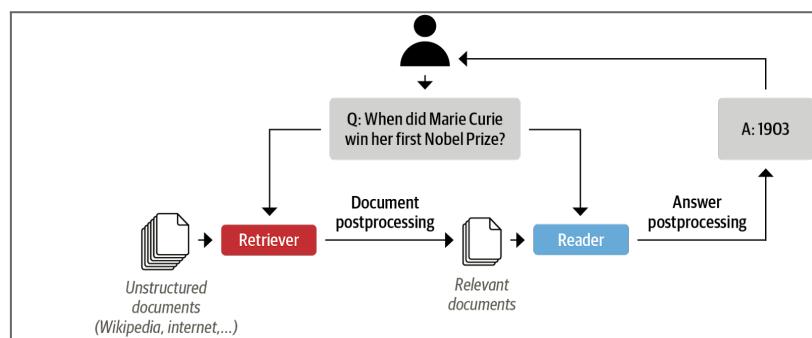


TriviaQA: Open-domain Question Answering

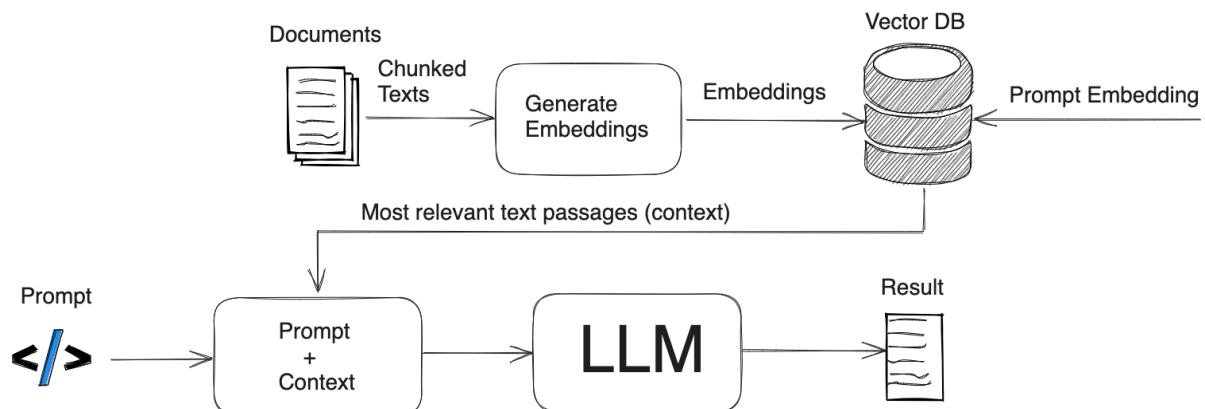
1. Description

Open-domain question answering (Voorhees and Tice, 2000) is a longstanding task in natural language processing, in which the task involves answering factoid questions from a large knowledge corpus, such as Wikipedia (Wikipedia, 2004) or BookCorpus (Zhu et al., 2015), among others.



Recently, Open-domain Question Answering (ODQA) systems have attracted considerable attention due to the great advances in LLMs. The traditional framework of the ODQA system is the Retriever-Reader (Chen et al., 2017) built by combining two modules: 1) an information retriever (IR) and 2) a reader – see the figure above. The task of the **retriever** is to retrieve evidence pieces from a large corpus of documents, using IR methods such as TFIDF (Chen et al., 2017), BM25 (Mao et al., 2021), and DPR (Karpukhin et al., 2020). The goal of the **reader** is to understand and reason the retrieved evidence to yield the answer.

Nowadays, ODQA systems have been substituted by the well-known Retrieval Augmented Generation (RAG) based systems. The figure below shows a similar architecture traditionally used in ODPQA, in which a **retrieval** component is combined with a **generator** component.



The **retriever** Component searches for relevant information and evidence across different databases, documents, and knowledge sources. It is responsible for the initial step of

retrieving relevant information from external knowledge sources. It uses retrieval techniques such as keyword-based search, document retrieval, or structured database queries to fetch pertinent data. The retriever can employ pre-built indexes (vector DBs), search algorithms, or APIs to access various knowledge sources, including databases, documents, websites, and more. Its primary goal is to compile a set of contextually relevant information that can be used to enrich the user's query.

The **generator** component takes the retrieved information, along with the user's original query, and generates the final response or output. Nowadays, the generator employs generative LLMs to generate human-like text that is contextually relevant, coherent, and informative. The generator ensures that the response aligns with the user's query and incorporates the factual knowledge retrieved from external sources.

There is an option to use a **re-ranker** that refines the retrieved information by assessing its relevance and importance. Rankers use various algorithms, such as text similarity metrics, context-aware ranking models, or machine learning techniques, to evaluate the quality of the retrieved content.

2. Objectives

The project aims to build and evaluate an open-domain QA system based on RAG. Specifically, the goal is to build the **retriever** and **generator** of the RAG system, that is, given a query and a set of documents related to the query, the system should retrieve the passage that potentially contains the answer and provide the final answer.

The proposed task will provide the opportunity to explore alternative retrieval approaches that retrieve evidence pieces from a large knowledge corpus. Note that a great variety of Retrieval Augmented Generation (RAG) can be implemented in this scenario. You can check more on RAG in (Gao et al. 2023; Fan et al. 2024).

We propose to use TriviaQA¹ (Joshi et al., 2017) as a benchmark to evaluate your systems. TriviaQA provides over 650K **question-answer-evidence** triples as a reading comprehension dataset. It also includes 95K question-answer pairs authored by trivia enthusiasts and independently gathered evidence documents, six per question on average, that provide high-quality distant supervision for answering the questions.

The figure below shows two question-answer pairs with sample excerpts as evidence documents. Note that the task can be cast as a reading comprehension task² (RC), where a *Question* and an *Excerpt* are given and the *Answer* needs to be produced. Evidence documents are given in the dataset, but you need to be aware that the document might be too long (e.g. Wikipedia entry) to use as an input excerpt. So you would need to think of different alternatives to conveniently retrieve the correct pieces of evidence.

¹ <https://nlp.cs.washington.edu/triviaqa/>

² The task consists in giving a query (q) and a text that potentially contains the answers (also known as passage – p), the model returns the answer (i.e. (p, q) → a)

For this dataset, evidence of each question-answer pair is collected from two sources: Wikipedia and web searches. Although the dataset provides cases where the question-answer pair only contains evidence documents from one source.

Question: The Dodecanese Campaign of WWII that was an attempt by the Allied forces to capture islands in the Aegean Sea was the inspiration for which acclaimed 1961 commando film?

Answer: The Guns of Navarone

Excerpt: The Dodecanese Campaign of World War II was an attempt by Allied forces to capture the Italian-held Dodecanese islands in the Aegean Sea following the surrender of Italy in September 1943, and use them as bases against the German-controlled Balkans. The failed campaign, and in particular the Battle of Leros, inspired the 1957 novel **The Guns of Navarone** and the successful 1961 movie of the same name.

Question: American Callan Pinckney's eponymously named system became a best-selling (1980s-2000s) book/video franchise in what genre?

Answer: Fitness

Excerpt: Callan Pinckney was an American fitness professional. She achieved unprecedented success with her Callanetics exercises. Her 9 books all became international best-sellers and the video series that followed went on to sell over 6 million copies. Pinckney's first video release "Callanetics: 10 Years Younger In 10 Hours" outsold every other **fitness** video in the US.

The dataset is organised in a way where experiments can be run separately using Wikipedia triples only or web-search triples only. You can find further details about the dataset in the paper (Joshi et al. 2017) and Section 5 ([Materials](#)).

As stated above, evidence documents can be very long (e.g., a whole Wikipedia article). At the same time, the dataset at hand can be considered quite large, so you will need to think about different strategies to prepare the training and evaluation sets in advance, and how you need to run the inference when testing.

To-Dos (expected minimum amount of work). Students have to complete the following tasks:

- Understand the task and dataset at hand. Section 5 contains a short description of the dataset. We highly recommend reading the paper that describes the dataset: <https://aclanthology.org/P17-1147/>
- Create training and validation partitions from the training (`wikipedia-train.json`), and use the original validation file as the test (`verified-wikipedia-test.json`). To create a validation set, you **MUST take the first 7900 questions** (and associated documents) **from the training files**.
- Implement the **retrieval** module. Retrieval methods can be generally categorised into **sparse** (TF-IDF, BM25) and **dense** (vector stores, which store documents as embeddings for semantic similarity). Exploration of different retrieval techniques will be considered for the final grade.
- Implementation of the **generator**. Typically, generative LLMs are used to generate the answer. Evaluate different generative models in this setting (take a look at <https://chat.lmsys.org/>). In case you are sort of GPUs, one option is to use quantized

models that are run (only for inference, no training) on CPUs (more information in Section 5).

- Evaluate the models using the scorer provided by the authors of the TriviaQA dataset. <https://github.com/mandarjoshi90/triviaqa>

Extra To-Dos. Alternatively, students can also conduct experiments with evidence coming from web searches. As you did for Wikipedia triples, you would need to create a validation set from the training data and use the original validation for testing. In this case, you MUST take the first 9500 examples from the training set to create the new validation set.

We encourage you to explore different approaches to RAG-based ODQA. A good starting point is to check the advances described in the following surveys on RAG^{3,4}, as well as papers reported in HF's trending⁵.

Finally, in this project, students have the opportunity to apply many of the recent advances that occurred in NLP, so take the opportunity to explore:

- Different pre-trained LLMs architectures. Some foundational models require a huge amount of computational resources, so take this into account when defining your approach. Approaches that require less computation might also be interesting to apply (e.g., Quantization and Parameter Efficient Fine Tuning like LoRA and QLoRA).
- Different approaches to retrieval models (BM25, DPR, etc.). This might be very costly in terms of GPUs and disks.
- Exploration of Retrieval Augmented Generation (RAG) to include additional information in the prompt to guide the generative models toward the correct answer.
- Exploration of rankers that sort the retrievers' output.
- Different fine-tuning techniques to train retrievers, rerankers, and modules.

3. Submission

Report and Code

Students need to write 4-6 pages describing their open QA system. The document to be presented must somehow include the typical sections of a research article:

1. **Introduction:** The context of the problem to be solved will be presented.
2. **Methodology:** The technical aspect, that is, the approach that was followed to carry out the project, will be clearly described. You should collect important details about how the experimentation and analysis were designed.
3. **Presentation of the results.** In this section, the results obtained are presented. Result tables must be included correctly, using the specified evaluation metrics. It is also recommended to perform a **qualitative analysis**. The assessment and discussion of the results will be positively considered.
4. **Conclusions.** The conclusions of the work will be presented very briefly.

³ <https://arxiv.org/abs/2405.06211>

⁴ <https://arxiv.org/abs/2312.10997>

⁵ <https://huggingface.co/papers/trending>

The documentation format will be that of ACL. You can obtain the format templates for MSword, Latex, and Overleaf at the following link:
https://2023.aclweb.org/calls/style_and_formatting/

Along with the documentation, the student must submit the code developed during the project. The code will be presented on notebooks such as Colaboratory or Kaggle, or on Github. **IMPORTANT:** The version history must accurately reflect the project's development. Projects with a version history consisting **of a single code upload will not be corrected and will be considered failed.**

Important dates

- 08/10/2025 Make groups of three and let the lecturers know
- 02/12/2025 - 03/12/2025 Project presentations
- 12/12/2025 Submission of code and report

4. Evaluation criteria

We have defined the following criteria to evaluate the project.

- **Technical part (60%)** We will consider the originality and complexity of the technical part. If the minimum amount of work is done, without any exploration, 60% of the technical part will be obtained. Exploration of different approaches and evaluation will give extra credit. If the student needs any guidance on the exploration, please do not hesitate to contact us.
- **Documentation (30%)** The clarity, structure, and discussion carried out in the document will be considered.
- **Presentation (10%)** The clarity, structure, and discussion carried out in the presentation will be considered.

5. Materials

Description of the TriviaQA dataset

The dataset can be downloaded from: <https://nlp.cs.washington.edu/triviaqa/> (RC version 1.0). The compressed file is 2.6GB, and the decompressed file needs 7.6GB on your hard disk. The list of data files contained in the dataset is the following (the following description is copied from the dataset's readme):

LIST OF DATA FILES

- qa/wikipedia-train.json, qa/web-train.json
- qa/[verified-]wikipedia-dev.json, qa/[verified-]web-dev.json

These files contain [verified] questions, answers, and document names in the train/dev set for the wikipedia/web domain. The details of the verified evaluation set are described in section 4 (evidence analysis) of the paper. The wikipedia/web documents are listed in the json array "EntityPages"/"SearchResults" for each question. The "Filename" field in each element of the array indicates the relative path of the file inside the wikipedia/web directory in evidence.

- qa/wikipedia-test-without-answers.json, qa/web-test-without-answers.json

These files contain questions and document names (no answers) in the train/dev set for the wikipedia/web domain. We are withholding the test answers for a later release. Please check the website -- <http://nlp.cs.washington.edu/triviaqa/> -- for details of the test evaluation.

- evidence/web, evidence/wikipedia

These directories contain documents in the train/dev/test set for the wikipedia/web domain. The documents are referenced in the json array "EntityPages"/"SearchResults" for each question in the QA pair files. The "Filename" field in each element of the array indicates the relative path of the file inside the wikipedia/web directory.

The following commands might be helpful to download the dataset and start working with it:

```
$> wget https://nlp.cs.washington.edu/triviaqa/data/triviaqa-rc.tar.gz
$> mkdir /content/triviaqa-rc
$> tar -xzvf "/content/triviaqa-rc.tar.gz" -C "/content/triviaqa-rc"
$> rm "/content/triviaqa-rc.tar.gz"
```

Dataset loaders. Another option is to use existing data loaders of dataset:

- 😊 [huggingface/datasets](#) Easiest way to download and start working with dataset (see next section).
- ⚡ [tensorflow/datasets](#) Good documentation to understand the structure of the dataset

TriviaQA on HuggingFace Datasets

Datasets (<https://huggingface.co/docs/datasets/>) is a library that helps access many datasets including those commonly used in NLP. We strongly recommend using this option as it facilitates integration with HuggingFace code.

To inspect what is contained in the dataset you may `get_dataset_config_names()` function. For example, you can inspect the TriviaQA in datasets:

```
from datasets import get_dataset_config_names

domains = get_dataset_config_names('trivia_qa')
domains

['rc',
 'rc.nocontext',
 'unfiltered',
 'unfiltered.nocontext',
 'rc.web',
```

```
'rc.web.nocontext',
'rc.wikipedia',
'rc.wikipedia.nocontext']
```

'rc' partitions contain the whole dataset. Note that downloading and processing things that you don't need might be expensive.

'rc.wikipedia' and 'rc.web' are the triples that contain evidence from Wikipedia and web searches, respectively. We encourage you to use these partitions at the beginning.

Data loading can be done with `load_dataset()`. For example, to download the train, validation, and test partitions, you can run the following code:

```
from datasets import load_dataset

%time trivia_qa_wikipedia = load_dataset('trivia_qa',
name="rc.wikipedia")
```

Downloading and preparing dataset trivia_qa/rc.wikipedia (download: 2.48 GiB, generated: 3.89 GiB, post-processed: Unknown size, total: 6.37 GiB) to /root/.cache/huggingface/datasets/trivia_qa/rc.wikipedia/1.2.0/e73c5e47a8704744fa9ded33504b35a6c098661813d1c2a09892eb9b9e9d59ae...

Downloading data files: 100%
1/1 [07:24<00:00, 444.38s/it]
Downloading data: 100%
2.67G/2.67G [07:24<00:00, 1.32MB/s]
Extracting data files: 100%
1/1 [03:15<00:00, 195.80s/it]
Dataset trivia_qa downloaded and prepared to /root/.cache/huggingface/datasets/trivia_qa/rc.wikipedia/1.2.0/e73c5e47a8704744fa9ded33504b35a6c098661813d1c2a09892eb9b9e9d59ae. Subsequent calls will reuse this data.

100%
3/3 [00:00<00:00, 102.24it/s]
CPU times: user 4min 20s, sys: 1min 31s, total: 5min 51s
Wall time: 12min 12s

As you can see, the downloaded dataset is approximately 6.3GB large and can take about some minutes to download. Once you download it, you can start understanding how the dataset is organized.

Fine-tuned models

There are many transformer QA models fine-tuned in TriviaQA:

https://huggingface.co/models?pipeline_tag=question-answering&sort=trending&search=triviaqa

Parameter-efficient models

Hugging Face provides several ways to quantize a model. This family of methods allows representing the data and the model with fewer bits, making it a useful technique for reducing memory usage when it comes to large language models (LLMs). Hugging Face's `bitsandbytes` can be a good choice to explore:

<https://huggingface.co/docs/bitsandbytes>

In addition, when training the model, quantization is not sufficient. For these cases, a combination of quantization with PEFT can be a good strategy for training even the largest models on a single GPU. We recommend checking the different options given by Hugging Face:

<https://huggingface.co/docs/peft/>

Related to efficient training of LLMs, using unsloth  can be a good choice. It provides a great variety of PEFT techniques for fast and efficient training, as well as a good amount of quantization methods. It can be easily integrated with Hugging Face models. Check the framework here:

<https://unsloth.ai/>

Tools and libraries for RAG

Take a look at [Langchain](#), [Pinecone](#), and [LlamaIndex](#), which provide modules to streamline the process of retrieval augmentation, making it more efficient and user-friendly.

Accelerated inference

vLLM is an open-source library designed for fast and efficient LLM inference and serving. It provides a memory-efficient attention mechanism that avoids GPU memory fragmentation, and supports continuous batching, reducing latency when handling multiple requests for high throughput generation. Interestingly, it integrates with popular models from Hugging Face, enabling text generation with just a few lines of code:

<https://docs.vllm.ai/>

Run your LLMs locally

Finally, Ollama  offers a user-friendly interface for running large language models (LLMs) locally, particularly on MacOS, Linux, and Windows platforms. This powerful tool provides immense value to researchers, developers, and anyone seeking to explore the capabilities of language models. Ollama supports a diverse range of models. Check here:

<https://ollama.com>

Tutorials and examples

- Hugging Face tutorials:
 - QA tutorials:
 - <https://huggingface.co/tasks/question-answering>
 - https://huggingface.co/docs/transformers/tasks/question_answering
 - RAG tutorial:
 - Simple RAG:
https://huggingface.co/learn/cookbook/en/rag_zephyr_langchain
 - Advanced RAG:
https://huggingface.co/learn/cookbook/en/advanced_rag
 - More LLM recipes in HF: <https://huggingface.co/learn/cookbook>

Compute

You will need to run the experiments in Colab or Kaggle, or any hosted service that provides some GPUs. For example, students should be able to fine-tune a small quantized Qwen on TriviaQA. As the dataset is very large, efficient checkpointing is going to be important.

- Colab: <https://colab.research.google.com/>
- Kaggle: <https://www.kaggle.com/>
- Paperspace: <https://www.paperspace.com/>
- Amazon SageMaker Studio: <https://aws.amazon.com/es/pm/sagemaker/>
- Lightning AI Studio: <https://lightning.ai/>

References

TriviaQA - Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

T5 - Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. [How Much Knowledge Can You Pack Into the Parameters of a Language Model?](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online. Association for Computational Linguistics.

BART- Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. [BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

DPR - Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. [Dense Passage Retrieval for Open-Domain](#)

Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Fusion-in-decoder: Gautier Izacard and Edouard Grave. 2021. [Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

RAG - Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, Lewis M, Yih WT, Rocktäschel T, Riedel S. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*. 2020;33:9459-74.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. [Reading Wikipedia to Answer Open-Domain Questions](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Association for Computational Linguistics (ACL)*, pages 6086–6096.

Qin Zhang, Shangsi Chen, Dongkuan Xu, Qingqing Cao, Xiaojun Chen, Trevor Cohn, and Meng Fang. 2023. [A Survey for Efficient Open Domain Question Answering](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14447–14465, Toronto, Canada. Association for Computational Linguistics.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, Hannaneh Hajishirzi. 2017. [Bidirectional Attention Flow for Machine Comprehension](#). *International Conference on Learning Representations (ICRL)*

Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. [Retrieval-augmented generation for large language models: A survey](#). arXiv preprint arXiv:2312.10997.

Fan, W., Ding, Y., Ning, L., Wang, S., Li, H., Yin, D., Chua, T.S. and Li, Q., 2024, August. [A survey on rag meeting llms: Towards retrieval-augmented large language models](#). In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 6491-6501).