Multi-task learning

One area of recent interesting is *multi-task learning*

- Training a model to implement multiple tasks

A model that implements a single task computes

$$p(\text{output} \mid \text{input})$$

A model that implements several tasks computes

$$p(\text{output} \mid \text{input}, \text{task-id} \ )$$

When training a model for multiple tasks, the training examples would look something like:

$$\begin{array}{llll}(\textsf{\textbf{Translate to French}}, & \text{English text}, & & \text{French Text}) \\ (\textsf{\textbf{Answer the question}}, & \text{document}, & \text{question}, & \text{answer})\end{array}$$

Text is almost a universal encoding so NLP is a natural way of expressing multiple tasks.

So a natural extensions of a Language Model is to solve multiple tasks

- Encode your specific task as an input that can be handled by a Language Model
- That's one advantage of Byte Pair Encoding
    - No special per-task pre-processing needed for a task's training set

We will take the idea of Multi-task learning one step further

- Learning how to solve a task **without** explicitly training a model !

Pre-train, prompt, predict

We have presented the "Unsupervised Pre-Training + Supervised Fine-Tuning" paradigm.

Considering that

- Language models seen to learn universal, task-independent language representation
- Text-to-text is a universal API for NLP tasks

We can raise the question

- Is Supervised Fine-Tuning even necessary ?
- Can a Language Model learn
    - to solve a task that it has not been explicitly trained on
    - by "learning" the task at *test time*
        - no parameter updates

There are some practical impediments to answering this question

- How does the LM model "understand" that it is being asked to solve a particular task ?
- How does the LM model "understand" the input-output relationship involved in the new task ?

The solution to both impediments is to *condition the LLM* by pre-pending *exemplars* (*demonstrations*) at the start of every query

- examples providing "context" in which to evaluate future prompts
- paradigmatic examples demonstrating a prompt and desired response

We will call this the *pre-prompt* or *context*

For example, we can describe Translation between languages with the following pre-prompt

```
Translate English to French

sea otter =>  loutre de mer

peppermint => menthe poivree

plush giraffe => girafe peluche
```

The pre-prompt consists of

- an initial string describing the task: "Translate English to French"
- a number of examples
    - English input, French output, Separated by a =>

The expectation is that when the user presents the prompt

```
cheese =>
```

the model will respond with the French translation of `cheese`.

- the "next words" predicted by the Language Modeling

Note that the exemplars are given at *inference* time **not** training time

- the model's weights are **not updated**
- the examplars only condition the model into generating specific output

This paradigm has been called "Pre-train, Prompt, Predict" (https://arxiv.org/pdf/2107.13586.pdf)

More formally:

- Let C ("context") denote the pre-prompt.
- Let $\mathbf{x}$ denote the "query" (e.g., `cheese =>`)

The unconditional Language Modeling objective

$$p(\mathbf{y}|\mathbf{x})$$

is to create the sequence $\mathbf{y}$ that follows the sequence of prompt $\mathbf{x}$.

Here, the pre-prompt conditions the model's objective

$$p(\mathbf{y}|\mathbf{x}, C)$$

to create the sequence $\mathbf{y}$ that follows from the exemplars C and prompt $\mathbf{x}$.

To turn this into a Language Modeling task using the Universal API

- we need to represent the exemplars and the prompt as a sequence.

We create the sequence $\dot{\mathbf{x}}$ by concatenating

- some number k of exemplars: $\langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \ldots, \langle \mathbf{x}^{(k)}, \mathbf{y}^{(k)} \rangle$
- the prompt string $\mathbf{x}$
- delimiting elements by separator characters $\langle \mathrm{SEP}_1 \rangle$. $\langle \mathrm{SEP}_2 \rangle$

$$
\begin{aligned}
\dot{\mathbf{x}} = \mathrm{concat}(\quad & \mathbf{x}^{(1)}, \langle \mathrm{SEP}_1 \rangle, \mathbf{y}^{(1)}, \langle \mathrm{SEP}_2 \rangle, \\
& \vdots \\
& \mathbf{x}^{(k)}, \langle \mathrm{SEP}_1 \rangle, \mathbf{y}^{(k)}, \langle \mathrm{SEP}_2 \rangle, \\
& \mathbf{x} \\
&)
\end{aligned}
$$

The LLM then computes

$$p(\mathbf{y}|\dot{\mathbf{x}})$$

For convenience, we will just write this as the conditional probability

$$p(\mathbf{y}|\mathbf{x}, C)$$

Zero shot learning: learning to learn

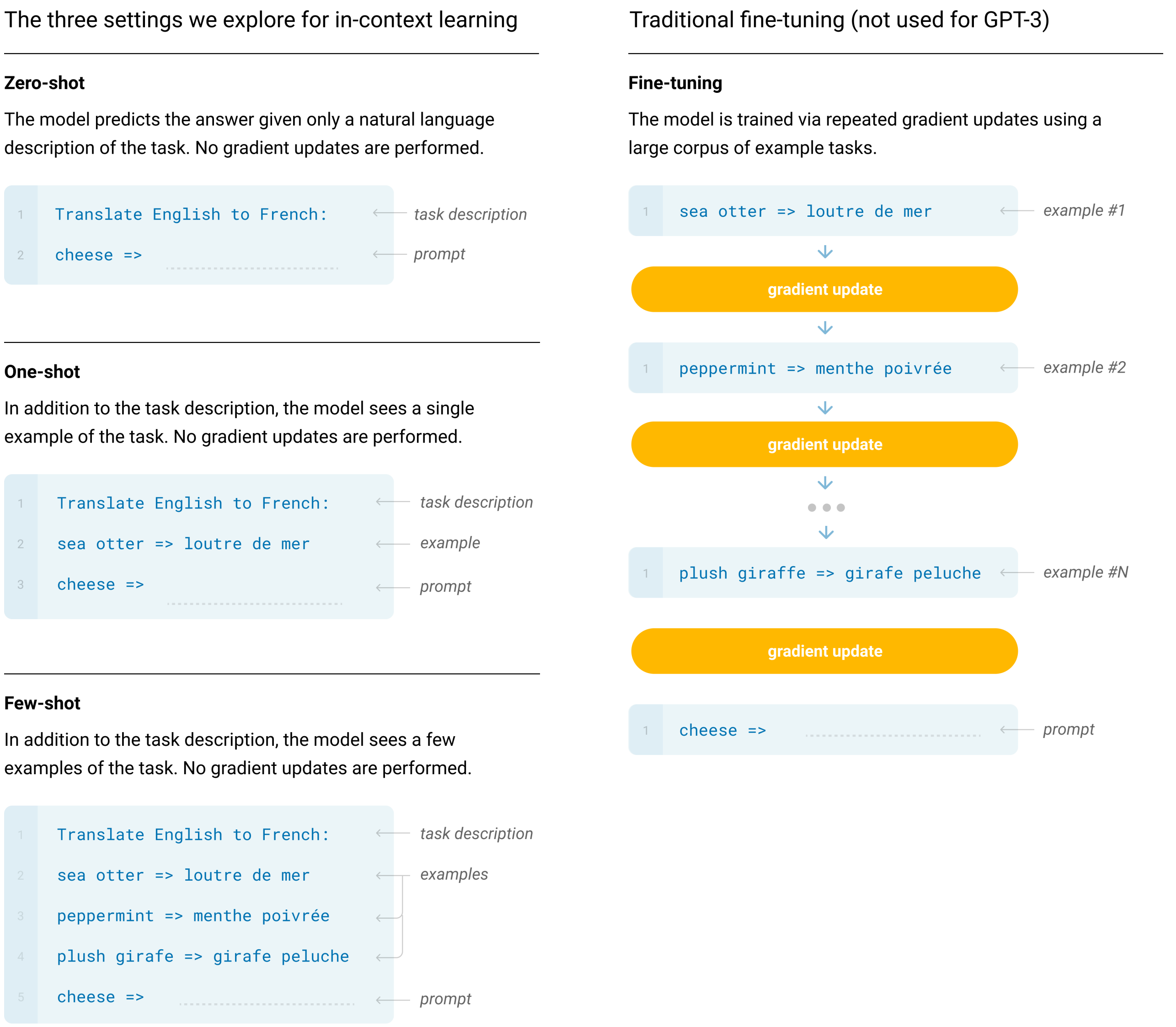
Let k denote the number of exemplars in the pre-prompt.

We can ask how well a LLM performs on an unknown query with varying size of k.

- **Few shot learning**: $10 \leq k \leq 100$ typically
- **One shot learning**: $k = 1$
- **Zero shot learning** $k = 0$

A picture will help

Few/One/Zero shot learning

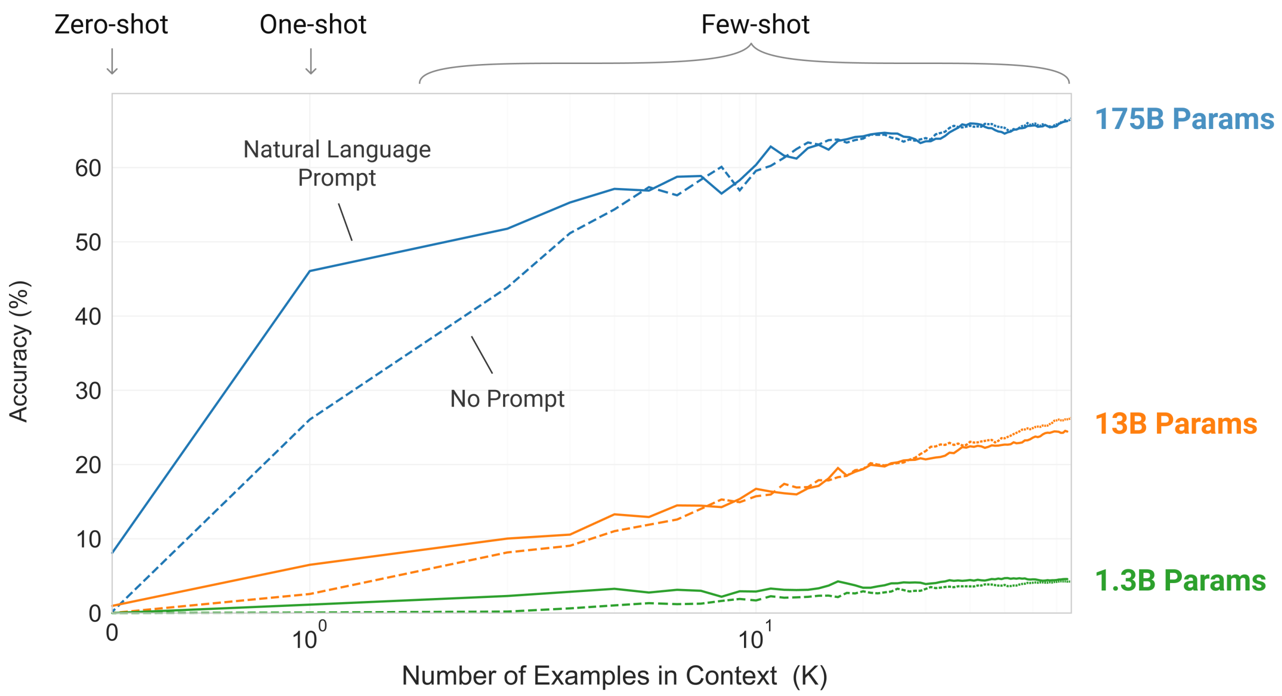


Picture from: https://arxiv.org/pdf/2005.14165.pdf

Is this even possible ?! Learning a new task with **zero** exemplars ?

Let's look at the reported results from the third generation GPT-3 model.

Few/One/Zero shot learning



Picture from: https://arxiv.org/pdf/2005.14165.pdf

Prompt engineering

You see how the behavior of the LLM can be affected by the exact form of the prompt.

There is a whole literature on creating successful prompts: Prompt engineering (https://arxiv.org/pdf/2107.13586.pdf), Chain of thought prompting (https://arxiv.org/pdf/2201.11903.pdf)

- Providing enough context to condition the model to "understand"
    - That "Translate English to French" relates to some examples seen (implicitly) in training
    - and that the string => suggests a relationship between the input and output
        - perhaps generalizing examples seen in training

OpenAI provides helpful examples (https://platform.openai.com/examples) for prompting.

See Appendix G (https://arxiv.org/pdf/2005.14165.pdf#page=51) (pages 50+) for examples of prompts for many other tasks.

Chain of thought prompting

[Paper: Chain of thought prompting (https://arxiv.org/pdf/2201.11903.pdf)](https://arxiv.org/pdf/2201.11903.pdf)

In school, students are often tasked with solving problems involving multiple steps.

LLM's are better at multi-step reasoning tasks when they have been conditioned to answer step by step.

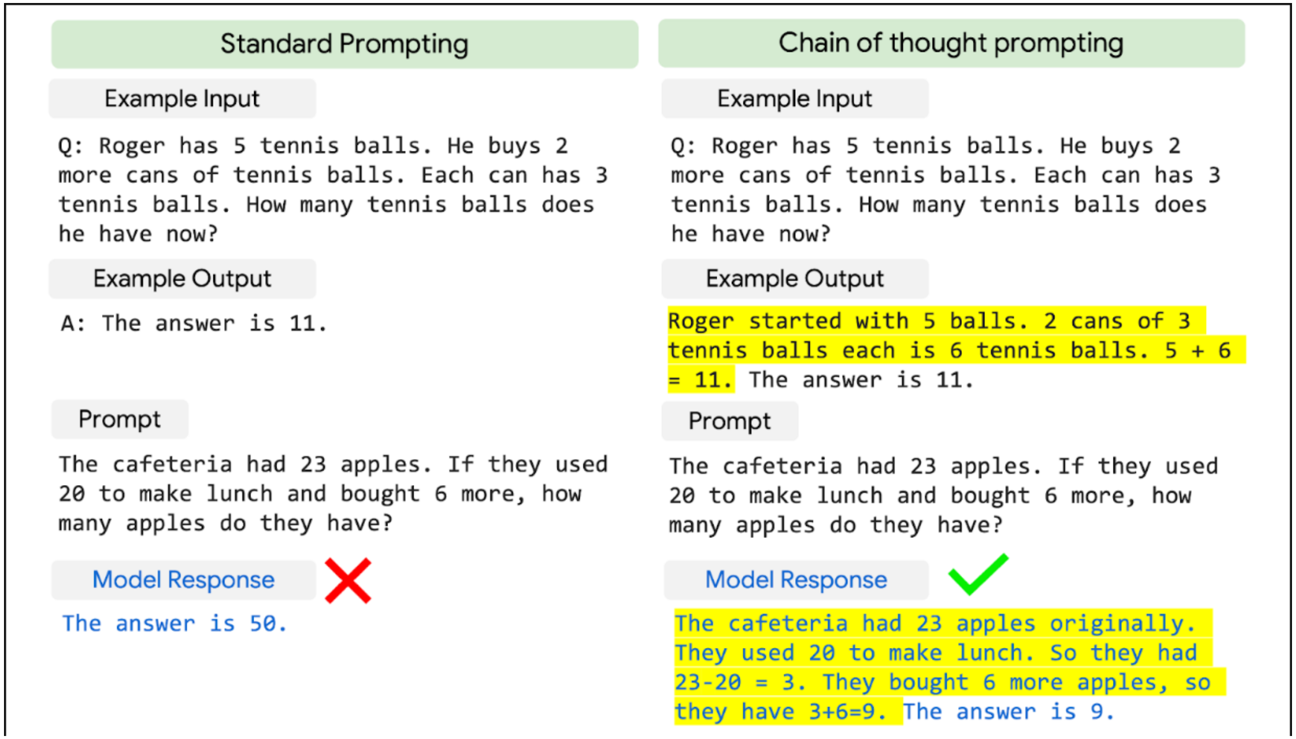
We call this *chain of thought (CoT)* prompting

The exemplars used in CoT prompting

- demonstrate step by step reasoning in the expected output

We can see the difference in the exemplar's "Example Output" section

- using "Standard Prompting" (on the left)
- versus using "CoT Prompting" (on the right)



How does this apply to the case of *zero* exemplars (zero-shot learning) ?

It turns out that step by step reasoning can be elicited

- Just by adding the phrase ["Let's think step by step" (https://arxiv.org/pdf/2205.11916.pdf)](https://arxiv.org/pdf/2205.11916.pdf) to the end of the query

Let's see an example.

Let's ask ChatGPT to solve a multi-step reasoning problem in a zero-shot setting.

As you can see: it comes close, by produces an incorrect answer.

KE

Each can contains 3 balls. I start with 5 cans. At the end, all are empty except for one can with 2 balls. How many balls did I use

You used 15 balls.

This can be determined by starting with the number of cans you have (5) and multiplying it by the number of balls in each can (3) to get 15. Then you subtract the number of balls left in the last can (2) from the total number of balls you started with (15) to get your answer.

Now, let's run the same query but append a request to answer step-by-step.

KE

Each can contains 3 balls. I start with 5 cans. At the end, all are empty except for one can with 2 balls. How many balls did I use. LETS THINK STEP BY STEP

Sure, here is one way to think about the problem step by step:

1. Start by identifying the information given in the problem:
   - Each can contains 3 balls
   - I start with 5 cans
   - At the end, all cans are empty except for one can with 2 balls
2. Next, think about what information you need to find:
   - How many balls did I use
3. Now, you can use the information you have to find the number of balls you used.
   - You started with 5 cans, each containing 3 balls, for a total of 5 cans * 3 balls per can = 15 balls
   - At the end, you have one can with 2 balls remaining, so you must have used 15 balls - 2 balls = 13 balls

So, you used 13 balls.

Using zero-shot to create new applications

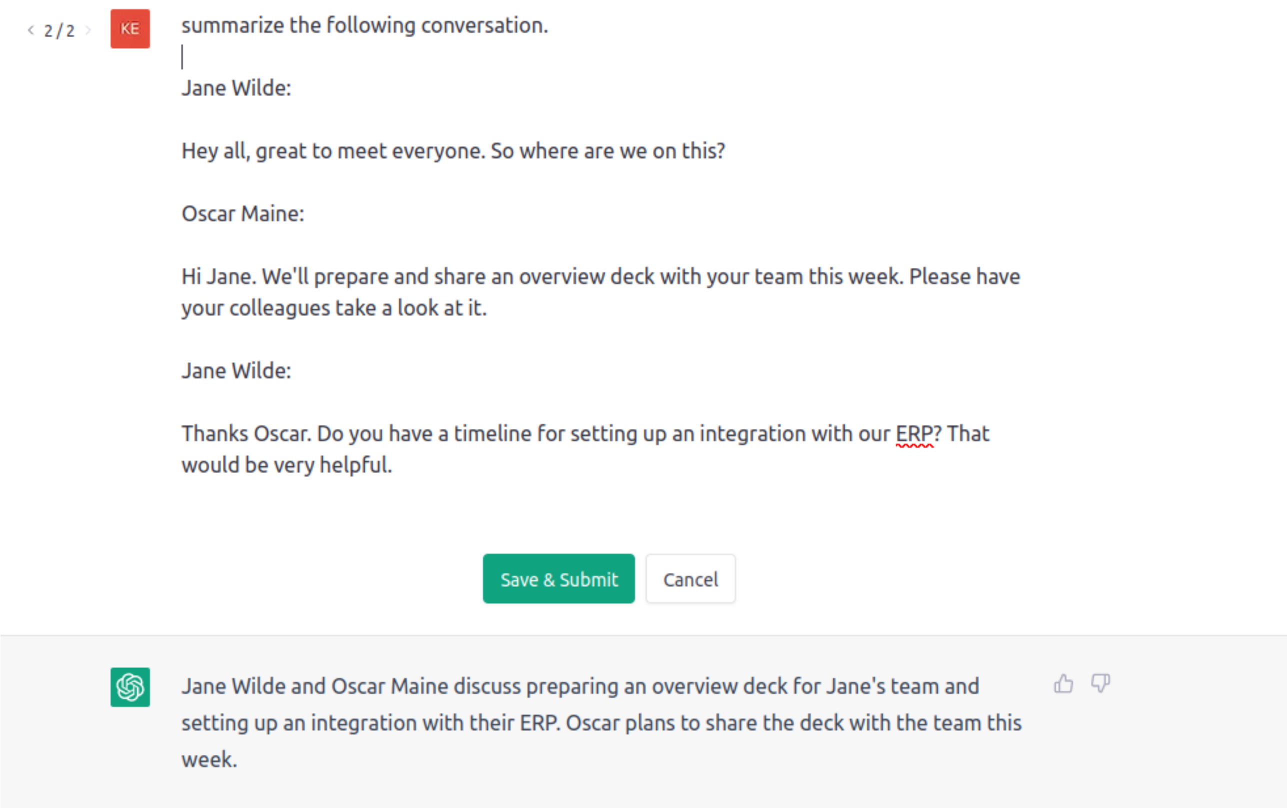
With a little cleverness, one can almost trivially create a new application using a LLM in zero-shot mode

- create the prefix of a prompt describing the task
- append the user input to the prefix to complete the prompt

Here we use `ChatGPT` [(https://chat.openai.com/chat)](https://chat.openai.com/chat) to create an app that summarizes a conversation

- we create a prompt with a "place-holder" (in braces `{..}`) for user input

```
prompt = Summarize the following conversation: {user input}
```



Here we use ChatGPT as a programming assistant

prompt = Write a Python function that does the following: {task description}

KE

Write a Python program that does the following: add a list of numbers

Here is a sample Python program that adds a list of numbers:

Copy code

```
# Define a list of numbers
numbers = [1, 2, 3, 4, 5]

# Initialize the sum to 0
sum = 0

# Iterate over the numbers in the list
for number in numbers:
    # Add the current number to the sum
    sum += number

# Print the final sum
print(sum)
```

Some more, creative examples

- Spreadsheet add-in to perform lookups (https://twitter.com/pavtalk/status/1285410751092416513)
- Generate a web page from a description (https://twitter.com/sharifshameem/status/1283322990625607681)

References found in: http://ai.stanford.edu/blog/understanding-incontext/ (http://ai.stanford.edu/blog/understanding-incontext/)

How is zero-shot learning possible ? Some theories

Theory 1

- The training set contains explicit instances of these out of sample tasks

Theory 2

- The super-large training sets contain *implicit* instances of these out of sample tasks
    - For example: an English-language article quoting a French speaker in French with English translation

One thing that jumps out from the graph:

- Bigger models are more likely to exhibit meta-learning

Theory 3

The training sets are so big that the model "learns" to create groups of examples with a common theme

- Even with the large number of parameters, the model capacity does not suffice for example memorization

Another thing to consider

- The behavior of an RNN depends on *all* previous inputs
    - It has memory (latent state, etc.)

So Few Shot Learning may work by "priming" the memory with parameters for a specific task

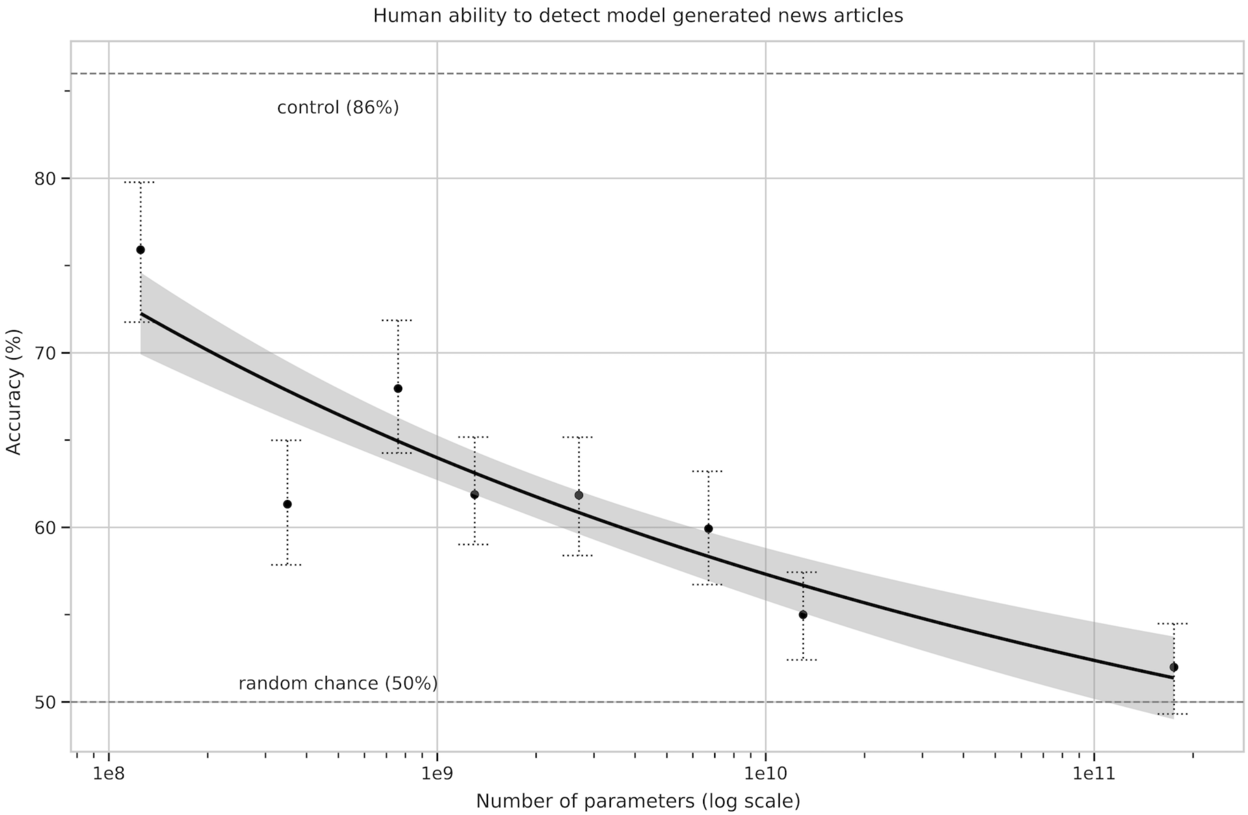
Social concerns

The team behind GPT is very concerned about potential misuse of Language Models.

To illustrate, they conducted an experiment in having a Language Model construct news articles

- Select title/subtitle of a genuine news article
- Have the Language Model complete the article from the title/subtitle
- Show humans the genuine and generated articles and ask them to judge whether the article was written by a human

Human accuracy in detecting model generated news articles



Picture from: https://arxiv.org/pdf/2005.14165.pdf

The bars show the range of accuracy across the 80 human judges.

- 86% accuracy detecting articles created by a really bad model (the control)
- 50% accuracy detecting articles created by the biggest models

It seems that humans might have difficulty distinguishing between genuine and generated articles.

The fear is that Language Models can be used

- to mislead
- to create offensive speech

In [2]:

```python
print("Done")
```

```
Done
```