

Pre-train, prompt, predict

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

- Adapt a LM to a new task with further training

We have also introduced *In Context Learning*

- Adapt a LM to a new task by providing demonstrations of the new Target task behavior
- via examples in the prompt

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

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

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

For example, we can describe Translation between languages with the following context

Translate English to French

sea otter => loutre de mer

peppermint => menthe poivree

plush giraffe => girafe peluche

The *exemplars* (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

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: $\langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \dots, \langle \mathbf{x}^{(k)}, \mathbf{y}^{(k)} \rangle$
- the prompt string \mathbf{x}
- delimiting elements by separator characters $\langle \text{SEP}_1 \rangle, \langle \text{SEP}_2 \rangle$

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

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

Does Pre-Train, Prompt, Predict work ?

We can begin to answer this question by

- examining the behavior of a Pre-Trained LLM
- on a new task
- using k exemplars
 - varying k

Depending on k , we refer to the behavior of the LLM by slightly different names

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

A picture will help

The three settings we explore for in-context learning

Zero-shot

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



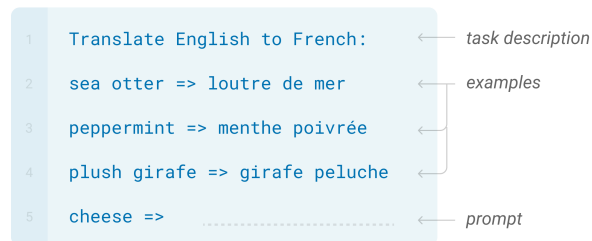
One-shot

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



Few-shot

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



Traditional fine-tuning (not used for GPT-3)

Fine-tuning

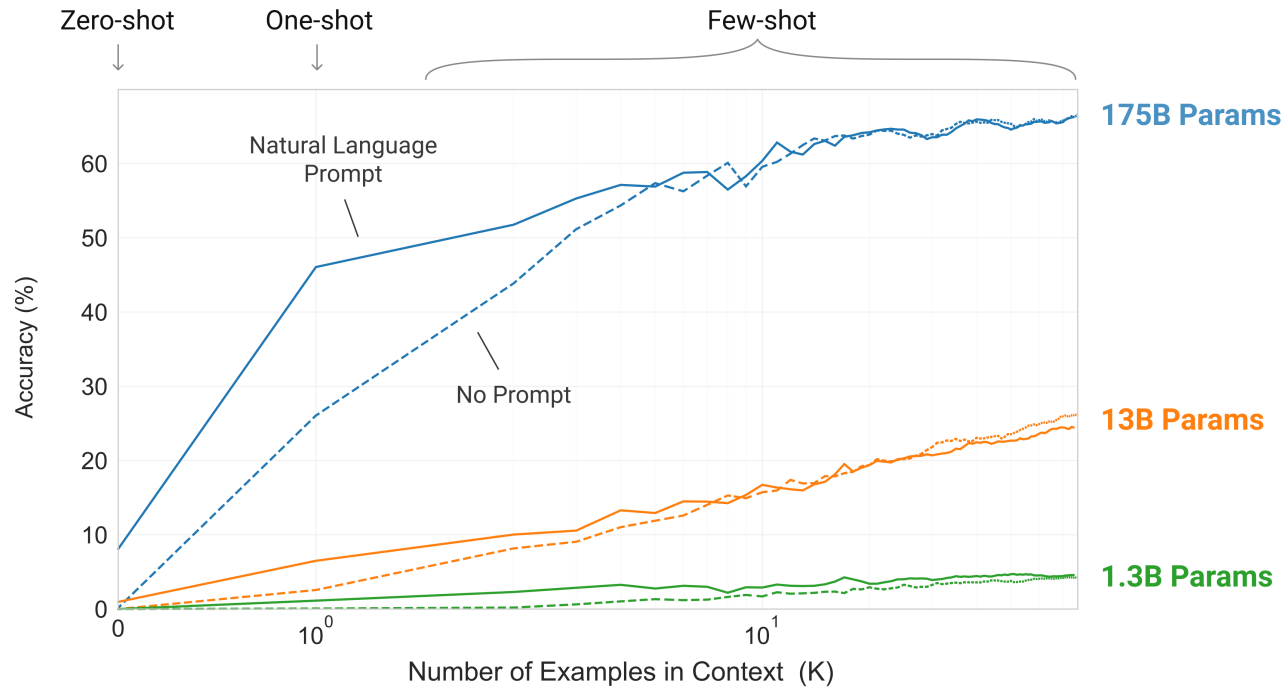
The model is trained via repeated gradient updates using a large corpus of example tasks.



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

Let's look at the reported In-Context Learning results of 3 LLM's of varying size.

Few/One/Zero shot learning



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

A couple of observations

- Bigger models (more weights/parameters) perform better than smaller models
 - compare the 175 Billion parameter model to the smaller models
- More exemplars (greater k) helps
 - but not much for the smallest model
- As the size of the model grows: In-Context Learning behavior improves
 - we sometimes refer to this as behavior that "emerges" only when a model is sufficiently large
- Zero shot learning works !
 - but this is a behavior that only emerges for very large models

Few-shot learning: let's experiment

The [HuggingFace platform \(https://huggingface.co/\)](https://huggingface.co/) has libraries of pre-trained models for many tasks, including Language models.

There is a clean API for using these models in code (I recommend their on-line [course \(https://huggingface.co/\)](https://huggingface.co/) if you want to play with it).

But they also host many of their models for interactive use.

This is valuable not just for the obvious reason of ease of use

- some models are too big to load on the machines available to us

For fun, let's try using k-shot learning in order to get a Pre-Trained Language model to classify whether a short movie review is positive or negative.

Movie review sentiment: few shot learning GPT-2 (<https://huggingface.co/gpt2?text=this+movie+was+great%3A+positive%0A%0A+one+of+the+best+films+of+the+year>)

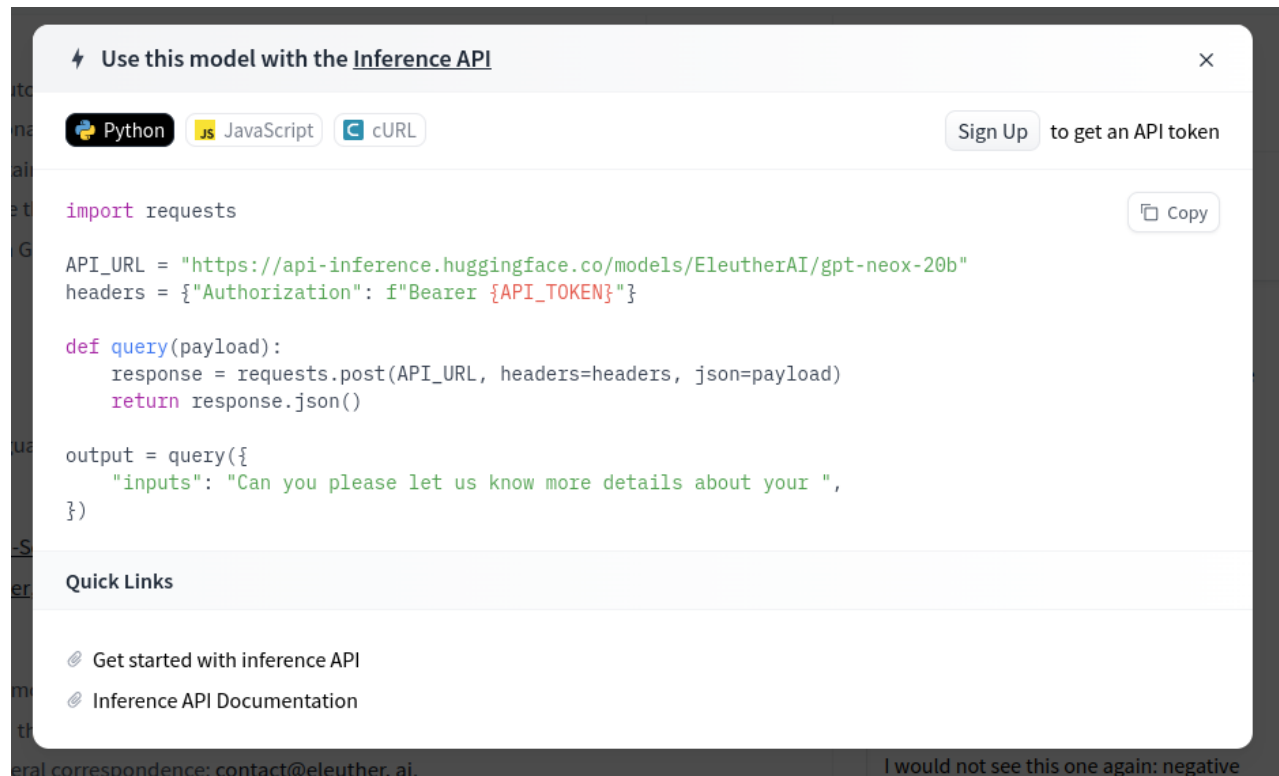
Movie review sentiment: few shot learning GPT-J 6B (<https://huggingface.co/EleutherAI/gpt-j-6B?text=this+movie+was+great%3A+positive%0A%0A+one+of+the+best+films+of+the+year>)

Movie review sentiment: few shot learning: gpt-neox-20b (<https://huggingface.co/EleutherAI/gpt-neox-20b?text=this+movie+was+great%3A+positive%0A%0A+one+of+the+best+films+of+the+year>)

You can try cutting and pasting the prompt into the hosted inference instance of other models.

If you click on the Deploy button and choose the Inference API drop-down

- you will see Python code for querying the model programmatically.



The screenshot shows a modal window titled "Use this model with the Inference API". It features three tabs for different languages: Python (selected), JavaScript, and cURL. A "Sign Up" button is present next to the text "to get an API token". Below the tabs, a code editor displays Python code for querying the API. A "Copy" button is located to the right of the code. At the bottom, there is a "Quick Links" section with two links: "Get started with inference API" and "Inference API Documentation".

```
import requests

API_URL = "https://api-inference.huggingface.co/models/EleutherAI/gpt-neox-20b"
headers = {"Authorization": f"Bearer {API_TOKEN}"}

def query(payload):
    response = requests.post(API_URL, headers=headers, json=payload)
    return response.json()

output = query({
    "inputs": "Can you please let us know more details about your ",
})
```

Quick Links

- 🔗 Get started with inference API
- 🔗 Inference API Documentation

For general correspondence: contact@eleuther.ai | I would not see this one again: negative

Prompt engineering

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

- the number of exemplars

It is also the case that slightly changing the words (and order of words) affects behavior.

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

OpenAI provides [helpful examples](https://platform.openai.com/examples) (<https://platform.openai.com/examples>) for prompting.

See [Appendix G](https://arxiv.org/pdf/2005.14165.pdf#page=51) (<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>)

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)

Standard Prompting	Chain of thought prompting
<p>Example Input</p> <p>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?</p>	<p>Example Input</p> <p>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?</p>
<p>Example Output</p> <p>A: The answer is 11.</p>	<p>Example Output</p> <p>Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p>
<p>Prompt</p> <p>The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Prompt</p> <p>The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Response</p> <p>The answer is 50.</p>	<p>Model Response</p> <p>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.</p>

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>) 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, but produces an incorrect answer.



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.

The reasoning logic is correct, but the math is wrong.

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



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 $5 \text{ cans} * 3 \text{ balls per can} = 15 \text{ balls}$
 - At the end, you have one can with 2 balls remaining, so you must have used $15 \text{ balls} - 2 \text{ balls} = 13 \text{ balls}$

So, you used 13 balls.

The "LETS THINK STEP BY STEP" seems to condition the model into getting the math correct

- Without CoT: answers comes before reasoning
- With CoT: reasoning precedes answer

It is the conditioning of providing the reasoning before the answer that seems to improve the behavior.

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>) 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}

< 2/2 >  summarize the following conversation.
|
Jane Wilde:

Hey all, great to meet everyone. So where are we on this?

Oscar Maine:




Hi Jane. We'll prepare and share an overview deck with your team this week. Please have your colleagues take a look at it.

Jane Wilde:

Thanks Oscar. Do you have a timeline for setting up an integration with our ERP? That would be very helpful.

Save & Submit

Cancel

 Jane Wilde and Oscar Maine discuss preparing an overview deck for Jane's team and setting up an integration with their ERP. Oscar plans to share the deck with the team this week.  

Here we use ChatGPT as a programming assistant


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



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

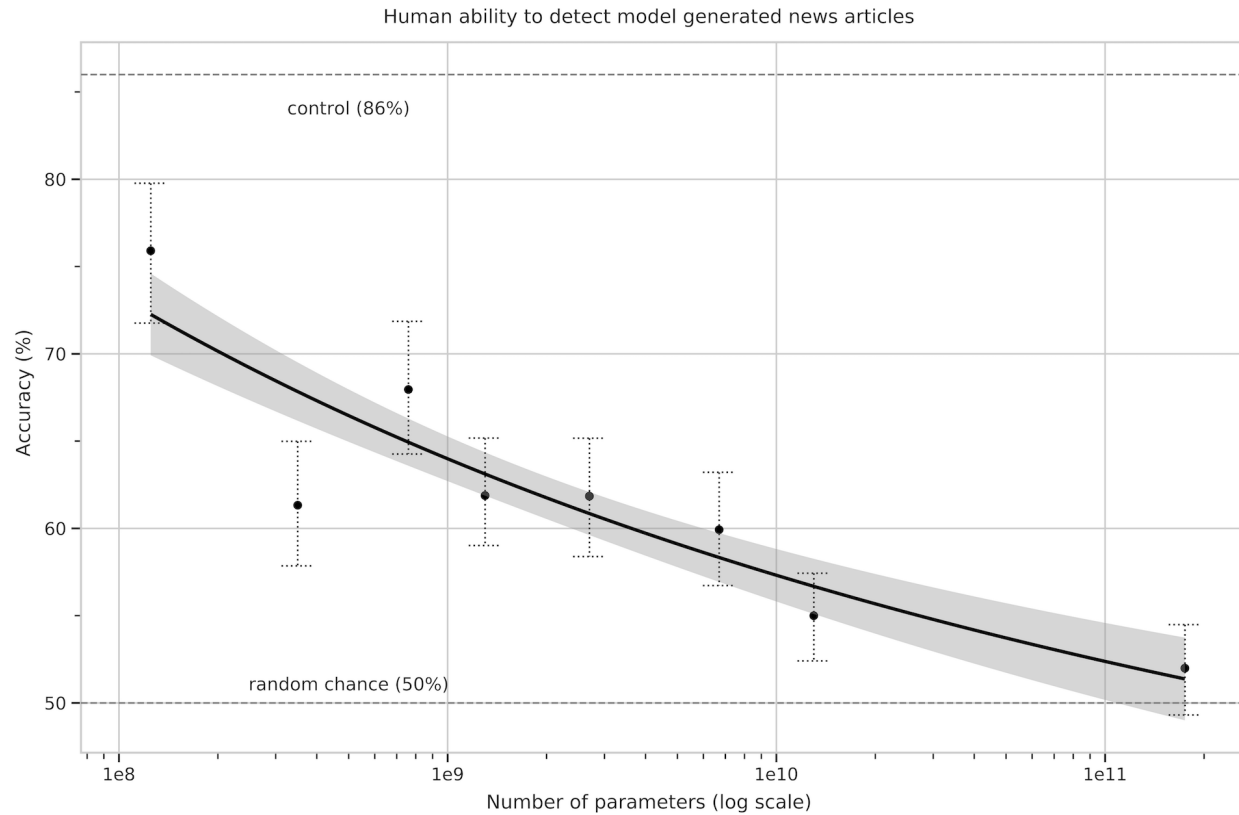
Social concerns

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]: `print("Done")`

Done

