COMP0087 25/26
Lecture 6: Prompting

30/01/2026

Code examples: https://github.com/yaolu/prompt

A Brief History of Prompt Learning

Language Models are Unsupervised Multitask Learners

”I’m not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile [I’m not a fool].**

In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: ”**Mentez mentez, il en restera toujours quelque chose**,” which translates as, ”**Lie lie and something will always remain.**”

“I hate the word ‘**perfume**,”’ Burr says. ‘It’s somewhat better in French: ‘**parfum**.’

If listened carefully at 29:55, a conversation can be heard between two guys in French: “**-Comment on fait pour aller de l’autre coté? -Quel autre coté?**”, which means “**- How do you get to the other side? - What side?**”.

If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?

“**Brevet Sans Garantie Du Gouvernement**”, translated to English: “**Patented without government warranty**”.

Table 1. Examples of naturally occurring demonstrations of English to French and French to English translation found throughout the WebText training set.

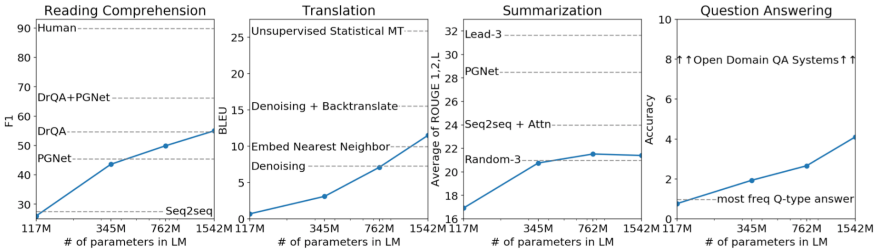


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural Questions (Kwiatkowski et al., 2019). Section 3 contains detailed descriptions of each result.

Due to concerns about large language models being used to generate deceptive, biased, or abusive language at scale, we are only releasing a much smaller version of GPT-2 along with sampling code. We are not releasing the dataset, training code, or GPT-2 model weights.

Time Travel to 2019: GPT-2

Example Code 1

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Our group is part of the UCL Computer Science department, affiliated with "
    "CSML and based at 90, High Holborn, London. We also organise the South "
    "England Natural Language Processing Meetup. If you are interested in "
    "doing a PhD with us, please have a look at these instructions. We also "
    "host a weekly reading group, you can find more details here."
)
# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=32)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output:

```
We are also a member of the University of Cambridge's Computer Science Department. We are also a member of the University of Cambridge's Computer Science Department.
```

Time Travel to 2019: GPT-2

Example Code 2

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Our group is part of the UCL Computer Science department, affiliated with "
    "CSML and based at 90, High Holborn, London. We also organise the South "
    "England Natural Language Processing Meetup. If you are interested in "
    "doing a PhD with us, please have a look at these instructions. We also "
    "host a weekly reading group, you can find more details here."
    " TL;DR"
)
# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=32)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output:

```
: We are a group of computer scientists who are interested in learning about the world of computer science. We are looking for people who are interested in learning about the
```

Time Travel to 2019: GPT-2

Example Code 3

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Our group is part of the UCL Computer Science department, affiliated with "
    "CSML and based at 90, High Holborn, London. We also organise the South "
    "England Natural Language Processing Meetup. If you are interested in "
    "doing a PhD with us, please have a look at these instructions. We also "
    "host a weekly reading group, you can find more details here."
    " tl;dr"
)
# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=32)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output:

```
: We are a group of computer scientists who have been working on the problem of language processing for over 20 years.
We are a group of people who have been
```

Time Travel to 2019: GPT-2

Example Code 4

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Our group is part of the UCL Computer Science department, affiliated with "
    "CSML and based at 90, High Holborn, London. We also organise the South "
    "England Natural Language Processing Meetup. If you are interested in "
    "doing a PhD with us, please have a look at these instructions. We also "
    "host a weekly reading group, you can find more details here."
    " tldr"
)
# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=32)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output:

```
.org.uk The University of Cambridge The University of Cambridge is a research university in the UK. It is a research university with a focus
```

```
# whois.nic.uk

    No match for "tldr.org.uk".

    This domain name has not been registered.

    WHOIS lookup made at 04:56:15 01-Mar-2024
```

Time Travel to 2019: GPT-2

Example Code 5

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Our group is part of the UCL Computer Science department, affiliated with "
    "CSML and based at 90, High Holborn, London. We also organise the South "
    "England Natural Language Processing Meetup. If you are interested in "
    "doing a PhD with us, please have a look at these instructions. We also "
    "host a weekly reading group, you can find more details here."
    " tldr:"
)
# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=32)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output:

```
http://www.tldr.org/ The University of Cambridge The University of Cambridge is a research university in the UK. It is
```

Language Model as Unsupervised Task Learner

TL;DR	Yes
tl;dr	Yes
tldr	No
tldr:	No
TL;dr	?
summary:	?

Time Travel to 2020: GPT-3

This article is more than **3 years old**

Elon Musk-backed OpenAI to release text tool it called dangerous

The API gives firms access to a text generation AI for use in coding and data entry

OpenAI was founded with a $1bn endowment in 2015, backed by Elon Musk. He has since left the board, but remains as a donor. Photograph: Hannibal Hanschke/Reuters

OpenAI, the machine learning nonprofit co-founded by Elon Musk, has released its first commercial product: a rentable version of a text generation tool the organisation once deemed too dangerous to release.

Dubbed simply "the API", the new service lets businesses directly access the most powerful version of GPT-3, OpenAI's general purpose text generation AI.

Language Models are Few-Shot Learners

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions - something which current NLP systems still largely struggle to do. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning approaches. Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters, 10x more than any previous non-sparse language model, and test its performance in the few-shot setting. 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. GPT-3 achieves strong performance on many NLP datasets, including translation, question-answering, and cloze tasks, as well as several tasks that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic. At the same time, we also identify some datasets where GPT-3's few-shot learning still struggles, as well as some datasets where GPT-3 faces methodological issues related to training on large web corpora. Finally, we find that GPT-3 can generate samples of news articles which human evaluators have difficulty distinguishing from articles written by humans. We discuss broader societal impacts of this finding and of GPT-3 in general.

Time Travel to 2020: GPT-3

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
```

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
```

Few-shot

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

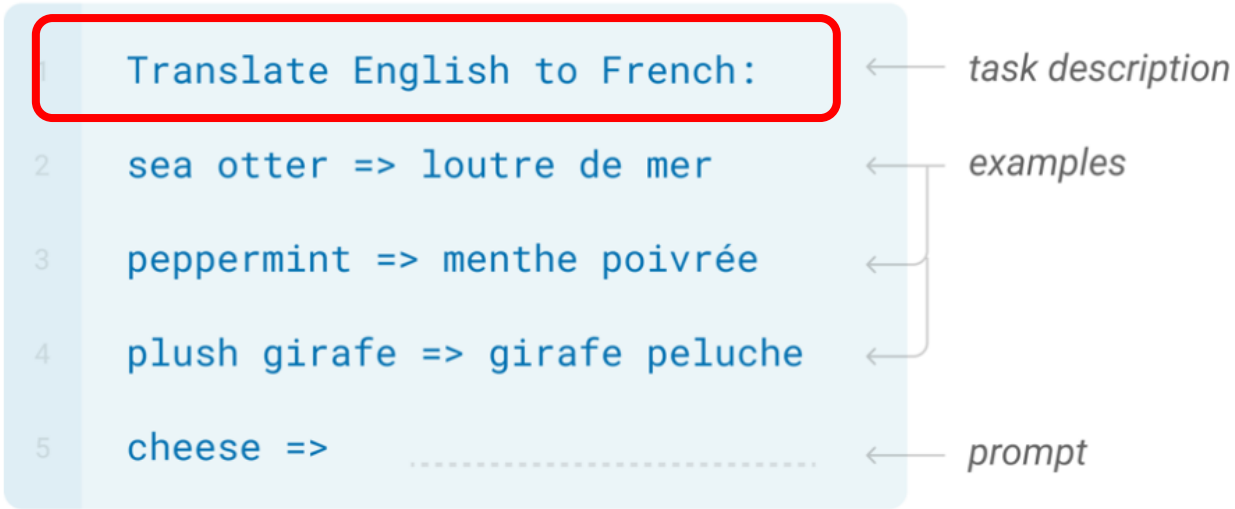
```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

Task Description
Examples (Demonstrations)
Prompt

Time Travel to 2020: GPT-3

Few-shot

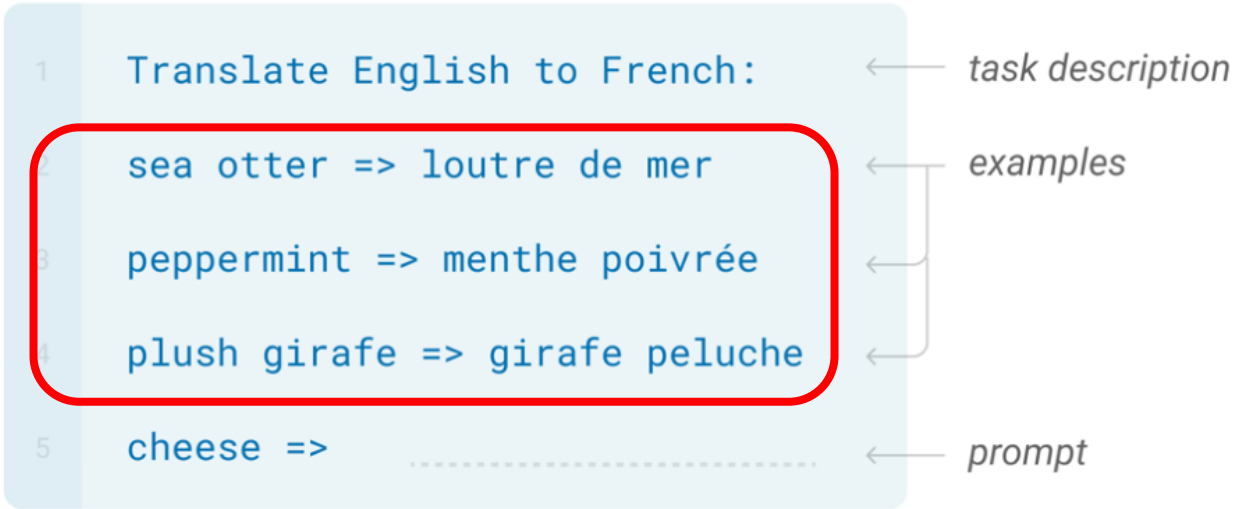
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Time Travel to 2020: GPT-3

Few-shot

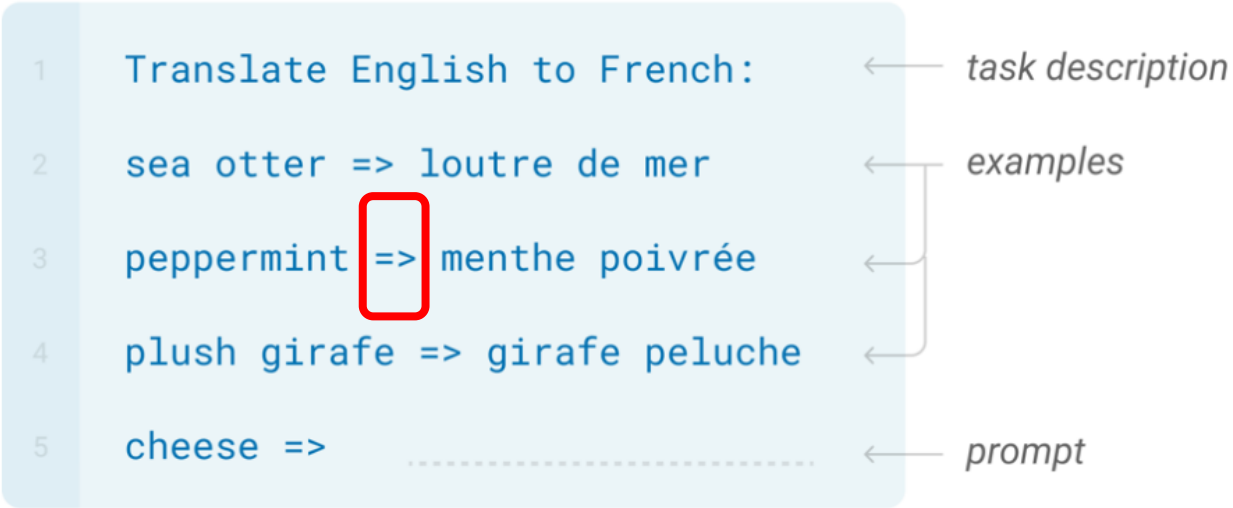
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Time Travel to 2020: GPT-3

Few-shot

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



Show me the code …

Example Code 2-1: worst movie of this year

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Review: featuring an oscar-worthy performance\nSentiment: positive\n"
    "Review: completely messed up\nSentiment: negative\n"
    "Review: masterpiece\nSentiment: positive\n"
    "Review: the action is stilted\nSentiment: negative\n"
    "Review: by far the worst movie of the year\nSentiment:"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=1)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output: `negative`

Let’s try more

Example Code 2-2: best movie of this year

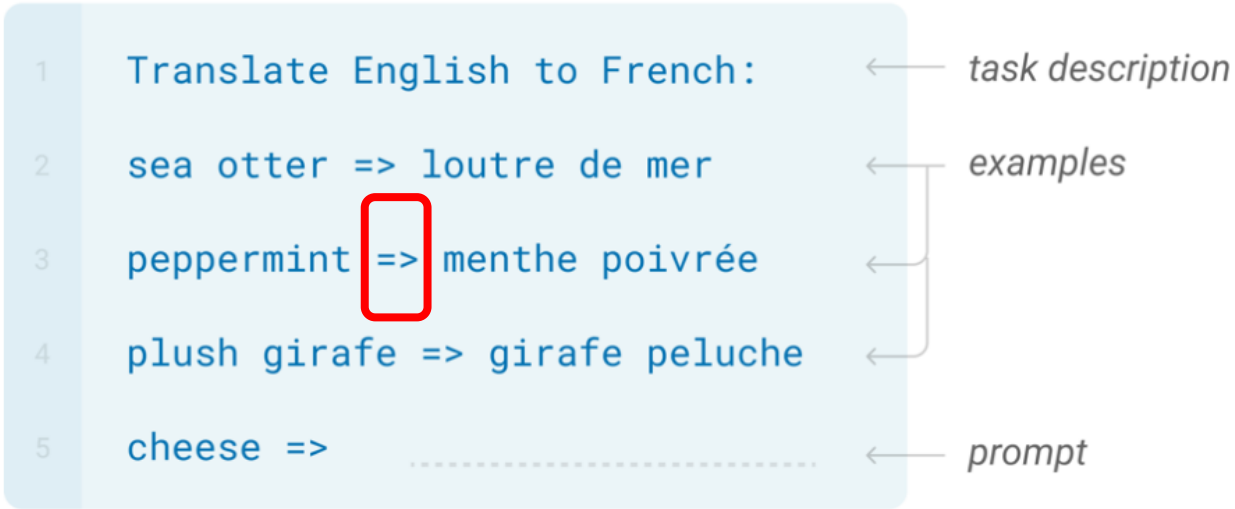
```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Review: featuring an oscar-worthy performance\nSentiment: positive\n"
    "Review: completely messed up\nSentiment: negative\n"
    "Review: masterpiece\nSentiment: positive\n"
    "Review: the action is stilted\nSentiment: negative\n"
    "Review: by far the best movie of the year\nSentiment:"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=1)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output: `negative`

How to solve it?

Few-shot

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



Maybe => magic?

Example Code 2-3: (=>) magic

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Review: featuring an oscar-worthy performance => Sentiment: positive\n"
    "Review: completely messed up => Sentiment: negative\n"
    "Review: masterpiece => Sentiment: positive\n"
    "Review: the action is stilted => Sentiment: negative\n"
    "Review: by far the best movie of the year => Sentiment:"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=1)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output: `positive`

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "Review: featuring an oscar-worthy performance => Sentiment: positive\n"
    "Review: completely messed up => Sentiment: negative\n"
    "Review: masterpiece => Sentiment: positive\n"
    "Review: the action is stilted => Sentiment: negative\n"
    "Review: by far the worst movie of the year => Sentiment:"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=1)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output: `negative`

Maybe => magic?

Replace newline with =>

100% accuracy

With four examples

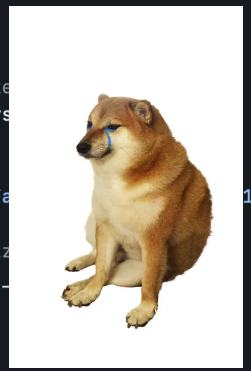
Try more openai magic

Example Code 2-4: use GPT-3 template

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "featuring an oscar-worthy performance => positive\n"
    "completely messed up => negative\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the best movie of the year =>"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=1)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output: `positive`

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "featuring an oscar-worthy performance => positive\n"
    "completely messed up => negative\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the worst movie of the year =>"
)

# Generate input IDs from the document using the toke
input_ids = tokenizer.encode(document, return_tensors

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=Fa                1)

# Decode the model output into text using the tokeniz
output_text = tokenizer.decode(model_output[0, input_

# Print the output text
print(output_text)
```



Output: `positive`

Unpredictable performance.

Prompt Engineer 101: Never be overconfident.

```
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

- Example selection
- Template format
- Example ordering
- Label selection
- …

Take a closer look at failure case

Example Code 2-4: use GPT-3 template

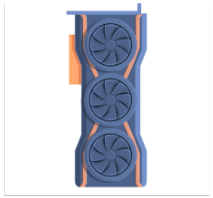
```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "featuring an oscar-worthy performance => positive\n"
    "completely messed up => negative\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the best movie of the year =>"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=1)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output: `positive`

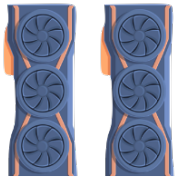
```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "featuring an oscar-worthy performance => positive\n"
    "completely messed up => negative\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the worst movie of the year =>"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

# Generate model output using the input IDs
model_output = model.generate(input_ids, do_sample=False, max_new_tokens=1)

# Decode the model output into text using the tokenizer
output_text = tokenizer.decode(model_output[0, input_ids.shape[1]:])

# Print the output text
print(output_text)
```

Output: `positive`

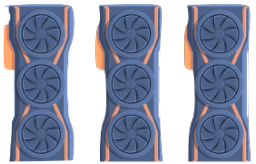
Larger Model?



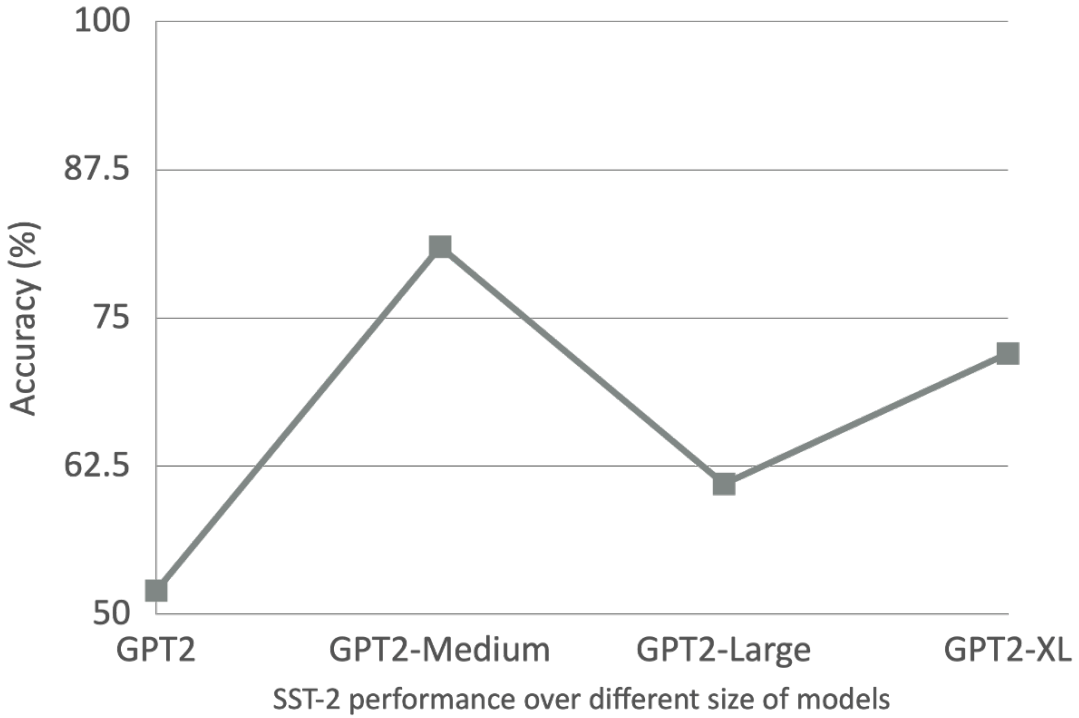
Negative



Positive



Negative



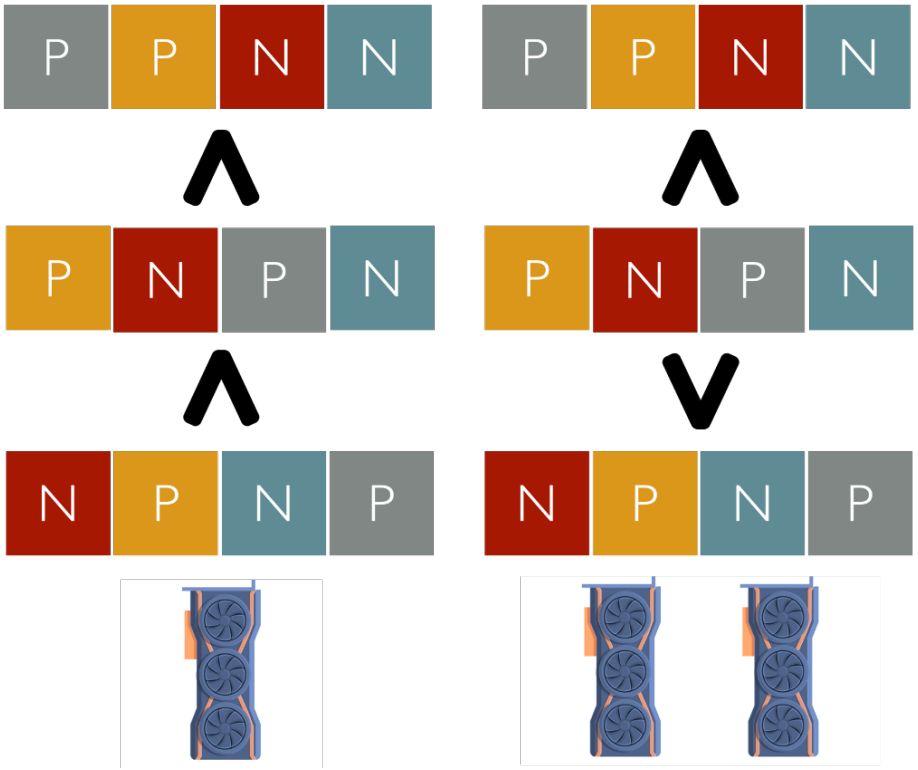
Change the ordering: try again

```
document_a = (
    "completely messed up => negative\n"
    "the action is stilted => negative\n"
    "masterpiece => positive\n"
    "featuring an oscar-worthy performance => positive\n"
    "by far the worst movie of the year =>"
)

document_b = (
    "featuring an oscar-worthy performance => positive\n"
    "masterpiece => positive\n"
    "completely messed up => negative\n"
    "the action is stilted => negative\n"
    "by far the worst movie of the year =>"
)

document_c = (
    "completely messed up => negative\n"
    "featuring an oscar-worthy performance => positive\n"
    "the action is stilted => negative\n"
    "masterpiece => positive\n"
    "by far the worst movie of the year =>"
)
```

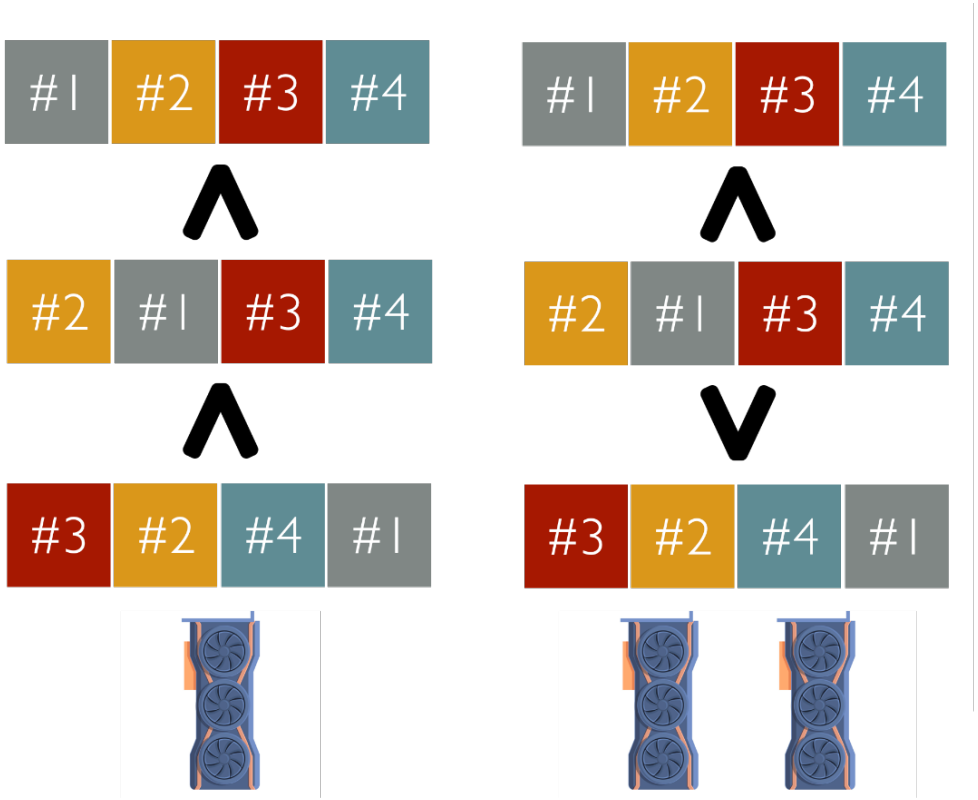
```
document_a = (
    "completely messed up => negative\n"
    "the action is stilted => negative\n"
    "masterpiece => positive\n"
    "featuring an oscar-worthy performance => positive\n"
    "by far the best movie of the year =>"
)

document_b = (
    "featuring an oscar-worthy performance => positive\n"
    "masterpiece => positive\n"
    "completely messed up => negative\n"
    "the action is stilted => negative\n"
    "by far the best movie of the year =>"
)

document_c = (
    "completely messed up => negative\n"
    "featuring an oscar-worthy performance => positive\n"
    "the action is stilted => negative\n"
    "masterpiece => positive\n"
    "by far the best movie of the year =>"
)
```

Ordering A: (positive, positive) 50%
Ordering B: (positive, negative) 100%
Ordering C: (positive, positive) 50%

Label Ordering?



Example Ordering?



Order Sensitivity (positional bias?) of Prompts

Review: I like this movie.
Sentiment: Good

Review: I hate this movie.
Sentiment: Bad

Review: Excellent!.
Sentiment: ???

Accuracy: 50 %

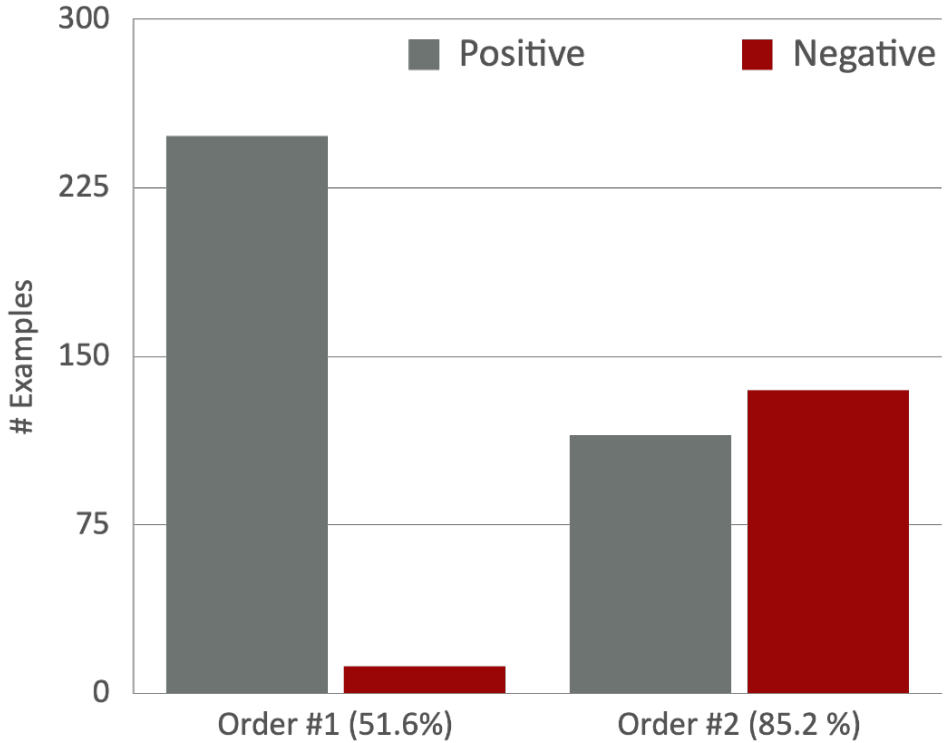
Review: I hate this movie.
Sentiment: Bad

Review: I like this movie.
Sentiment: Good

Review: Excellent!.
Sentiment: ???

Accuracy: 85 %

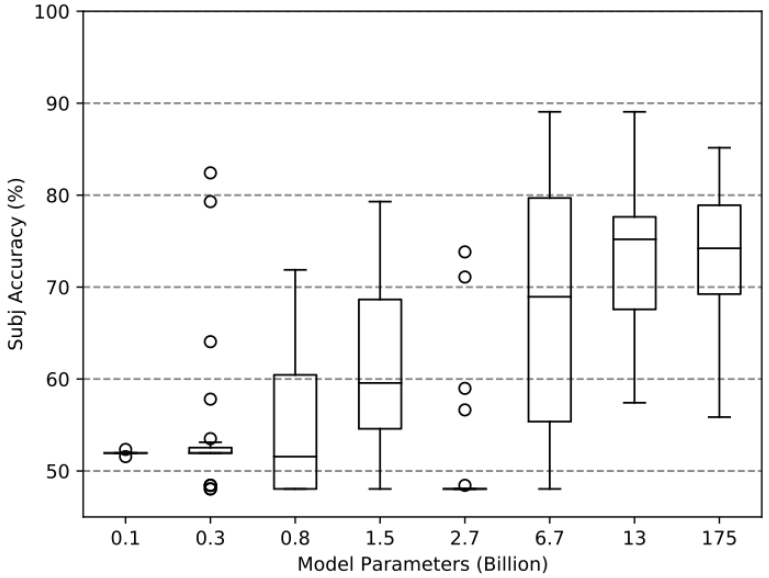
A deeper look



Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, Pontus Stenetorp

When primed with only a handful of training samples, very large, pretrained language models such as GPT-3 have shown competitive results when compared to fully-supervised, fine-tuned, large, pretrained language models. We demonstrate that the order in which the samples are provided can make the difference between near state-of-the-art and random guess performance: essentially some permutations are "fantastic" and some not. We analyse this phenomenon in detail, establishing that: it is present across model sizes (even for the largest current models), it is not related to a specific subset of samples, and that a given good permutation for one model is not transferable to another. While one could use a development set to determine which permutations are performant, this would deviate from the true few-shot setting as it requires additional annotated data. Instead, we use the generative nature of language models to construct an artificial development set and based on entropy statistics of the candidate permutations on this set, we identify performant prompts. Our method yields a 13% relative improvement for GPT-family models across eleven different established text classification tasks.



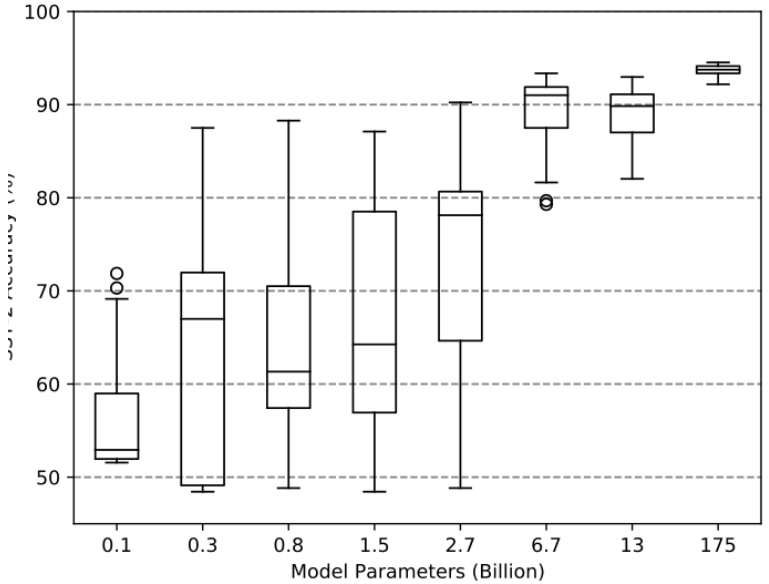


Figure 1: Four-shot performance for 24 different sample orders across different sizes of GPT-family models (GPT-2 and GPT-3) for the SST-2 and Subj datasets.

Calibrate Before Use: Improving Few-Shot Performance of Language Models

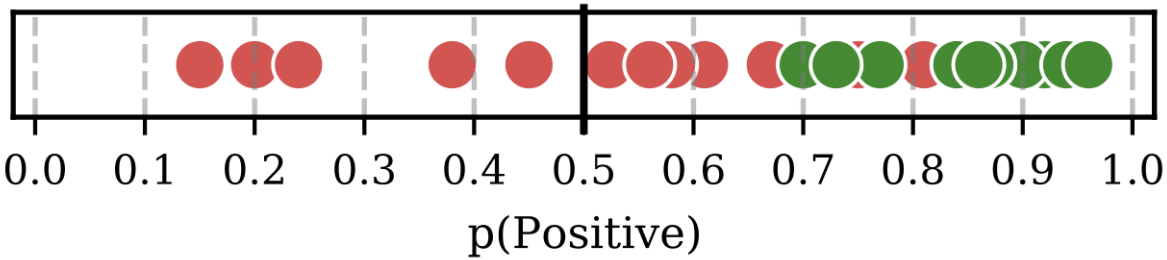
Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, Sameer Singh

GPT-3 can perform numerous tasks when provided a natural language prompt that contains a few training examples. We show that this type of few-shot learning can be unstable: the choice of prompt format, training examples, and even the order of the training examples can cause accuracy to vary from near chance to near state-of-the-art. We demonstrate that this instability arises from the bias of language models towards predicting certain answers, e.g., those that are placed near the end of the prompt or are common in the pre-training data. To mitigate this, we first estimate the model's bias towards each answer by asking for its prediction when given the training prompt and a content-free test input such as "N/A". We then fit calibration parameters that cause the prediction for this input to be uniform across answers. On a diverse set of tasks, this contextual calibration procedure substantially improves GPT-3 and GPT-2's average accuracy (up to 30.0% absolute) and reduces variance across different choices of the prompt.

Comments: ICML 2021

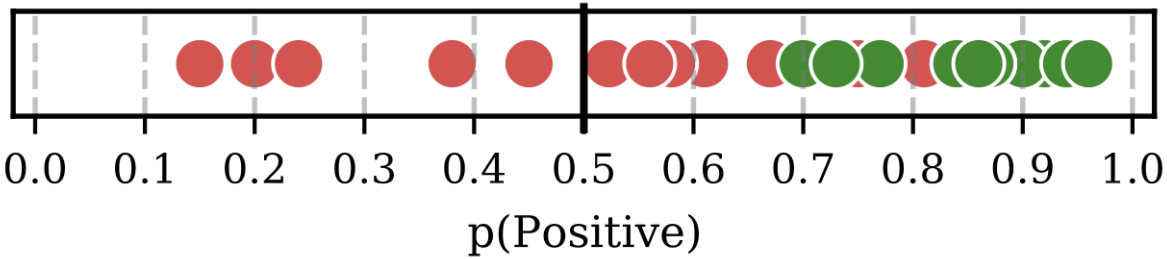
Calibration of prompting predictions

The Impact of Biases on Model Predictions We find that the end result of the above three biases is typically a simple shift in the model's output distribution. For example, Figure 5 visualizes this shift for a SST-2 sentiment prompt.



Calibration of prompting predictions

The Impact of Biases on Model Predictions We find that the end result of the above three biases is typically a simple shift in the model's output distribution. For example, Figure 5 visualizes this shift for a SST-2 sentiment prompt.



$$\hat{\mathbf{q}} = \text{softmax}(\mathbf{W}\hat{\mathbf{p}} + \mathbf{b}), \quad (1)$$

where a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$ are applied to the original probabilities $\hat{\mathbf{p}}$ to get the new probabilities

Calibration of prompting predictions

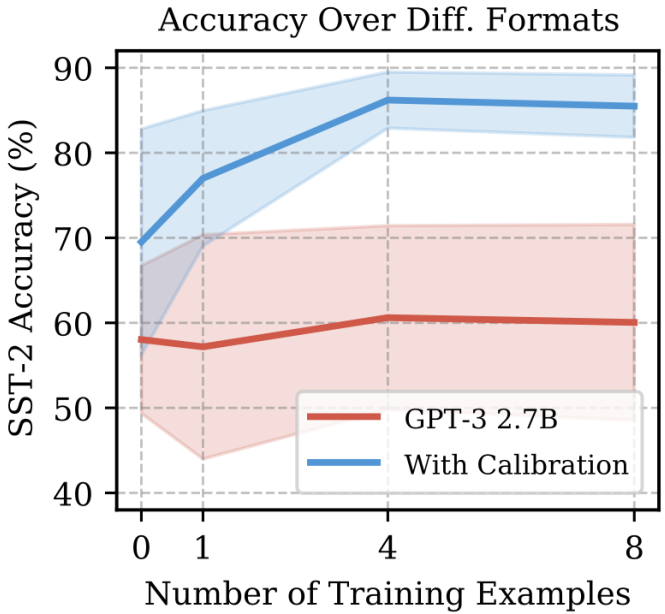


Figure 7. GPT-3 has high variance across different prompt formats; contextual calibration reduces this variance and improves mean accuracy. We show the mean accuracy (± standard deviation) over 15 different prompt formats for SST-2.

Input: Subpar acting. Sentiment: Negative
Input: Beautiful film. Sentiment: Positive
Input: N/A Sentiment:

Content-free Input	SST-2	AGNews
Uncalibrated Baseline	66.5	48.5
N/A	74.2	64.5
[MASK]	74.5	63.8
''	72.9	64.7
N/A, [MASK], ''	79.0	66.5
the	69.1	59.0
abc	77.5	57.3
the man.	79.4	62.0
dasjhasjkdhjskdhds	79.3	64.5
nfjkhdvy84tr9bpuirvwe	78.4	65.5

Table 3. We show the accuracy for 1-shot SST-2 and 0-shot AGNews over different choices for the content-free input. The choice of content-free input matters, however, *many good choices exist*. The token '' indicates the empty string. Recall that in our experiments, we ensemble over N/A, [MASK], and the empty string.

Coding with distribution and calibrate it

Example code 3-2: let's take a look at distribution

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "featuring an oscar-worthy performance => positive\n"
    "completely messed up => negative\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the worst movie of the year =>"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

positive_token_id = 3967
negative_token_id = 4633
with torch.inference_mode():
    model_output = model(input_ids)
    prob_dist = model_output.logits[:, -1, :].softmax(dim=-1)
print(f"positive probability: {prob_dist[:, positive_token_id]}")
print(f"negative probability: {prob_dist[:, negative_token_id]}")
```

Output: `positive probability: tensor([0.3157])  negative probability: tensor([0.2891])`

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

document = (
    "featuring an oscar-worthy performance => positive\n"
    "completely messed up => negative\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the best movie of the year =>"
)

# Generate input IDs from the document using the tokenizer
input_ids = tokenizer.encode(document, return_tensors='pt')

positive_token_id = 3967
negative_token_id = 4633
with torch.inference_mode():
    model_output = model(input_ids)
    prob_dist = model_output.logits[:, -1, :].softmax(dim=-1)
print(f"positive probability: {prob_dist[:, positive_token_id]}")
print(f"negative probability: {prob_dist[:, negative_token_id]}")
```

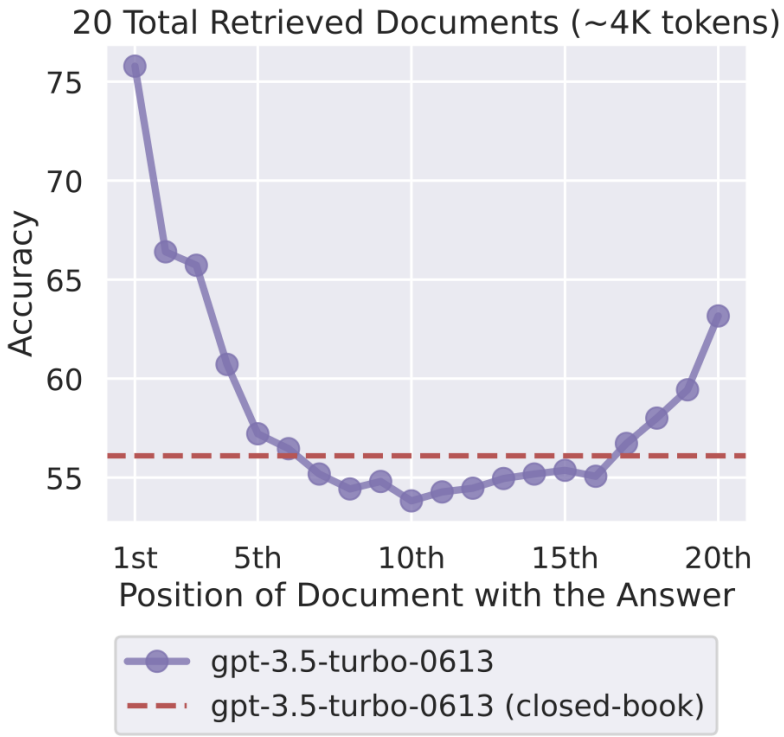
Output: `positive probability: tensor([0.3802])  negative probability: tensor([0.1540])`

Position Bias for Modern Models: ChatGPT

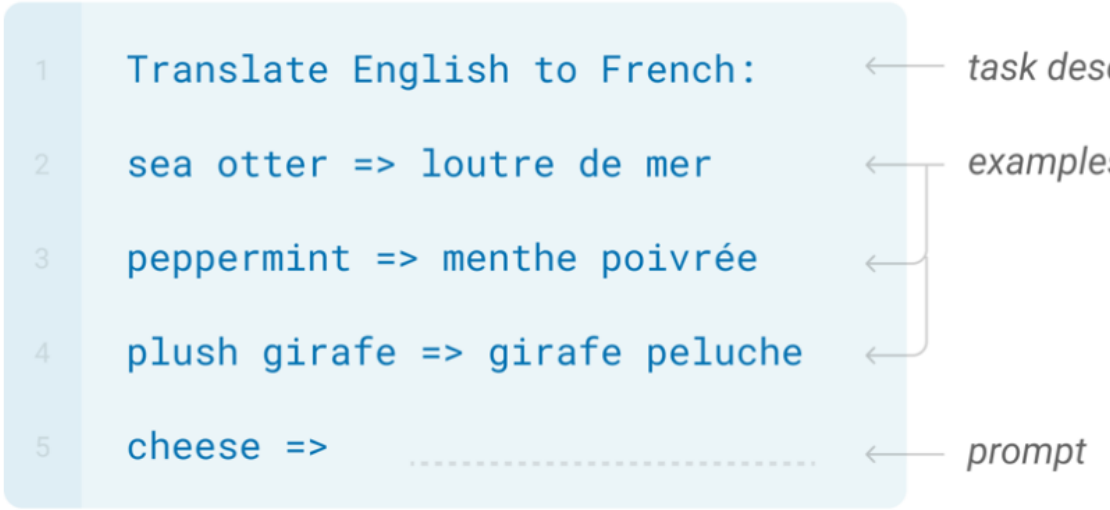
Lost in the Middle: How Language Models Use Long Contexts

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, Percy Liang

While recent language models have the ability to take long contexts as input, relatively little is known about how well they use longer context. We analyze the performance of language models on two tasks that require identifying relevant information in their input contexts: multi-document question answering and key-value retrieval. We find that performance can degrade significantly when changing the position of relevant information, indicating that current language models do not robustly make use of information in long input contexts. In particular, we observe that performance is often highest when relevant information occurs at the beginning or end of the input context, and significantly degrades when models must access relevant information in the middle of long contexts, even for explicitly long-context models. Our analysis provides a better understanding of how language models use their input context and provides new evaluation protocols for future long-context language models.



Prompt Engineer 101: Keep Positional Bias in mind.



- Example selection
- Template format
- Example ordering
- Label selection
- …

Let’s discuss labels

Label words	Accuracy
	mean (std)
great/terrible	**92.7 (0.9)**
good/bad	92.5 (1.0)
cat/dog	91.5 (1.4)
dog/cat	86.2 (5.4)
terrible/great	83.2 (6.9)

Label Examples Credit to:

Making Pre-trained Language Models Better Few-shot Learners

By

Tianyu Gao, Adam Fisch and Danqi Chen

Let’s discuss labels

```
# cat for positive, dog for negative
document = (
    "featuring an oscar-worthy performance => cat\n"
    "completely messed up => dog\n"
    "masterpiece => cat\n"
    "the action is stilted => dog\n"
    "by far the worst movie of the year =>"
)
```

	Prob of being positive	Prob of being negative
positive/negative	0.3157	0.2891
cat/dog	0.1997	0.1463
dog/cat	0.2070	0.1719

Simple Heuristics (code example 4-2)

```
document = (
    "featuring an oscar-worthy performance => positive\n"
    "completely messed up => negative\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the worst movie of the year =>"
)
```

```
[(0.31566739082336426, ' positive'),
 (0.2891405522823334, ' negative'),
 (0.02993960492312908, ' bad'),
 (0.013837959617376328, ' good'),
 (0.009425444528460503, ' very'),
 (0.008499860763549805, ' great'),
 (0.005330370739102364, ' terrible'),
 (0.004886834882199764, ' not'),
 (0.004764571785926819, ' perfect'),
 (0.004176552407443523, ' no')]
```

Open Question: Can we use digits as labels?

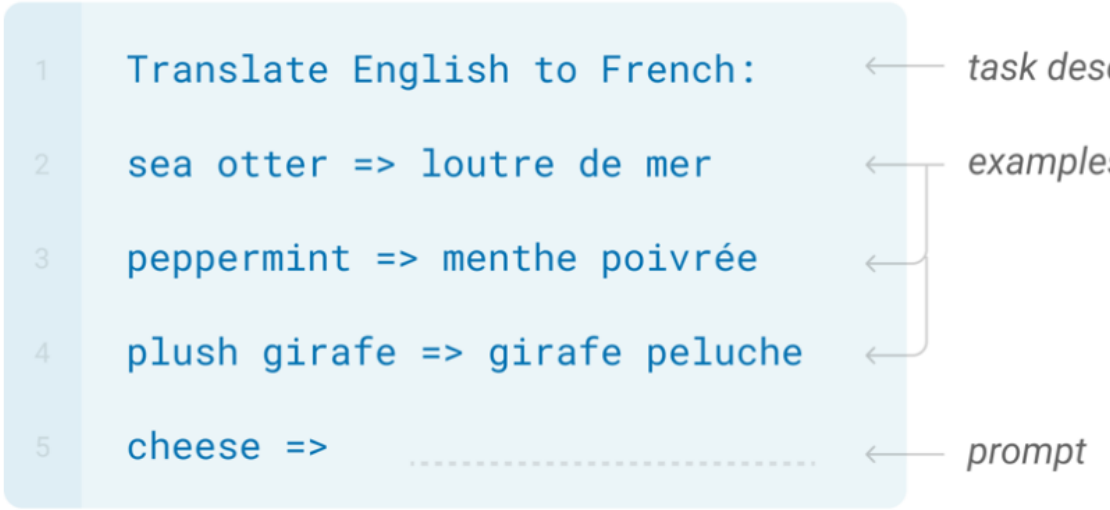
```
document = (
    "featuring an oscar-worthy performance => 0\n"
    "completely messed up => 1\n"
    "masterpiece => 0\n"
    "the action is stilted => 1\n"
    "by far the worst movie of the year =>"
)
```

```
[(0.4380252659320831, ' 1'),
 (0.340424627065658 57, ' 0'),
 (0.10940475016832352, ' 2'),
 (0.021524671465158463, ' 3'),
 (0.013400197960436344, ' 4'),
 (0.009645469486713 41, ' 5'),
 (0.004922178573906422, ' 6'),
 (0.004260277841240168, ' 8'),
 (0.003846667939797044, ' 9'),
 (0.003295806935057044, ' 7')]
```

```
document = (
    "featuring an oscar-worthy performance => 7\n"
    "completely messed up => 11\n"
    "masterpiece => 7\n"
    "the action is stilted => 11\n"
    "by far the worst movie of the year =>"
)
```

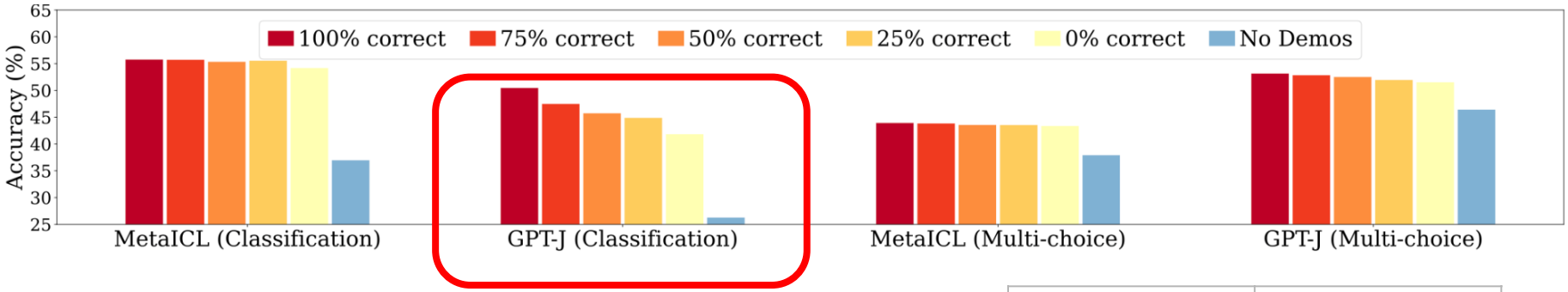
```
[(0.18294943869113922, ' 7'),
 (0.09998831897974014, ' 9'),
 (0.09698300063610077, ' 8'),
 (0.08610329777002335, ' 6'),
 (0.0774054080247879, ' 11'),
 (0.0677629262208 9386, ' 10'),
 (0.06541603803634644, ' 5'),
 (0.04681426286697388, ' 4'),
 (0.03447182849049568, ' 12'),
 (0.027355313301086426, ' 3')]
```

Prompt Engineer 101: Use semantically meaningful labels



- Example selection
- Template format
- Example ordering
- Label selection
- …

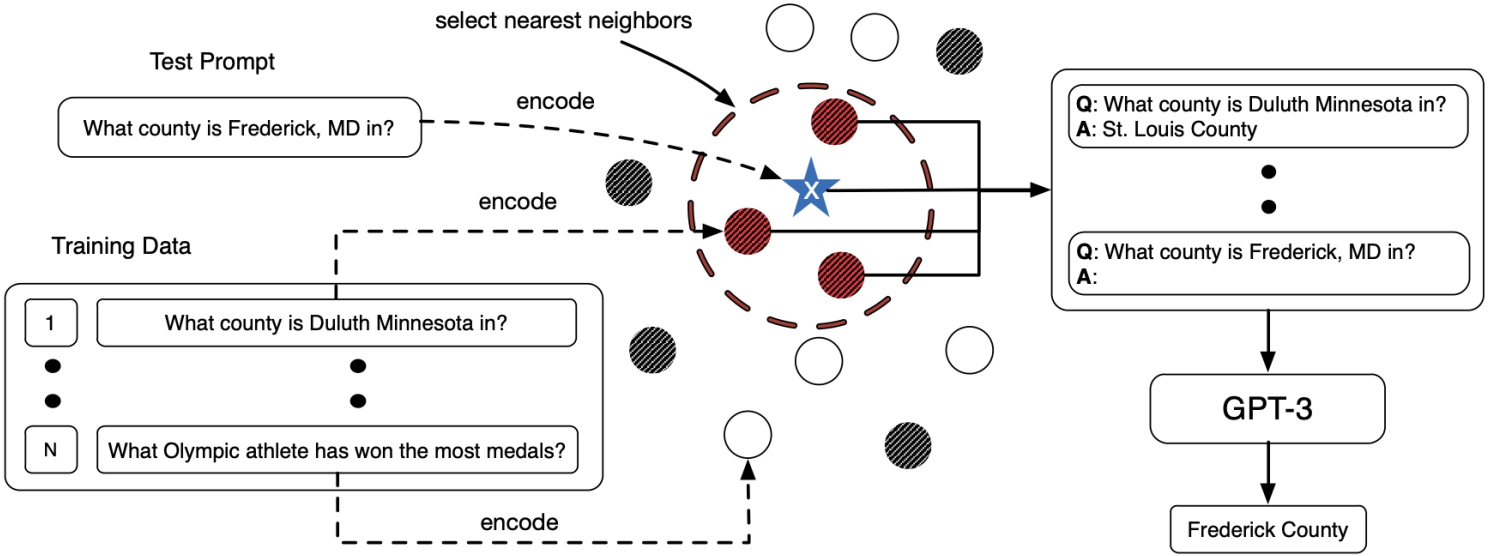
Really? (Rethinking the role of demonstrations)




```
document = (
    "featuring an oscar-worthy performance => negative\n"
    "completely messed up => positive\n"
    "masterpiece => positive\n"
    "the action is stilted => negative\n"
    "by far the worst movie of the year =>"
)
```

100% correct	31.6 positive 28.9 negative 3.0 bad 1.3 good
50% correct	38.8 positive 38.4 negative 1.6 bad 0.9 good

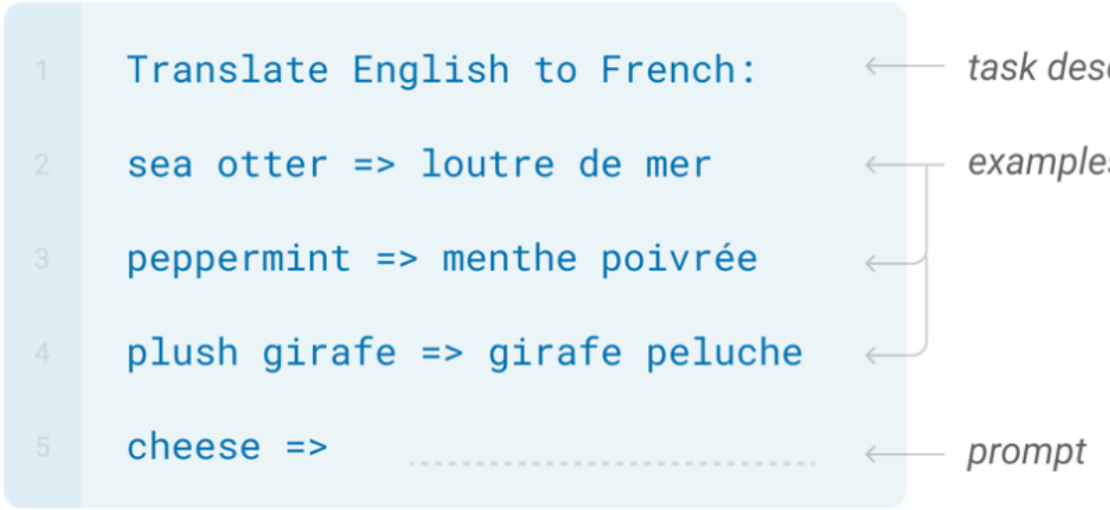
Context is important?



What Makes Good In-Context Examples for GPT-3?

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, Weizhu Chen

Prompt Engineer 101: Context is still under-explored.



- Example selection
- Template format
- Example ordering
- Label selection
- …

Now … complex tasks

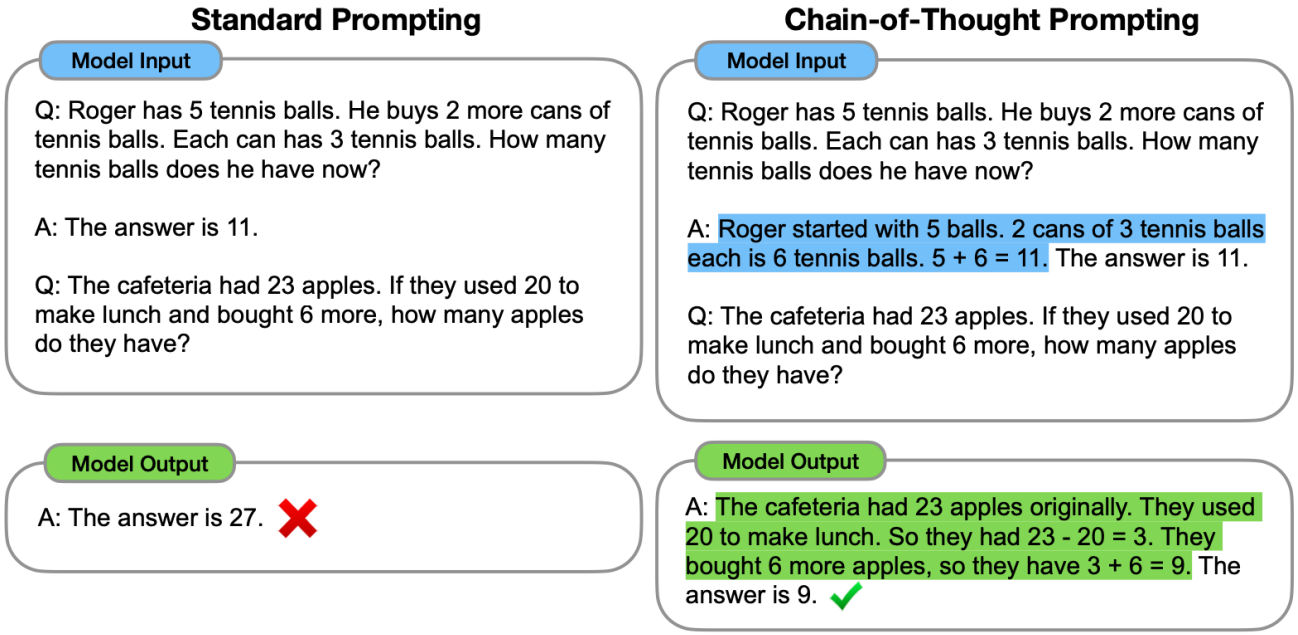


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." *Advances in Neural Information Processing Systems* 35 (2022): 24824-24837.

Coding time with GPT2…

```
document = (
    "Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls.
    "A: The answer is 11.\n"
    "Q: The cafeteria had 23 apples. If they used 20 to make lunch and
)
```

Output:

```
A: The answer is 3.

Q: Roger has a lot of
money
```

```
document = (
    "Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis ball
    "A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11.
    "Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many ap
)
```

Output:

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

Q: Roger has
```

Why? Open question for you …

```
document = (
    "Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls.
    "A: The answer is 11.\n"
    "Q: The cafeteria had 23 apples. If they used 20 to make lunch and
)
```

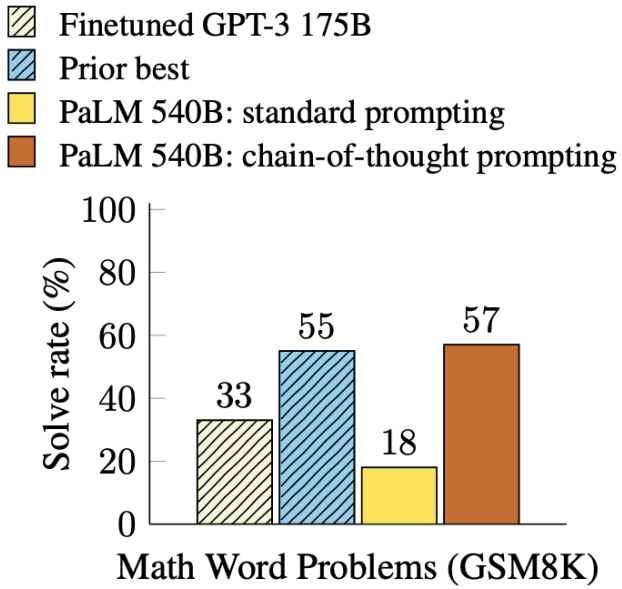
```
document = (
    "Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis bal
    "A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11.
    "Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many a
)
```

Output:

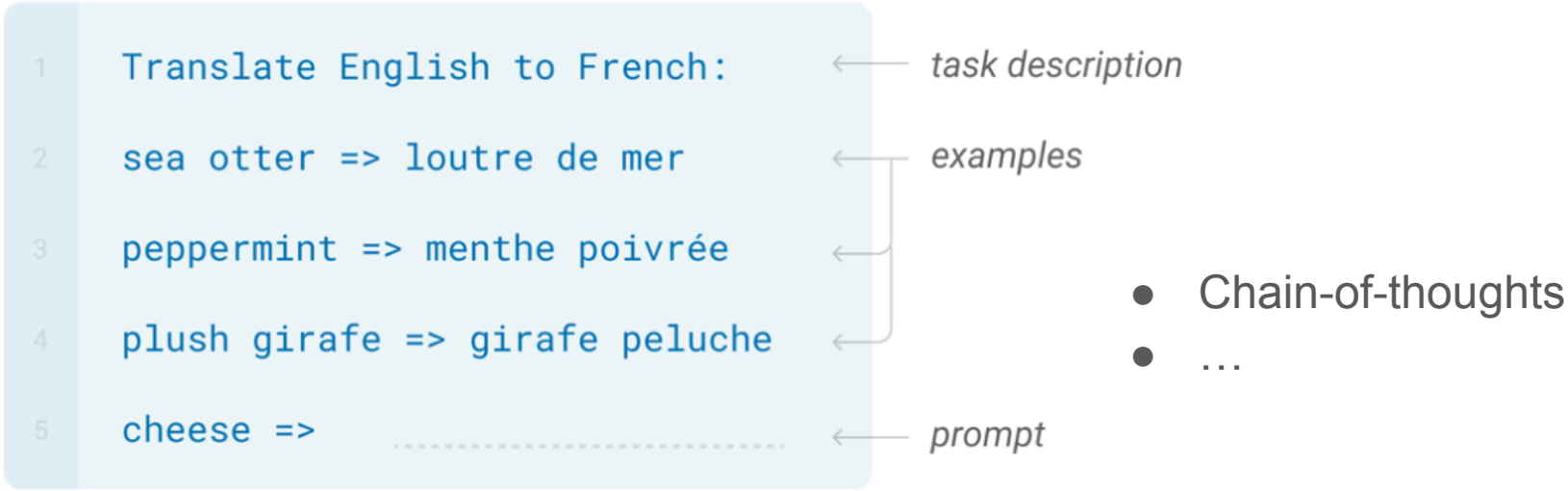
```
A: The answer is 3.

Q: Roger has a lot of
money
```

Output:

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

Q: Roger has
```



Prompt Engineer 101: No universal solution for prompting



- Chain-of-thoughts
- …

Complex Chain-of-Thought

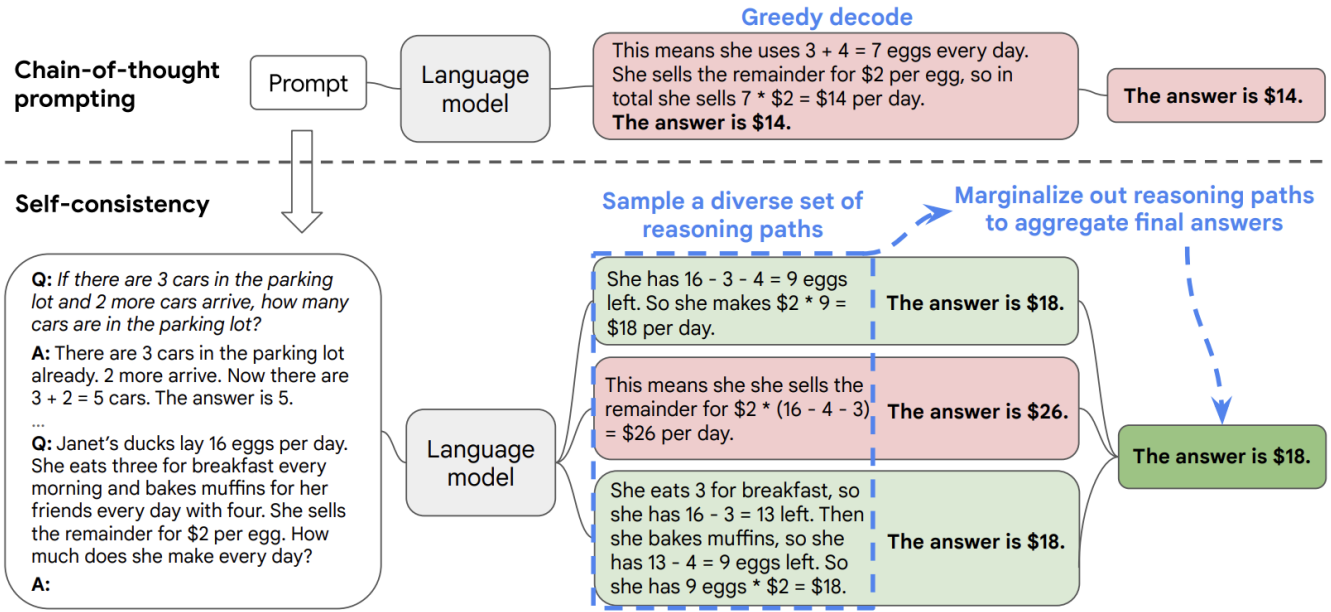


Figure 1: The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the "greedy decode" in CoT prompting by sampling from the language model's decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set.

Wang, Xuezhi, et al. "Self-Consistency Improves Chain of Thought Reasoning in Language Models." *The Eleventh International Conference on Learning Representations*. 2022.

Can we automate reasoning text writing?

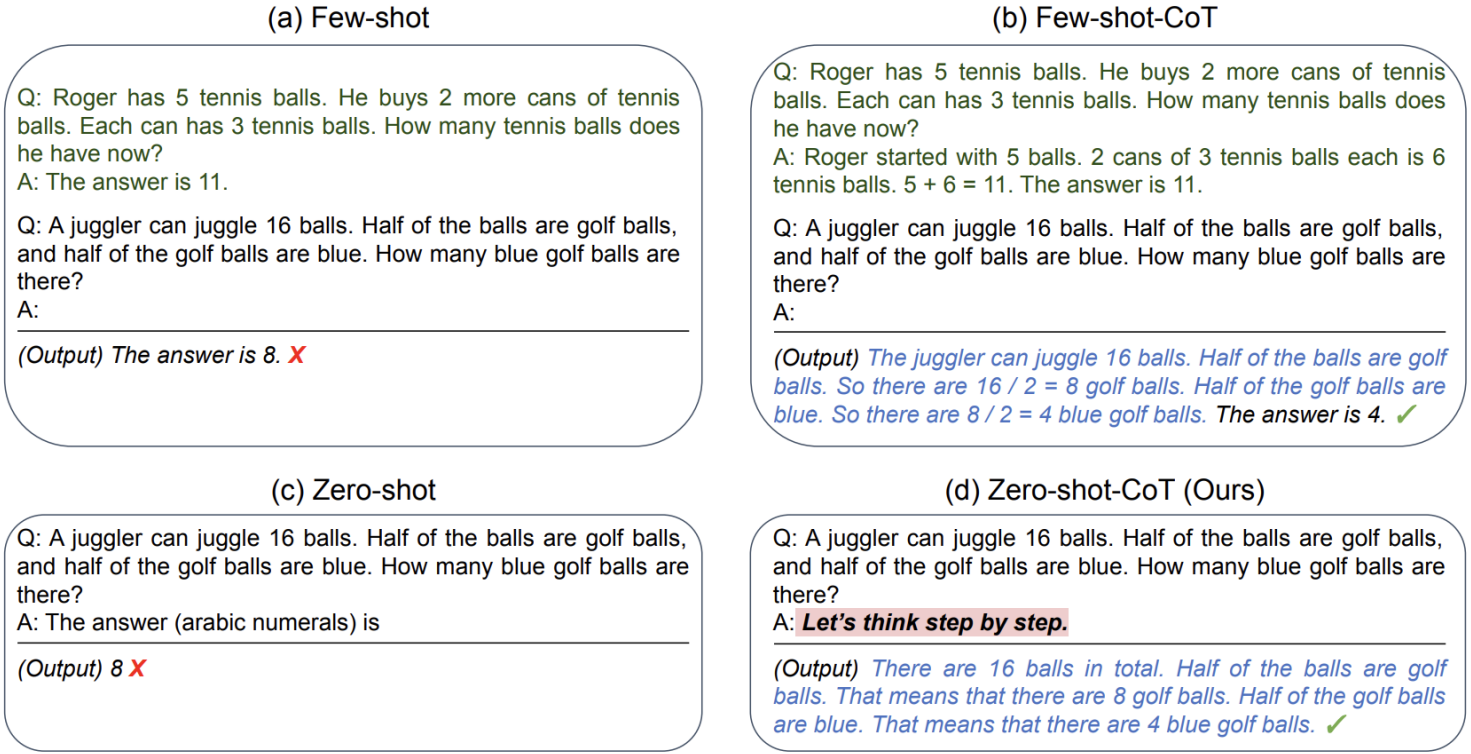


Figure 1: Example inputs and outputs of GPT-3 with (a) standard Few-shot ([Brown et al., 2020]), (b)

Kojima, Takeshi, et al. "Large language models are zero-shot reasoners." *Advances in neural information processing systems* 35 (2022): 22199-22213.

What do you think, Dr. GPT2?

```
document = (
    "Q: A juggler can juggle 16 balls.
    "A:"
)
```

Output:

```
A: The number of balls is determined by the
number of balls in the ball. The number of balls
is determined by the number of balls in the
ball. The number
```

```
document = (
    "Q: A juggler can juggle 16 balls. H
    "A: The answer (arabic numerals) is"
)
```

Output:

```
A: 16.
```


```
document = (
    "Q: A juggler can juggle 16 balls.
    "A: Let's think step by step."
)
```

Output:

```
A: The first ball is a ball that is about 1/4
inch long. The second ball is a ball that is
about 1/4 inch long. The third
```

Program-aided Language Models

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

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 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

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)

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 tennis balls.
```
tennis_balls = 5
```
2 cans of 3 tennis balls each is
```
bought_balls = 2 * 3
```
tennis balls. The answer is
```
answer = tennis_balls + bought_balls
```

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

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
```

Hey Programmer Mistral 7B

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

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 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

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)

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 tennis balls.
`tennis_balls = 5`
2 cans of 3 tennis balls each is
`bought_balls = 2 * 3`
tennis balls. The answer is
`answer = tennis_balls + bought_balls`

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

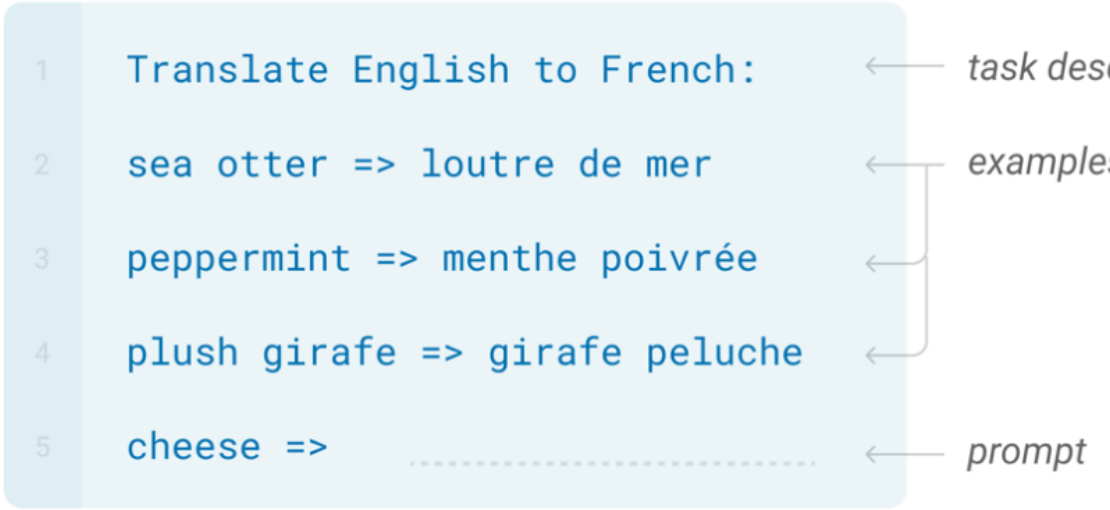
A: The bakers started with 200 loaves
`loaves_baked = 200`
They sold 93 in the morning and 39 in the afternoon
`loaves_sold_morning = 93`
`loaves_sold_afternoon = 39`
The grocery store returned 6 loaves.
`loaves_returned = 6`
The answer is
```
answer = loaves_baked - loaves_sold_morning
  - loaves_sold_afternoon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

```
>>> print(answer)
74
```

Output:

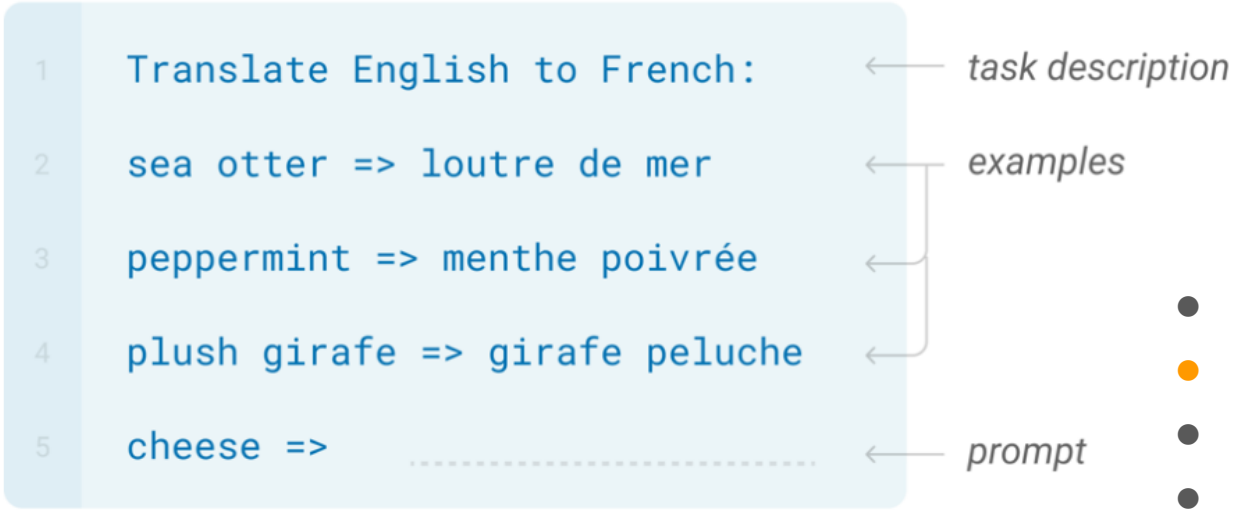
```
loaves = 200

sold_morning = 93

sold_afternoon = 39

returned = 6

answer = loaves - (sold_morning +
sold_afternoon + returned)

…
```

Prompt Engineer 101: First few tokens are important …



- Let’s think step by step
- TL;DR
- The answer (arabic numbers) is
- …

Finally … Template Design



- Example selection
- Template format
- Example ordering
- Label selection
- …

Template Design in MLM-based Models

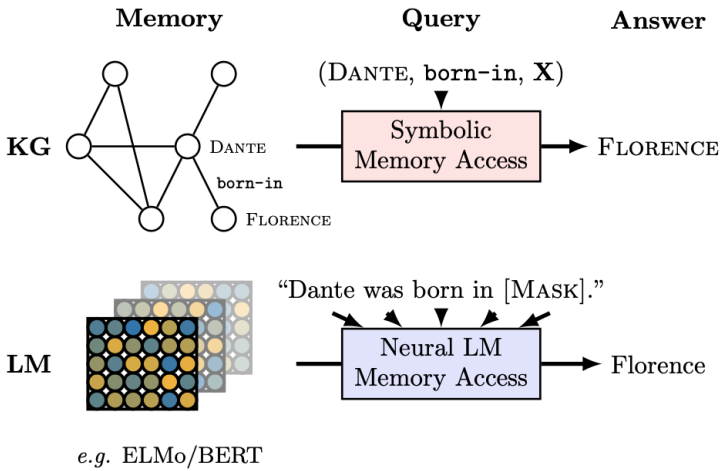


Figure 1: Querying knowledge bases (KB) and language models (LM) for factual knowledge.

Prompts	
manual	DirectX *is developed by* y_{man}
mined	y_{mine} *released the* DirectX
paraphrased	DirectX *is created by* y_{para}

Top 5 predictions and log probabilities

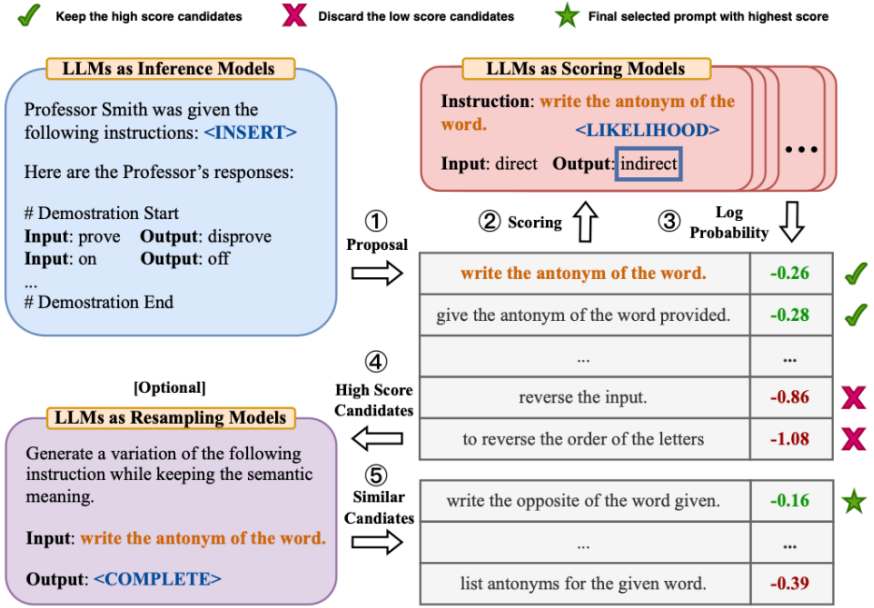
	y_{man}		y_{mine}		y_{para}	
1	Intel	-1.06	Microsoft	-1.77	Microsoft	-2.23
2	Microsoft	-2.21	They	-2.43	Intel	-2.30
3	IBM	-2.76	It	-2.80	default	-2.96
4	Google	-3.40	Sega	-3.01	Apple	-3.44
5	Nokia	-3.58	Sony	-3.19	Google	-3.45

Figure 1: Top-5 predictions and their log probabilities using different prompts (manual, mined, and paraphrased) to query BERT. Correct answer is underlined.

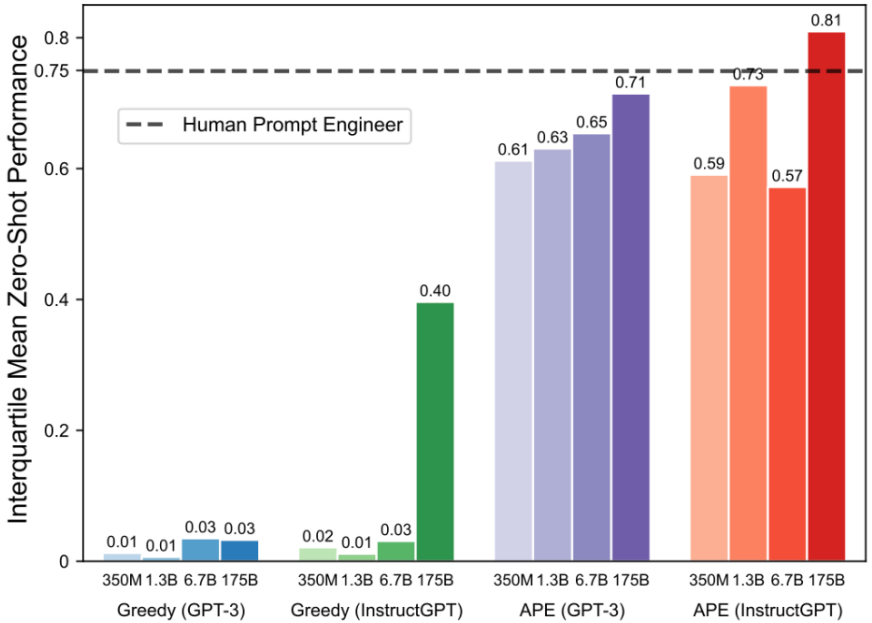
[1] Petroni, Fabio, et al. "Language models as knowledge bases?." *arXiv preprint arXiv:1909.01066* (2019).

[2] Jiang, Zhengbao, et al. "How can we know what language models know?." *Transactions of the Association for Computational Linguistics* 8 (2020): 423-438.

Common practice for automated template design



(a) Automatic Prompt Engineer (APE) workflow



(b) Interquartile mean across 24 tasks

Let's think step by step?

Table 1: Top instructions with the highest GSM8K zero-shot test accuracies from prompt optimization with different optimizer LLMs. All results use the pre-trained `PaLM 2-L` as the scorer.

Source	Instruction	Acc
Baselines		
(Kojima et al., 2022)	Let's think step by step.	71.8
(Zhou et al., 2022b)	Let's work this out in a step by step way to be sure we have the right answer.	58.8
	(empty string)	34.0
Ours		
`PaLM 2-L-IT`	Take a deep breath and work on this problem step-by-step.	**80.2**
`PaLM 2-L`	Break this down.	79.9
`gpt-3.5-turbo`	A little bit of arithmetic and a logical approach will help us quickly arrive at the solution to this problem.	78.5
`gpt-4`	Let's combine our numerical command and clear thinking to quickly and accurately decipher the answer.	74.5

ANSWER:

Let's think step by step?

Table 1: Top instructions with the highest GSM8K zero-shot test accuracies from prompt optimization with different optimizer LLMs. All results use the pre-trained `PaLM 2-L` as the scorer.

Source	Instruction	Acc
Baselines		
(Kojima et al., 2022)	Let's think step by step.	71.8
(Zhou et al., 2022b)	Let's work this out in a step by step way to be sure we have the right answer.	58.8
	(empty string)	34.0
Ours		
`PaLM 2-L-IT`	Take a deep breath and work on this problem step-by-step.	**80.2**
`PaLM 2-L`	Break this down.	79.9
`gpt-3.5-turbo`	A little bit of arithmetic and a logical approach will help us quickly arrive at the solution to this problem.	78.5
`gpt-4`	Let's combine our numerical command and clear thinking to quickly and accurately decipher the answer.	74.5

What if we do one-shot case, which one is the best?

Prompt Engineering: A human view

Original good movie -> positive

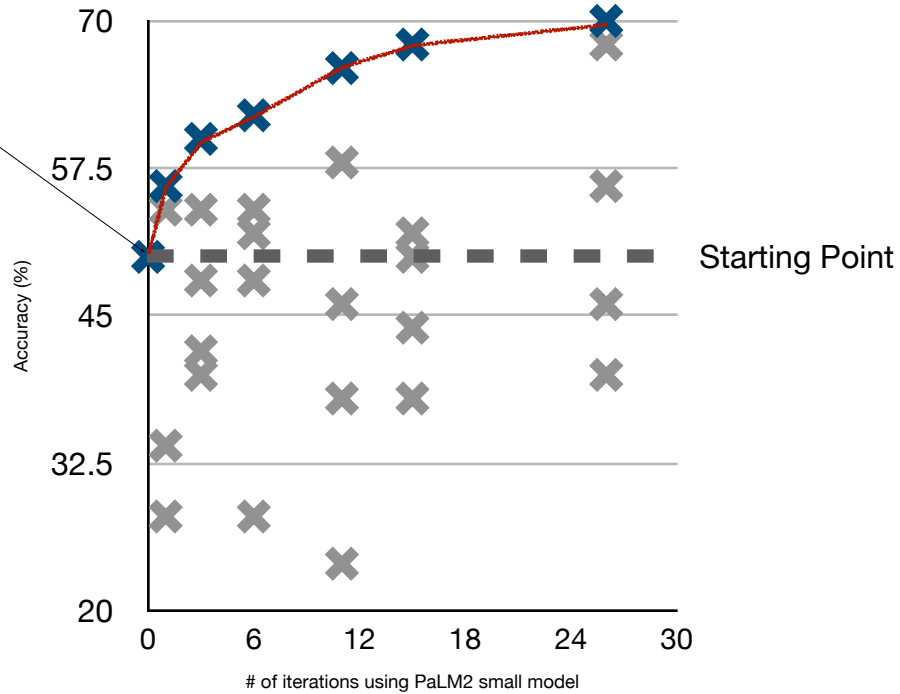
Prompt Engineering: A human view

Original	good movie -> positive
Human Rephrased	good movie Sentiment: positive
	good movie The sentiment is positive
	good movie The sentiment of the movie review is positive
	good movie It is positive
	good movie It is good

Automatic Prompt Optimisation

Prompt: BBH (object counting)

I have a car, and a toaster. How many objects do I have? Count the number of items in the list. 2



Source: Large Language Models as Optimizers. Yang et. al., 2024

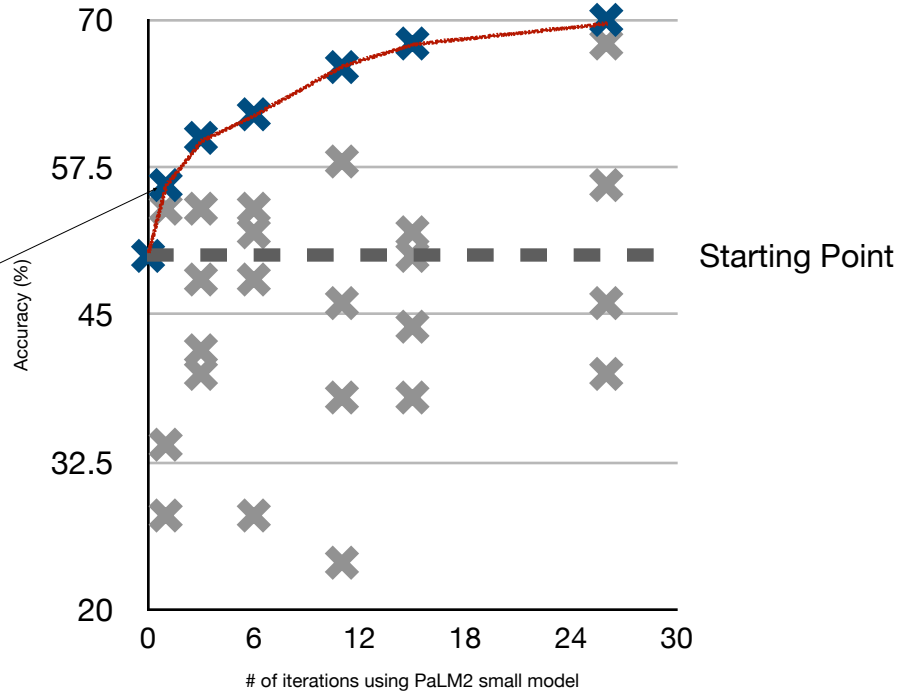
Automatic Prompt Optimisation

Prompt: BBH (object counting)

I have a car, and a toaster. How many objects do I have? Count the number of items in the list. 2

LLM Generated Prompt #1: BBH (object counting)

I have a car, and a toaster. How many objects do I have? How many items are there in this list? 2



Source: Large Language Models as Optimizers. Yang et. al., 2024

Automatic Prompt Optimisation

Prompt: BBH (object counting)

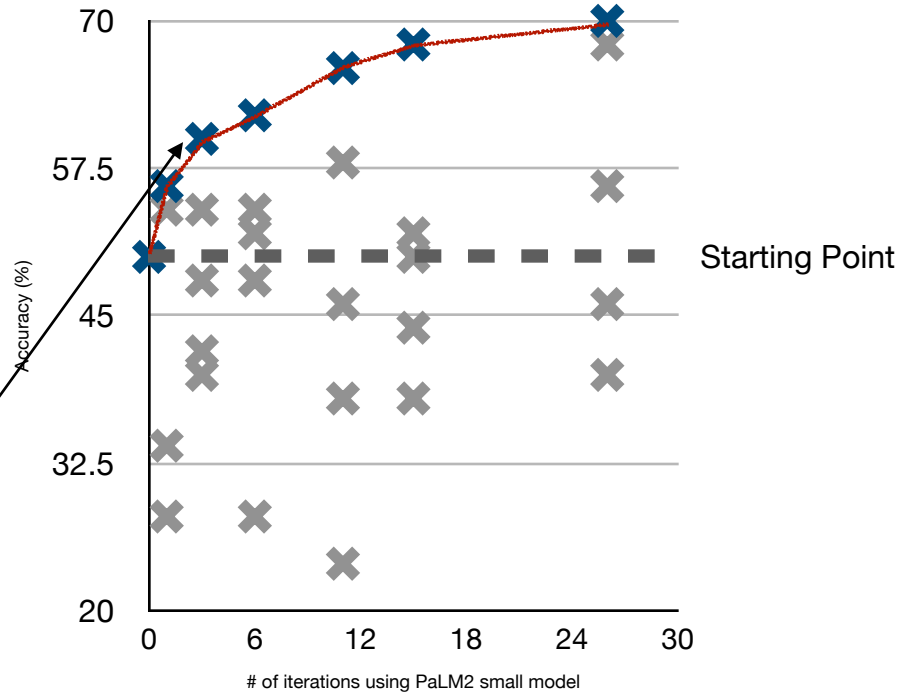
I have a car, and a toaster. How many objects do I have? Count the number of items in the list. 2

LLM Generated Prompt #1: BBH (object counting)

I have a car, and a toaster. How many objects do I have? How many items are there in this list? 2

LLM Generated Prompt #2: BBH (object counting)

I have a car, and a toaster. How many objects do I have? Please list all the items that you want me to count, separated by commas. I will count them and tell you how many items there are. 2



Source: Large Language Models as Optimizers. Yang et. al., 2024

Automatic Prompt Optimisation

Prompt: BBH (object counting)

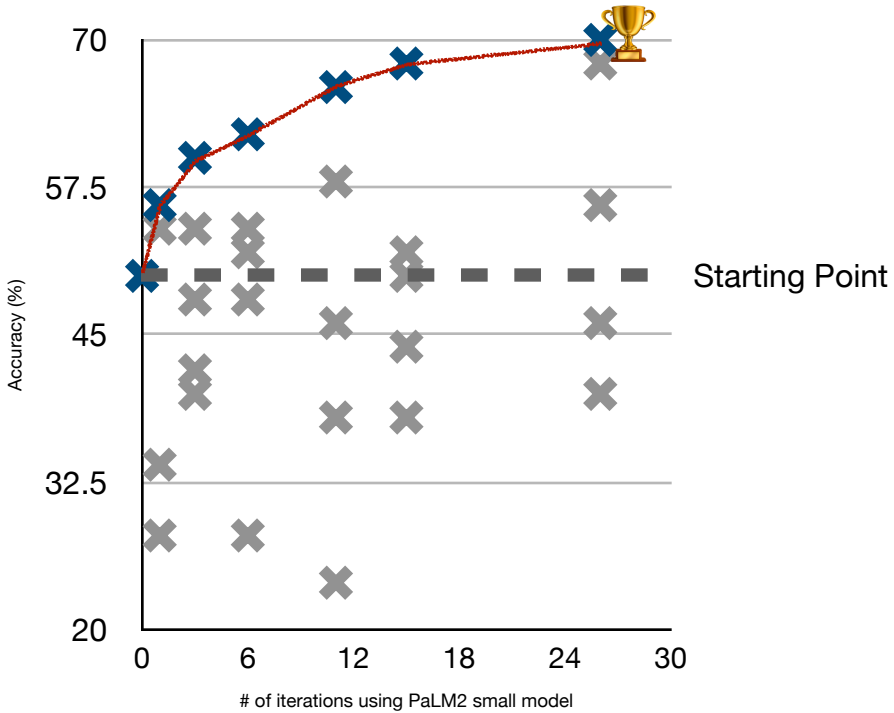
I have a car, and a toaster. How many objects do I have? Count the number of items in the list. 2

LLM Generated Prompt #1: BBH (object counting)

I have a car, and a toaster. How many objects do I have? How many items are there in this list? 2

LLM Generated Prompt #2: BBH (object counting)

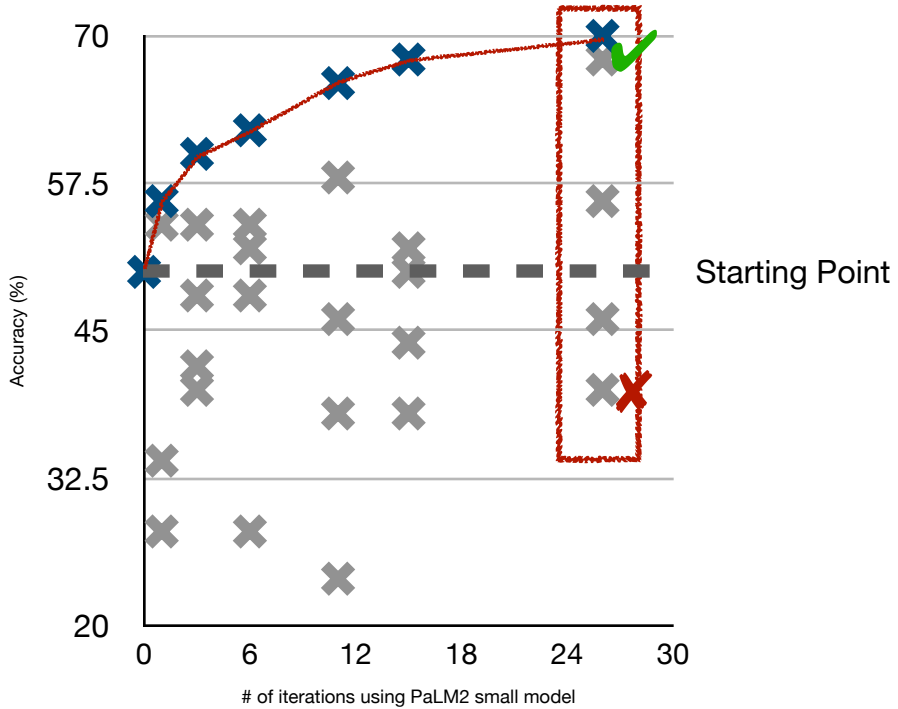
I have a car, and a toaster. How many objects do I have? Please list all the items that you want me to count, separated by commas. I will count them and tell you how many items there are. 2



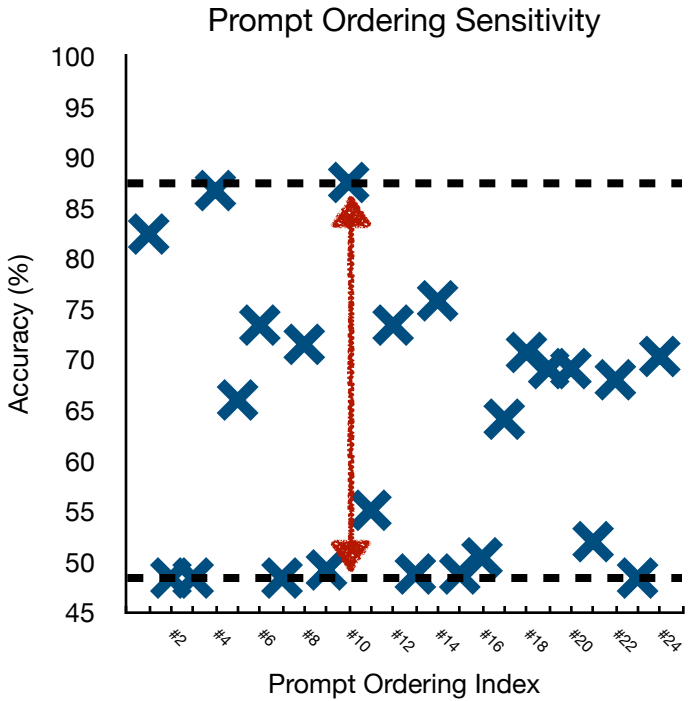
Source: Large Language Models as Optimizers. Yang et. al., 2024

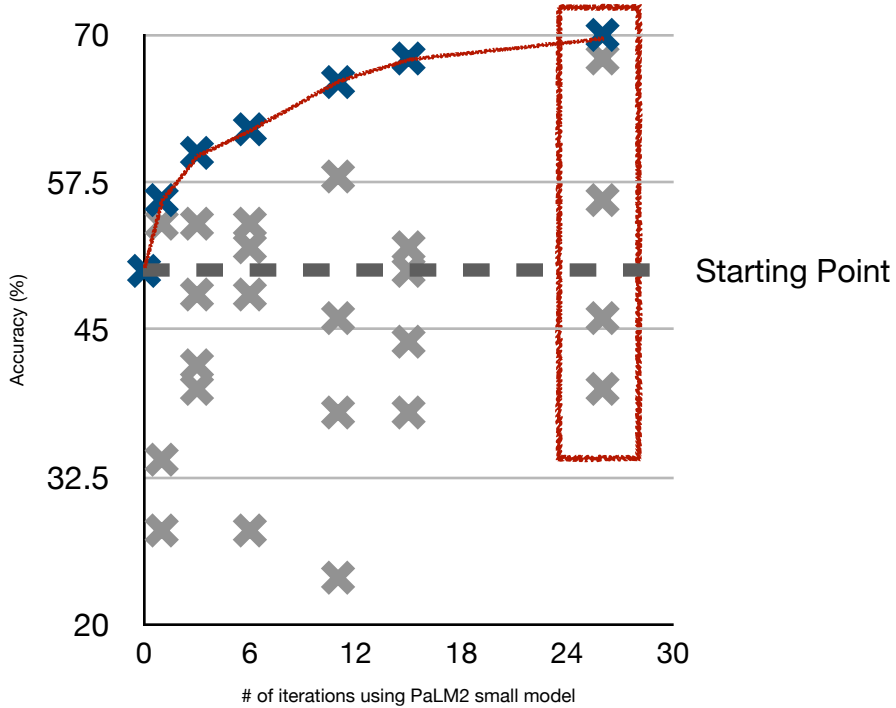
The shadow behind best prompt

- Please list the items you have, one by one, separated by commas. I will count them and tell you how many there are. ✔
- What are you trying to count? Please list all the items one by one, separated by commas, and I will tell you how many items there are in each category. ✘



Looks familiar?





Curse of LM sensitivity

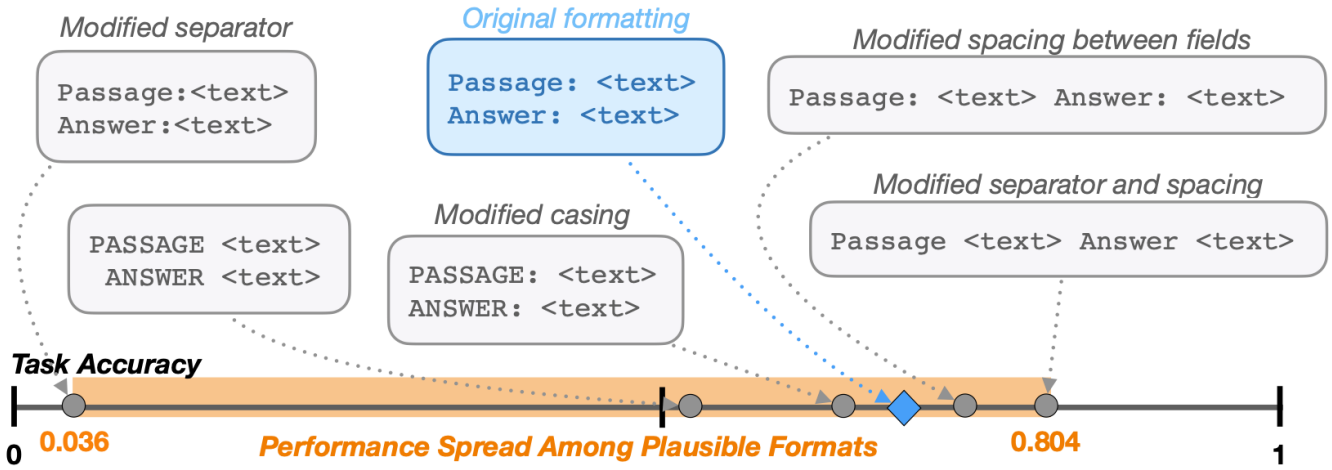


Figure 1: Slight modifications in prompt format templating may lead to significantly different model performance for a given task. Each <text> represents a different variable-length placeholder to be replaced with actual data samples. Example shown corresponds to 1-shot LLaMA-2-7B performances for task280 from SuperNaturalInstructions (Wang et al., 2022). This StereoSet-inspired task (Nadeem et al., 2021) requires the model to, given a short passage, classify it into one of four types of stereotype or anti-stereotype (gender, profession, race, and religion).

Sclar, Melanie, et al. "Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting." *arXiv preprint arXiv:2310.11324* (2023).

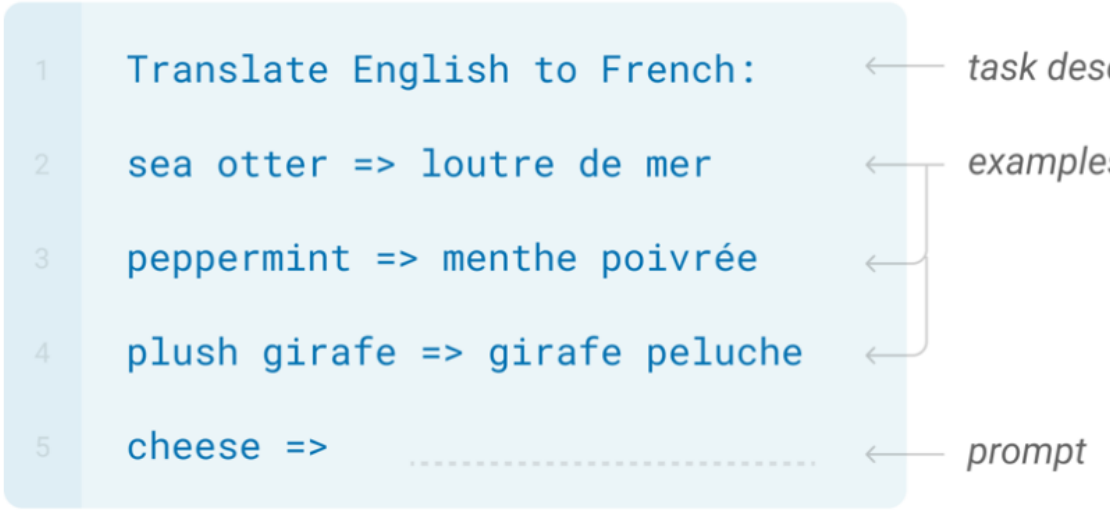
Curse of LM sensitivity - replace ANSWER with !@#$%

Strategy	Seperator	Score
OPRO	The new text is the following:	92.2
OPRO-ICL	00:57	92.2
Random Vocabulary	obliged\u0442\u0438\u0435Circ song	92.2
Random w/o Context	Home Business New \u2018	92.2
Random with Context	**GW - The Wall Street	92.2

Table 6: Performant separators discovered in the training process on AGNews using LLAMA2 7B, we report the accuracy score over the training set.

Lu, Yao, et al. "Prompt Optimisation with Random Sampling." *arXiv preprint arXiv:2311.09569* (2023).

Prompt Engineer 101: Always keep LM sensitivity in mind



- Example selection
- Template format
- Example ordering
- Label selection
- …

Connection with Instructional Models

They have similar fundamental flaws as GPT-2.

Issues like prompt template sensitivity, order issue, and positional bias issue still persist in all widely-used chat models.

We still have very limited knowledge about why and how to resolve these issues other than adding more data to tune it.

Thank you!