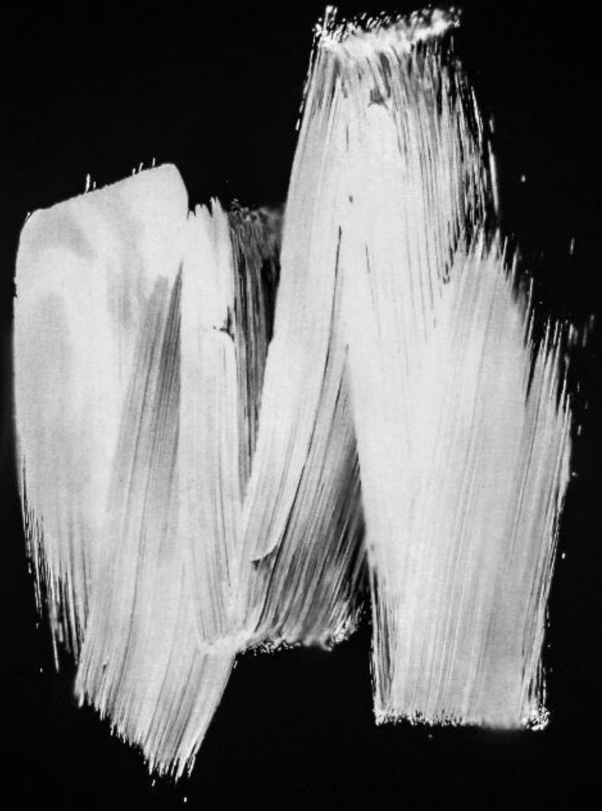


NLP with Transformers
Chapter7 (Question

1/23/24

발표자: 김영주

Chapter Preview

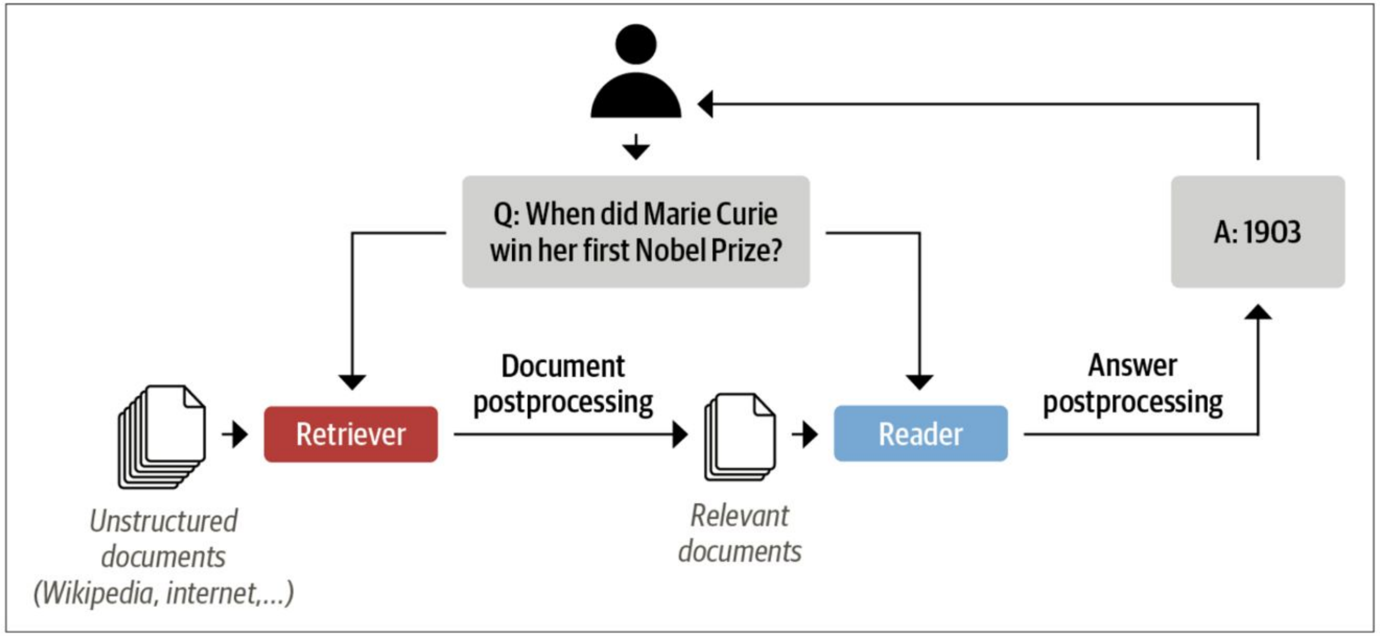
💡 There are many flavor of QA systems.

This Chapter focuses on **extractive QA systems.**

It consists a **two-stage process**

1. **retrieve relevant documents** from the question.
2. **extract answers** from them.

The retriever-reader architecture for Extractive QA systems



목차

Extracting Answers from Text

Extracting Answers from Text

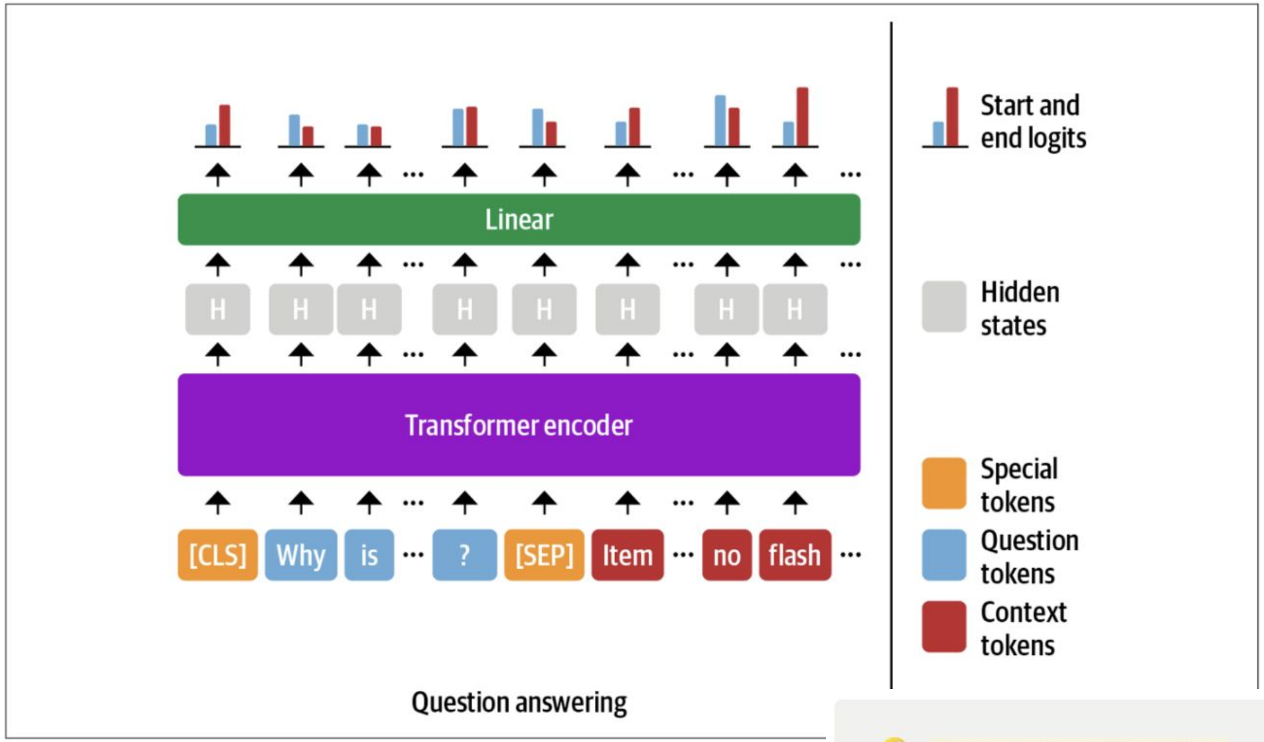
question
Does the keyboard lightweight?

context
I really like this keyboard. I give it 4 stars because it doesn't have a CAPS LOCK key so I never know if my caps are on. But for the price, it really suffices as a wireless keyboard. I have very large hands and this keyboard is compact, but I have no complaints.

answers.text
[this keyboard is compact]

Span Classification

The span classification head for QA task



🔑 **Span classification**

The most common way to extract answers from text is by **framing the problem as a span classification task.**

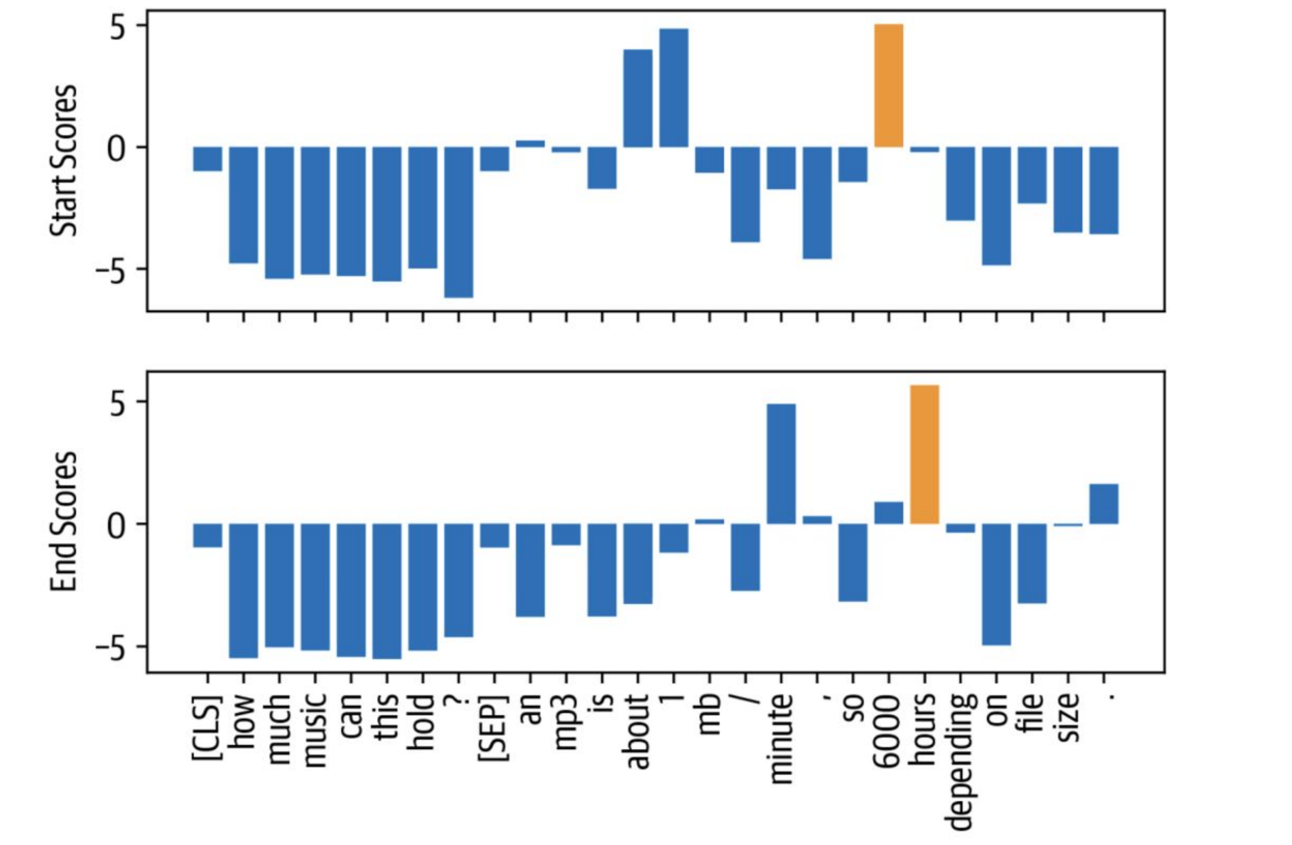
The **start and end tokens of an answer span act as the labels that a model needs to predict.**

Extracting Answers from Exampl

Input

```
[CLS] how much music can this hold? [SEP] an mp3 is about 1 mb / minute, so
about 6000 hours depending on file size. [SEP]
```

Output



Extracting Answers from Text

☝️ Sometimes, **the documents may not contain the answer.**
In these cases the model will assign **a high start and end score to the [CLS] token.**

🤔 Still, this method can produce out-of-scope answers by selecting tokens that belong to the question instead of the context.

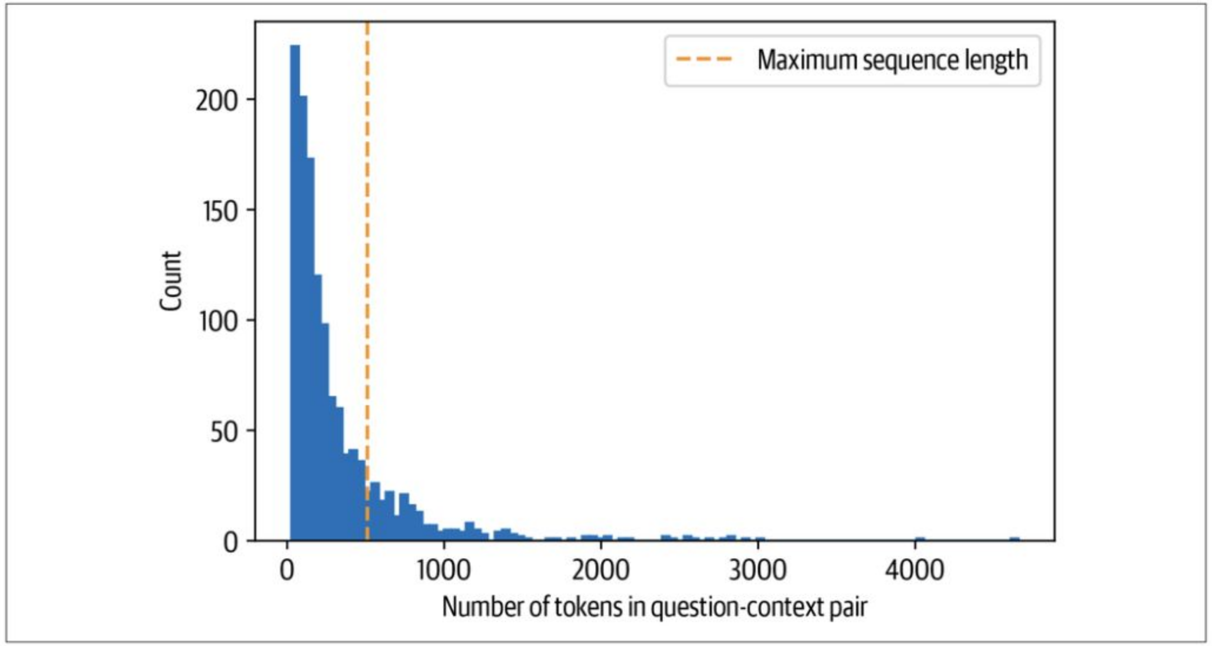
☝️ In practice, the pipeline computes the best combination of start and end indices subject to various **constraints(heuristics)** such as **being in-scope, requiring the start indices to precede the end indices**, and so on.

Dealing with long

🤔 **Dealing with long passages**

Looking at one problem is that the context often contains more tokens than the maximum sequence length of the model.

For QA, **we can't just truncate** because **the answer to a question could lie near the end** of the context and thus would be removed by truncation.

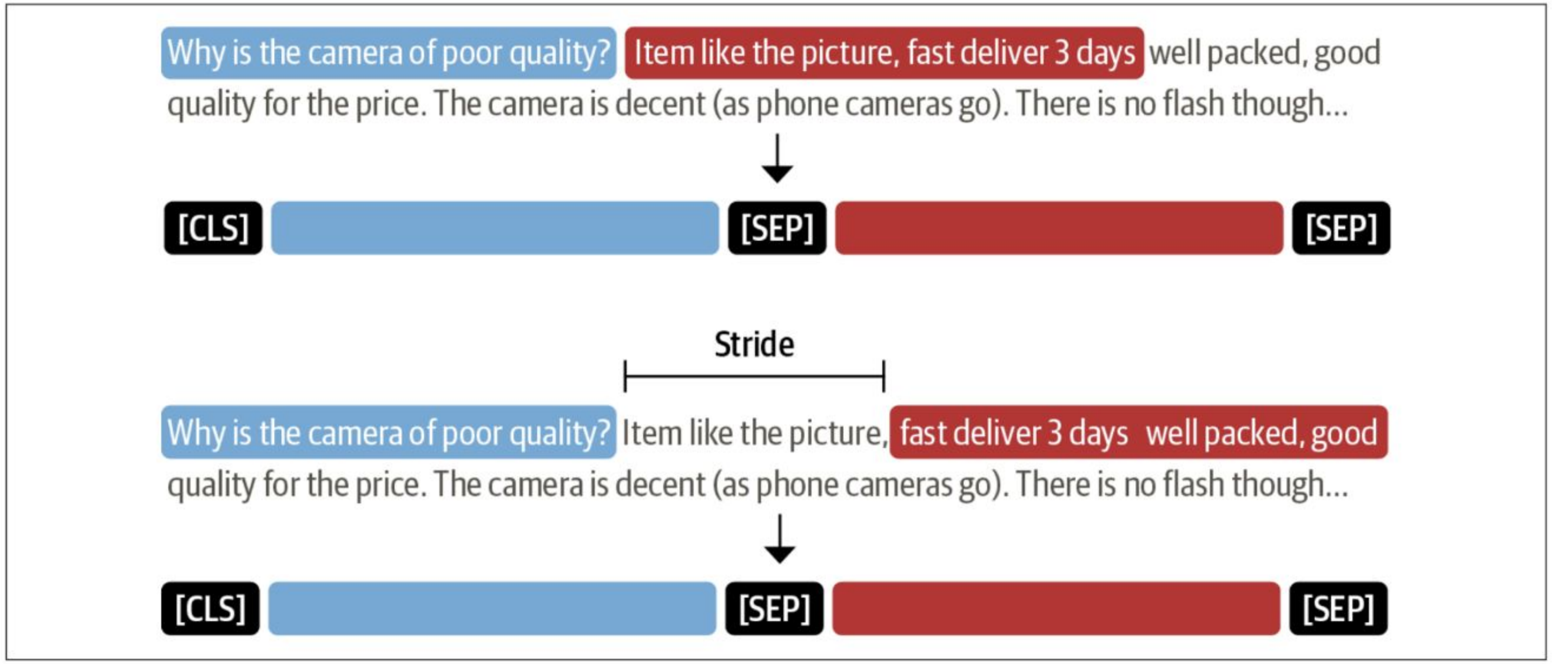


Dealing with long passages

💡 **Apply a sliding window!**

The standard way to deal with this is to apply a sliding window across the inputs, where each window contains a passage of tokens that fit in the model's context.

Multiple question-context pairs for long documents using sliding window



Encoder Models for the QA task

💡 These are a few models to use as the Encoder for the QA task.

Baseline transformer models that are fine-tuned on SQuAD 2.0

Transformer	Description	Number of parameters	F_1-score on SQuAD 2.0
MiniLM	A distilled version of BERT-base that preserves 99% of the performance while being twice as fast	66M	79.5
RoBERTa-base	RoBERTa models have better performance than their BERT counterparts and can be fine-tuned on most QA datasets using a single GPU	125M	83.0
ALBERT-XXL	State-of-the-art performance on SQuAD 2.0, but computationally intensive and difficult to deploy	235M	88.1
XLM-RoBERTa-large	Multilingual model for 100 languages with strong zero-shot performance	570M	83.8

Modern QA Systems

Intro

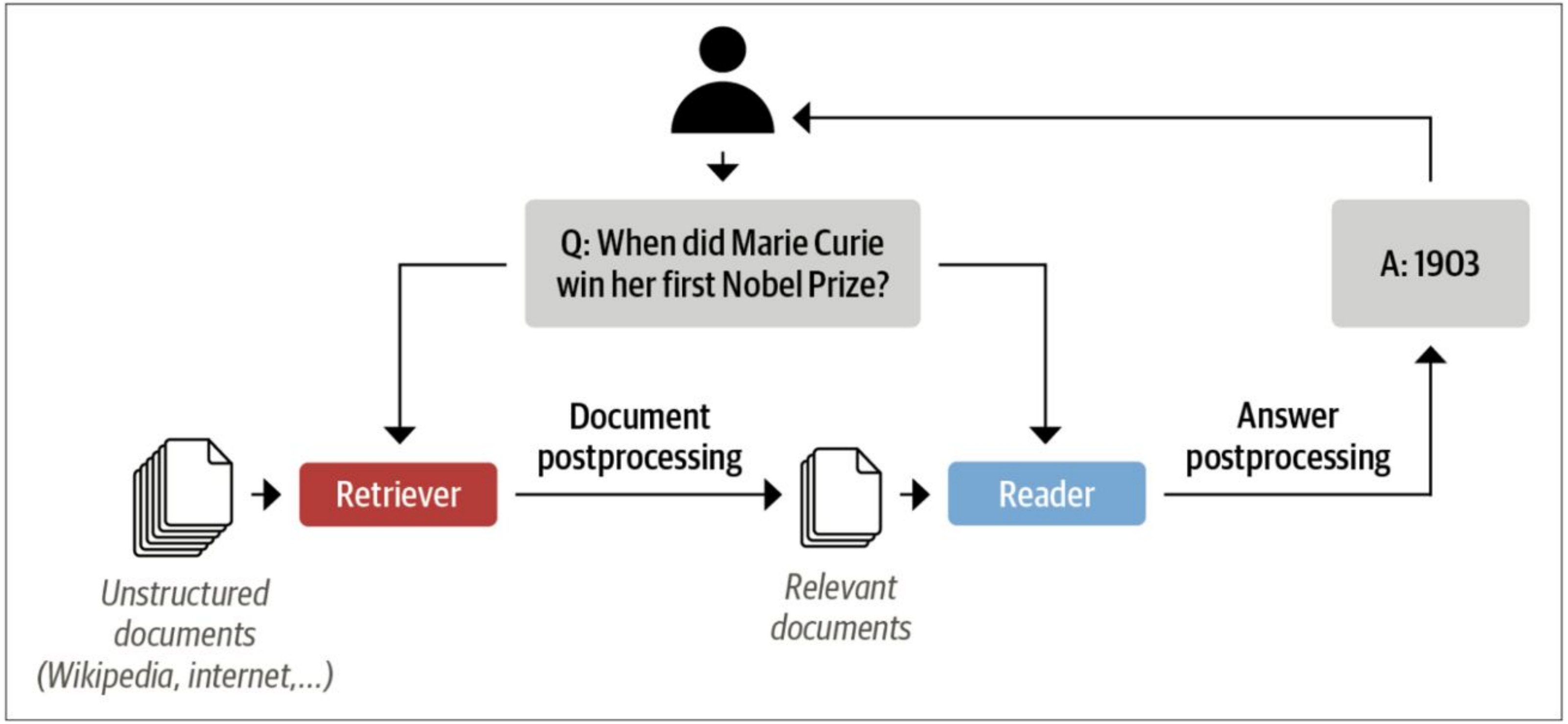
🤔 In reality a system's users will only provide a question.
So, **we need some way of selecting relevant documents in the database.**

😟 One way to do this would be to **concatenate all the documents and feed them to the model** as a single, long context.
Although simple, the drawback of this approach is that the context can become extremely long and thereby introduce an **unacceptable latency** for our users' queries.

To handle all together, modern QA systems are typically based on the **retriever-reader architecture**, which has two main components:

Retriever:
Responsible for **retrieving relevant documents for a given query.**

Reader:
Responsible for **extracting an answer from the documents** provided by the retriever.



Modern QA Systems

Retriever(Sparse)

💡 Retrievers are usually categorized as **sparse** or **dense**.

- **Sparse retrievers**
  - **use word frequencies** to represent each document and query as a **sparse vector.**
  - the relevance of a query and a document is then determined by **computing an inner product of the vectors.**
  - Eg. TF-IDF, BM25

🔑 **BM25**

- represents the question and context as sparse vectors
- improved version of TF-IDF

Given a query Q, containing keywords $q_1, \ldots, q_n$, the BM25 score of a document D is:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

  - **saturates TF values quickly**
  - **normalizes the document length.**
    - **this makes short documents be favored over long ones.**

Retriever(Dense)

🧐 **Limitations of Sparse retrievers**

A well-known limitation of sparse retrievers like BM25 is that they **can fail to capture the relevant documents if the user query contains terms that don't match exactly those of the review.**

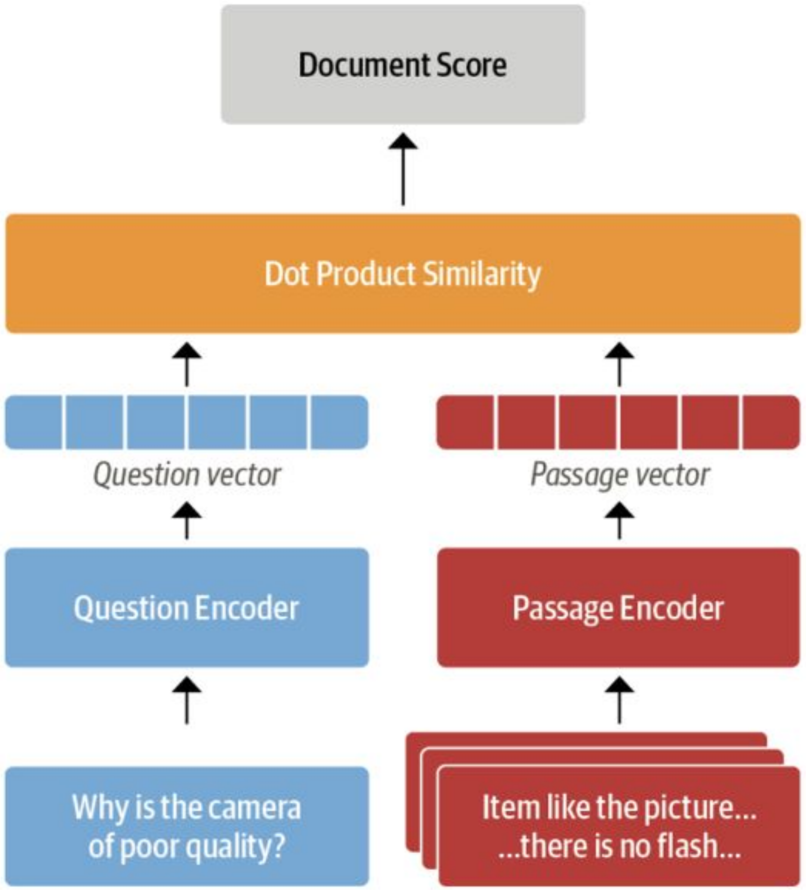
☝️ One promising alternative is to **use dense embeddings** to represent the question and document.

- **Dense retrievers**
  - **use encoders** like transformers to represent the query and document as **contextualized embeddings.**(which are dense vectors)
  - these **embeddings encode semantic meaning**, and allow dense retrievers to improve search accuracy by understanding the content of the query and document.
  - Eg. Embedding, DPR(Dense Passage Retrieval)

🔑 **DPR(Dense Passage Retrieval)**

- the current state of the art
- uses two BERT models as **encoders**
  - **each for the question and the passage.**
  - these encoders **map** the input text into a **d-dimensional vector** representation **of the [CLS] token.**
  - encoders are trained by giving them questions with relevant (positive) passages and irrelevant (negative) passages
  - the goal is to **learn that relevant question-passage pairs have a higher similarity.**

DPR's bi-encoder architecture



Evaluating the Retriever

Evaluating the Retriever

☝️ **Recall**

- the fraction of all relevant documents that are retrieved.

✌️ **mAP**

- rewards retrievers that can place the correct answers higher up in the document ranking

Reader

🔑 Responsible for **extracting an answer from the documents** provided by the retriever. (What we did earlier)

Evaluating the Reader.

☝️ **Exact Match (EM)**

- A **binary metric**
- EM = 1 if the characters in the predicted and ground truth answers match exactly
- EM = 0 otherwise
- If no answer is expected, the model gets EM = 0 if it predicts any text at all.

☝️ **F1-score**

- Measures **the harmonic mean of the precision and recall.**

Example

```
pred = "about 6000 hours"
label = "6000 hours"
print(f"EM: {compute_exact(label, pred)}")
print(f"F1: {compute_f1(label, pred)}")
```

```
EM: 0
F1: 0.8
```

precision: 2/2, recall: 2/3

🔑 **Tracking both metrics** is a good strategy **to balance the trade-off** between underestimating (EM) and overestimating(F1-score) model performance.

In general, there are multiple valid answers per question
→ the best score is selected over all possible answers.

The overall EM and F1-score for the model are then obtained by averaging over the individual scores of each question-answer pair.

HayStack & ElasticSearch

Document store(Elastic Search)

🔑 A **QA system in real usage needs a Document store**(database).
The book recommends Elastic Search.

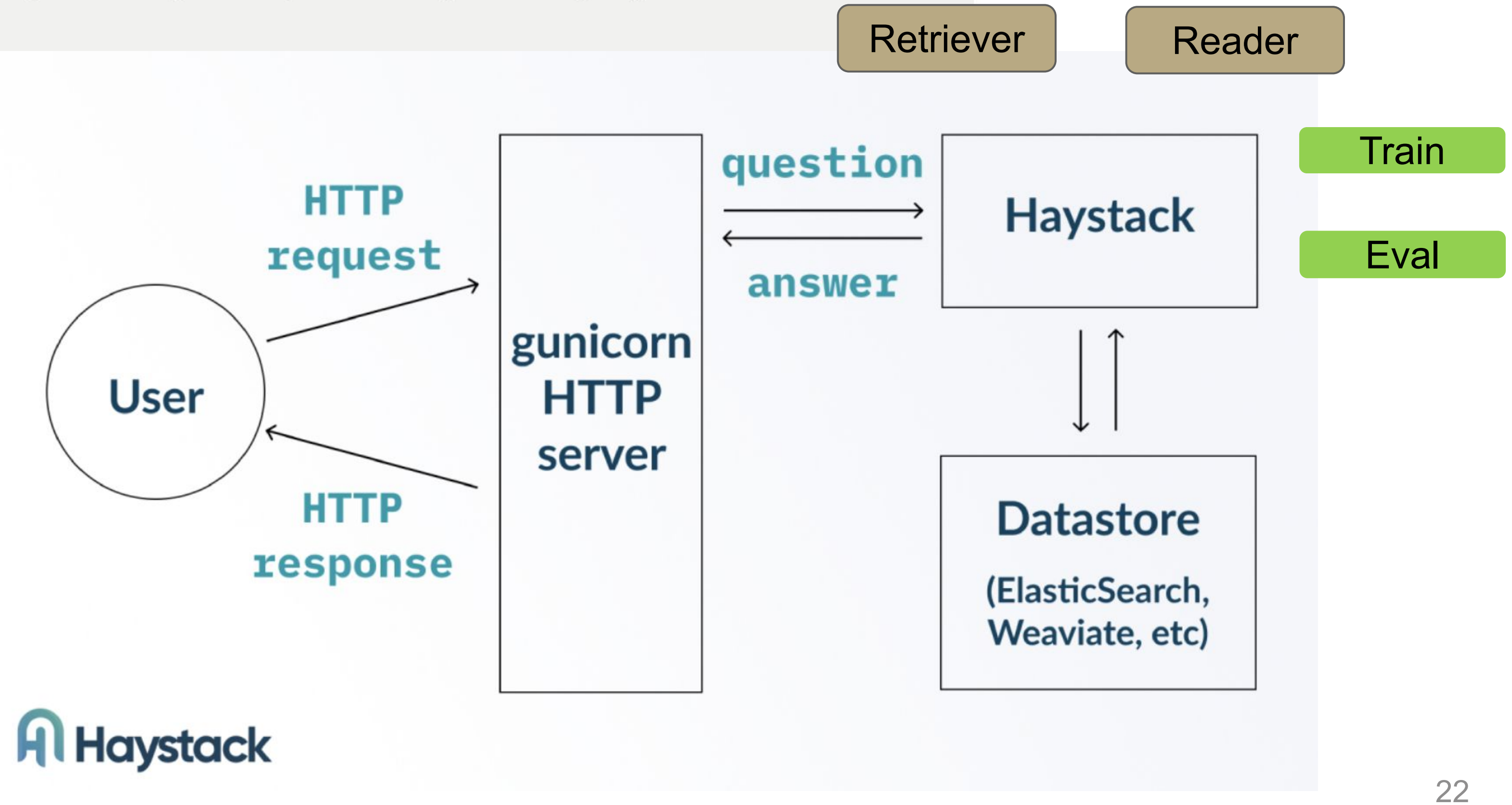
💿 **Elastic Search**

Elasticsearch is a search engine that is capable of handling a diverse range of data types, including textual, numerical, geospatial, structured, and unstructured.
Its ability to store huge volumes of data and quickly filter it with full-text search features makes it especially well suited for developing QA systems.

HayStack

🧐 We need **a powerful Libary to handle** all these complicated tasks.

⚡ **Haystack** is based on the retriever-reader architecture, abstracts much of the complexity involved in building these systems, and integrates tightly with 🤗Transformers.



Going Beyond Extractive QA

Going Beyond Extractive QA

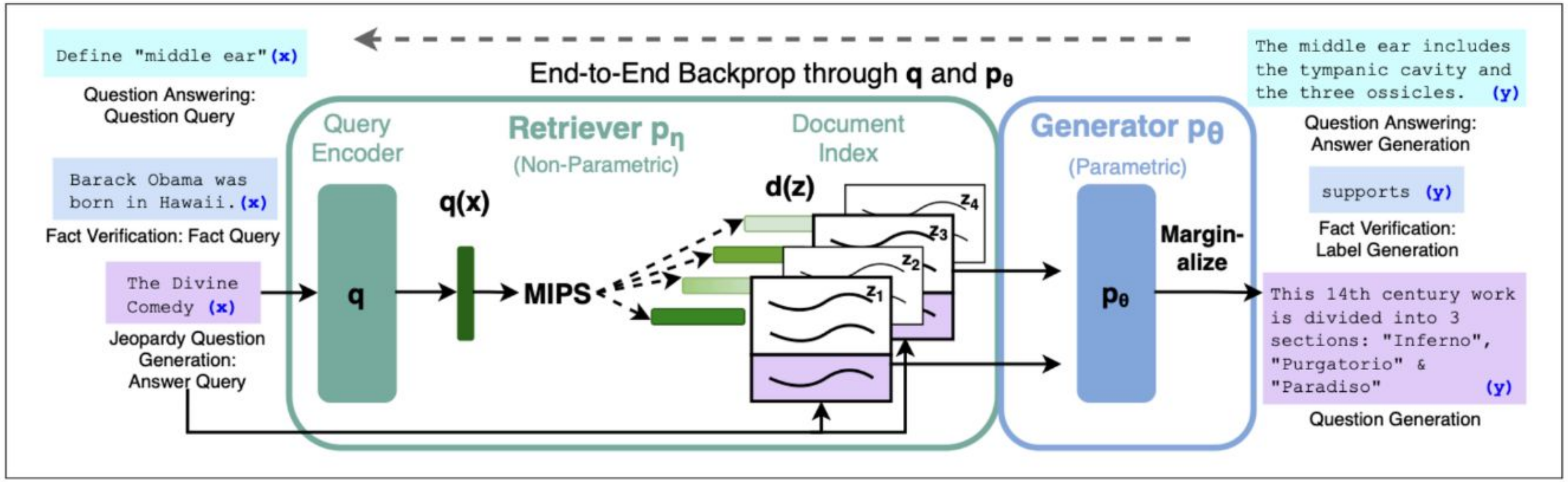
Generative(Abstractive) QA

Instead extracting answers as spans of text in a document
generate answers with a pretrained language model.

RAG(retrieval-augmented generation)

- sota model when the book was written
- extends the classic retriever-reader architecture
    - uses **DPR as retriever**
    - uses a **generator instead of a reader**
        - a pretrained seq-to-seq transformer like T5 or BART
        - receives **latent vectors** of documents from DPR and then iteratively **generates an answer based on the query and these documents.**

The RAG architecture



rf. MIPS: Maximum inner-product search

Thankyo