IMEN574/IDSC574 Programing for Data Science

4. LLM Prompting

Spring 2026

Instructor: **Sangwoo Mo**

How LLMs Work in ~~5~~ a few Slides (For Engineers)

Basics

- LLMs (large language models) are autoregressive models for next-token prediction

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_1, x_2, \ldots, x_{t-1})$$



$$P(\text{write}, \text{a}, \text{for}, \text{loop}, \text{for}, \text{idx}, \text{in}, \text{range}\ldots) = P(\text{range}|\text{write}, \text{a}, \text{for}, \text{loop}, \text{for}, \text{idx}, \text{in}) \cdot P(\text{in}|\text{write}, \text{a}, \text{for}, \text{loop}, \text{for}, \text{idx}) \cdot \ldots \cdot P(\text{a}|\text{write}) \cdot P(\text{write})$$

Next Token Prediction – Inference

Given “many”, predict the next token

Many

apple: 0.01
don: 0.001
...
intelligence: 0.00001
...
words: 0.02

Next Token Prediction – Inference

Given “many”, predict the next token

`Many`

apple: 0.01
don: 0.001
…
intelligence: 0.00001
…
words: 0.02

Given “many words”, predict the next token

`Many words`

apple: 0.00003
don: 0.03
…
intelligence: 0.00001
…
words: 0.0000001

Next Token Prediction – Training

Original sentence

```
Many words don't map to one token: indivisible.
```

Given “many”, predict the next token

```
Many
```

apple: 0.01
don: 0.001
…
intelligence: 0.00001
…
words: 0.02

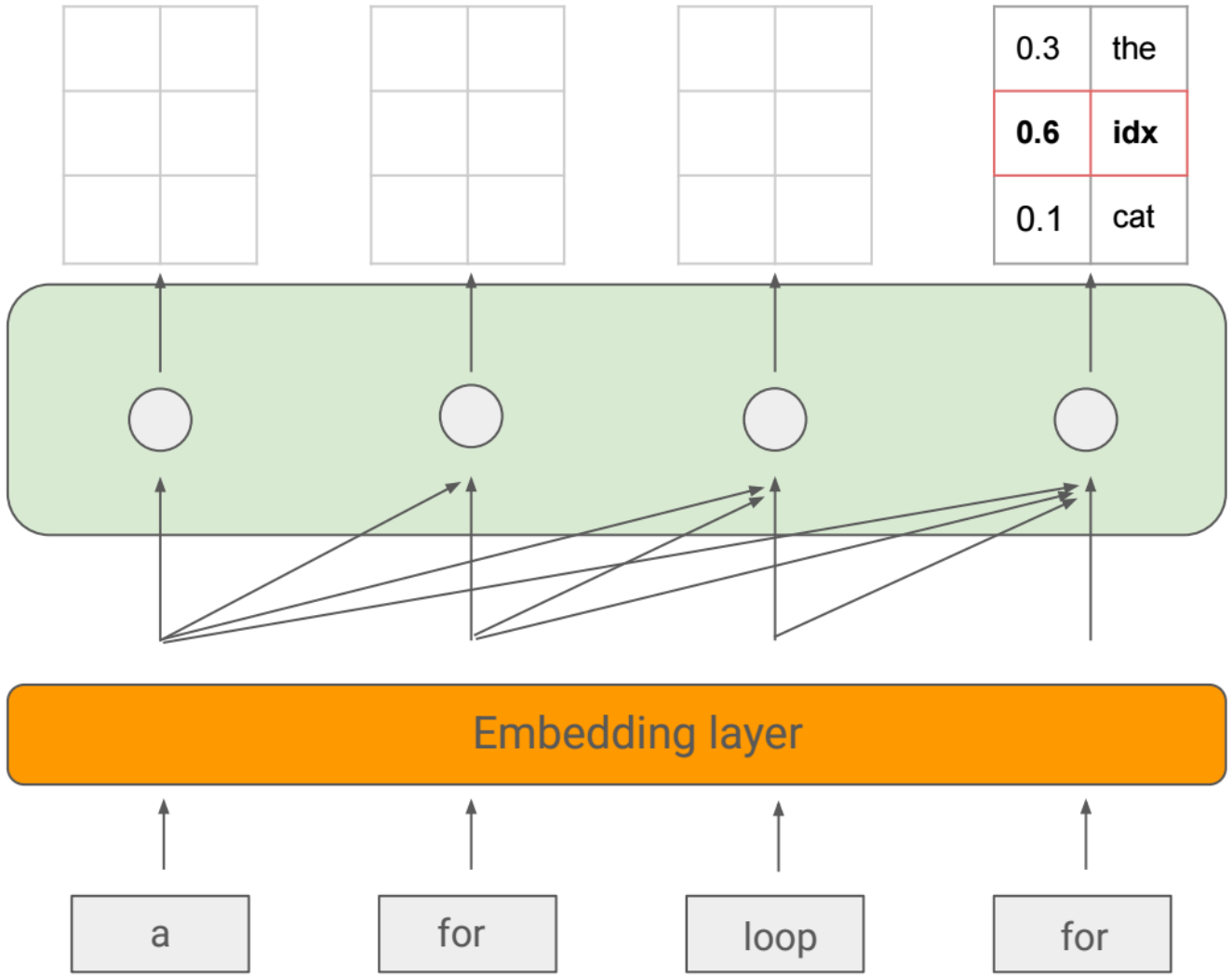
Given “many words”, predict the next token

```
Many words
```

apple: 0.00003
don: 0.03
…
intelligence: 0.00001
…
words: 0.0000001

Probability of a sentence is a product of conditional probabilities. Maximize this.

Basics



Get probability distribution over most likely next token

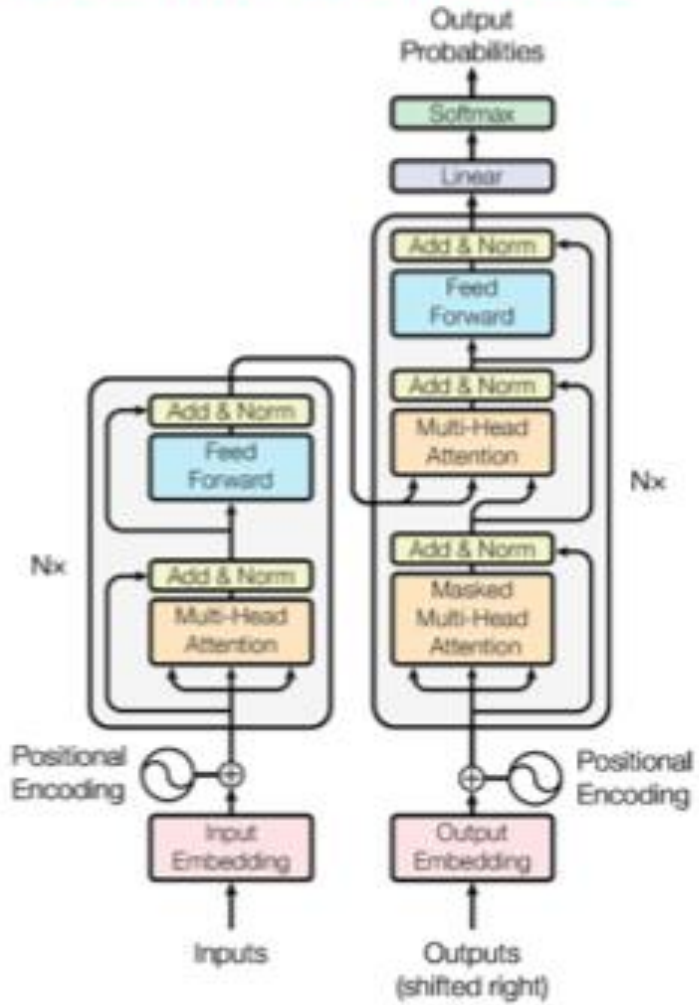
Transformers layers (12-96+) using self-attention mechanism (Viswani et. al. 2017)

Convert tokens into fixed-dimensional numerical vectors (~1-3K dimensions)

Tokenize inputs using fixed vocabulary

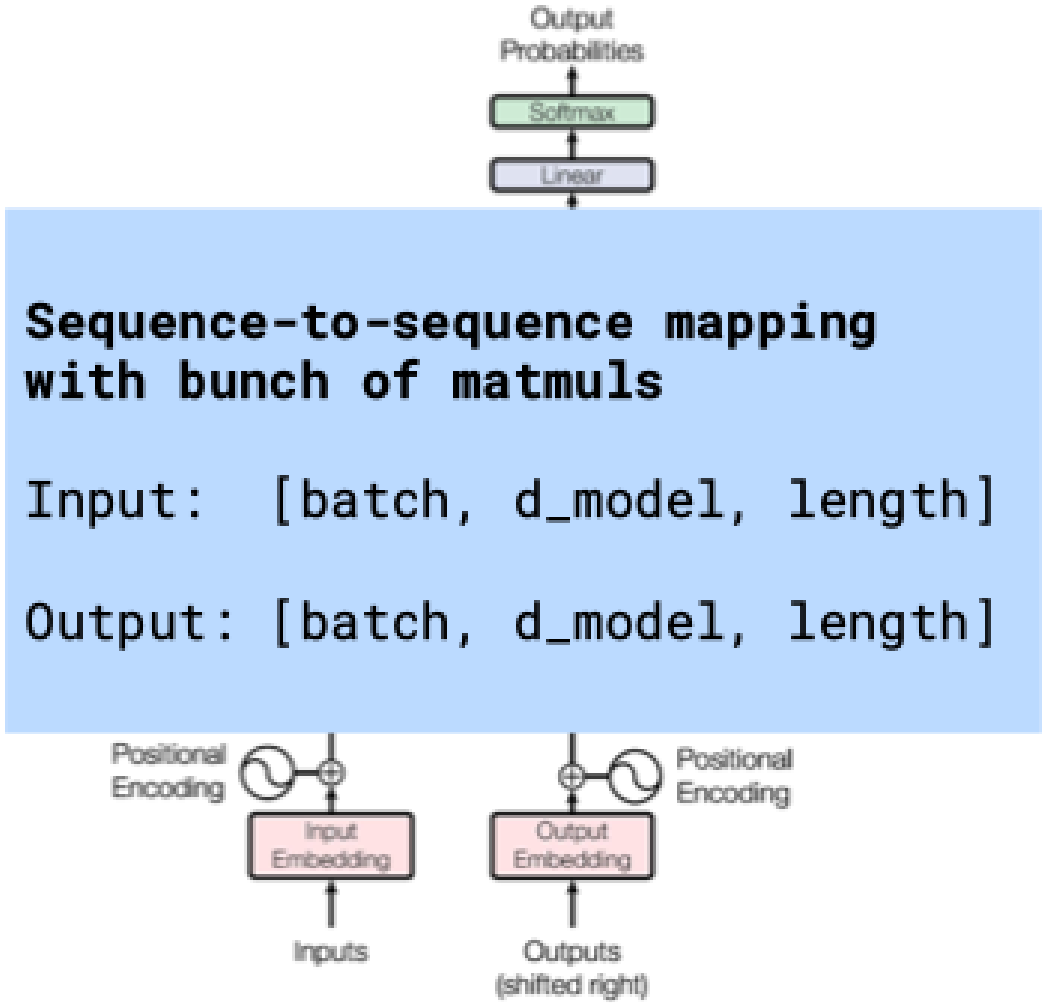
Transformer

All LLMs so far use Transformer architecture



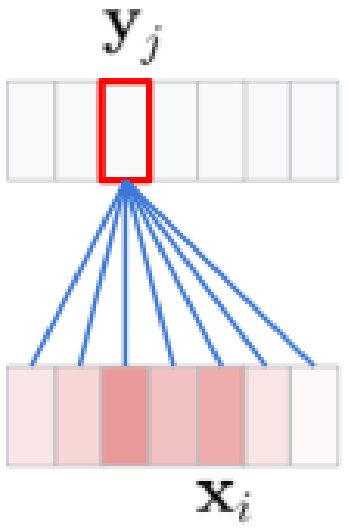
Transformer

Let's take a "functional" viewpoint on the Transformer



Transformer – Self-Attention

- Each of the tokens (=vectors) attends to all tokens
  - Extra tricks: learned key, query, and value projections, inverse-sqrt scaling in the softmax, and multi-headed attention (omit for simplicity)
- It's a set operation (permutation-invariant)
  - ...and hence need "position embeddings" to "remember" the spatial structure
- It's a global operation
  - Aggregates information from all tokens



$$\boldsymbol{\alpha}_j = softmax(\frac{K\mathbf{x}_1 \cdot Q\mathbf{x}_j}{\sqrt{d_K}}, \ldots, \frac{K\mathbf{x}_n \cdot Q\mathbf{x}_j}{\sqrt{d_K}})$$

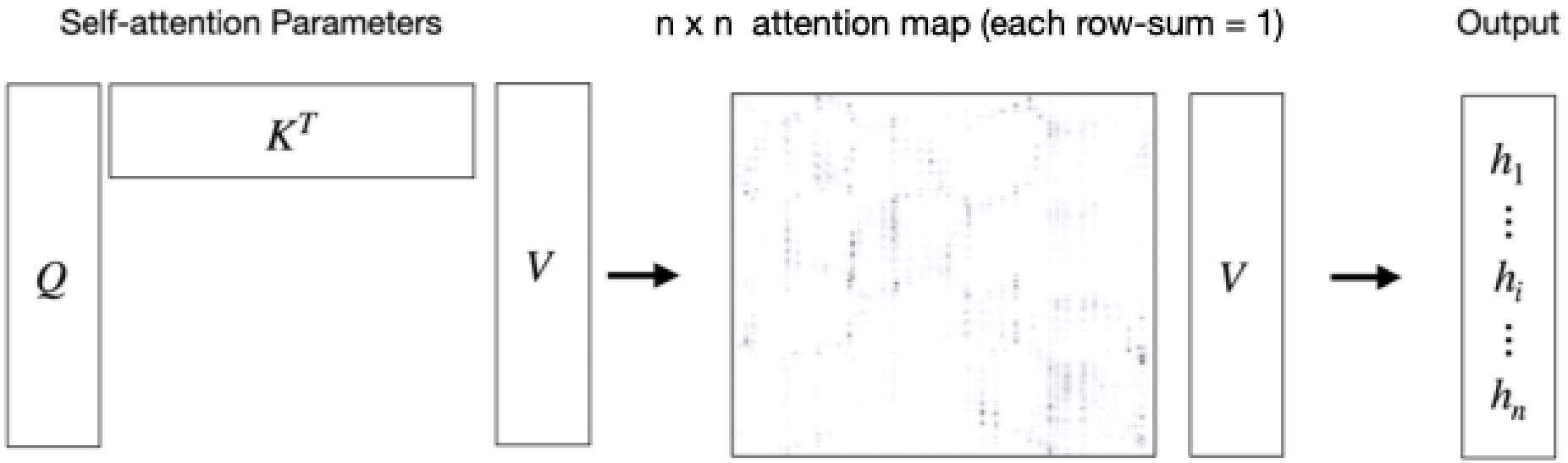
$$\mathbf{y}_j = \sum_{i=1}^{n} \alpha_{ji} V\mathbf{x}_i$$

Simplified! Multi-headed attention not shown

Vaswani et al., *Attention Is All You Need*, NeurIPS 2017

Transformer – Self-Attention

Transformer self-attention



Output matrix H = $\text{softmax}(\frac{1}{\sqrt{d}} QK^T) \cdot V$

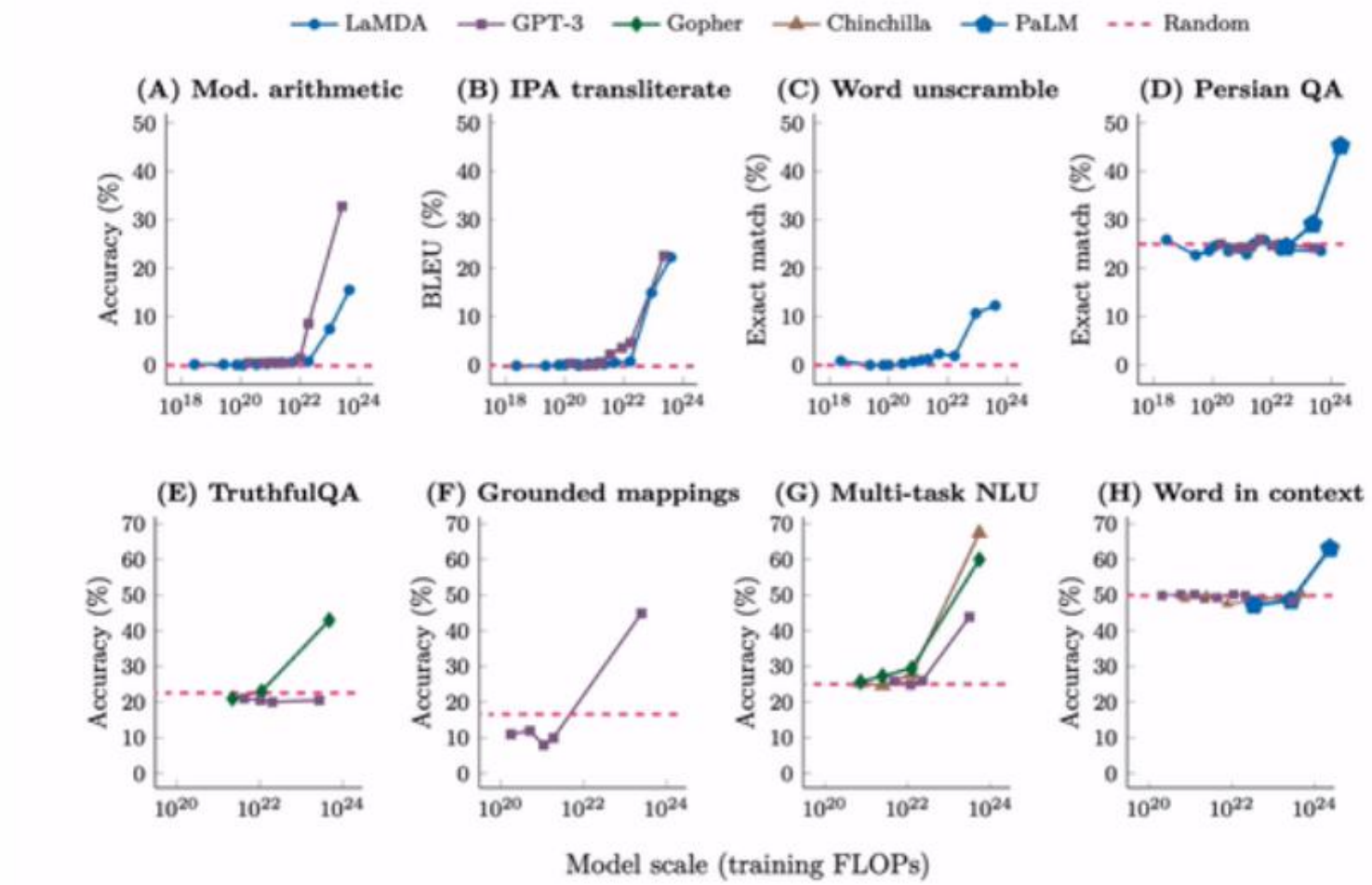
Self-attention explicitly models interactions between all pairs of input embeddings

Scaling Laws

Some abilities emerge with scale

Having the right perspective is crucial

Scaling Laws



Emergent Abilities of Large Language Models
Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph et al. (2022)

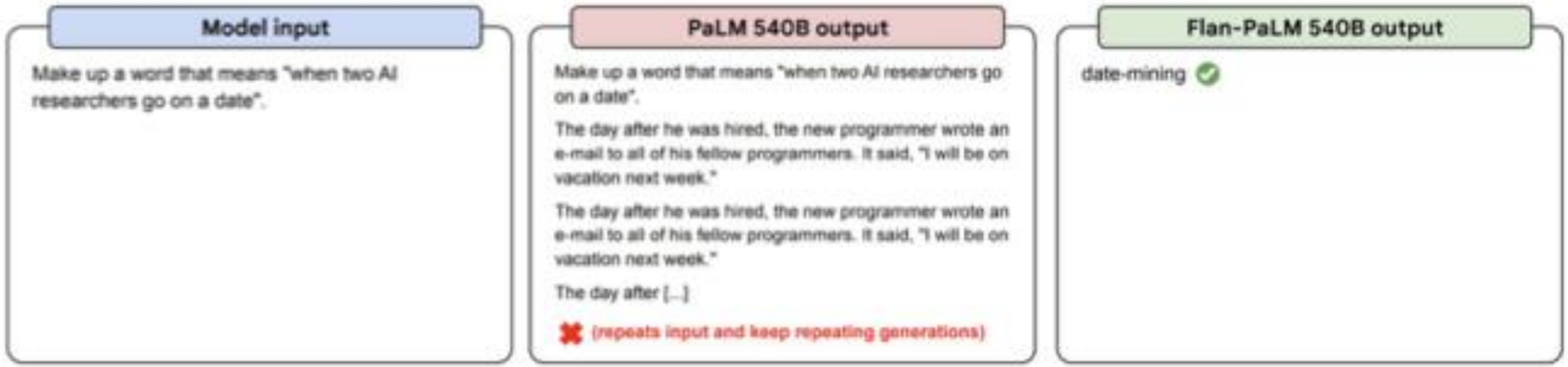
Next Token Prediction is Not Enough

Scaling doesn’t solve all problems

We also need post-training

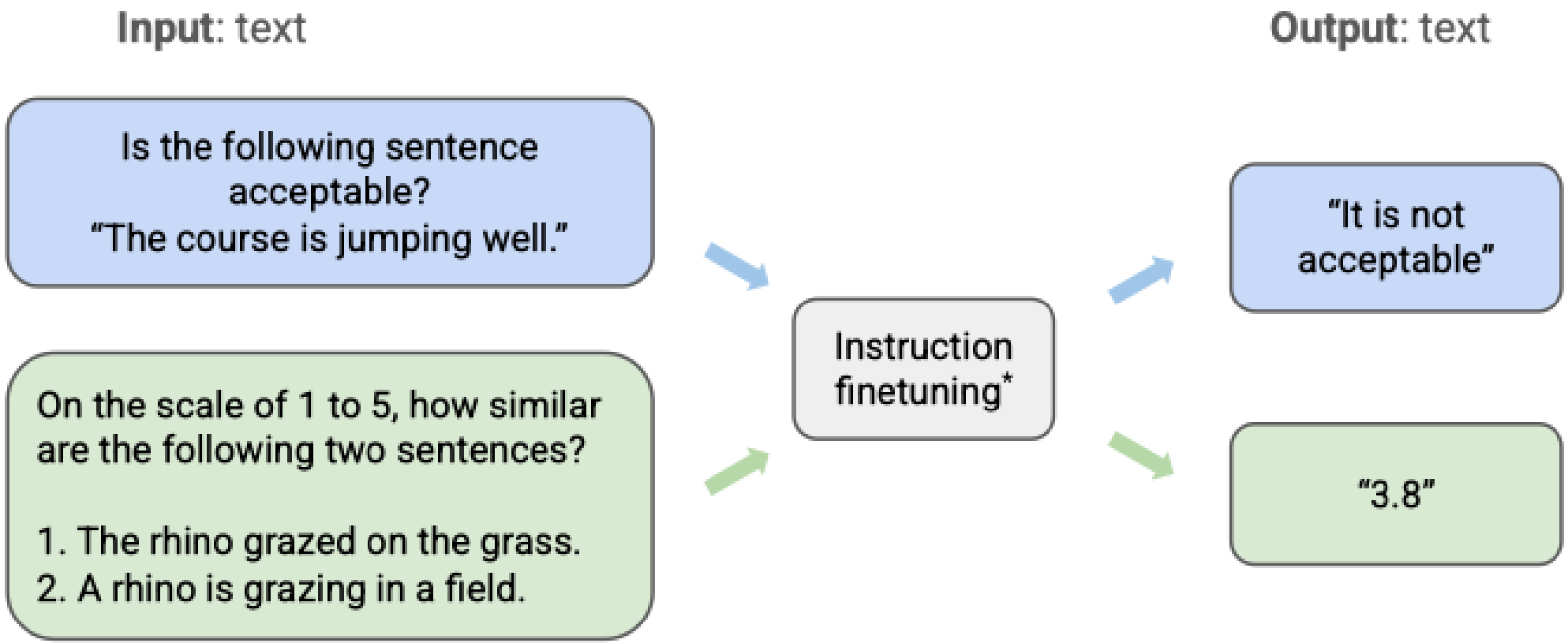
Next Token Prediction is Not Enough

We can't talk to the pretrained model directly



Scaling Instruction-Finetuned Language Models
Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus et al. (2022)

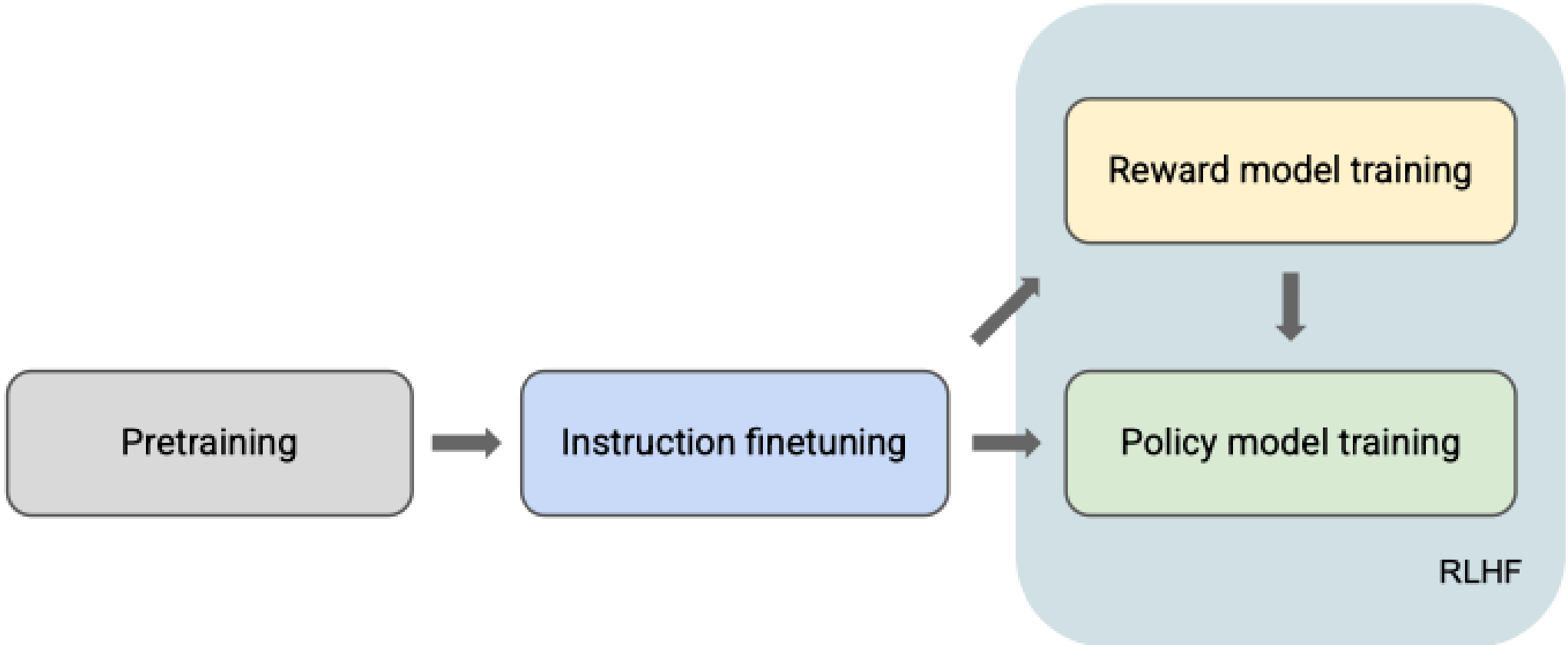
Instruction Fine-tuning



Tasks are unified. So for an unseen task, the model just needs to respond to the natural language instruction

*Wei et al. (2021), Sanh et al. (2021), Ouyang et al. (2022)

Training Process



Training Process

- Stage 1
  - Self-supervised pretraining
  - Teach the model notion of language on a variety of often public data sources
  - 100s of billions to trillion+ tokens (language and code)
  - Common Crawl, Wikipedia, StackExchange, Public Github repos
  - *Write a for loop* → **that could be used in a piece of code**
- Stage 2
  - Supervised finetuning
  - Teach model to follow instructions
  - High-quality, curated prompt-response pairs (“*what is the capital of Croatia*” -> “*Zagreb is the capital*”)
  - Tens of thousands to 100s of thousands of pairs
  - *Write a for loop* → **ok here’s a for loop…**
- Stage 3
  - Preferencing tuning
  - Align model outputs with human preferences (helpfulness, correctness, readability)
  - Collect pairs of outputs for same prompt and train reward model to predict preferred output
  - Tens of thousands to 100s of thousands of human-labeled comparisons
  - *Write a for loop* → **for idx in range(10):**

Training Process

- Reasoning models
  - Extend training with chain-of-thought reasoning traces
  - Tool-use integration
  - Get human preferences on reasoning steps
  - Reinforcement learning to learn how to evaluate reasoning traces, backtrack, etc
- Size
  - GPT-3/Claude 3.5 Sonnet - 175B parameters
  - LLaMA 3.1 - 405B parameters
  - GPT-4 - 1.8T (reported)

In practice

- Strengths
  - Expert-level code completion
  - Code understanding
  - Code fixing
- Limitations
  - Hallucinations
    - Generating non-existent/out-of-date APIs (mitigated with robust context engineering)
  - Context window limits
    - ~100-200K tokens but not all are created equal
  - Latency
    - Seconds to minutes per request depending on task (plan and delegate accordingly)
  - Cost
    - $1-3 per million input tokens, $10+ per million output tokens for best models

LLM Power Prompting

Prompting background

- Prompts are the *lingua franca* for getting LLMs to do what we want and also effectively programming them

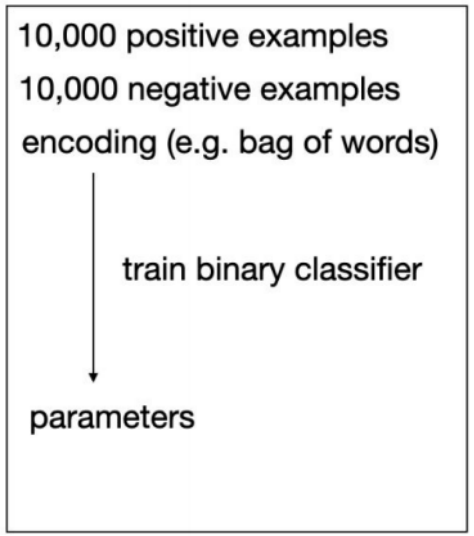
Software 1.0

```python
def simple_sentiment(review: str) -> str:
    """Return 'positive' or 'negative' based on a tiny keyword lexicon."""
    positive = {
        "good", "great", "excellent", "amazing", "wonderful", "fantastic",
        "awesome", "loved", "love", "like", "enjoyed", "superb", "delightful"
    }
    negative = {
        "bad", "terrible", "awful", "poor", "boring", "hate", "hated",
        "dislike", "worst", "dull", "disappointing", "mediocre"
    }

    score = 0
    for word in review.lower().split():
        w = word.strip(".,!?;:")          # crude token clean-up
        if w in positive:
            score += 1
        elif w in negative:
            score -= 1

    return "positive" if score >= 0 else "negative"
```

Software 2.0



Software 3.0

You are a sentiment classifier. For every review that appears between the tags
<REVIEW> ... </REVIEW>, respond with **exactly one word**, either POSITIVE or NEGATIVE (all-caps, no punctuation, no extra text).

Example 1
<REVIEW>I absolutely loved this film—the characters were engaging and the ending was perfect.</REVIEW>
POSITIVE

Example 2
<REVIEW>The plot was incoherent and the acting felt forced; I regret watching it.</REVIEW>
NEGATIVE

Example 3
<REVIEW>An energetic soundtrack and solid visuals almost save it, but the story drags and the jokes fall flat.</REVIEW>
NEGATIVE

Now classify the next review.

Courtesy of [Andrej Karpathy]

Prompting Background

- Prompting is both an art and a science
- Black-box nature of LLMs means there's some magic to *LLM-whispering* effectively
- …but there are established techniques that have empirically improved LLM performance

Zero-shot prompting

- Ask the LLM to do a thing
  - no support
  - no examples

Write me a Rust for-loop that iterates over a list of strings for every, printing every value in an even index

K-shot prompting

- Ask the LLM to do the thing but give it some examples of how to do it
  - Sometimes called *in-context learning*
- k-shot
  - 1, 3, 5 (some empirical results justify these numbers)
- Ideal for tasks that don’t have too many reasoning steps

Write a for-loop iterating over a list of strings using the naming convention in our repo.



Write a for-loop iterating over a list of strings using the naming convention in our repo. Here are some examples of how we typically format variable names.

<example>
var StRaRrAy = [‘cat’, ‘dog’, ‘wombat’]
</example>

<example>
def func CaPiTaLiZeStR = () => {}
</example>

Chain-of-Thought Prompting

- Show reasoning steps for a given task
  - Multi-shot CoT
    - Work out reasoning traces
  - Zero-shot CoT
    - “Let’s think step-by-step”
  - Prompt for reasoning in explicit <reasoning> tags
- Best for steps with multiple logical steps
  - Programming
  - Math

Multi-shot CoT

Write a function to check if a number is a perfect cube and a perfect square.

I want to write a function to check if a number is a perfect cube and a perfect square. Make sure to provide your reasoning first. Here are some examples of how to provide reasoning for a coding task.

<example>
Write a function that finds the maximum element in a list.
Steps: Initialize a variable with the first element. Traverse the list, comparing…
</example>

<example>
Write a function that checks is a number is a palindrome
Steps: Take the number. Reverse the elements in the numbers. Check if …
</example>

Zero-shot CoT

Write a function to find the longest increasing subsequence in an array.

Write a function to find the longest increasing subsequence in an array.

Think step by step about the subproblems before coding. Include worked out examples of subarrays you are considering as you answer this question.

Self-consistency Prompting

- Sample multiple times from output (typically with CoT) and aggregate most common results
- Reduces hallucinations/incorrect answers via a form of model ensembling by sampling diverse reasoning paths

What's the root cause for this error:
Traceback (most recent call last):
File "example.py", line 3, in <module>
print(nums[i])
IndexError: list index out of range

What's the root cause for this error:
Traceback (most recent call last):
File "example.py", line 3, in <module>
print(nums[i])
IndexError: list index out of range

Prompt 5x

Take majority result

Tool Use

- Allow LLM to defer to an external system that it needs to interact with
- One of the most important techniques for reducing hallucinations and enabling the autonomy of LLMs

After you have fixed this IndexError can you ensure that the CI tests still pass?

Fix the IndexError. Ensure the CI tests still pass once you have made the fix. Here are the available tools.

<tools>
pytest -s /path/to/unit_tests

pytest -v /path/to/integration_tests
</tools>

Retrieval Augmented Generation

- Infuse the LLM with contextual data
- Keeps LLMs up-to-date (without retraining)
  - Faster iteration
- You get interpretability and citations for free
- Reduces hallucinations

Extend the UserAuthService class to check that the client provides a valid OAuth token.

I want to extend the UserAuthService class to check that the client provides a valid OAuth token.

Here is how the UserAuthService works now:

<code_snippet>
def issue_oauth_token():
….
</code_snippet>

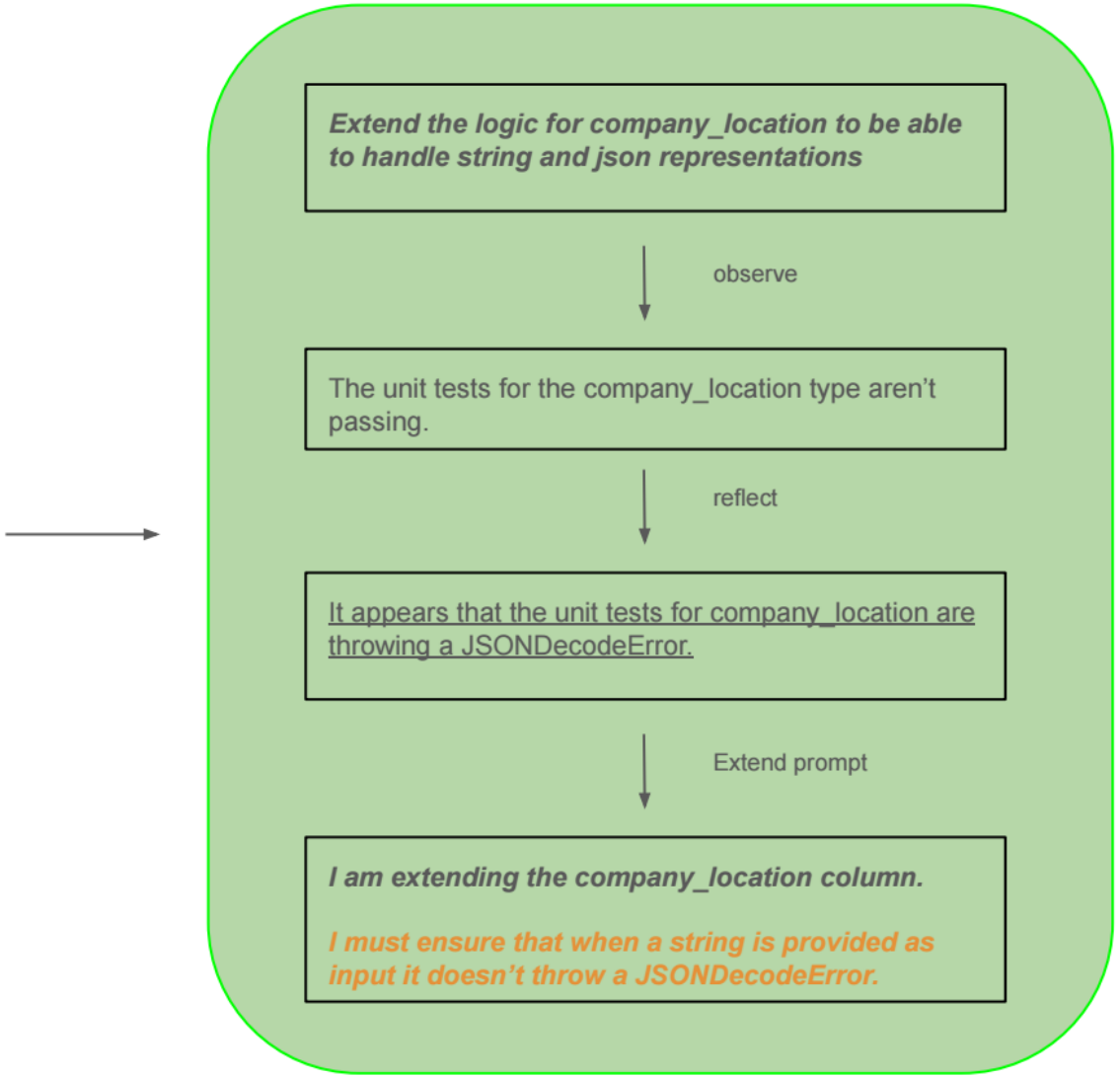
Here is the path to the requests-oauthlib documentation:

<url>
https://requests-oauthlib.readthedocs.io/en/latest/
</url>

Reflexion

- Make the LLM *reflect* on its output
- Verbal feedback from environment signals are reincorporated into the LLM by augmenting the context
- After an action is taken in an environment and an observation made, add a prompt suffix:
  - “Now critique your answer. Was it correct? If not, explain why and try again.”
- Multi-turn prompting
  - Turn 1: Model gives a first try.
  - Turn 2: You ask “*Was that correct? Reflect and revise if needed.*”

Ensure that the company_location column can handle string and json representations.



Additional Terminology

- System prompt
  - First message provided to LLM (usually not seen by end user)
  - Provides persona, rules about LLM output, style

The assistant is Claude, created by Anthropic.

The current date is {{currentDateTime}}.

Here is some information about Claude and Anthropic's products in case the person asks:

This iteration of Claude is Claude Opus 4.1 from the Claude 4 model family. The Claude 4 family currently consists of Claude Opus 4.1, Claude Opus 4, and Claude Sonnet 4. Claude Opus 4.1 is the most powerful model for complex challenges.

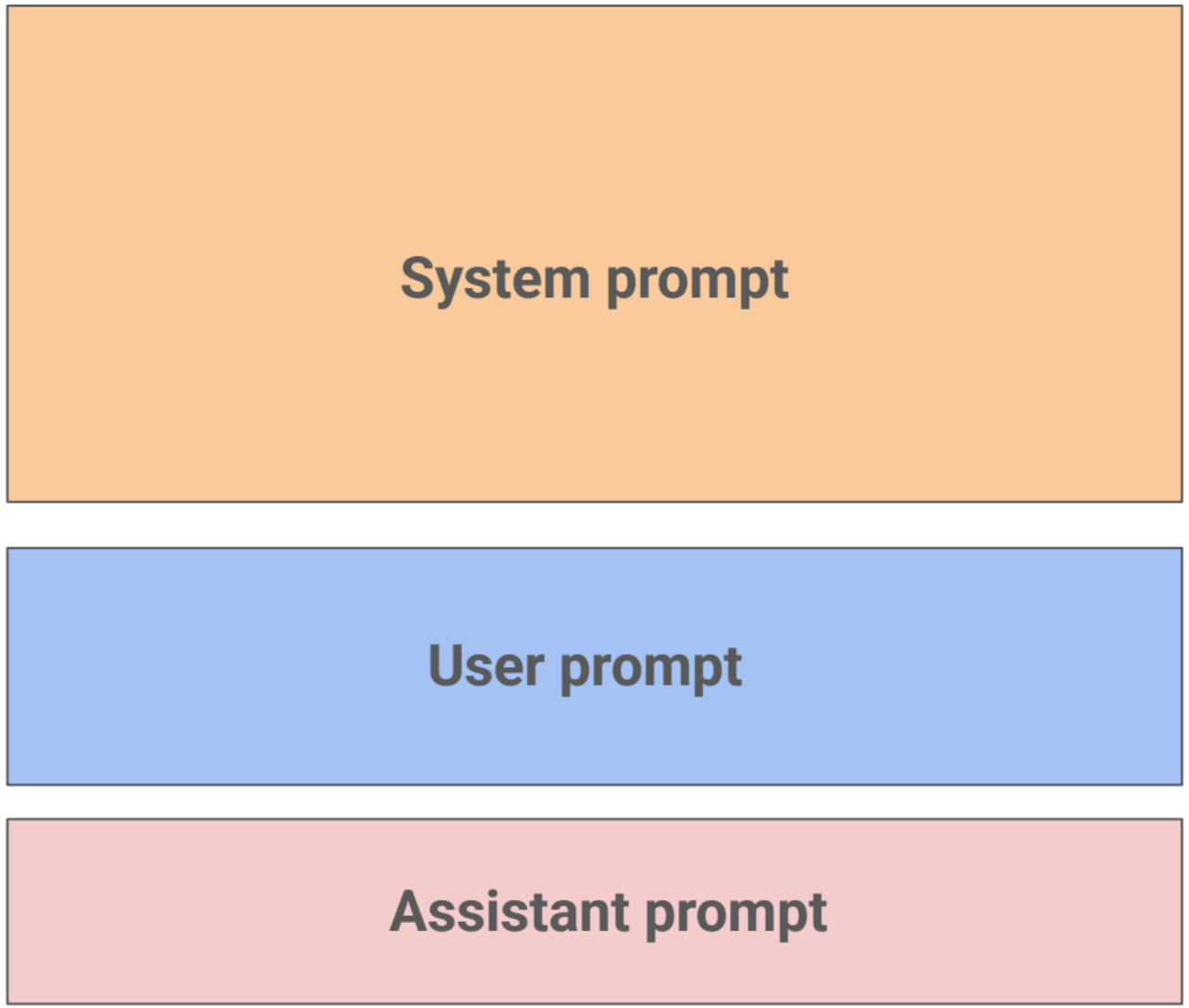
•••

Claude provides emotional support alongside accurate medical or psychological information or terminology where relevant.

Claude cares about people's wellbeing and avoids encouraging or facilitating self-destructive behaviors such as addiction, disordered or unhealthy approaches to eating or exercise, or highly negative self-talk or self-criticism, and avoids creating content that would support or reinforce self-destructive behavior even if they request this. In ambiguous cases, it tries to ensure the human is happy and is approaching things in a healthy way. Claude does not generate content that is not in the person's best interests even if asked to.

Additional Terminology

- User prompt
  - The actual ask or instruction from a human
  - Basically all of our examples so far
- Assistant
  - What the LLM actually generates


System prompt
User prompt
Assistant prompt

Best Practices

- Prompt improvement
  - https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/prompt-improver
- Clear prompting
  - Give prompt to someone with minimal context and if they're confused an LLM will be too
- Use role prompting aggressively to make system prompts more powerful

You are a helpful assistant that loves programming at the level of a senior software developer and is very detailed and pedantic in your answers.

python

Copy code

```python
def is_palindrome(s: str) -> bool:
    """
    Check if a given string is a palindrome.

    A palindrome is a sequence that reads the same backward as forward.
    This function normalizes the string by:
    1. Removing all non-alphanumeric characters.
    2. Converting to lowercase for case-insensitive comparison.

    Args:
        s (str): Input string.

    Returns:
        bool: True if `s` is a palindrome, False otherwise.
    """
    import re

    # Normalize the input: remove non-alphanumeric characters
    cleaned = re.sub(r'[^A-Za-z0-9]', '', s).lower()

    # Compare cleaned string with its reverse
    return cleaned == cleaned[::-1]
```

*"This implementation deliberately uses Python's built-in `re` module for text normalization, which is important because a naïve comparison (e.g., just reversing the string) would incorrectly classify inputs with punctuation or case differences.

Best Practices

- Prompts should be formatted with structure

Here are the logs:
<log>
LOG MESSAGE
<log> and the stack trace:
<error>
STACK TRACE
<error>

Best Practices

- Be explicit about what you want (languages, tech stacks, libraries, constraints)
- Decompose tasks