CS310 Natural Language Processing
自然语言处理
Lecture 12: In-Context Learning and Question Answering

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

Overview

- In-Context Learning
- Prompting
- Question Answering

Emergent Abilities of LLMs: GPT (2018)

- Generative Pretrained Transformer (**GPT**) (OpenAI, Radford et al., 2018)
- 117M params
- 12 layers of transformer decoders
- Trained on BooksCorpus (4.6GB text)

- GPT showed that language model at scale (LLM) is an effective pretraining technique for downstream tasks
  - such as entailment, question answering, commonsense reasoning
- Pretraining + fine-tuning

outshined by BERT in the same year

Emergent Abilities of LLMs: GPT-2 (2019)

涌现

- GPT-2 (Radford et al., 2019)
- 1.5B params; same architecture as GPT (bigger, 117M ⇒ 1.5B)
- Trained on **much more data**: 4.6GB ⇒ 40GB of internet text data

- One key emergent ability in GPT-2 is **zero-shot learning**:
- The ability to do many tasks with **no examples**, and **no gradient updates**

Emergent zero-shot learning of GPT-2

- Question answering:

```
Passage: The 2008 Summer Olympics torch relay was run
from March 24 until August 8, 2008, prior to the 2008
Summer Olympics, with the theme of “one world, one dream”.

Q: What was the theme
A:
```

Emergent zero-shot learning of GPT-2

- Comparing sentence probabilities

The cat couldn't fit into the hat because it was too big.

Does "it" = "the cat" or "the hat"?

Is P(... because the cat was too big) >=
P(... because the hat was too big)?

Emergent zero-shot learning of GPT-2

- Summarization by adding **prompts**

"To induce summarization behavior we add the text **TL;DR:** after the article and generate 100 tokens with Top-k random sampling with k = 2"

TL; DR ⇒ Too Long; Didn't Read

"… approach the performance of classic neural baselines and just barely outperforms selecting 3 random sentences from the article."

- Zero-shot behavior comes from creative ways of specifying the task

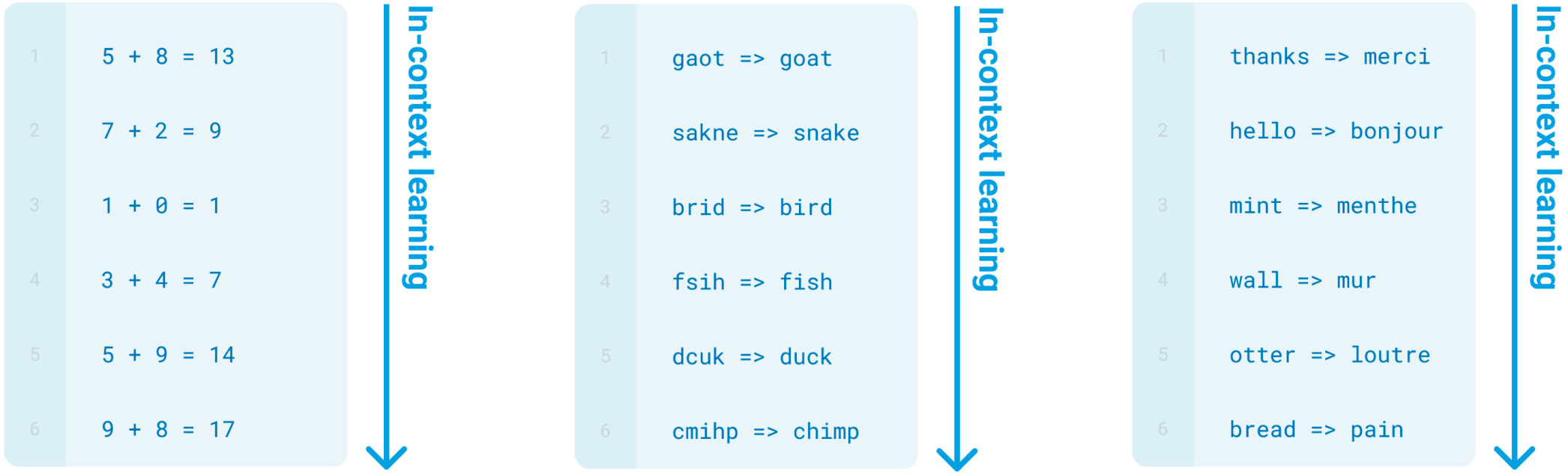
Emergent Abilities of LLMs: GPT-3 (2019)

- GPT-3 (175B parameters; Brown et al., 2020); same model and architecture as GPT-2
- Big increase in size: 1.5B ⇒ 175B
- Further increase in data: 40GB ⇒ 600+GB

"For all tasks, GPT-3 is applied without any gradient updates or fine-tuning, with tasks and **few-shot** demonstrations specified purely via text interaction with the model. "

Emergent few-shot learning of GPT-3

- Few-shot learning: Specify a task by simply **prepending examples before the question**
  - Also known as **in-context learning**; no gradient updates are performed when learning a new task



Source: Brown et al., 2020

Explanation: Zero-shot

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

Source: Brown et al., 2020

Explanation: One-shot

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Source: Brown et al., 2020

Explanation: Few-shot

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:
2  sea otter => loutre de mer
3  peppermint => menthe poivrée
4  plush girafe => girafe peluche
5  cheese =>
```

task description

examples

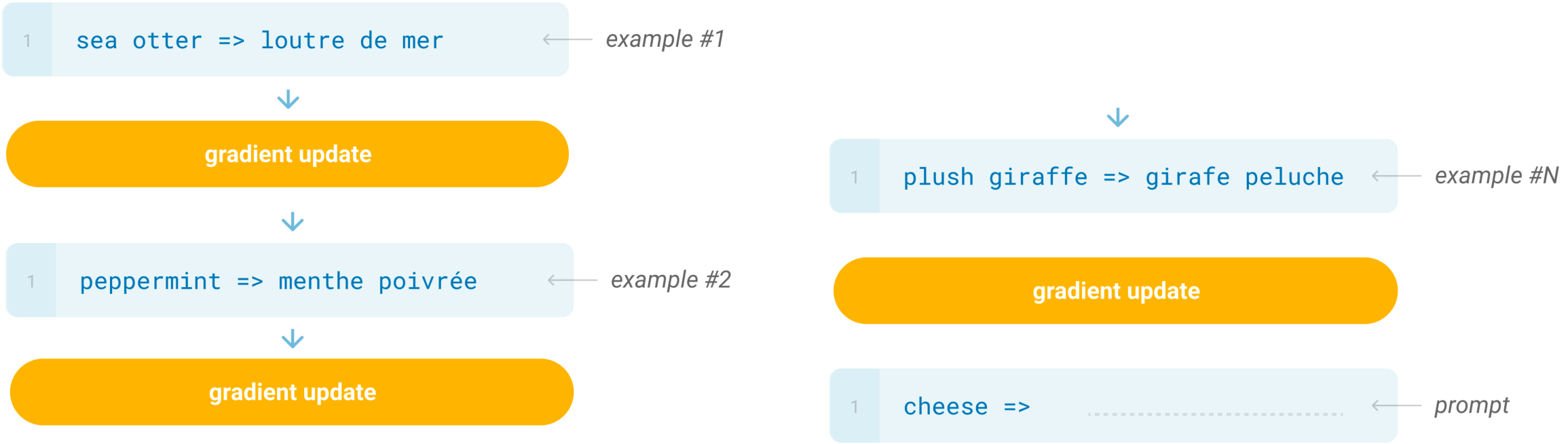
prompt

Source: Brown et al., 2020

Compared to traditional fine-tuning (not use in GPT-3)

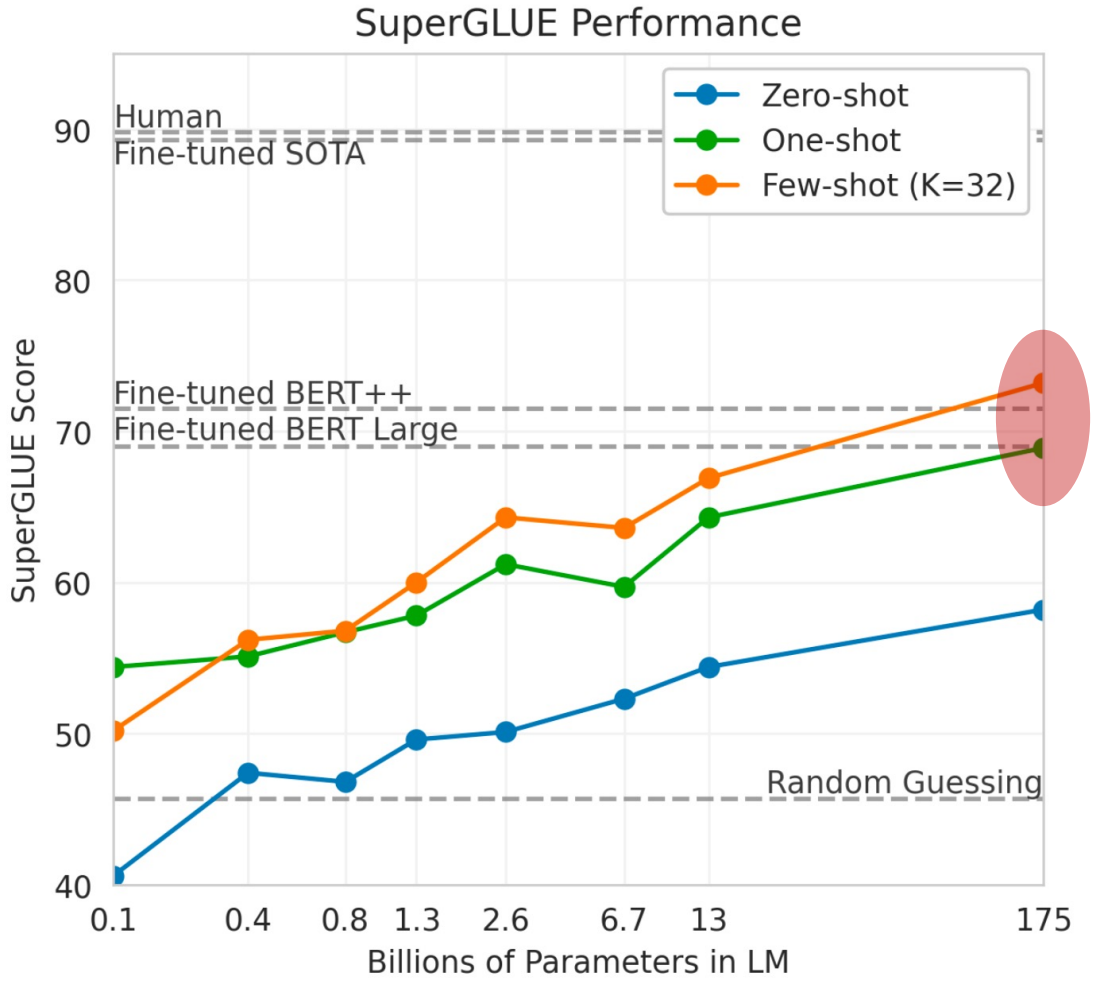
Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.



Source: Brown et al., 2020

Performance Comparison

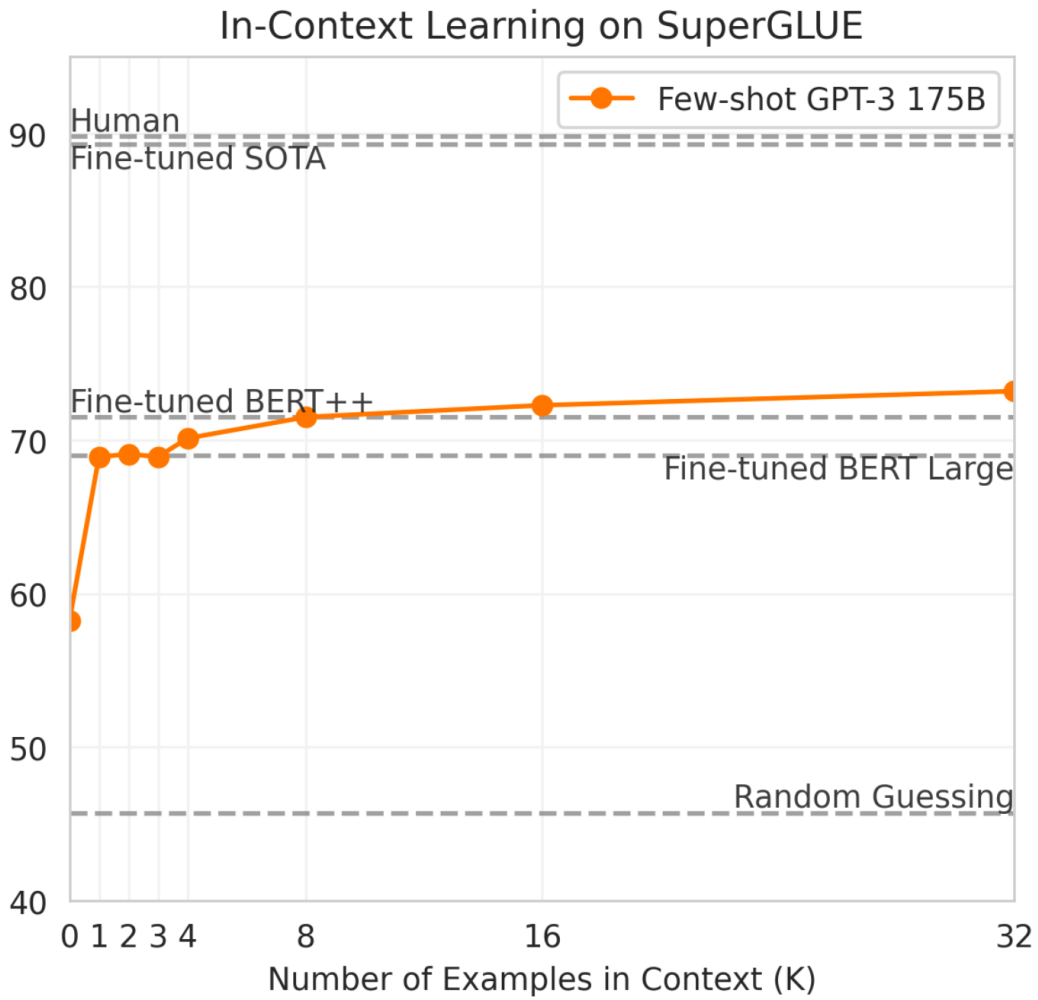


- BERT Large: fine-tuned on SuperGLUE training set (125K examples)
- BERT++: fine-tuned on MultiNLI (392K), SWAG (113K), and SuperGLUE (630K)

Difference between BERT-Large and BERT++ is roughly equivalent to the difference between GPT-3 with **one-shot** and **few-shot**

Source: Brown et al., 2020

Performance Comparison

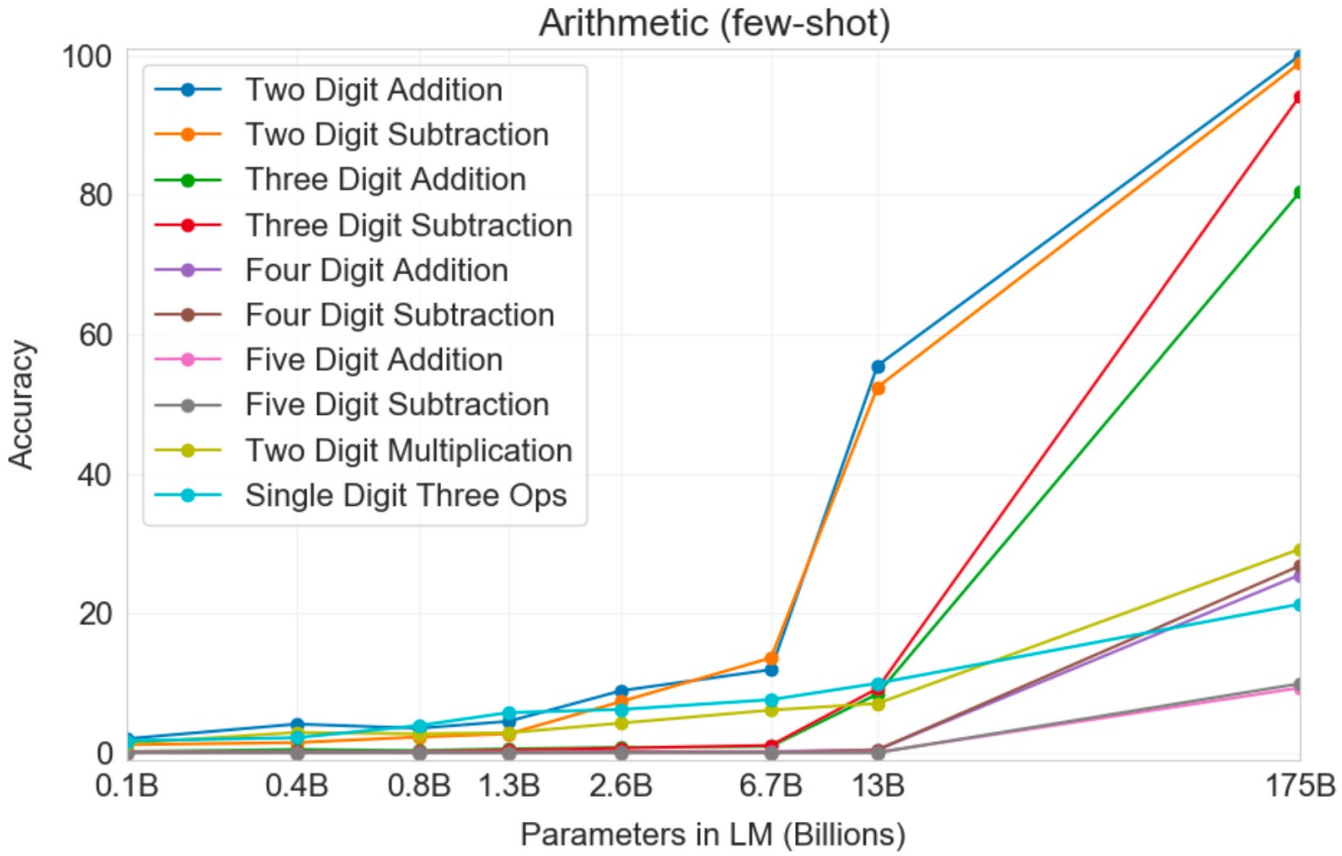


Difference between BERT-Large and BERT++ is roughly equivalent to the difference between GPT-3 with **one-shot** (1 example) and **few-shot** (8 examples)

Source: Brown et al., 2020

涌现

Few-shot learning: An emergent property of model size



Significant jump from the second largest model (GPT-3 13B) to the largest model (GPT-3 175B)

Model Name	n_{params}
GPT-3 Small	125M
GPT-3 Medium	350M
GPT-3 Large	760M
GPT-3 XL	1.3B
GPT-3 2.7B	2.7B
GPT-3 6.7B	6.7B
GPT-3 13B	13.0B
GPT-3 175B or "GPT-3"	175.0B

Source: Brown et al., 2020

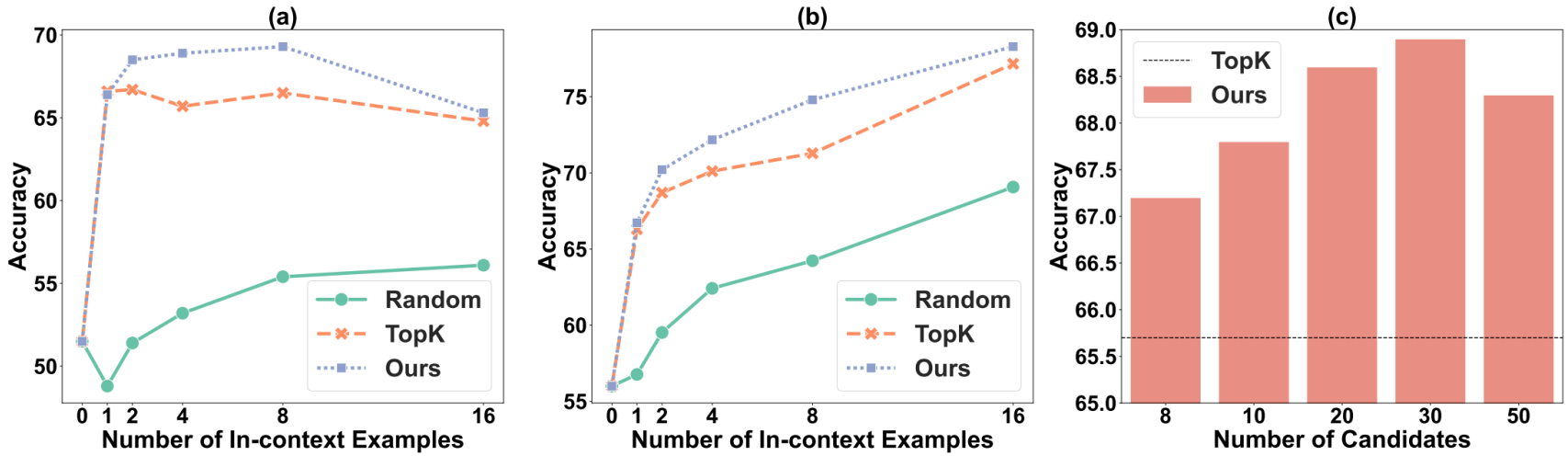
In-Context Learning (formally)

- Combine task description and context examples (few shots) as the input prompt to LLMs

$$\underbrace{\text{LLM}}_{\text{Model}}\Big(\underbrace{I}_{\text{Task description}}, \underbrace{f(x_1, y_1), \ldots, f(x_k, y_k)}_{\text{Few-shot examples}}, f(\underbrace{x_{k+1}}_{\text{Input}}, \underbrace{\_\_\_}_{\text{Answer}})\Big) \rightarrow \hat{y}_{k+1}$$

Factors of In-Context Learning

- Main factor: Number and quality of few-shot examples



Top-K: Select the top k semantically related examples
(a simple baseline)

Revisiting Demonstration Selection Strategies in In-Context Learning. ACL 2024.

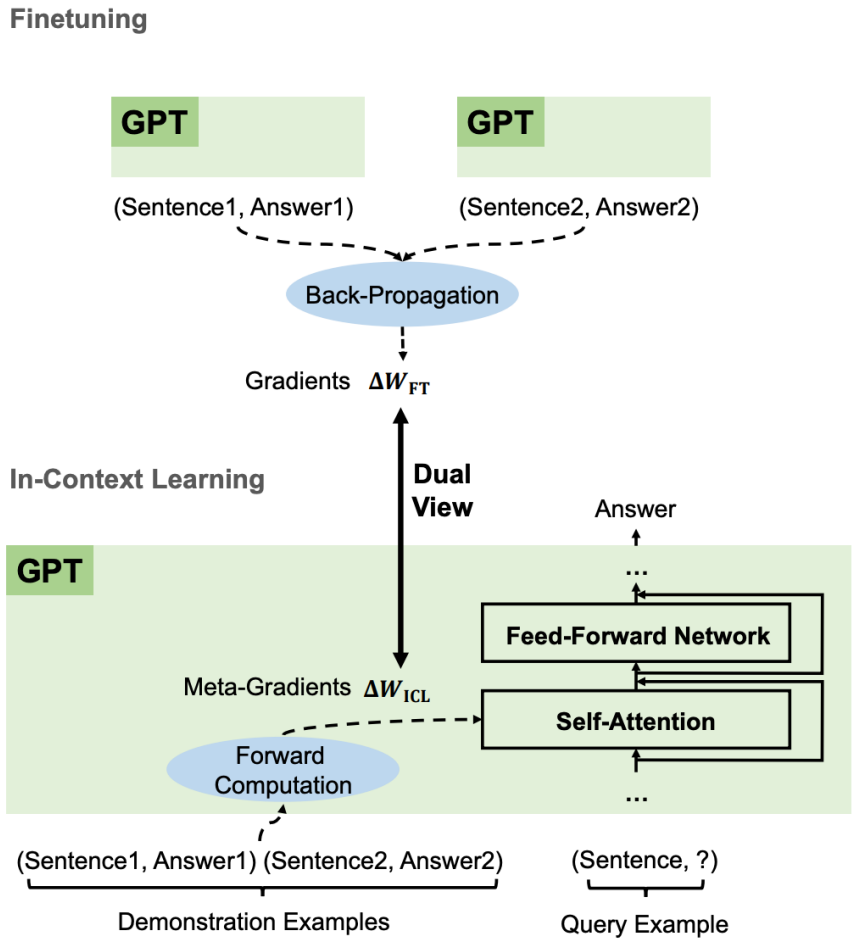
Why In-Context Learning happens?

- In-context learning = task recognition + task learning
  - Task recognition: recognize what the task is about by analyzing the few-shot examples.
    (Even small models ~350M can do)
  - Task learning: actually learn new information from the examples
    (Larger models ~70B can do better)

- ICL performs implicit gradient descent (Dai et al., 2023)

ICL as implicit gradient descent

- LLMs produce meta-gradients for ICL through forward computation.



Why Can GPT Learn In-Context? Language Models Implicitly Perform Gradient Descent as Meta-Optimizers. ACL 2023

Overview

- In-Context Learning
- **Chain-of-Thought Prompting**
- Question Answering

Limits and improvement of prompting

- Some tasks seem too hard for even large LMs to learn through prompting alone
- Especially tasks involving richer, **multi-step reasoning**

- Solution: **Chain-of-thought** prompting

Chain of thought: a series of intermediate reasoning steps

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

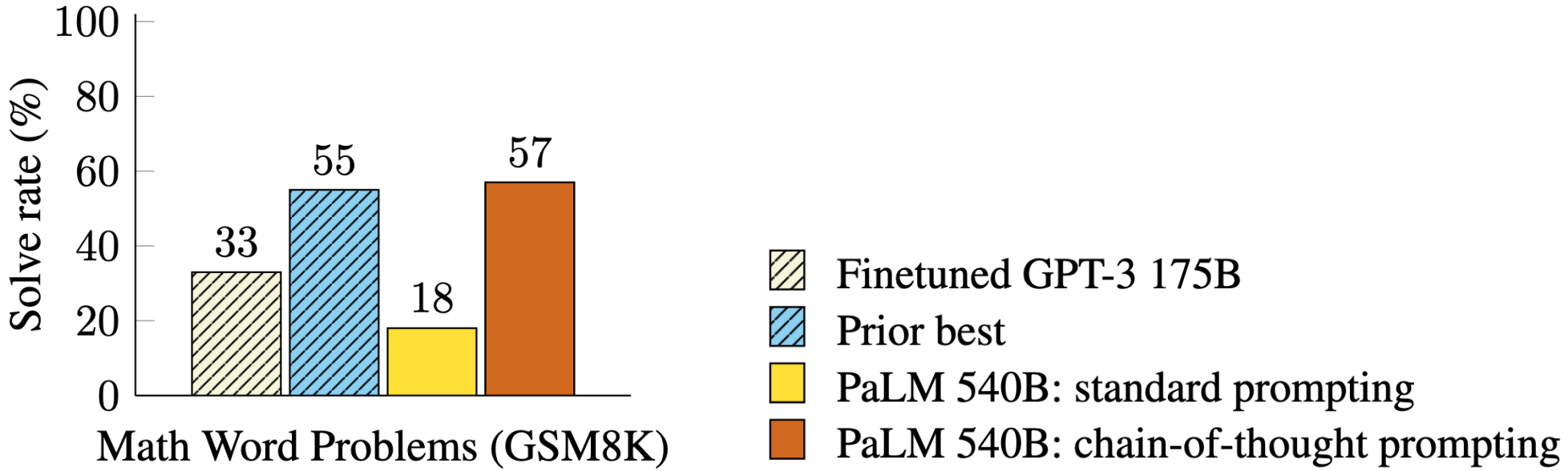
Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Wei et al., 2022

Chain of thought: a series of intermediate reasoning steps

- Prompting a **PaLM 540B** with just eight chain-of-thought exemplars achieves state-of-the-art accuracy on the GSM8K benchmark of math word problems (middle school math), surpassing even finetuned GPT-3 with a verifier.

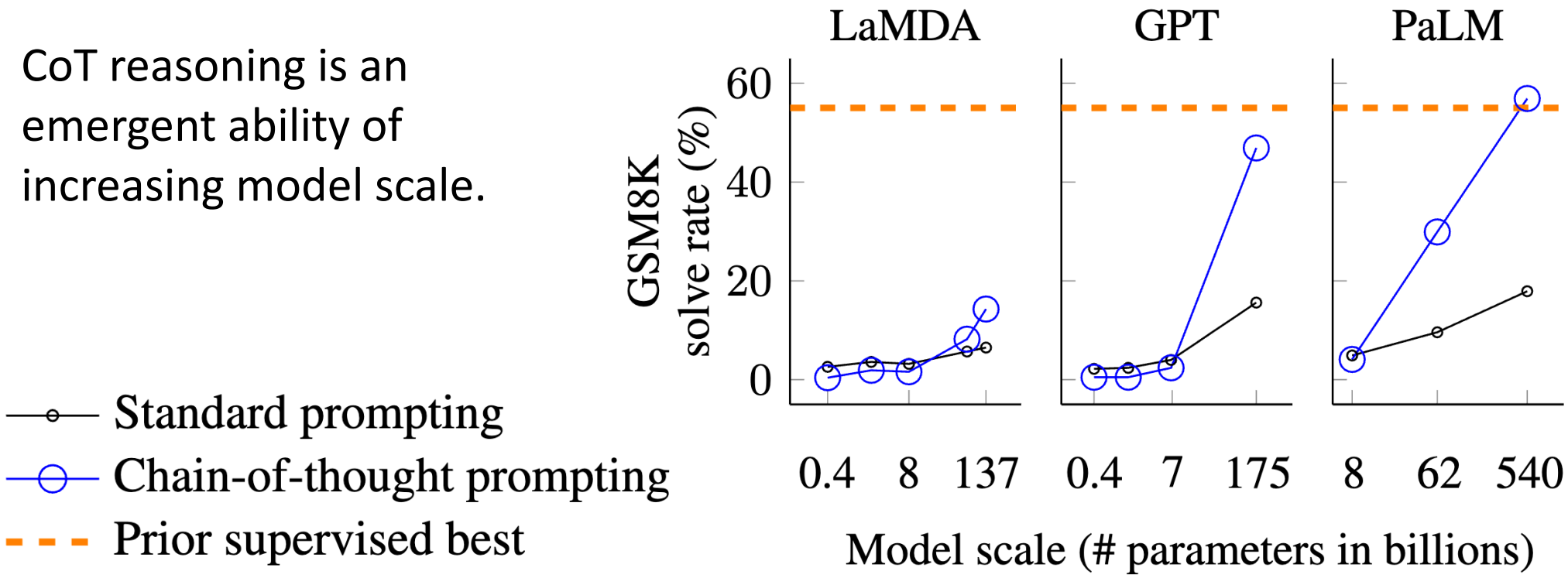


Wei et al., 2022

CoT prompting is also an emergent property

涌现

- CoT reasoning is an emergent ability of increasing model scale.



Zero-shot Chain of Thought

- Do we even need few-shot examples of reasoning?
- Can we just "**ask**" the model to reason through steps?
- Kojima et al., 2022: "LLMs are decent **zero-shot reasoners** by simply adding "Let's think step by step" before each answer."

Zero-shot Chain of Thought

Kojima et al., 2022

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

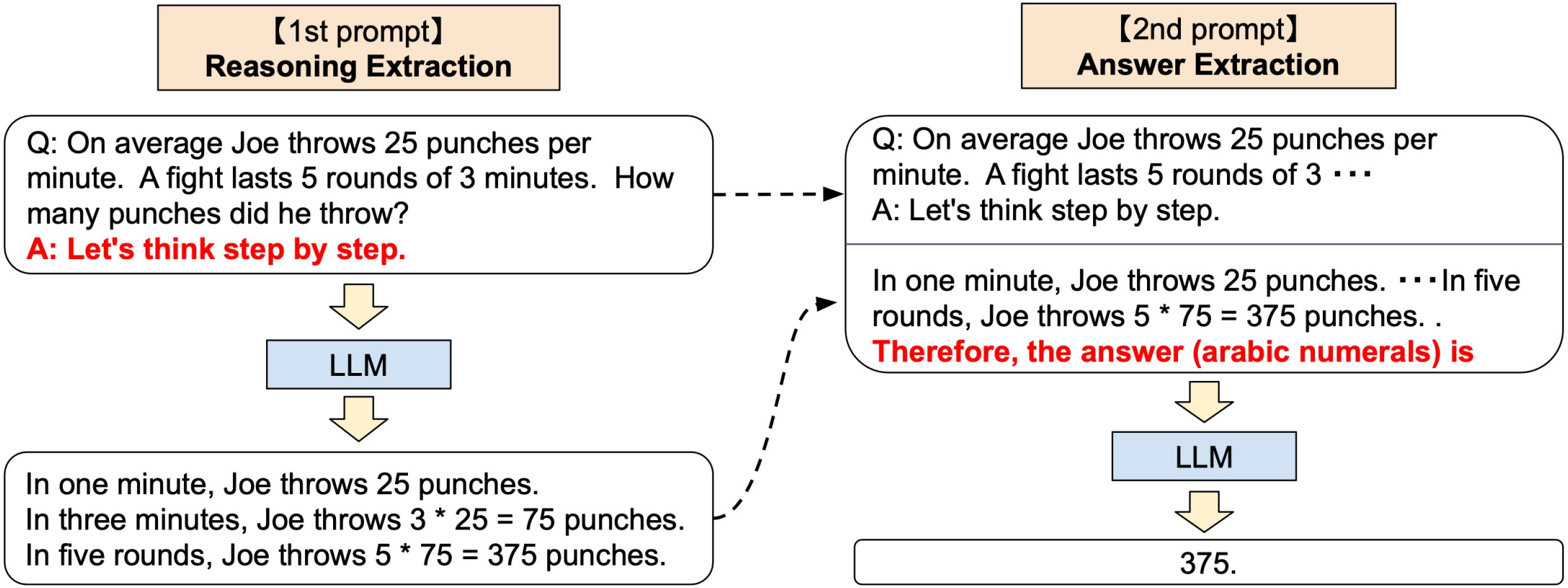
(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Detailed Method of CoT (Kojima et al., 2022)



- First use the first “reasoning” prompt to extract a full reasoning path from a language model
- Then use the second “answer” prompt to extract the answer

Zero-shot CoT performance

Kojima et al., 2022

- Greatly outperform zero-shot; Manual CoT (with few-shot) is still better

	MultiArith	GSM8K
Zero-Shot	**17.7**	**10.4**
Few-Shot (2 samples)	33.7	15.6
Few-Shot (8 samples)	33.8	15.6
Zero-Shot-CoT	**78.7**	**40.7**
Few-Shot-CoT (2 samples)	84.8	41.3
Few-Shot-CoT (4 samples : First) (*1)	89.2	-
Few-Shot-CoT (4 samples : Second) (*1)	90.5	-
Few-Shot-CoT (8 samples)	93.0	48.7
Zero-Plus-Few-Shot-CoT (8 samples) (*2)	**92.8**	**51.5**

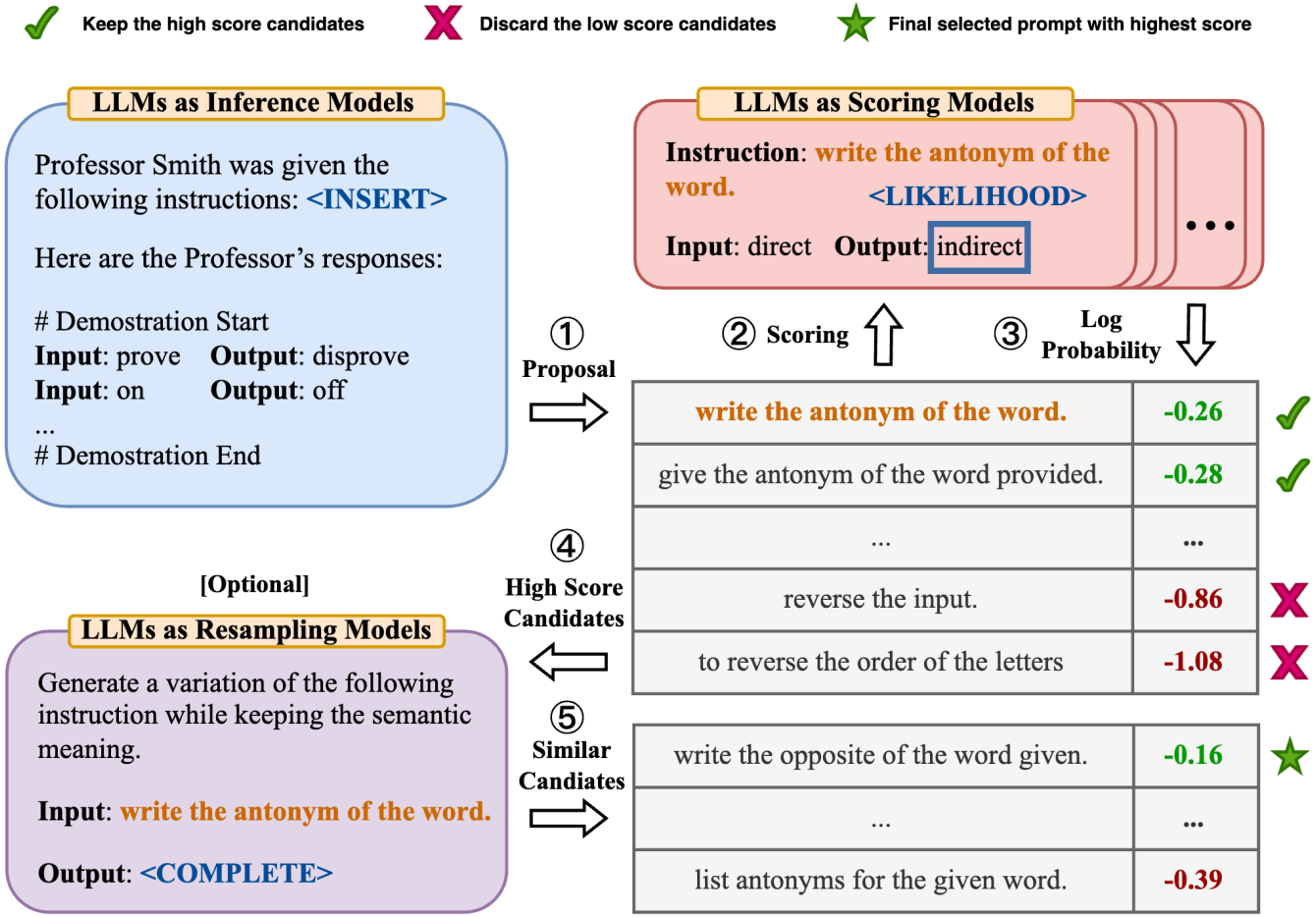
Few shot CoT is still better

How does prompt selection affect 0-S CoT?

No.	Category	Template	Accuracy
1	instructive	Let's think step by step.	**78.7**
2		First, (*1)	77.3
3		Let's think about this logically.	74.5
4		Let's solve this problem by splitting it into steps. (*2)	72.2
5		Let's be realistic and think step by step.	70.8
6		Let's think like a detective step by step.	70.3
7		Let's think	57.5
8		Before we dive into the answer,	55.7
9		The answer is after the proof.	45.7
10	misleading	Don't think. Just feel.	18.8
11		Let's think step by step but reach an incorrect answer.	18.7
12		Let's count the number of "a" in the question.	16.7
13		By using the fact that the earth is round,	9.3
14	irrelevant	By the way, I found a good restaurant nearby.	17.5
15		Abrakadabra!	15.5
16		It's a beautiful day.	13.1
-		(Zero-shot)	17.7

Kojima et al., 2022

What's the best CoT prompt?



- Zhou et al., 2023: LLMs are human level prompt engineers
- **Automatic Prompt Engineer (APE)**: Automatic instruction generation and selection

1. It generates several instruction candidates
2. Executes them using the target model, and selects the most appropriate instruction based on the evaluation score

Best prompt from APE

Zhou et al., 2023

No.	Category	Zero-shot CoT Trigger Prompt	Accuracy
1	APE	Let's work this out in a step by step way to be sure we have the right answer.	**82.0**
2	Human-Designed	Let's think step by step. (*1)	78.7
3		First, (*2)	77.3
4		Let's think about this logically.	74.5
5		Let's solve this problem by splitting it into steps. (*3)	72.2
6		Let's be realistic and think step by step.	70.8
7		Let's think like a detective step by step.	70.3
8		Let's think	57.5
9		Before we dive into the answer,	55.7
10		The answer is after the proof.	45.7
-		(Zero-shot)	17.7

Take-home message from zero-shot CoT

- There is enormous zero-shot knowledge hidden inside LLMs
- LLMs can be seen as general-purpose computers that execute programs specified by prompts (in natural language)
- Prompt engineering process can be automated

Downside of Prompting

- **Inefficiency**: The prompt needs to be processed every time to make a prediction.
- **Suboptimal performance**: Prompting generally performs worse than fine-tuning
- **Sensitivity**: to the wording of the prompt (Webson & Pavlick, 2022), order of examples (Zhao et al., 2021; Lu et al., 2022), etc.
- **Lack of clarity**: about what the model learns from the prompt.
  - Even random labels work! (Zhang et al., 2022; Min et al., 2022)

Overview

- In-Context Learning
- Prompting
- **Question Answering (QA)**
  - What is QA?
  - Information Retrieval; Tf-idf
  - Retriever-based QA; Datasets
    - Answer Span Extraction
    - Retrieval-Augmented Generation

What is Question Answering?

- To build a system that **automatically** answer questions posed by human in natural language

"The Ultimate Question Of Life, The Universe, and Everything"

"42"

(from movie *Hitchhiker's Guide to the Galaxy*)

QA System focuses on *factoid* questions

- **factoid questions**: Questions that can be answered with simple facts expressed in short texts
- Ex. 1: Where is the Louvre Museum located?
- Ex. 2: What is the distance from Moon to Earth?

- One way: to directly ask a large language model (LLM)
  - Using prompts: "Q: What is the distance from Moon to Earth? A: "
- **Problems**:
- LLMs hallucinate; not calibrated
- No access to proprietary/private/personal data: email, private documents, …

Current Solution to QA

- Two-stage **retriever/reader** model

- Stage 1. Retriever algorithms: Use information retrieval (IR) to retrieve relevant documents

- Stage 2. Reader algorithms: Either **extract** or **generate** an answer

Brief Overview of Information Retrieval (IR)

- **Information retrieval, IR**: Retrieval of all kinds of media based on user information needs. IR system ≈ **search engine**
- We focus on **ad hoc (临时) retrieval**: a user poses a query to an IR system, which then returns an ordered set of documents from some collection

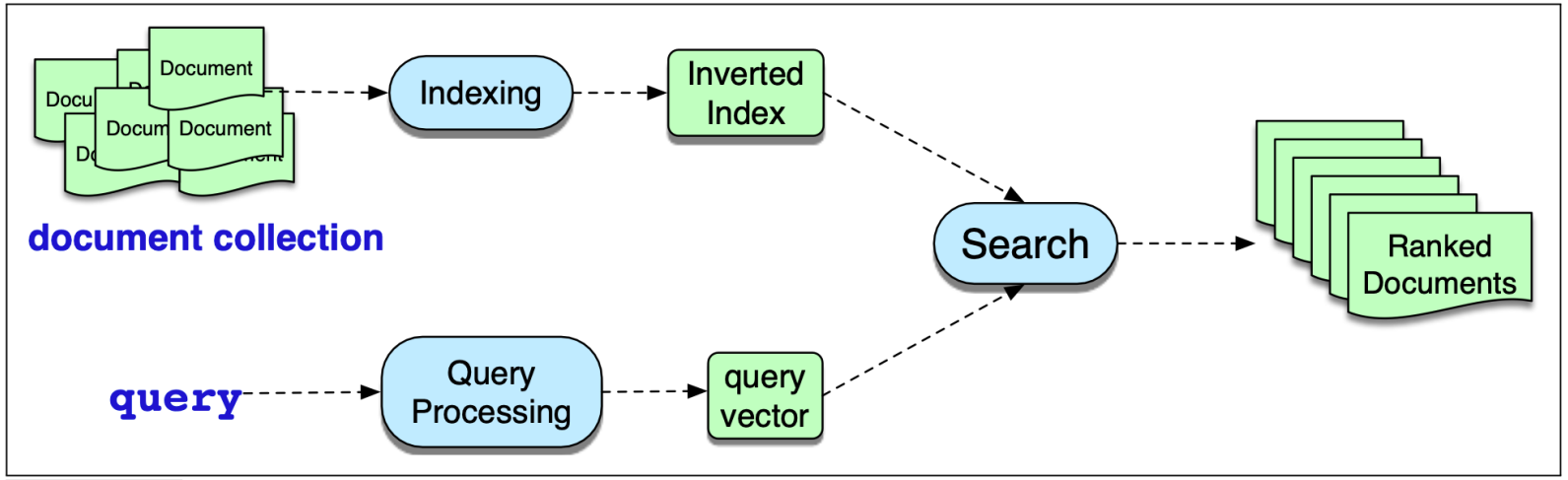


Figure 14.1 The architecture of an ad hoc IR system.

Query: a user's information need expressed as a set of **terms**

Term refers to a word/phrase in a collection of documents

Figure from SLP3, Ch 14

How to match a document a query?

- Compute a term weight for each document term
- Common method: **tf-idf** and BM25
  - **tf**: term frequency
  - **idf**: inverse document frequency
- tf-idf ≜ tf × idf (product of the two)

term t; document d

$$\text{tf}_{t,d} = \begin{cases} 1+\log_{10}\text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **tf**: words that occur more often in a document are likely to be informative about the document's content
- Use the $\log_{10}$ of word frequency count rather than raw count
- Why? A word appearing 100 times doesn't make it 100 times more likely

Tf-idf

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

term t; document d

term occurs 0 times in document: tf = 0
term occurs 1 times in document: tf = 1
term occurs 10 times in document: tf = 2, …

- **document frequency** df_t of a term t is the number of documents it occurs in
- Terms that occur in only **a few** documents are useful for discriminating those documents from the rest of the collection;
- terms that occur across the entire collection aren't as helpful (*the, a, an,* …)
- **inverse document frequency** or **idf** is defined as:

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

N: total number of documents
The fewer documents in which t occurs, the higher idf_t

Inverse document frequency example

- Some idf values for some words in the corpus of Shakespeare plays

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Extremely informative words that occur in only one play like *Romeo*

good or *sweet* tare completely non-discriminative since they occur in all 37 plays

Scoring with tf-idf

- We can score document d by the cosine of its vector $\vec{\boldsymbol{d}}$ with the query vector $\vec{\boldsymbol{q}}$:

$$\text{score}(q,d) = \cos(\vec{\boldsymbol{q}},\vec{\boldsymbol{d}}) = \frac{\vec{\boldsymbol{q}}\cdot\vec{\boldsymbol{d}}}{|\vec{\boldsymbol{q}}|\cdot|\vec{\boldsymbol{d}}|}$$

- in which $\vec{\boldsymbol{q}}$ and $\vec{\boldsymbol{d}}$ are vectors of query length n, whose values are the **tf-idf** values (normalized):

$$\vec{\boldsymbol{q}} = \frac{[\text{tfidf}(t_1,q),\ldots,\text{tfidf}(t_n,q)]}{\sqrt{\sum_{t\in q}\text{tfidf}^2(t,q)}}$$

$$\vec{\boldsymbol{d}} = \frac{[\text{tfidf}(t_1,d),\ldots,\text{tfidf}(t_n,d)]}{\sqrt{\sum_{t\in d}\text{tfidf}^2(t,d)}}$$

$$\text{score}(q,d) = \sum_{t_i\in q}\frac{\text{tfidf}(t_i,q)}{\sqrt{\sum_{t\in q}\text{tfidf}^2(t,q)}}\cdot\frac{\text{tfidf}(t_i,d)}{\sqrt{\sum_{t\in q}\text{tfidf}^2(t,q)}}$$

Tf-idf scoring example

- A collection of 4 nano documents

Query:	sweet love
Doc 1:	Sweet sweet nurse! Love?
Doc 2:	Sweet sorrow
Doc 3:	How sweet is love?
Doc 4:	Nurse!

Query vector $\vec{q} = (0.383, 0.924)$

	Query					
word	**cnt**	**tf**	**df**	**idf**	**tf-idf**	**n'lized** = tf-idf/$\|q\|$
sweet	1	1	3	0.125	0.125	0.383
nurse	0	0	2	0.301	0	0
love	1	1	2	0.301	0.301	0.924
how	0	0	1	0.602	0	0
sorrow	0	0	1	0.602	0	0
is	0	0	1	0.602	0	0
$\|q\| = \sqrt{.125^2 + .301^2} = .326$						

Tf-idf scoring example

Query vector $\vec{q} = (0.383, 0.924)$

Document 1					
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**× q**
sweet	2	1.301	0.163	0.357	**0.137**
nurse	1	1.000	0.301	0.661	0
love	1	1.000	0.301	0.661	**0.610**
how	0	0	0	0	0
sorrow	0	0	0	0	0
is	0	0	0	0	0
$\lvert d_1 \rvert = \sqrt{.163^2 + .301^2 + .301^2} = .456$					

Document 2					
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**×q**
sweet	1	1.000	0.125	0.203	**0.0779**
nurse	0	0	0	0	0
love	0	0	0	0	**0**
how	0	0	0	0	0
sorrow	1	1.000	0.602	0.979	0
is	0	0	0	0	0
$\lvert d_2 \rvert = \sqrt{.125^2 + .602^2} = .615$					

$\vec{d}_1 = (0.357, 0.661)$

$\text{score}(\vec{q}, \vec{d}_1) = \mathbf{0.747}$

$\vec{d}_2 = (0.203)$

$\text{score}(\vec{q}, \vec{d}_1) = \mathbf{0.0779}$

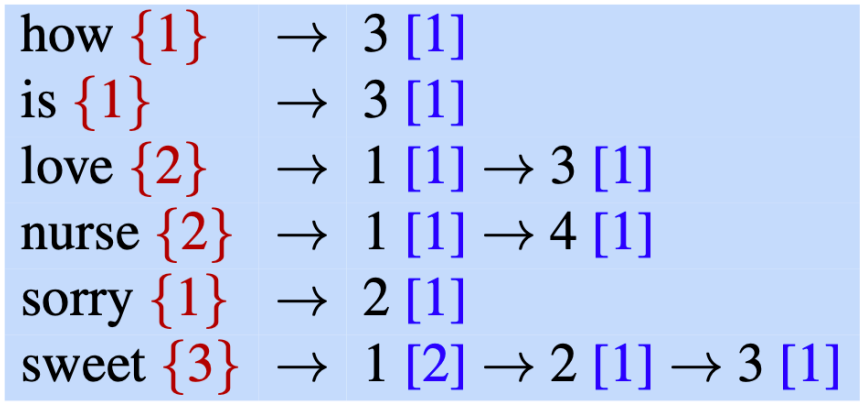
Therefore, d_1 is more relevant

Query: sweet love
Doc 1: Sweet sweet nurse! Love?
Doc 2: Sweet sorrow

Efficient Implementation: Inverted Index

- The basic search problem in IR is to find all documents $d \in C$ that contain a term $q \in Q$
- Use the data structure **inverted index**: given a query term, returns a list of documents that contain the term
- Contains two parts: *dictionary* and *postings*

dictionary: a list of terms, each pointing to a postings list for the term (including document frequency)



posting list: a list of document IDs associated with each term (including term frequency etc.)

IR with Dense Vectors

- **Flaws of TF-IDF -- Vocabulary mismatch problem**: it only works if there is exact overlap of words between the query and document
- **Solution**: Using dense vectors to represent queries/documents
  dating back to Latent semantic indexing vectors, all the way to modern times via encoders like BERT

Present both query q and document d to a single encoder, allowing self-attention to see all tokens from both q and d

$$\boldsymbol{h} = \text{BERT}(q; [\text{SEP}]; d)\ [\text{CLS}]$$

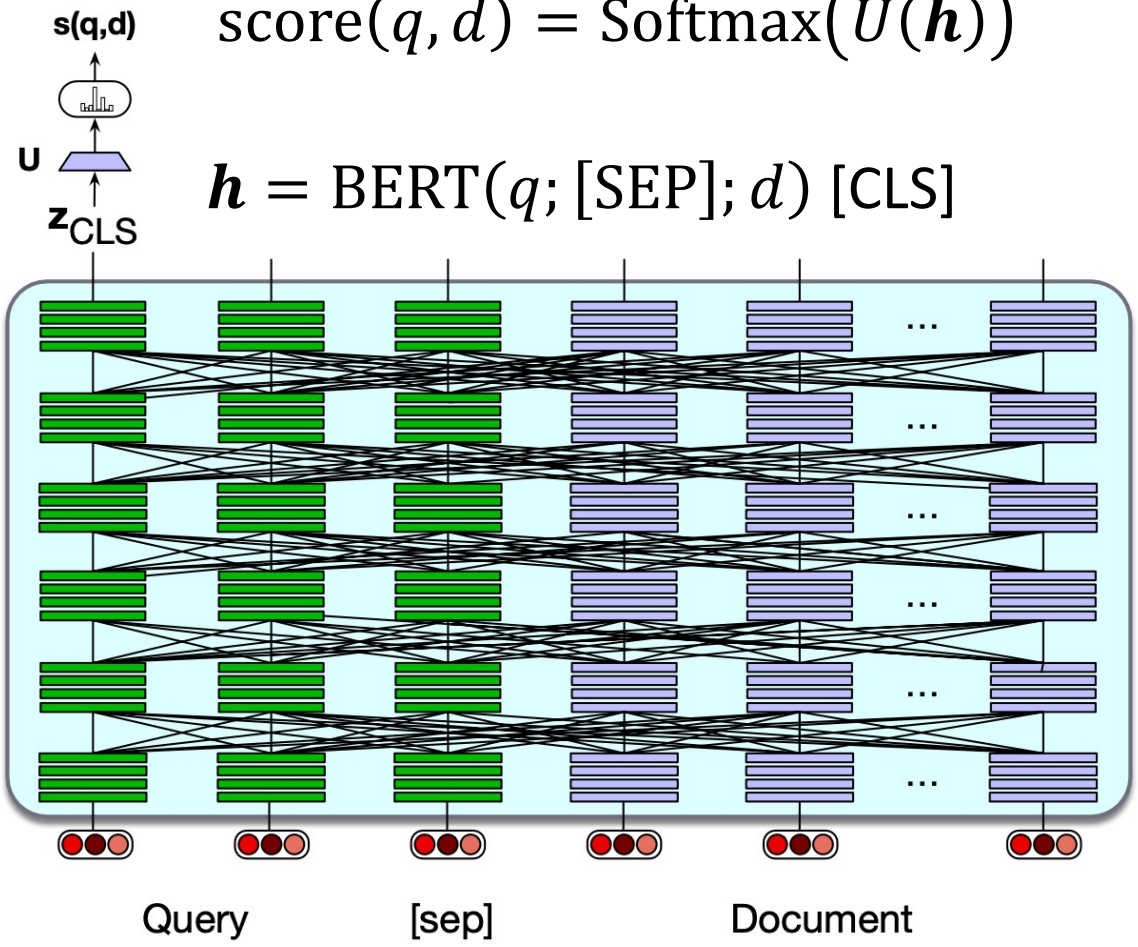
$$\text{score}(q, d) = \text{Softmax}\big(U(\boldsymbol{h})\big)$$

Predict the similarity score between q and d

Single BERT Encoder for IR

$$\text{score}(q, d) = \text{Softmax}\big(U(\boldsymbol{h})\big)$$

$$\boldsymbol{h} = \text{BERT}(q; [\text{SEP}]; d)\ [\text{CLS}]$$



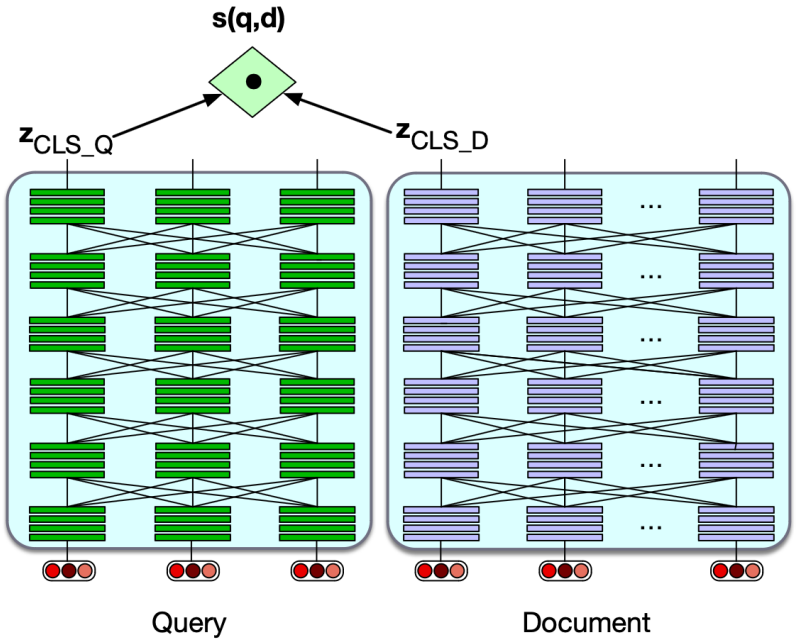
In practice, documents are broken up into smaller passages such as non-overlapping fixed-length chunks of ~100 tokens,

so that the q and d and can fit in the BERT 512-token window

Drawback: expense in computation!
Every time we get a query, have to pass every single document through a BERT encoder jointly with the new query!

More Efficient Way: Bi-Encoder

- Two separate encoder models:
  one to encode the query BERT_Q, and one to encode the document, BERT_D
- Encode each document and store the document vectors in advance
- When a query comes in, just encode this query, and compute the dot product between it and each candidate document



$$\boldsymbol{h}_q = \mathrm{BERT}_Q(q)\ [\mathrm{CLS}]$$

$$\boldsymbol{h}_d = \mathrm{BERT}_D(d)\ [\mathrm{CLS}]$$

$$\mathrm{score}(q,d) = \boldsymbol{h}_q \cdot \boldsymbol{h}_d$$

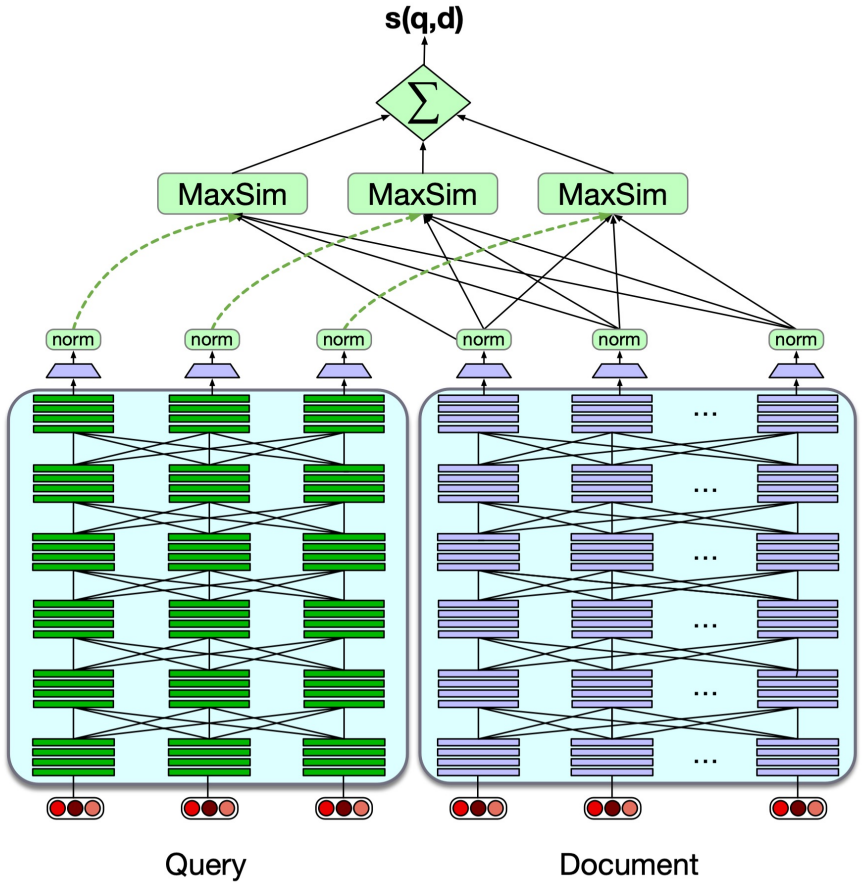
Cheaper in computation, but less accurate, since it does not take full advantage of the interaction between query tokens and document tokens

Alternative: Token-level similarity scores

- **ColBERT** (Khattab et al., 2021) computes the score between a query q and a document d as a sum of maximum similarity (MaxSim) between tokens in q and tokens in d

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

More accurate than the bi-encoder method



Implementation Efficiency

- For dense vector-based IR, efficiency is also an important issue
- since every possible document must be ranked for its similarity to the query
- Bottle-neck: Finding the set of document vectors that have the highest dot product with a query vector -- **nearest neighbor search** problem
- Can be approximated with algorithms like Faiss (Johnson et al., 2017)

Evaluation of IR System

- **Precision**: the fraction of returned documents that are relevant
- **Recall**: the fraction of all relevant documents that are returned

- A system returned **T** ranked documents, a subset **R** of which are relevant
- **U** documents in the collection are relevant to the request

$$Precision = \frac{|R|}{|T|} \qquad Recall = \frac{|R|}{|U|}$$

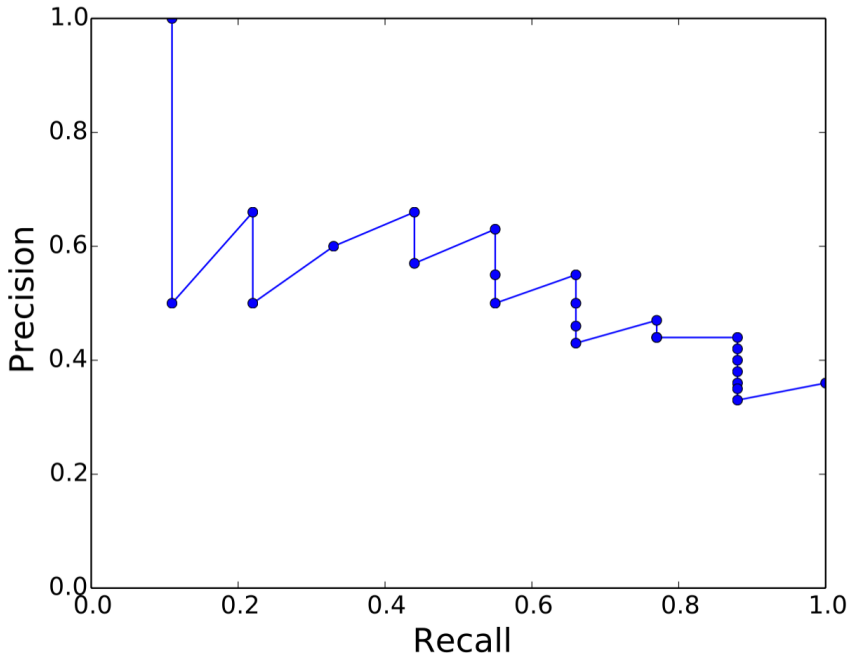
Evaluation of IR System

- More advanced evaluation: How well a system put **relevant** documents higher in the **ranking**
- Rank-specific precision and recall
  - **Recall is non-decreasing**; when a relevant document is encountered, recall increases, and when a non-relevant document is found it remains unchanged.
  - **Precision jumps up and down**.

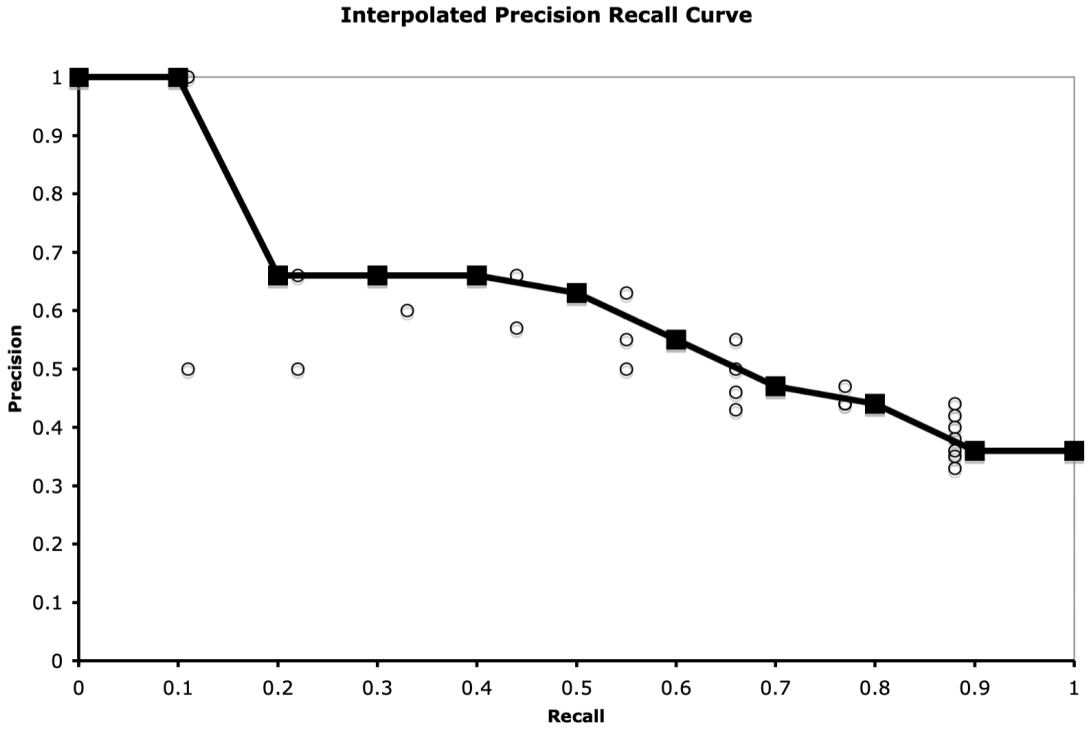
Rank	Judgment	Precision_{Rank}	Recall_{Rank}
1	R	1.0	.11
2	N	.50	.11
3	R	.66	.22
4	N	.50	.22
5	R	.60	.33
6	R	.66	.44
7	N	.57	.44
8	R	.63	.55
9	N	.55	.55
10	N	.50	.55
11	R	.55	.66
12	N	.50	.66
13	N	.46	.66
14	N	.43	.66
15	R	.47	.77
16	N	.44	.77
17	N	.44	.77
18	R	.44	.88
19	N	.42	.88
20	N	.40	.88
21	N	.38	.88
22	N	.36	.88
23	N	.35	.88
24	N	.33	.88
25	R	.36	1.0

Evaluation of IR System

- Precision and recall curves



Interpolated Precision	Recall
1.0	0.0
1.0	.10
.66	.20
.66	.30
.66	.40
.63	.50
.55	.60
.47	.70
.44	.80
.36	.90
.36	1.0



Current Solution to QA

- Two-stage **retriever/reader** model

- Stage 1. Retriever algorithms: Use information retrieval (IR) to retrieve relevant documents
- Stage 2. Reader algorithms: Either extract or generate an answer

Reader

- Extractor: **span extraction** ⇒ find spans of text that answer the question over the retrieved passages
- Generator: **retrieval-augmented generation** ⇒ Take a large pretrained LM, design the prompt based on the retrieved passage, and generate the answer token by token

Two-staged retriever/reader model

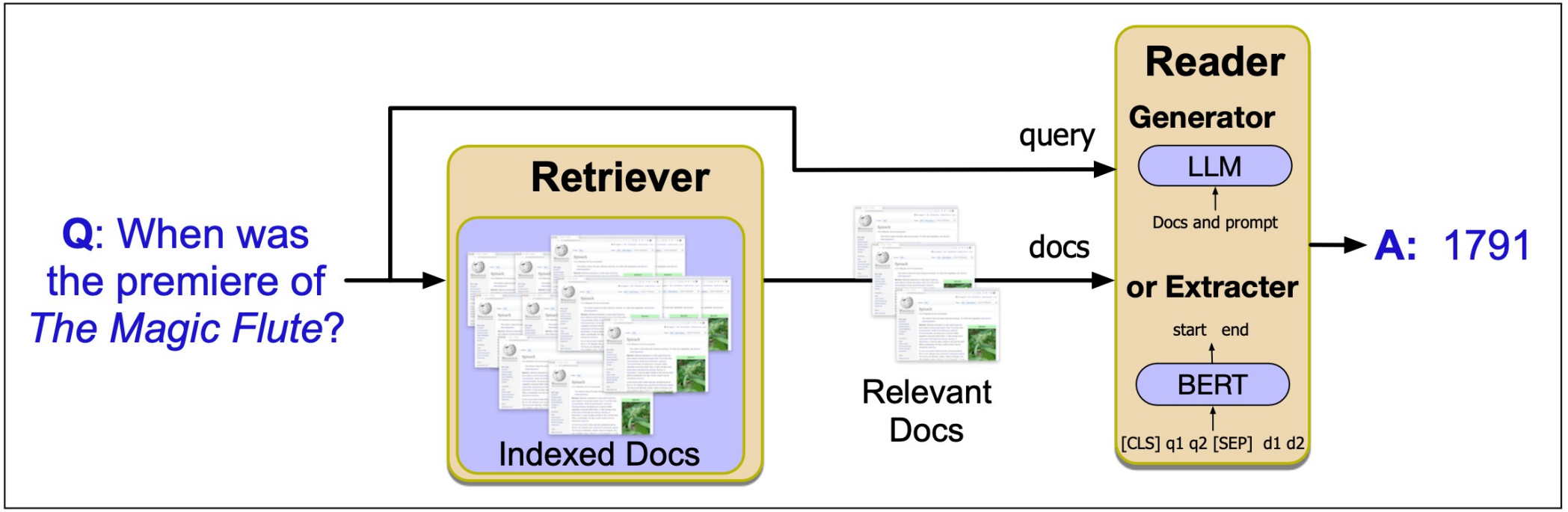


Figure from SLP3, Ch 14

Reader: Answer Span Extraction

Beyoncé Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".
Q: "In what city and state did Beyoncé grow up?" A: "Houston, Texas"
Q: "What areas did Beyoncé compete in when she was growing up?" A: "singing and dancing"
Q: "When did Beyoncé release Dangerously in Love?" A: "2003"

Figure 14.11 A (Wikipedia) passage from the SQuAD 2.0 dataset (Rajpurkar et al., 2018) with 3 sample questions and the labeled answer spans.

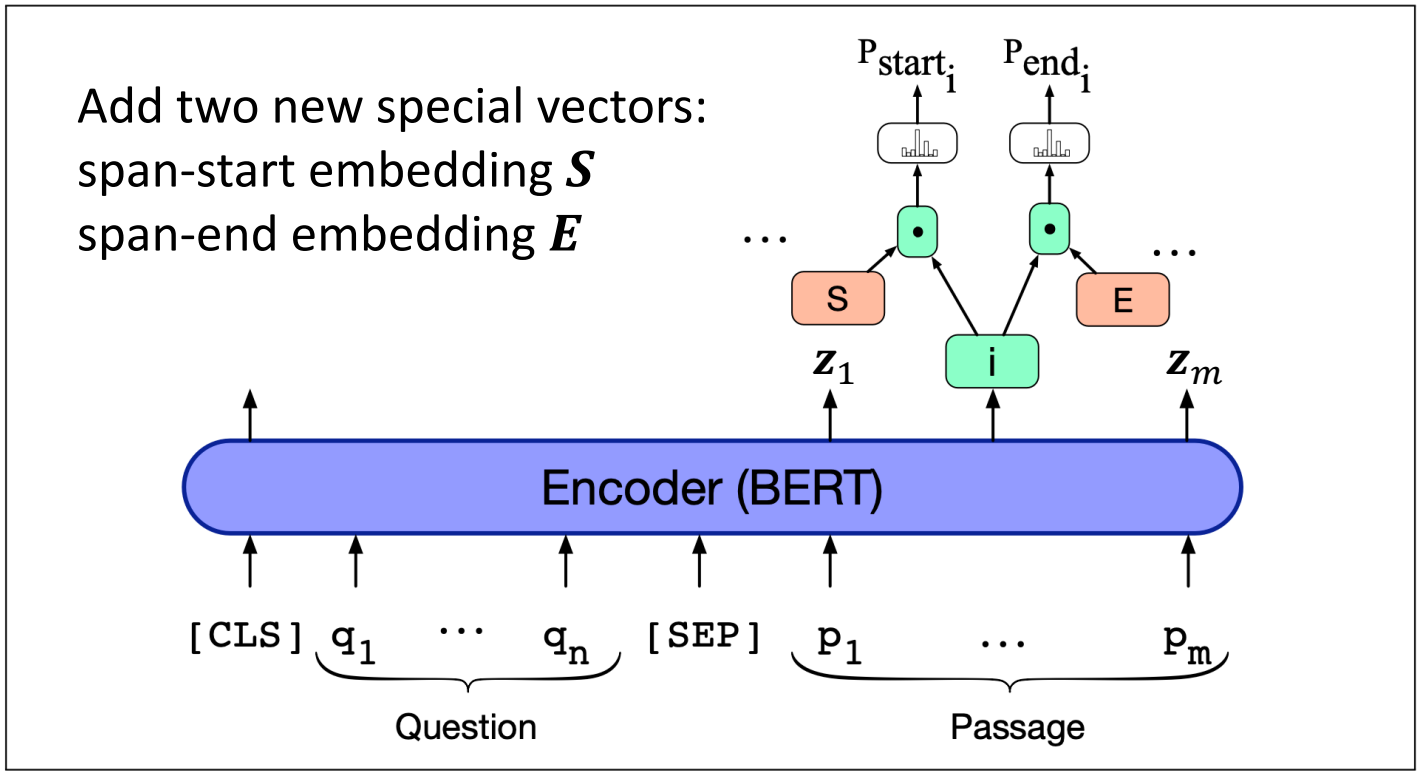
Span labeling task: identify in the passage a span (continuous string of text) that constitutes an answer

Span Labeling

- Given a question q of n tokens $q_1, \ldots, q_n$ and a passage p of m tokens $p_1, \ldots, p_m$
- **Goal**: compute the probability $P(a|q,p)$ of each possible span a is the answer
- Span a starts at position a_s and ends at a_e, then estimate the probability by:
- $P(a|q,p) = P_{\text{start}}(a_s|q,p)P_{\text{end}}(a_e|q,p)$
- For each token p_i in passage, compute two probabilities:
  - $P_{\text{start}}(i) \Rightarrow p_i$ is the start of answer span
  - $P_{\text{end}}(i) \Rightarrow p_i$ is the end of answer span
- Goal becomes:

$$\max_{i,j \in [1,m]; j \geq i} P_{\text{start}}(i) P_{\text{end}}(j)$$

Span Labeling



$$P_{\text{start}}(i) = \frac{\exp(\boldsymbol{S} \cdot \boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{S} \cdot \boldsymbol{z}_j)}$$

$$P_{\text{end}}(i) = \frac{\exp(\boldsymbol{E} \cdot \boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{E} \cdot \boldsymbol{z}_j)}$$

Goal: $\max_{i,j \in [1,m]; j \geq i} P_{\text{start}}(i) P_{\text{end}}(j)$

$$\max_{i,j \in [1,m]; j \geq i} \boldsymbol{S} \cdot \boldsymbol{z}_i + \boldsymbol{E} \cdot \boldsymbol{z}_j$$

The score for candidate span from position i to j

What if answer is not contained in passage?

- Many datasets contain (question, passage) pairs in which the answer is not contained in the passage
- Need a way to estimate this “none” probability
- Done by treating [CLS] token as the answer, i.e., a_s and a_e all point to [CLS]

Retrieval-based QA Datasets

- Reading comprehension datasets containing tuples of (*passage, question, answer*)
- Including *passage* eliminates the need for information retrieval
- A system can be trained to predict a span in passage as answer, given a question

- Stanford Question Answering Dataset (**SQuAD**) (Rajpurkar et al., 2016)
  - Over 150,000 questions
  - Passage from Wikipedia; SQuAD 2.0 includes unanswerable questions

Retrieval-based QA Datasets

- **HotpotQA** dataset (Yang et al., 2018): Showing crowd workers multiple context documents and asked to create questions that require reasoning
- Both SQuAD and HotpotQA are created by annotators who have first read the passage may make their questions easier to answer

- Datasets from questions that were not written with a passage in mind
- **TriviaQA** dataset (Joshi et al., 2017) Trivia: 琐事，娱乐和消遣
  - 94K questions written by trivia enthusiasts, with supporting documents (Wikipedia and web)
  - 650K question-answer-evidence triples
  - Relatively complex, compositional questions
  - Requires more cross sentence reasoning

Retrieval-based QA Datasets

- **MS MARCO** (Microsoft Machine Reading Comprehension) (Nguyen et al., 2016)
  - 1 million real anonymized questions from Microsoft Bing query logs
  - with a human generated answer and 9 million passages
  - https://microsoft.github.io/msmarco/

- **Natural Questions** dataset (Kwiatkowski et al., 2019)
  - Anonymized queries to the Google search engine
  - Annotators are presented a query, along with a Wikipedia page from the top 5 search results
  - To annotate a paragraph-length **long** answer and a **short** span answer, or mark **null** if the text doesn't contain the paragraph.
  - https://ai.google.com/research/NaturalQuestions

Evaluation of Retrieval-based QA

- QA is commonly evaluated using **mean reciprocal rank**, or **MRR**

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

- If the system returned 5 answers, but the first 3 are wrong, then the hightest-ranked correct answer is ranked 4^{th} and thus the reciprocal rank is $\frac{1}{4}$

- Alternative methods:
- **Exact match**: The % of predicted answers that match the gold answer exactly
- **F-1 score**: average F-1 over all questions

Current Solution to QA

- Two-stage **retriever/reader** model

- Stage 1. Retriever algorithms: Use information retrieval (IR) to retrieve relevant documents
- Stage 2. Reader algorithms: Either extract or generate an answer

Reader

- Extractor: **span extraction** ⇒ find spans of text that answer the question over the retrieved passages
- Generator: **retrieval-augmented generation** ⇒ Take a large pretrained LM, design the prompt based on the retrieved passage, and generate the answer token by token

Reader: Retrieval-Augmented Generation (RAG)

- Cast the QA task as word prediction: feeding the LM a question and a token like "A: " -- suggesting the answer should come next

Q: Who wrote the book ‘‘The Origin of Species"? A: $x_1, x_2, \ldots, x_n$

[Q:] q [A:]

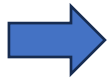
Conditional generation that optimizes: $P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | [Q:]; q; [A]; x_{<i})$

Problems: hallucination; no access to proprietary data

Retrieval-Augmented Generation (RAG)

- Idea: Conditioning on the retrieved passages as part of the prefix perhaps with some prompt like "Based on these text, answer this question"

Query q



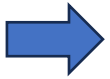
Retrieved passages $R(q)$

```
retrieved passage 1

retrieved passage 2

...

retrieved passage n
```



Append prompt text:

Based on these texts, answer this question: Q: Who wrote the book ‘‘The Origin of Species"? A:

Feed to LLM for generation

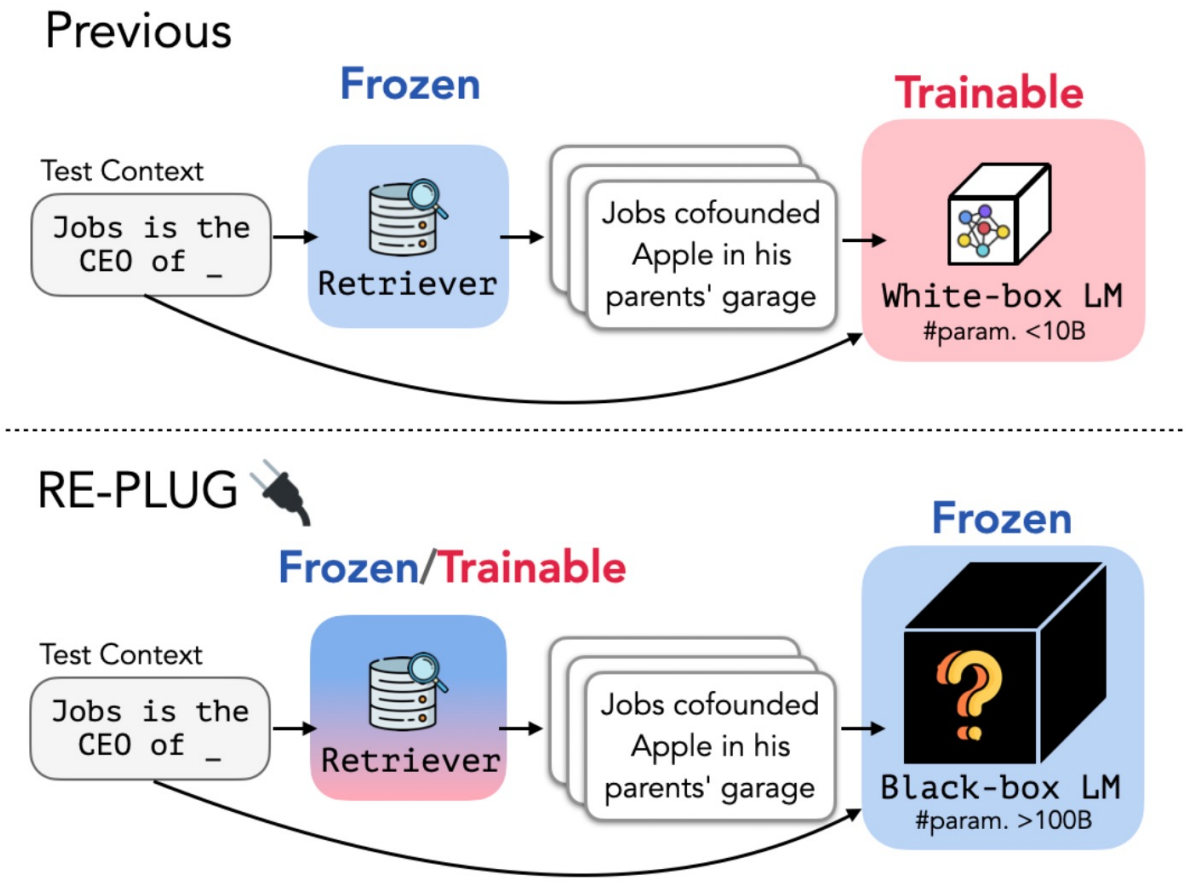
$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | R(q); \text{prompt}; [Q:]; q; [A]; x_{<i})$$

RAG details

- Just like span-based extractor, RAG requires a successful retriever, in two-stage setting as well
- **Multi-hop** architecture may be needed: a query q is used to retrieve documents, which are then appended to original q for a *second* stage retrieval
- Detailed prompt engineering is needed
- When combining private data with public data, externally hosted LLMs may be concerned

RAG Example: RePLUG

Shi et al., 2023

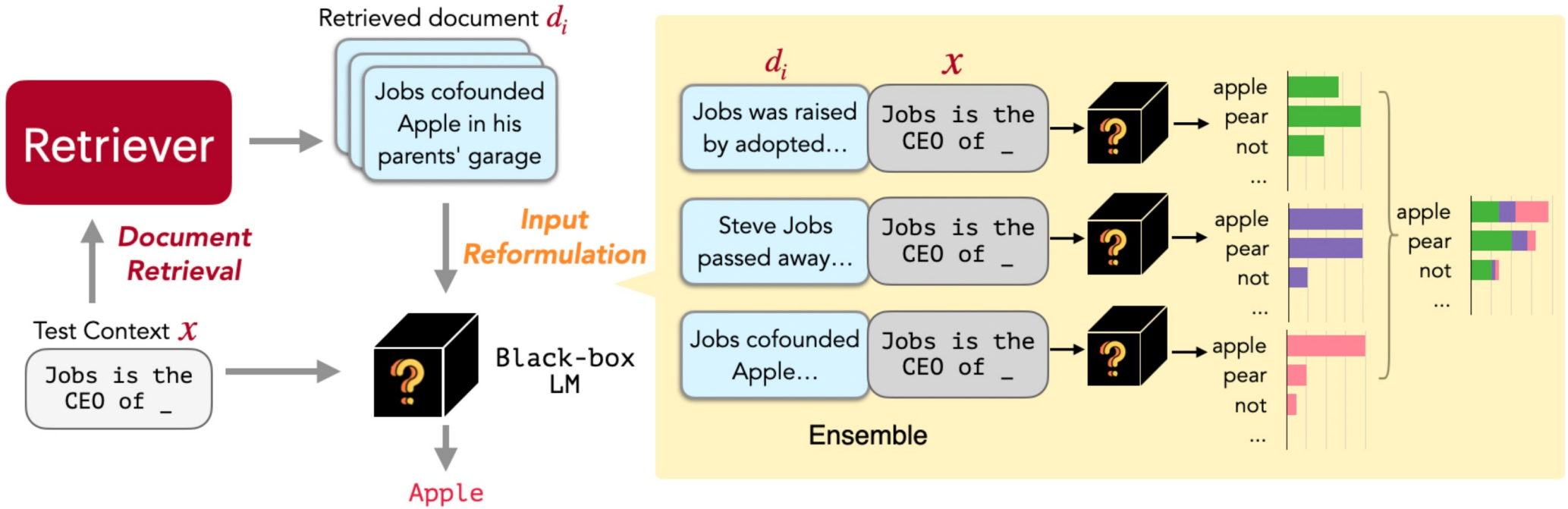


Treats the LM as a black box and augments it with a tuneable retrieval model

- Simply prepends retrieved documents to the input
- LM can be used to supervise the retrieval model

RAG Example: RePLUG

Shi et al., 2023



- Given an input context, REPLUG first retrieves a small set of relevant documents from an external corpus
- Then it prepends each document separately to the input context and ensembles output probabilities from different passes

References

- Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., & Deng, L. (2016). Ms marco: A human-generated machine reading comprehension dataset.
- Chen, D., Fisch, A., Weston, J., & Bordes, A. (2017). Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*.
- Joshi, M., Choi, E., Weld, D. S., & Zettlemoyer, L. (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.
- Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, *7*(3), 535-547.
- Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... & Yih, W. T. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.
- Shi, W., Min, S., Yasunaga, M., Seo, M., James, R., Lewis, M., ... & Yih, W. T. (2023). Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*.