

Natural Language Processing

AI51701/CSE71001

Lecture 19

11/23/2023

Instructor: Taehwan Kim

Announcement

❑ Assignment 3 was released

- Due: Dec. 3 at 11:59pm KST

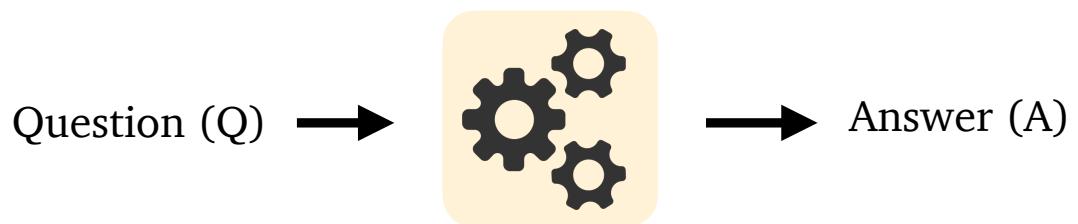
❑ Final project timeline

- Report due on 12/11,
Presentations on 12/12 and 12/14
- Using the last lecture (12/5) for team meeting time

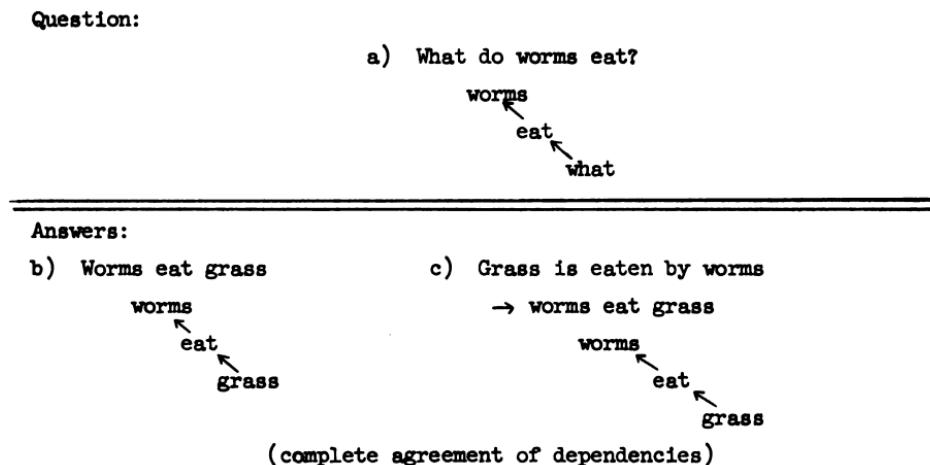
Question Answering

What is question answering?

- ❑ The goal of question answering is to build systems that **automatically** answer questions posed by humans in a **natural language**



- ❑ The earliest QA systems dated back to 1960s!
(Simmons et al., 1964)



Question answering: a taxonomy

❑ What information source does a system build on?

- A text passage, all Web documents, knowledge bases, tables, images..

❑ Question type

- Factoid vs non-factoid, open-domain vs closed-domain, simple vs compositional, ..

❑ Answer type

- A short segment of text, a paragraph, a list, yes/no, ...



Lots of practical applications

Google Where is the deepest lake in the world? X |

All Maps Images News Videos More Settings Tools

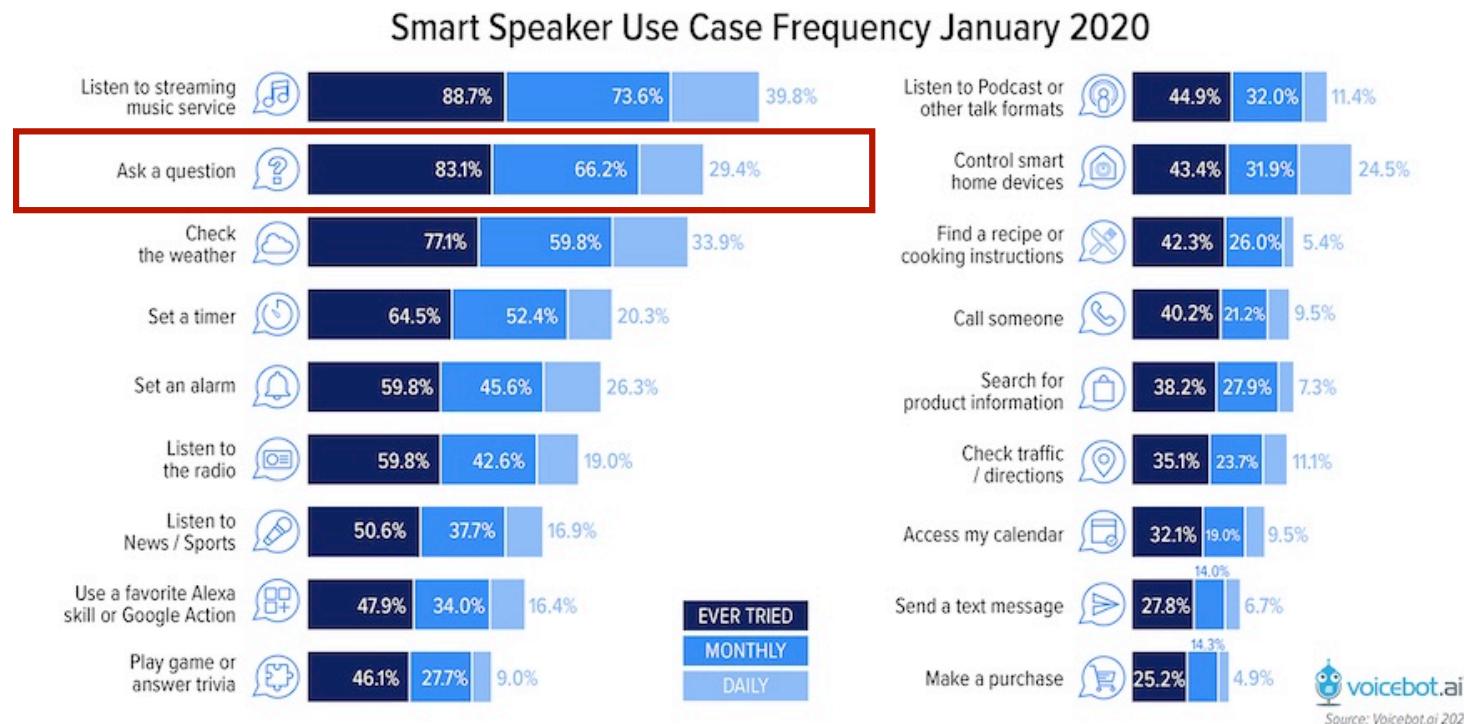
About 21,100,000 results (0.71 seconds)



Siberia

Lake **Baikal**, in Siberia, holds the distinction of being both the deepest lake in the world and the largest freshwater lake, holding more than 20% of the unfrozen fresh water on the surface of Earth.

Lots of practical applications

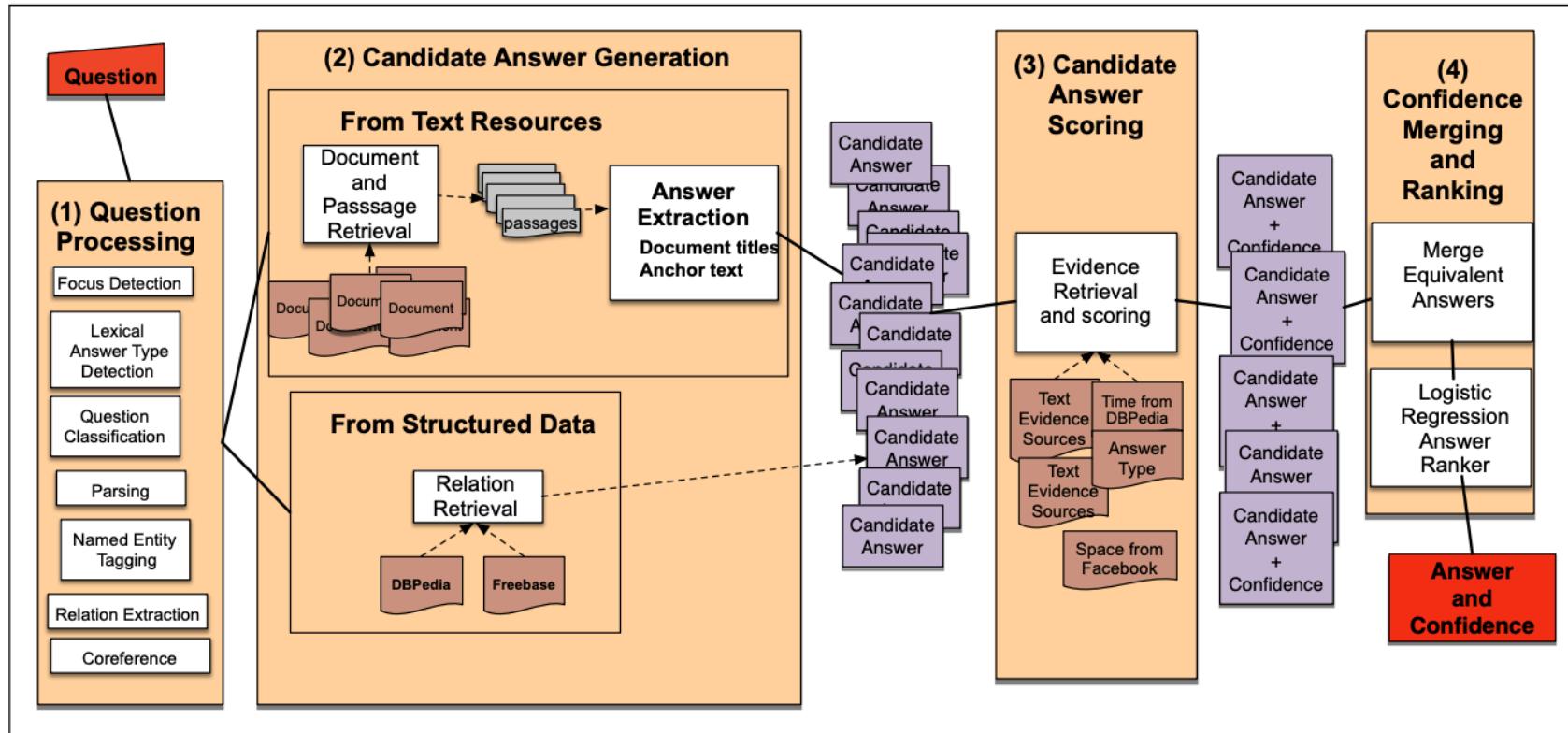


2011: IBM Watson beat Jeopardy champions



IBM Watson defeated two of Jeopardy's greatest champions in 2011

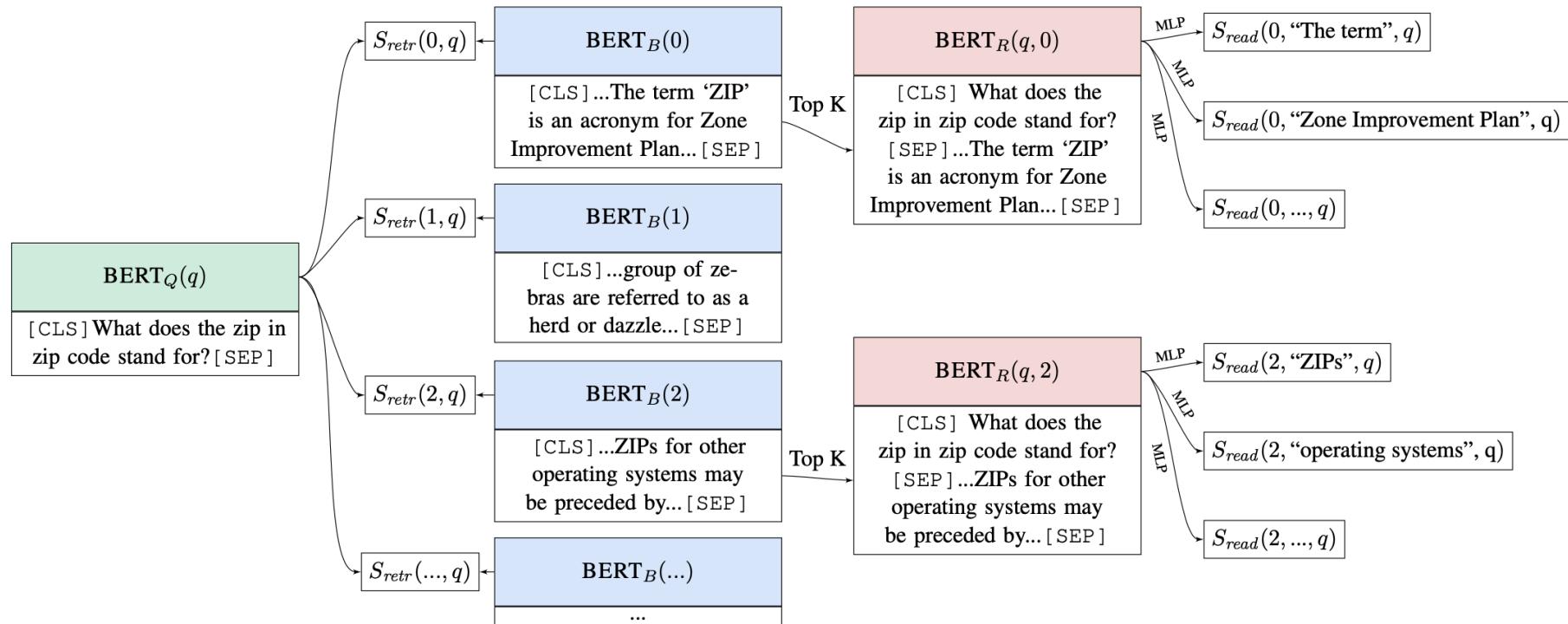
2011: IBM Watson beat Jeopardy champions



(1) Question processing, (2) Candidate answer generation, (3) Candidate answer scoring, and (4) Confidence merging and ranking.

Question answering in deep learning era

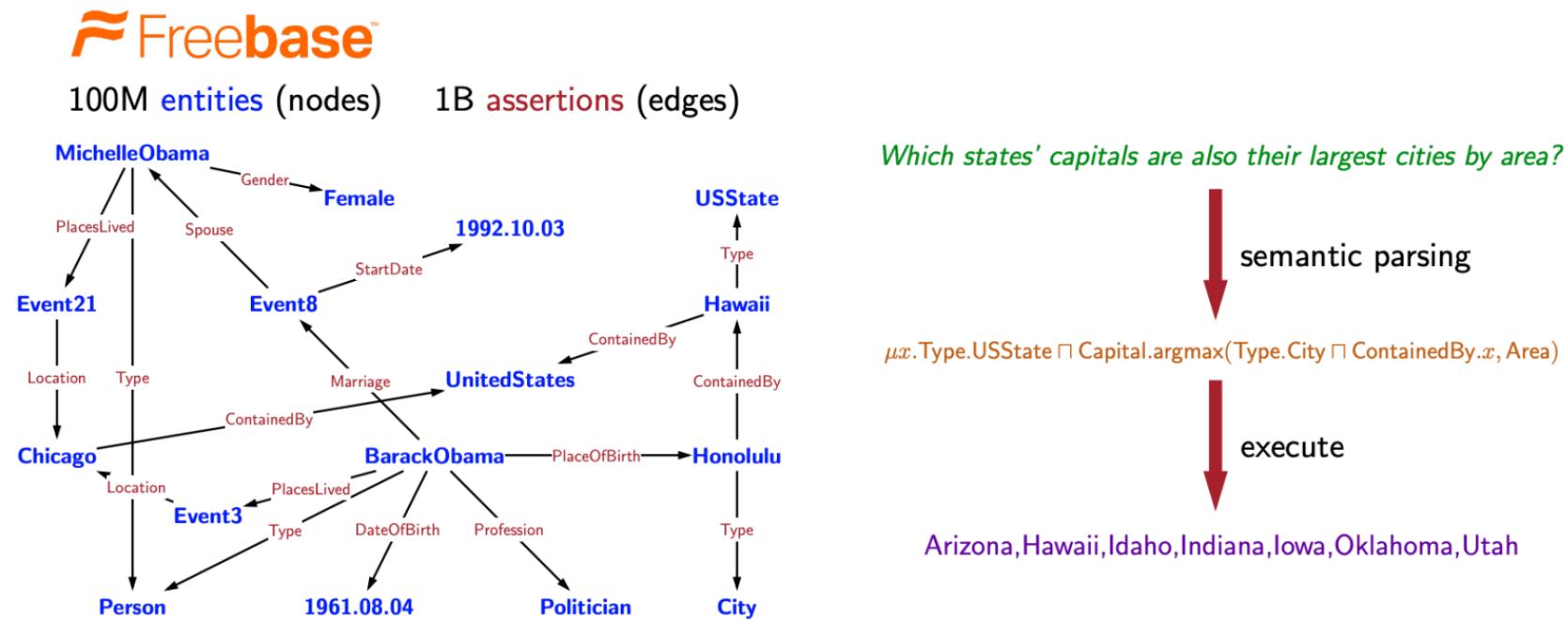
- ❑ Almost all the state-of-the-art question answering systems are built on top of end-to-end training and pre-trained language models (e.g., BERT)!



Beyond textual QA problems

- Today, we will mostly focus on how to answer questions based on **unstructured text**.

Knowledge based QA



Beyond textual QA problems

- Today, we will mostly focus on how to answer questions based on **unstructured text**.

Visual QA



What color are her eyes?
What is the mustache made of?



How many slices of pizza are there?
Is this a vegetarian pizza?

Reading comprehension

❑ **Reading comprehension** = comprehend a passage of text and answer questions about its content (P, Q) → A

Tesla was the fourth of five children. He had an older brother named Dane and three sisters, Milka, Angelina and Marica. Dane was killed in a horse-riding accident when Nikola was five. In 1861, Tesla attended the "Lower" or "Primary" School in Smiljan where he studied German, arithmetic, and religion. In 1862, the Tesla family moved to Gospic, Austrian Empire, where Tesla's father worked as a pastor. Nikola completed "Lower" or "Primary" School, followed by the "Lower Real Gymnasium" or "Normal School."

Q: What language did Tesla study while in school?

A: German

Why do we care about this problem?

- ❑ Useful for many practical applications
- ❑ Reading comprehension is an important testbed for evaluating how well computer systems understand human language
- ❑ Many other NLP tasks can be reduced to a reading comprehension problem:

Information extraction

(Barack Obama, educated_at, ?)

Question: Where did Barack Obama graduate from?

Passage: Obama was born in Honolulu, Hawaii.
After graduating from Columbia University in 1983,
he worked as a community organizer in Chicago.

(Levy et al., 2017)

Semantic role labeling

UCD **finished** the 2006 championship as Dublin champions ,
by **beating** St Vincents in the final .

finished

Who finished something? - UCD
What did someone finish? - the 2006 championship
What did someone finish something as? - Dublin champions
How did someone finish something? - by beating St Vincents in the final

beating

Who beat someone? - UCD
When did someone beat someone? - in the final
Who did someone beat? - St Vincents

(He et al., 2015)

Stanford question answering dataset (SQuAD)

- ❑ 100k annotated (passage, question, answer) triples
 - Large-scale supervised datasets are also a key ingredient for training effective neural models for reading comprehension!
- ❑ Passages are selected from English Wikipedia, usually 100~150 words.
- ❑ Questions are crowd-sourced.
- ❑ Each answer is a short segment of text (or span) in the passage.
 - This is a limitation— not all the questions can be answered in this way!
- ❑ SQuAD still remains the most popular reading comprehension dataset; it is “almost solved” today and the state-of-the-art exceeds the estimated human performance.

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Stanford question answering dataset (SQuAD)

- ❑ Evaluation: exact match (0 or 1) and F1 (partial credit).
- ❑ For development and test sets, 3 gold answers are collected, because there could be multiple plausible answers.
- ❑ We compare the predicted answer to each gold answer (a, an, the, punctuations are removed) and take max scores. Finally, we take the average of all the examples for both exact match and F1.
- ❑ Estimated human performance: EM = 82.3, F1 = 91.2

Q: What did Tesla do in December 1878?

A: {left Graz, left Graz, left Graz and severed all relations with his family}

Prediction: {left Graz and served}

Exact match: $\max\{0, 0, 0\} = 0$

F1: $\max\{0.67, 0.67, 0.61\} = 0.67$

Other question answering datasets

- ❑ TriviaQA: Questions and answers by trivia enthusiasts. Independently collected web paragraphs that contain the answer and seem to discuss question, but no human verification that paragraph supports answer to question
- ❑ Natural Questions: Question drawn from frequently asked Google search questions. Answers from Wikipedia paragraphs. Answer can be substring, yes, no, or NOT_PRESENT. Verified by human annotation.
- ❑ HotpotQA. Constructed questions to be answered from the whole of Wikipedia which involve getting information from two pages to answer a multistep query:

Q: Which novel by the author of “Armada” will be adapted as a feature film by Steven Spielberg?

A: *Ready Player One*

Neural models for reading comprehension

❑ How can we build a model to solve SQuAD?

- We are going to use **passage, paragraph and context**, as well as **question and query** interchangeably

❑ Problem formulation

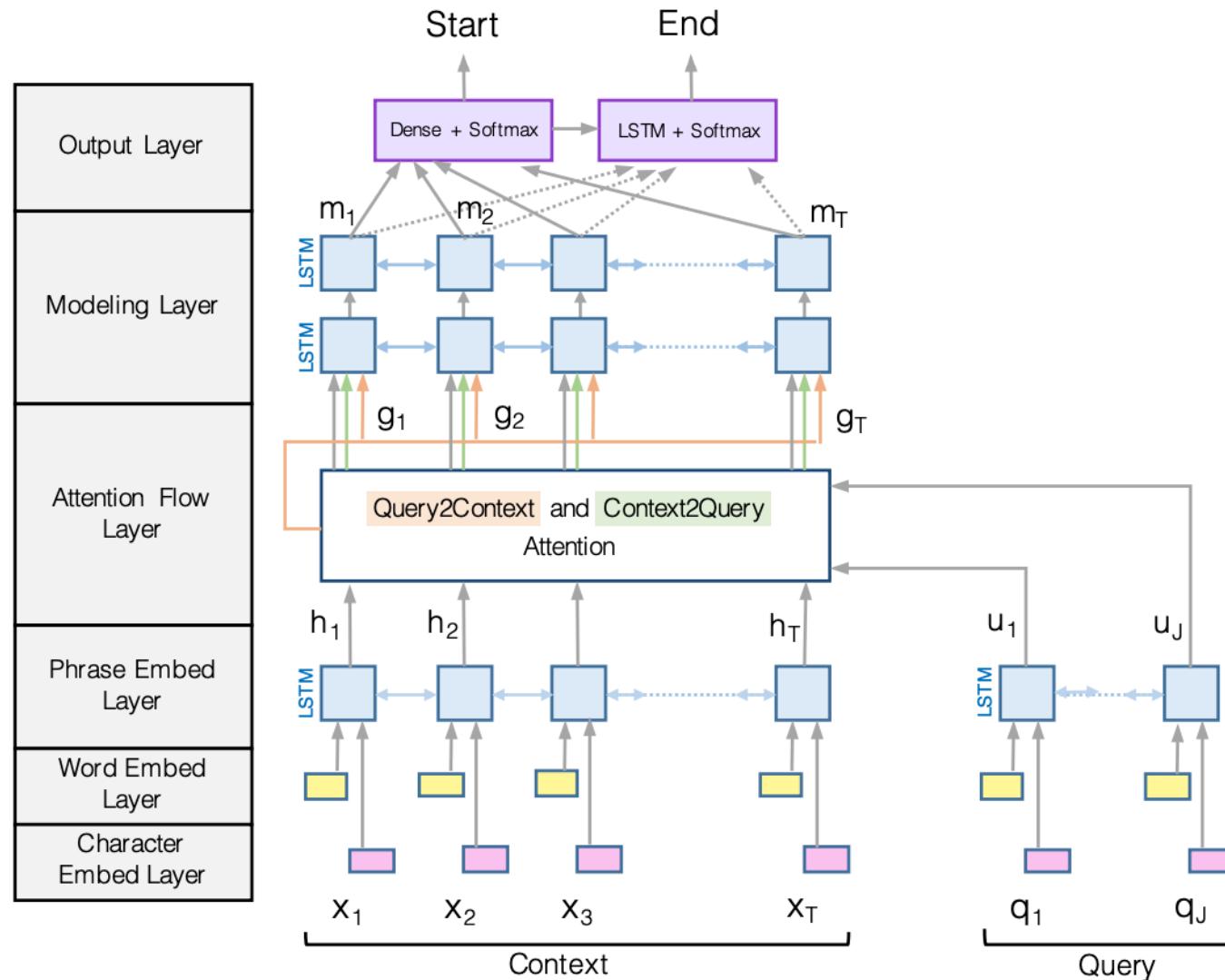
- Input: $C = (c_1, c_2, \dots, c_N)$, $Q = (q_1, q_2, \dots, q_M)$, $c_i, q_i \in V$ $N \sim 100, M \sim 15$
- Output: $1 \leq \text{start} \leq \text{end} \leq N$ answer is a span in the passage

❑ A family of LSTM-based models with attention (2016–2018)

- Attentive Reader (Hermann et al., 2015), Stanford Attentive Reader (Chen et al., 2016), Match- LSTM (Wang et al., 2017), BiDAF (Seo et al., 2017), Dynamic coattention network (Xiong et al., 2017), DrQA (Chen et al., 2017), R-Net (Wang et al., 2017), ReasoNet (Shen et al., 2017)..

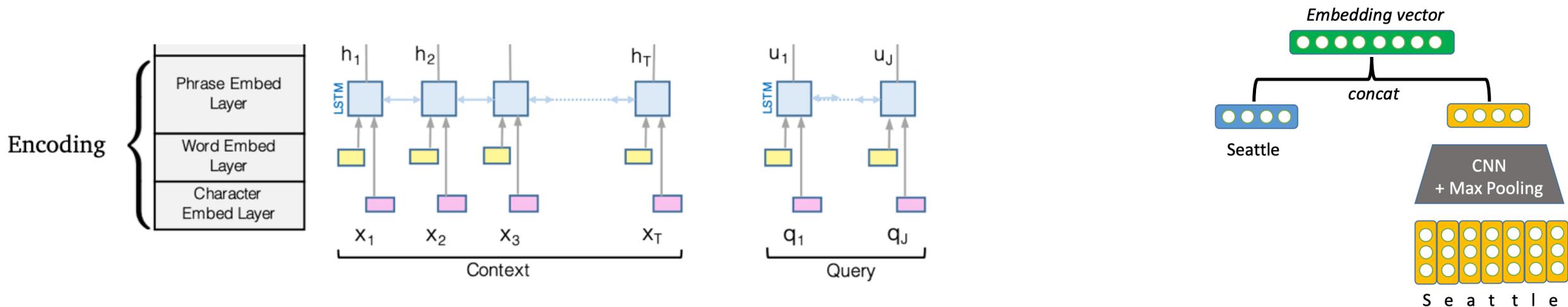
❑ Fine-tuning BERT-like models for reading comprehension (2019+)

BiDAF: the Bidirectional Attention Flow model

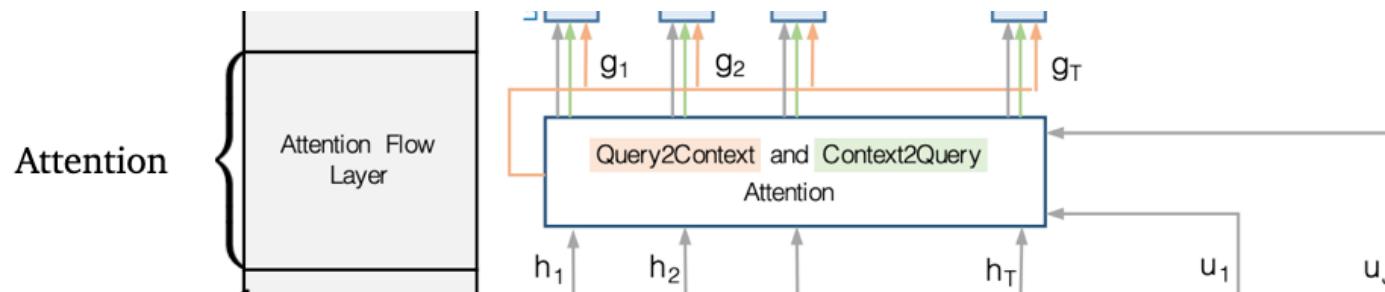


BiDAF: Encoding

- ❑ Use a concatenation of word embedding (GloVe) and character embedding (CNNs over character embeddings) for each word in context and query.
- ❑ Then, use two bidirectional LSTMs separately to produce contextual embeddings for both context and query.



BiDAF: Attention



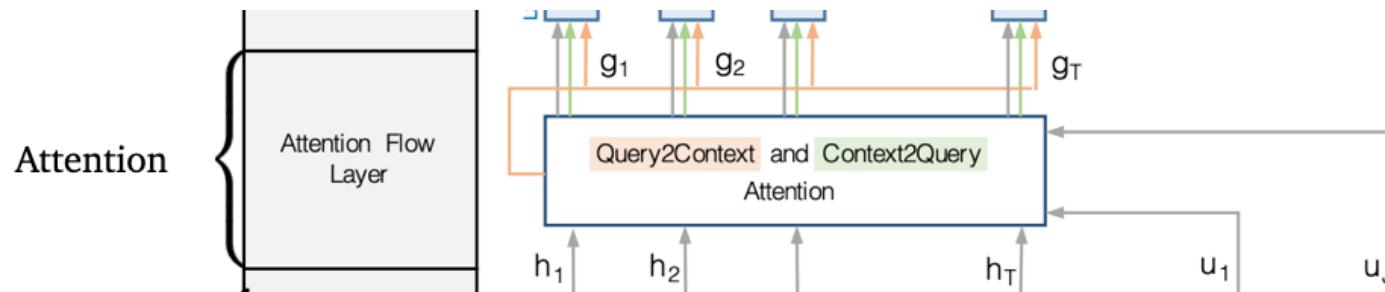
- Context-to-query attention: For each context word, choose the most relevant words from the query words.

Q: Who leads the United States?

C: Barak Obama is the president of the USA.

For each context word, find the most relevant query word.

BiDAF: Attention

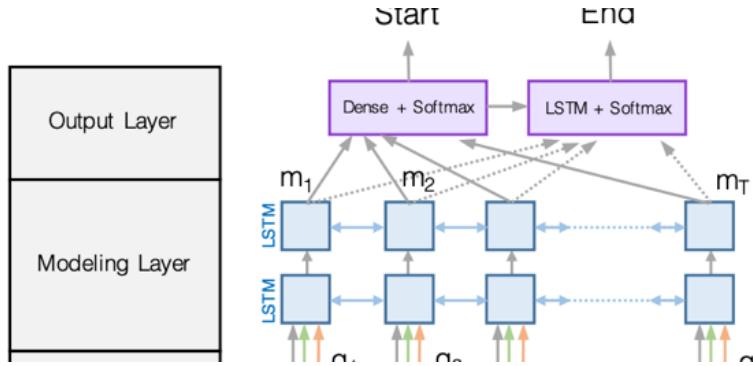


- Query-to-context attention: choose the context words that are most relevant to one of query words.

While Seattle's weather is very nice in summer, its weather is very rainy in winter, making it one of the most gloomy cities in the U.S. LA is ...

Q: Which city is gloomy in winter?

BiDAF: Modeling and output layers



The final training loss is

$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$

- ❑ Modeling layer: pass g_i to another two layers of bi-directional LSTMs.
 - Attention layer is modeling interactions between query and context
 - Modeling layer is modeling interactions within context words
- ❑ Output layer: two classifiers predicting the start and end positions

BiDAF: Performance on SQuAD

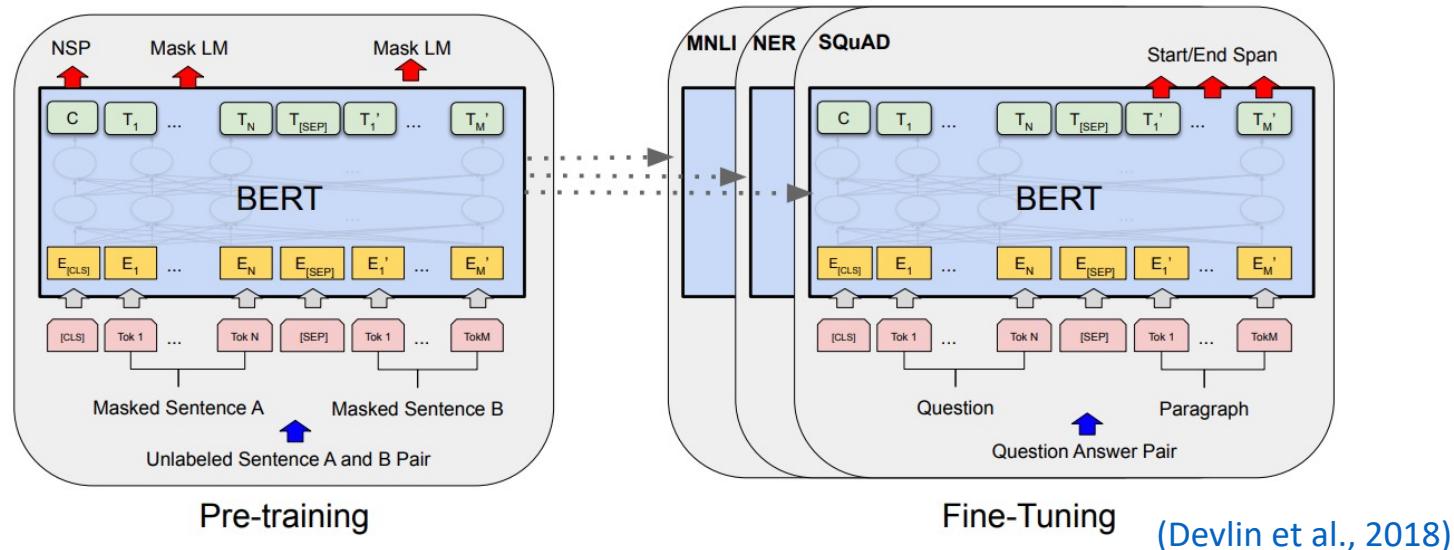
□ This model achieved 77.3 F1 on SQuAD v1.1.

- Without context-to-query attention \Rightarrow 67.7 F1
- Without query-to-context attention \Rightarrow 73.7 F1
- Without character embeddings \Rightarrow 75.4 F1

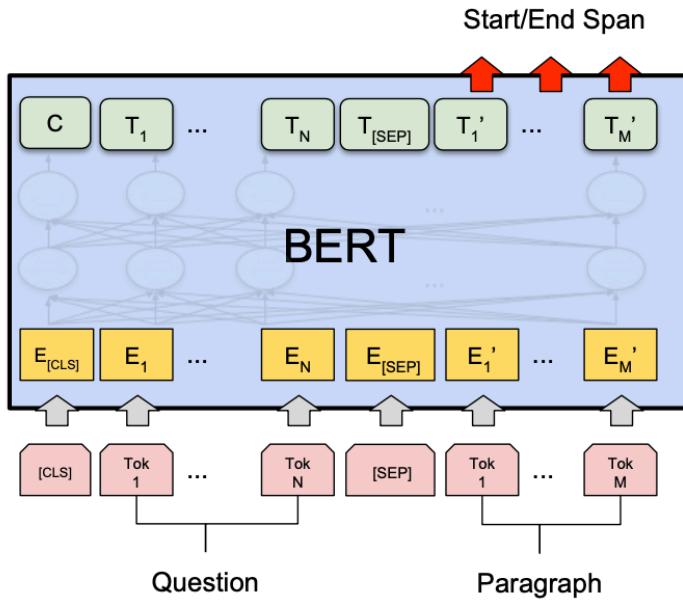
	F1
Logistic regression	51.0
Fine-Grained Gating (Carnegie Mellon U)	73.3
Match-LSTM (Singapore Management U)	73.7
DCN (Salesforce)	75.9
BiDAF (UW & Allen Institute)	77.3
Multi-Perspective Matching (IBM)	78.7
ReasoNet (MSR Redmond)	79.4
DrQA (Chen et al. 2017)	79.4
r-net (MSR Asia) [Wang et al., ACL 2017]	79.7
Human performance	91.2

BERT for reading comprehension

- ❑ BERT is a deep bidirectional Transformer encoder pre-trained on large amounts of text (Wikipedia + BooksCorpus)
- ❑ BERT is pre-trained on two training objectives:
 - Masked language model (MLM)
 - Next sentence prediction (NSP)
- ❑ BERTbase has 12 layers and 110M parameters, BERTlarge has 24 layers and 330M parameters



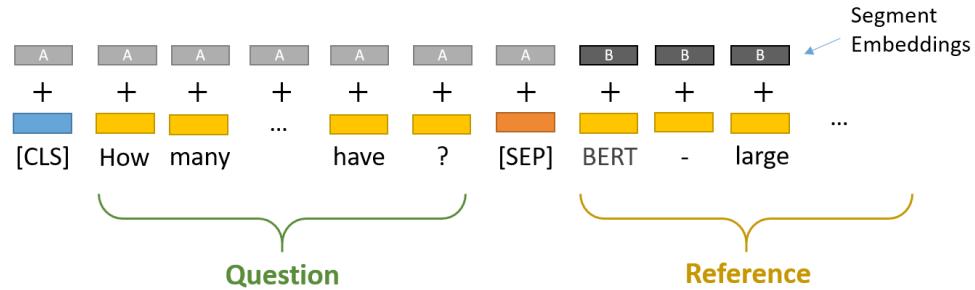
BERT for reading comprehension



Question = Segment A

Passage = Segment B

Answer = predicting two endpoints in segment B



Question: How many parameters does BERT-large have?

Reference Text: BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

Image credit: <https://mccormickml.com/>

$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$

$$p_{\text{start}}(i) = \text{softmax}_i(\mathbf{w}_{\text{start}}^\top \mathbf{h}_i)$$

$$p_{\text{end}}(i) = \text{softmax}_i(\mathbf{w}_{\text{end}}^\top \mathbf{h}_i)$$

where \mathbf{h}_i is the hidden vector of c_i , returned by BERT

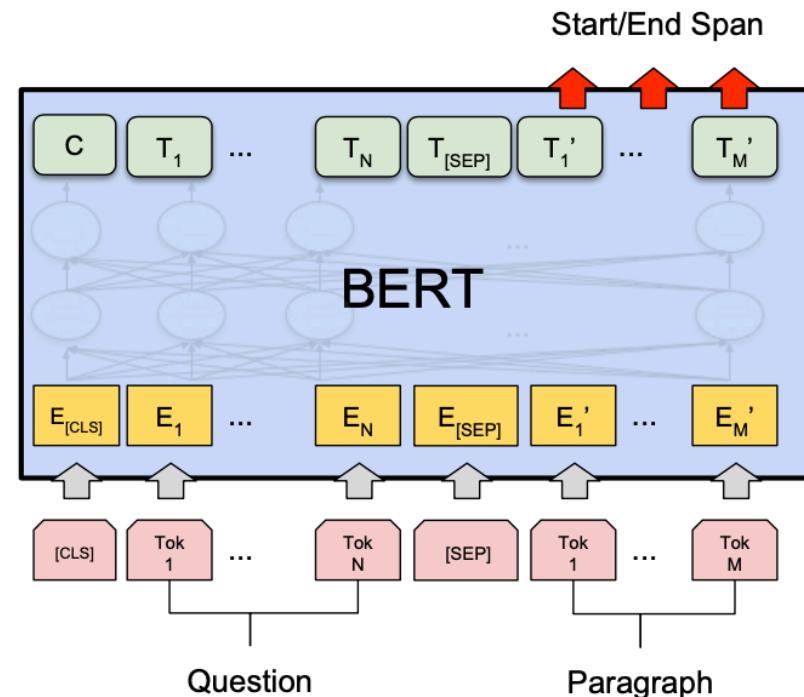
BERT for reading comprehension

$$\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$$

- ❑ All the BERT parameters (e.g., 110M) as well as the newly introduced parameters $h_{\text{start}}, h_{\text{end}}$ (e.g., $768 \times 2 = 1536$) are optimized together for L .
- ❑ It works amazingly well. Stronger pre-trained language models can lead to even better performance and SQuAD becomes a standard dataset for testing pre-trained models.

	F1	EM
Human performance	91.2*	82.3*
BiDAF	77.3	67.7
BERT-base	88.5	80.8
BERT-large	90.9	84.1
XLNet	94.5	89.0
RoBERTa	94.6	88.9
ALBERT	94.8	89.3

(dev set, except for human performance)



Comparisons between BiDAF and BERT models

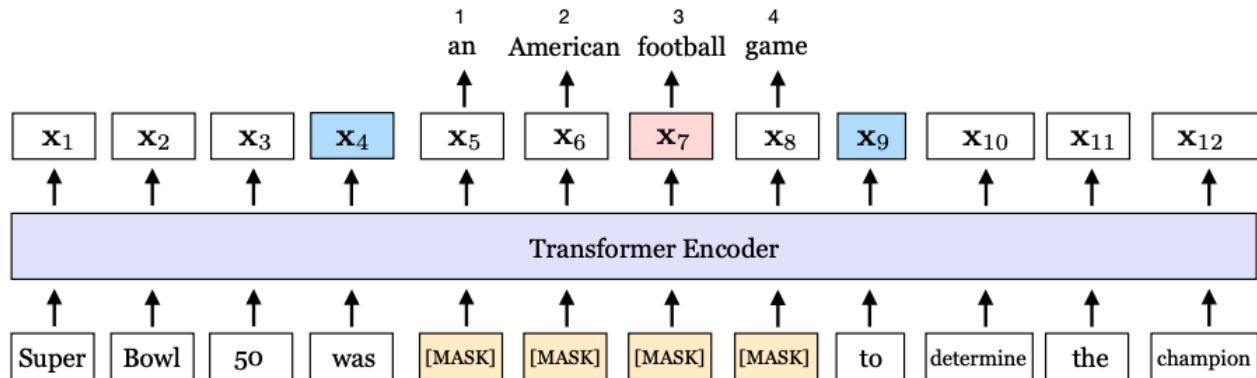
- ❑ BERT model has many many more parameters (110M or 330M).
BiDAF has ~2.5M parameters.
- ❑ BiDAF is built on top of several bidirectional LSTMs while BERT is built on top of Transformers (no recurrence architecture and easier to parallelize).
- ❑ BERT is **pre-trained** while BiDAF is only built on top of GloVe (and all the remaining parameters need to be learned from the supervision datasets).
- ❑ Pre-training is clearly a game changer but it is expensive..

Can we design better pre-training objectives?

The answer is yes!

$$\mathcal{L}(\text{football}) = \mathcal{L}_{\text{MLM}}(\text{football}) + \mathcal{L}_{\text{SBO}}(\text{football})$$

$$= -\log P(\text{football} \mid \mathbf{x}_7) - \log P(\text{football} \mid \mathbf{x}_4, \mathbf{x}_9, \mathbf{p}_3)$$

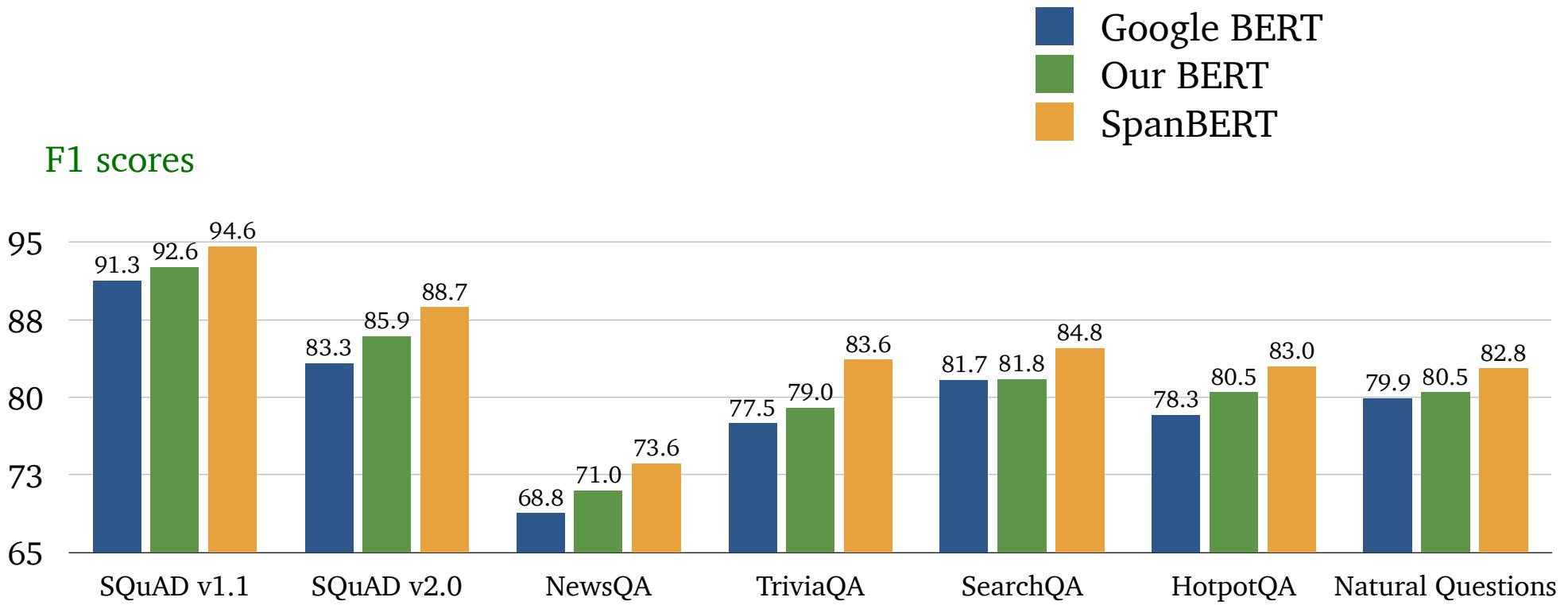


□ Two ideas:

- 1) masking contiguous spans of words instead of 15% random words
- 2) using the two end points of span to predict all the masked words in between = compressing the information of a span into its two endpoints

$$\mathbf{y}_i = f(\mathbf{x}_{s-1}, \mathbf{x}_{e+1}, \mathbf{p}_{i-s+1})$$

SpanBERT performance



Is reading comprehension solved?

- ❑ We have already surpassed human performance on SQuAD. Does it mean that reading comprehension is already solved? Of course not!
- ❑ The current systems still perform poorly on adversarial examples or examples from out-of-domain distributions

Article: Super Bowl 50

Paragraph: “*Peyton Manning became the first quarterback ever to lead two different teams to multiple Super Bowls. He is also the oldest quarterback ever to play in a Super Bowl at age 39. The past record was held by John Elway, who led the Broncos to victory in Super Bowl XXXIII at age 38 and is currently Denver’s Executive Vice President of Football Operations and General Manager. Quarterback Jeff Dean had jersey number 37 in Champ Bowl XXXIV.*”

Question: “What is the name of the quarterback who was 38 in Super Bowl XXXIII?”

Original Prediction: John Elway

Prediction under adversary: Jeff Dean

	Match Single	Match Ens.	BiDAF Single	BiDAF Ens.
Original	71.4	75.4	75.5	80.0
ADDSENT	27.3	29.4	34.3	34.2
ADDONESENT	39.0	41.8	45.7	46.9
ADDANY	7.6	11.7	4.8	2.7
ADDCOMMON	38.9	51.0	41.7	52.6

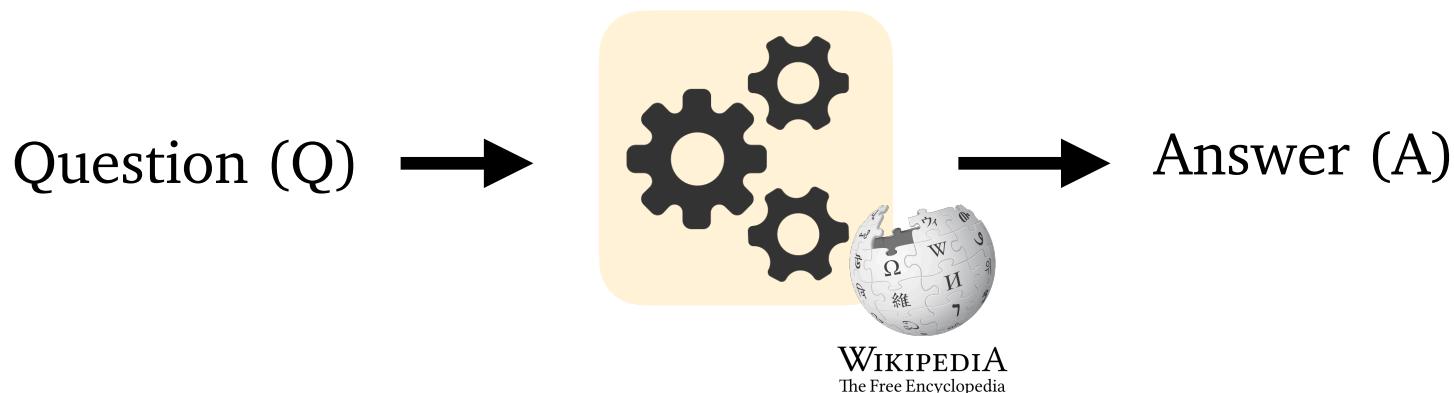
Is reading comprehension solved?

- ☐ Systems trained on one dataset can't generalize to other datasets:

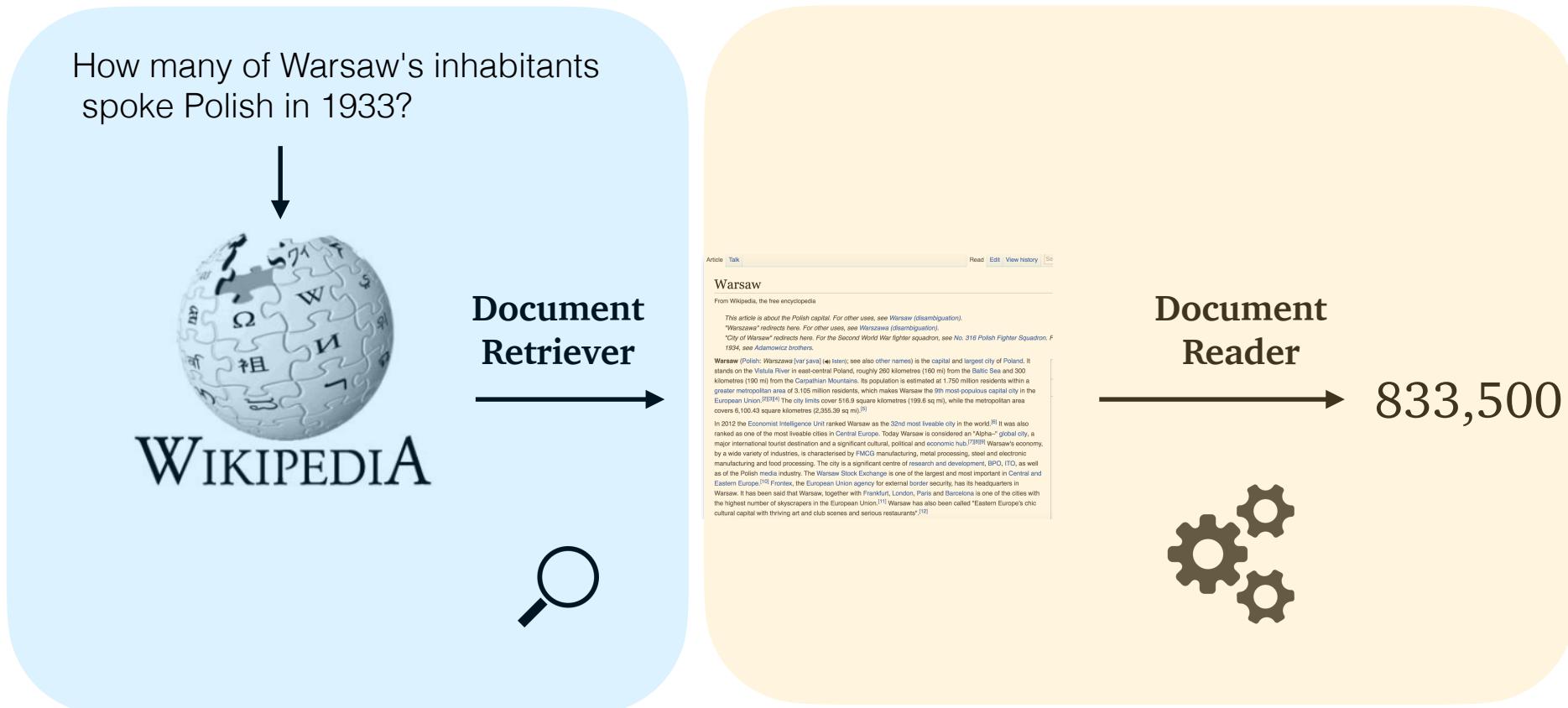
Fine-tuned on	Evaluated on				
	SQuAD	TriviaQA	NQ	QuAC	NewsQA
SQuAD	75.6	46.7	48.7	20.2	41.1
TriviaQA	49.8	58.7	42.1	20.4	10.5
NQ	53.5	46.3	73.5	21.6	24.7
QuAC	39.4	33.1	33.8	33.3	13.8
NewsQA	52.1	38.4	41.7	20.4	60.1

Open-domain question answering

- ❑ Different from reading comprehension, we don't assume a given passage.
- ❑ Instead, we only have access to a large collection of documents (e.g., Wikipedia). We don't know where the answer is located, and the goal is to return the answer for any open-domain questions.
 - In contrast to closed-domain systems that deal with questions under a specific domain (medicine, technical support).
- ❑ Much more challenging and a more practical problem!



Retriever-reader framework



<https://github.com/facebookresearch/DrQA>

Retriever-reader framework

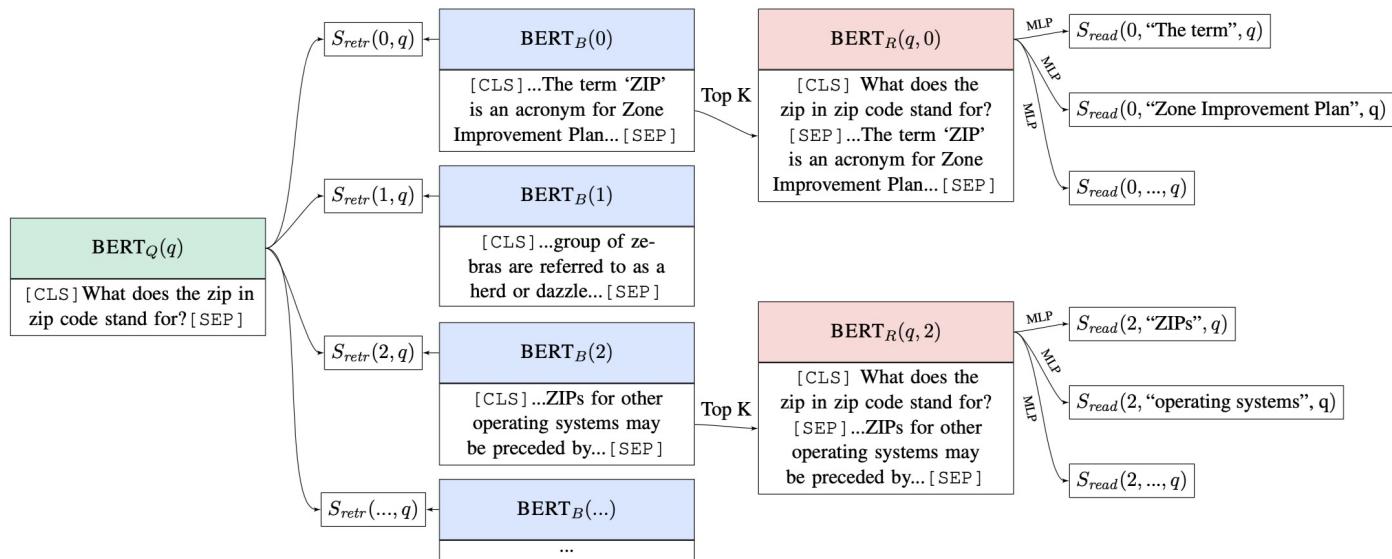
- ❑ Input: a large collection of documents $\mathcal{D} = D_1, D_2, \dots, D_N$ and Q
- ❑ Output: an answer string A

- ❑ Retriever: $f(\mathcal{D}, Q) \rightarrow P_1, \dots, P_K$ K is pre-defined (e.g., 100)
- ❑ Reader: $g(Q, \{P_1, \dots, P_K\}) \rightarrow A$ A reading comprehension problem!

- ❑ In DrQA,
 - Retriever = A standard TF-IDF information-retrieval sparse model (a fixed module)
 - Reader = a neural reading comprehension model that we just learned

We can train the retriever too

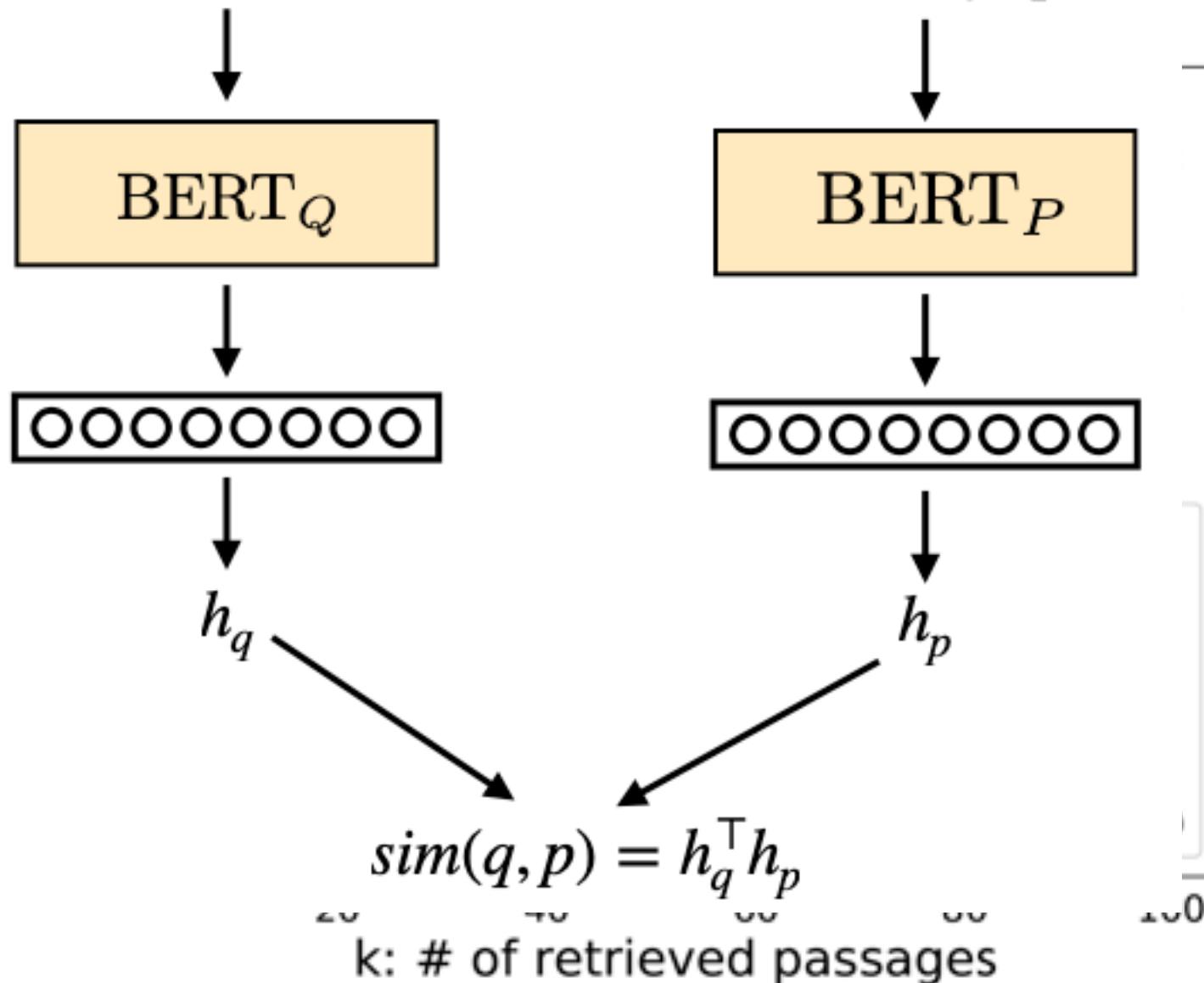
□ Joint training of retriever and reader



- Each text passage can be encoded as a vector using BERT and the retriever score can be measured as the dot product between the question representation and passage representation.
- However, it is not easy to model as there are a huge number of passages (e.g., 21M in English Wikipedia)

Question q

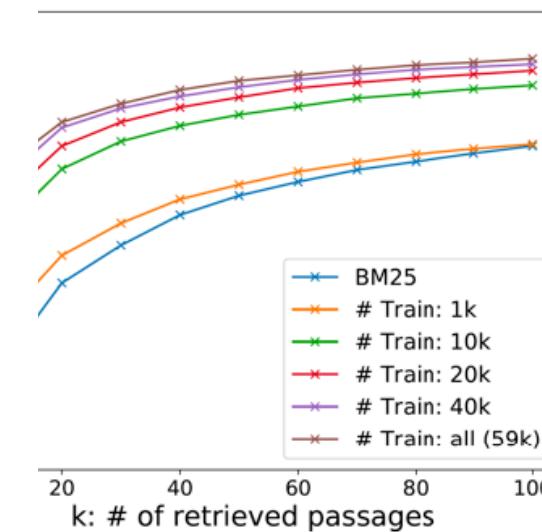
Passage p



ever too

ain the retriever using question-

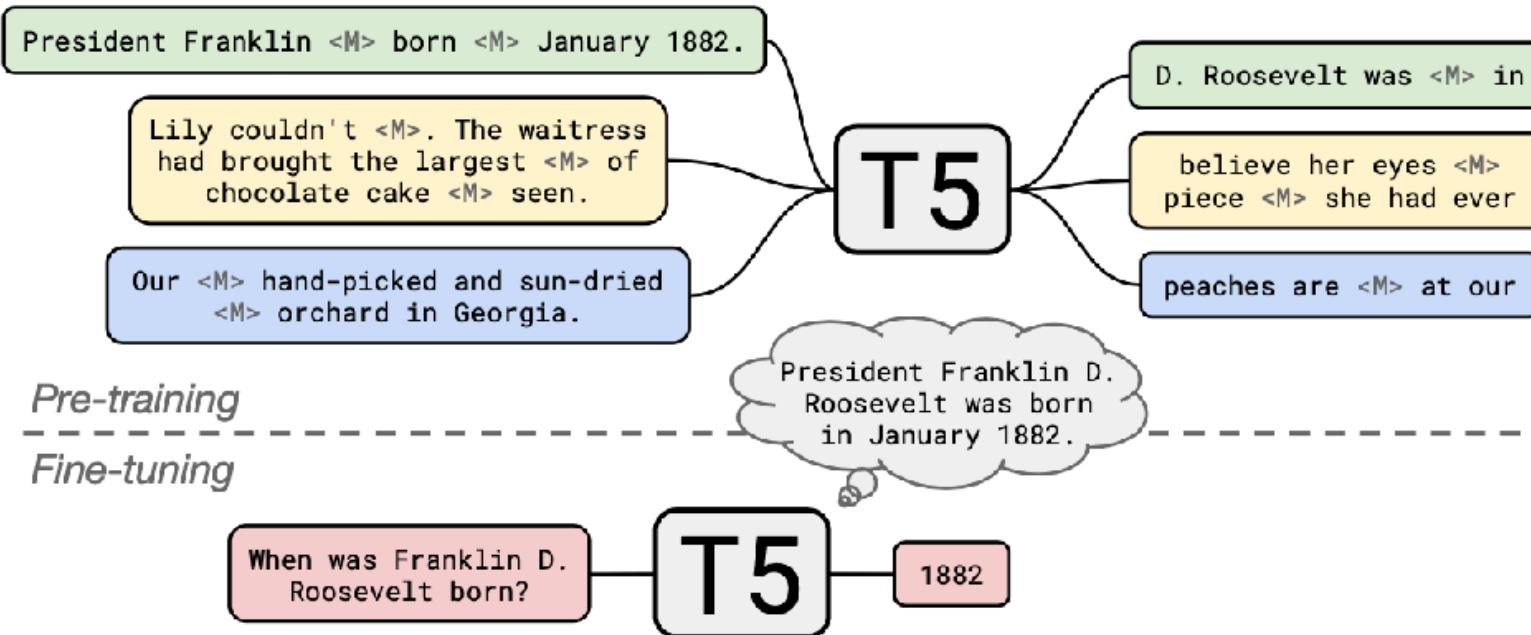
↳ Q/A pairs beat BM25!



is traditional IR retrieval models

Large language models can do open-domain QA well

- ... without an explicit retriever stage



Large language model-based QA (with web search!)

The screenshot shows a search interface with the following elements:

- Logo:** A stylized "YOU.com" logo.
- Search Bar:** A rounded rectangle containing the query "Where does Christopher D. Manning teach?". It includes a magnifying glass icon and a clear button (X).
- Filter Buttons:** A row of buttons for "All", "Chat" (which is highlighted in blue), "Images", "Videos", "News", "Maps", and "More".
- Result Summary:** A blue button labeled "Where does Christopher D. Manning teach?".
- Text Description:** A block of text stating that Christopher D. Manning is a professor of computer science and linguistics at Stanford University, linking to profiles [1][2] and the Director of the Stanford Artificial Intelligence Laboratory [1].
- Links:** Two blue links:
 - 1. Christopher Manning's Profile | Stanford Profiles**
<https://profiles.stanford.edu/chris-manning>
 - 2. Introduction to Information Retrieval: Manning, Christopher D ...**
<https://www.amazon.com/Introduction-Information-Retrieval-Christopher-Manning/dp/0521865719>
- Interaction Buttons:** At the bottom right are icons for upvote (up arrow), like (thumb up), and dislike (thumb down).

Problems with large language model-based QA

The screenshot shows a search interface with the following elements:

- Logo:** A stylized "YOU.com" logo.
- Search Bar:** The text "What is the most cited paper by Christopher D. Manning?" is entered into the search bar, which also includes a clear button ("X") and a search icon.
- Filter Buttons:** A horizontal row of buttons for "All", "Chat" (which is highlighted in blue), "Images", "Videos", "News", "Maps", and "More".
- Result Summary:** A blue-highlighted box contains the question "What is the most cited paper by Christopher D. Manning?".
- Result Content:** The response states: "The most cited paper by Christopher D. Manning is "Effective Approaches to Attention-Based Neural Machine Translation", which was co-authored by Minh-Thang Luong [1], Hieu Pham, and Christopher D. Manning. This paper has been cited over 18,400 times and is one of the most influential papers in the field of Natural Language Processing." Below this is a link: "1. Effective Approaches to Attention-based Neural Machine Translation" and the URL "https://arxiv.org/abs/1508.04025".
- Interaction Buttons:** Below the result content are upvote, downvote, and share icons.
- Input Field:** A "Ask me anything..." input field with a send icon.

Text overlay: "Seems totally reasonable!"

Text overlay: "But (1) it's not his most cited paper, and (2) it doesn't have that many citations. Yikes! Also the reference to a web page doesn't help."

Code Generation

(adapted from Gabriel Poesia's slides in Stanford CS224n)

Background: Program Synthesis

Program Synthesis

- ❑ Major long-standing challenge of AI: programs that write programs!
- ❑ Program synthesizer: program that takes a specification and outputs a program that satisfies it
- ❑ What specification?
 - A logical formula
 - Another equivalent program (e.g., a slower one)
 - Input/output examples
 - **Natural language description**

Program Synthesis from Logical Specifications

- ❑ When does synthesis make sense?
- ❑ When it's easier to describe *what* the program should do rather than *how* it should do it
- ❑ Unlike natural language generation, in program synthesis we can often test if a program satisfies the specification
- ❑ First attempt: logical formula describing what the program does

Program Synthesis Example: Sorting

❑ How would you logically specify a sorting algorithm?

❑ First attempt:

- Suppose the algorithm takes a list A and outputs a list B.
- Key property: B should be sorted
 - For all $i < \text{length}(B)$, $B[i] \leq B[i+1]$

Synthesizer

```
def sort(A):  
    return [1, 2]
```

Program Synthesis Example: Sorting

❑ How would you logically specify a sorting algorithm?

❑ Oops! Second attempt:

- Suppose the algorithm takes a list A and outputs a list B.
- Key property #1: B should be sorted
 - For all $i < \text{length}(B)$, $B[i] \leq B[i+1]$
- Key property #2: B should be a permutation of A
 - $\text{length}(B) = \text{length}(A)$
 - For all $B[i]$ there should exist some $A[j]$ such that $A[j] = B[i]$

$$\begin{aligned} & \forall k. (0 \leq k < n - 1) \implies (B[k] \leq B[k + 1]) & (1) \\ \wedge \quad & \forall k \exists j. (0 \leq k < n) \implies (0 \leq j < n \wedge B[j] = A[k]) & (2) \end{aligned}$$

(Gulwani, 2010)

Synthesizer

```
def sort(A):
    if len(A) <= 1: return A
    return (sort([x for x in A[1:] if x <= A[0]]) + [A[0]] +
            sort([x for x in A[1:] if x > A[0]]))
```

Synthesis from logical specifications

- ❑ Note that the problem is non-trivial: the specification says very little about how!
 - Not just a translation problem

- ❑ But logical specifications are hard to read, hard to write, hard to check

- ❑ Often easier to just write the program?

$$\begin{aligned} & \forall k. (0 \leq k < n - 1) \implies (B[k] \leq B[k + 1]) & (1) \\ & \wedge \quad \forall k \exists j. (0 \leq k < n) \implies (0 \leq j < n \wedge B[j] = A[k]) & (2) \end{aligned}$$

(Gulwani, 2010)

Synthesis from examples

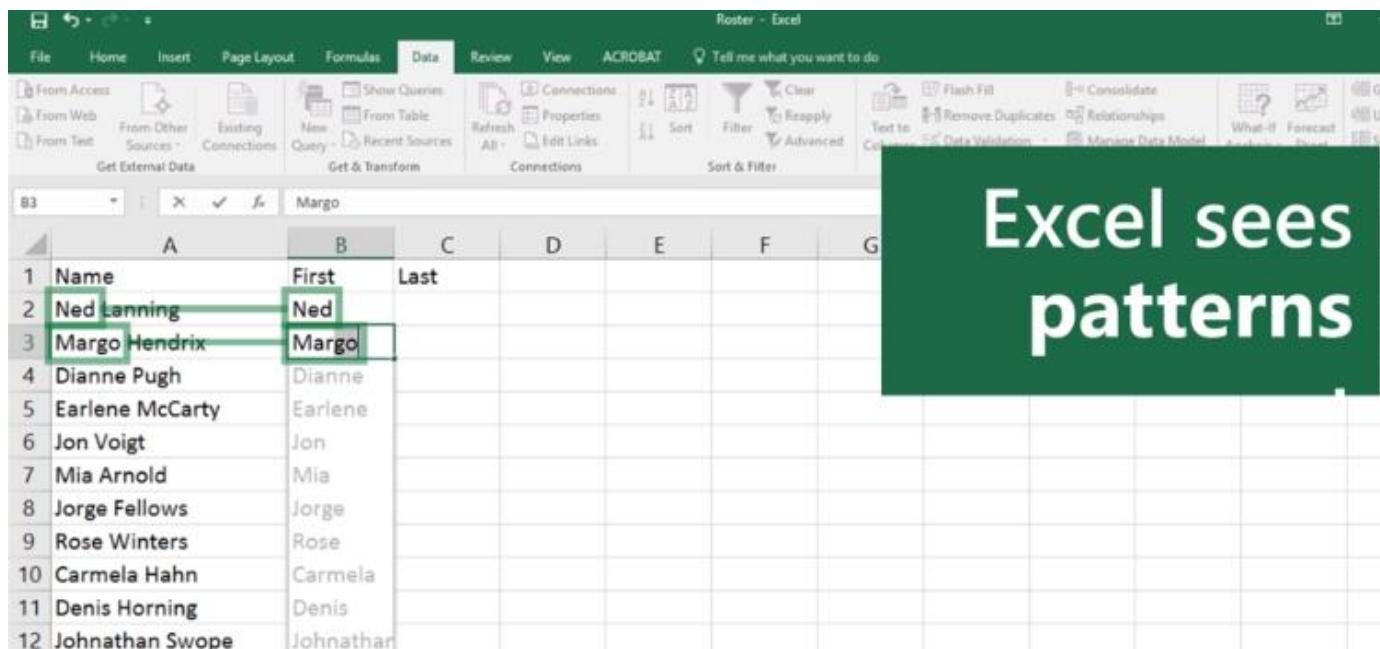
❑ Another kind of specification: input/output examples.

❑ How do you specify sorting a list?

- Given [3, 2, 1, 0], should return [0, 1, 2, 3]
- Given [1, 4, 2], should return [1, 2, 4]
- Given [9], should return [9]
- How about:
def sort(A):
 if len(A) == 4: **return** list(range(4))
 if len(A) == 3: **return** [1, 2, 4]
 return [9]

Synthesis from examples: FlashFill

- ❑ Perhaps the first massively deployed program synthesizer was FlashFill, in Microsoft Excel 2013
- ❑ Often worked with 1 or 2 examples!



The screenshot shows a Microsoft Excel spreadsheet titled "Roster - Excel". The data is organized into two columns: "Name" (A1:A12) and "First" (B1:B12). The first two rows have their first names extracted into column B. A green callout box with the text "Excel sees patterns" is overlaid on the screen, pointing towards the data. The "Data" tab is selected in the ribbon, and the "Flash Fill" button is visible in the ribbon bar.

	Name	First	Last
1	Ned Lanning	Ned	Lanning
2	Margo Hendrix	Margo	Hendrix
3	Dianne Pugh	Dianne	Pugh
4	Earlene McCarty	Earlene	McCarty
5	Jon Voigt	Jon	Voigt
6	Mia Arnold	Mia	Arnold
7	Jorge Fellows	Jorge	Fellows
8	Rose Winters	Rose	Winters
9	Carmela Hahn	Carmela	Hahn
10	Denis Horning	Denis	Horning
11	Johnathan Swope	Johnathan	Swope

Synthesis from examples: ambiguity

- ❑ Examples are always ambiguous: infinite number of satisfying programs
- ❑ There's an implicit human preference: some programs are obviously undesirable
 - To you, but not to a search algorithm
- ❑ What program is this?
 - "Jan" -> "January"
 - "Feb" -> "February"

Home > Microsoft Excel > Excel > Flash Fill - Wrong Pattern for Filling Month Names

Flash Fill - Wrong Pattern for Filling Month Names

Haytham Amairah TRUSTED CONTRIBUTOR
Feb 21 2019 11:59 PM

Flash Fill - Wrong Pattern for Filling Month Names

Hi all,

This is what the Flash Fill suggests to fill the full month names!

	A	B	C	D	E	F	G	H
1								
2								
3				Jan	January			
4				Feb	Februry			
5				Mar	Maruary			
6				Apr	Apruary			
7				May	Mayuary			
8				Jun	Junuary			
9				Jul	Juluary			
10				Aug	Auguary			
11				Sep	Sepuary			
12				Oct	Octuary			
13				Nov	Novuary			
14				Dec	Decuary			

Program Synthesis: Summary and Challenges

- ❑ A synthesizer should take a higher-level specification of a program and generate an implementation
- ❑ Many implications if we make this work: lower barrier to access programming, higher productivity, ...
- ❑ But many challenges:
 - Infinite space of programs
 - Enumerative search is impractical in real-world languages (e.g., Python)
 - Simple specifications are ambiguous: how to capture human preferences?

Program Synthesis with Language Models

Large language models can generate code

- ❑ Language models give us $P(\text{continuation} \mid \text{prefix})$ on Internet text data
 - "UNIST is located in " -> "**Ulsan**" "Seoul"
- ❑ "The following is a Python function that, when given the list [1, 3, 2], returns [1, 2, 3]:"
 - GPT-3: "def sort_list(lst):
 lst.sort()
 return lst"

Large language models can generate code

- ❑ GPT-3 was able to implement simple Python functions from docstrings without having been explicitly trained for that
- ❑ Code is massively available as training data from open source projects (e.g., over 120M public repositories on Github)
- ❑ Idea in OpenAI Codex: train large language model on majority code data
- ❑ Codex (v1): Same architecture as GPT-3, but with 12B parameters (vs 175B)

Evaluating language models for code generation

- ❑ Synthesis challenge: given a Python docstring, generate the function implementation
- ❑ How to ensure problems were not seen during training?
- ❑ Authors introduced HumanEval, a manually created dataset of 164 problems
- ❑ Each problem has a set of hidden tests; a program is correct if it passes all hidden tests
- ❑ pass@k: probability that, out of k samples, at least one is correct

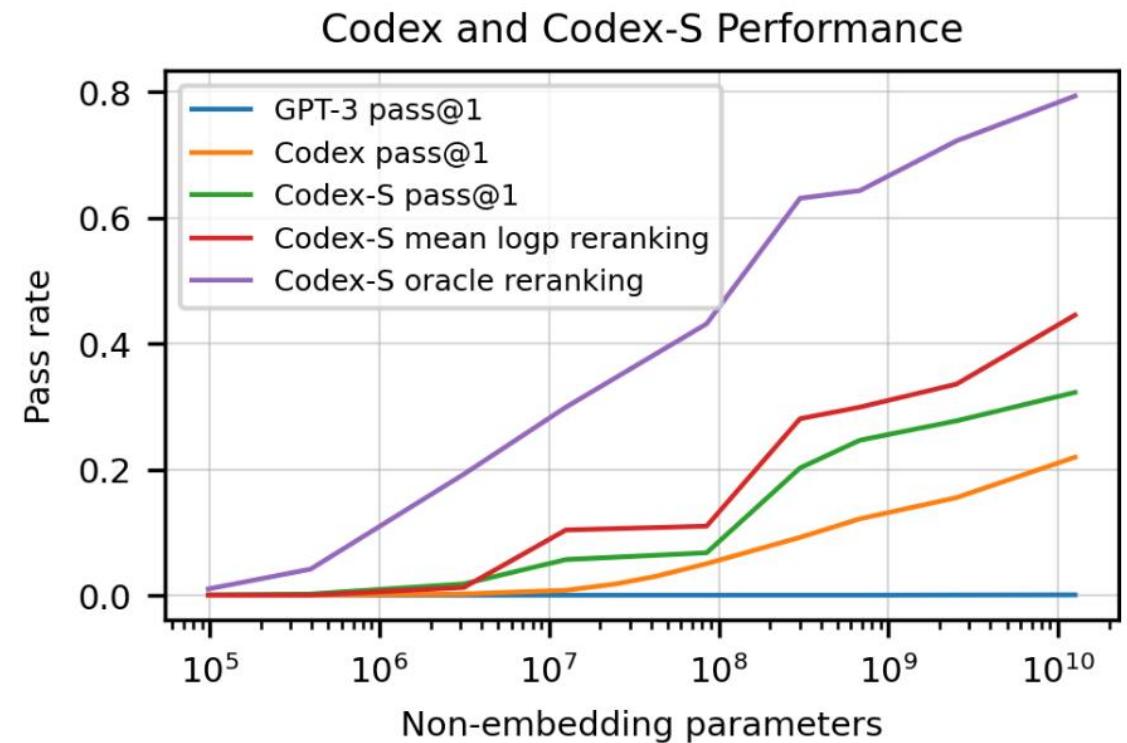
```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]

def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =>12
    solution([3, 3, 3, 3, 3]) =>9
    solution([30, 13, 24, 321]) =>0
    """
    return sum(lst[i] for i in range(0, len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

Evaluating language models for code generation

- ❑ GPT-3 fails on all problems
- ❑ Codex alone has non-trivial performance
- ❑ Fine-tuning on problems with this format of function synthesis (Codex-S) improves performance
- ❑ Sampling 100 programs, reranking and choosing best improves further



Evaluating language models for code generation

```
def is_prime(n):
    """Return true if a given number is prime, and
       false otherwise.
    >>> is_prime(6)
    False
    >>> is_prime(101)
    True
    >>> is_prime(11)
    True
    >>> is_prime(13441)
    True
    >>> is_prime(61)
    True
    >>> is_prime(4)
    False
    >>> is_prime(1)
    False
    """
        ### COMPLETION 1 (WRONG): ####
        for i in range(2, n):
            if n % i == 0:
                return False
        return True
        ### COMPLETION 2 (WRONG): ####
        for i in range(2, n):
            if n % i == 0:
                return False
        return True
        ### COMPLETION 3 (WRONG): ####
        # TODO: implement this function
        pass
        ### COMPLETION 4 (CORRECT): ####
        if n < 2: return False
        if n == 2: return True
        if n%2 == 0: return False
        return not any(n%k == 0 for k in range(3,int(n
                           **0.5)+1,2))
```

Evaluating language models for code generation

```
def vowels_count(s):
    """Write a function vowels_count which takes a
       string representing
       a word as input and returns the number of vowels in
       the string.
    Vowels in this case are 'a', 'e', 'i', 'o', 'u'.
    Here, 'y' is also a
    vowel, but only when it is at the end of the given
    word.

Example:
>>> vowels_count("abcde")
2
>>> vowels_count("ACEDY")
3
"""

```

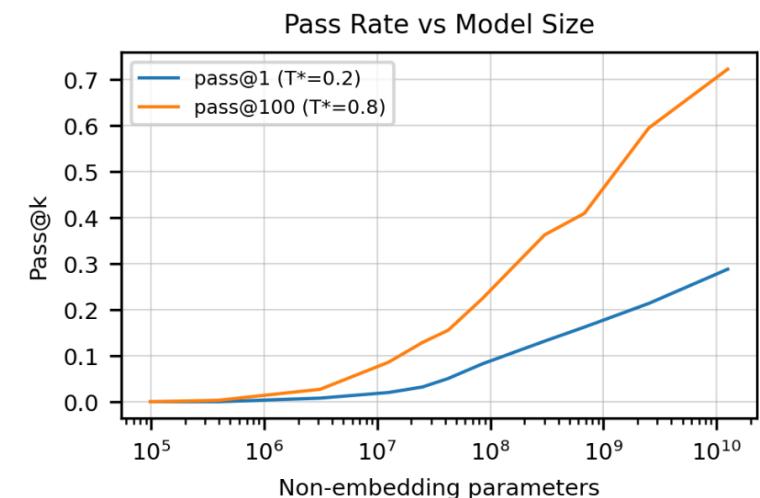
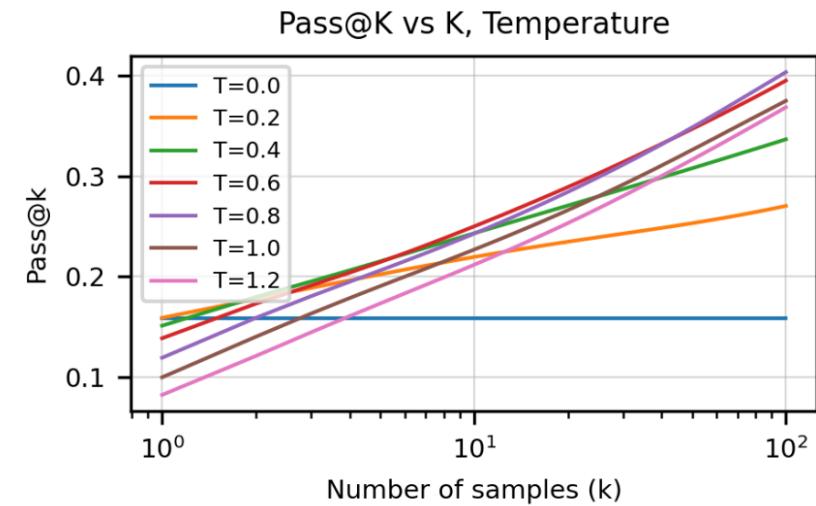
```
### COMPLETION 1 (WRONG): ###
vowels = "aeiou"
v = 0
for i in s:
    if i in vowels:
        v += 1
return v

### COMPLETION 2 (WRONG): ###
vowels_set = {'a', 'e', 'i', 'o', 'u', 'y'}
count = 0
for c in s:
    if c in vowels_set:
        count += 1
return count

### COMPLETION 4 (WRONG): ###
return 0
```

Sampling vs Temperature

- ❑ Sampling more programs increases the chance of getting one right
- ❑ Trade-off between temperature and $P(\text{correct})$:
 - Low temperature:
high likelihood (higher $P(\text{correct})$)
less diversity
 - Higher temperature:
lower likelihoods
more diversity



Scaling randomness: Softmax temperature (recap)

- ❑ Recall: On timestep t , the model computes a prob distribution P_t by applying the softmax function to a vector of scores $S \in \mathbb{R}^{|V|}$

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- ❑ You can apply a temperature hyperparameter τ to the softmax to rebalance P_t :

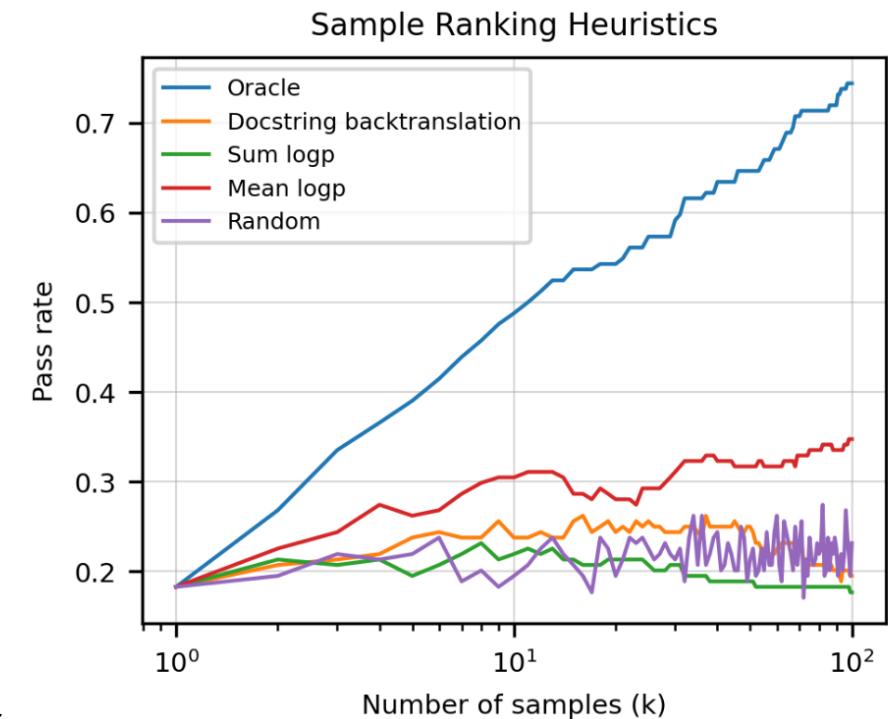
$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- ❑ Raise the temperature $\tau > 1$: P_t becomes more uniform
 - More diverse output (probability is spread around vocab)
- ❑ Lower the temperature $\tau < 1$: P_t becomes more spiky
 - Less diverse output (probability is concentrated on top words)

- ❑ Note: softmax temperature is not a decoding algorithm!
 - Temperature is a hyperparameter for decoding:
It can be tuned for both beam search and sampling.

Ranking

- ❑ For end-users, you don't want to present 100 choices
- ❑ Alternative #1: only sample small number of programs
- ❑ Alternative #2: sample large number of programs, but re-rank and only show top k for small k
- ❑ Oracle: run on all hidden tests and return the program that passes all, if any
- ❑ Alternatives: use model's log-probabilities to re-rank



From code to natural language

- ❑ The last tests sampled from $P(\text{code} \mid \text{docstring})$. What if we sample $P(\text{docstring} \mid \text{code})$?
- ❑ Less frequent in training data since docstring comes before
- ❑ Can synthetically create a dataset by inverting this order and fine-tune Codex to obtain Codex-D
- ❑ Pass@K estimated by human evaluation
- ❑ Lower performance than in the other direction!
Fine-tuning with teacher forcing did not help

MODEL	PASS @ 1	PASS @ 10
CODEX-S-12B	32.2%	59.5%
CODEX-D-12B	20.3%	46.5%

AlphaCode

- ❑ In 2022, DeepMind published AlphaCode, a system combining & expanding these ideas to solve competitive programming problems
- ❑ Most technical design choices in AlphaCode targeted faster sampling:
 - Unlike Codex, AlphaCode used an encoder-decoder Transformer (faster to encode the problem)
 - Unlike a regular Transformer, they used multi-query attention instead of full multi-head attention blocks (several query heads but single key/value)

(A) Problem (input)

You are given two strings s and t , both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.

When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed among those that aren't deleted yet (or does nothing if there are no characters in the current string). For example, if s is "abcb" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcba" and you press Backspace instead of the last two letters, then the resulting text is "a".

Your task is to determine whether you can obtain the string t , if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of t .

Input

The first line contains a single integer q ($1 \leq q \leq 10^5$) — the number of test cases.

The first line of each test case contains the string s ($1 \leq |s| \leq 10^5$). Each character of s is a lowercase English letter.

The second line of each test case contains the string t ($1 \leq |t| \leq 10^5$). Each character of t is a lowercase English letter.

It is guaranteed that the total number of characters in the strings over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print "YES" if you can obtain the string t by typing the string s and replacing some characters with presses of "Backspace" button, or "NO" if you cannot.

You may print each letter in any case (YES, yes, Yes will all be recognized as positive answer, NO, no and nO will all be recognized as negative answer).

Example

Input	Output
4	YES
ababa	NO
ba	NO
ababa	YES

Consider the example test from the statement.

In order to obtain "ba" from "ababa", you may press Backspace instead of typing the first and the fourth characters.

There's no way to obtain "bb" while typing "ababa".

There's no way to obtain "aaaa" while typing "aaa".

In order to obtain "ababa" while typing "aaaaaa", you have to press Backspace instead of typing the first character, then type all the remaining characters.

Note

If the letters at the end of both phrases don't match, the last letter must be deleted. If they do match we can move onto the second last letter and repeat.

Solution (output)

```
t=int(input())
for i in range(t):
    s=input()
    t=input()
    a=[]
    b=[]
    for j in s:
        a.append(j)
    for j in t:
        b.append(j)
    a.reverse()
    b.reverse()
    c=[]
    while len(b)!=0 and len(a)!=0:
        if a[0]==b[0]:
            c.append(b.pop(0))
            a.pop(0)
        elif a[0]!=b[0] and len(a)!=1:
            a.pop(0)
        elif a[0]!=b[0]:
            a.pop(0)
    if len(b)==0:
        print("YES")
    else:
        print("NO")
```

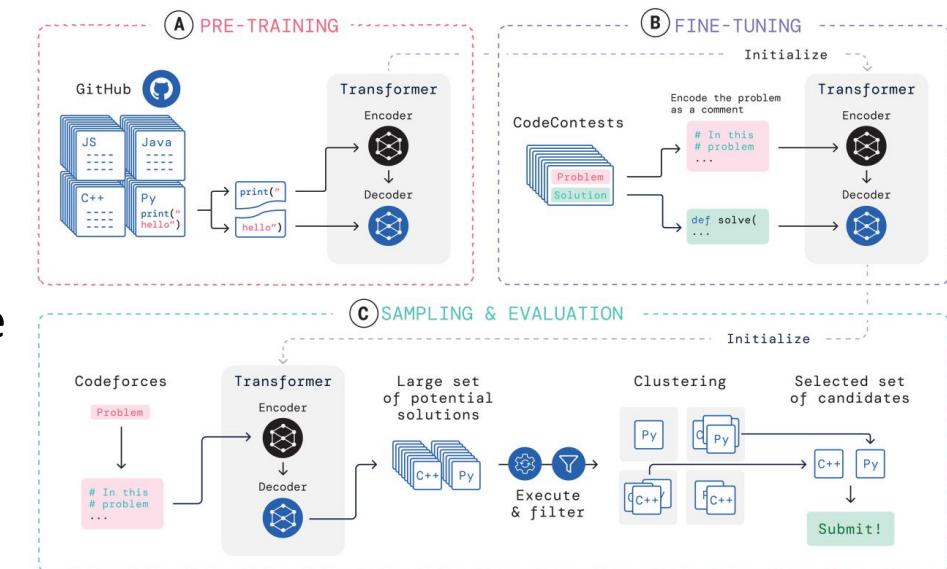
First the solution reads the two phrases.

If we've matched every letter, it's possible and we output that.

Backspace deletes two letters. The letter you press backspace instead of, and the letter before it.

AlphaCode: Pipeline

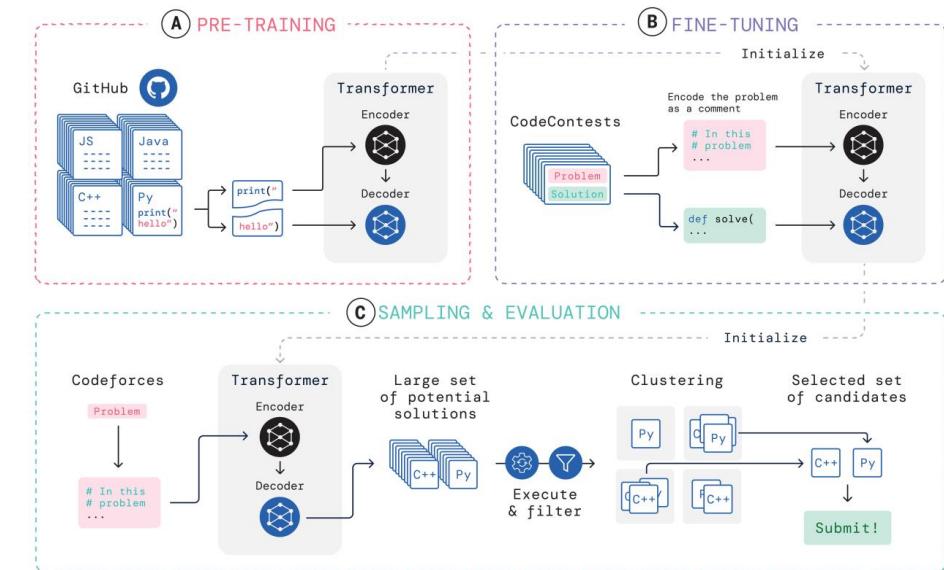
- ❑ Pre-training: standard cross-entropy loss training on 715GB of Github code; encoder had additional MLM loss
- ❑ Fine-tuning: human solutions to 13k competitive programming problems:
 - RL fine-tuning (GOLD); only need to learn how to produce one correct solution, instead of necessarily making *all* training solutions likely
 - Value-conditioning: use incorrect submissions to augment training, but prepend a comment saying whether solution was accepted or not



AlphaCode: Pipeline

□ Sampling:

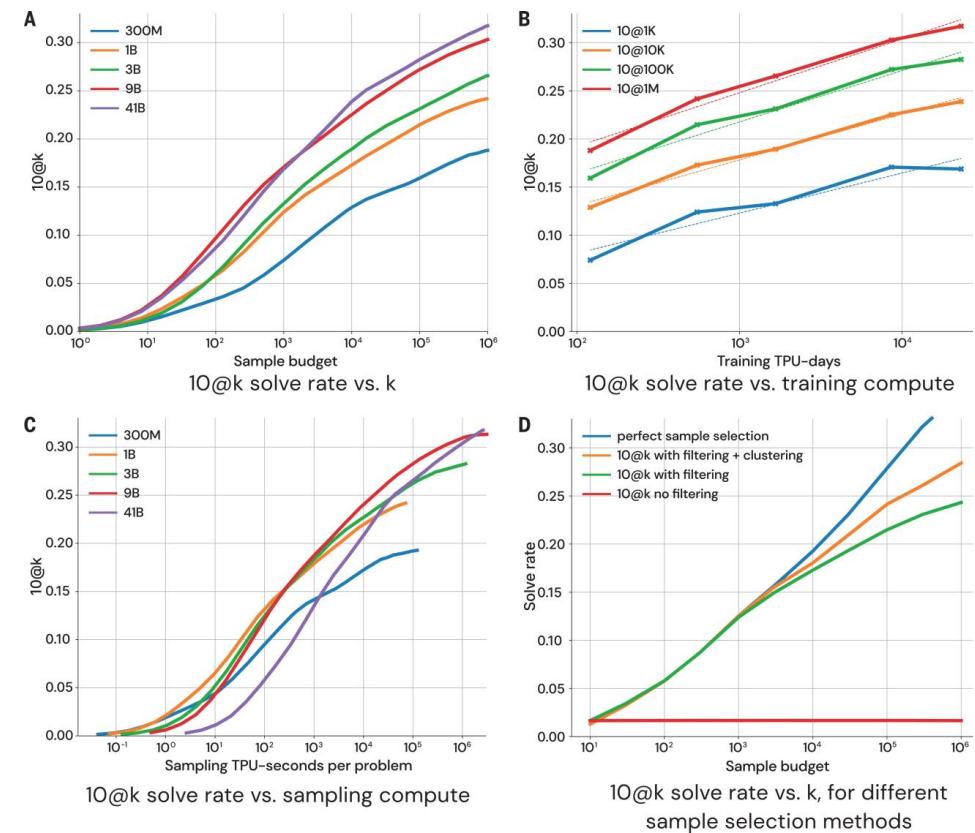
- Up to 100k samples per problem (!)
- Want to do up to 10 submissions per problem; only assume access to a small number of public tests
- Filtering: discard samples that fail public tests
- Clustering: trained a separate model to generate inputs; clustered generations by behavior on those inputs; submitted one exemplar from 10 largest clusters



AlphaCode: Results

- ❑ Log-linear scaling with sample budget
(10x samples approx. +6% solve rate)
- ❑ Log-linear scaling with compute
- ❑ Non-trivial performance on several
- ❑ Division 2 Codeforces contests (below)
 - "approximately corresponds to a novice programmer with a few months to a year of training"

Contest ID	1591	1608	1613	1615	1617	1618	1619	1620	1622	1623	Average
Maximum	43.5%	43.6%	59.8%	60.5%	65.1%	32.2%	47.1%	54.0%	57.5%	20.6%	48.4%
Estimated	44.3%	46.3%	66.1%	62.4%	73.9%	52.2%	47.3%	63.3%	66.2%	20.9%	54.3%
Minimum	74.5%	95.7%	75.0%	90.4%	82.3%	53.5%	88.1%	75.1%	81.6%	55.3%	77.2%



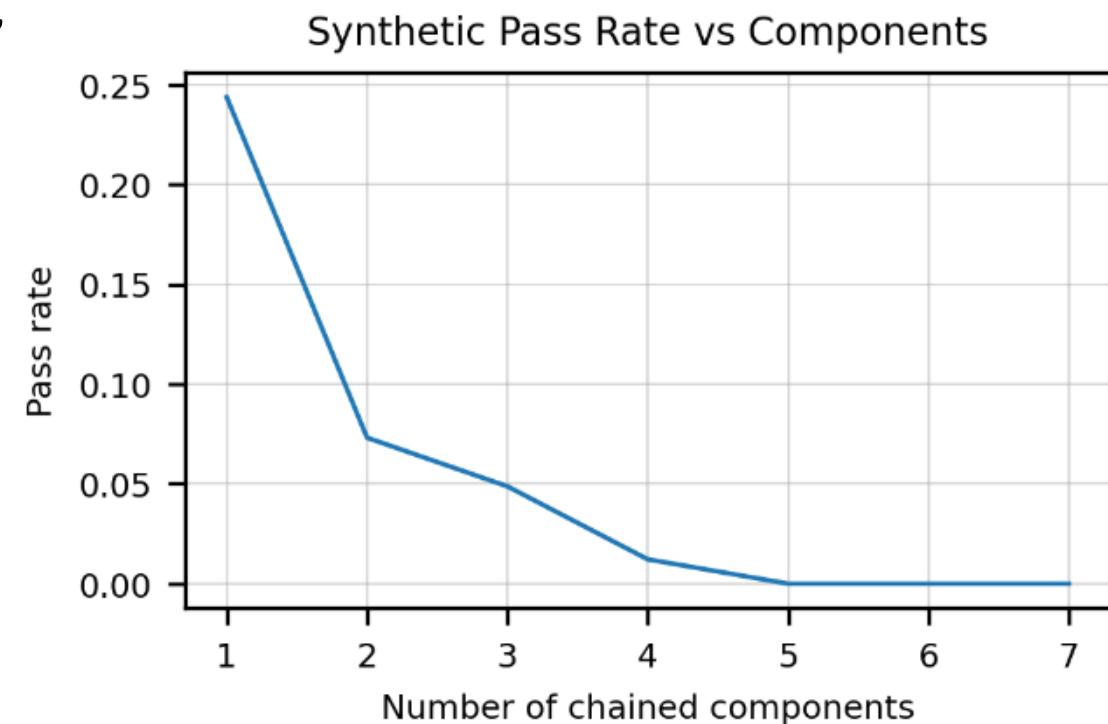
AlphaCode: Results & Takeaways

- ❑ Sampling more is the largest contributor!
Most of their methods don't improve things with at 1k samples
- ❑ Shows potential of simple Transformer-based synthesizers taken to scale
- ❑ But scale alone won't get this to competing in division 1 at current log-linear rates...

Fine-tuning setting	Solve rate	
	10@1K	10@1M
No enhancements	6.7% (6.5–6.8)	19.6% (18.2–20.4)
+ MLM	6.6% (6.2–7.0)	20.7% (19.1–21.3)
+ Tempering	7.7% (7.2–8.5)	21.9% (20.7–22.6)
+ Random tags and ratings	6.8% (6.4–7.0)	22.4% (21.3–23.0)
+ Value	10.6% (9.8–11.1)	23.2% (21.7–23.9)
+ GOLD	12.4% (12.0–13.0)	24.2% (23.1–24.4)
+ Clustering	12.2% (10.8–13.4)	28.4% (27.5–29.3)

Back to Codex: A note on compositionality

- ❑ If a human can trivially solve problem X (e.g., reverse a string), and also problem Y (e.g., compute string length), the problem "do X then Y" is still trivial
- ❑ This is not necessarily the case for LMs
- ❑ Codex paper reported an experiment with synthetic tasks made by chaining simple components like example above
- ❑ Result: performance decays exponentially as components increase, even if individually they're still trivial



Code language models: takeaways

- ❑ Transformer models trained on large amounts of code have non-trivial performance in generating programs in real-world programming languages
 - These results were unimaginable just a few years ago
- ❑ Sampling, testing and filtering can get quite far
 - Although it gets expensive fast:
"Training and evaluating our largest 41B model on Codeforces required a total of 2149 petaflop/s-days and 175 megawatt-hours [~16 times the average American household's yearly energy consumption (29)]."
- ❑ Still, many of these experiments assume a setting that fundamentally differs from real- world programming
 - Well-defined, self-contained, short problems; extensive existing correctness & performance tests; only need standard libraries; ...
- ❑ But code LMs can already be quite helpful tools if you can guide them!

Bonus: editing code in dialogue-style

I need to write a Python function called `count_common`. Here's a description: 'Write a function to count the most common words in a dictionary.'

Can you give me an example?

```
assert count_common(['one', 'two', 'three', 'four', 'five', 'one', 'two',  
'one', 'three', 'one']) == [('one', 4), ('two', 2), ('three', 2), ('four', 1)]
```

```
def count_common(list_of_words):  
    dict = {}  
    for word in list_of_words:  
        if word in dict:  
            dict[word] += 1  
        else:  
            dict[word] = 1  
    return sorted(dict.items(), key=lambda x: x[1])
```



tests fail

Close! But can you sort the output in reverse (descending) order?

```
def count_common(list_of_words):  
    dict = {}  
    for word in list_of_words:  
        if word in dict:  
            dict[word] += 1  
        else:  
            dict[word] = 1  
    return sorted(dict.items(), key=lambda x: x[1], reverse=True)
```



tests fail

Great. Now notice that the example only returns the top 4 results.

```
def count_common(list_of_words):  
    dict = {}  
    for word in list_of_words:  
        if word in dict:  
            dict[word] += 1  
        else:  
            dict[word] = 1  
    return sorted(dict.items(), key=lambda x: x[1], reverse=True)[:4]
```



tests pass

Hooray! 🎉🎉🎉

Programs as Tools For Language Models

Programs as tools

- ❑ Humans are effective in large part for our ability to create and use complex tools
 - What is $123 * 456$? You'll use a calculator
 - What time is it? You'll use a clock
 - What are the 5 largest airports in the world? You'll do a Web search
 - How many shoe boxes would fit in an average car trunk? You'll possibly do several searches to get data and use a calculator, and perhaps a volume conversion table
- ❑ A language model might have the strategy of how to solve these problems, but with standard decoding it can use no external tools
- ❑ This is quite limiting! E.g. Minerva, an LLM trained on mostly math, still performs frequent calculation errors when solving math problems.

Type of mistakes	Occurrences
Incorrect reasoning	82
Incorrect calculation	70
Misunderstands question	22
Uses incorrect fact	16
Solution too short	4
Hallucinated math objects	4
Other mistakes	3

Example: Calculator

Problem: Beth bakes 4, 2 dozen batches of cookies in a week. If these cookies are shared amongst 16 people equally, how many cookies does each person consume?

Solution: Beth bakes 4 2 dozen batches of cookies for a total of $4 \times 2 = <<4*2=8>>$ 8 dozen cookies

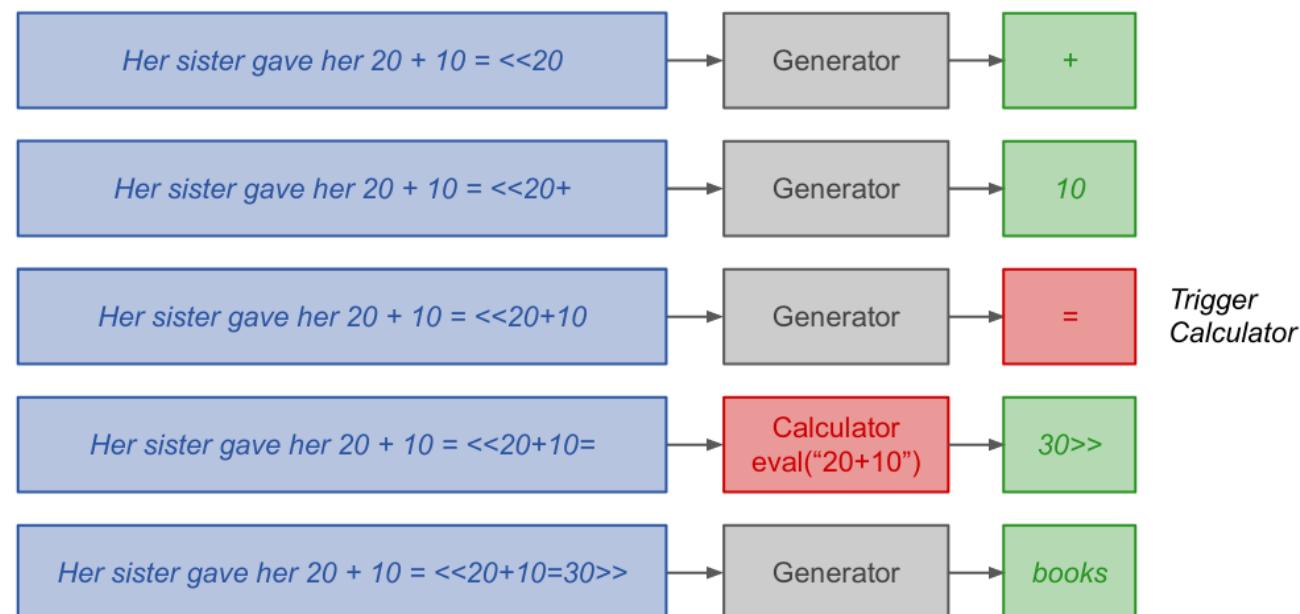
There are 12 cookies in a dozen and she makes 8 dozen cookies for a total of $12 \times 8 = <<12*8=96>>$ 96 cookies

She splits the 96 cookies equally amongst 16 people so they each eat $96 / 16 = <<96/16=6>>$ 6 cookies

Final Answer: 6

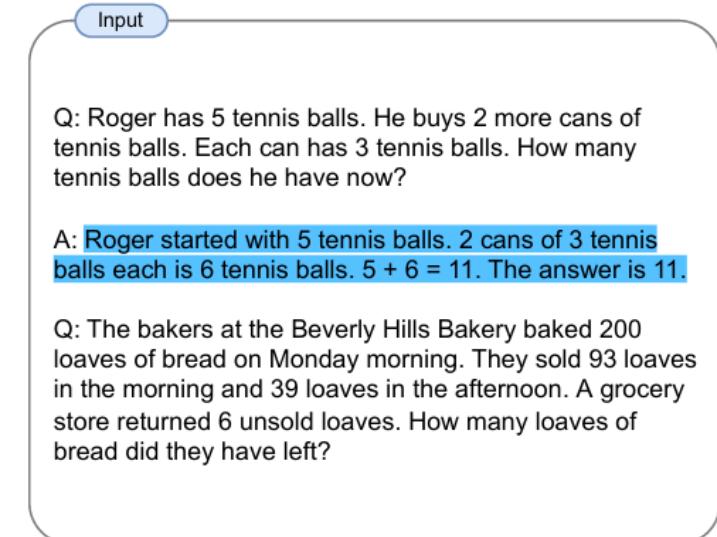
❑ Key idea:

- Watch for special input token during decoding
- When model generates that token, call external tool (e.g., calculator)
- Paste the result in decoding sequence and keep decoding
- Model will now generate conditioned on the tool's result!



Example: Python programs to solve reasoning problems

Chain-of-Thought (Wei et al., 2022)



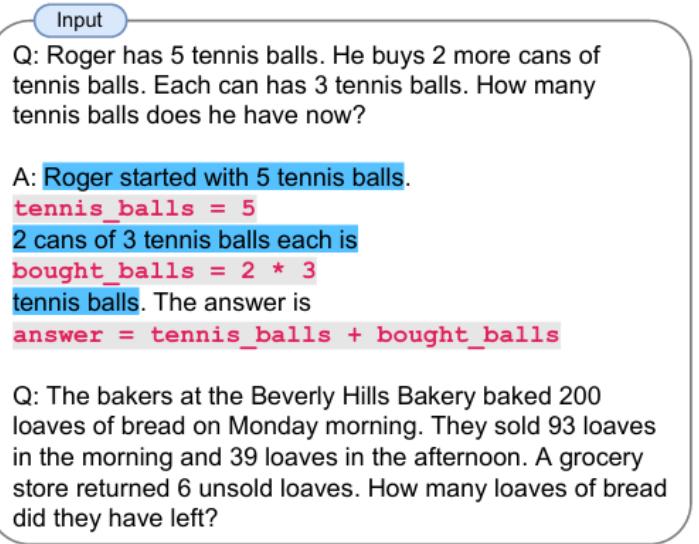
Model Output

A: The bakers started with 200 loaves. They sold 93 in the morning and 39 in the afternoon. So they sold $93 + 39 = 132$ loaves. The grocery store returned 6 loaves. So they had $200 - 132 - 6 = 62$ loaves left.

The answer is 62.



Program-aided Language models (this work)



Model Output

A: The bakers started with 200 loaves
`loaves_baked = 200`
They sold 93 in the morning and 39 in the afternoon
`loaves_sold_morning = 93`
`loaves_sold_afternoon = 39`
The grocery store returned 6 loaves.
`loaves_returned = 6`
The answer is
`answer = loaves_baked - loaves_sold_morning`
`- loaves_sold_afternoon + loaves_returned`

`>>> print(answer)`

74



(Gao et al, 2022)

Example: Toolformer

- ❑ First example (calculator) trained the model to use the calculator by having annotations in the training dataset
- ❑ Second (PAL) used few-shot prompting
- ❑ Toolformer introduced a self-supervised approach to teach models to use new tools:
 - Start with a few examples of each tool and larger dataset for the task without tool use
 - Use in-context learning to insert candidate API calls in training examples
 - Call APIs, evaluate whether the result decreases perplexity of the rest of the solution
 - Fine-tune model on cases where it does
 - Result: model can now often use APIs

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

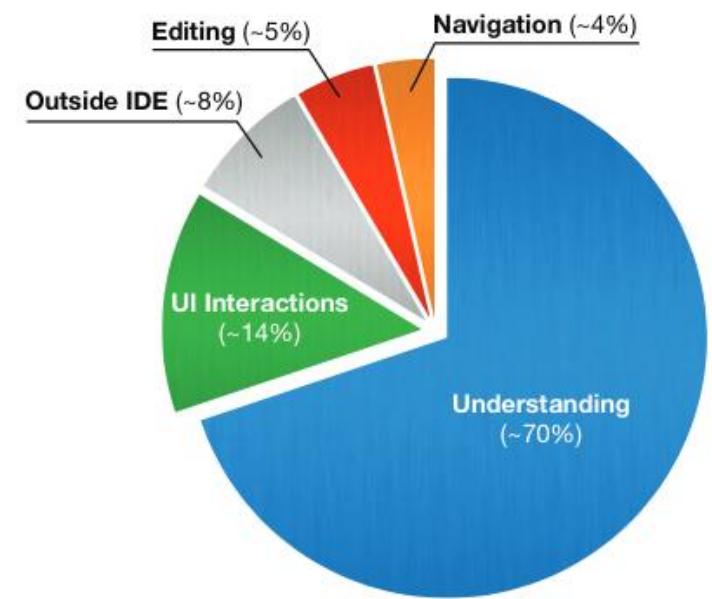
The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

Limitations and Discussion

Limitations and Discussion

- ❑ How much of a programmer's job can current language models automate?
- ❑ Relatively very little! Most time in actual software engineering is **not** spent writing code
- ❑ Chart shows just time spent with IDE open. A lot more happens outside that time (talking, prioritizing, meetings)
- ❑ A ton of time deciding & discussing *what* to build rather than actually building it
- ❑ Even when coding, much of it is editing rather than writing new code



(Minelli et al, 2015)

Limitations and Discussion

❑ Debugging is very incremental and interactive:

- Write & run your code
- See outputs / compiler errors
- Maybe add a breakpoint / print statement
- Edit, repeat

❑ Re-sampling in Codex / AlphaCode ignores outputs/errors

- As programs grows, probability that a full sample will solve it completely decays exponentially

❑ Active research in learning to find and fix errors with LMs! (e.g. (Yasunaga & Liang, 2021))

- Still different setting from open-ended debugging that humans do

Limitations and Discussion

- ❑ Lots of public code to pre-train, but does not cover everything:
 - New or internal libraries
 - New programming languages
 - New language features
- ❑ Language models fail many tests of code understanding
 - Example: code execution. Given code and inputs, what does it output?
 - (Austin et al, 2021) found that even fine-tuned models struggle

Limitations and Discussion

- ❑ Public code repositories have lots of code with bugs
- ❑ Generated code often has functional or security bugs. Still need to understand it!
- ❑ (Perry & Srivastava et al, 2022) ran a user study where participants solved programming tasks with and without Codex
 - *"Overall, we find that participants who had access to an AI assistant based on OpenAI's codex-davinci-002 model wrote significantly less secure code than those without access"*
 - *"Additionally, participants with access to an AI assistant were more likely to believe they wrote secure code than those without access to the AI assistant."*
- ❑ General psychological phenomenon known as Automation Bias

Concluding remarks

- ❑ Many of these capabilities were completely out of reach until very recently
- ❑ Fascinating intersection between natural and programming languages:
 - Natural language is ambiguous, flexible, contextual
 - Programming languages are unambiguous, rigid, precise
 - Humans can bridge these two, and LMs now start to do as well
- ❑ Programs are a general representation for reasoning: formal mathematics (theorem- proving languages), legal contracts, tool use, ...
- ❑ Extremely active areas of research!