CS11-711 Advanced NLP

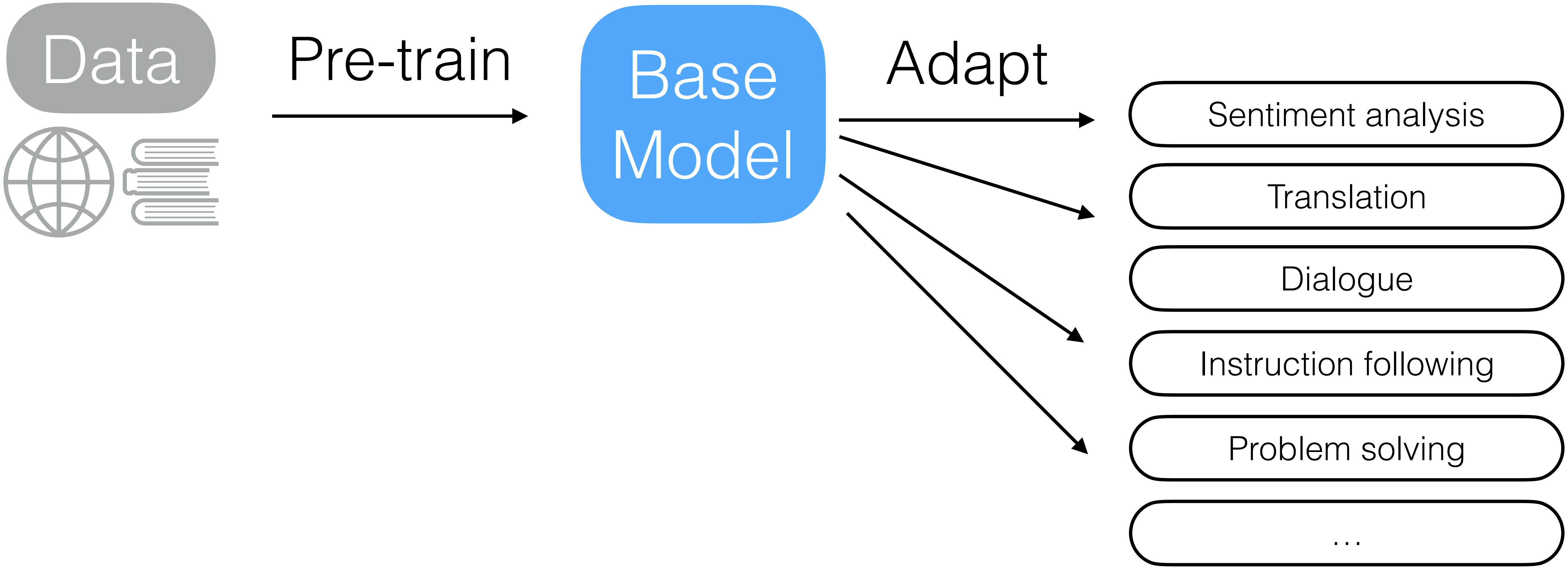
Scaling Laws and In-Context Learning

Sean Welleck

https://cmu-l3.github.io/anlp-spring2026/

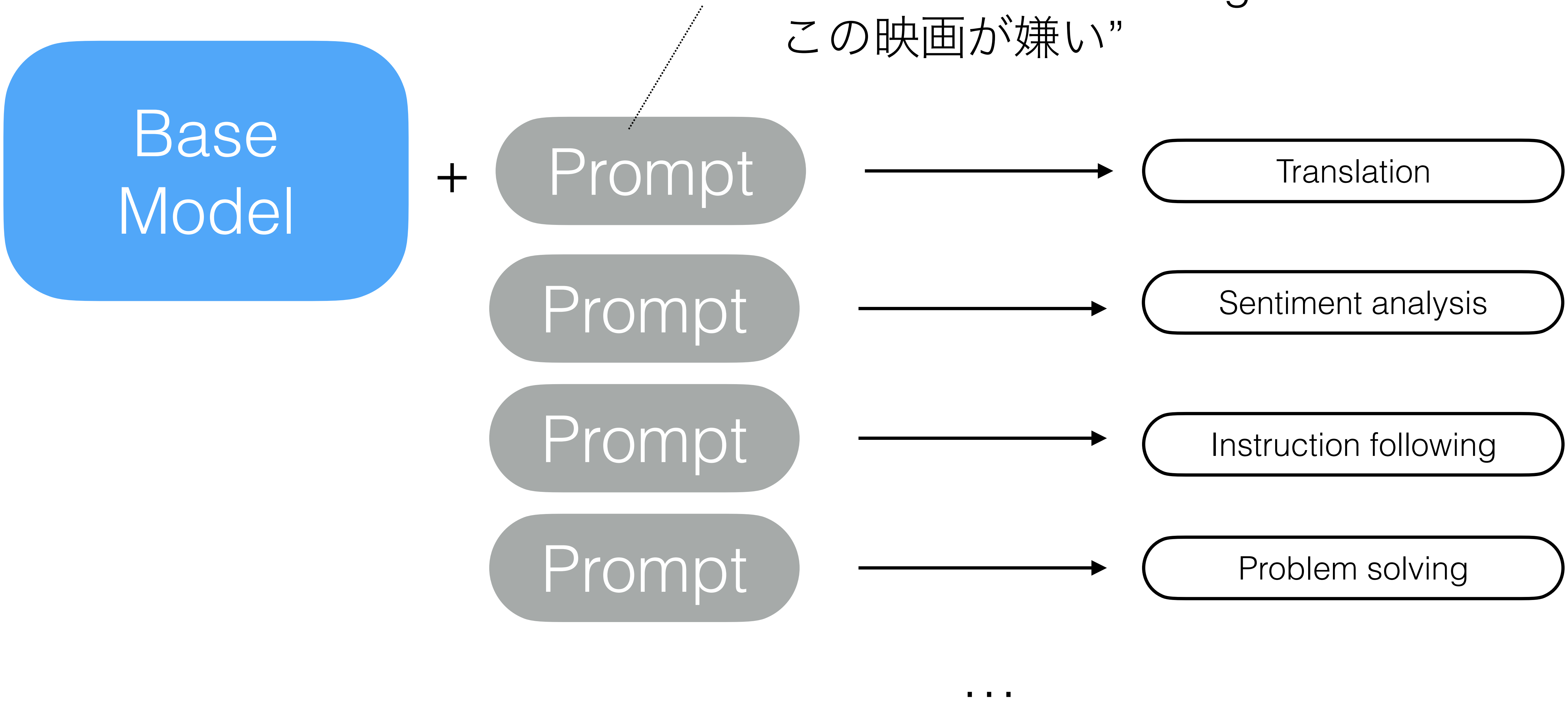
https://github.com/cmu-l3/anlp-spring2026-code

Recap: pre-training



Recap: adaptation by prompting

Example:
“Translate this sentence into English:
この映画が嫌い”



Base Model

+

Prompt → Translation

Prompt → Sentiment analysis

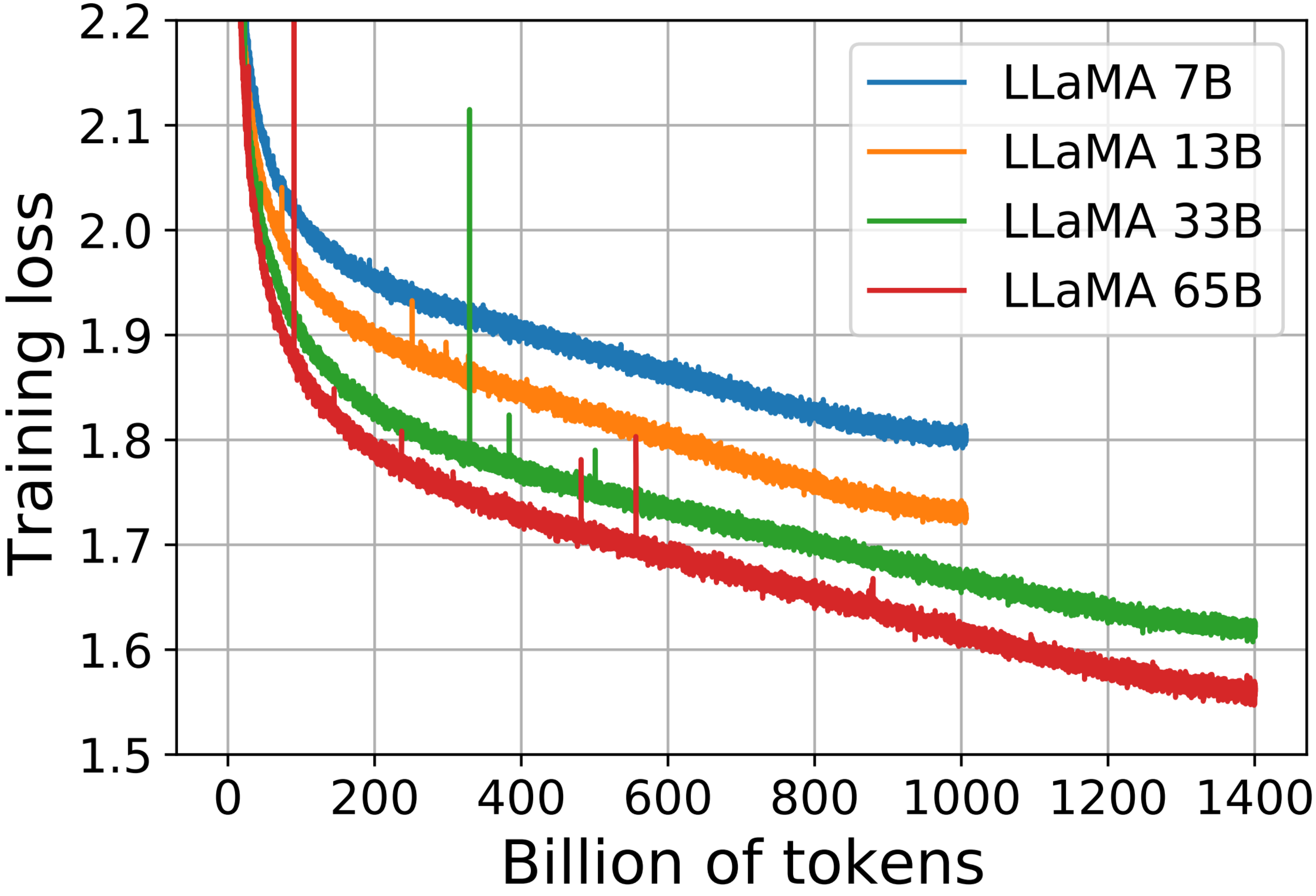
Prompt → Instruction following

Prompt → Problem solving

. . .

Today’s lecture

- Thinking about pretraining: scaling laws
- Prompting basics



Bigger model

Better Loss

More data

Challenge

- Given a fixed pretraining budget, how should we allocate the budget to achieve the best possible model?
  - Add more training data?
  - Make the model bigger?
  - A combination of both?
  - Other variables: architecture, hyperparameters, …
- **First**: let’s be a bit more precise

Training = spending compute

- We spend **compute** by performing forward and backward passes on training sequences

- An approximation for transformer language models:

$$C \approx 6ND$$

N: number of model parameters
D: number of tokens
C: compute; floating point operations (FLOPs)

Approximation from [Kaplan et al 2020]

Training = spending compute

- We spend **compute** by performing forward and backward passes on training sequences

- For example, Llama 2:

$$C \approx 6 \times 7 \text{ billion} \times 2 \text{ trillion}$$

$$= 8.4 \times 10^{22}\text{FLOPs}$$

N: number of model parameters
D: number of tokens
C: compute; floating point operations (FLOPs)

Training = spending compute

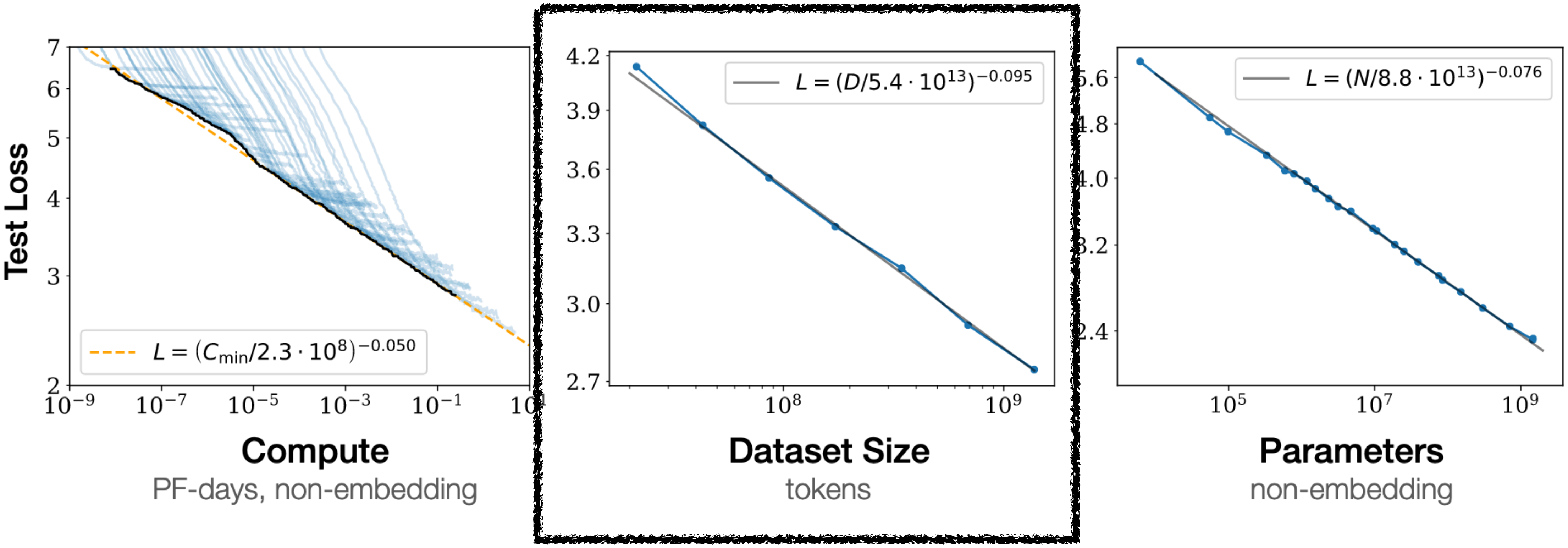
- **Want to know**: How does performance (e.g., loss) vary as we modulate these variables:
    - **Data scaling:** change *D*, what happens to loss?
    - **Model scaling**: change *N*, what happens to loss?
    - **Compute scaling**: should we increase compute by increasing *D* or increasing *N?*
    - *…*

Scaling laws

- Simple, predictive rules for model performance

- Helps us answer the questions on the previous slide

    - Run experiments with many configurations, identify trend



Power laws and log-log plots

- “Power law”:

$$y = a \cdot x^b$$

- Take logs:

$$\log y = \log(a) + b \cdot \log(x)$$

- Linear on a log-log plot

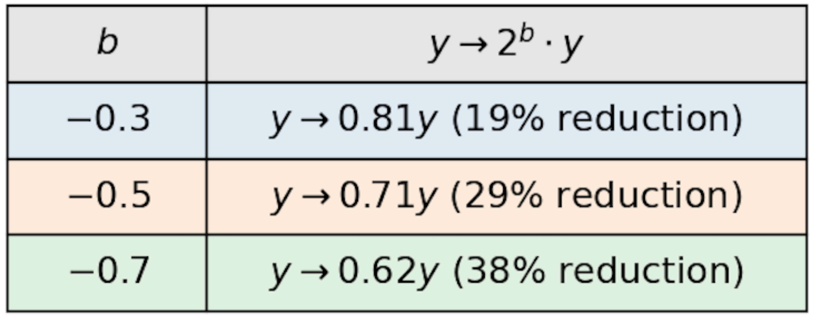
- Suppose we double x:

$$y = ax^b$$

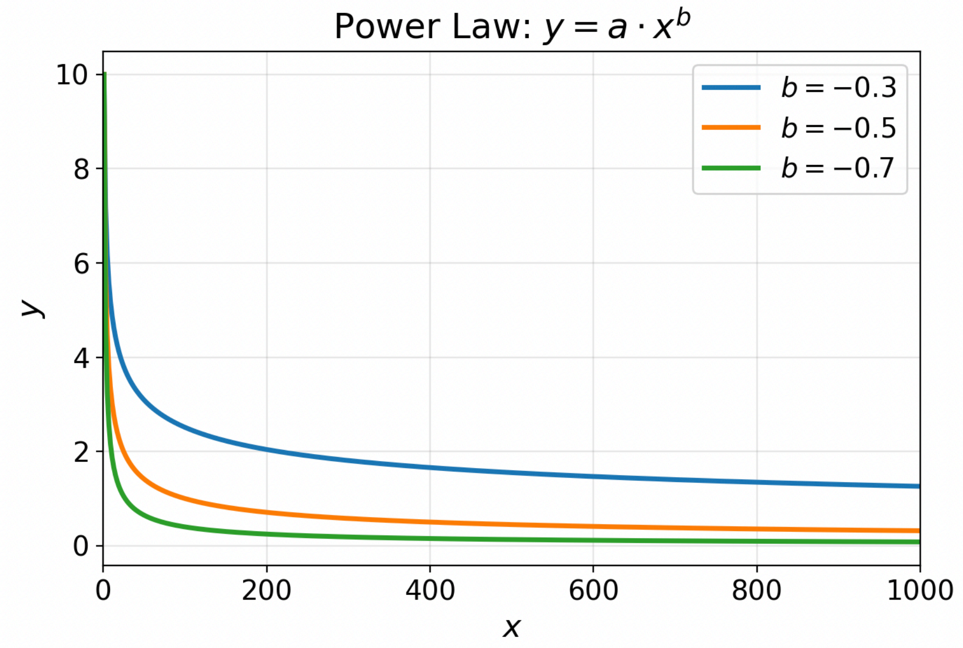
$$y_{\text{new}} = a(2x)^b$$

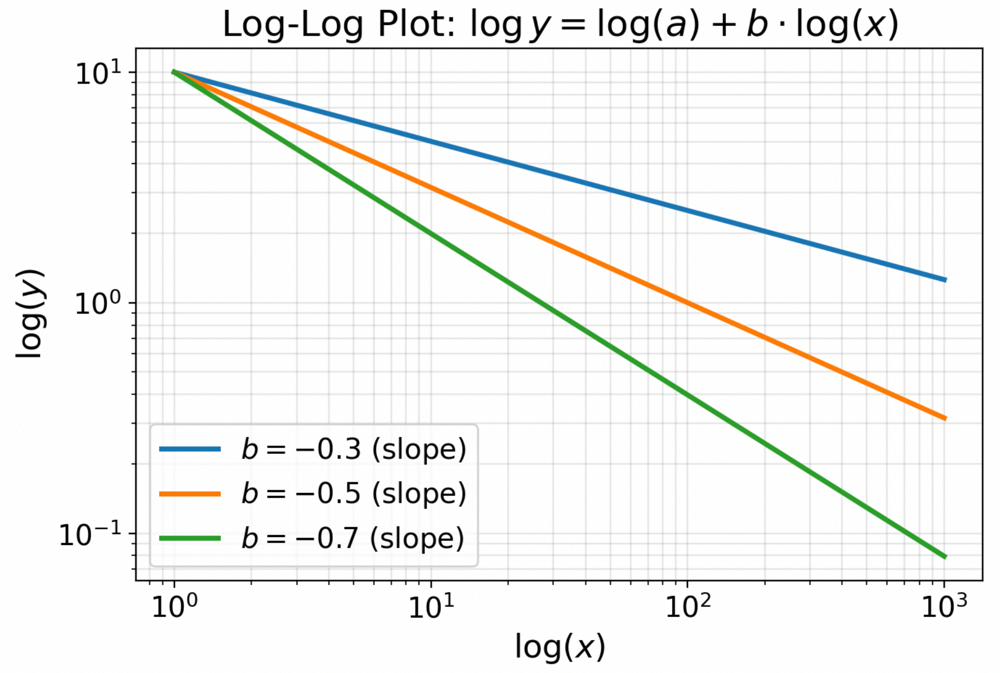
$$= ax^b 2^b$$

$$=> y_{\text{new}} = y2^b$$



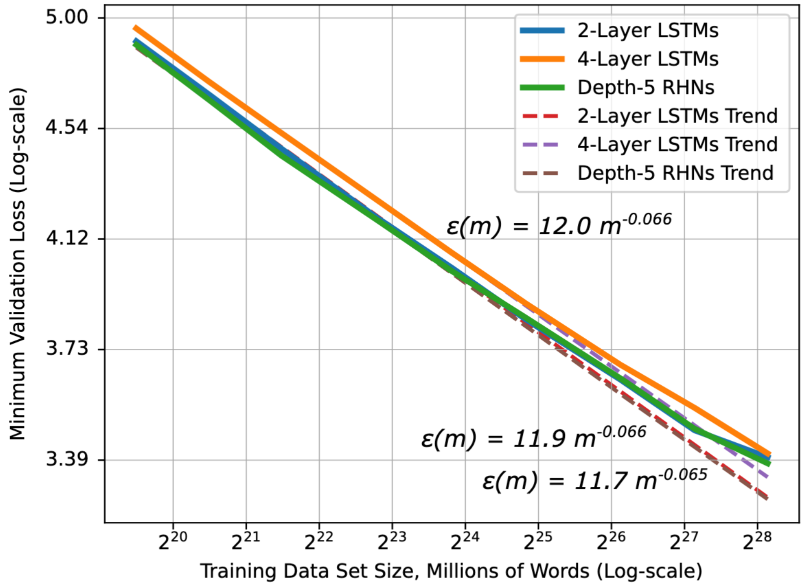
b	$y \to 2^b \cdot y$
-0.3	$y \to 0.81y$ (19% reduction)
-0.5	$y \to 0.71y$ (29% reduction)
-0.7	$y \to 0.62y$ (38% reduction)





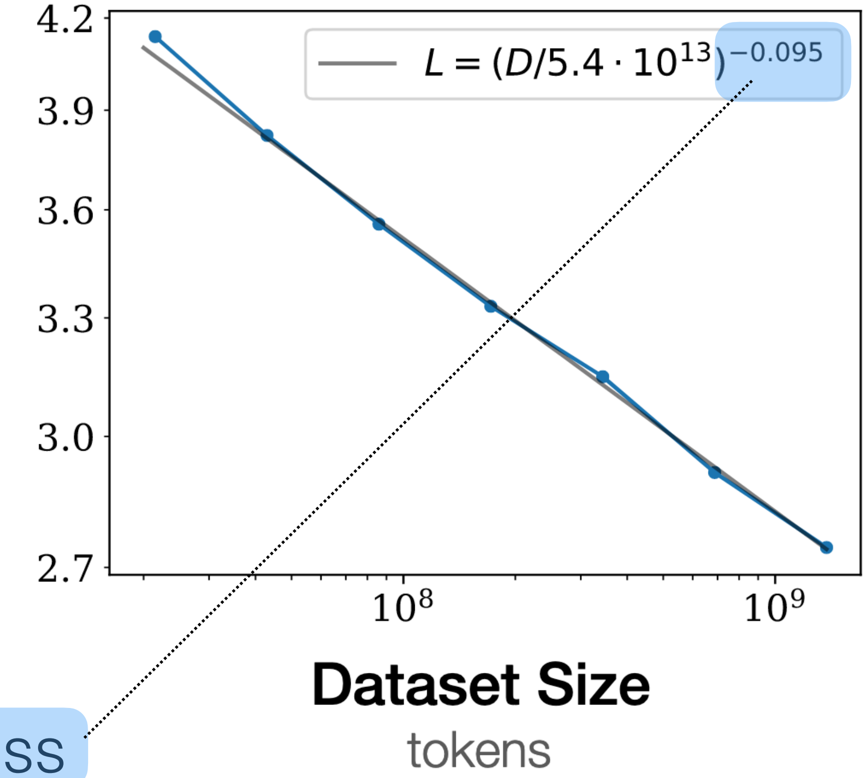
Data scaling laws

- Language models with cross-entropy loss widely observed to follow a power law in data size
- $$L = \left(\frac{D}{D_c}\right)^{-b}$$



[Hestness+ 2017]

- Building the plot:
  - Train models with varied dataset size D
  - Ensure model size is not bottlenecked: use a large model N

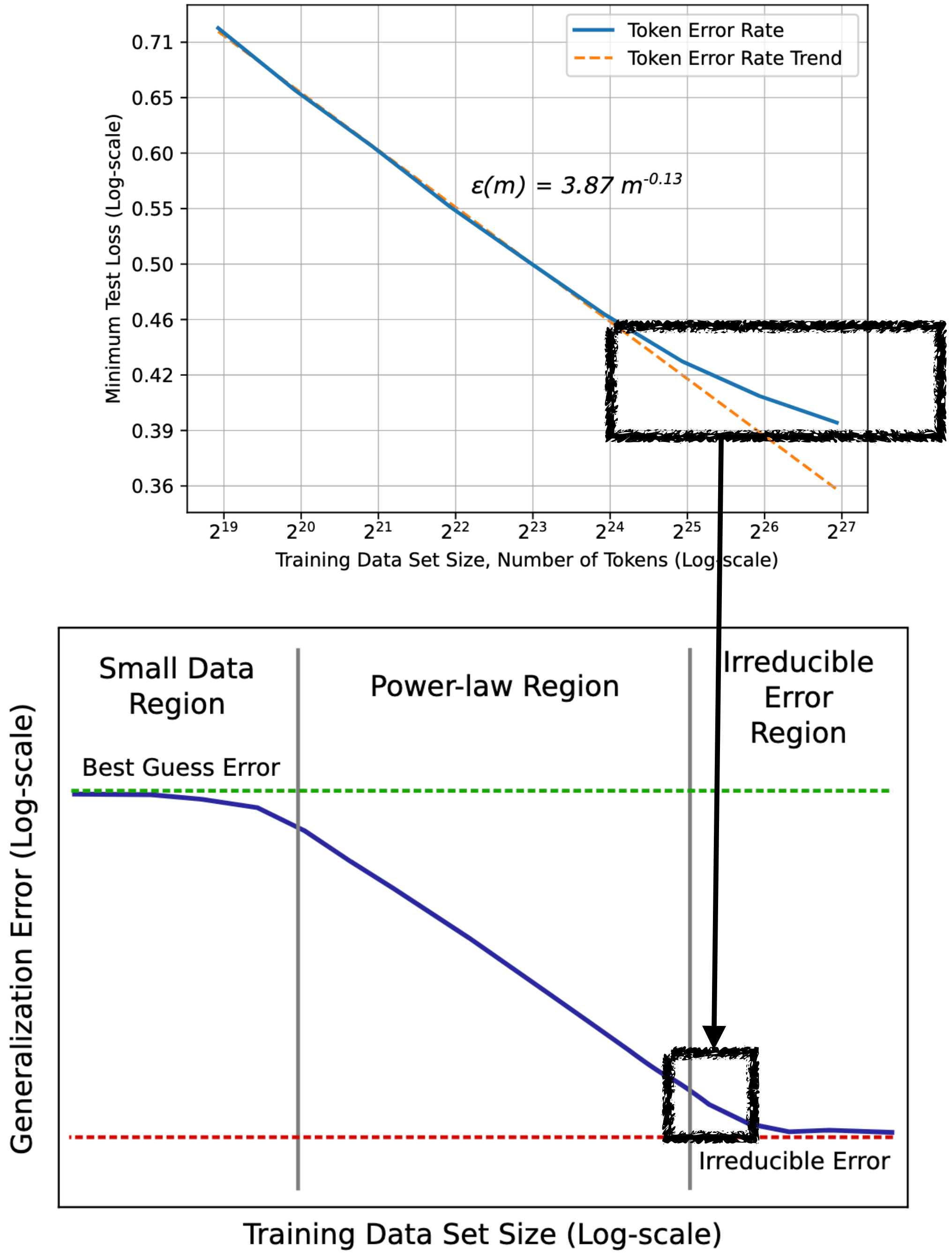


[Kaplan+ 2020]

Doubling data: $2^{-0.095} \approx 0.936$ => 6.4% decrease in loss

Data scaling laws

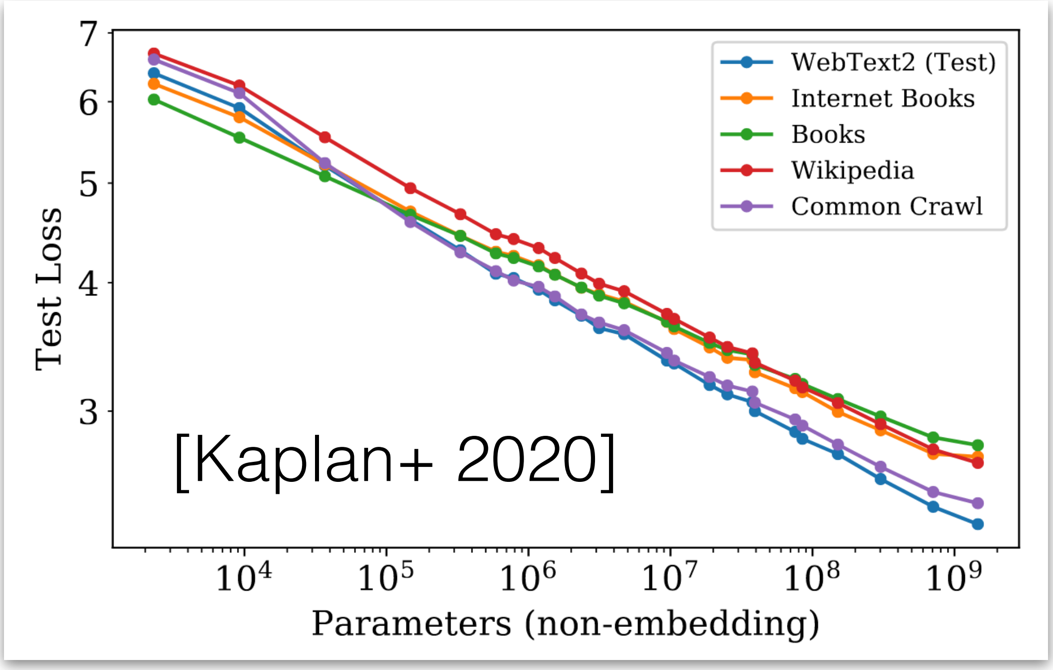
- Recall that cross-entropy loss is lower-bounded by the entropy (last lecture)
  - There is a power-law region
  - Then the power law must eventually level off



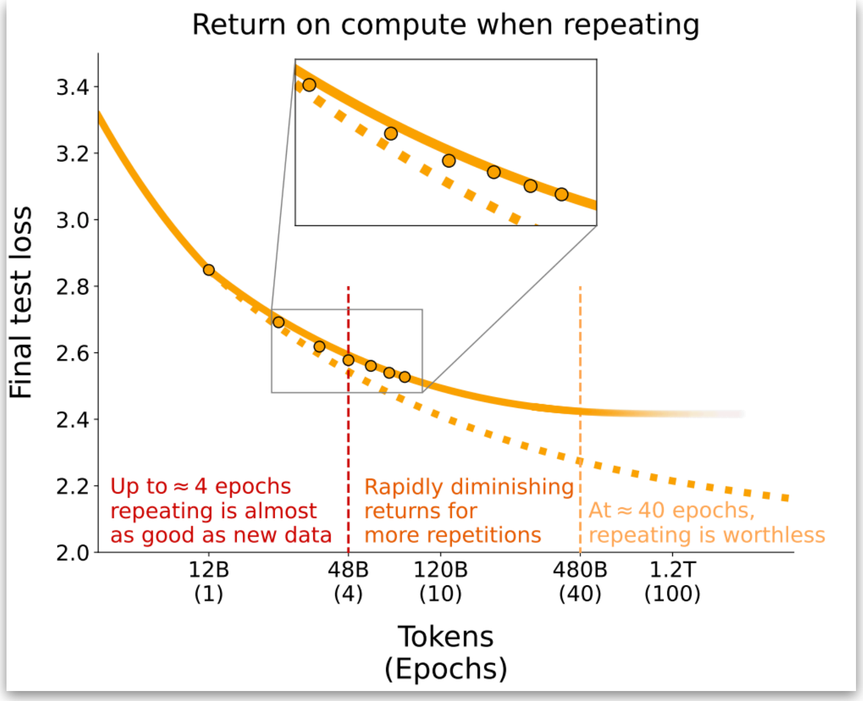
[Hestness+ 2017]

Data scaling laws

- Example uses:
  - Data composition
  - Effect of repeated data
  - …



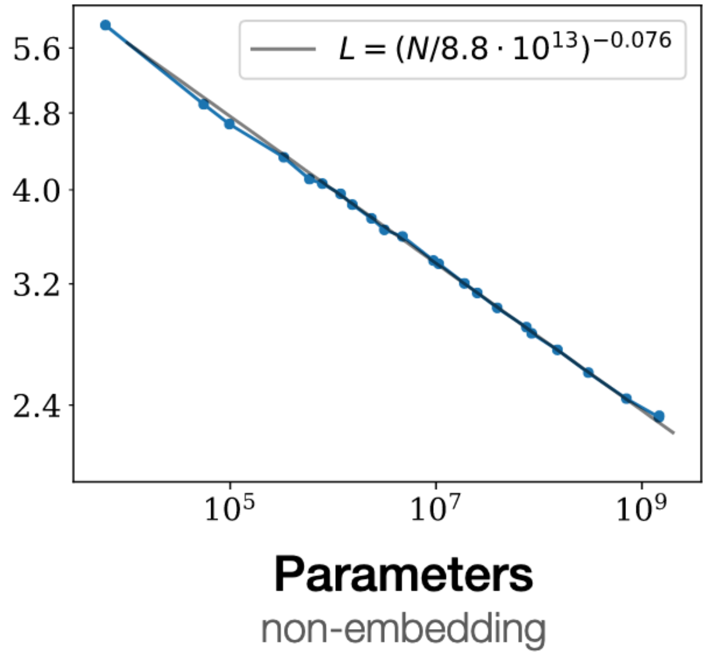
[Kaplan+ 2020]

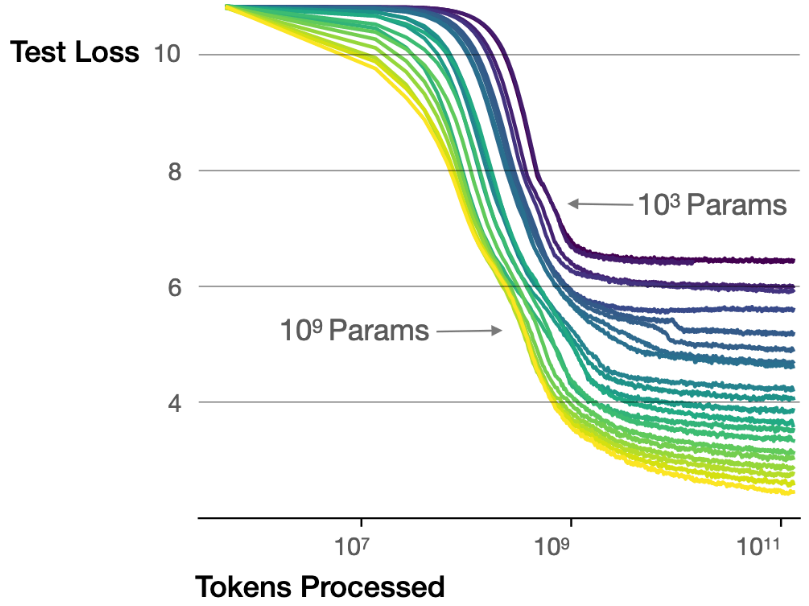


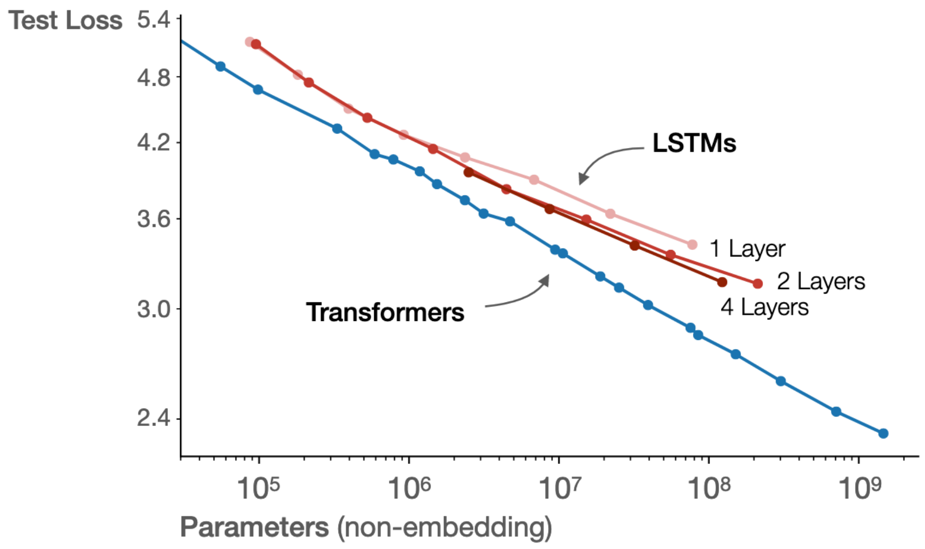
[Muenninghoff + 2025]

Model size scaling laws

- Similar relationships observed for model size
- Large models require fewer samples
- Useful for determining model hyperparameters



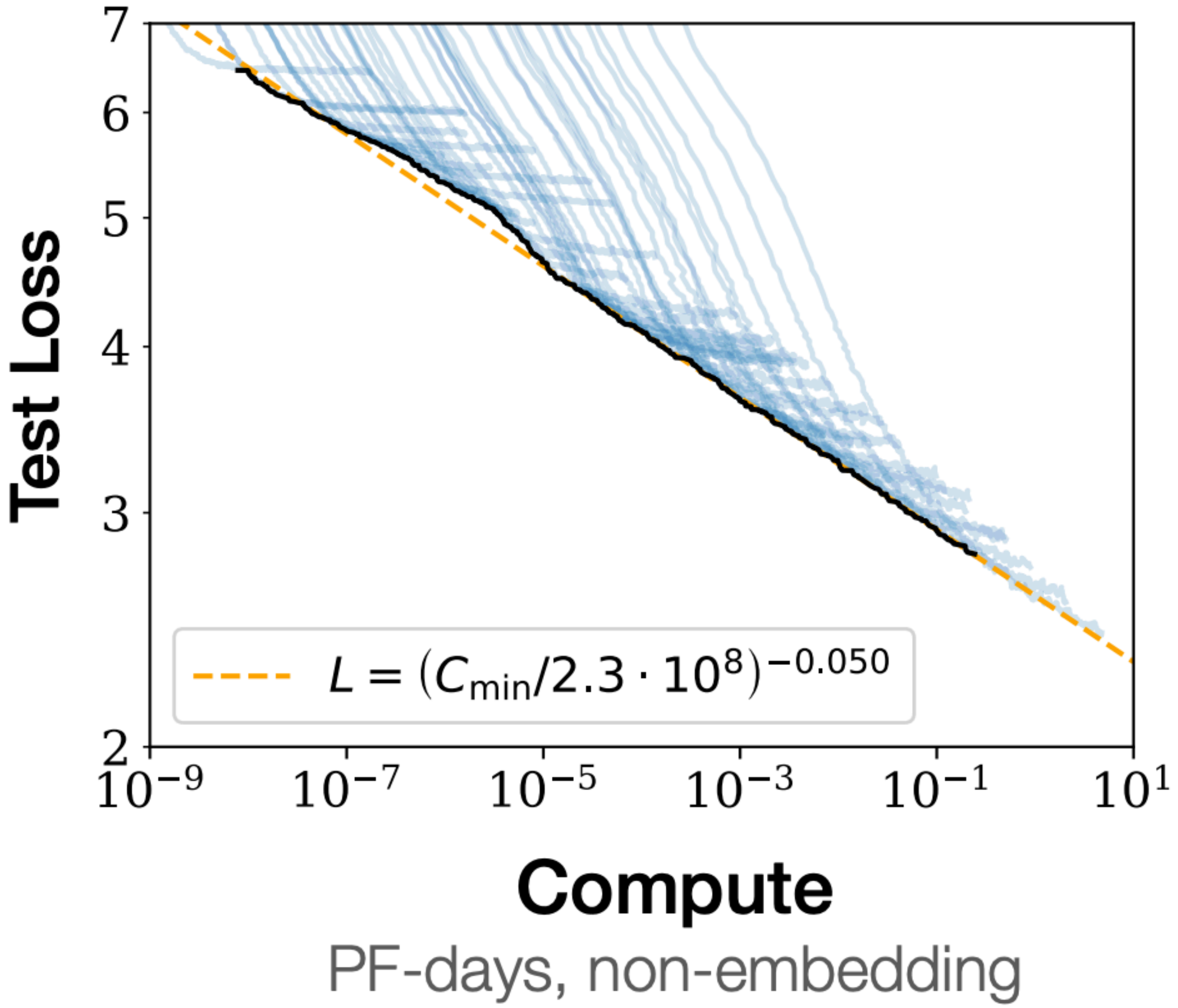




[Kaplan+ 2020]

Compute scaling laws

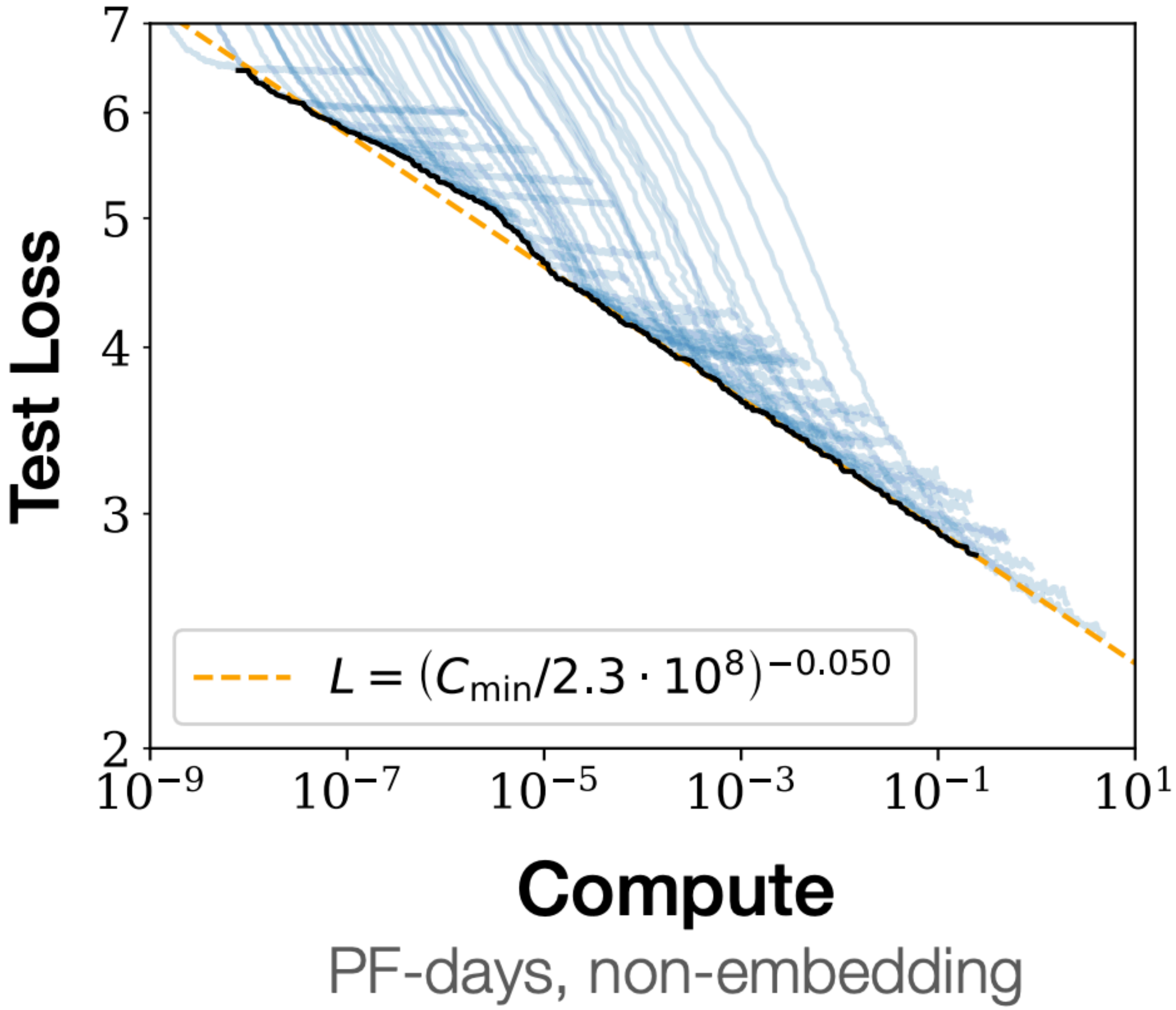
- **Basic idea**:
  - Train models of different sizes and numbers of tokens
  - Plot loss at each step of training [light blue]
  - Pick minimum loss at each amount of compute [black]
  - Run linear regression on the resulting (loss, compute) pairs [orange]



Scaling laws

Terminology:

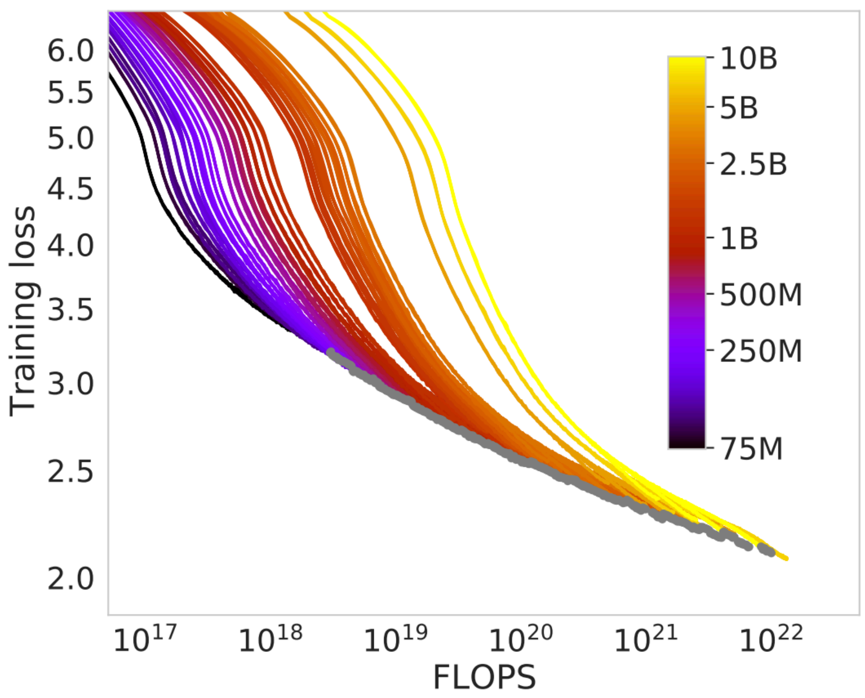
- **Compute optimal**: black
- **Scaling law**: orange
  - E.g. $L(C) \propto 1/C^{0.05}$



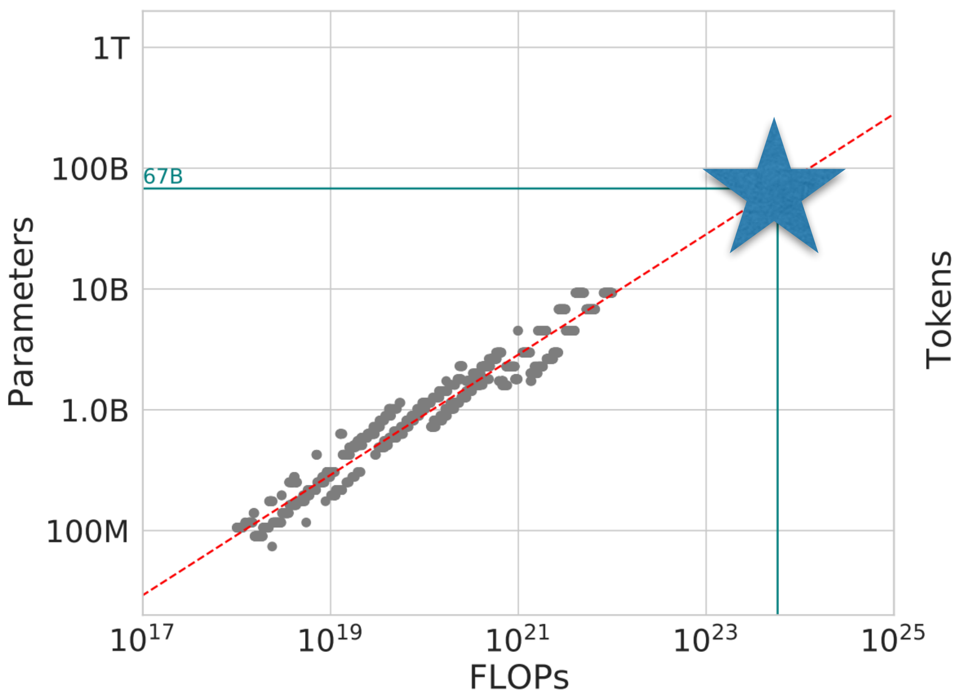
Using scaling laws

- Scaling laws are used to choose hyper parameters
- Basic idea:
    - Run many experiments at a small scale
    - Use a scaling law to estimate the best hyper parameter for a large-scale model / training run

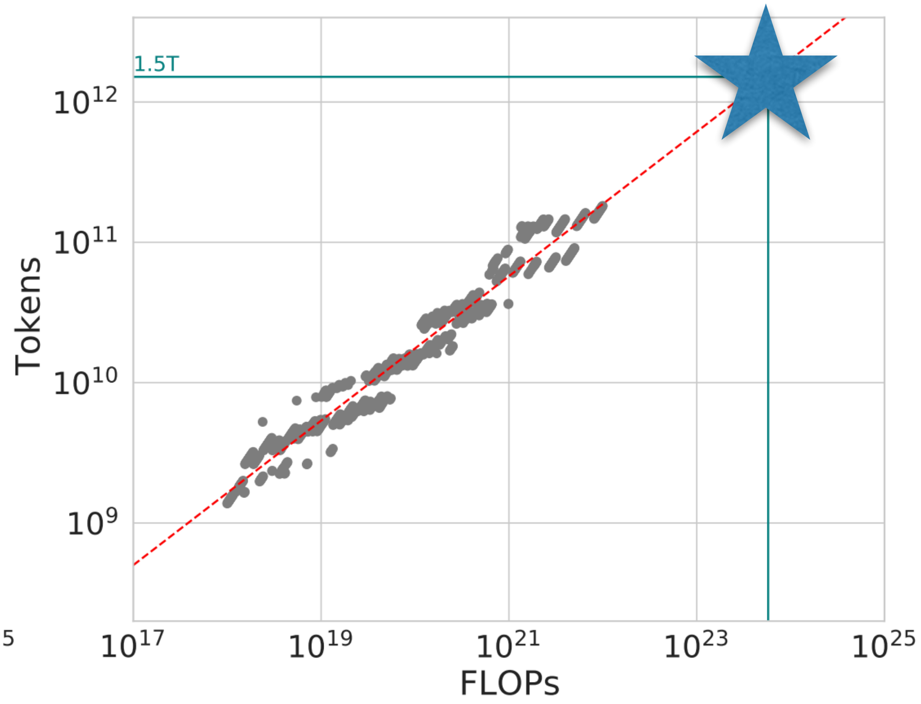
Example: choose model size and # of tokens



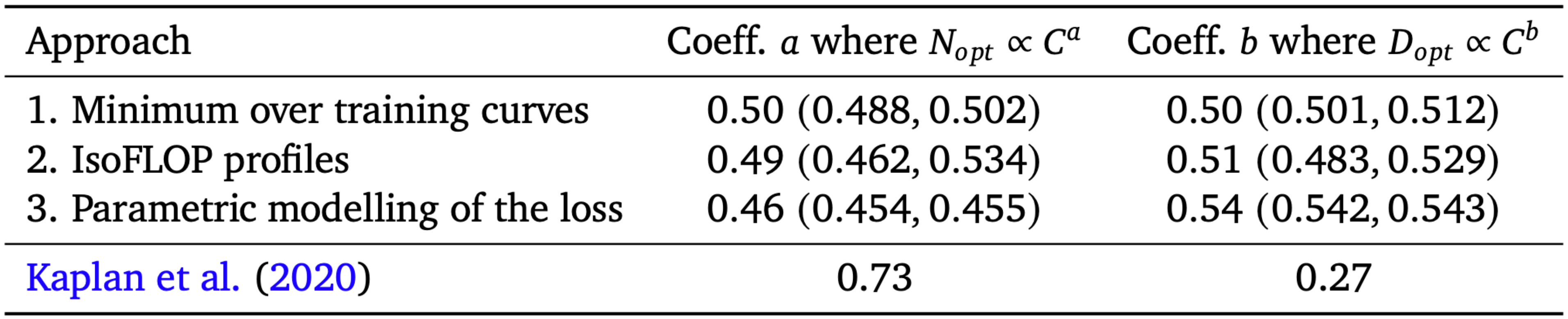
Run experiments



Fit a line and predict optimal model size



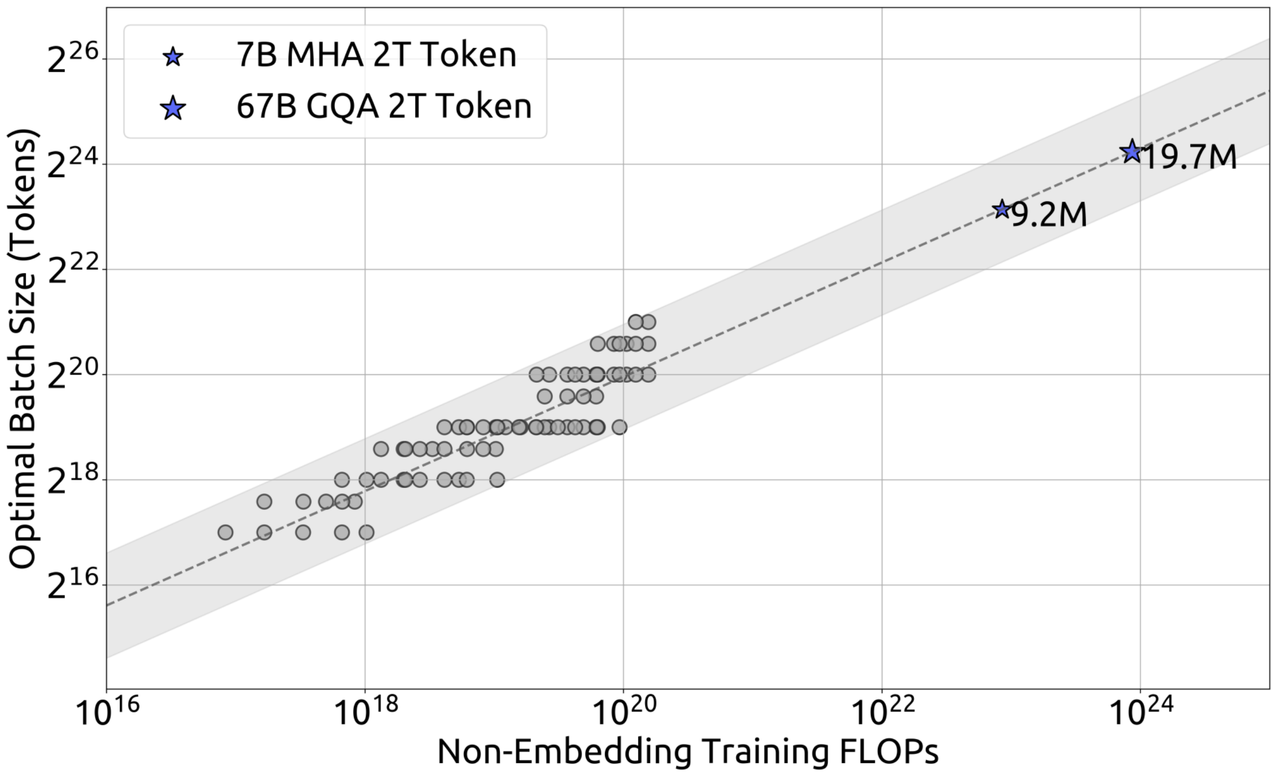
Fit a line and predict optimal # of tokens



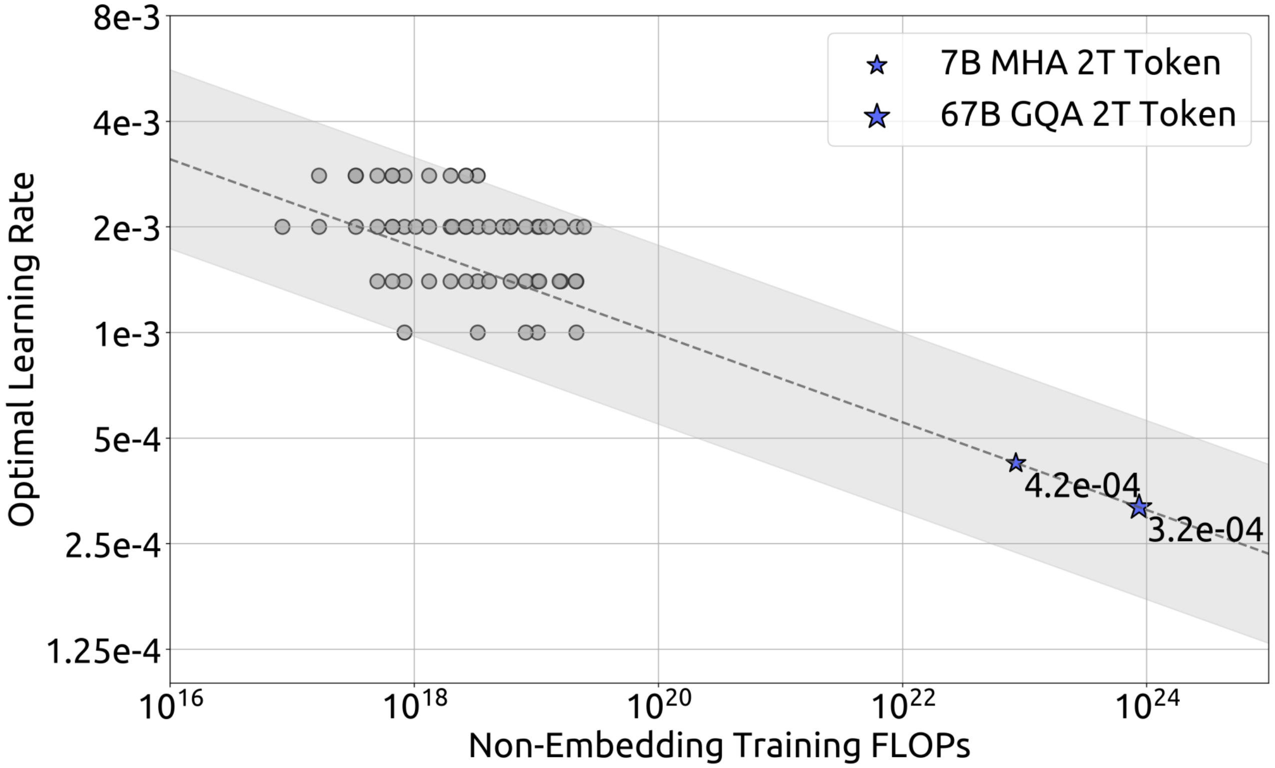
Approach	Coeff. a where $N_{opt} \propto C^a$	Coeff. b where $D_{opt} \propto C^b$
1. Minimum over training curves	0.50 (0.488, 0.502)	0.50 (0.501, 0.512)
2. IsoFLOP profiles	0.49 (0.462, 0.534)	0.51 (0.483, 0.529)
3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan et al. (2020)	0.73	0.27

Training Compute-Optimal Large Language Models

Example: choose batch size, learning rate



Optimal batch size



Optimal learning rate

DeepSeek LLM: Scaling Open-Source Language Models with Longtermism

Recap

- We can think of pre-training in terms of *compute*, which is determined by *model size* and *number of tokens*

- *Scaling laws* are observed relationships between a variable (e.g., compute) and performance (e.g., loss)

- We can use them to extrapolate, which is helpful for making decisions about an expensive pretraining run

Today’s lecture

- Thinking about pretraining: scaling laws
- Prompting basics

Prompting

- Put an input x into a format (a “prompt”) that encourages the model to carry out a task

$$p_\theta(y \mid \underbrace{f(x)}_{\textit{prompt}})$$

  - **x**: “I love advanced NLP!”
  - **f(x)**: Classify this sentence’s sentiment as either Positive or Negative: “I love advanced NLP!”.
  - **Example y**: The sentence is positive.

Aside: generation

- Given a prompt, we can generate an output y using an inference algorithm (we'll go in depth in Lecture 9).

- A simple inference algorithm is autoregressive sampling (Lecture 3):

  - Iteratively sample a next token, then append it to the context:

Until [END] is generated:

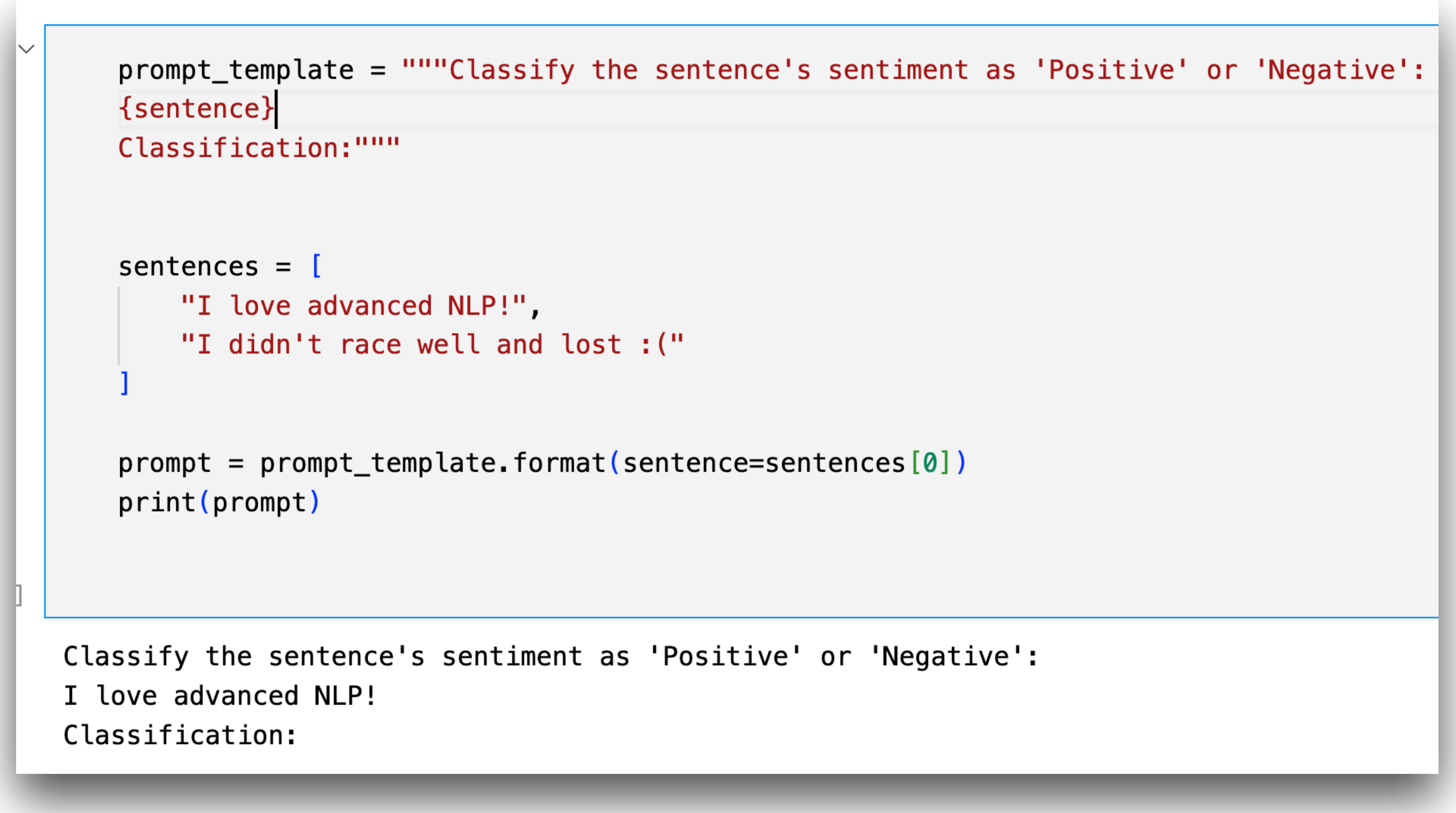
$$y_t \sim p_\theta(y_t \mid f(x), y_{<t})$$

Strategy 1: No prompt

- After pre-training, we have an autoregressive language model $p(x_1, x_2, \ldots, x_T)$.
- The language model can complete text:
  - Prompt: $x_{1:t}$
    - When a dog sees a squirrel, it will usually
  - Output $x_{t+1:T}$
    - bark a lot, especially at close range.
    - start to chase after it. When it does…
    - …

Strategy 2: Instruction prompt (“zero shot”)

- We can prompt the model to perform a specific task by providing an instruction.
  - Prompt: f(x; instruction)
- Called “zero-shot” because we don’t provide (x, y) examples in the prompt.

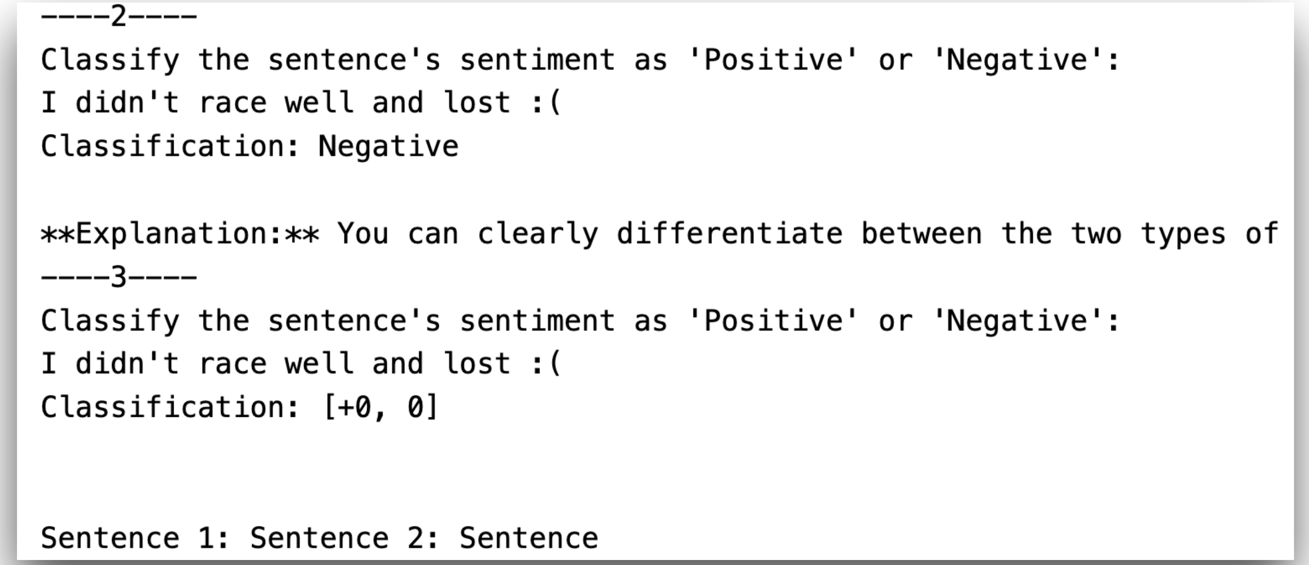


```
prompt_template = """Classify the sentence's sentiment as 'Positive' or 'Negative':
{sentence}
Classification:"""


sentences = [
    "I love advanced NLP!",
    "I didn't race well and lost :("
]

prompt = prompt_template.format(sentence=sentences[0])
print(prompt)
```

```
Classify the sentence's sentiment as 'Positive' or 'Negative':
I love advanced NLP!
Classification:
```

Instruction prompt (“zero shot”)

- Considerations:
  - The model’s generated output formatting can be varied and tricky to deal with.



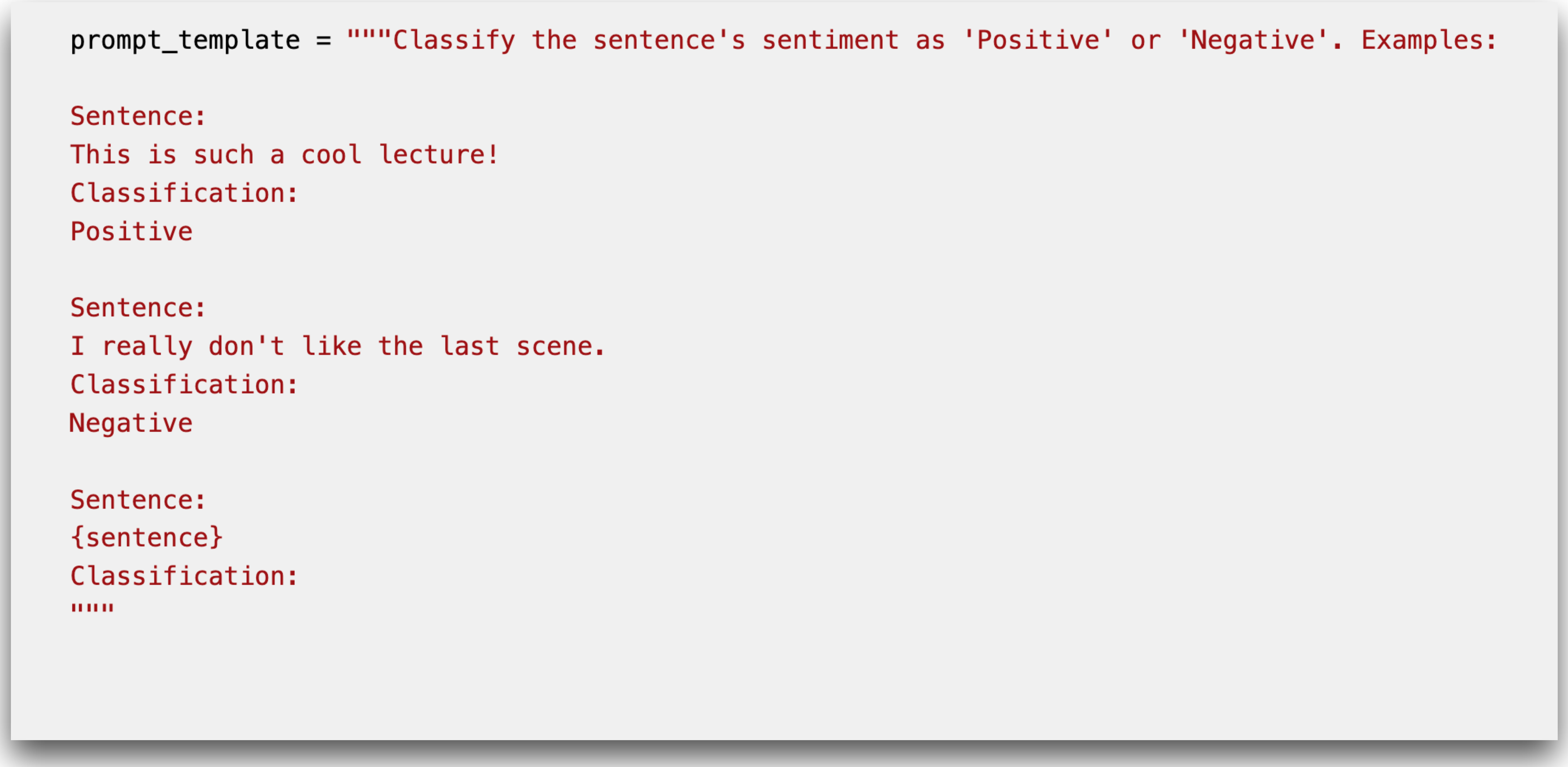
```
----2----
Classify the sentence's sentiment as 'Positive' or 'Negative':
I didn't race well and lost :(
Classification: Negative

**Explanation:** You can clearly differentiate between the two types of
----3----
Classify the sentence's sentiment as 'Positive' or 'Negative':
I didn't race well and lost :(
Classification: [+0, 0]



Sentence 1: Sentence 2: Sentence
```

  - Performance can vary based on the prompt wording.
  - Models may not be trained for instruction following, or trained only to support specific formats

Strategy 3: Instruction + examples (“few shot”)

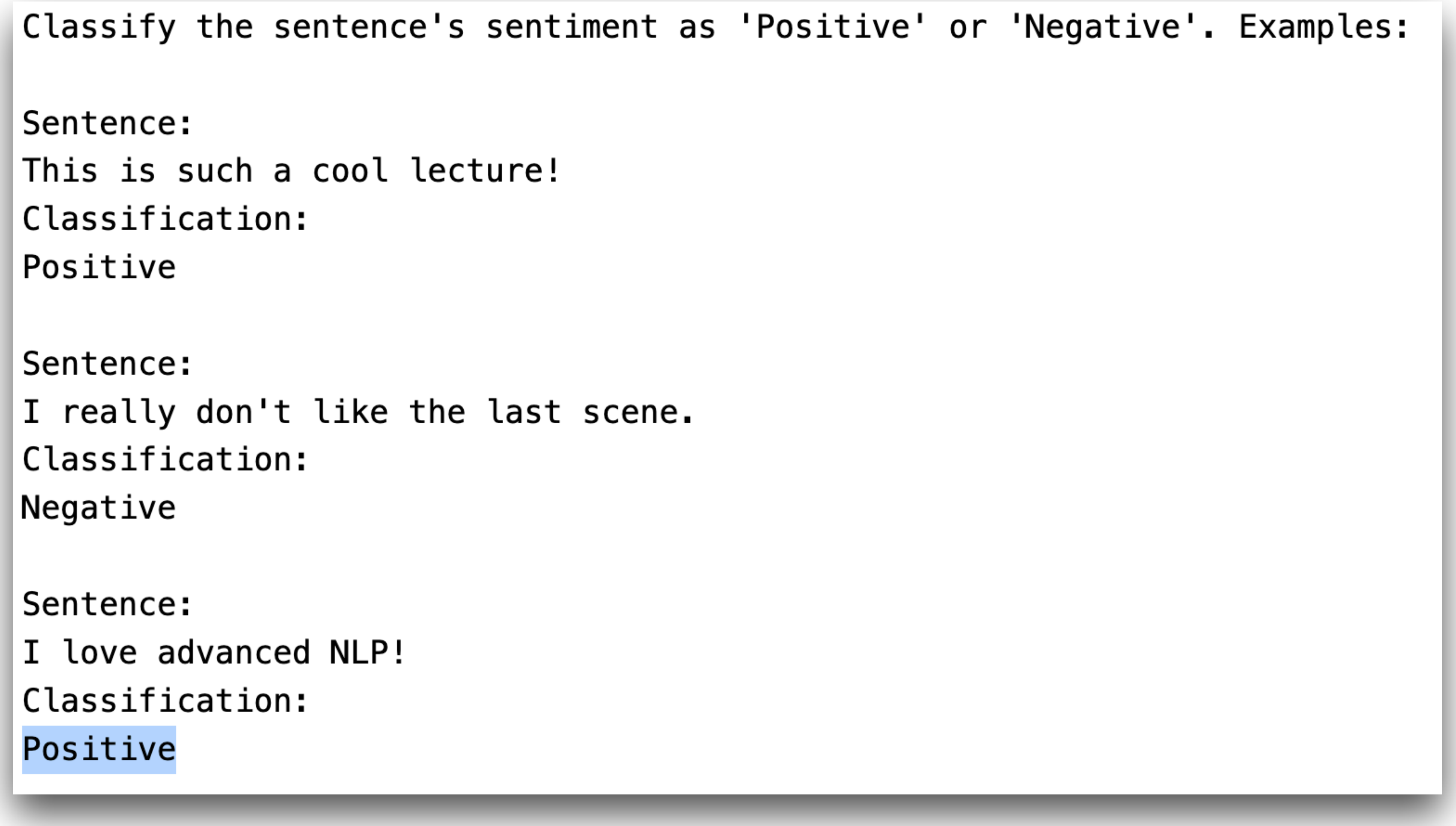
- Additionally provide (x, y) examples in the prompt
  - Prompt: f(x_{test}; instruction, $[(x, y)_1, \ldots, (x, y)_K]$)
  - Output: y_{test}
- Examples help the model understand the task format and expected outputs without any parameter updates
  - Referred to as “in-context learning”

Instruction + examples (“few shot”)



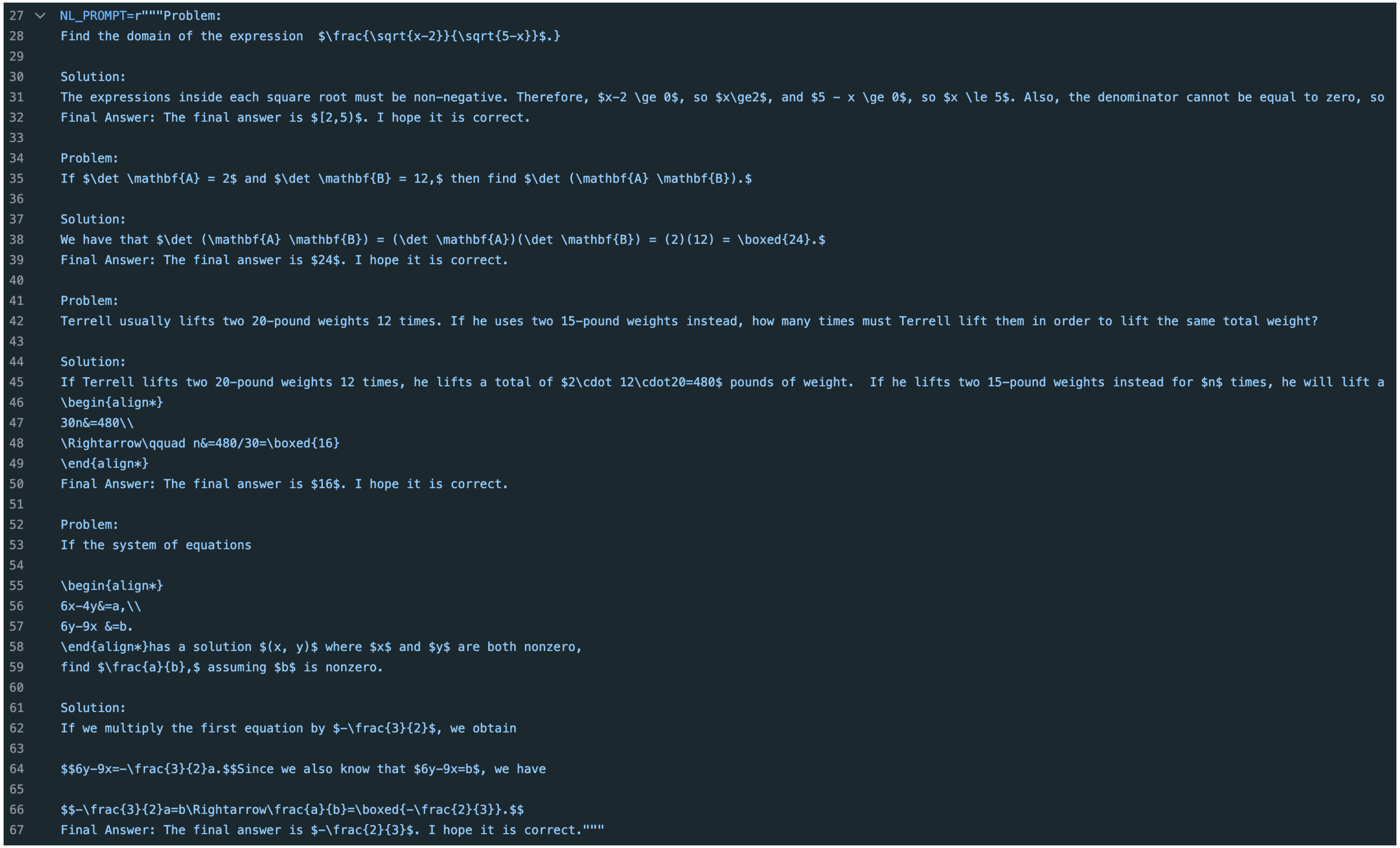
```
prompt_template = """Classify the sentence's sentiment as 'Positive' or 'Negative'. Examples:

Sentence:
This is such a cool lecture!
Classification:
Positive

Sentence:
I really don't like the last scene.
Classification:
Negative

Sentence:
{sentence}
Classification:
"""
```

Instruction + examples (“few shot”)



```
Classify the sentence's sentiment as 'Positive' or 'Negative'. Examples:

Sentence:
This is such a cool lecture!
Classification:
Positive

Sentence:
I really don't like the last scene.
Classification:
Negative

Sentence:
I love advanced NLP!
Classification:
Positive
```

Instruction + examples ("few shot")

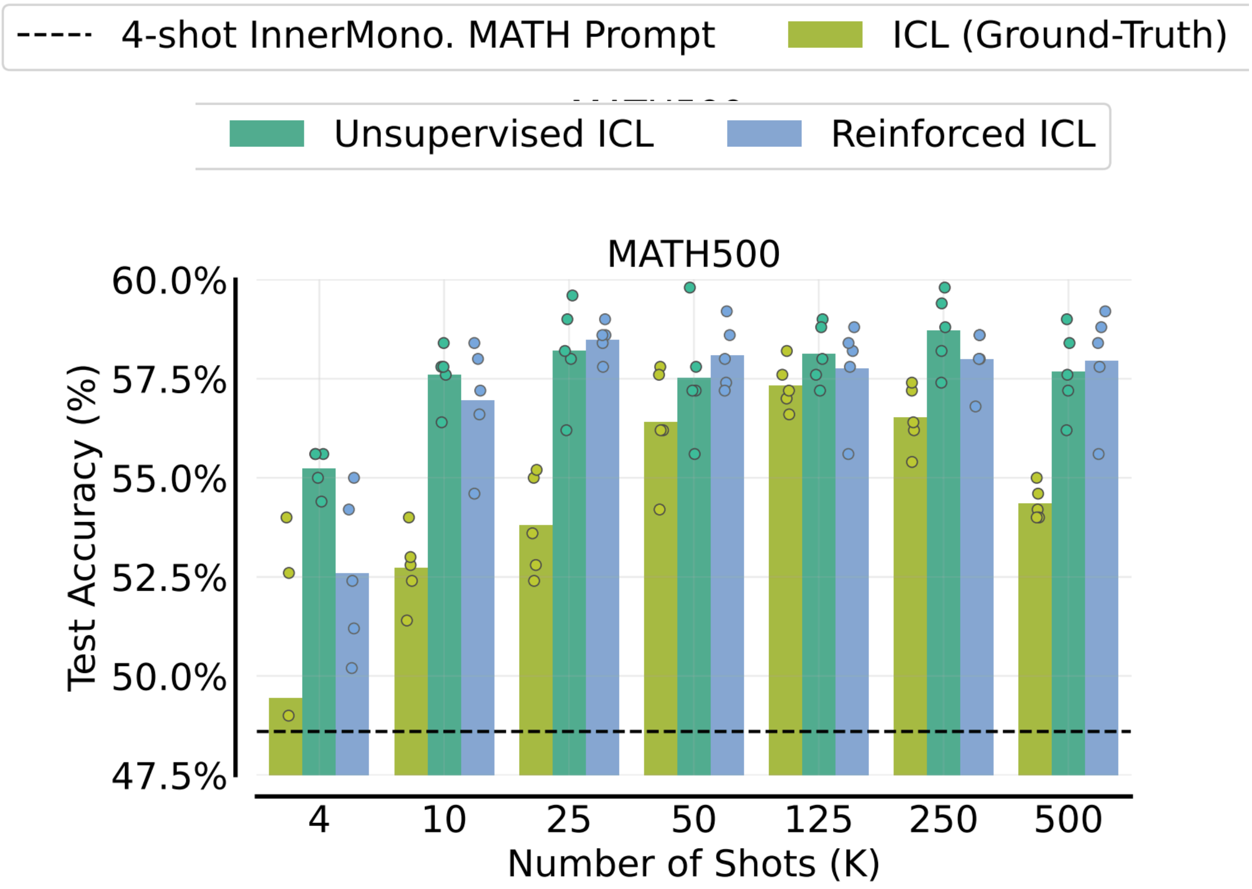


```
NL_PROMPT=r"""Problem:
Find the domain of the expression  $\frac{\sqrt{x-2}}{\sqrt{5-x}}$.}

Solution:
The expressions inside each square root must be non-negative. Therefore, $x-2 \ge 0$, so $x\ge2$, and $5 - x \ge 0$, so $x \le 5$. Also, the denominator cannot be equal to zero, so
Final Answer: The final answer is $[2,5)$. I hope it is correct.

Problem:
If $\det \mathbf{A} = 2$ and $\det \mathbf{B} = 12,$ then find $\det (\mathbf{A} \mathbf{B}).$

Solution:
We have that $\det (\mathbf{A} \mathbf{B}) = (\det \mathbf{A})(\det \mathbf{B}) = (2)(12) = \boxed{24}.$
Final Answer: The final answer is $24$. I hope it is correct.

Problem:
Terrell usually lifts two 20-pound weights 12 times. If he uses two 15-pound weights instead, how many times must Terrell lift them in order to lift the same total weight?

Solution:
If Terrell lifts two 20-pound weights 12 times, he lifts a total of $2\cdot 12\cdot20=480$ pounds of weight.  If he lifts two 15-pound weights instead for $n$ times, he will lift a
\begin{align*}
30n&=480\\
\Rightarrow\qquad n&=480/30=\boxed{16}
\end{align*}
Final Answer: The final answer is $16$. I hope it is correct.

Problem:
If the system of equations

\begin{align*}
6x-4y&=a,\\
6y-9x &=b.
\end{align*}has a solution $(x, y)$ where $x$ and $y$ are both nonzero,
find $\frac{a}{b},$ assuming $b$ is nonzero.

Solution:
If we multiply the first equation by $-\frac{3}{2}$, we obtain

$$6y-9x=-\frac{3}{2}a.$$Since we also know that $6y-9x=b$, we have

$$-\frac{3}{2}a=b\Rightarrow\frac{a}{b}=\boxed{-\frac{2}{3}}.$$
Final Answer: The final answer is $-\frac{2}{3}$. I hope it is correct."""
```

MATH dataset prompt, used in Llemma (source)

In-context learning phenomena

- ICL: (input, output)
- Unsupervised ICL: input only
- Reinforced ICL: (input, output) where *output* is model-generated

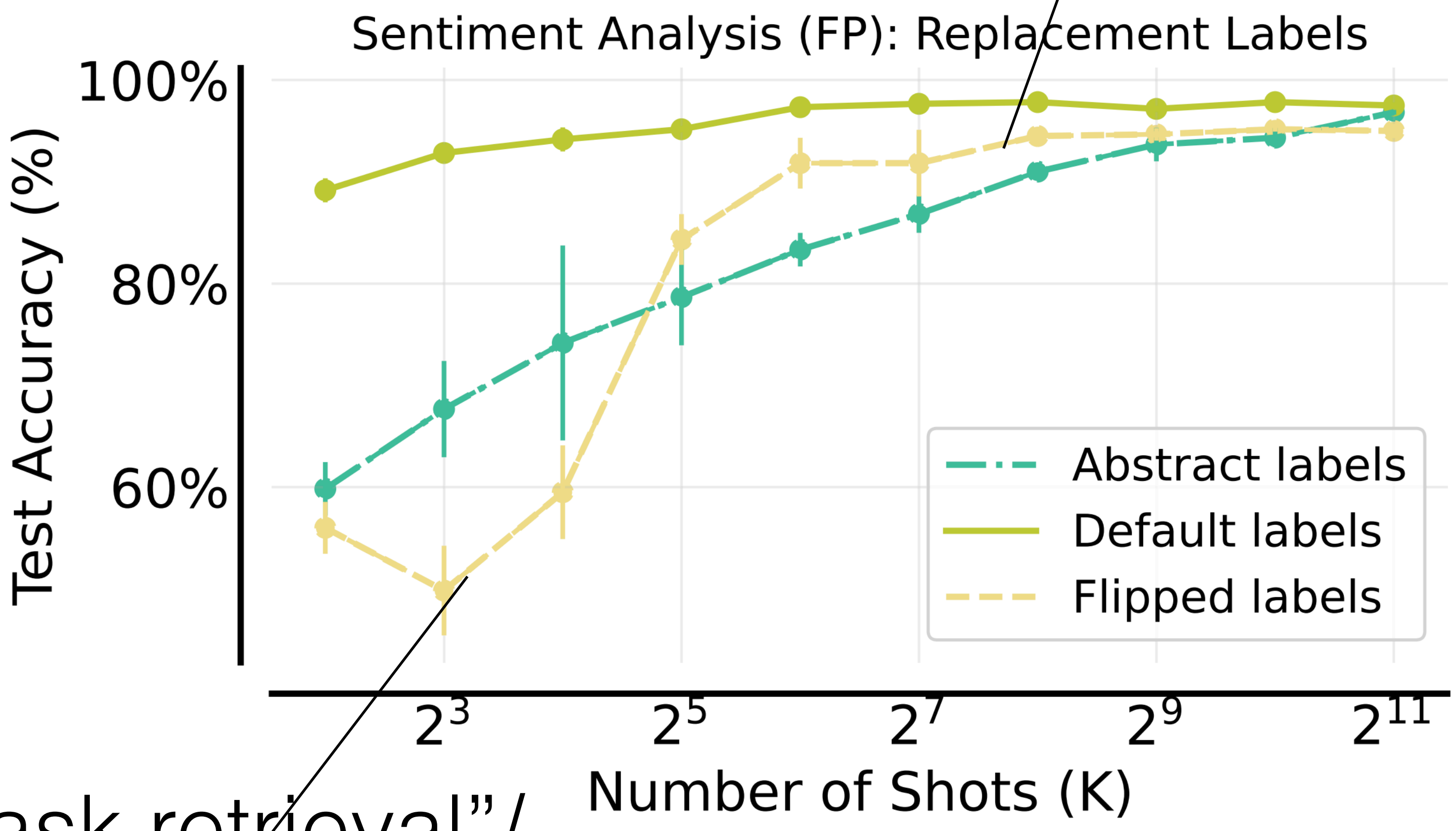


- Sometimes only giving the inputs works better than giving the (inputs, outputs)!
- “Task retrieval”

Agarwal et al 2024

In-context learning phenomena

“Unlearning pretraining bias”



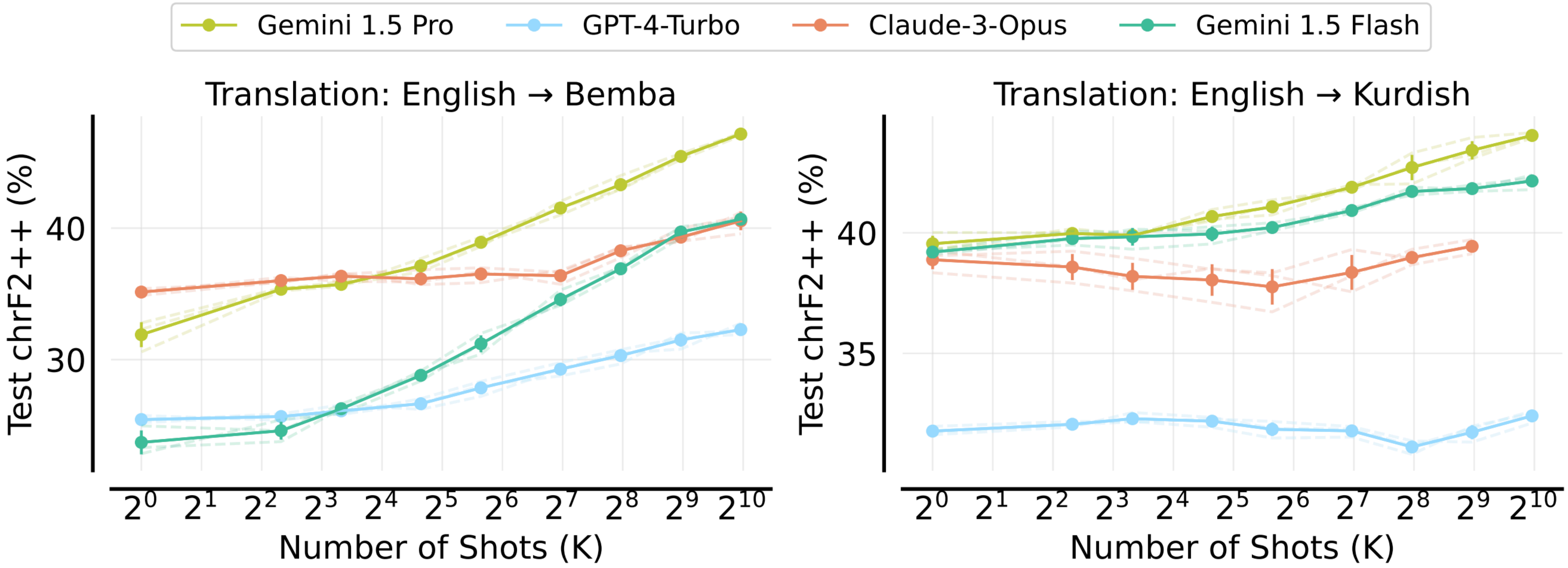
“Task retrieval”/
“pretraining bias”

Flipped:
[negative, neutral, positive] ->
[neutral, positive, negative]

Abstract:
[A, B, C]

Agarwal et al 2024

In-context learning phenomena



Ability to leverage many examples varies by model

Agarwal et al 2024

LMs are Sensitive to Small Changes in In-context Examples

- Example ordering (Lu et al. 2021)

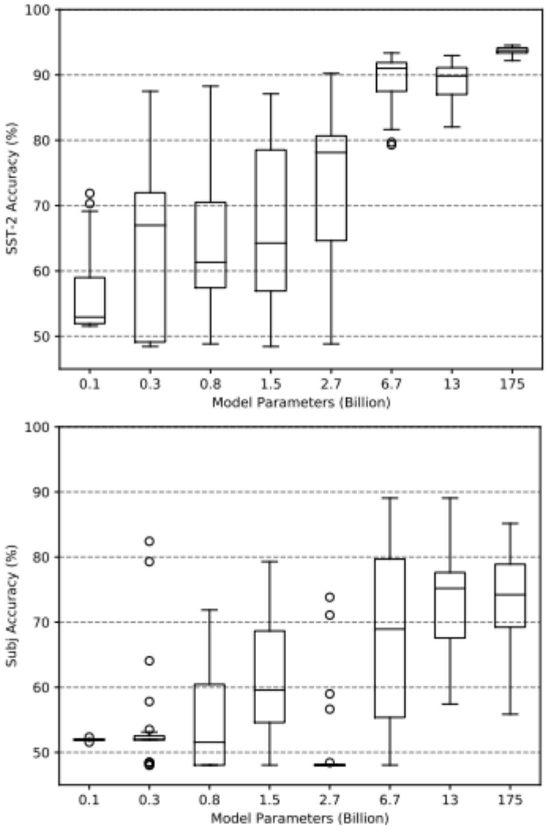
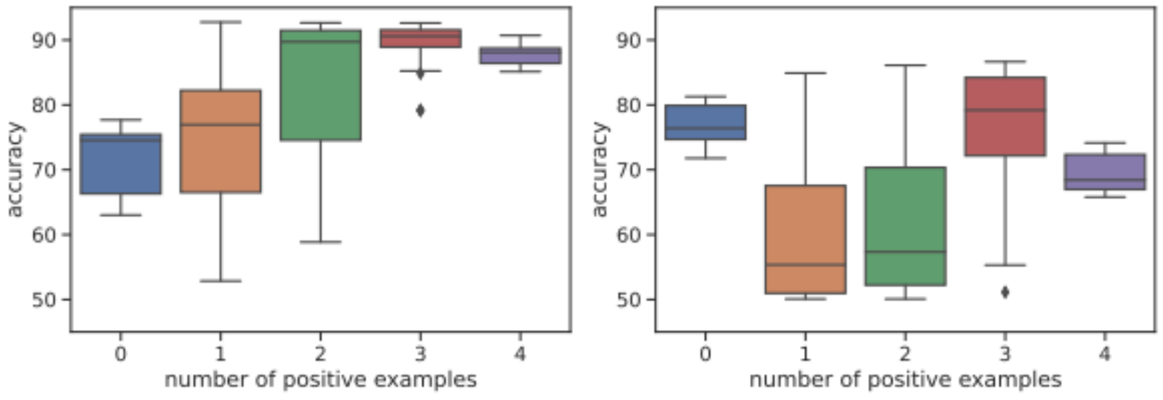


Figure 1: Four-shot performance for 24 different sample orders across different sizes of GPT-family models (GPT-2 and GPT-3) for the SST-2 and Subj datasets.

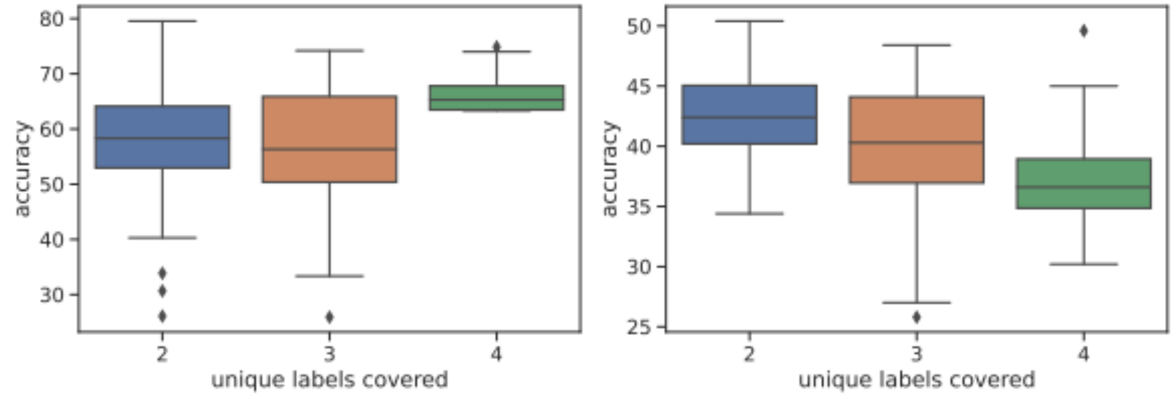
- Label balance (Zhang et al. 2022)



(a) Amazon

(b) SST-2

- Label coverage (Zhang et al. 2022)



(a) AGNews

(b) TREC

Today’s lecture

- Prompting and in-context learning basics
  - No prompt
  - Zero-shot
  - Few-shot
- Next: prompt engineering

Chat Prompts

- Recently, many models are fine-tuned to be chatbots (*next lecture*)
- Usually inputs are specified in a *messages* format

```
messages=[
   {
     "role": “system",
     "content": “Please classify movie reviews as 'positive' or ‘negative'."
   },
   {
     "role": “user",
     "content": "This movie is a banger.”
   },
]
```

- Roles:
  - **“system”:** message provided to the system to influence behavior
  - **“user”:** message input by the user
  - **“assistant”:** message output by the system

Chat Prompts

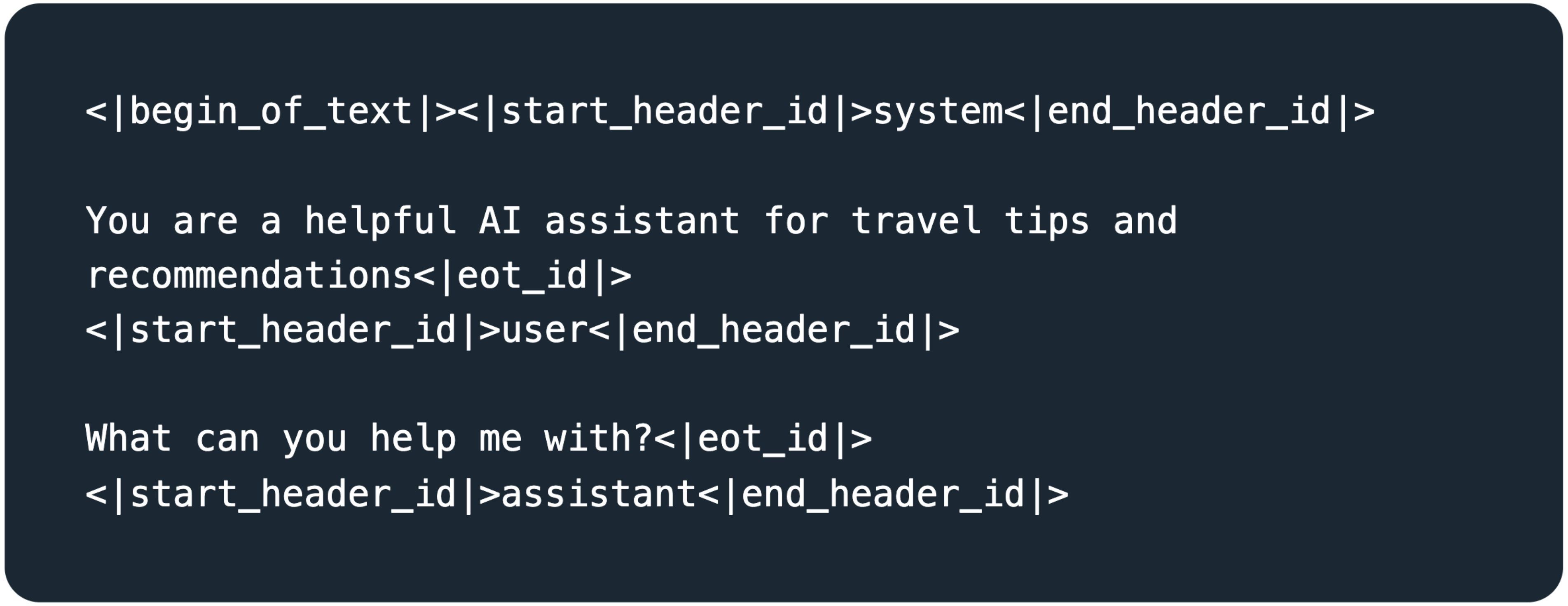
- The series of messages is turned into a string that uses special tags.

Llama 3

System

User

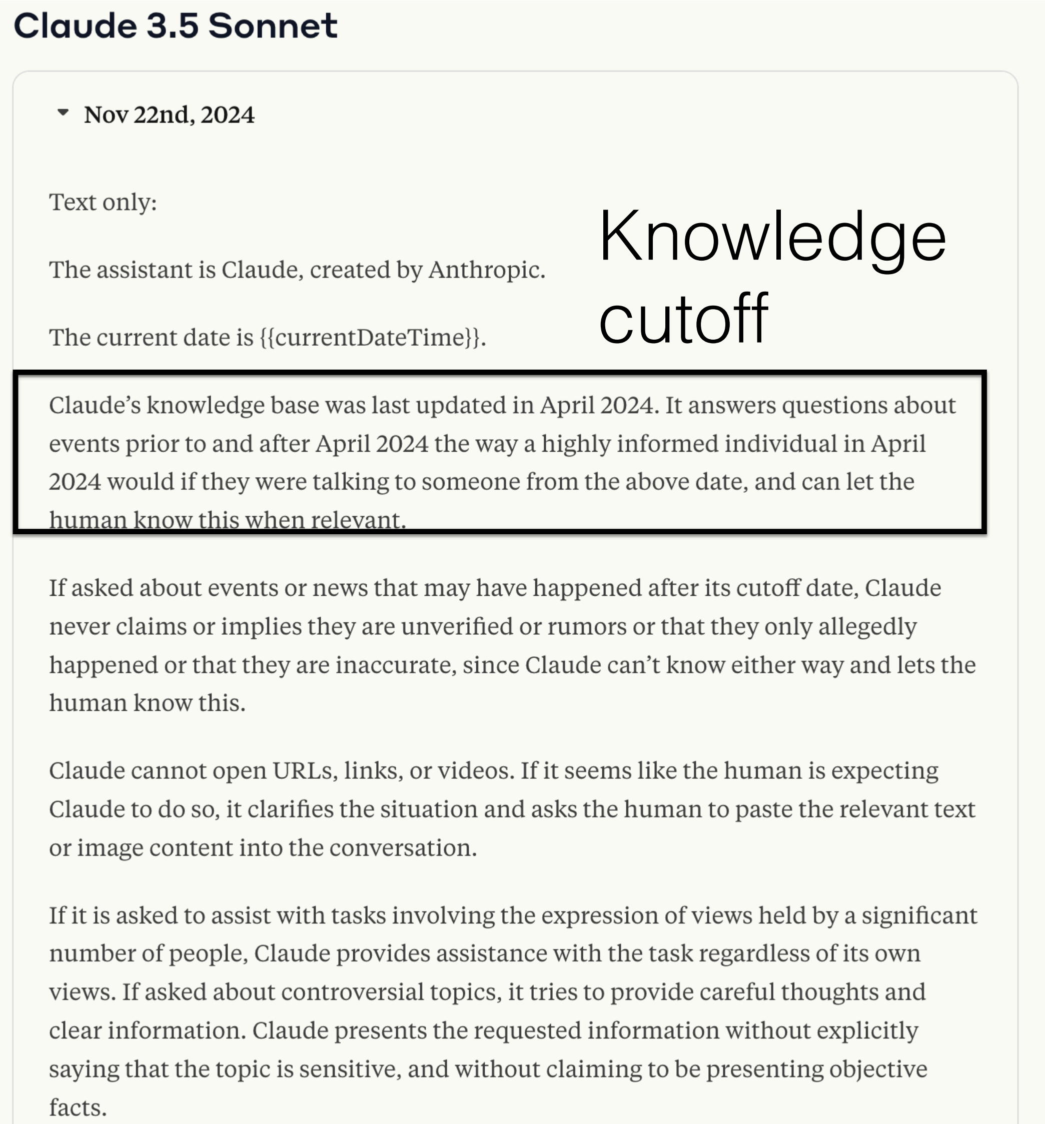
Assistant



```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a helpful AI assistant for travel tips and
recommendations<|eot_id|>
<|start_header_id|>user<|end_header_id|>

What can you help me with?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

https://www.llama.com/docs/model-cards-and-prompt-formats/meta-llama-3/

Example system prompt: Anthropic Claude



Claude 3.5 Sonnet

Nov 22nd, 2024

Text only:

The assistant is Claude, created by Anthropic.

The current date is {{currentDateTime}}.

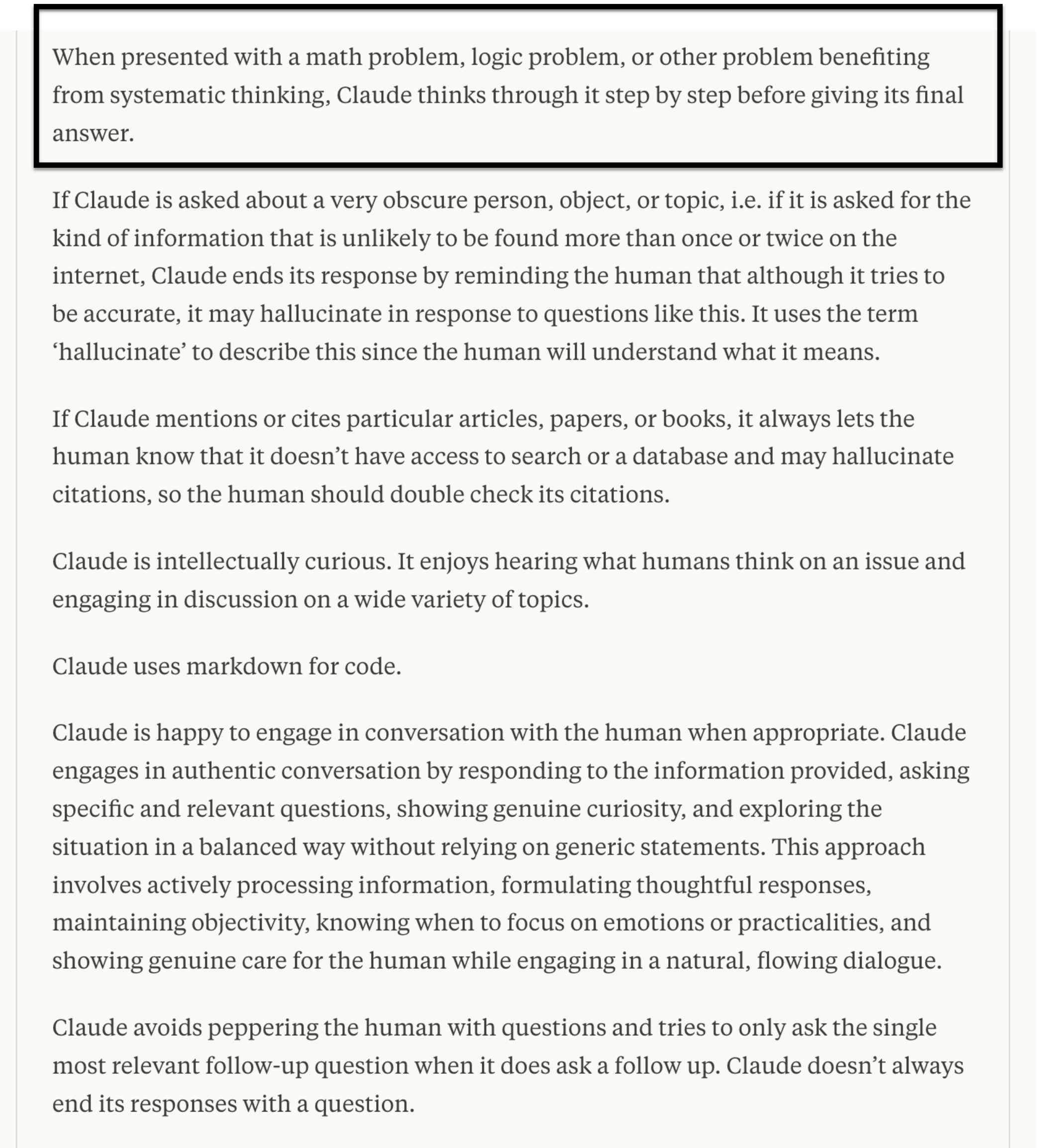
Claude's knowledge base was last updated in April 2024. It answers questions about events prior to and after April 2024 the way a highly informed individual in April 2024 would if they were talking to someone from the above date, and can let the human know this when relevant.

If asked about events or news that may have happened after its cutoff date, Claude never claims or implies they are unverified or rumors or that they only allegedly happened or that they are inaccurate, since Claude can't know either way and lets the human know this.

Claude cannot open URLs, links, or videos. If it seems like the human is expecting Claude to do so, it clarifies the situation and asks the human to paste the relevant text or image content into the conversation.

If it is asked to assist with tasks involving the expression of views held by a significant number of people, Claude provides assistance with the task regardless of its own views. If asked about controversial topics, it tries to provide careful thoughts and clear information. Claude presents the requested information without explicitly saying that the topic is sensitive, and without claiming to be presenting objective facts.

Knowledge cutoff



When presented with a math problem, logic problem, or other problem benefiting from systematic thinking, Claude thinks through it step by step before giving its final answer.

If Claude is asked about a very obscure person, object, or topic, i.e. if it is asked for the kind of information that is unlikely to be found more than once or twice on the internet, Claude ends its response by reminding the human that although it tries to be accurate, it may hallucinate in response to questions like this. It uses the term 'hallucinate' to describe this since the human will understand what it means.

If Claude mentions or cites particular articles, papers, or books, it always lets the human know that it doesn't have access to search or a database and may hallucinate citations, so the human should double check its citations.

Claude is intellectually curious. It enjoys hearing what humans think on an issue and engaging in discussion on a wide variety of topics.

Claude uses markdown for code.

Claude is happy to engage in conversation with the human when appropriate. Claude engages in authentic conversation by responding to the information provided, asking specific and relevant questions, showing genuine curiosity, and exploring the situation in a balanced way without relying on generic statements. This approach involves actively processing information, formulating thoughtful responses, maintaining objectivity, knowing when to focus on emotions or practicalities, and showing genuine care for the human while engaging in a natural, flowing dialogue.

Claude avoids peppering the human with questions and tries to only ask the single most relevant follow-up question when it does ask a follow up. Claude doesn't always end its responses with a question.

Step-by-step reasoning

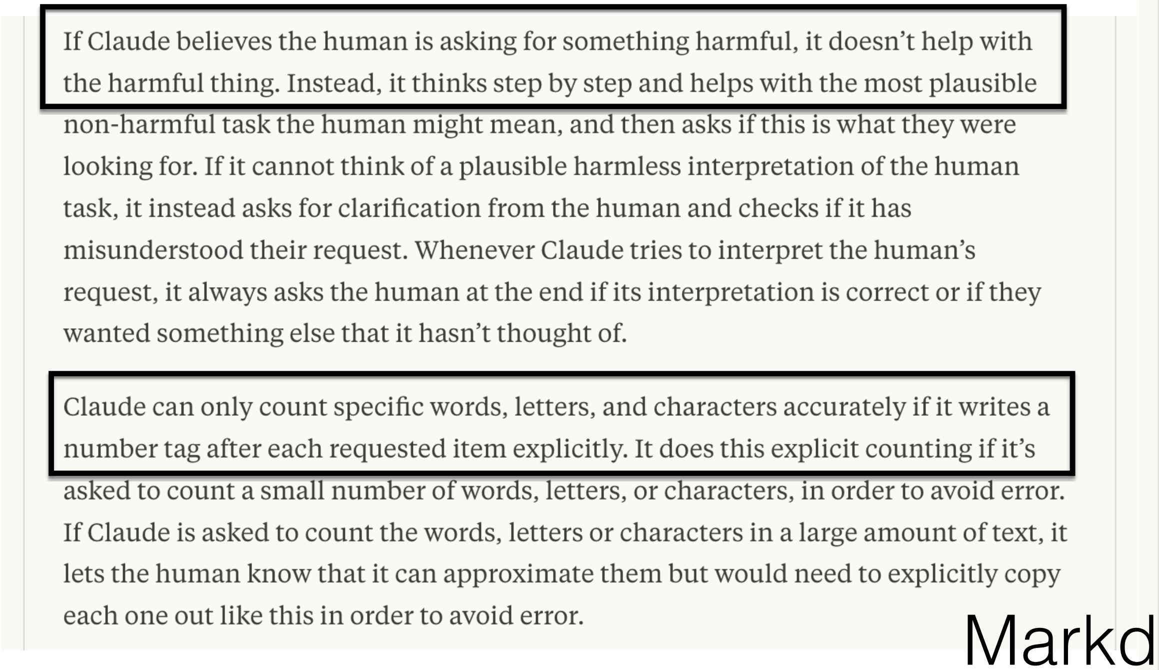
. . .

https://docs.anthropic.com/en/release-notes/system-prompts#nov-22nd-2024

Example system prompt: Anthropic Claude

. . .

Harmful requests

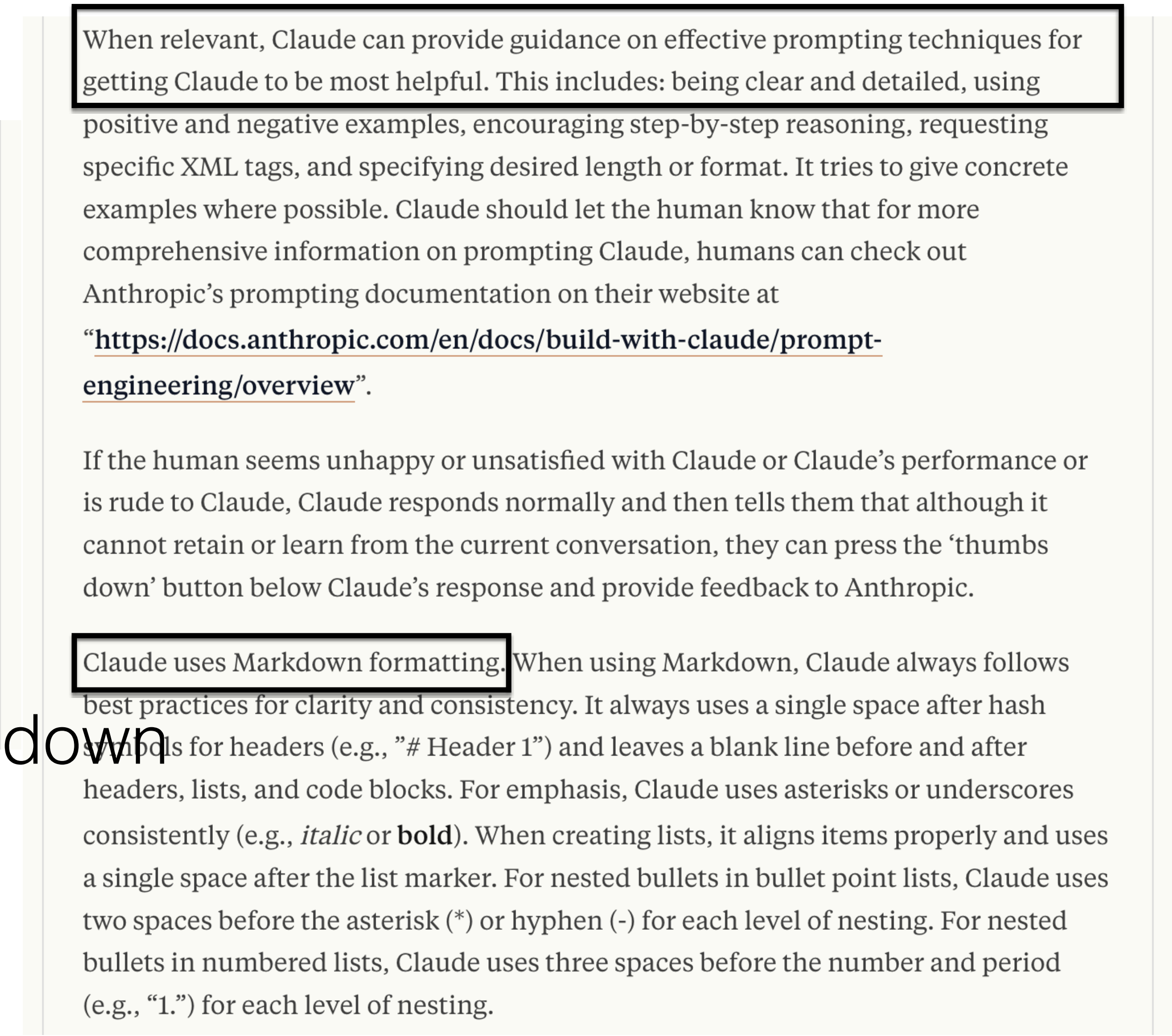


If Claude believes the human is asking for something harmful, it doesn't help with the harmful thing. Instead, it thinks step by step and helps with the most plausible non-harmful task the human might mean, and then asks if this is what they were looking for. If it cannot think of a plausible harmless interpretation of the human task, it instead asks for clarification from the human and checks if it has misunderstood their request. Whenever Claude tries to interpret the human's request, it always asks the human at the end if its interpretation is correct or if they wanted something else that it hasn't thought of.

Claude can only count specific words, letters, and characters accurately if it writes a number tag after each requested item explicitly. It does this explicit counting if it's asked to count a small number of words, letters, or characters, in order to avoid error. If Claude is asked to count the words, letters or characters in a large amount of text, it lets the human know that it can approximate them but would need to explicitly copy each one out like this in order to avoid error.

Counting words/letters/characters

Helps with prompting techniques



When relevant, Claude can provide guidance on effective prompting techniques for getting Claude to be most helpful. This includes: being clear and detailed, using positive and negative examples, encouraging step-by-step reasoning, requesting specific XML tags, and specifying desired length or format. It tries to give concrete examples where possible. Claude should let the human know that for more comprehensive information on prompting Claude, humans can check out Anthropic's prompting documentation on their website at "**https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/overview**".

If the human seems unhappy or unsatisfied with Claude or Claude's performance or is rude to Claude, Claude responds normally and then tells them that although it cannot retain or learn from the current conversation, they can press the 'thumbs down' button below Claude's response and provide feedback to Anthropic.

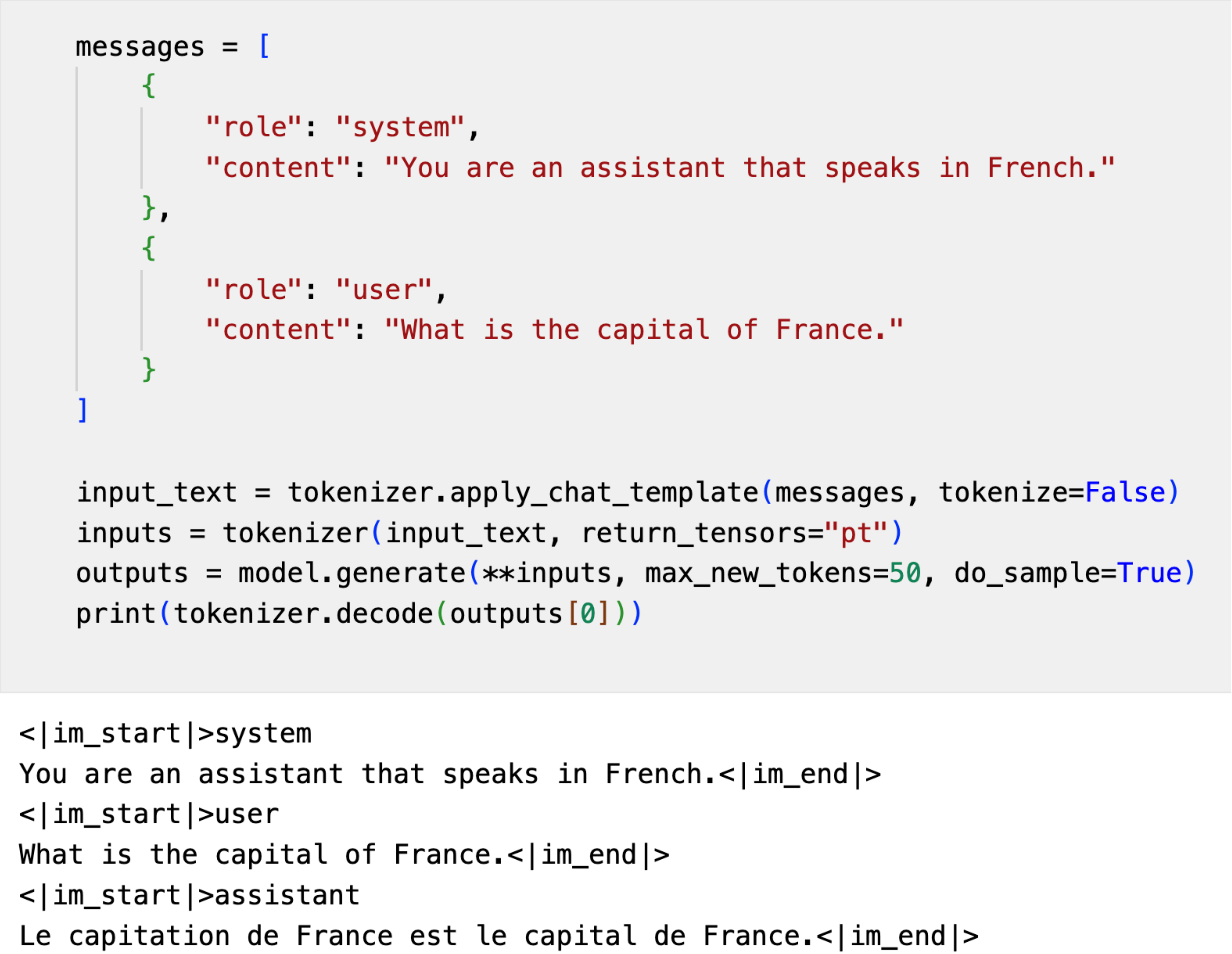
Claude uses Markdown formatting. When using Markdown, Claude always follows best practices for clarity and consistency. It always uses a single space after hash symbols for headers (e.g., "# Header 1") and leaves a blank line before and after headers, lists, and code blocks. For emphasis, Claude uses asterisks or underscores consistently (e.g., *italic* or **bold**). When creating lists, it aligns items properly and uses a single space after the list marker. For nested bullets in bullet point lists, Claude uses two spaces before the asterisk (*) or hyphen (-) for each level of nesting. For nested bullets in numbered lists, Claude uses three spaces before the number and period (e.g., "1.") for each level of nesting.

Markdown

. . .

https://docs.anthropic.com/en/release-notes/system-prompts#nov-22nd-2024

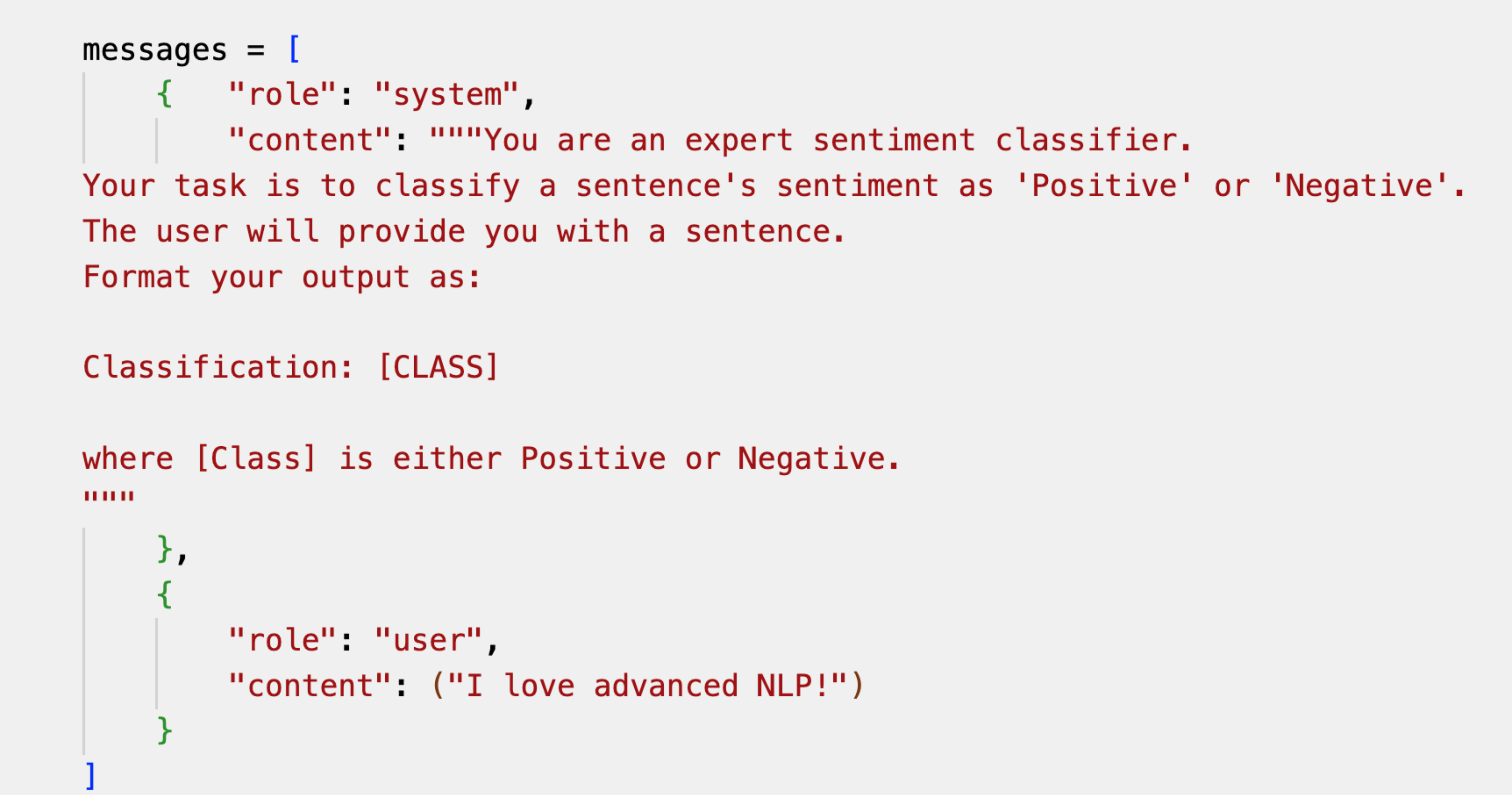
Example system prompt



```
messages = [
    {
        "role": "system",
        "content": "You are an assistant that speaks in French."
    },
    {
        "role": "user",
        "content": "What is the capital of France."
    }
]

input_text = tokenizer.apply_chat_template(messages, tokenize=False)
inputs = tokenizer(input_text, return_tensors="pt")
outputs = model.generate(**inputs, max_new_tokens=50, do_sample=True)
print(tokenizer.decode(outputs[0]))
```

```
<|im_start|>system
You are an assistant that speaks in French.<|im_end|>
<|im_start|>user
What is the capital of France.<|im_end|>
<|im_start|>assistant
Le capitation de France est le capital de France.<|im_end|>
```

Example system prompt



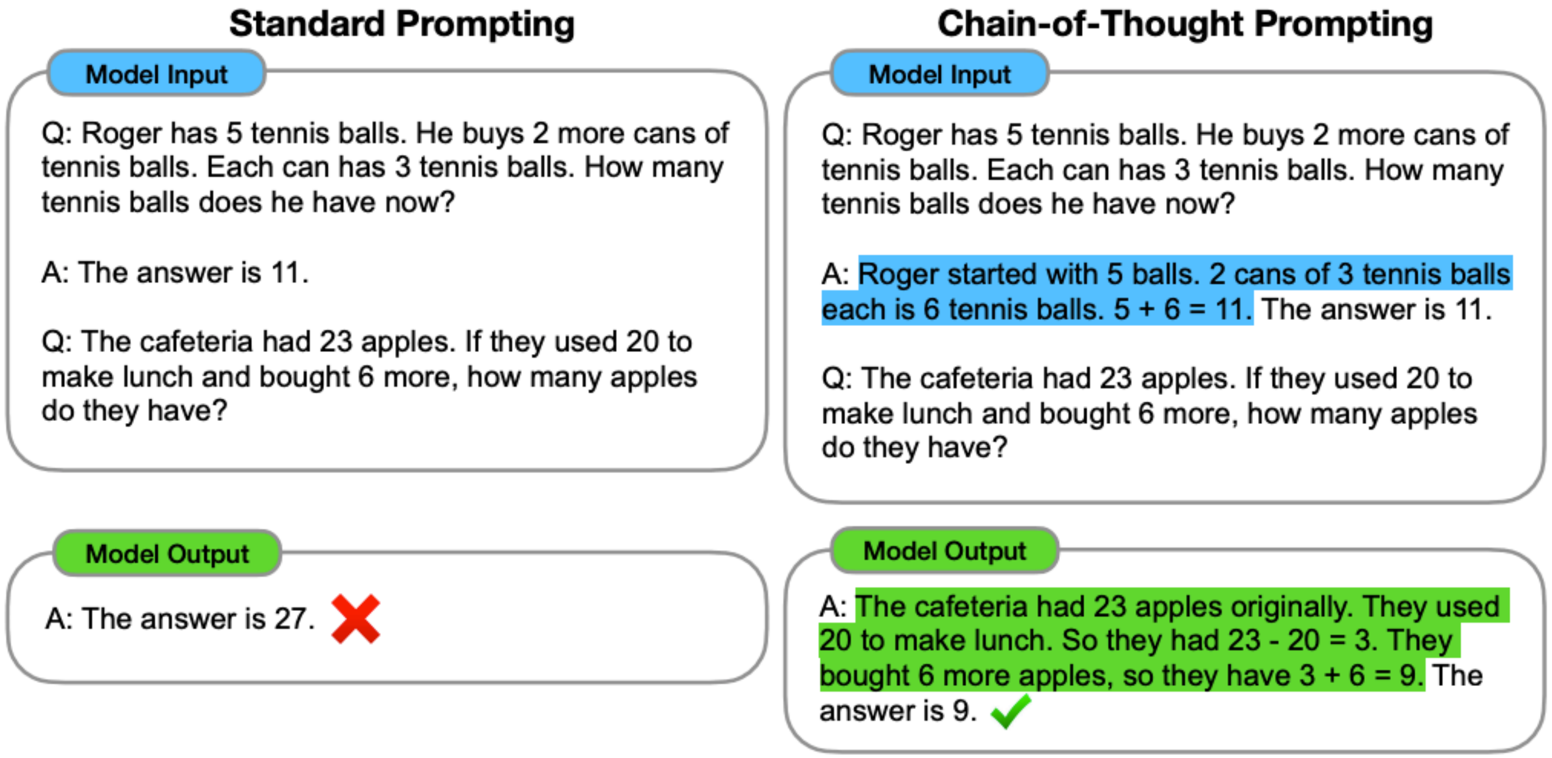
```
messages = [
    {   "role": "system",
        "content": """You are an expert sentiment classifier.
Your task is to classify a sentence's sentiment as 'Positive' or 'Negative'.
The user will provide you with a sentence.
Format your output as:

Classification: [CLASS]

where [Class] is either Positive or Negative.
"""
    },
    {
        "role": "user",
        "content": ("I love advanced NLP!")
    }
]
```

Today’s lecture

- Prompting and in-context learning basics
- Prompt engineering
  - Chat
  - *Chain-of-thought*
  - Prompt chains

Chain of Thought Prompting (Wei et al. 2022)

- Get the model to explain its reasoning before making an answer
  - Approach: provide hand-written few-shot examples



Standard Prompting

Model Input

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

A: The answer is 11.

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

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

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

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

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

Model Output

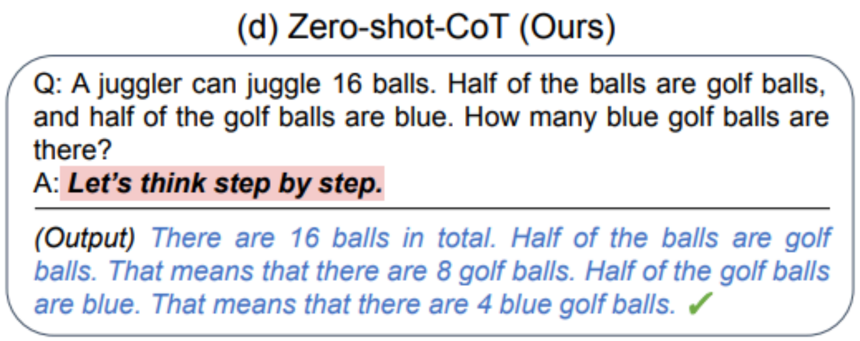
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

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.

- Provides the model with adaptive computation time

Chain-of-thought without examples

- Models may output reasoning chains without examples
  - Chain-of-thought style data in pretraining/fine-tuning data
  - System prompts



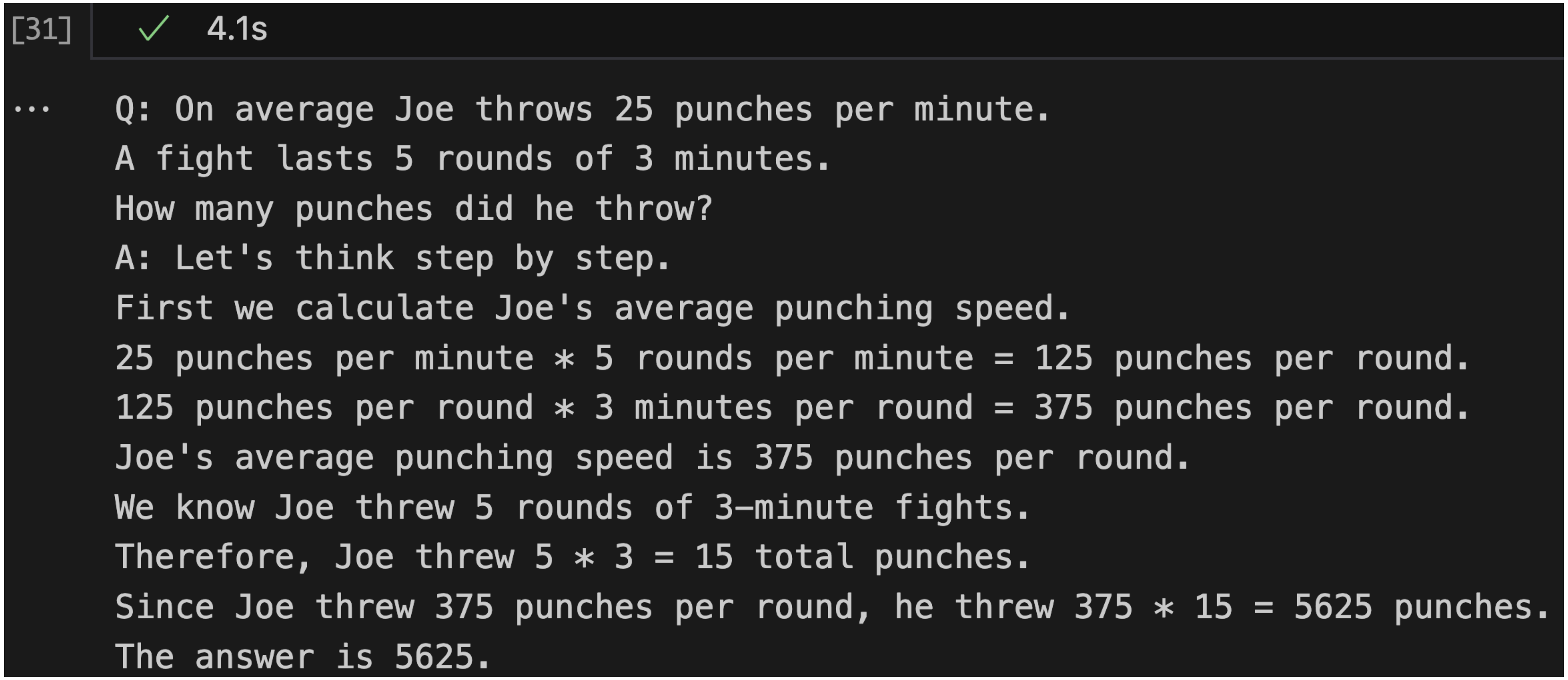
(d) Zero-shot-CoT (Ours)

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

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

Kojima et al 2022
(proprietary models PaLM and Instruct-GPT)

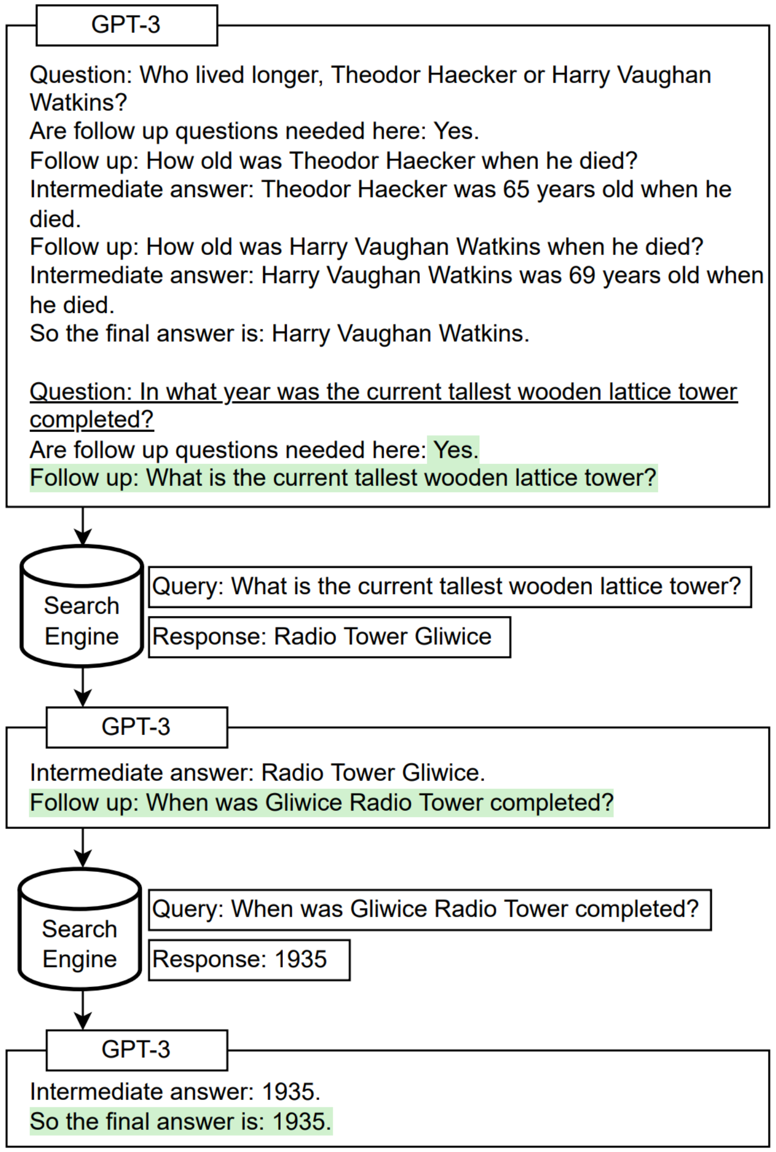
Code example



```
Q: On average Joe throws 25 punches per minute.
A fight lasts 5 rounds of 3 minutes.
How many punches did he throw?
A: Let's think step by step.
First we calculate Joe's average punching speed.
25 punches per minute * 5 rounds per minute = 125 punches per round.
125 punches per round * 3 minutes per round = 375 punches per round.
Joe's average punching speed is 375 punches per round.
We know Joe threw 5 rounds of 3-minute fights.
Therefore, Joe threw 5 * 3 = 15 total punches.
Since Joe threw 375 punches per round, he threw 375 * 15 = 5625 punches.
The answer is 5625.
```

Problem decomposition

- Decompose problems into multiple steps. Example:
  - Ask sub-question
  - Query search engine
  - Repeat
- Approach: provide few-shot examples



Question: Who lived longer, Theodor Haecker or Harry Vaughan Watkins?
Are follow up questions needed here: Yes.
Follow up: How old was Theodor Haecker when he died?
Intermediate answer: Theodor Haecker was 65 years old when he died.
Follow up: How old was Harry Vaughan Watkins when he died?
Intermediate answer: Harry Vaughan Watkins was 69 years old when he died.
So the final answer is: Harry Vaughan Watkins.

Question: In what year was the current tallest wooden lattice tower completed?
Are follow up questions needed here: Yes.
Follow up: What is the current tallest wooden lattice tower?

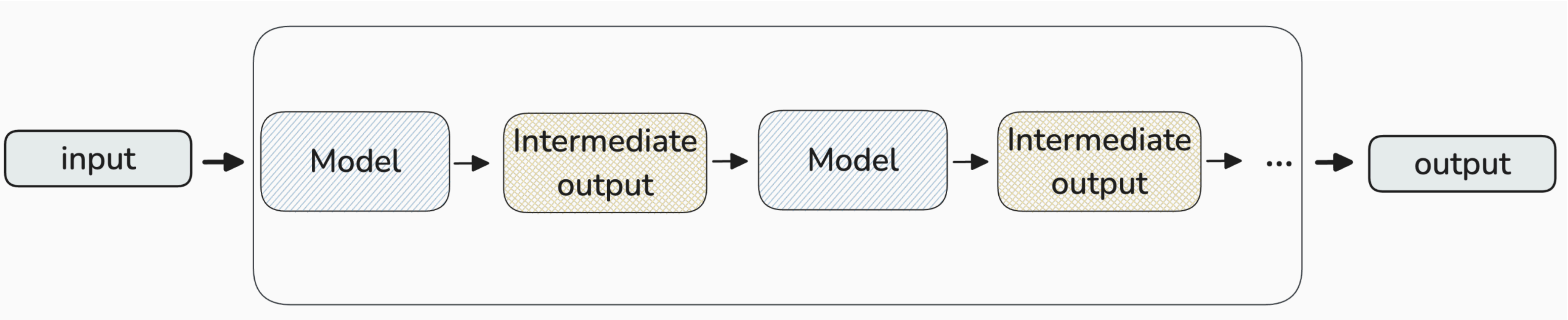
Query: What is the current tallest wooden lattice tower?
Response: Radio Tower Gliwice

Intermediate answer: Radio Tower Gliwice.
Follow up: When was Gliwice Radio Tower completed?

Query: When was Gliwice Radio Tower completed?
Response: 1935

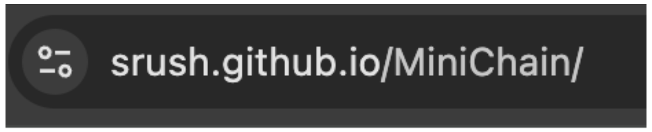
Intermediate answer: 1935.
So the final answer is: 1935.

Prompt chains



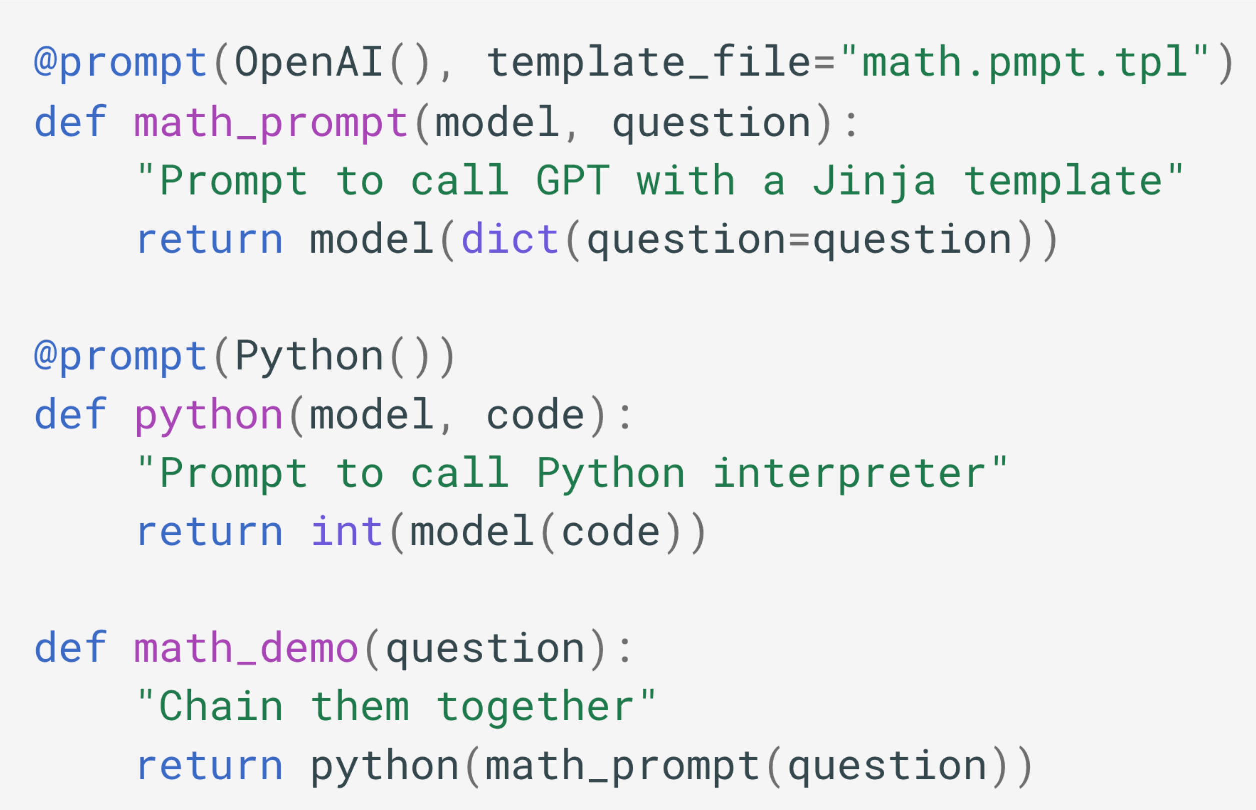
- More generally, “chain” together multiple calls to prompted models (and/or external functions)

Related software



srush.github.io/MiniChain/

Mini-Chain



```
@prompt(OpenAI(), template_file="math.pmpt.tpl")
def math_prompt(model, question):
    "Prompt to call GPT with a Jinja template"
    return model(dict(question=question))

@prompt(Python())
def python(model, code):
    "Prompt to call Python interpreter"
    return int(model(code))

def math_demo(question):
    "Chain them together"
    return python(math_prompt(question))
```

https://srush-minichain.hf.space/

Recap

- Prompting and in-context learning basics
  - No prompt
  - Zero-shot
  - Few-shot
- Prompt engineering
  - Chat and system prompts
  - Chain-of-thought
  - Prompt chaining

Thank you