Assignment 3: Building your own RAG system

Release date: 03/03 (Tue)
Early milestone due date: 03/17 (Tue) 05:59PM **(No late days can be applied)**
Due date: 03/19 (Thu) 05:59PM
Latest possible due date with late days: 03/22 (Sun) 05:59PM

Overview

Large language models (LLMs) are incredible in answering many questions about the world, but they struggle with questions from niche topics. A common technique here is to augment LLMs' knowledge with documents that are relevant to the question. In this assignment, you will develop a retrieval augmented generation (RAG) model that is capable of answering questions about UC Berkeley EECS — any questions that you can find answers from a page under eecs.berkeley.edu.

So far in your machine learning classes, you may have experimented with standardized tasks and datasets that were easily accessible. However, in the real world, NLP practitioners often have to solve a problem from scratch (like this one!). This includes gathering and cleaning data, annotating your data, choosing a model, iterating on the model, and possibly going back to change your data. In this assignment, you'll get to experience this full process.

Please note that you'll be building your own system end-to-end for this assignment, and there is no starter code. You must collect your own data and develop a model of your choice on the data. We will be releasing the inputs for the test set a few days before the assignment deadline, and you will run your already-constructed system over this data and submit the results. We also ask you to follow several experimental best practices, and describe the result in your report.

The key checkpoints for this assignments are

1. Understand the task specification and annotate the evaluation data.
2. Prepare your raw data.
3. Develop a retrieval augmented generation model.
4. Generate results, run ablation studies, prepare a written report (template provided at the end of this guideline), and submit your work.

Summary of your deliverables

Between the release date and the early milestone due date:

- You will build a RAG model, based on y**our own curated retrieval corpus** and your **own validation data**. We will only provide 10 example question-answer pairs.
- Your submission should take questions as inputs and produce answers. We will evaluate your model on a **hidden development set** (100 questions), with questions

and answers annotated as of **03/03** (release date).
- **Submission limit: 5**
- Late days: Not allowed for the early milestone.

Between the early milestone due date and the final due date:
- On **03/17 7:00PM** (1 hour after the early milestone due date), we will release the hidden development set and the reference retrieval corpus (text corpus). You may use these resources to further develop your final RAG model.
- We will evaluate your final submission on a **hidden test set** (100 questions), with questions and answers annotated as of **03/17**.
- You will also submit a **written report** based on the pre-defined questionnaire (see the questions at the end of this guideline). We recommend starting your ablations and drafting the report before the early milestone.
- **Submission limit: 5**
- Late days: Allowed (up to 3 days).

Grading (This is a brief overview; please refer to the Grading section for full details.)
- 20% based on your model performance on the hidden dev set at the early milestone due date
- 40% based on your model performance on the hidden test set at the final due date
- 40% based on your written report
- This assignment also has extra credits – see the Grading section

Understand the task specification and annotate the evaluation data

You will be working on the task of factoid question answering (QA). Your model should be able to answer a question whose answer can be *extracted* from a paragraph from a web page under eecs.berkeley.edu, reachable from the main page.

Recommended QA Data Scope – The hidden dev and test set will follow the same guidelines.

- English-only
- Answers must be found on the EECS website, matching the regex: `https?:\/\/(?:www\d*\.)?eecs\.berkeley\.edu(?:\/[^\s]*)?`
    - Pages that require login / credentials are not included.
- Answers must be obtainable from plain text or tables only (no images, visuals, or PDFs).
- Each question must be answerable using information from a single page (no multi-page aggregation). Questions may require combining information from multiple passages as long as they are on the same page.
- Answers should be short text spans (e.g., named entities, numbers, dates), under 10 words.
- Most questions should be extractive, meaning the answer appears directly on the evidence page.
- We may include:

- (1) Yes/No questions (answers must be Yes or No) – up to 5%
- (2) Abstractive questions (non-extractive but still short answers, e.g., those requiring counting or simple arithmetic) – up to 5%

Other considerations

- For questions with multiple valid answers, you can include multiple reference answers per line in reference_answers.json (separated by a vertical bar "|").
- During official evaluation, we will apply string normalization (see this function – it lowercases everything and removes punctuations) and check whether your prediction exactly matches any of the valid answers. If so, it will be considered correct.

You will first create the evaluation data to test your model, by navigating the EECS website yourself. This will be the data that is critical to test, understand, and analyze your model.

In order to do so, you will annotate questions and their answers. You will consider the following:

- ☐ Domain relevance: Your test data should be similar to the data that your model will finally be tested on (questions about EECS).
- ☐ Diversity: Your test data should cover a wide range of questions about EECS.
- ☐ Size: Your test data should be large enough to distinguish between good and bad models. We recommend having at least 100 questions in your test data.
- ☐ Quality: An important component of every data annotation effort is to estimate its quality. A standard approach is to measure inter-annotator agreement (IAA). To measure this, two members of your team should annotate a random subset of your test set and compute IAA on this subset.

To help you get started, here are some example questions:

- Questions that could be answered by just prompting an LLM.
  - "Which university did Dan Klein receive his Ph.D.?"
- Questions that could be better answered by augmenting LLM with relevant documents.
  - "Which year did Dan Klein receive the Grace Murray Hopper Award?"
  - "How many GSI hours do Berkeley EECS students need to obtain a doctoral degree?"
- Questions that are likely answered only through augmentation.
  - "What is the office number of Dan Klein?"
  - "Which email address should CS PhD students send their Ph.D. Student Review to?"
- Questions that are sensitive to temporal signals
  - "What is the title of the dissertation of Dan Klein's most recent Ph.D. graduate?"
  - "Who is the winner of the Eugene L. Lawler Prize in 2024-25?"

Ensuring Data Quality

An important component of every data annotation effort is to estimate its quality. A standard approach is to measure inter-annotator agreement (IAA). To measure this, at least two members of your team should annotate a random subset of your test set. Compute IAA on this subset and report your findings. You should annotate at least 30% of your test set to measure IAA.

Prepare your raw data

To build a RAG model, you will first need to compile a retrieval corpus – a knowledge resource of relevant documents. You will crawl the EECS websites and get HTML pages (please feel free to ignore files, such as PDFs and JPGs).

You will need to clean this data and convert it into a file format that suits your model development. Here are some tools that you could consider using.

- Beautifulsoup4
- resiliparse
- Prompting a language model to clean up the HTML

By the end of this step, you will have a collection of documents that will serve as the knowledge resource for your RAG system.

Tip: The quality of the raw data you prepare will significantly impact your RAG model's performance. We strongly recommend investing sufficient time in curating and validating your retrieval corpus.

Develop a retrieval augmented model

You will now build a RAG model, using the following three components

- A datastore
- Document & query embedder
- A retrieval index
- Generator (a.k.a. LLM)

You may use external libraries such as Hugging Face, vLLM, SGLang, BM25, and FAISS. If you're unsure whether a specific library is allowed, please check with the instructor.

A few key questions you may ask:

- What should be the retrieval unit? Document? Passage? A chunk of text (if then, how many words should each chunk have)?
- Should I use sparse retrieval (BM25) or dense retrieval? Or hybrid?
- If I use dense retrieval, which embedding model should I use? Which FAISS index should I use?
    - **Note that the embedding model should be 400M params or less.**
- How many documents/passages should I retrieve?
- How should an LLM take these documents/passages and the original question and get the answer to the question?
- Which LLM should I use?

    - To iterate more quickly on model choices, we recommend starting with a model that has 8B parameters or fewer.
    - **Note : Testing will be done using openrouter models. Only the following models are allowed:**
        - `"meta-llama/llama-3.1-8b-instruct",`
        - `"meta-llama/llama-3-8b-instruct",`
        - `"qwen/qwen3-8b",`
        - `"qwen/qwen-2.5-7b-instruct",`
        - `"allenai/olmo-3-7b-instruct",`
        - `"mistralai/mistral-7b-instruct"`
-

Tip: Be aware of the resource constraints described here and design your RAG model accordingly. In particular, the model should return the answer to each question within 0.6 seconds on average.

Generate results and perform ablations

You will test your model on the test data you created. You will use “Exact Match” and “F1” as evaluation metrics (script). Please note that you should keep your system generated responses as concise as possible, as the hidden dev and test sets include short answers only. You can use the provided reference set as a reference. You can download it from https://drive.google.com/drive/folders/1Aad_kpZzfWX7K7iJcQe3o-G_J3WNlakg?usp=sharing.

Use these metrics to perform ablations (e.g., comparing between several different design choices of the model) and decide on your final RAG model.

Tip: In addition to Exact Match and F1 on the predicted answers, also compute “recall” – i.e., whether any of the retrieved text includes ground-truth URL (if annotated) or contains the answer string. Looking at this number together with Exact Match/F1 helps you diagnose whether the remaining room for improvement lies in the retrieval model or the generator.

[Optional] Further improve your model

If you complete the steps above, including comprehensive ablations, you’re likely to reach the score that receives full points. Optionally, you're welcome to explore additional methods to further improve your model's performance. Your approach is flexible, as long as your model does not access the internet at runtime (i.e., using APIs or web services is not allowed during execution, though offline use, such as during preprocessing, is fine).

As noted in the Grading section, extra credit will be awarded as follows (capped at 20%)

- 10% if your team ranks among the top 3 on either the hidden development set or the hidden test set (top 3 = approximately top 5%)

- 10% to 0–2 teams with the most creative RAG solutions
- 10% to 0–1 team with the most creative QA datasets beyond the official scope

Example ideas:

- Use a language model to rewrite documents so they are easier for the generator LLM to process and understand.
- Use an LLM to create a large training set of (question, answer) pairs, and fine-tune your RAG model on this date.
- Use an LLM to create a large training set of (question, answer) pairs, and train an LLM (without retrieval) to memorize all possible test-time questions.
- Use an LLM to create a large training set of (question, answer) pairs, and build a retrieval model that finds the closest question at test time and copies the corresponding answer. (See this paper for reference.)
- Use a long-context LLM, without retrieval. (See this paper for reference.)
- Use a larger LLM, and apply techniques to compress its weights or speed up inference.

RAG Model Submission Guidelines

Your final submission should include (1) the RAG model – the code and the model artifact, including your retrieval datastore, and (2) the QA data you created.

DEPENDENCIES : Additionally, your code needs to work with a fixed set of requirements files which are limited to the following libraries. The environment will use python 3.10.12. The autograder environment will not contain any other libraries. Please copy and paste the exact dependenies below to create your local conda environment.

```
torch>=2.0.0
transformers>=4.30.0
sentence-transformers>=2.2.2
faiss-cpu>=1.7.4
rank-bm25>=0.2.2
numpy>=1.24.0
tqdm>=4.64.0
matplotlib>=3.5.0
pandas>=1.5.0
seaborn>=0.11.0
```

SUBMISSION DETAILS:

1. Submit a zip that contains your code and a required run.sh entrypoint. Each submission should have a bash file called run.sh which will be called by the autograder for evaluation.
2. **Your submission must include a file named run.sh (the autograder looks for this EXACT filename).**
3. The autograder will run your script exactly as:

```
bash run.sh <questions_txt_path> <predictions_out_path>
```

4. Your run.sh must accept exactly those two positional arguments:
    a. $1: path to input questions txt file
    b. $2: path where you must write predictions txt file

5. Your run.sh should use python3 (not python) to run your code.
6. Input format: questions txt file has one question per line.
7. Output format: write one predicted answer per line to the provided output path.
8. Number/order constraint: output must have the same number of lines as input, in the same order.
9. **We will provide you with another file called llm.py for all LLM API calls (OpenRouter call wrapper). Use this file exactly and do not modify this at all. We check for this and return 0 score if this is detected.**
10. Do not call OpenRouter directly from other files or try to make an api call from anywhere else in the codebase. We check for this return 0 score if this is detected.
11. Grading metric is token-level F1:
    a. Hidden Dev set contributes up to 20 points
    b. Hidden Test set contributes up to 40 points
12. Total is out of 60 points
13. Your code must finish within the autograder time limits; if it fails or times out, score may be 0 for that split. The time limit for evaluation is 30 minutes, although it should finish much earlier than that ideally. On your local machine, you should aim for 1 second per question latency, so the entire evaluation should finish locally within 2 minutes.
14. Keep paths in your code relative and robust; do not hardcode local machine-specific absolute paths.
15. The gradescope environment has a RAM of 4GB and does not have a GPU. Make sure your code works within these constraints.
16. **The embedding model needs to be 400MB or less.**
17. List of allowed models to use and experiments with:
    a. `"meta-llama/llama-3.1-8b-instruct",`
    b. `"meta-llama/llama-3-8b-instruct",`
    c. `"qwen/qwen3-8b",`
    d. `"qwen/qwen-2.5-7b-instruct",`
    e. `"allenai/olmo-3-7b-instruct",`
    f. `"mistralai/mistral-7b-instruct"`

Tips:

- We will have one question per line in `data/questions.txt`
- We will have one prediction per line in `data/answers.txt` – This means you should make sure that the prediction does not include any newline character within it.
- We will use "Exact Match" and "F1" as evaluation metrics (script).
- Make sure to add a timeout exception for each question. Sometime openrouter times out and it's better to have a placeholder answer when this happens instead of doing the entire evaluation again.
- Make sure to not touch the llm.py . It should be exactly the same. We will overwrite your llm.py file with what we provide and then run the autograder, which means if you made any modifications to the original file, that will not show up in the autograder run.
- The autograder could take a long time to evaluate submissions. Two files are evaluated on gradescope each with 100 questions (one for early deadline, one for late deadline).

You can estimate the time for your pipeline for 200 questions from your local runs, and maybe add 5-7 additional minutes for gradescope processing.

Written Report Guidelines

Each team submits one written report. There is no required template, but please use 11-point font (e.g., Times, Times New Roman, or Arial), 1-inch margins, and standard formatting. Figures and tables are permitted/encouraged.

The report is limited to **1.5 pages of main text**, excluding tables and figures. The total length, including tables and figures, must not exceed **2 pages**.

The report should include the response to each of the following questions.

Q1. QA Data Creation: Describe how you created your QA dataset. Report the statistics, including the data size and inter-annotator agreement (IAA). Include 3+ samples from your dataset.

Q2. Retrieval Corpus: Describe how you constructed your retrieval corpus and how you evaluated it (e.g., manual inspection or ablations within your RAG model). After receiving the reference retrieval corpus, conduct ablations comparing your corpus and the reference corpus within your RAG model. If the reference corpus performs better, speculate on how it may have been constructed.

Q3. RAG System: Describe your RAG system, including the overall architecture, pipeline, component design, and other design choices you considered.

Q4. Ablations: Select two ablations you conducted. Describe each and report results (tables or figures). While you likely performed more than two ablations, choose the two most interesting ones (e.g., the largest impact, most unexpected findings).

Q5. Error analysis: Select a random subset of incorrect predictions (i.e., cases with an F1 score of 0.0) from your best performing model, and perform an error analysis by categorizing the error types. In particular, some errors might be coming from limitations of the evaluation metric itself. Be sure to include “false negatives due to metric limitations” as one of the categories, and propose potential improvements to the metric.
Q6. Takeaways & Future Ideas: Include a brief paragraph summarizing your key takeaways from this assignment, and describe additional ideas you would explore with more time, such as alternative RAG approaches or additional ablations.

Please allocate space according to where you invested the most effort, and emphasize the sections that reflect your most substantial contributions. For example, if you spent significant time designing a creative QA dataset, you may devote more space to Q1. If you developed a RAG pipeline that substantially deviates from the baseline approach discussed in class, you may allocate more space to Q3.

Please also include a Contribution Statement, a GenAI statement (see the policy here), and references if applicable. These sections do not count toward the page limit. Appendices are not accepted.

Grading

RAG scores based on hidden dev set at the early milestone due date (20%)

- Full credit if your F1 score exceeds **50%**
- Scores between **10%** and **40% (F1 score)** will be linearly interpolated for 90% credit. Scores from **40%** to **50%** will linearly receive the remaining 10% credit. Scores below 10% will receive 0 credit.
- Runtime penalty: -20% of your score for every 30 seconds beyond the 1-minute limit. For instance, if your runtime is 80 seconds, you will get -20% of your score; if your runtime is 140 seconds, you will get -60% of your score.

RAG scores based on the hidden test set (40%)

- Full credit if your F1 score exceeds **60%**
- Scores between **20%** and **60% (F1 score)** will be linearly interpolated for 90% credit. Scores from **50%** to **60%** will linearly receive the remaining 10% credit. Scores below 20% will receive 0 credit.
- Runtime penalty: Same as above

Written report (40%)

Extra credits (capped to 20%)

- 10% if your team ranks among the top 3 on either the hidden development set or the hidden test set (top 3 = approximately top 5%)
- 10% to 0–2 teams with the most creative RAG solutions
    - Make sure you highlight this aspect in the written report!
- 10% to 0–1 team with the most creative QA datasets beyond the official scope
    - Make sure you highlight this aspect in the written report!

Acknowledgements

This assignment is heavily inspired by Carnegie Mellon University's CS11-711 Advanced NLP Assignment 2.