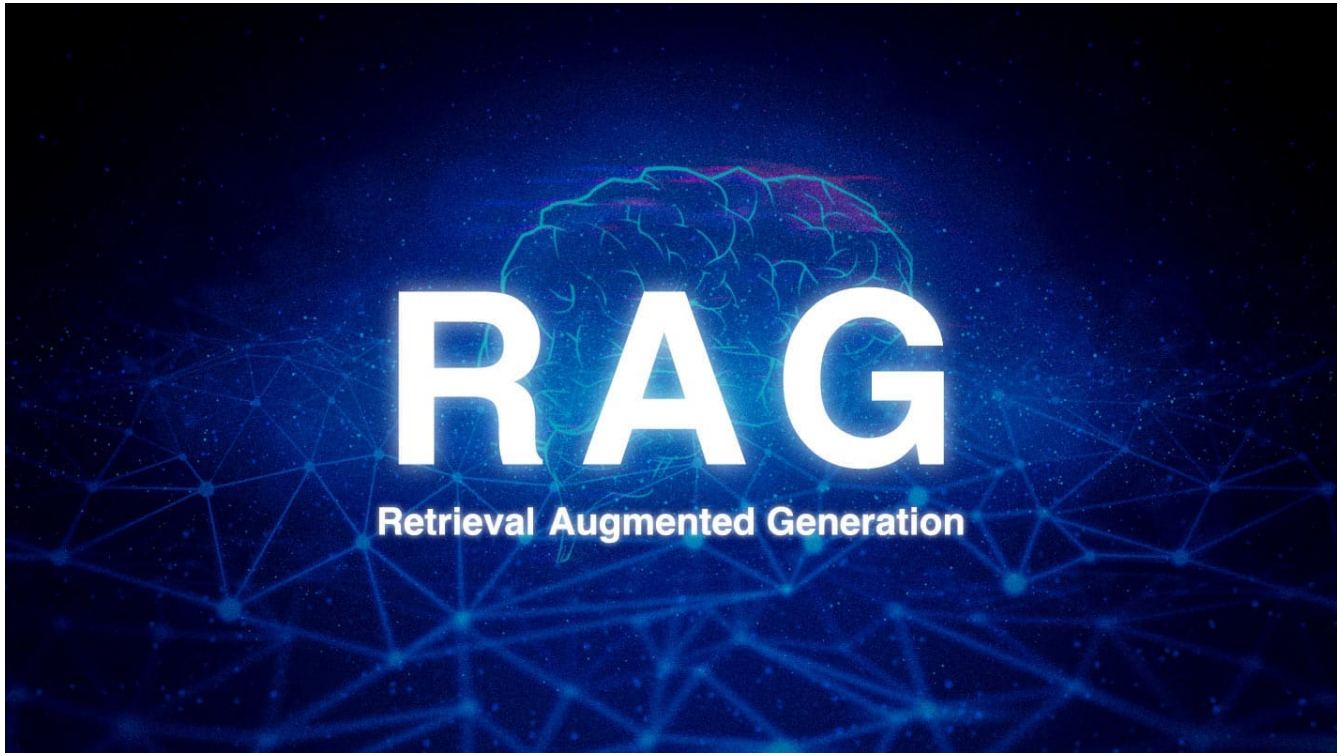
Natural Language Processing

Lecture : Retrieval Augmented Generation

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Introduction



©Medium

Introduction

LLMs have an enormous amount of knowledge encoded in their parameters. However, LLMs

- may lead to hallucination
- may not be up-to-date with their knowledge
- do not provide textual evidence to support their answer
- are unable to answer questions from proprietary data

Introduction

Retrieval-augmented generation (RAG) is a two-step process combining external knowledge search with prompting.

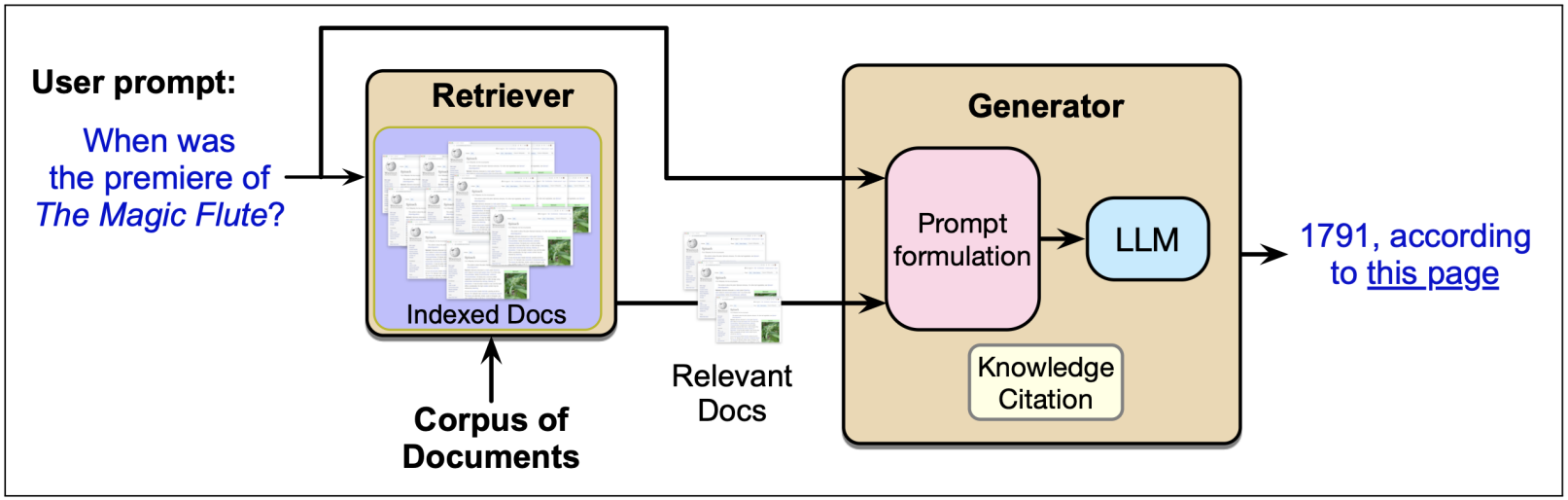
Retrieval: given a query, some information retrieval neural model fetches relevant documents/passages.

Based on document and query embedding, and a learnable similarity measure.

Generation: fetched documents are wrapped into a prompt, along with query, and passed to the LLM (also called the reader) to generate relevant response.

Introduction

The two RAG stages: retrieval and generation.



Neural information retrieval

©Medium

Neural information retrieval

Traditional information retrieval methods, such as tf-idf or BM-25, suffer from the **vocabulary mismatch problem**: they work only if there is exact overlap of words between the query and the document.

The solution to this problem is an approach that can handle synonymy, using dense embeddings. This can be implemented via encoders like BERT.

Neural information retrieval

We present the query q and the document d to a single BERT encoder.

The transformer cross-attention builds a representation that is sensitive to the meanings of both q and d.

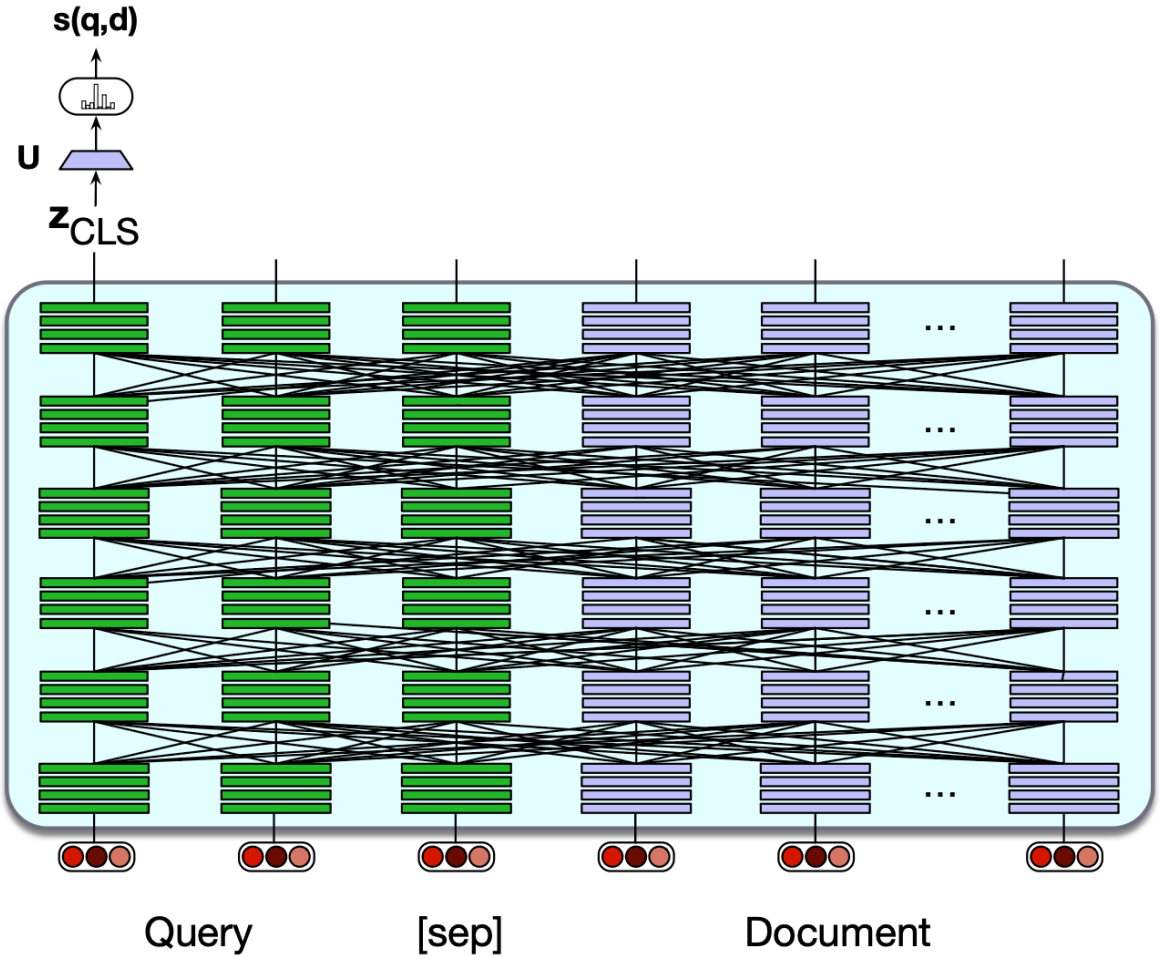
Finally, a linear layer $\mathbf{U}$ at the `[CLS]` token is **trained** to predict a **relevance score** for q, d:

$$\begin{aligned} \mathbf{z} &= \text{BERT}(q; \texttt{[SEP]}; d)[\texttt{[CLS]}] \\ \text{score}(q, d) &= \text{softmax}(\mathbf{U} \cdot \mathbf{z}) \end{aligned}$$

Some of BERT's parameters might be **fine-tuned** as well for the relevance task. The resulting model is called **cross-encoder**.

Neural information retrieval

Representation of the cross-encoder model.



Neural information retrieval

To favour semantic search, documents are broken up into **passages** that fit into a buffer of max N tokens.

Passages may **overlap**, to avoid splitting information.

The query, the passage, and the special tokens `[CLS]` and `[SEP]` have to fit into the BERT 512-token window.

Neural information retrieval

Every time we get a query, we have to pass **every** single passage in our collection through BERT! This is impractical for real cases.

A much more efficient architecture is the **bi-encoder**. In this architecture we use two separate encoders

- BERT_D for the document / passage
- BERT_Q for the query

We can encode each document and store the vectors in advance.

When a query comes in, we encode just this query and then use the **dot product** as the score for each candidate document.

Neural information retrieval

If we use the [CLS] token embedding to represent d and q, we have the following equations

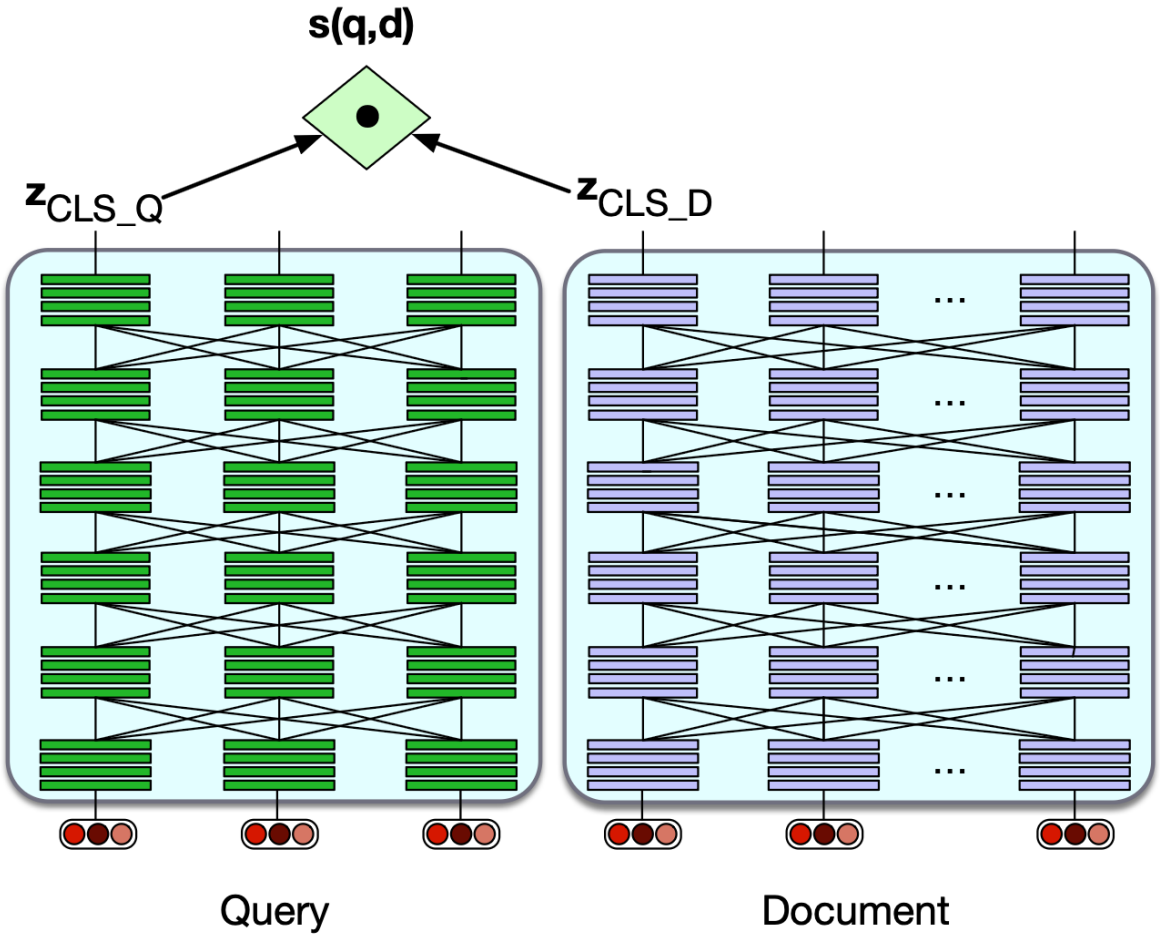
$$\begin{aligned} \mathbf{z}_q &= \text{BERT}_Q(q)[\texttt{[CLS]}] \\ \mathbf{z}_d &= \text{BERT}_D(d)[\texttt{[CLS]}] \\ \text{score}(q,d) &= \mathbf{z}_q \cdot \mathbf{z}_d \end{aligned}$$

The bi-encoder is **much cheaper** than a cross-encoder.

However, it is **less accurate** since it cannot take full advantage of all the possible meaning interactions between tokens in d and q.

Neural information retrieval

Representation of the bi-encoder model.



Neural information retrieval

Hybrid approaches

- use sparse encoders (cheaper methods like BM-25) for computing relevance ranking among all passages
- rerank the top N passages from previous step, using expensive methods like BERT

Neural information retrieval

An alternative approach is **ColBERT** (contextualized late interaction over BERT), which reconciles efficiency and contextualization.

This method separately encodes q and d into contextual representations for each token. Encoding of d can be done in advance.

The idea is that a relevant document will have tokens that are contextually very similar to the query.

For each token in q, ColBERT finds the most contextually similar token in d, and then sums up these similarities.

Neural information retrieval

Assume two separate BERT encoders for query q and passage d

- $q = q_1, \ldots, q_N$ is encoded into vectors $\mathbf{q}_1, \ldots, \mathbf{q}_N$
- $d = d_1, \ldots, d_m$ is encoded into vectors $\mathbf{d}_1, \ldots, \mathbf{d}_m$

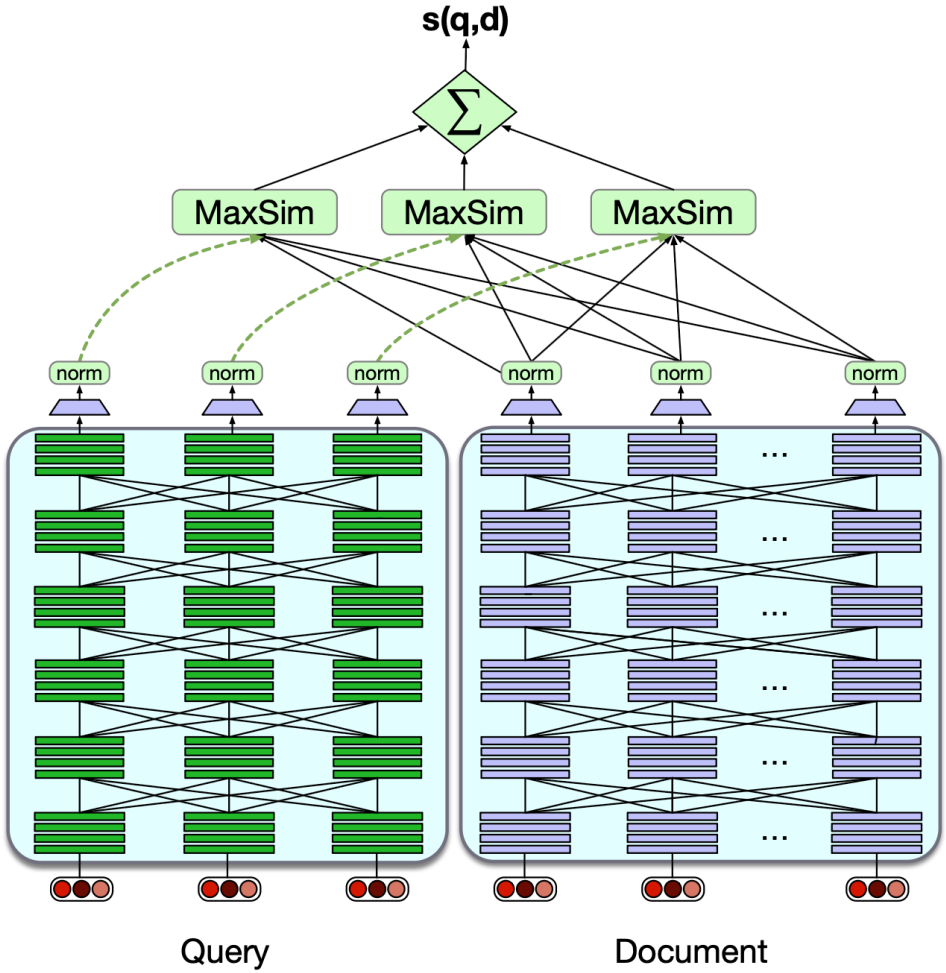
A linear layer $\mathbf{U}$ is applied to reduce these vectors' dimensions for storage efficiency, and vectors are rescaled to unit length, producing the final vectors $\mathbf{E}_{q_i}$ and $\mathbf{E}_{d_j}$.

The ColBERT scoring function is

$$\text{score}(q, d) \quad = \quad \sum_{i=1}^{N} \max_{j=1}^{m} \mathbf{E}_{q_i} \cdot \mathbf{E}_{d_j}$$

Neural information retrieval

Representation of ColBERT model.



Neural information retrieval

The two BERT encoders are **finetuned** end-to-end and the linear layers for dimensional reduction are trained from scratch.

Training set consists of triples $\langle q, d^+, d^- \rangle$

- q is a query
- d^+ is a positive fragment
- d^- is a negative fragment (random sampling)

Training is **contrastive** and uses a cross-entropy loss.

Compare with skipgram algorithm for static word embeddings.

Neural information retrieval

In neural information retrieval, we need to be able to do efficient ranking using the computed scores.

Finding the set of fragment vectors that have the highest dot product with a query vector is an instance of the problem of **nearest neighbor search**.

Faiss (Facebook AI similarity search) is a popular library for approximate nearest neighbor vector search.

Generation

©Medium

Generation

The idea of generation is to **condition** on the retrieved passages, jointly with some prompt text.

Let $\mathcal{D}$ be a document collection and let q be some user query. The basic **RAG algorithm** consists of the following steps

- call a retriever to return $R(q) = d_1, \ldots, d_k$ the top-k relevant fragments from $\mathcal{D}$
- create a prompt that includes q and $R(q)$
- call an LLM with the constructed prompt

Generation

Example : The resulting prompts might look something like

Schematic of a RAG Prompt

```
retrieved passage 1

retrieved passage 2

...

retrieved passage k

Based on these texts, answer this question: What year
was the premiere of The Magic Flute?
```

Generation

More formally, we **reduce** the Q/A problem to the problem of computing the following probability.

For given q and $R(q)$, the task for the language model is to generate text according to this probability model (symbol ';' denotes string concatenation)

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid R(q) \ ; \ \texttt{prompt} \ ; \ \texttt{[Q:]} \ ; \ q \ ; \ \texttt{[A:]} \ ; \ x_{<i})$$

Summary

Important advantages of RAG approach

- external knowledge source injects into LLM recent/specific information that wasn't available during pre-training
- external knowledge source can be updated, no need to retrain the LLM
- RAG works well also with small-size, more manageable LLMs
- allows fact checking and reduces hallucinations

Advanced RAG methods

©Medium

Advanced RAG methods

Several variants of the basic RAG paradigm have been explored.

Systems provide the user with evidence for any factual statements, in the form of URLs or **citation** references.

The simplest way for generating citations is to specify instructions as part of the prompt.

We can perform **instruction tuning** on an LLM, using a dataset of questions annotated with retrieved passages and correct answers.

Some RAG architectures add a neural **reranker** that reranks fragments after they are retrieved.

Advanced RAG methods

So-called **multi-hop** questions require iterative retrieval mechanisms, enabling reasoning across multiple documents

Graph RAG integrates graph-based data structures, enabling richer and more accurate generative output, particularly for tasks requiring relational understanding.

Advanced RAG methods

Agentic RAG represents a paradigm shift by introducing autonomous agents capable of dynamic decision-making and workflow optimization.

Key characteristics of agentic RAG

- agents independently evaluate and manage retrieval strategies based on query complexity
- system incorporates feedback loops to improve retrieval accuracy and response relevance
- agents dynamically orchestrate tasks, enabling efficiency in real-time applications

Datasets and evaluation

©Medium

Datasets

Several datasets that contain questions annotated with answers.

These datasets can be used both for instruction tuning and for evaluation of RAG systems.

Datasets can be distinguished in two classes, on the basis of the original purpose of the questions

- questions meant for **information-seeking**, such as in web search
- questions designed for **probing**: evaluating or testing systems or humans

Datasets

Some popular datasets related to information-seeking

- **Natural Questions**, a set of anonymized English queries to the Google search engine annotated with answers based on Wikipedia
- **MS MARCO** (Microsoft machine reading comprehension), a collection of anonymized English questions from Microsoft Bing with human generated answers and passages that can be used both to test retrieval ranking and question answering
- **TyDi QA**, a collection of question-answer pairs from 11 languages, including Arabic, Bengali, Kiswahili, Russian, and Thai; input is a question and some passages from Wikipedia, and system must mark the minimal answer span, or answer NULL if no passage contains the answer

Datasets

On the probing side

- **MMLU** (massive multitask language understanding) is a multiple answer dataset including knowledge and reasoning questions in 57 areas such as medicine, mathematics, computer science, law, and others

  We have already introduced MMLU in some previous lecture.

Evaluating question answering systems

In question answering (QA) we distinguish two typologies of tasks

- **closed book** QA: tasks where questions should be answered by the LLM without consulting any external knowledge
- **open book** QA: tasks where question come with one or more documents (for example in RAG)

Evaluating question answering systems

Two techniques commonly employed to evaluate QA systems.

For multiple choice questions use **exact match**, defined as the number of predicted answers that match the gold answer exactly

For questions with free text answers use average **F1 score**

- compute precision and recall based on the words shared between the two answers
- compute the F1 score as the harmonic mean of precision and recall
- average F1 over all questions

Evaluating question answering systems

While powerful, average F1 score has two blind spots

- use of synonyms will not be seen as a match
- answers are treated as bags of words, therefore system is not penalized for scrambled, nonsensical word order

Many researchers now supplement average F1 with

- BERTScore
- LLM-based evaluation to check for semantic meaning
- other semantic metrics like ROUGE or METEOR, borrowed from the areas of machine translation and of text summarization