariya@kuleuven.be

Evaluation To benchmark your IR subsystem, you will, for each dataset, receive a set $\mathcal{Q}$ of queries q_j along with the K_j IDs of the documents $d_{i_{(j,1)}} \dots d_{i_{(j,K_j)}}$ that have any relevance to each query, as well as scores $r_{(j,1)} \dots r_{(j,K_j)}$ grading the relative relevance between those documents on some integer scale $\geqslant 1$ and allowing duplicates. These $\mathcal{Q} = \{(q_j, [i_{(j,1)} \dots i_{(j,K_j)}], [r_{(j,1)} \dots r_{(j,K_j)}])\}_{j=1}^{M}$ will have JSON format.

Model You can use any publicly accessible generative LM, although we do recommend you stick to a transformer (Vaswani et al., 2017) fine-tuned to be instructed (similar to ChatGPT, and unlike GPT-3) that can be found as public checkpoints on the HuggingFace hub. We provide you with the code to download and load one such model, `Mistral-7B-Instruct` (Jiang et al., 2023), with some settings that make it easier to fit in memory. As long as your choice is adequately motivated, you're also free to explore OpenAI's API, although help from the TAs will be limited to the HuggingFace ecosystem.

Compute We provide code on Google Colab.[1] We encourage you to use Colab for its free GPU access (on a limited daily basis), since you need a GPU for inference with LLMs. Colab allows you to choose when to run with and when to run without GPU, so make sure not to waste GPU time when you don't need it (more detailed instructions regarding this can be found on the Colab page).

If you have your own GPU, you are welcome to use it for this project, provided that it has enough VRAM for loading your model.[2]

2.1 Part 1: Core

You are given the *cooking recipes* dataset introduced by Majumder et al. (2019). This dataset contains over 180 000 recipes and over 700 000 recipe reviews covering 18 years of user interactions and uploads on *food.com*. We are only interested in the recipes subset for this project. We have filtered the dataset such that it contains only the relevant metadata for your implementation, and processed it for easy integration with the `datasets` library. For each recipe in the dataset, the following information is provided:

`name`	the name of the recipes as originally posted (e.g. "baked winter squash mexican style")
`tags`	the user-provided tags associated with the recipe (e.g. "60-minutes-or-less")
`description`	a story the user has attached to the recipe
`ingredients`	the list of ingredients required for making the recipe
`steps`	the step-by-step instructions for making the recipe, in order

You are tasked with constructing a RAG pipeline with word-level IR that answers queries about this recipe dataset, and discussing the following points.

1. Architecture:
   - Draw a diagram containing all the relevant components needed to go from query to generative output. Don't forget to apply the relevant preprocessing steps.
   - Briefly, but precisely, describe the role of each component.
2. Term vocabulary:
   - Describe how you determine which terms should go in the term vocabulary V.
   - Describe if you pre-determine its size: if yes, how, and if no, why.
   - Add some kind of support for multi-word expressions and explain how you detect them.
3. Document embeddings:
   - Which fields of the dataset do you include to compute your document embeddings? You should validate this decision with an experiment down the line.
   - After embedding the documents d_i as TF-IDF vectors $\vec{d_i}$, what mechanism does your implementation use to derive query vectors? (Be precise!)
   - How does your mechanism handle queries that don't contain any words of your vocabulary?

[1] https://colab.research.google.com/drive/1xDUaWaBEAX_38L0XmDGFdegaeIFPXX34?usp=sharing

[2] In any case, you might want to limit your experiments to models smaller than 7B parameters, to ensure that you are not spending more time than necessary on inference.

4. Retrieval:
    - Which similarity/distance operator did you try to use for nearest-neighbours search?
    - If you vary the amount of retrieved neighbours for the above operator(s), explain your mechanism (there is a lot of room for creativity in your design here!), and if you don't, explain how you found your fixed amount.[3]
    - Report the macro- and micro-average of precision, recall and F_1 of documents your IR system retrieved versus the documents that should have been retrieved across $\mathcal{Q}$. That's *six* metrics.
    - Formally define *mean average precision (MAP)* (that is: show the formula you use to compute it, and explain what *every* symbol means *correctly* and *without room for ambiguity*). Also compute it for your system.
5. Qualitative analysis (IR):
    - Use the output of 5 different queries to show 5 different drawbacks of TF-IDF.
6. Prompt:
    - Give more than one set of instructions to guide your LM subsystem with the relevant information to answer the query. Give the exact prompt template each time. *Note:* make sure you read the documentation of the LM so that you know which syntax it was trained on!
    - Which fields of the dataset do you use in your instructions?
7. Qualitative analysis (LM):
    - Does the LM understand the RAG task?
    - Use the output of 5 different queries to show that the RAG system can answer questions that require *reasoning* across recipes, rather than the recipes *themselves* being an answer.
    - Verify that the LM is actually basing its answer on the supplied recipes, rather than its inherent knowledge.
    - Do you notice significant changes in the output when you include the scores from the nearest-neighbour search?

2.2 Part 2: Extensions

You are provided with a second dataset, this time containing Wikipedia articles and search queries produced by Frej et al. (2020) using the WikIR toolkit. It is about 1/2 GiB in size. We provide you with the code[4] that downloads the data and loads the documents and queries as Python objects.

Relative to cooking recipes, Wikipedia articles have a much richer word vocabulary – partly because they are much longer, partly because of jargon and proper names. Hence, word-level IR may fall short.

1. Neural document embeddings:
    - Use a pre-trained sentence transformer to generate document embeddings. Illustrate, using one query and one document from the dataset, how your chosen model handles cases where a word that isn't part of its vocabulary.
    - Recompute the retrieval metrics of part 1 using this model for both datasets. Additionally, define and compute *normalised discounted cumulative gain (nDCG)* for this dataset.
2. Compression:
    - Working with longer documents poses different challenges for both the IR subsystem and the LM subsystem. Why?
    - Choose one of the two subsystems, and implement one way of ameliorating the challenges it faces due to long documents.

[3] Note that you *cannot* use the K_j in the dataset, since you don't have this for queries not in $\mathcal{Q}$.
[4] https://colab.research.google.com/drive/1utiZidQdC0G7J2ferTLqpTCuUxESFwA_?usp=sharing

3. Security:
    - Can a user trick your LM by injecting their own prompt? If yes, try to make your LM subsystem more secure.

To finish, share with us 3 things you learnt from this project.

3 Format

In contrast to prior editions of the project, the main report will *no longer be written* due to past cases of AI-driven plagiarism. Instead, you will defend your work in an oral presentation with questions.

Structure If you struggle coming up with a structure for your presentation yourself, you can lay it out like a typical paper: you introduce your system and highlight which parts you're most proud of, then you lay out which experiments you want to do, and then show the results of those experiments.

Content Regardless of the structure, your presentation should answer the above questions. As for your slides: preferably, they don't contain too many words and mainly contain figures and code which you refer to when speaking. The slides are *not* supposed to be self-contained, so they are *not* supposed to be like lecture slides: as a rule of thumb, if reading your slides at home is equivalent to listening to you while you present them, you're doing it wrong.

Timing The presentation for both parts combined has a **hard time limit of 20 minutes**. You can cut it short if you still answer all the above questions. You will be cut off if you exceed this limit. Please practice your timing beforehand and build some margin into it.

4 Deliverables

What we expect to receive from you:

1. 1 PDF with the **slides** of your presentation named `IRSE-slides-r0123456.pdf`. *Note:* just like in a report, you should provide a citation for every non-trivial claim you make and for any external resource (dataset/model) you use.
2. 1 PDF that serves as a **handout** for things that are too unwieldy to be shown on a slide (e.g. uncompressed tables, large diagrams ...), named `IRSE-handout-r0123456.pdf`. During your presentation, you can assume that the teaching team has this in front of them.
    - *Note:* You are required to use the LaTeX template on Toledo.
    - *Note:* You are required to **track how much time you spend on programming, and how much you spend on everything else**. The template has fields to fill these in. These metrics will help us optimise the project for future students.
3. 1 ZIP of all Python **code** you implemented, organised in `.py` files, named `IRSE-code-r0123456.zip`. *Jupyter notebooks will not be accepted.* Make sure that it is clear what should be run to reproduce all the results found in your report (e.g., one central file `main.py` that imports functions from other files). We encourage you to annotate calls that take a long time with a time estimate.

5 Deadlines

We expect all your deliverables by **May 19th at 6 AM**. That means: *the latest regular submission is at 05:59:59 in the morning.* Toledo submissions past the deadline are possible, but when you are x minutes late, $\frac{x}{3.6}\%$ of the total possible grade will be subtracted from your score.

Beware that there is *no* resit for the project in August; whatever project grade you receive in June will be kept even if you retake the exam. This is in line with the course's ECTS card.

References

Frej, Jibril, Schwab, Didier, and Chevallet, Jean-Pierre (May 2020). "WIKIR: A Python Toolkit for Building a Large-scale Wikipedia-based English Information Retrieval Dataset". In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Ed. by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Marseille, France: European Language Resources Association, pp. 1926–1933. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.237/ (visited on 2025-04-25).

Jiang, Albert Q., Sablayrolles, Alexandre, Mensch, Arthur, Bamford, Chris, Chaplot, Devendra Singh, Casas, Diego de las, Bressand, Florian, Lengyel, Gianna, Lample, Guillaume, Saulnier, Lucile, Lavaud, Lélio Renard, Lachaux, Marie-Anne, Stock, Pierre, Scao, Teven Le, Lavril, Thibaut, Wang, Thomas, Lacroix, Timothée, and Sayed, William El (Oct. 2023). *Mistral 7B*. arXiv:2310.06825 [cs]. DOI: 10.48550/arXiv.2310.06825. URL: http://arxiv.org/abs/2310.06825 (visited on 2024-02-20).

Lewis, Patrick, Perez, Ethan, Piktus, Aleksandra, Petroni, Fabio, Karpukhin, Vladimir, Goyal, Naman, Küttler, Heinrich, Lewis, Mike, Yih, Wen-tau, Rocktäschel, Tim, Riedel, Sebastian, and Kiela, Douwe (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 9459–9474. URL: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html (visited on 2024-02-24).

Majumder, Bodhisattwa Prasad, Li, Shuyang, Ni, Jianmo, and McAuley, Julian (Nov. 2019). "Generating Personalized Recipes from Historical User Preferences". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Hong Kong, China: Association for Computational Linguistics, pp. 5976–5982. DOI: 10.18653/v1/D19-1613. URL: https://aclanthology.org/D19-1613.

Ouyang, Long, Wu, Jeff, Jiang, Xu, Almeida, Diogo, Wainwright, Carroll L., Mishkin, Pamela, Zhang, Chong, Agarwal, Sandhini, Slama, Katarina, Ray, Alex, Schulman, John, Hilton, Jacob, Kelton, Fraser, Miller, Luke, Simens, Maddie, Askell, Amanda, Welinder, Peter, Christiano, Paul, Leike, Jan, and Lowe, Ryan (Mar. 2022). *Training language models to follow instructions with human feedback*. arXiv:2203.02155 [cs]. DOI: 10.48550/arXiv.2203.02155. URL: https://cdn.openai.com/papers/Training_language_models_to_follow_instructions_with_human_feedback.pdf (visited on 2023-02-04).

Touvron, Hugo, Lavril, Thibaut, Izacard, Gautier, Martinet, Xavier, Lachaux, Marie-Anne, Lacroix, Timothée, Rozière, Baptiste, Goyal, Naman, Hambro, Eric, Azhar, Faisal, Rodriguez, Aurelien, Joulin, Armand, Grave, Edouard, and Lample, Guillaume (Feb. 2023). *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971 [cs]. DOI: 10.48550/arXiv.2302.13971. URL: http://arxiv.org/abs/2302.13971 (visited on 2024-02-24).

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, and Polosukhin, Illia (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. URL: https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (visited on 2022-10-29).