AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Prompt Engineering

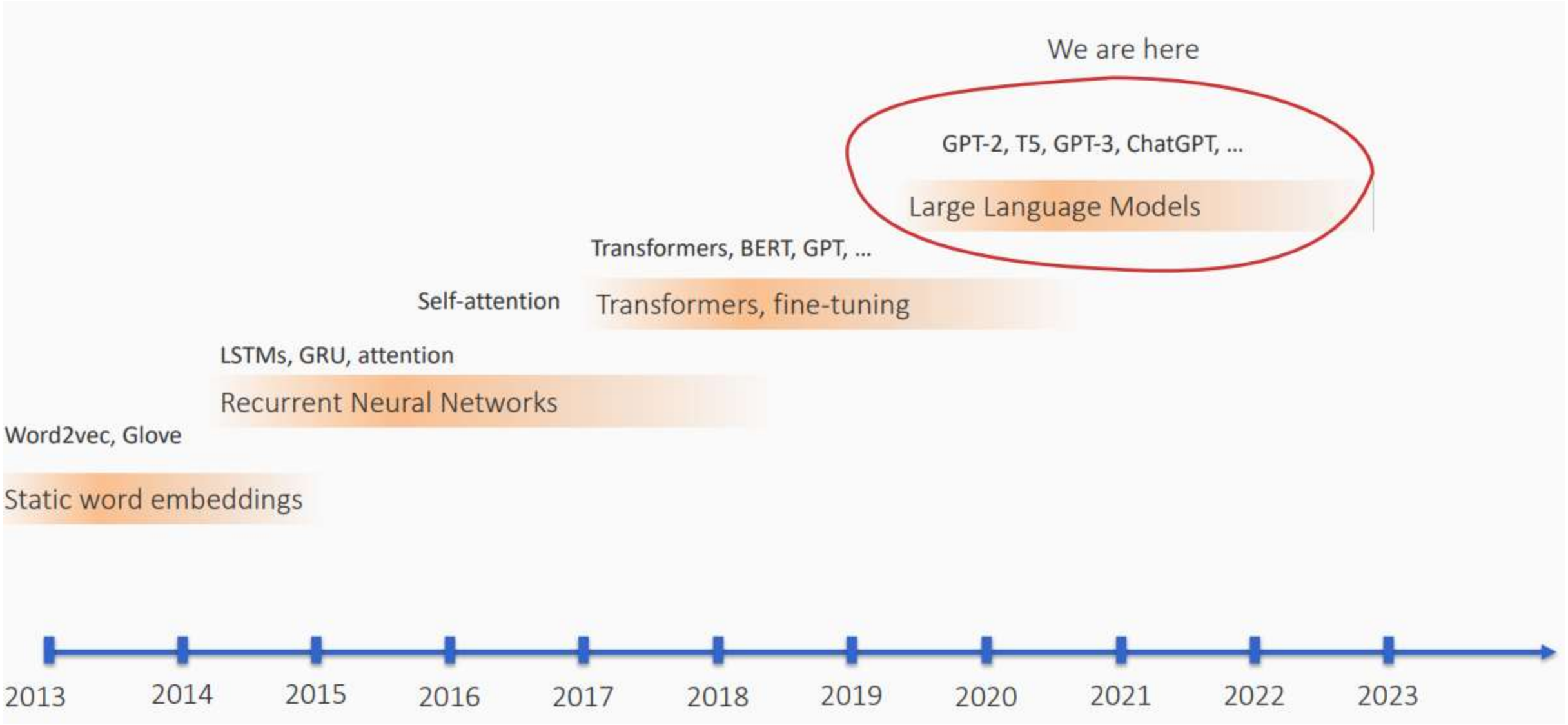
Outline

1. Fundamentals of Prompt Engineering
2. Shot-Based Prompting-Zero-Shot, One-Shot, Few-Shot
3. Reasoning-Based Prompting
    - Chain-of-Thought (CoT) Prompting
    - One-Shot CoT
    - Few-Shot CoT
    - Self-Consistency CoT
    - Tree of Thoughts (ToT)
4. Interaction and Feedback-Based Prompting
    - ReAct Prompting (Reasoning + Acting)
    - Positive and Negative Prompting
    - Iterative Prompting
    - Model-Guided Prompting
5. Instruction and Role-Based Prompting
    - Instruction Prompting
    - Role Prompting

5. Knowledge-Augmented Prompting
    - Generated Knowledge Prompting (GKP)

Fundamentals of Prompt Engineering

Where are we?


We are here
GPT-2, T5, GPT-3, ChatGPT, ...
Large Language Models
Transformers, BERT, GPT, ...
Self-attention
Transformers, fine-tuning
LSTMs, GRU, attention
Recurrent Neural Networks
Word2vec, Glove
Static word embeddings
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023

“pre-train, prompt, and predict”.

In Paradigm : “pre-train, prompt, and predict”. In this paradigm, instead of adapting pre-trained LMs to downstream tasks via objective engineering, **downstream tasks are reformulated to look more like those solved during the original LM training with the help of a textual prompt..**
In this way, by selecting the appropriate prompts we can manipulate the model behavior so that the pre-trained LM itself can be used to predict the desired output, sometimes even without any additional task-specific training

For example, when recognizing the emotion of a social media post,

“I missed the bus today.”, we may continue with a prompt “I felt so ”, and ask the LM to fill the blank with an emotion-bearing word. Or

if we choose the prompt “English: I missed the bus today. French: ”), an LM may be able to fill in the blank with a French translation

The advantage of this method is that, given a suite of appropriate prompts, a single LM trained in an entirely unsupervised fashion can be used to solve a great number of tasks (Brown et al., 2020; Sun et al., 2021).
However, as with most conceptually enticing prospects, there is a catch – **this method introduces the necessity for prompt engineering, finding the most appropriate prompt to allow a LM to solve the task at hand.**

What are prompts?

- **Prompts** involve instructions and context passed to a language model to achieve a desired task
- **Prompt engineering** is the practice of developing and optimizing prompts to efficiently use language models (LMs) for a variety of applications
  - Prompt engineering is a useful skill for AI engineers and researchers to improve and efficiently use language models

```
What is prompt engineering?

Prompt engineering is a process of creating a set of prompts,
or questions, that are used to guide the user toward a desired
outcome. It is an effective tool for designers to create user
experiences that are easy to use and intuitive. This method is
often used in interactive design and software development, as
it allows users to easily understand how to interact with a
system or product..
```

simple examples of prompts may be:

- *"Write an email asking for 3 days' leave from my manager."*
- *"Create a line drawing of a koala riding a bike on Mars."*

What is prompt Engineering?

Prompt engineering is the practice of guiding large language model (LLM) outputs by providing the model ,context on the type of information to generate.

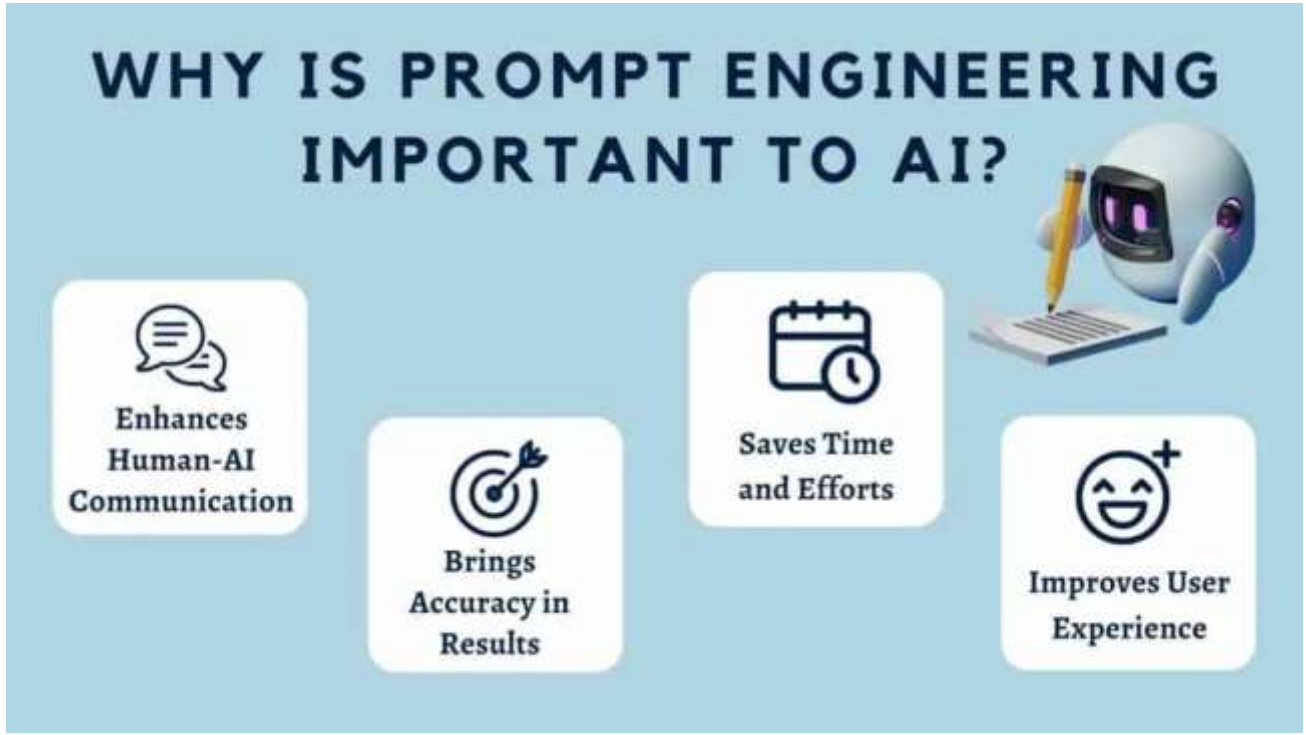
Depending on the LLM, prompts can take the form of text, images, or even audio. Embedding refers to the process of encoding any kind of information into numerical format by representing key features of the input as a numerical vector. The LLM can then perform mathematical operations on these embeddings to generate a desired output. Once a user inputs a prompt, it is embedded and then sent to the model, acting as an instruction set for how the model should generate its output.

You can achieve a lot with simple prompts, but the quality of results depends on how much information you provide it and how well-crafted the prompt is.

A prompt can contain information like the *instruction* or *question* you are passing to the model and
include other details such as *context*, *inputs*, or *examples*.

You can use these elements to instruct the model more effectively to improve the quality of results.

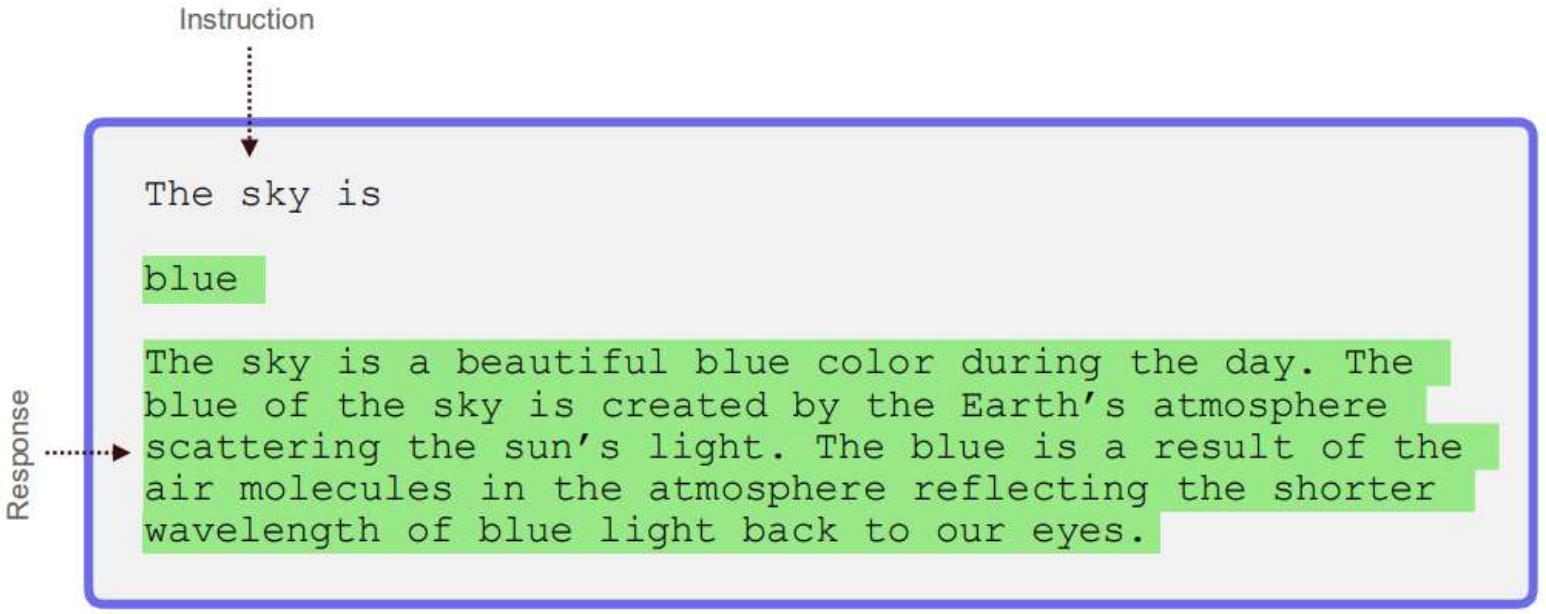
Why?


WHY IS PROMPT ENGINEERING IMPORTANT TO AI?
Enhances Human-AI Communication
Brings Accuracy in Results
Saves Time and Efforts
Improves User Experience

How?


STEPS INVOLVED IN PROMPT ENGINEERING
STEP 01
Defining the Task
STEP 02
Identifying the Inputs
STEP 03
Creating Detailed Prompts
STEP 04
Refining the Prompts

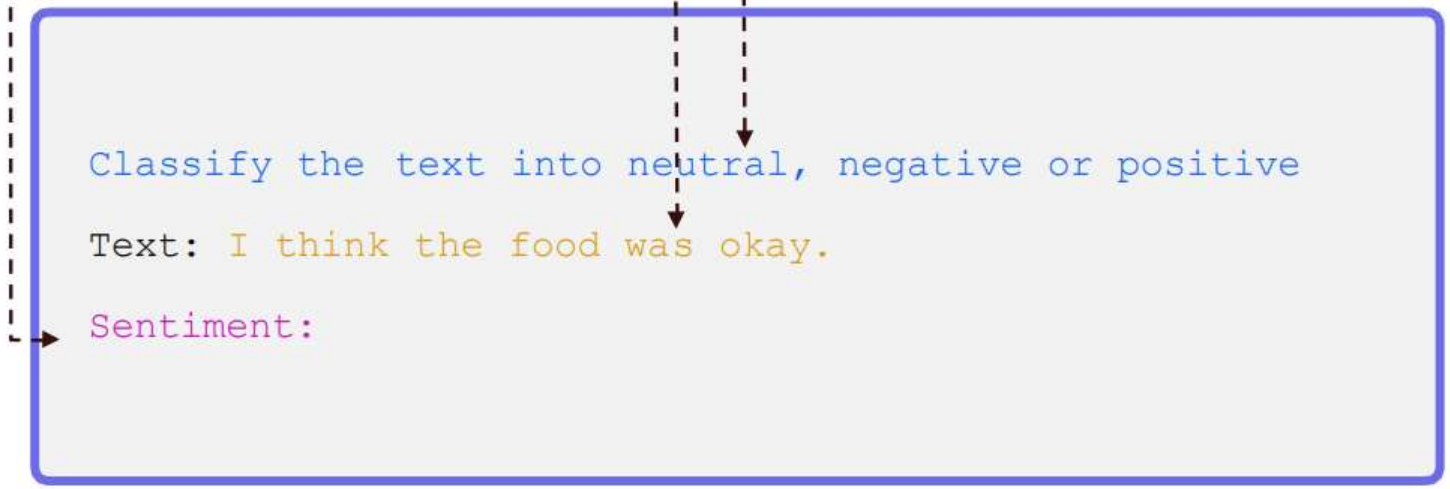
First Basic Prompt



Model: "text-davinci-003"
temperature: 0.7
top-p: 1

Elements of a Prompt

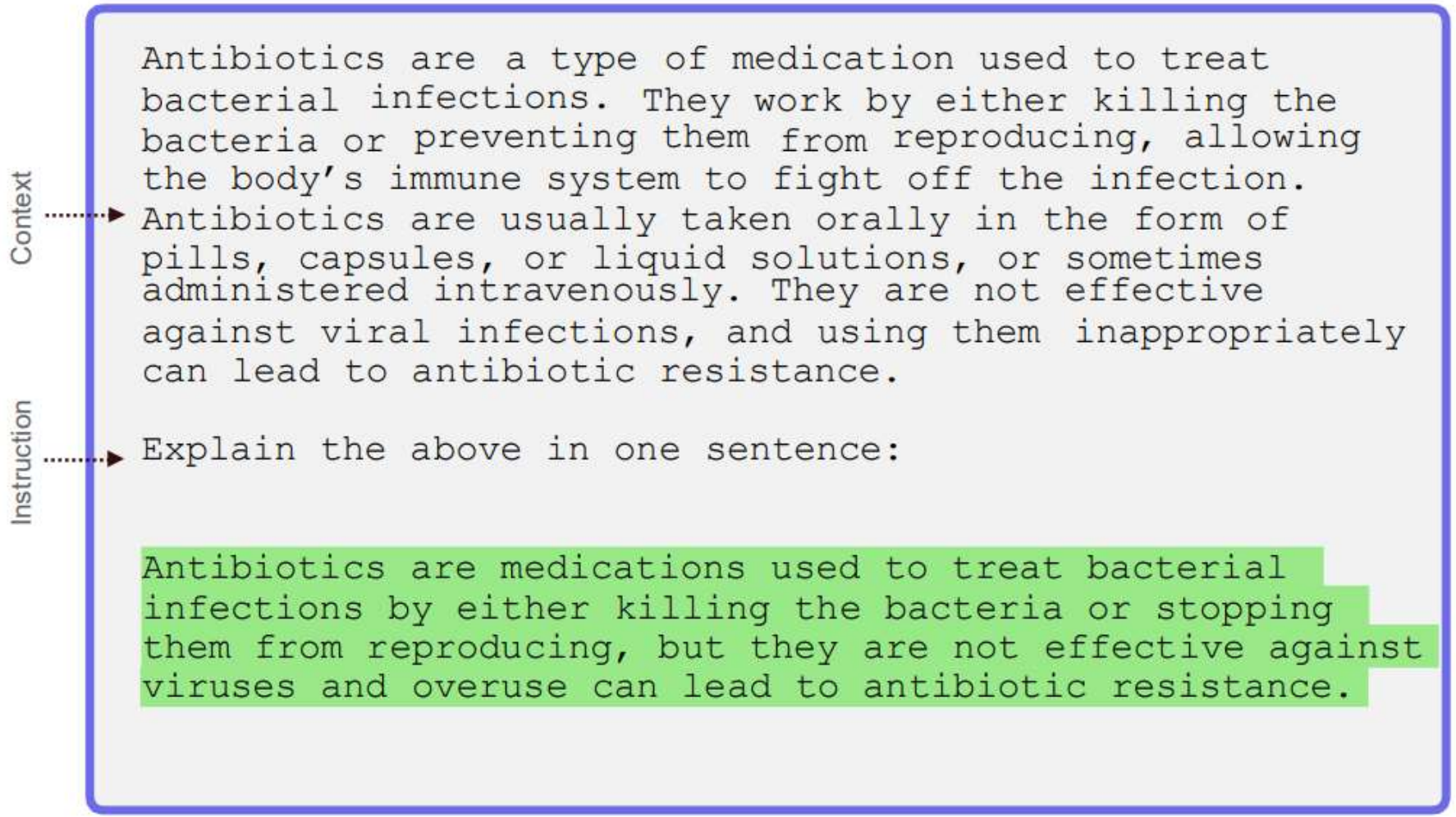
- A prompt is composed with the following components:
  - Instructions — guides the task
  - Context — background information
  - Input data — specific content to be processed
  - Output indicator — how the result should be presented



Designing Prompts for Different Tasks

- Text Summarization
- Question Answering
- Text Classification
- Role Playing
- Code Generation
- Reasoning

Text Summarization



Question Answering

```
Answer the question based on the context below. Keep the
answer short and concise. Respond "Unsure about answer"
if not sure about the answer.

Context: Teplizumab traces its roots to a New Jersey drug
company called Ortho Pharmaceutical. There, scientists
generated an early version of the antibody, dubbed OKT3.
Originally sourced from mice, the molecule was able to
bind to the surface of T cells and limit their cell-
killing potential. In 1986, it was approved to help
prevent organ rejection after kidney transplants, making
it the first therapeutic antibody allowed for human use.

Question: What was OKT3 originally sourced from?

Answer: Mice.
```

Context obtained from: https://www.nature.com/articles/d41586-023-00400-x

Text Classification

```
Classify the text into neutral, negative or positive.

Text: I think the food was okay.

Sentiment: Neutral
```

Code Generation

```
"""
Table departments, columns = [DepartmentId,
DepartmentName]
Table students, columns = [DepartmentId, StudentId,
StudentName]
Create a MySQL query for all students in the Computer
Science Department
"""

SELECT StudentId, StudentName
FROM students
WHERE DepartmentId IN (SELECT DepartmentId FROM
departments WHERE DepartmentName = 'Computer Science');
```

Reasoning

```
The odd numbers in this group add up to an even number: 15,
32, 5, 13, 82, 7, 1.

Solve by breaking the problem into steps. First, identify
the odd numbers, add them, and indicate whether the result
is odd or even.

Odd numbers: 15, 5, 13, 7, 1
Sum: 41
41 is an odd number.
```

In-Context Learning (ICL)

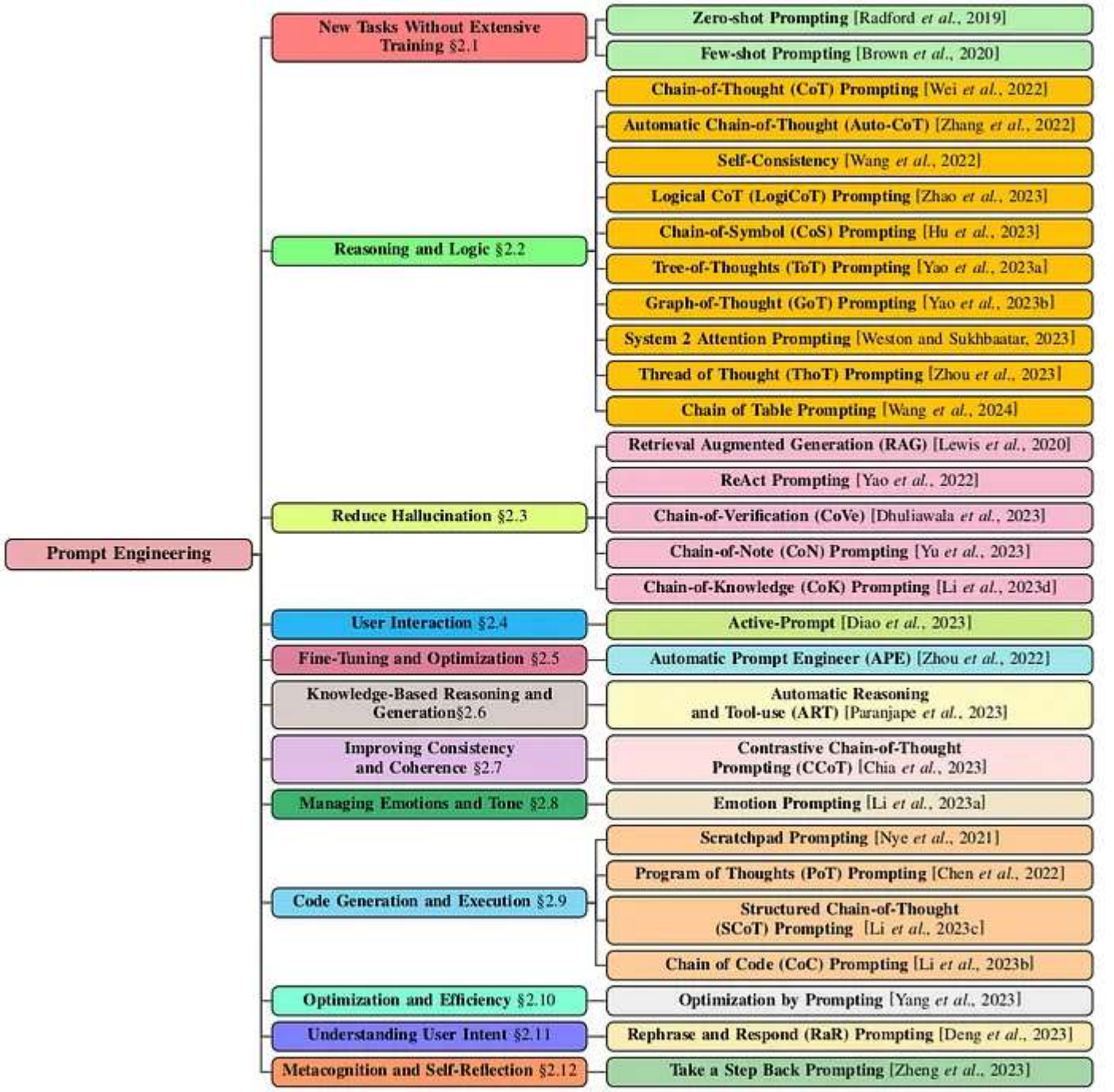
Allows models to learn from examples **within the prompt**—no extra training or fine-tuning needed.

Examples guide the AI to **understand the task and output style** by leveraging pattern recognition.

Useful when **instructions alone aren't enough**, or when a specific format/style is required.

Example-driven prompts help the model generalize to new, similar inputs.

Prompt engineering techniques in LLMs

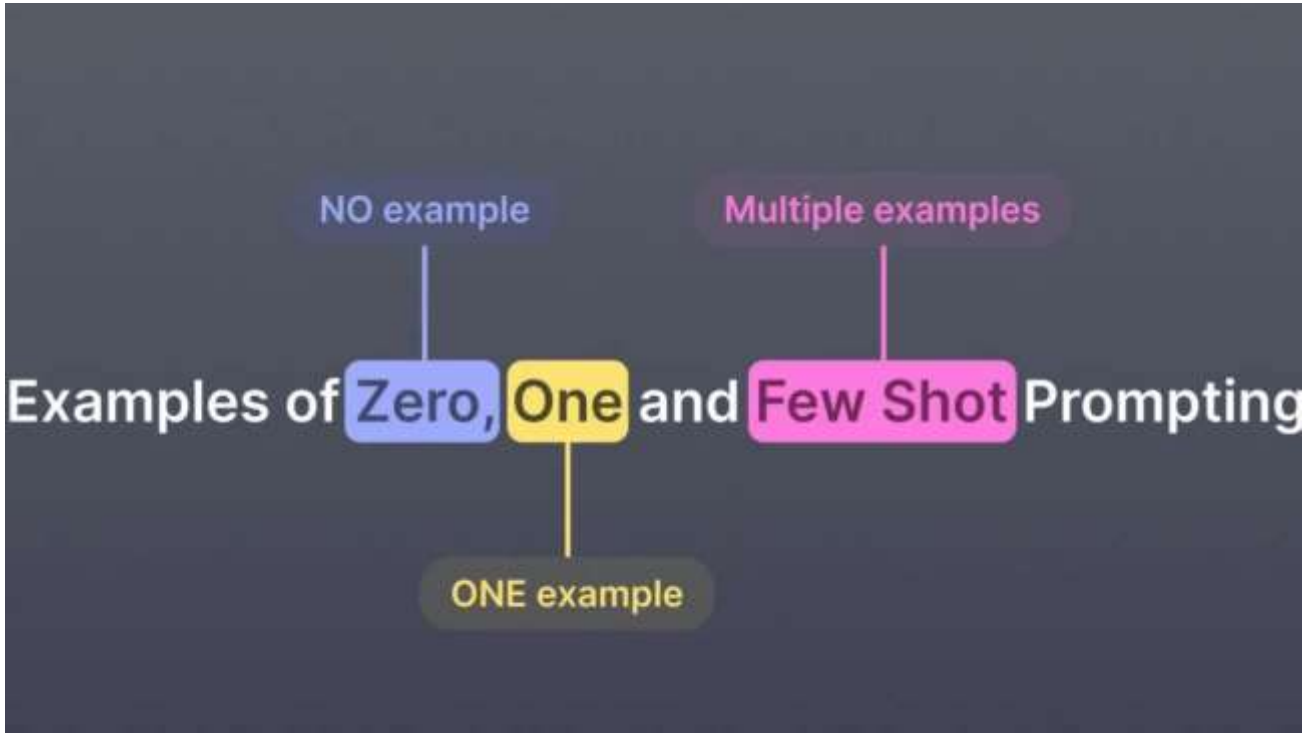


Types of Prompting

1. **Shot-Based Prompting-Zero-Shot, One-Shot, Few-Shot**
2. **Reasoning-Based Prompting**
   - **Chain-of-Thought (CoT) Prompting**
   - **One-Shot CoT**
   - **Few-Shot CoT**
   - **Self-Consistency CoT**
   - **Tree of Thoughts (ToT)**
3. **Instruction and Role-Based Prompting**
   - **Instruction Prompting**
   - **Role Prompting**
4. **Interaction and Feedback-Based Prompting**
   - **ReAct Prompting (Reasoning + Acting)**
   - **Positive and Negative Prompting**
   - **Iterative Prompting**
   - **Model-Guided Prompting**

1. Shot-Based Prompting

"**Shots**" = number of examples in the prompt.



Zero-Shot Prompting

- we give the model a direct instruction to perform a task without providing any examples or demonstrations.
- This means the model has to rely entirely on its pre-trained knowledge to figure out how to complete the task

Zero-Shot Prompt Copy

```
Classify the sentiment of the following text as positive, negative, or neutral.
Text: I think the vacation was okay.
Sentiment:
```

The model will provide a classification based solely on the task description, without seeing any examples beforehand. The output might be:

AI Output Copy

```
Neutral
```

Examples of zero-shot prompting in practice:

- **Multilingual Translation**

```
Translate the following English sentence into Japanese: 'The quick brown fox jumps over the lazy dog.'
```

- **Question Answering**

```
Answer the following question based on the provided news article: 'What were the main findings of the recent study on climate change?'
```

- **Content Summarization**

```
Summarize the key points of the given research paper on quantum mechanics.
```

Advantages	
Versatility	Enables AI to handle a wide range of tasks without task-specific training.
Resource Efficiency	Reduces the need for large labeled datasets, saving time and energy.
Rapid Deployment	Allows faster development and execution for new tasks.
Generalization	Performs well even with limited information or examples.
Reduced Cost	Minimizes the cost of model development due to lower data and resource needs.
Customization	Prompts can be tailored to specific tasks for better task alignment.

Limitations	
Scalability Issues	Scaling to many tasks may require more resources and significant tuning.
Complex Domain Knowledge	May struggle with tasks requiring deep, field-specific expertise.
Limited Control	Users have less influence over the model's internal decision-making process.

One-shot Prompting

- **Enhances zero-shot prompting by providing a single example before the new task, which helps clarify expectations and improves model performance**

One-Shot Prompt Copy

```
Classify the sentiment of the following text as positive, negative, or neutral.
Text: The product is terrible.
Sentiment: Negative
Text: I think the vacation was okay. Sentiment:
```

Here, the model is shown a single example ("The product is terrible. Sentiment: Negative") before it processes the new input. This allows the model to better understand what it should do next. The output might now be more reliable:

AI Output Copy

```
Neutral
```

Few-Shot Prompting

- provides two or more examples, which helps the model recognize patterns and handle more complex tasks. With more examples, the model gains a better understanding of the task, leading to improved accuracy and consistency.

Few-Shot Prompt Copy

```
Classify the sentiment of the following text as positive, negative, or neutral.
Text: The product is terrible. Sentiment: Negative
Text: Super helpful, worth it Sentiment: Positive
Text: It doesnt work! Sentiment:
```

Here is the output of this prompt when passed through GPT-4. In this case, the model has two examples to learn from, making it more likely to generate an accurate response for the new input:

AI Output Copy

```
Negative
```

Few shot

Prompt Copy

```
INPUT: Software Engineer - Python specialist needed at TechCorp. 5+ years experience required. Salary range $90,000
- $120,000. Remote work available. Apply by June 30, 2024. OUTPUT: Position: Software Engineer Specialization:
Python Company: TechCorp Experience Required: 5+ years Salary Range: $90,000 - $120,000 Work Type: Remote
Application Deadline: June 30, 2024
INPUT: Marketing Manager for GlobalBrand. MBA preferred. 3-5 years in consumer goods marketing. $75K-$95K DOE.
Hybrid work model. Applications close July 15, 2024. OUTPUT: Position: Marketing Manager Company: GlobalBrand
Education: MBA preferred Experience Required: 3-5 years Industry: Consumer goods Salary Range: $75,000 - $95,000
Work Type: Hybrid Application Deadline: July 15, 2024
INPUT: Data Scientist wanted at AI Innovations Ltd. PhD in Computer Science or related field. Minimum 2 years
industry experience. Competitive salary €60,000 - €80,000 based on experience. On-site work in Berlin office. Apply
by August 31, 2024. OUTPUT:
```

Here's the potential output. Notice how we used the INPUT, OUTPUT format instead of the colon (:) format. We will address the signifigance of this later.

AI Output Copy

```
Position: Data Scientist Company: AI Innovations Ltd Education: PhD in Computer Science or related field Experience
Required: Minimum 2 years Salary Range: €60,000 - €80,000 Work Type: On-site Location: Berlin Application Deadline:
August 31, 2024
```

Aspect	Zero-Shot Prompting	One-Shot Prompting	Few-Shot Prompting
Definition	No example is provided in the prompt.	One example is included in the prompt.	Two or more examples are provided in the prompt.
Training Need	No task-specific training or examples needed.	Minimal task guidance via one example.	More detailed task guidance via multiple examples.
Ease of Use	Easiest to implement.	Slightly more effort to include a relevant example.	Requires careful selection of multiple representative examples.
Model Understanding	Relies solely on pre-trained knowledge.	Gains some task-specific clarity from the example.	Recognizes patterns and structure more clearly.
Performance	May be less accurate, especially for complex tasks.	Moderately improved performance.	Generally better performance with nuanced tasks.
Best Use Case	Simple or common tasks with clear instructions.	Tasks with a unique output style or structure.	Complex tasks or those with ambiguous requirements.
Example Complexity	No examples: "Translate to French: Good morning"	One example: "English: Hello → French: Bonjour\nTranslate: Hi"	Multiple examples of inputs and outputs in the prompt.

How to Choose the Right Prompting Technique

Zero-shot prompting: Use this when the task is simple, well-understood, or frequently encountered in the model's training data. It's efficient for tasks like basic arithmetic, general queries, or sentiment classification for common phrases.

One-shot prompting: This is helpful for tasks that need more specific guidance or when the model struggles with ambiguity. Providing a single example can clarify the task, improving accuracy in tasks like basic classification or structured information extraction.

Few-shot prompting: Best used for complex tasks requiring multiple examples to establish patterns. This technique is ideal for tasks that involve varied inputs, require precise formatting, or demand a higher degree of accuracy, such as generating structured outputs or handling nuanced classifications.

Types of Prompting

1. Shot-Based Prompting-Zero-Shot, One-Shot, Few-Shot
2. Reasoning-Based Prompting
   - Chain-of-Thought (CoT) Prompting
   - One-Shot CoT
   - Few-Shot CoT
   - Self-Consistency CoT
   - Tree of Thoughts (ToT)
3. Interaction and Feedback-Based Prompting
   - ReAct Prompting (Reasoning + Acting)
   - Positive and Negative Prompting
   - Iterative Prompting
   - Model-Guided Prompting
4. Instruction and Role-Based Prompting
   - Instruction Prompting
   - Role Prompting

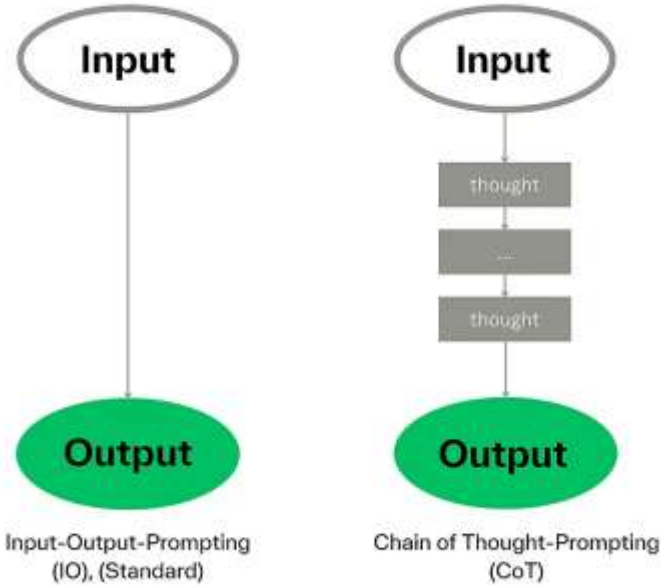
Reasoning-Based Prompting

- **Chain-of-Thought (CoT) Prompting**
- **One-Shot CoT**
- **Few-Shot CoT**
- **Self-Consistency CoT**
- **Tree of Thoughts (ToT)**

- designing prompts that help the model **simulate logical reasoning or problem-solving processes** rather than just providing a direct answer
- Mimics human-like **thinking patterns**

Chain-of-Thought (CoT) Prompting

complex reasoning capabilities through intermediate reasoning steps.



- Helps the model **think systematically** and build a solution strategy step by step.
- Involves a **series of linked questions or steps** that guide reasoning toward the answer.
- Encourages the AI to **break down complex tasks** into smaller, logical parts.

Problem:
"You have 3 keys. Only one opens a locked door. You can try them one at a time. What is the minimum number of tries needed to be sure you open the door?"

Thought 1: - Try key A first. - If it works, done in 1 try. - If it doesn't, try key B. - If it works, done in 2 tries. - If not, key C must be the correct one → 3 tries.

Thought 2: - Try key B first. - If it works, done in 1 try. - If not, try A or C... [Continue reasoning through multiple strategies.]

Evaluate: - In any sequence, worst case = 3 tries. - Best case = 1 try. - To be sure the door opens → need up to 3 tries. Answer: **Minimum tries needed to guarantee opening the door is 3.**

Step 1:
Prompt: "What is the weather?"

Step 2:
Output: "The weather refers to the atmospheric conditions at a specific location at a specific time."

Step 3:
Prompt: "How does the weather affect our daily lives?"

Step 4:
Output: "The weather affects our daily lives in many ways. It determines what we wear, influences our travel plans, and can even impact our mood."

Step 5:
Prompt: "Give some examples of how the weather can affect mood."

Step 6:
Output: "Bad weather, such as rain and darkness, can lead to a gloomy mood. On the other hand, sunny weather can positively affect mood and increase motivation."

Step 7:
Prompt: Generate a continuous response that further explores the connection between weather and mood.

Chain-of-thought

Chain-of-thought prompting is a technique designed to **guide the AI model through a logical reasoning process**. Rather than just asking for an output, this strategy incorporates a "chain of thought" that demonstrates how to arrive at the correct answer step by step. This approach is especially valuable for tasks requiring critical thinking or problem-solving skills, like mathematical calculations or complex queries.

Example. Suppose you're in charge of optimizing fuel efficiency for a transportation company's fleet of trucks. You can employ chain-of-thought prompting to guide the model through this complex task as follows.

Prompt: "You're the fleet manager for a transportation company with 50 diesel trucks. Each truck gets 6 miles per gallon, and the fleet covers a total of 10,000 miles each day. You're looking to reduce fuel consumption to save costs. Let's think step by step."

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. ✓

(a) Few-shot

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: 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:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

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: 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:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. ***The answer is 4.*** ✓

(c) Zero-shot

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: The answer (arabic numerals) is

(Output) 8 ✗

(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. ✓

Chain of thought

```
[ ]  prompt = """
     15 of us want to go to a restaurant.
     Two of them have cars
     Each car can seat 5 people.
     Two of us have motorcycles.
     Each motorcycle can fit 2 people.

     Can we all get to the restaurant by car or motorcycle?
     """
     response = llama(prompt)
     print(response)
```

```
prompt = """
15 of us want to go to a restaurant.
Two of them have cars
Each car can seat 5 people.
Two of us have motorcycles.
Each motorcycle can fit 2 people.

Can we all get to the restaurant by car or motorcycle?

Think step by step.
Explain each intermediate step.
Only when you are done with all your steps,
provide the answer based on your intermediate steps.
"""
response = llama(prompt)
print(response)
```

```
[ ]  prompt = """
     15 of us want to go to a restaurant.
     Two of them have cars
     Each car can seat 5 people.
     Two of us have motorcycles.
     Each motorcycle can fit 2 people.

     Can we all get to the restaurant by car or motorcycle?

     Think step by step.
     """
     response = llama(prompt)
     print(response)
```

```
[ ]  prompt = """
     15 of us want to go to a restaurant.
     Two of them have cars
     Each car can seat 5 people.
     Two of us have motorcycles.
     Each motorcycle can fit 2 people.

     Can we all get to the restaurant by car or motorcycle?
     Think step by step.
     Provide the answer as a single yes/no answer first.
     Then explain each intermediate step.
     """

     response = llama(prompt)
     print(response)
```

Prompt:

```
The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.
The odd numbers in this group add up to an even number: 17,  10, 19, 4, 8, 12, 24.
A: Adding all the odd numbers (17, 19) gives 36. The answer is True.
The odd numbers in this group add up to an even number: 16,  11, 14, 4, 8, 13, 24.
A: Adding all the odd numbers (11, 13) gives 24. The answer is True.
The odd numbers in this group add up to an even number: 17,  9, 10, 12, 13, 4, 2.
A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is False.
The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:
```

Output:

```
Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer is False.
```

One-Shot & Few-Shot Chain-of-Thought (CoT) Prompting

One-Shot CoT: Provides **a single example** with detailed reasoning steps.
Few-Shot CoT: Includes **multiple examples** of reasoning-based answers.

One-Shot CoT Example

Prompt:

Q: If you have 3 apples and you buy 2 more, how many apples do you have?
A: Let's think step by step.
I start with 3 apples. Then I buy 2 more apples. So, 3 + 2 = 5 apples.
Therefore, the answer is **5**.

New Question:

Q: If you have 4 pencils and find 3 more, how many pencils do you have?
A: Let's think step by step.
I start with 4 pencils. Then I find 3 more pencils. So, 4 + 3 = **7** pencils.

Few-Shot CoT Example

Prompt:

Q1: If a book costs $5 and you buy 2 books, how much do you spend?
A: Let's think step by step.
One book costs $5. Buying 2 books means 5 × 2 = $10.
So, the answer is **$10**.
Q2: If there are 6 students and each student has 3 notebooks, how many notebooks are there in total?
A: Let's think step by step.
Each of the 6 students has 3 notebooks. So, 6 × 3 = 18 notebooks.
So, the answer is **18**.

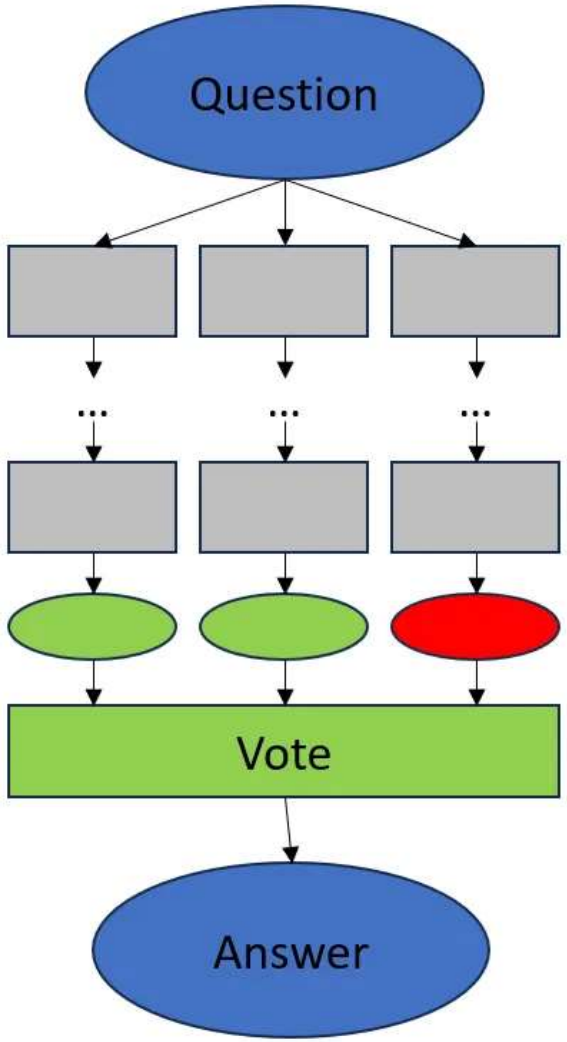
New Question:

Q3: If there are 4 bags and each bag contains 5 apples, how many apples are there in total?
A: Let's think step by step.
Each of the 4 bags has 5 apples. So, 4 × 5 = **20** apples.

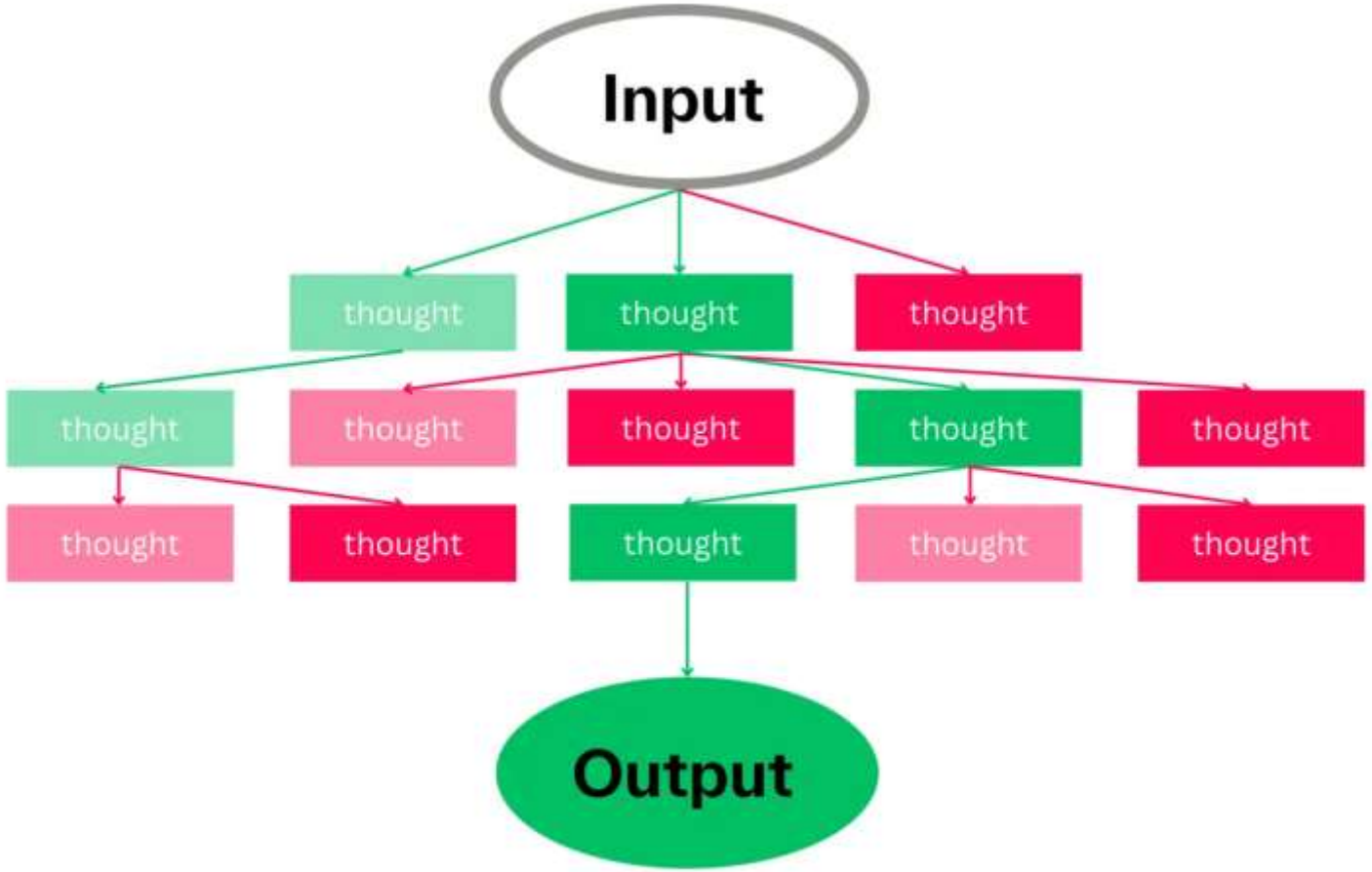
Chain-of-Thought (CoT) with Self-Consistency

- Introduced by Wang et al. in *"Self-Consistency Improves Chain of Thought Reasoning in Language Models"* (2022).
- Enhances standard CoT by improving the **accuracy and reliability** of generated responses.

- Assumes the **correct answer exists** in the model and appears in **majority of multiple responses**.
- The same question is asked **multiple times**, generating **diverse reasoning paths**.
- A **majority vote** is taken across responses to select the most consistent, likely correct answer.
- This process **synthesizes one final, reliable answer** from multiple attempts.
- Results in **higher-quality reasoning and improved performance** on complex tasks.



TREE OF THOUGHTS-PROMPTING



Tree of Thoughts-Prompting
(ToT)

Prompt:

I have the following problem statement:

<Relationship between weather and mood>

Go through the following phases to solve it:

1. **Brainstorm** four different solutions, considering various factors.
2. **Evaluate** the potential of each proposed solution. Consider the pros and cons, initial effort, implementation difficulties, potential challenges, and expected outcomes. Assign a probability of success and a confidence level to each option based on these factors and rank them. Announce the two highest-rated solutions.
3. For the two highest-rated solutions, do the following: Develop each solution into two different implementation variants that consider different aspects.
4. **Deepen** the thought process for each implementation variant of each solution, and generate potential scenarios, strategies for implementation, and ways to overcome potential obstacles. Also consider possible unexpected outcomes and how they could be addressed. Assign a probability of success and a confidence level to each implementation variant based on these factors and rank them. Announce the highest-rated implementation variant for each solution.
5. Based on the given solutions and their implementation variants, **select the most promising solution** with the most promising implementation variant.
6. **In summary**, **outline** the final solution in its final implementation variant.

Types of Prompting

1. Shot-Based Prompting-Zero-Shot, One-Shot, Few-Shot
2. Reasoning-Based Prompting
   - Chain-of-Thought (CoT) Prompting
   - One-Shot CoT
   - Few-Shot CoT
   - Self-Consistency CoT
   - Tree of Thoughts (ToT)
3. Interaction and Feedback-Based Prompting
   - ReAct Prompting (Reasoning + Acting)
   - Positive and Negative Prompting
   - Iterative Prompting
   - Model-Guided Prompting
4. Instruction and Role-Based Prompting
   - Instruction Prompting
   - Role Prompting
5. Knowledge-Augmented Prompting
   - Generated Knowledge Prompting (GKP)

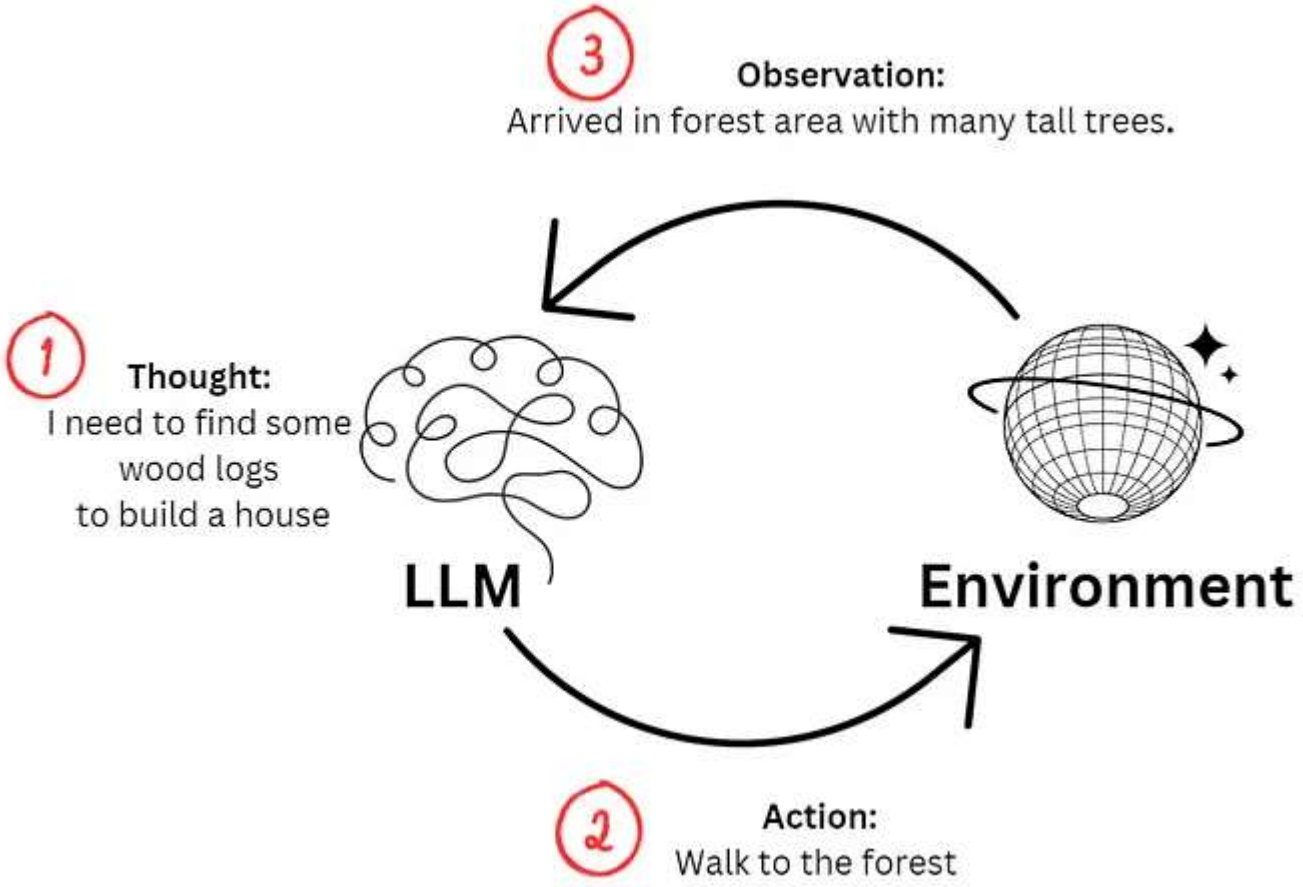
Interaction and Feedback-Based Prompting

- **ReAct Prompting (Reasoning + Acting)**
- **Positive and Negative Prompting**
- **Iterative Prompting**
- **Model-Guided Prompting**

- A prompting approach where the **LLM interacts with intermediate outputs, external environments, or iterative prompts** to refine its reasoning and actions.
- Dynamic and adaptable.

ReAct

- **ReAct = "Reasoning and Acting"** – separates reasoning from action-taking in AI models.
- Feeds **observations to the LLM**, allowing it to **update context dynamically**.
- Model **re-evaluates and responds** step-by-step based on updated insights.
- **Different from Chain-of-Thought (CoT)** – reasoning isn't static or embedded in one prompt.
- Enables **structured handling of complex queries**, improving output quality.
- Helps LLMs produce **more informed, accurate, and coherent** responses.



ReAct

(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple … originally designed to control the Front Row media center program …

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports',' Front Row (software) ', …]

Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software …

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: Finish[keyboard function keys]

Positive and negative prompting

Positive and negative prompting are techniques used to **guide the model's output in specific directions.** As the names suggest, positive prompting encourages the model to include certain types of content or responses, while negative prompting discourages it from including other specific types. This framing can be vital in controlling the direction and quality of the model's output.

Here's what your positive prompt could look like:

"You are a sustainability expert. Generate a list of five feasible strategies for a small business to reduce its carbon footprint."

Building on the above scenario, you should ensure the model doesn't suggest too expensive or complicated strategies for a small business. So you extend your initial prompt as follows:

"***You are a sustainability expert. Generate a list of five feasible strategies for a small business to reduce its carbon footprint. Do not include suggestions that require an initial investment of more than $10,000 or specialized technical expertise."***

Adding the negative prompting here filters out too expensive or technical strategies, tailoring the model's output to your specific requirements.

Iterative prompting

Iterative prompting is a strategy that involves **building on the model's previous outputs to refine, expand, or dig deeper into the initial answer.** This approach enables you to break down complex questions or topics into smaller, more manageable parts, which can lead to more accurate and comprehensive results.

The key to this technique is paying close attention to the model's initial output. You can then create follow-up prompts to explore specific elements, inquire about subtopics, or ask for clarification. This approach is useful for projects that require in-depth research, planning, or layered responses.

Iterative prompting

- Example. Your initial prompt might be:
- "I am working on a project about fraud prevention in the travel industry. Please provide me with a general outline covering key aspects that should be addressed."
- Assume the model's output includes points like identity verification, secure payment gateways, and transaction monitoring.
- Follow-up prompt 1: "Great, could you go into more detail about identity verification methods suitable for the travel industry?"
- At this point, the model might elaborate on multifactor authentication, biometric scanning, and secure documentation checks.
- Follow-up prompt 2: "Now, could you explain how transaction monitoring can be effectively implemented in the travel industry?"
- The model could then discuss real-time monitoring, anomaly detection algorithms, and the role of machine learning in identifying suspicious activities.

- This iterative approach allows you to go from a broad question to specific, actionable insights in a structured manner, making it particularly useful for complex topics like fraud prevention in the travel industry.

Model guided prompting

Another useful approach is model-guided prompting, which **flips the script by instructing the model to ask you for the details it needs to complete a given task.** This approach minimizes guesswork and discourages the model from making things up.

Example. Suppose you work in travel tech and you want an AI model to generate a FAQ section for a new travel booking feature on your platform. Instead of just asking the model to **"create a FAQ for a new booking feature," which could result in generic or off-target questions and answers,** you could prompt the AI as follows:

"I need you to generate a FAQ section for a new travel booking feature we're launching. Can you ask me for the information you need to complete this?"

ChatGPT might then ask you, "What is the name of the new travel booking feature?" and "What is the primary purpose or functionality of this new feature?" among other things.

Types of Prompting

1. Shot-Based Prompting-Zero-Shot, One-Shot, Few-Shot
2. Reasoning-Based Prompting
   - Chain-of-Thought (CoT) Prompting
   - One-Shot CoT
   - Few-Shot CoT
   - Self-Consistency CoT
   - Tree of Thoughts (ToT)
3. Interaction and Feedback-Based Prompting
   - ReAct Prompting (Reasoning + Acting)
   - Positive and Negative Prompting
   - Iterative Prompting
   - Model-Guided Prompting
4. Instruction and Role-Based Prompting
   - Instruction Prompting
   - Role Prompting
5. Knowledge-Augmented Prompting
   - Generated Knowledge Prompting (GKP)

Instruction Prompting

- providing **clear, explicit instructions** to the model about how to perform a task.
- Helps **direct the model's focus** towards a specific goal.

- **Examples of Instruction Prompting:**
- **Summarization**
- *Instruction*: "Summarize the following article in 3 sentences."
  Model Output: [Concise summary of the article.]
- **Sentiment Analysis**
- *Instruction*: "Classify the sentiment of the following text as Positive, Negative, or Neutral."
  Model Output: Positive.
- **Text Generation**
- *Instruction*: "Write a 200-word essay on the impact of climate change on agriculture."
  Model Output: [Generated essay.]

Role Prompting

The role-playing technique employs a unique approach to crafting prompts: instead of using examples or templates to guide the model's output**, you assign a specific "role" or "persona" to the AI model**. This often includes explicitly explaining the intended audience, the AI's role, and the goals of the interaction. The roles and goals offer contextual information that helps the model understand the purpose of the prompt and the tone or level of detail it should aim for in its response.

Example. Consider the prompt: *"You are a prompt engineer, and you need to explain to a 6-year-old kid what your job is."*

- **Prompt**: "Your role is a machine learning expert who gives highly technical advice to senior engineers who work with complicated datasets. Explain the pros and cons of using PyTorch."
- **Expectation**: The model provides a more detailed and technical explanation suitable for senior engineers.

- **Role Prompting**: Instructs the model to adopt a specific role or perspective, guiding the output to be more contextually appropriate and detailed.

Role Prompting

- Roles give context to LLMs what type of answers are desired.
- Llama 2 often gives more consistent responses when provided with a role.
- First, try standard prompt and see the response.

```
[ ]  prompt = """
     How can I answer this question from my friend:
     What is the meaning of life?
     """
     response = llama(prompt)
     print(response)
```

```
[ ]  role = """
     Your role is a life coach \
     who gives advice to people about living a good life.\
     You attempt to provide unbiased advice.
     You respond in the tone of an English pirate.
     """

     prompt = f"""
     {role}
     How can I answer this question from my friend:
     What is the meaning of life?
     """
     response = llama(prompt)
     print(response)
```

You can think about giving explicit instructions as using rules and restrictions to how Llama 2 responds to your prompt.

- Stylization
  - `Explain this to me like a topic on a children's educational network show teaching elementary students.`
  - `I'm a software engineer using large language models for summarization. Summarize the following text in under 250 words:`
  - `Give your answer like an old timey private investigator hunting down a case step by step.`
- Formatting
  - `Use bullet points.`
  - `Return as a JSON object.`
  - `Use less technical terms and help me apply it in my work in communications.`
- Restrictions
  - `Only use academic papers.`
  - `Never give sources older than 2020.`
  - `If you don't know the answer, say that you don't know.`

```
[ ]  prompt = """
     Who won the 2023 Women's World Cup?
     """
     response = llama(prompt)
     print(response)
```

- As you can see, the model still thinks that the tournament is yet to be played, even though you are now in 2024!
- Another thing to **note** is, July 18, 2023 was the date the model was released to public, and it was trained even before that, so it only has information upto that point. The response says, "the final match is scheduled to take place in July 2023", but the final match was played on August 20, 2023.

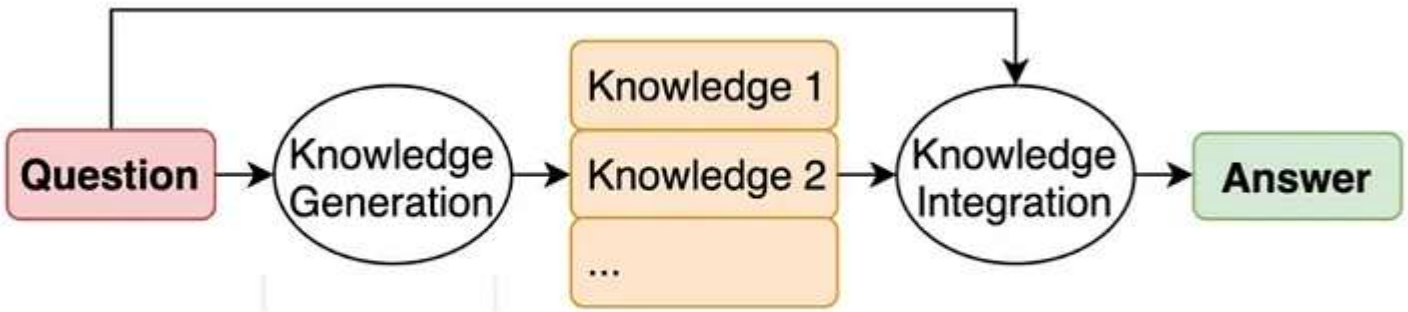
```
context = """
The 2023 FIFA Women's World Cup (Māori: Ipu Wahine o te Ao FIFA i 2023)[1] was the ninth edition of the FIFA Women's World Cup, the quadrennial international women's football champior
This tournament was the first to feature an expanded format of 32 teams from the previous 24, replicating the format used for the men's World Cup from 1998 to 2022.[2] The opening mat
Spain were crowned champions after defeating reigning European champions England 1–0 in the final. It was the first time a European nation had won the Women's World Cup since 2007 anc
Of the eight teams making their first appearance, Morocco were the only one to advance to the round of 16 (where they lost to France; coincidentally, the result of this fixture was si
Australia's team, nicknamed the Matildas, performed better than expected, and the event saw many Australians unite to support them.[15][16][17] The Matildas, who beat France to make t
It was the most attended edition of the competition ever held.
"""
```

```
[ ] prompt = f"""
    Given the following context, who won the 2023 Women's World cup?
    context: {context}
    """
    response = llama(prompt)
    print(response)
```

Types of Prompting

1. Shot-Based Prompting-Zero-Shot, One-Shot, Few-Shot
2. Reasoning-Based Prompting
   - Chain-of-Thought (CoT) Prompting
   - One-Shot CoT
   - Few-Shot CoT
   - Self-Consistency CoT
   - Tree of Thoughts (ToT)
3. Interaction and Feedback-Based Prompting
   - ReAct Prompting (Reasoning + Acting)
   - Positive and Negative Prompting
   - Iterative Prompting
   - Model-Guided Prompting
4. Instruction and Role-Based Prompting
   - Instruction Prompting
   - Role Prompting
5. Knowledge-Augmented Prompting
   - Generated Knowledge Prompting (GKP)

Generated Knowledge Prompting



Generated Knowledge (Liu et al.)

model **first generates relevant background or contextual knowledge** and then uses that generated knowledge to inform the task-specific prompt.

Task	NumerSense	QASC
Prompt	Generate some numerical facts about objects. Examples: Input: **penguins have <mask> wings.** Knowledge: *Birds have two wings. Penguin is a kind of bird.* ... Input: **a typical human being has <mask> limbs.** Knowledge: *Human has two arms and two legs.* Input: **{question}** Knowledge:	Generate some knowledge about the input. Examples: Input: **What type of water formation is formed by clouds?** Knowledge: *Clouds are made of water vapor.* ... Input: **The process by which genes are passed is** Knowledge: *Genes are passed from parent to offspring.* Input: **{question}** Knowledge:


CodeLlama
WizardCoder
Phind-CodeLlama
OpenAssistant-Codellama
NexusRaven
Mistral
Mistralic
SQLCoder-7B
Mistral-OpenOrca
StarCoder
SQLCoder-15B
WizardCoder-15B
Llama2
OpenChat
CodeLlama
Nous-Hermes
Vicuna
NSQL
Redmond-Puffin
Llama2-32K
OpenAssistant-Llama2
WizardLM
GPT
GPT-3.5-Turbo
GPT-4
Stable
StableLM
StableCode
OpenAssistant-StableLM
Pythia
Pythia-ChatBase
OpenAssistant-Pythia
Claude
Claude-2
PaLM
PaLM-2
Falcon
Falcon
Qwen
Qwen
Map of LLMs
October 2023

Getting an LLM

Large language models are deployed and accessed in a variety of ways, including:

1.Self-hosting: Using local hardware to run inference. Ex. running Llama 2 on your Macbook Pro using llama.cpp.

1. Best for privacy/security or if you already have a GPU.

2.Cloud hosting: Using a cloud provider to deploy an instance that hosts a specific model. Ex. running Llama 2 on cloud providers like AWS, Azure, GCP, and others.

1. Best for customizing models and their runtime (ex. fine-tuning a model for your use case).

3.Hosted API: Call LLMs directly via an API. There are many companies that provide Llama 2 inference APIs including AWS Bedrock, Replicate, Anyscale, Together and others.

1. Easiest option overall.

Llama Models

In 2023, Meta introduced the Llama language models (Llama Chat, Code Llama, Llama Guard). These are general purpose, state-of-the-art LLMs.

Llama 2 models come in 7 billion, 13 billion, and 70 billion parameter sizes. Smaller models are cheaper to deploy and run arger models are more capable.

Llama 2

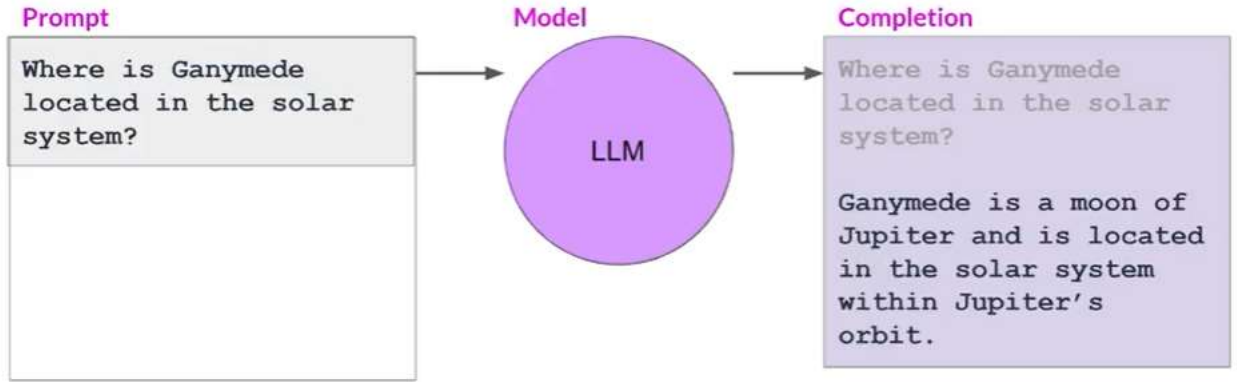
1. `llama-2-7b` - base pretrained 7 billion parameter model
2. `llama-2-13b` - base pretrained 13 billion parameter model
3. `llama-2-70b` - base pretrained 70 billion parameter model
4. `llama-2-7b-chat` - chat fine-tuned 7 billion parameter model
5. `llama-2-13b-chat` - chat fine-tuned 13 billion parameter model
6. `llama-2-70b-chat` - chat fine-tuned 70 billion parameter model (flagship)

Code Llama

1. `codellama-7b` - code fine-tuned 7 billion parameter model
2. `codellama-13b` - code fine-tuned 13 billion parameter model
3. `codellama-34b` - code fine-tuned 34 billion parameter model
4. `codellama-7b-instruct` - code & instruct fine-tuned 7 billion parameter model
5. `codellama-13b-instruct` - code & instruct fine-tuned 13 billion parameter model
6. `codellama-34b-instruct` - code & instruct fine-tuned 34 billion parameter model
7. `codellama-7b-python` - Python fine-tuned 7 billion parameter model
8. `codellama-13b-python` - Python fine-tuned 13 billion parameter model
9. `codellama-34b-python` - Python fine-tuned 34 billion parameter model

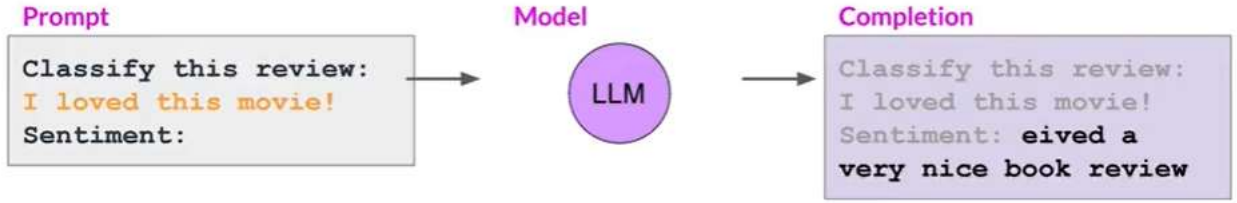
Code Llama is a code-focused LLM built on top of Llama 2 also available in various sizes and finetunes:

Prompt and prompt engineering (**In-context Learning**)



Context window: typically a few thousand words

In-context learning (ICL) - zero shot inference



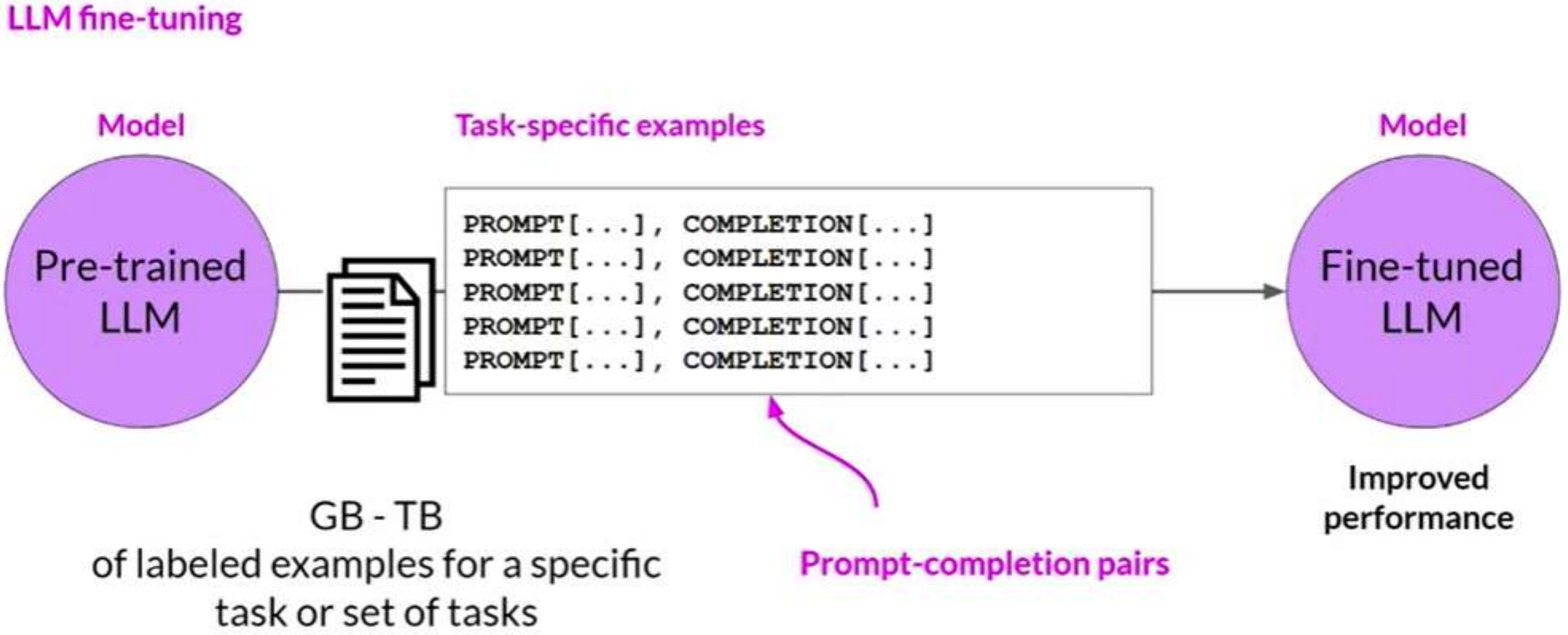
Fine tuning – why?

Limitations of in-context learning

- In-context learning may not work for smaller models LLM
- Examples take up space in the context window

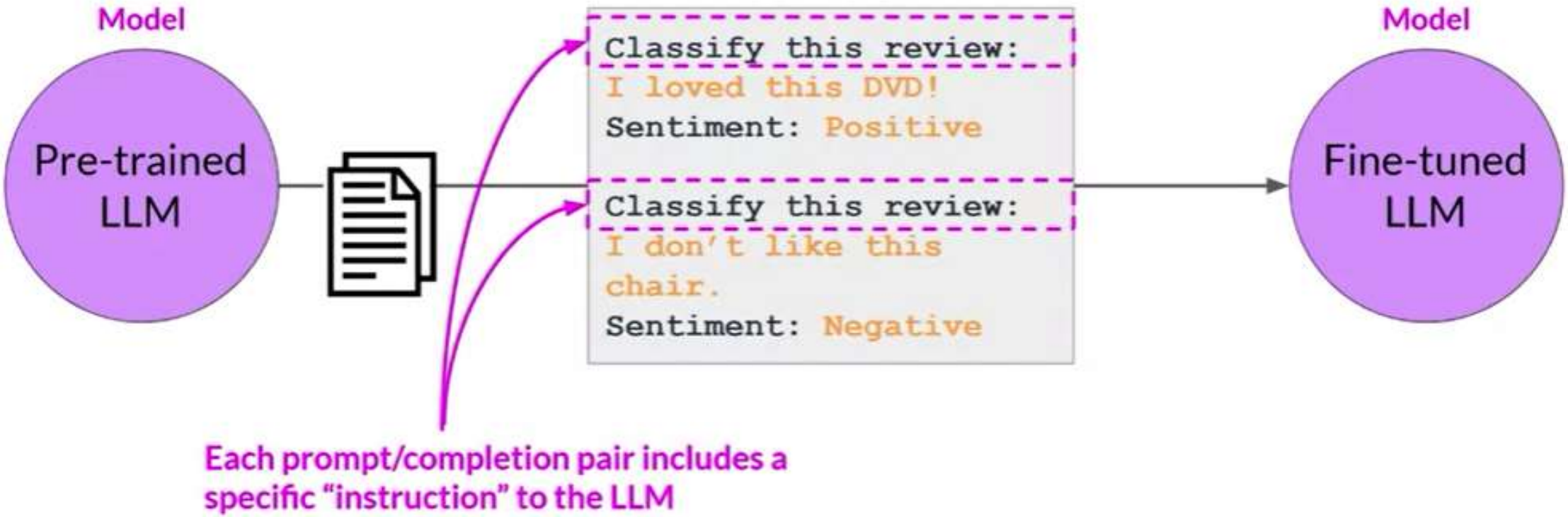
Instead, try **fine-tuning** the model

LLM fine-tuning at a high level



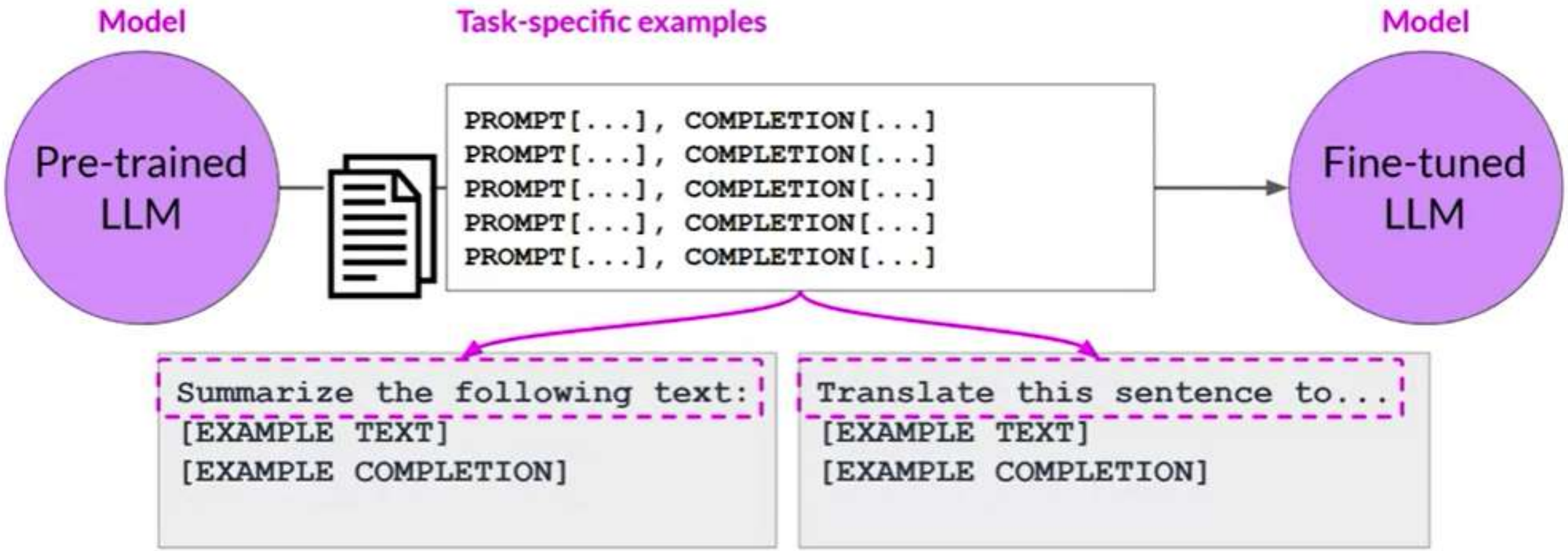
Using prompts to fine-tune LLMs with instruction

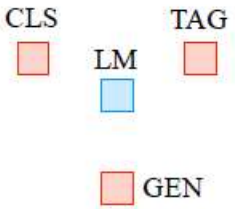
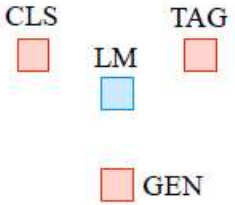
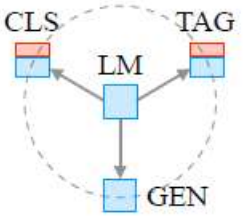
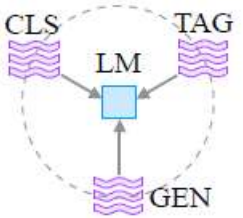
LLM fine-tuning



Using prompts to fine-tune LLMs with instruction

LLM fine-tuning



Paradigm	Engineering	Task Relation
a. Fully Supervised Learning (Non-Neural Network)	Features (e.g. word identity, part-of-speech, sentence length)	
b. Fully Supervised Learning (Neural Network)	Architecture (e.g. convolutional, recurrent, self-attentional)	
c. Pre-train, Fine-tune	Objective (e.g. masked language modeling, next sentence prediction)	
d. Pre-train, Prompt, Predict	Prompt (e.g. cloze, prefix)	

Zero- shot prompting

Zero-shot prompting

Zero-shot prompting is one of the most straightforward yet versatile techniques in prompt engineering. At its core, it involves **providing the language model with a single instruction**, often presented as a question or statement, without giving any additional examples or context. The model then generates a response based on its training data, essentially "completing" your prompt in a manner that aligns with its understanding of language and context.

Zero-shot prompting is exceptionally useful for generating fast, on-the-fly responses to a broad range of queries.

Consider you are looking for a sentiment analysis of a hotel review. You could use the following prompt.

“Extract the sentiment from the following review: ‘The room was spacious, but the service was terrible.’”

Without prior training on sentiment analysis tasks, the model can still process this prompt and provide you with an answer, such as "The review has a mixed sentiment: positive towards room space but negative towards the service."

ZERO SHOT EXAMPLE

```
[ ] complete_and_print("Text: This was the best movie I've ever seen! \n The sentiment of the text is: ")
    # Returns positive sentiment

    complete_and_print("Text: The director was trying too hard. \n The sentiment of the text is: ")
    # Returns negative sentiment
```

```
==============
Text: This was the best movie I've ever seen!
 The sentiment of the text is:
==============
 Sure! Based on the text you provided, the sentiment of the text is POSITIVE. The use of the word "best" and "ever seen" both convey a positive sentiment.

==============
Text: The director was trying too hard.
 The sentiment of the text is:
==============
 Sure, I'd be happy to help! The sentiment of the text "The director was trying too hard" is neutral. It does not convey any positive or negative emotions. The phrase "trying too hard

Please let me know if you have any other questions!
```

```
[ ]  prompt = """
     Message: Hi Amit, thanks for the thoughtful birthday card!
     Sentiment: ?
     """

     response = llama(prompt)
     print(response)
```

One –shot prompting

One-shot prompting is a technique where a **single example guides the AI model's output**. This example can be a question-answer pair, a simple instruction, or a specific template. The aim is to align the model's response more closely with the user's specific intentions or desired format.

Let's consider a language model that has never been trained to generate recipes.
With one-shot prompting, you provide the model with a single example recipe:
Prompt: **"Generate a recipe for chocolate chip cookies."**
Example Recipe: **"Ingredients: butter, sugar, eggs, flour, chocolate chips. Instructions: Preheat oven to 350°F. Mix butter and sugar..."**
Even though the model hasn't seen this specific example during training, it can use the structure of the provided example to generate a new recipe:
Generated Recipe: **"Ingredients: margarine, brown sugar, egg substitute, all-purpose gluten-free flour, dairy-free chocolate chips. Instructions: Preheat oven to 350°F. Cream margarine and brown sugar..."**

ONE SHOT EXAMPLE

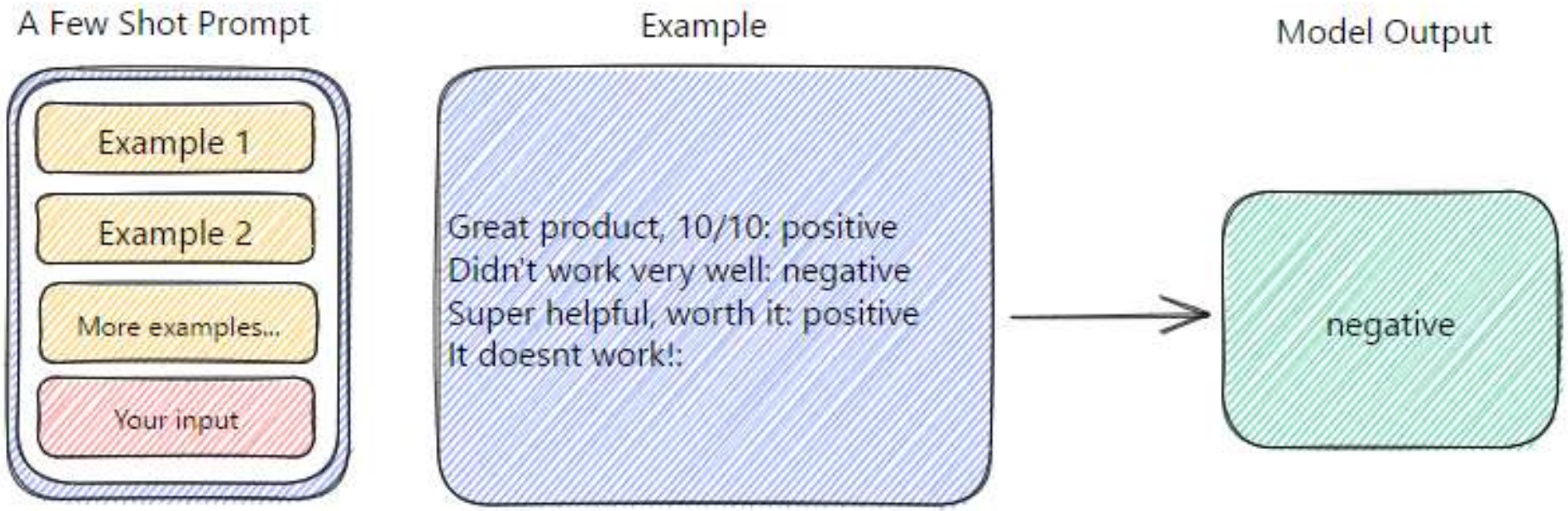
```python
def one_shot_complete_and_print(prompt: str, example: str, model: str = DEFAULT_MODEL):
    # Combine the example and the prompt
    full_prompt = f"{example}\n\n{prompt}"
    print(f'==============\n{full_prompt}\n==============')
    response = completion(full_prompt, model)
    print(response, end='\n\n')

# Define the one-shot example
one_shot_example = """
Text: I love sunny days!
The sentiment of the text is: positive
"""

# Use the function with one-shot learning
one_shot_complete_and_print("Text: This was the best movie I've ever seen! \nThe sentiment of the text is:", one_shot_example)
# Should return positive sentiment

one_shot_complete_and_print("Text: The director was trying too hard. \nThe sentiment of the text is:", one_shot_example)
# Should return negative sentiment
```

Few Shot example



Prompt: Given a few sentences, classify their sentiment as Positive, Negative, or Neutral.
Few-Shot Examples:
1."I love this product!" - Positive
2."This is the worst experience ever." - Negative
3."It's an average movie, nothing special." - Neutral
The model uses these examples to understand the sentiment analysis task, identifying key words and phrases indicative of each sentiment category.

response

INPUT: I thought it was okay

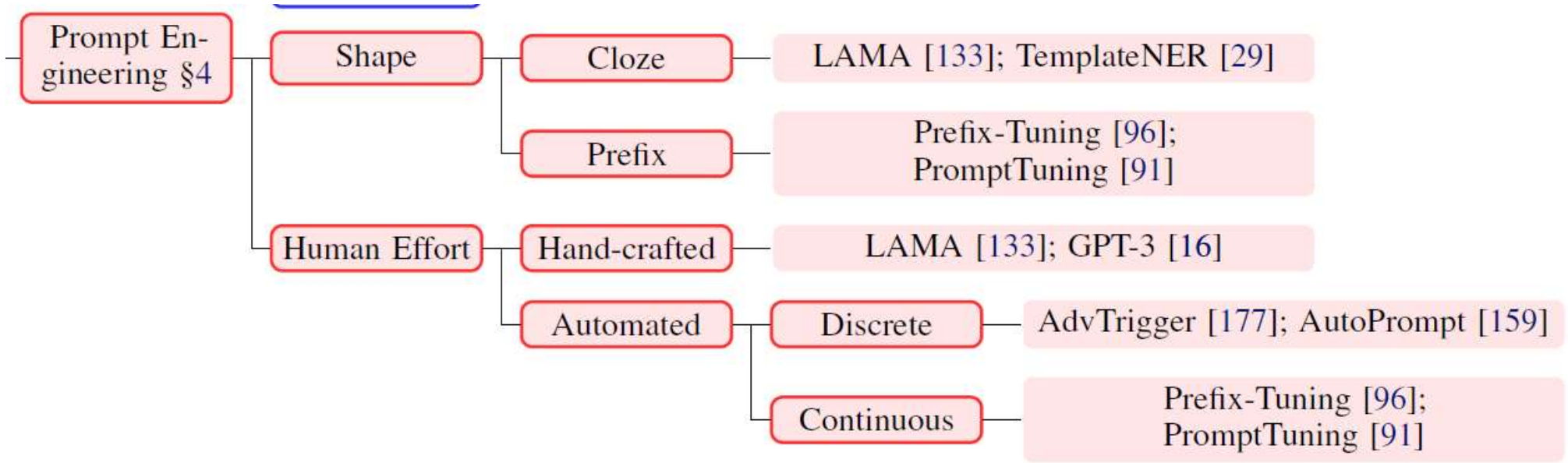
Sure, I'd be happy to help! As a sentiment classifier, I can analyze the messages you provide and give the percentage of positive, neutral, and negative sentiment for each one. Here are my assessments for the messages you've given so far:

1. "I liked it" -7 0% positive, 30% neutral, 0% negative
2. "It could be better" - 0% positive, 50% neutral, 50% negative
3. "It's fine" - 25% positive, 50% neutral, 25% negative
4. "I thought it was okay" - 25% positive, 40% neutral, 35% negative

Please keep in mind that these assessments are based on my analysis of the language used in each message, and may not reflect the actual sentiment or intention of the speaker. Additionally, it's important to note that sentiment analysis can be subjective and may not always be accurate or reliable.

Prompt Engineering

Prompts are designed to instruct the model on the kind of output expected. Here are some common types of prompts, with examples for various tasks:



Prompt Engineering

•**Human Effort Prompting** focuses on crafting detailed, context-rich, and specific prompts to guide the model towards generating high-quality responses.
•**Shape Prompting** involves structuring the prompts to influence the format and style of the model's outputs, ensuring the generated text adheres to a desired structure.

Shape Prompting

Shape prompting involves structuring prompts in a way that influences the "shape" or structure of the model's response. This could mean specifying the format, style, or sequence of the output. This technique is particularly useful for tasks where the structure of the output is as important as the content itself, such as generating lists, bullet points, tables, or following a specific writing style.

Characteristics:

1. **Output Structure**: Prompts guide the model to produce outputs in a specific format or style.
2. **Formatting Instructions**: They include explicit instructions about the layout of the response.
3. **Consistency**: Ensures the model generates consistent and predictable outputs.

Hand effort-Discrete Prompting

Discrete Prompting

Discrete prompting involves providing distinct and separate prompts for each task or query. Each prompt is self-contained and does not rely on the context from previous interactions. This approach is straightforward and often used when tasks are independent of each other or when the model needs to handle a variety of unrelated queries.

Characteristics:

1.Independence: Each prompt is independent and does not require context from previous prompts.

2.Clarity: Prompts are clear and specific to the task at hand.

3.Simplicity: This method is simple to implement as each query is treated separately.

Prompt 1: "What is the capital of France?“

Prompt 2: "List three benefits of exercise.“

Prompt 3: "Describe the process of photosynthesis."

Hand –Effort Continuous Prompting

Continuous prompting involves maintaining context across multiple prompts, allowing the conversation or task to build progressively. This approach is useful for complex tasks that require context retention, such as carrying on a conversation, multi-step problem-solving, or detailed storytelling.

Characteristics:

1. **Context Retention**: The model retains context from previous interactions.
2. **Coherence**: Ensures coherence and continuity in the generated responses.
3. **Complexity**: More complex to manage as it involves maintaining state and context across prompts.

Prompt 1: "Tell me about renewable energy."
Response 1: "Renewable energy comes from natural sources that are constantly replenished, such as sunlight, wind, and water."

Prompt 2: "What are the benefits of using renewable energy?"
Response 2: "The benefits include reducing greenhouse gas emissions, decreasing air pollution, and providing sustainable energy sources."

Prompt 3: "Can you explain how solar panels work?"
Response 3: "Solar panels convert sunlight into electricity using photovoltaic cells. These cells absorb sunlight and generate an electric current through the photovoltaic effect."

Shape prompt : Cloze prompt , Prefix prompt

Cloze prompts are distinctively structured to include **blanks** or missing words within a text string, posing a **fill-in-the-blank type** challenge to LLMs.

Prefix prompts, in contrast to cloze prompts, offer a unique approach particularly well-suited for generative tasks and applications involving standard auto-regressive LLMs. These prompts are designed to provide the **beginning, or the prefix, of a text string**, which aligns well with the sequential processing capabilities of these models.

Shape prompt : Cloze prompt , Prefix prompt

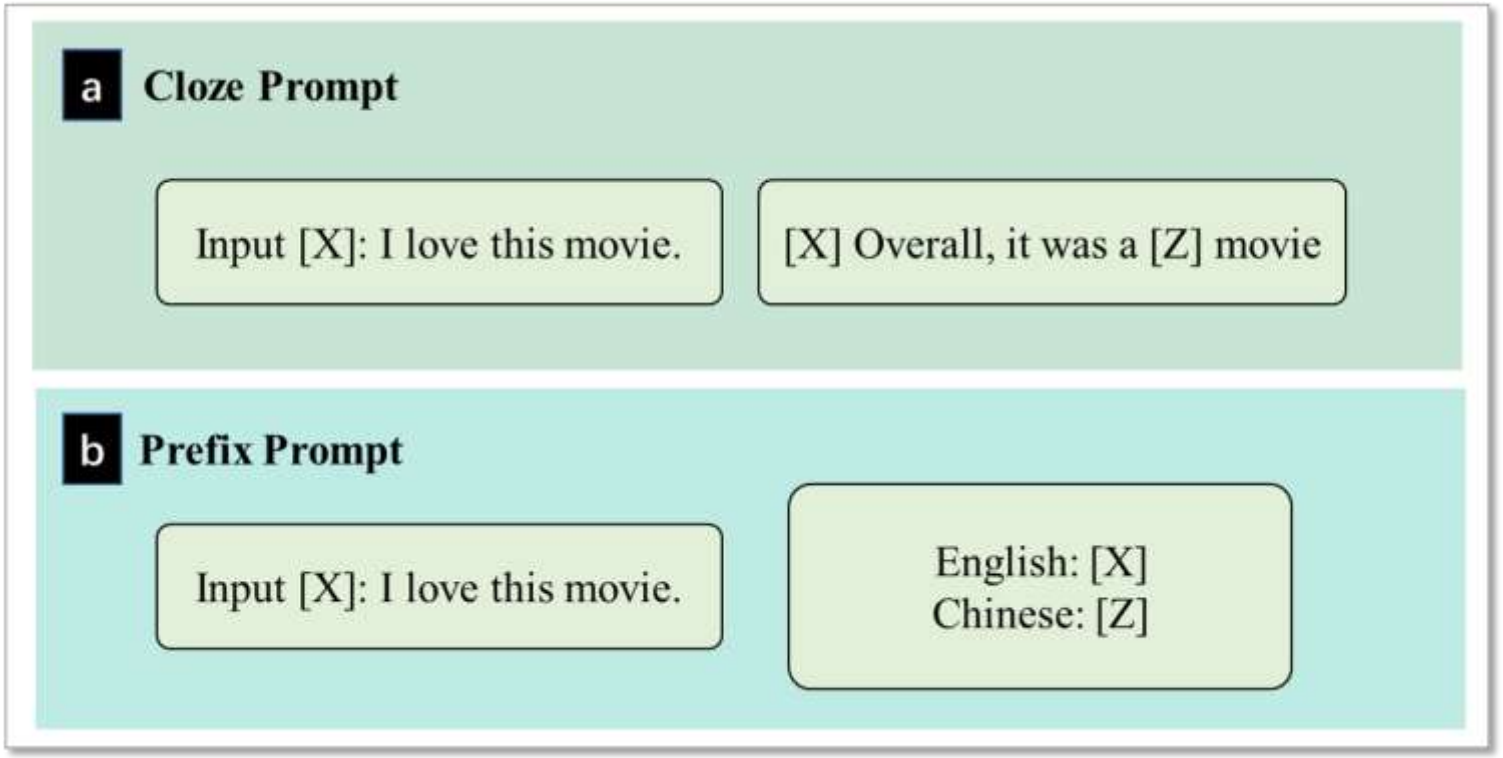
Prompt engineering refers to the process of creating a **prompt function** that guides a model to perform effectively in downstream tasks.
This process typically involves two steps.

First, a template is applied, which is a text string that contains two slots:

- an input slot [X] for the input text x and
- an answer slot [Z] for the intermediately generated answer text z

that will later be mapped into the final output y.

Then, the input slot [X] is filled with the input text x.



you can explicitly mention the template in your code using placeholders such as [], {}, or other markers to indicate where dynamic content should be inserted

Cloze Prefix Prompt Engineering

Steps for Cloze Prefix Prompt Engineering

1.Define the Task: Identify what you want the model to do (e.g., complete a sentence, answer a question, translate text).

2.Create the Prefix: Write the initial part of the text that provides sufficient context for the model.

3.Insert the Cloze: Leave a blank or specify a placeholder where the model should provide the missing information.

4.Refine the Prompt: Ensure the prompt is clear and concise to guide the model effectively.

Structure of Cloze Prefix Prompts

1.Prefix: The beginning portion of the text, which sets up the context or the start of the sentence.

2.Cloze: The blank or missing part that the model needs to fill in.

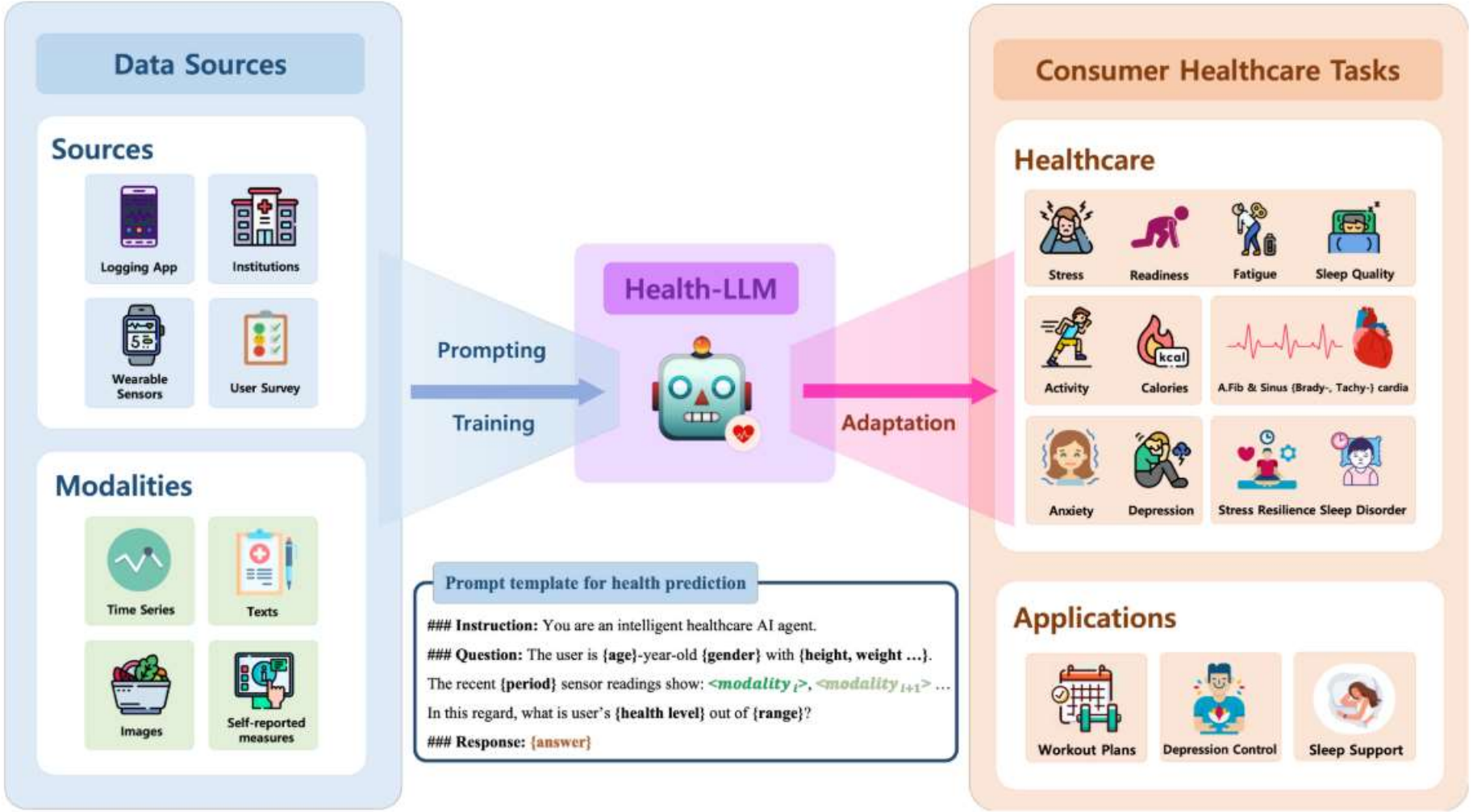


Figure 1: **Health-LLM.** We presents a framework for evaluating LLM performance on a diverse set of health prediction tasks, training and prompting the models with multi-modal health data.

healthcare

Topic	Dataset	Task	Metric	Prompt	Target
MHealth	PMData	Stress Prediction	MAE ↓	{Target} refers to [Health Knowledge]*. Given the [User Info]*, and [Period] sequence of Steps: [Steps], Calories Burn: [Calories], Resting Heart Rate: [RHR], Sleep Duration: [SleepMinutes], Mood: [Mood]. What will my stress level be?	[Stress]
	LifeSnaps	Stress Resilence Prediction	MAE ↓	[Target] refers to [Health Knowledge]*. Given the [User Info]* and following [Period] sequence of data, predict the Stress Resilience Index. Stress Score: [StressScore], Positive Affect Score: [PosAffectScore], Negative Affect Score: [NegAffectScore], Lightly Active Minutes: [Duration], Moderately Active Minutes: [Duration], Very Active Minutes: [Duration], Sleep Efficiency: [SleepEfficiency], Sleep Deep Ratio: [SleepDeepRatio], Sleep Light Ratio: [SleepLightRatio], Sleep REM Ratio: [SleepREMRatio].	[Stress Resilience]
	GLOBEM	Estimate of PHQ4 Depression Estimate of PHQ4 Anxiety	MAE ↓ MAE ↓	[Target] refers to [Health Knowledge]*. Steps during last [Period] sequence of maximum, minimum, average, median, standard deviation daily step count were [ListOfSteps] respectively. Sleep during last [Period] sequence of sleep efficiency, duration the user stayed in bed after waking up, duration the user spent to fall asleep, duration the user stayed awake but still in bed, duration the user spent to fall asleep are [ListOfDurations] in average. In this regard, what would be [Target]?"	[PHQ]

PARAMETER EFFICIENT FINE TUNING (PEFT)

- **Additive**
  - Adapters
  - Sparse Adapters
  - IA3
  - Soft prompt
    - Prompt Tuning
    - Prefix Tuning
    - P-Tuning
    - LlaMA Adapter
- **Selective**
  - BitFit
  - Freeze and Reconfigure
- **Reparameterization** (LORA,QLORA)
- **Hybrid method** that is a combination of Reparameterization and Selective

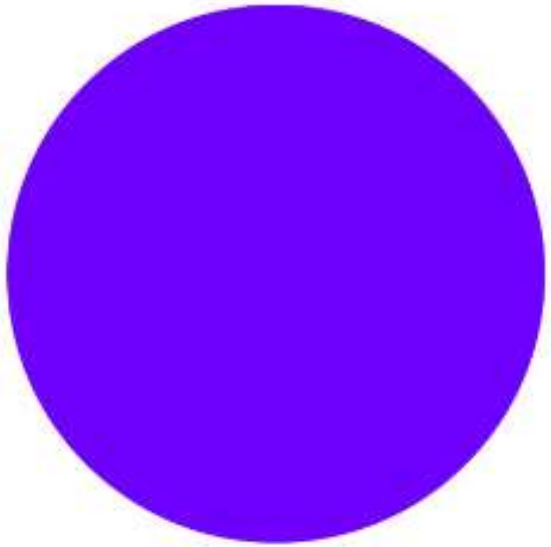
Courtesy:, deeplearning.ai ,

Model sizes are still growing (?)

Publically available model sizes: 350M → 176B
Single-GPU RAM: 16Gb → 80Gb

Model size scales almost **two orders of magnitude** quicker than single-GPU memory

3



4

Before, we were worried we cannot pre-train models.
Now, can we even fine-tune them?

Large Language Models (LLMs) are quite large by name. These models usually have anywhere from **7 to 70 billion parameters.** To load a 70 billion parameter model in full precision would require **280 GB of GPU memory**! To train that model you would update billions of tokens over millions or billions of documents. The computation required is substantial for updating those parameters. The self-supervised training of these models is expensive, **costing companies up to $100 million**.

Parameter-Efficient Fine-Tuning (PEFT) can significantly help in adapting large language models (LLMs) for various tasks while overcoming limitations associated with traditional fine-tuning methods

Transformer - recap

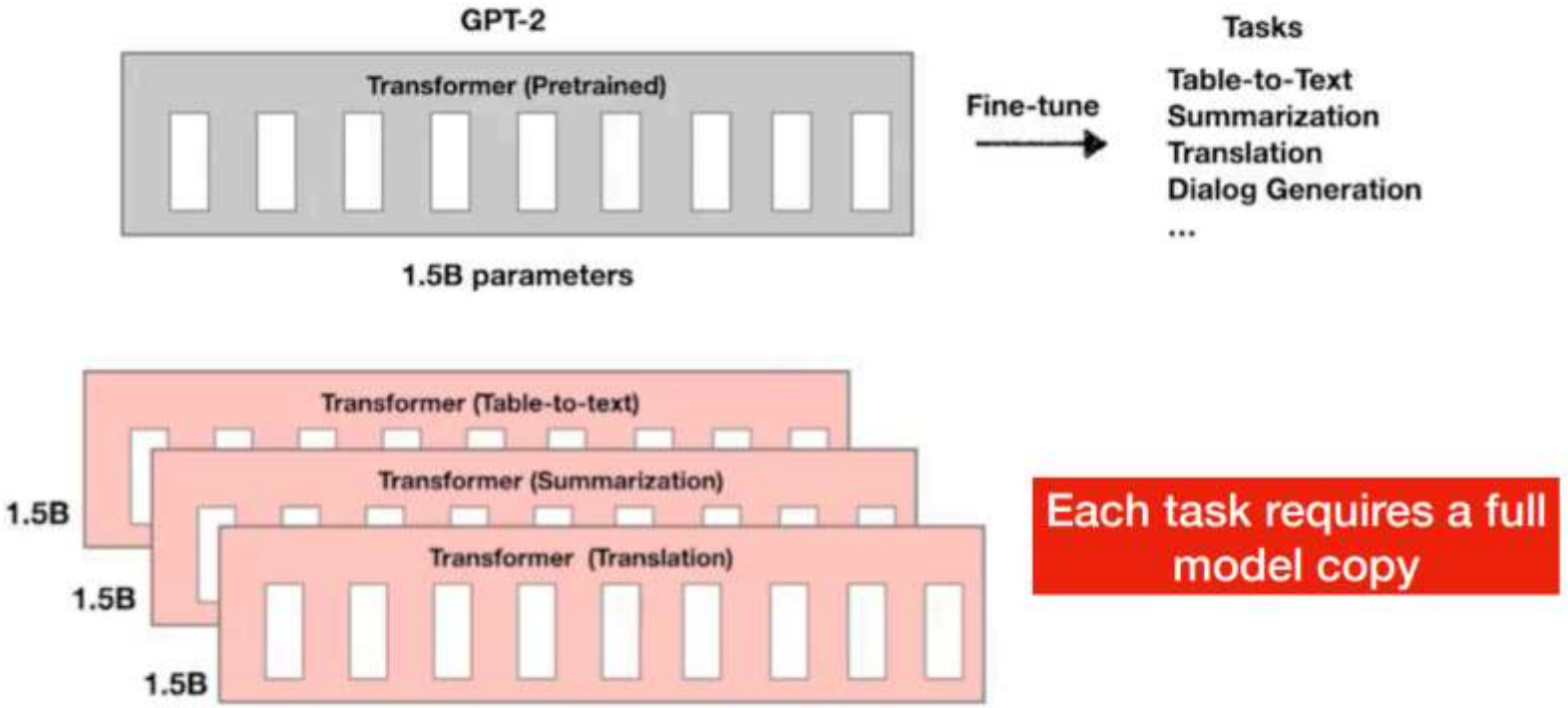
```
def self_attention(x):
    k = x @ W_k
    q = x @ W_q
    v = x @ W_v
    return softmax(q @ k.T) @ v

def transformer_block(x):
    """ Pseudo code by author based on [2] """
    residual = x
    x = self_attention(x)
    x = layer_norm(x + residual)
    residual = x
    x = FFN(x)
    x = layer_norm(x + residual)
    return x
```

Does this work? Yes!

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Parameter efficient Fine-tuning (PEFT)



- Fine-tuning a model requires to store as many parameters as the original model
  - High storage complexity
- PEFT: An approach for finetuning large language models in a parameter-efficient manner

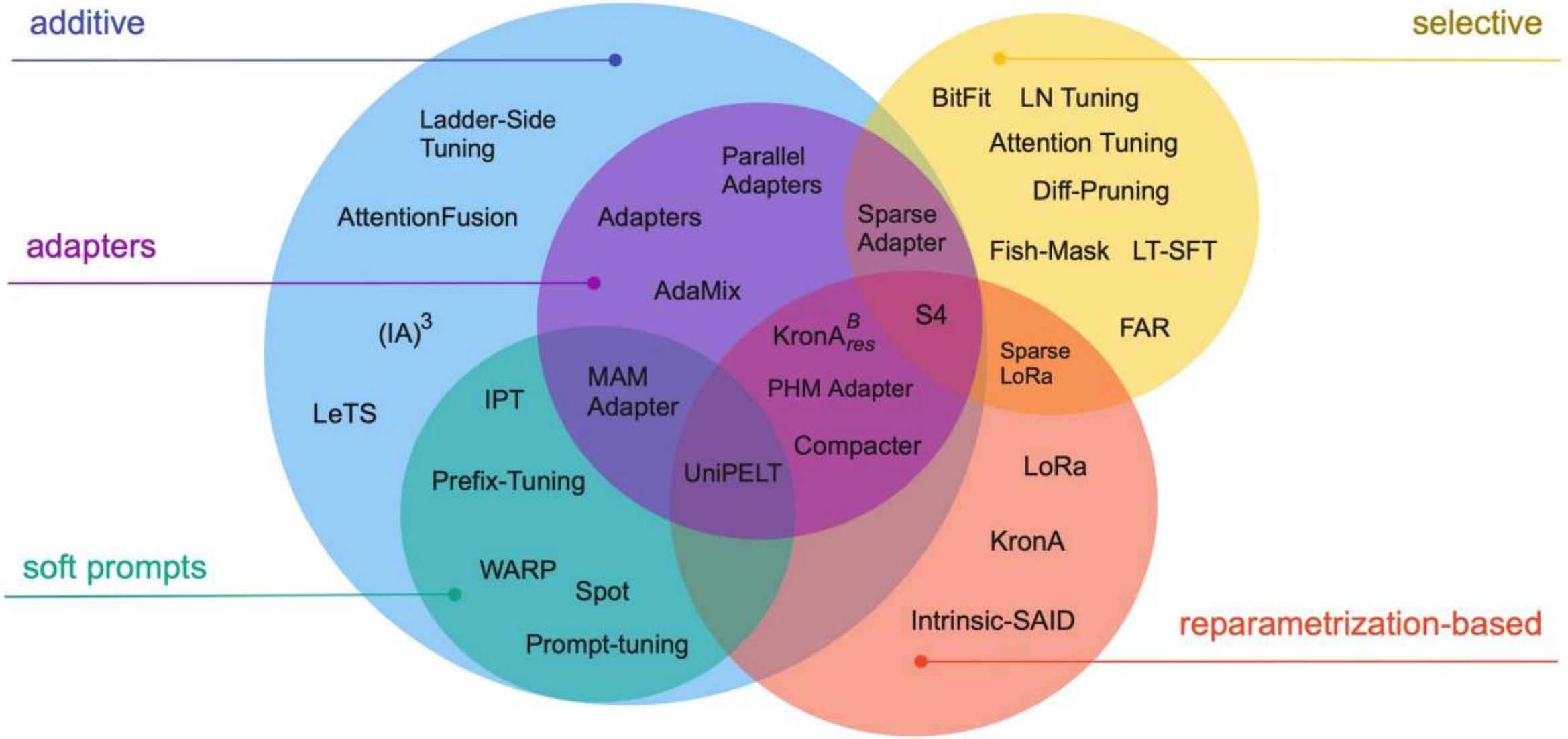
Source: peft (anoopsarkar.github.io)

- PEFT – classifications
  - Does the method introduce new parameters to the model?
  - Does it fine-tune a small subset of existing parameters?
  - Does the method aim to minimize memory footprint or storage efficiency?

- **Additive methods**
- **Selective methods**
- **Reparametrization-based methods**
- **Hybrid methods**

Lialin, V., Deshpande, V., & Rumshisky, A. (2023). Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

PEFT – classifications



Lialin, V., Deshpande, V., & Rumshisky, A. (2023). Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

Additive Method

Additive methods are probably the easiest to grasp.

The goal of additive methods is to **add an additional set of parameters or network layers** to augment the model.

When fine-tuning the data you **update the weights only of these newly added parameters**.

This makes training computationally easier and also adapts to smaller datasets

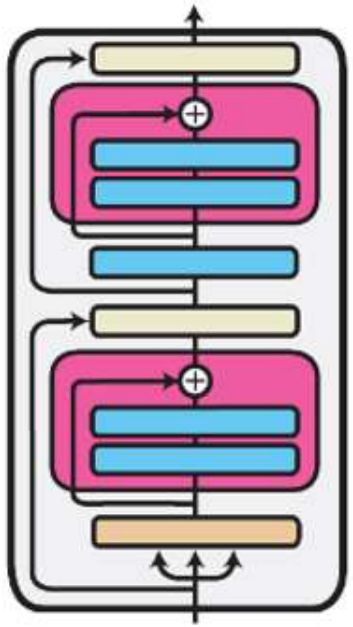
Additive Methods

- Key idea: **Augment the existing pre-trained model with additional parameters or layers and train only the added parameters**
- Introduce additional parameters. However, achieve significant training time and memory efficiency improvements
- Two approaches
  - Adapters
  - Soft prompts

Adapters

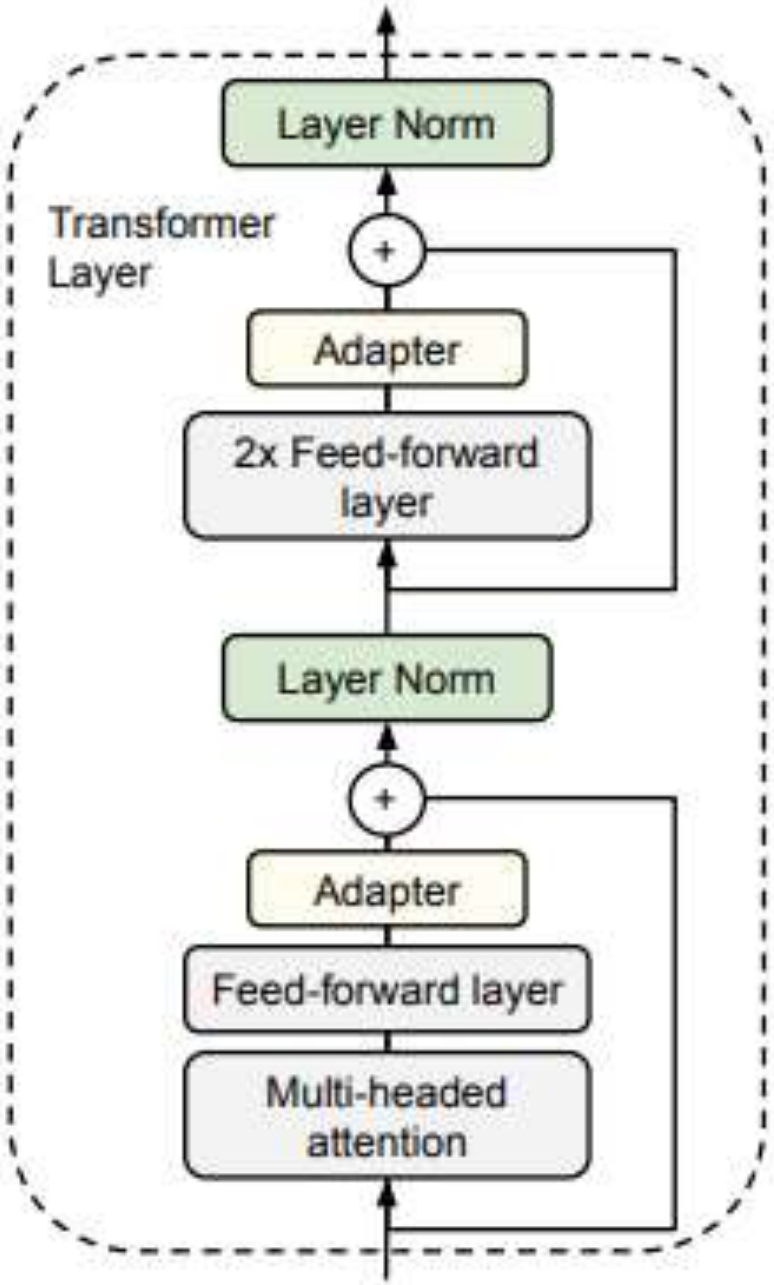
This technique was introduced in Houlsby et al [4]. The goal of adapters is to add small fully connected networks after Transformer sub-layers and learn those parameters

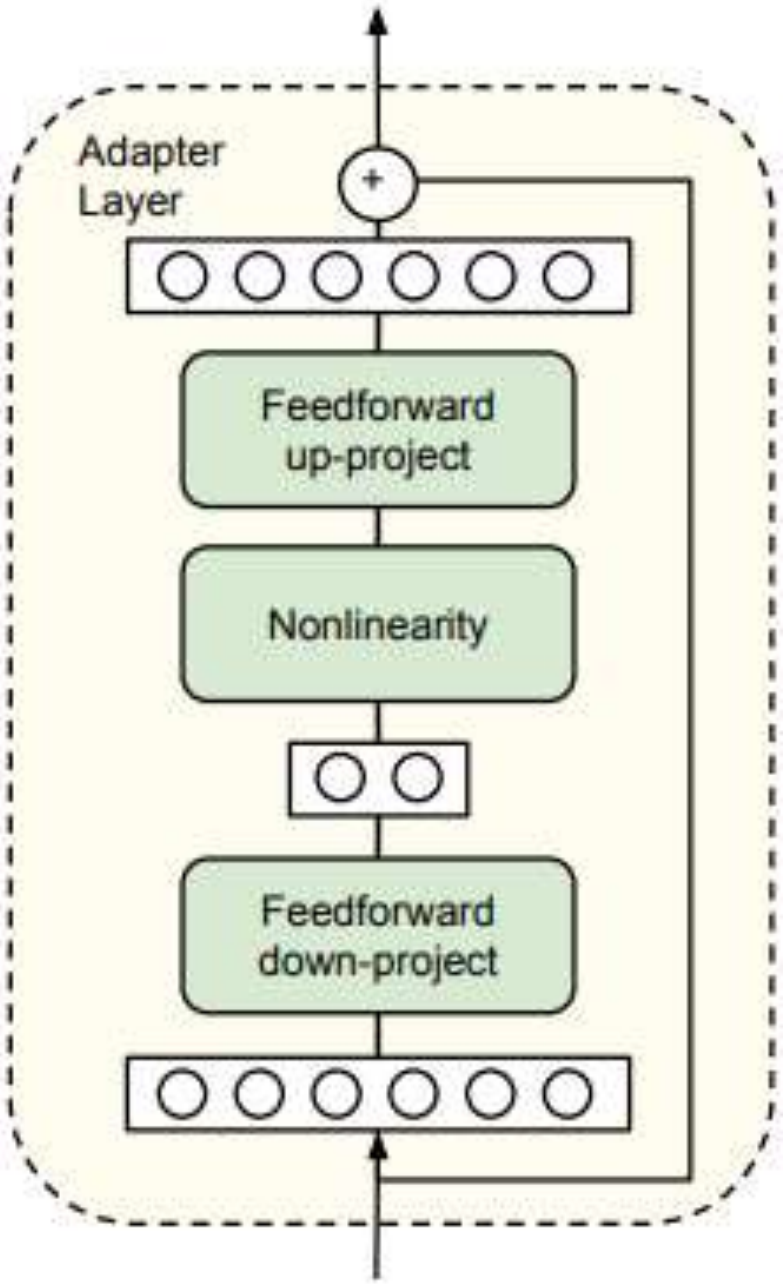
```
def transformer_block_adapter(x):
    """Pseudo code from [2] """
    residual = x
    x = self_attention(x)
    x = FFN(x)  # adapter
    x = layer_norm(x + residual)
    residual = x
    x = FFN(x)
    x = FFN(x)  # adapter
    x = layer_norm(x + residual)
    return x
```



https://arxiv.org/abs/1902.00751

Image source: adapterhub.ml





Source: Houlsby, Neil, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. "Parameter-efficient transfer learning for NLP." In *International Conference on Machine Learning*, pp. 2790-2799. PMLR, 2019.

Adapters

- Attach a small fully connected layer at every layer of the transformer
  - Inserts small modules (adapters) between transformer layers
- Adapter layer performs a down projection to project the input hidden layer information to a lower-dimensional space, followed by a non-linear activation function and an up projection
- A residual connection to generate the final form
- Only adapter layers are trainable, while the parameters of the original LLMs are frozen

$$h = h + f\left(hW_{down}\right)W_{up}$$

$$W_{up} \in \square^{r \times d} \qquad W_{down} \in \square^{d \times r}$$

Down projection: This is a linear transformation that reduces the dimensionality of the input hidden layer information (h) from the original LLM. Imagine squeezing a high-dimensional vector into a lower-dimensional one, focusing on the most relevant aspects for the specific task.

Let h be the input vector representing the hidden layer information from the original LLM (usually a high-dimensional vector).

We define a down-projection matrix (W_down) with dimensions appropriate for the desired reduction in dimensionality. This matrix essentially "compresses" the information.

The down projection is calculated by multiplying the input vector h with the down-projection matrix W_down:

z = W_down * h

Here, z represents the lower-dimensional representation of the input data.

Up projection: After processing in the lower-dimensional space, the data (z) is projected back to its original size (h'). This might seem redundant, but it allows the model to integrate the learned information from the lower-dimensional space back into the original context, enriching the representation.

Up Projection:

After processing in the lower-dimensional space, we want to project the data back to its original size.

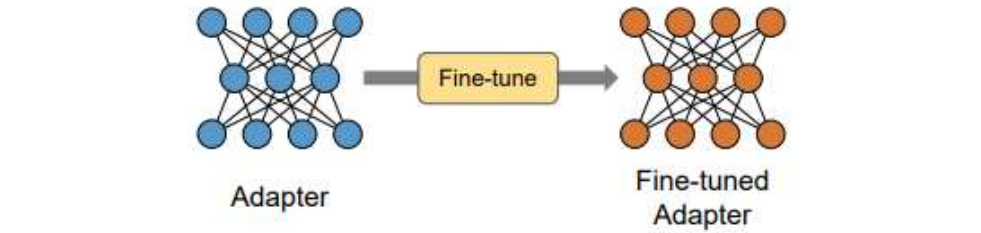
We define an up-projection matrix (W_up) with dimensions that allow us to expand the data back to the original size of the input vector h.

The up projection is calculated by multiplying the lower-dimensional vector z with the up-projection matrix W_up:

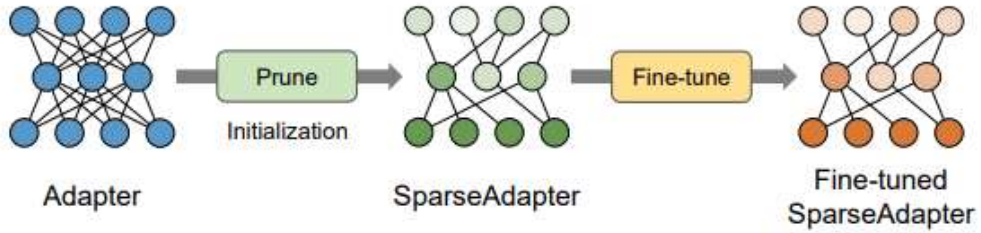
$$h' = W_{up} * z$$

Here, h' represents the data projected back to its original dimensionality.

- Sparse Adapter
  - Pruned adapters
  - Reduces the model size of neural networks by pruning redundant parameters and training the rest ones



(a) Standard Adapter Tuning.



(b) SparseAdapter Tuning.

Source:He, Shwai, Liang Ding, Daize Dong, Miao Zhang, and Dacheng Tao. "Sparseadapter: An easy approach for improving the parameter-efficiency of adapters." *arXiv preprint arXiv:2210.04284* (2022).

Implementing **sparse adaptation** involves incorporating sparsity techniques within the adapter layers of a PEFT (Parameter-Efficient Fine-Tuning) framework for large language models (LLMs).

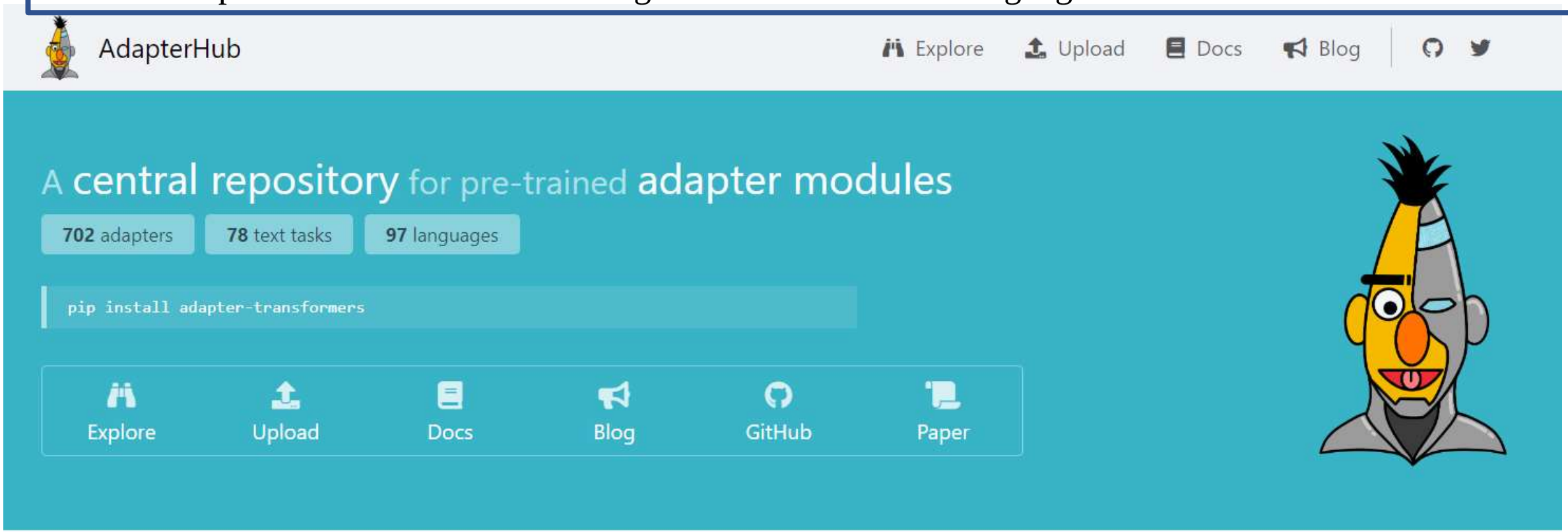
Choosing a Sparsity Technique:

There are several ways to introduce sparsity in adapter layers. Here are two common approaches:

- **Pruning:**
  - Start with a fully connected adapter layer (all weights have non-zero values).
  - During or after training, iteratively remove connections (set their weights to zero) considered unimportant based on specific criteria.
  - Common pruning algorithms include:
    - **Magnitude-based pruning:** Removes weights with values below a certain threshold.
    - **Gradient-based pruning:** Removes weights that contribute minimally to the gradient during training.
- **Magnitude Thresholding:**
  - Initialize all weights in the adapter layer with random values.
  - After training, set weights with absolute values below a certain threshold to zero, effectively making them inactive.

- AdapterHub
  - An easy-to-use and extensible adapter training and sharing framework for transformer-based model

AdapterHub is a framework designed to simplify the process of integrating, training, and using adapter modules for parameter-efficient fine-tuning of transformer-based language models.



Additive PEFT (IA3)- **Infused Adapter by Inhibiting and Amplifying Inner Activations**

Let's consider the scaled dot-product attention found in a normal transformer:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

we are working with an additive method, we are seeking to add parameters to this network. We want the dimensionality to be quite small. $(IA)^3$ proposes the following **new vectors to be added to the attention mechanism:**

- We just added column vectors l_k and l_v and take the Hadamard product between the column vector and the matrix (multiply the column vector against all columns of the matrix).

$$\text{softmax}\left(\frac{Q(l_k \odot K^T)}{\sqrt{d_k}}\right)(l_v \odot V)$$

We also introduce one other learnable column vector l_{ff} that is added to the feed forward layers as follow:

$$(l_{ff} \odot \gamma(W_1 x))W_2$$

In this example, gamma is the activation function applied to the product between the weights and input

The Hadamard product ⊙,. Each element (c_ij) in the resulting matrix C is calculated by multiplying the corresponding elements (a_ij and b_ij) from matrices A and B at the same position (i, j).

Additive PEFT: (IA)3

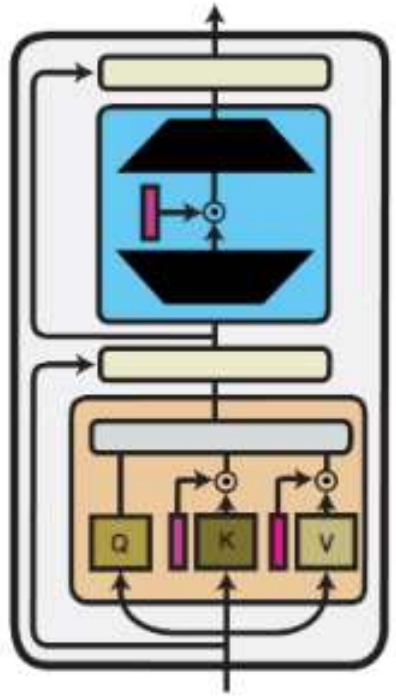
One-sentence idea:

Rescale key, value, and hidden FFN activations

Pseudocode:

```
def transformer_block_with_ia3(x):
    residual = x
    x = ia3_self_attention(x)
    x = LN(x + residual)
    residual = x
    x = x @ W_1           # FFN in
    x = l_ff * gelu(x)    # (IA)3 scaling
    x = x @ W_2           # FFN out
    x = LN(x + residual)
    return x

def ia3_self_attention(x):
    k, q, v = x @ W_k, x @ W_q, x @ W_v
    k = l_k * k
    v = l_v * v
    return softmax(q @ k.T) @ V
```



https://arxiv.org/abs/2205.05638

Image source: adapterhub.ml

Soft Prompting

With soft-prompts our goal is to add information to the base model that is more specific to our current task. With prompt tuning we accomplish this by creating a set of parameters for the **prompt tokens and injecting this at the beginning of the network**.

Soft-prompting is a technique that tries to avoid this dataset creation. In hard prompting, we are creating data in a discrete representation (picking words.) In soft-prompting, we seek a **continuous representation of the text we will input to the model**

Depending on the technique, there are different methods for how the information is added to the network. The core idea is that the base model does not optimize the text itself but rather the **continuous representation (i.e. some type of learnable tensor) of the prompt text.** This can be some form of embedding or some transformation applied to that embedding.

Prompt Tuning:

•**Trainable Prompt Parameters:** This approach focuses on creating a set of trainable parameters specifically for the prompt tokens. These parameters are essentially learned representations of the task that guide the LLM.

```
def prompt_tuning(seq_tokens, prompt_tokens):
    """ Pseudo code from [2]. """
    x = seq_embedding(seq_tokens)
    soft_prompt = prompt_embedding(prompt_tokens)
    model_input = concat([soft_prompt, x], dim=seq)
    return model(model_input)
```

Example sentiment Analysis

Imagine you want to fine-tune a pre-trained LLM for sentiment analysis. Here's how prompt tuning might work:

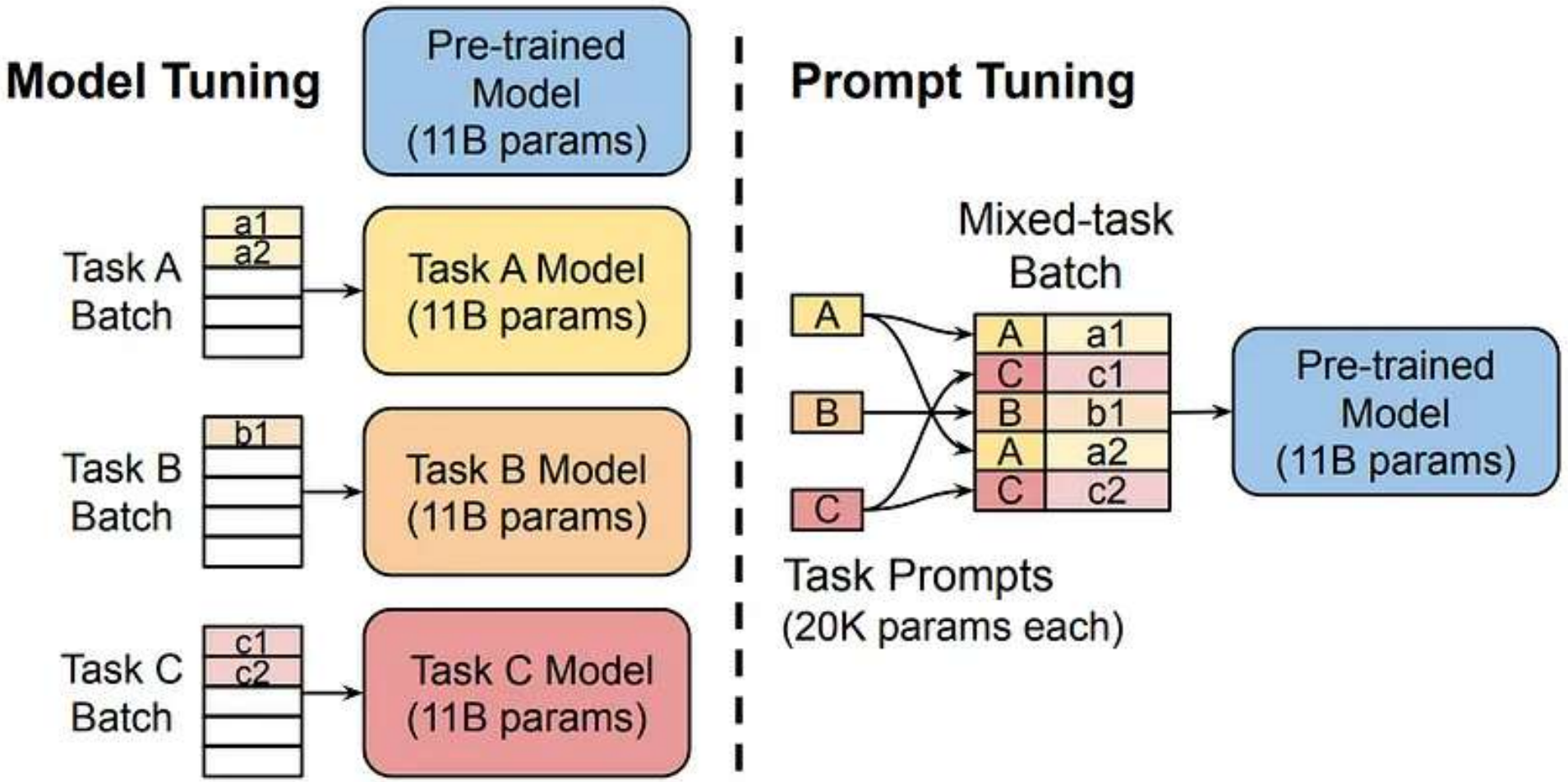
Define Prompt Template: You create a prompt template with special tokens representing the task and sentiment categories (positive, negative, neutral).

An example template could be:**[SENTIMENT] sentiment: __label__ , review: "This movie was fantastic!“**
[SENTIMENT] is a special token representing the task (sentiment analysis).__label__ is a placeholder that will be filled with the predicted sentiment during inference.“

This movie was fantastic!" is the actual review text you want the model to analyze.

Trainable Prompt Parameters: The model learns a set of parameters (embeddings) for each special token in the prompt template. These parameters capture the task-specific information.

Feeding the Input: During training and inference, you combine the prompt template with the actual review text, replacing __label__ with a special "unknown" token.


Model Tuning
Pre-trained Model (11B params)
Task A Batch
a1
a2
Task A Model (11B params)
Task B Batch
b1
Task B Model (11B params)
Task C Batch
c1
c2
Task C Model (11B params)
Prompt Tuning
Mixed-task Batch
A
B
C
A a1
C c1
B b1
A a2
C c2
Pre-trained Model (11B params)
Task Prompts
(20K params each)

Soft Prompt

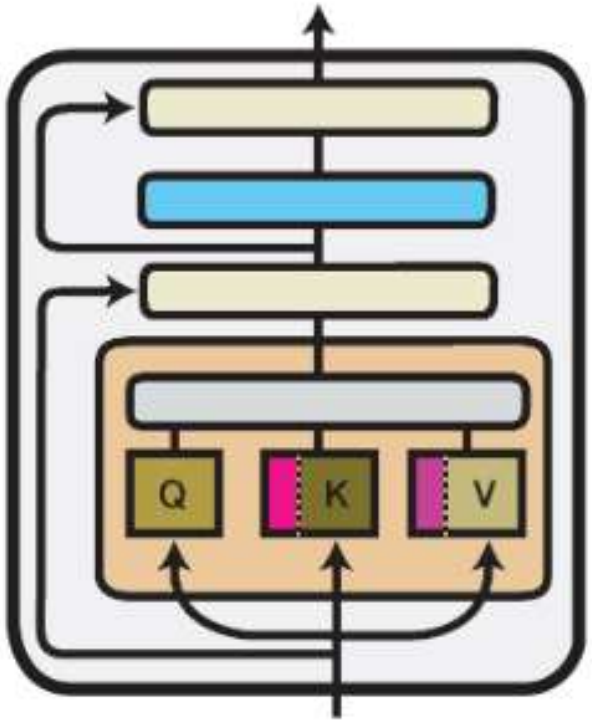
Additive PEFT: Prompt tuning

One-sentence idea:

Fine-tune part of your attention input – soft prompt

Pseudocode:

```
def prompt_tuning_attention(input_ids):
    q = x @ W_q
    k = cat([s_k, x]) @ W_k   # prepend a
    v = cat([s_v, x]) @ W_v   # soft prompt
    return softmax(q @ k.T) @ V
```



P-Tuning

Soft Prompting

P-Tuning is prompt-tuning but encoding the prompt using an LSTM.

Here prompt is a function that takes a context x and a target y and organizes itself into a template T. The authors provide the example sequence

“The capital of Britain is [MASK]”.

Here the prompt is “The capital of … is …”, the context is “Britain” and the target is [MASK]

. We can use this formulation to create two sequences of tokens, everything before the context and everything after the context before the target.

```
def p_tuning(seq_tokens, prompt_tokens):
    """Pseudo code for p-tuning created by Author."""
    h = prompt_embedding(prompt_tokens)
    h = LSMT(h, bidirectional=True)
    h = FFN(h)

    x = seq_embedding(seq_tokens)
    model_input = concat([h, x], dim=seq)

    return model(model_input)
```

LLaMA adapter

Soft Prompting

LLaMA adapter introduce Adaptation Prompts, which are soft-prompts appended with the input to the transformer layer. These adaption prompts are inserted in the L topmost of the N transformer layers.

With additive methods LLaMA adapter introduce a new set of parameters that have some random initialization over the weights. Because of this random noise added to the LM, we can potentially experience unstable fine-tuning which can cause a problem with large loss values at the early stages.

To solve this problem, the authors introduce a gating factor, initialized to 0, that is multiplied by the self attention mechanism. The product of the gating factor and self-attention is referred to as **zero-init attention**. The gating value is adaptively tuned over the training steps to create a smoother update of the network parameters.

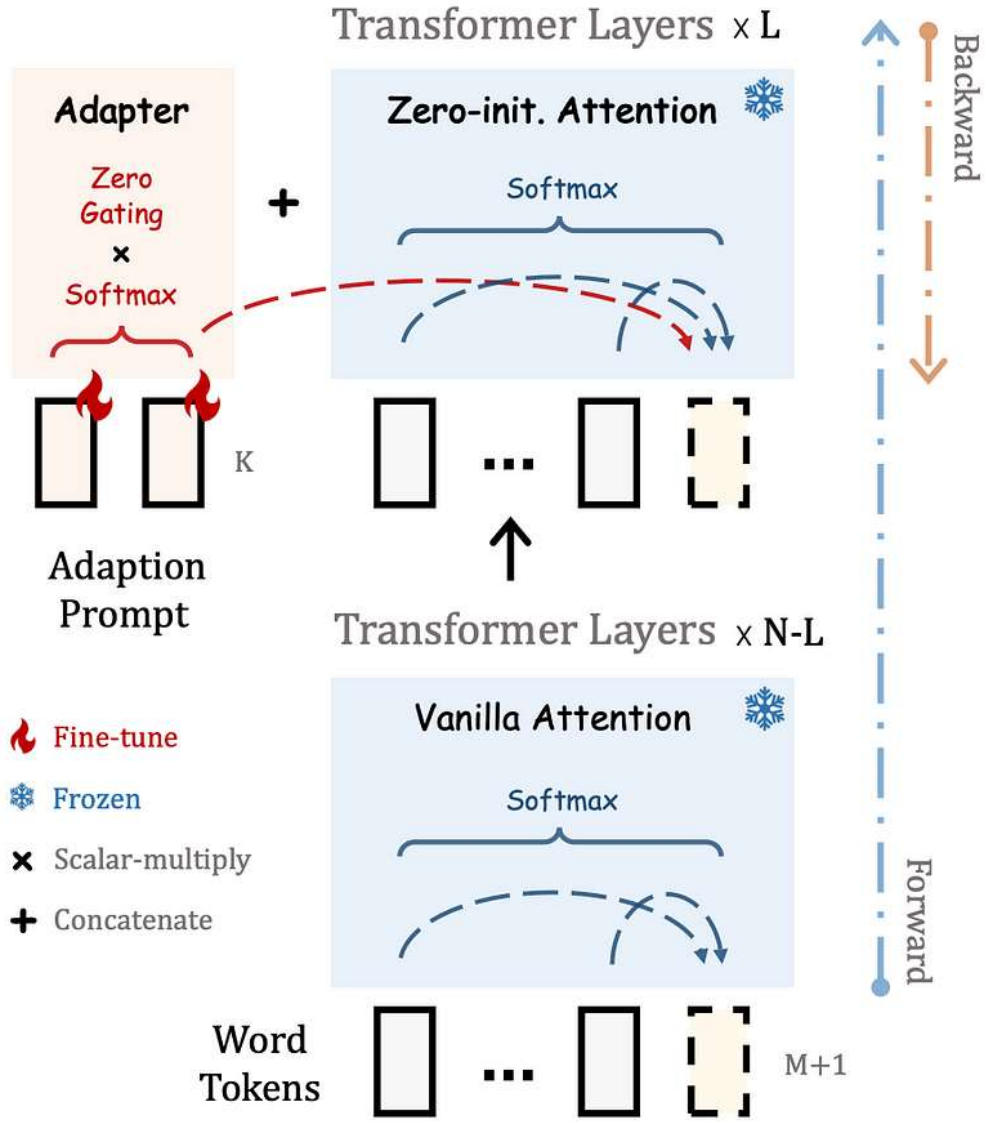
LLaMA adapter

```
def transformer_block_llama_adapter(x, soft_prompt, gating_factor):
    """LLaMA-Adapter pseudo code created by Author"""
    residual = x

    adaption_prompt = concat([soft_prompt, x], dim=seq)
    adaption_prompt = self_attention(adaption_prompt) * gating_factor  # ze

    x = self_attention(x)
    x = adaption_prompt * x
    x = layer_norm(x + residual)
    residual = x
    x = FFN(x)
    x = layer_norm(x + residual)

    return x
```



Selective Methods

- Selectively fine-tuning some parts of the network
  - Fine-tuning **only a few top layers** of a network
  - Based on the **type** of the layer
  - Internal structure of the network (tuning **only model biases**)
  - Tuning only **particular rows of the parameter matrix**
  - **Sparsity-**based approaches

Selective PEFT: BitFit

One-sentence idea:

Fine-tune only model biases

Pseudocode:

```
params = (p for n, p
          in model.named_parameters()
          if "bias" in n)
optimizer = Optimizer(params)
```

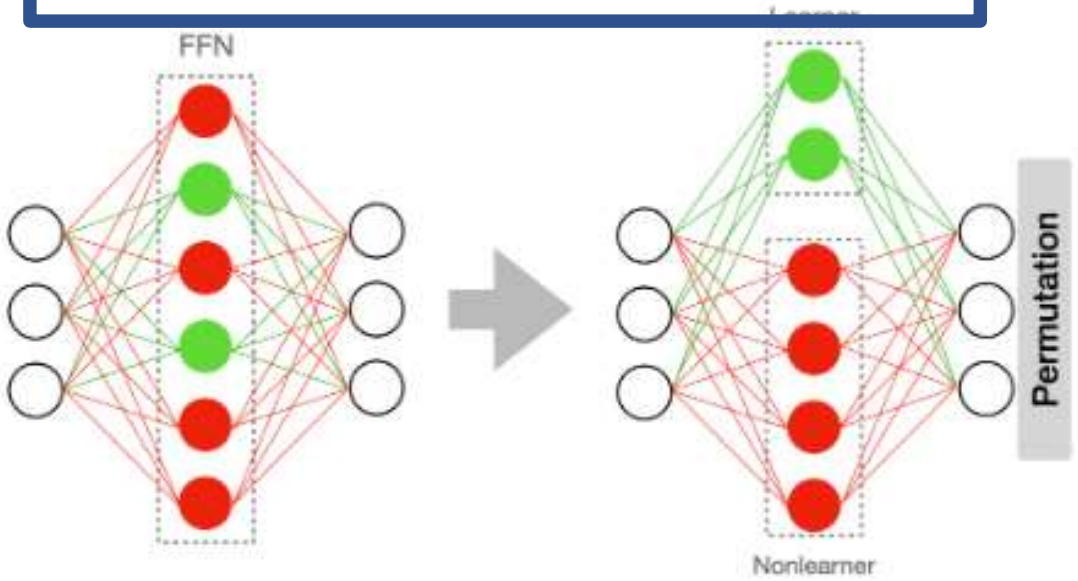
Selective PEFT: Freeze and Reconfigure

One-sentence idea:

Selects columns of parameter matrices to train and reconfigures linear layers into trainable and frozen

```
def far_layer(x):
    h1 = x @ W_t
    h2 = x @ W_f
    return concat([h1, h2], dim=-1)
```

Split your linear layer into 2 matrices based on information measure or gradient measure- Frozen matrix and Trainable matrix



Freezing: This involves keeping certain layers of the pre-trained LLM's architecture unchanged during fine-tuning. These frozen layers typically represent **the LLM's core capabilities for understanding language**.
Reconfiguring: Specific parts of the LLM, particularly the later layers, are reconfigured to **adapt to the target task**

Reparametrization-based methods- LoRA

Courtesy:, deeplearning.ai ,

Reparametrization-based methods

Reparameterization refers to transforming or reformulating the parameters of a model in a way that simplifies certain aspects of the training or optimization process.
This can involve:

- **Simplifying the Optimization Landscape**: Making the parameter space easier to navigate during training.
- **Improving Efficiency**: Reducing the number of parameters or computations required.
- **Introducing Constraints**: Incorporating prior knowledge or constraints more naturally into the model.

- Leverage low-rank representations to minimize the number of trainable parameters
- Basic intuition: neural networks have low dimensional representations
- Common approaches
  - Intrinsic SAID
  - LoRA
  - QLoRA

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Abstract

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Lo**w-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.

Low-Rank Adaptation, or LoRA

Low-Rank Adaptation, or LoRA, **freezes the pretrained model weights** and **injects trainable rank decomposition matrices** into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks

- Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times.
- LoRA performs on-par or better than finetuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters,
- a higher training throughput, and, unlike adapters, no additional inference latency*

*Inference latency refers to the time it takes for a model to process an input and produce an output. It's a crucial factor in real-time applications where quick responses are necessary

Rank of a Matrix

The rank of the matrix gives the number of linearly independent column vectors of the matrix and this number also means the dimension of the linear space these vectors span. For example:

For example:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This matrix has 3 column vectors which span the 3 dimensional space, they are it's trivial base vectors, so they are linearly independent...

Row-Echelon Form *: A non-zero matrix is said to be in a row-echelon form if all zero rows occur as bottom rows of the matrix and if the first non-zero element in any lower row occurs to the right of the first non-zero entry in the higher row.*

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 5 \\ 0 & 0 & 0 \end{bmatrix}$$

- Rank of a matrix is equal to the number of non-zero rows if it is in Echelon Form.

There are many equivalent definitions of the rank of a matrix A.

The following two conditions are equivalent to each other

1. The largest linearly independent subset of columns of A has size k. That is, all n columns of A arise as linear combinations of only k of them.
2. The largest linearly independent subset of rows of A has size k. That is, all m rows of A arise as linear combinations of only k of them.

if $A = YZ^T$, then all of A's columns are linear combinations of the k columns of Y, and all of A's rows are linear combinations of the k rows of Z^T

The "rank" of a matrix is the dimension of that space spanned by the vectors it contains.

If we $\begin{bmatrix} 7 & -2 \\ 3 & -1 \\ 5 & 5 \end{bmatrix}$ vectors [7, 3, 5] and [-2, -1, 5] into a matrix:

the rank of the matrix is the dimension of the space that you get by taking all combinations of the vectors. We've already done that, and saw that the space spanned by [7, 3, 5] and [-2, -1, 5] was a plane. In this case, the rank is 2 (because a plane is 2 dimensional).

Understanding Rank

In linear algebra, the rank of a matrix is a measure of the number of linearly independent rows or columns in the matrix. For a matrix $M \in \mathbb{R}^{m \times n}$:

- **Full Rank:** The matrix has full rank if its rank is equal to the smaller of the number of rows or columns, i.e., $\text{rank}(M) = \min(m, n)$.
- **Low Rank:** A matrix is considered low-rank if its rank is significantly less than the smaller dimension, i.e., $\text{rank}(M) \ll \min(m, n)$.

In practical terms, a low-rank matrix can be approximated by the product of two smaller matrices. For example, a matrix M of size $m \times n$ with rank r can be approximated as:

$$M \approx AB$$

where $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$.

Full-Rank Matrix Example

Consider the following 3x3 matrix W:

$$W = \begin{pmatrix} 4 & 2 & 3 \\ 3 & 1 & 4 \\ 2 & 1 & 3 \end{pmatrix}$$

This matrix is full-rank if its rank is 3, which is the maximum for a 3x3 matrix.

Low-Rank Matrix Example

A low-rank matrix is one whose rank is less than the maximum possible. For example, consider the matrix M:

$$M = \begin{pmatrix} 2 & 4 & 4 \\ 1 & 2 & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

This matrix is rank-1 (it has linearly dependent rows/columns), which is less than its dimension of 3.

Low-Rank Decomposition

Given a matrix W, we want to decompose it into two matrices A and B such that $W \approx AB$ where A and B have a smaller rank.

For our full-rank matrix W:

$$W = \begin{pmatrix} 4 & 2 & 3 \\ 3 & 1 & 4 \\ 2 & 1 & 3 \end{pmatrix}$$

Let's decompose W into A and B with a rank of 2.

Thus, the low-rank approximation of W is:

$$W \approx AB = \begin{pmatrix} -4.6 & -0.65 \\ -4.3 & 0.88 \\ -3.5 & -0.14 \end{pmatrix} \begin{pmatrix} -0.57 & -0.30 & -0.77 \\ -0.73 & 0.65 & 0.22 \end{pmatrix}$$

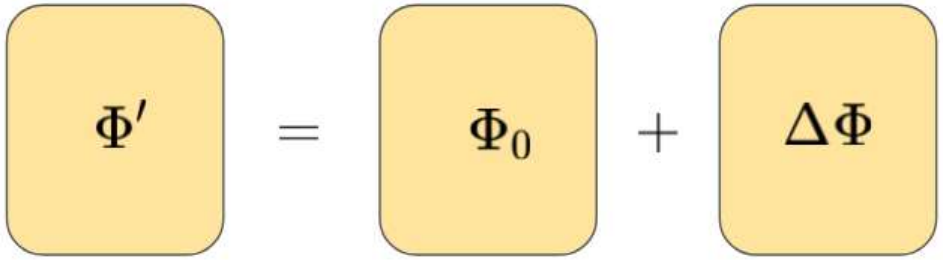
Verification

Multiplying A and B should yield an approximation of W:

$$AB = \begin{pmatrix} -4.6 & -0.65 \\ -4.3 & 0.88 \\ -3.5 & -0.14 \end{pmatrix} \begin{pmatrix} -0.57 & -0.30 & -0.77 \\ -0.73 & 0.65 & 0.22 \end{pmatrix} \approx \begin{pmatrix} 4.00 & 2.01 & 3.01 \\ 3.01 & 0.98 & 4.02 \\ 2.01 & 1.00 & 3.01 \end{pmatrix}$$

The resulting matrix is a close approximation of the original W, demonstrating the effectiveness of the low-rank decomposition.

- LoRA: Low-Rank Adaptation of Large Language Models
  - Try to achieve a **small number of task-specific parameters**
  - While the weights of a pre-trained model have full rank on the pre-trained tasks, the LoRA authors point out that pre-trained large language models have a low "intrinsic dimension"* when they are adapted to a new task



Decompose the weight changes, ΔW, into a lower-rank representation

*The intrinsic dimension refers to the minimum number of parameters required to effectively represent the information in a model.

Original Dense Layer

Consider a dense layer in a neural network with weights $W \in \mathbb{R}^{d \times d}$, where d is the dimension of the input and output. The operation performed by this layer is typically:

$$Y = WX + b$$

where:

- $X \in \mathbb{R}^{d \times n}$ is the input to the layer.
- $Y \in \mathbb{R}^{d \times n}$ is the output of the layer.
- $b \in \mathbb{R}^{d}$ is the bias term (which we can ignore for simplicity in our explanation).

Low-Rank Adaptation (LoRA)

Instead of updating W directly during fine-tuning, LoRA proposes to represent the change in W as a low-rank update. Specifically:

$$W' = W + \Delta W$$

where $\Delta W \approx AB$ with:

- $A \in \mathbb{R}^{d \times r}$
- $B \in \mathbb{R}^{r \times d}$
- r is the rank, which is much smaller than d.

So the new weight matrix after adaptation becomes:

$$W' = W + AB$$

Training Process

During training, instead of optimizing W' directly, we optimize the low-rank matrices A and B. The adaptation process can be described as follows:

1. **Forward Pass:** Compute the output using the modified weights.
   $$Y = (W + AB)X$$
2. **Backward Pass:** Compute gradients with respect to A and B and update these matrices accordingly.

Full Fine-Tuning

1. **Initialization:**
    - The model starts with pre-trained weights Φ_0.
2. **Updating Weights:**
    - During fine-tuning, the weights are updated to $\Phi_0 + \Delta\Phi$.
    - This update process involves adjusting the weights to better fit the new task's data.

Full Fine-Tuning

3. **Objective Function:**

- The goal is to maximize the conditional language modeling objective, which can be written as:

$$\max_{\Phi} \sum_{(x,y)\in Z} \sum_{t=1}^{|y|} \log(P_\Phi(y_t|x, y_{<t}))$$

- Here, Z is the training dataset consisting of pairs (x, y), and the inner summation runs over the tokens in the output sequence y.
- $P_\Phi(y_t|x, y_{<t})$ is the probability of the token y_t given the input x and the previous tokens $y_{<t}$.

Z is context-target pairs

4. **Drawbacks:**

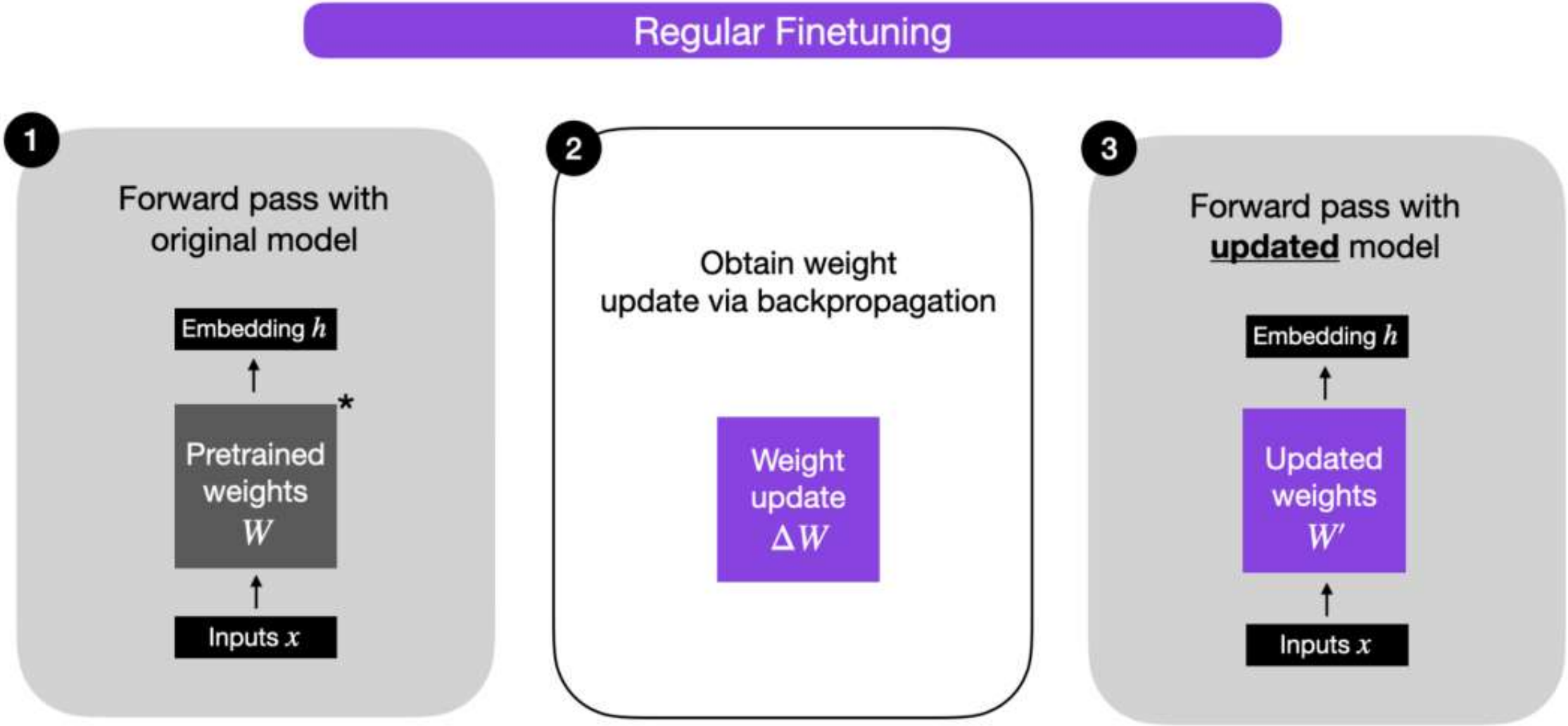
- Full fine-tuning requires learning a separate set of parameters $\Delta\Phi$ for each task.
- The size of $\Delta\Phi$ is the same as the original weights Φ_0, which can be very large (e.g., 175 billion parameters for GPT-3).
- Storing and deploying multiple fine-tuned models for different tasks is challenging due to the large size.

Parameter-Efficient Approach (LoRA)

1. **Reduced Parameter Set:**
   - Instead of learning a large $\Delta\Phi$ for each task, we encode the task-specific parameter increment $\Delta\Phi$ using a much smaller set of parameters Θ.
   - The size of Θ ($|\Theta|$) is much smaller than the size of the original weights ($|\Phi_0|$).
2. **Optimizing Over Θ:**
   - The task of finding $\Delta\Phi$ is now transformed into optimizing over the smaller set of parameters Θ:

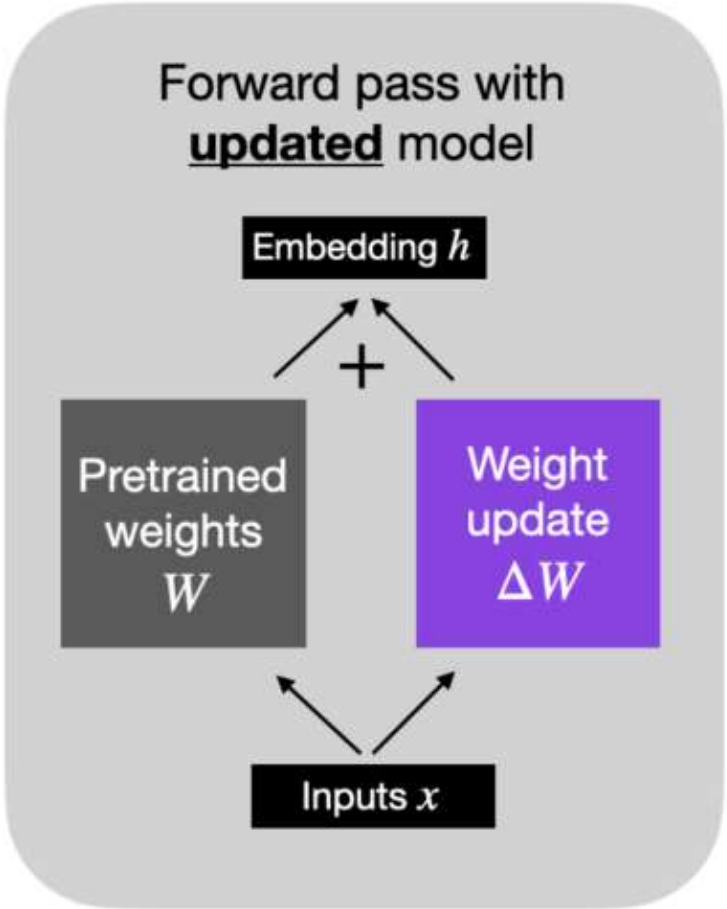
$$\max_{\Theta} \sum_{(x,y)\in Z} \sum_{t=1}^{|y|} \log(P_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t}))$$

3. **Low-Rank Representation:**
   - $\Delta\Phi$ is encoded using a low-rank representation, which makes it both compute- and memory-efficient.
   - For example, if the pre-trained model is GPT-3 with 175 billion parameters, the number of trainable parameters Θ can be as small as 0.01% of the original size.



people.cs.umass.edu/~miyyer/cs685/slides/multilingual.pdf

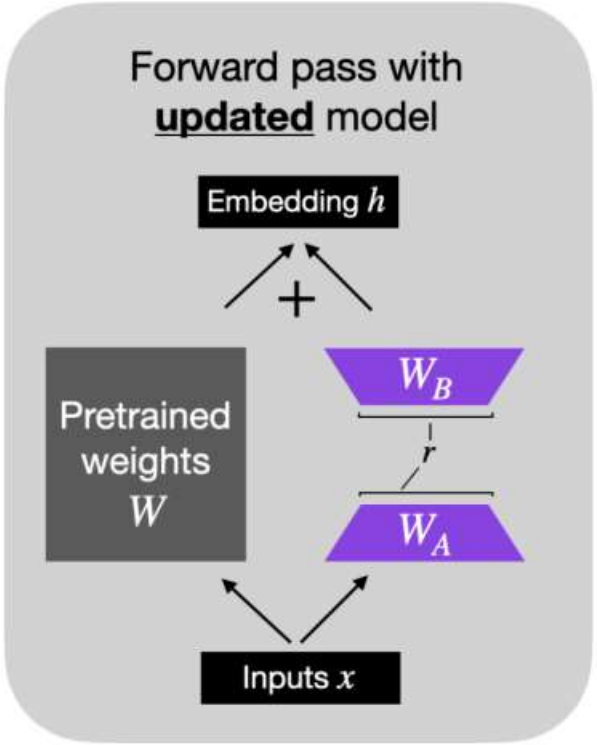
Alternative formulation (regular finetuning)

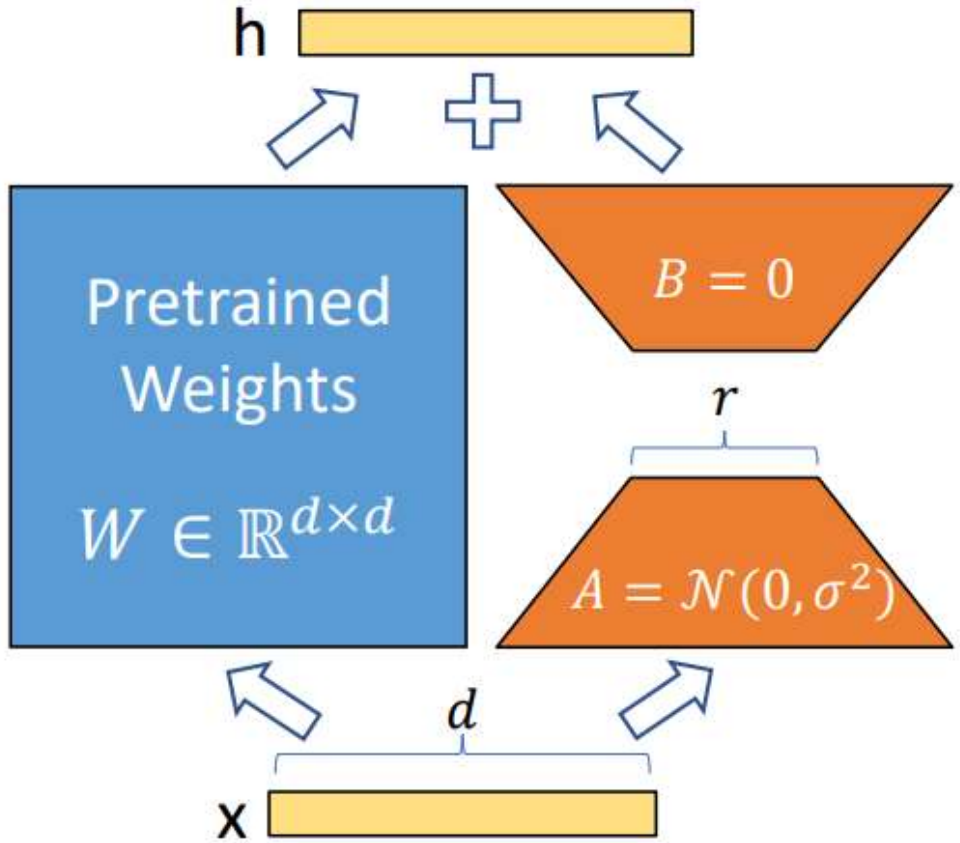


Pre-trained language models have a low intrinsic dimension

Possible to learn efficiently with a low-dimensional parameterization

LoRA weights, W_A and W_B, represent ΔW





Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

- The major downside of fine-tuning is that the new model contains as many parameters as in the original model

LoRA possesses several key advantages.

• A pre-trained model can be shared and used to build many small LoRA modules for different tasks. We can freeze the shared model and efficiently **switch tasks by replacing the matrices A and B** , reducing the storage requirement and task-switching overhead significantly.

• LoRA makes training more efficient and **lowers the hardware barrier** to entry by up to 3 times when using adaptive optimizers since we **do not need to calculate the gradients or maintain the optimizer states for most parameters**. Instead, we only optimize the injected, much smaller low-rank matrices.

• Simpler linear design allows us **to merge the trainable matrices with the frozen weights when deployed**, introducing no inference latency compared to a fully fine-tuned model, by construction.

• LoRA is orthogonal to many prior methods and **can be combined with many of them**, such as prefix-tuning

Intrinsic Dimension

Intrinsic Dimension

- **Intrinsic Dimension:** Refers to the minimal number of parameters or degrees of freedom required to effectively represent the essential information in a dataset or a model.
- **Full Dimension:** Refers to the actual number of parameters in the original matrix or model.

In the context of neural networks, even though the weight matrices have a high dimensionality, the actual effective transformations needed to adapt the model for specific tasks often lie in a lower-dimensional subspace. This is what we refer to as the intrinsic dimension.

LORA Aligns with Intrinsic Dimension

1. **Reduction to Essential Components:**
    - The low-rank approximation effectively captures the most significant components or directions of variation in the data or weight updates.
    - If W can be well-approximated by a rank-r decomposition, it suggests that the essential information in W is inherently low-dimensional.
2. **Parameter Efficiency:**
    - By using matrices A and B with dimensions $d \times r$ and $r \times d$ respectively, where $r \ll d$, we are significantly reducing the number of parameters.
    - This reduction is based on the premise that the updates needed for fine-tuning (captured by ΔW) lie within an r-dimensional subspace, which is the intrinsic dimension of the necessary adaptations.

The intrinsic dimension represents the minimal effective degrees of freedom.

By using a low-rank approximation, we are modeling the updates using only the essential parameters (aligned with the intrinsic dimension).

This approach avoids the redundancy of full-rank matrices and focuses on the core components that contribute most to the task-specific adaptation.

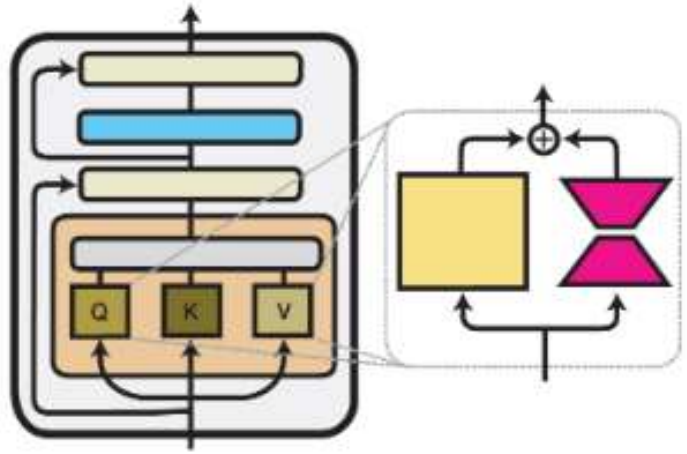
Reparametrization-based PEFT: LoRA

One-sentence idea:

Parameter update for a weight matrix in LoRA is decomposed into a product of two low-rank matrices

Pseudocode:

```
def lora_linear(x):
    h = x @ W # regular linear
    h += x @ W_A @ W_B # low-rank update
    return scale * h
```



https://arxiv.org/abs/2106.09685

Image source: adapterhub.ml

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

QLoRA

Courtesy:, deeplearning.ai , https://arxiv.org/pdf/2305.14314

what does Computation mean?

Computation refers to the **mathematical operations** that are performed on the weights and activations of the network during both the forward pass (when making predictions) and the backward pass (when updating the weights during training).

In a typical neural network, these computations are performed using **32-bit floating- point numbers**. This is because 32-bit floating-point numbers provide a good balance between **precision** (the ability to represent numbers accurately) and **range** (the range of numbers that can be represented). Using 32-bit floating-point numbers for all computations can be **memory-intensive.**

This is where quantization comes in.

Quantization is a technique to **reduce the precision** of the numbers used in the model. In the case of 4-bit quantization, the weights and activations of the network are compressed from 32-bit floating-point numbers to 4-bit integers. A 4-bit integer can range from -8 to 7.

QLoRA extends LoRA to enhance efficiency by quantizing weight values of the original network, from high-resolution data types, such as Float32, to lower-resolution data types like int4. This leads to reduced memory demands and faster calculations.

*Our best model family, which we name **Guanaco**, outperforms all previous openly released models on the Vicuna benchmark, reaching 99.3% of the performance level of ChatGPT while only requiring 24 hours of finetuning on a single GPU*

https://arxiv.org/pdf/2305.14314

QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers* **Artidoro Pagnoni*** **Ari Holtzman**

Luke Zettlemoyer

University of Washington
{dettmers,artidoro,ahai,lsz}@cs.washington.edu

Abstract

We present QLoRA, an efficient finetuning approach that reduces memory usage enough to finetune a 65B parameter model on a single 48GB GPU while preserving full 16-bit finetuning task performance. QLoRA backpropagates gradients through a frozen, 4-bit quantized pretrained language model into Low Rank Adapters (LoRA). Our best model family, which we name **Guanaco**, outperforms all previous openly released models on the Vicuna benchmark, reaching 99.3% of the performance level of ChatGPT while only requiring 24 hours of finetuning on a single GPU. QLoRA introduces a number of innovations to save memory without sacrificing performance: (a) 4-bit NormalFloat (NF4), a new data type that is information theoretically optimal for normally distributed weights (b) Double Quantization to reduce the average memory footprint by quantizing the quantization constants, and (c) Paged Optimizers to manage memory spikes. We use QLoRA to finetune more than 1,000 models, providing a detailed analysis of instruction following and chatbot performance across 8 instruction datasets, multiple model types (LLaMA, T5), and model scales that would be infeasible to run with regular finetuning (e.g. 33B and 65B parameter models). Our results show that QLoRA finetuning on a small high-quality dataset leads to state-of-the-art results, even when using smaller models than the previous SoTA. We provide a detailed analysis of chatbot performance based on both human and GPT-4 evaluations showing that GPT-4 evaluations are a cheap and reasonable alternative to human evaluation. Furthermore, we find that current chatbot benchmarks are not trustworthy to accurately evaluate the performance levels of chatbots. A lemon-picked analysis demonstrates where **Guanaco** fails compared to ChatGPT. We release all of our models and code, including CUDA kernels for 4-bit training.[2]

QLoRA

QLoRA is the extended version of LoRA which works by quantizing the precision of the weight parameters in the pre trained LLM to 4-bit precision. Typically, parameters of trained models are stored in a 32-bit format, but QLoRA compresses them to a 4-bit format. This reduces the memory footprint of the LLM, making it possible to finetune it on a single GPU. This method significantly **reduces the memory footprint, making it feasible to run LLM models on less powerful hardware, including consumer GPUs.**

According to QLoRA paper:

QLORA introduces **multiple innovations** designed to reduce memory use without sacrificing performance:

(1) 4-bit NormalFloat, an information theoretically optimal quantization data type for normally distributed data that yields better empirical results than 4-bit Integers and 4-bit Floats.

(2) Double Quantization, a method that quantizes the quantization constants, saving an average of about 0.37 bits per parameter (approximately 3 GB for a 65B model).

(3) Paged Optimizers, using NVIDIA unified memory to avoid the gradient checkpointing memory spikes that occur when processing a mini-batch with a long sequence length.

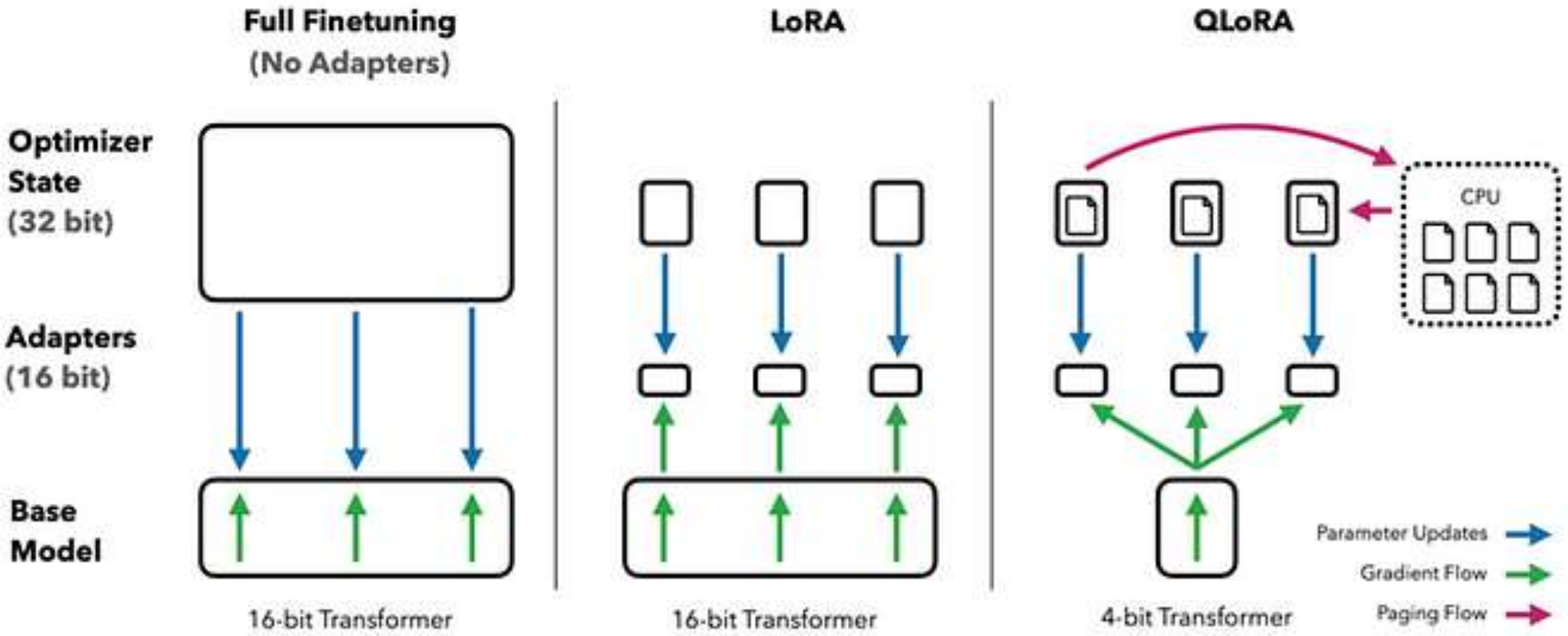


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

QLORA Finetuning

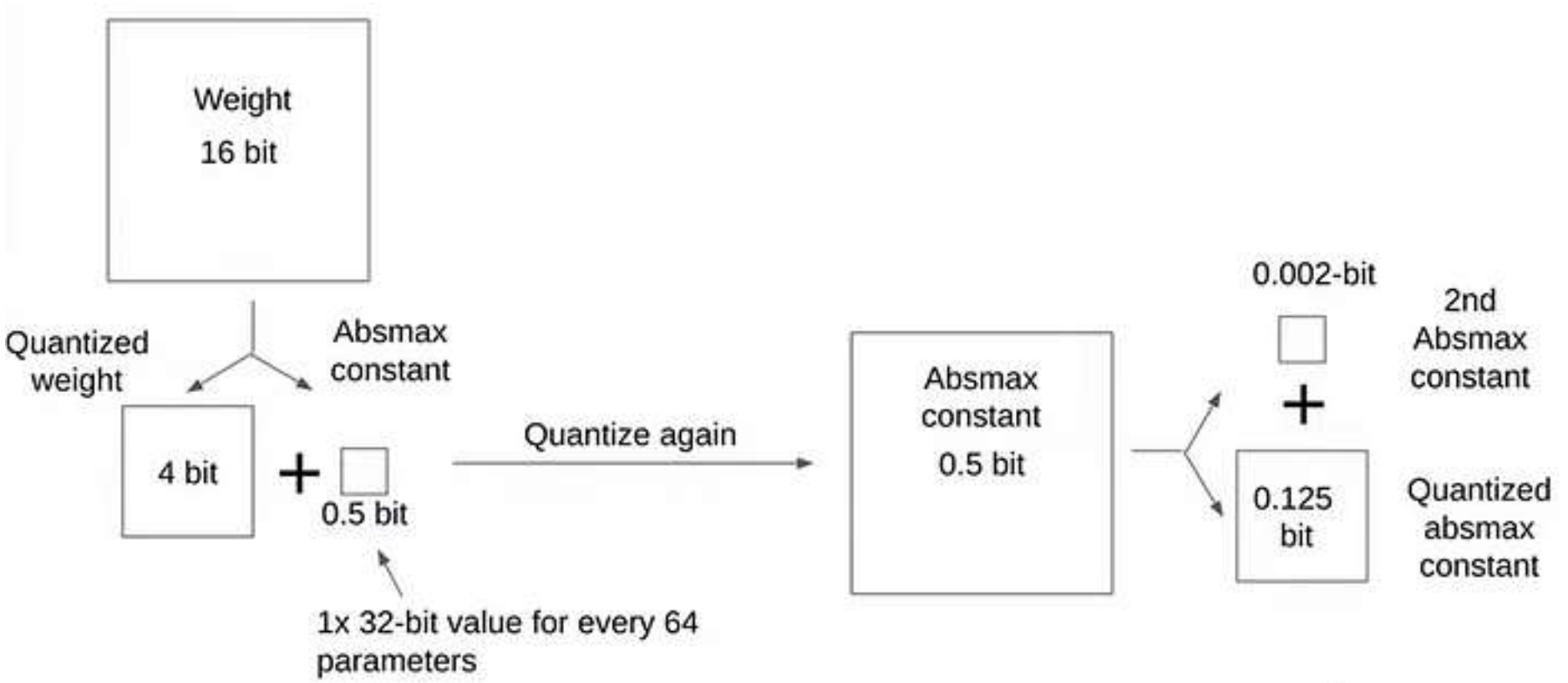


Image source: *Democratizing Foundation Models via k-bit Quantization by Tim Dettmers*

QLoRA Steps

- Normalisation
- Quantization
- Dequantization
- Double Quantization
  - First Level Quantization
  - Quantizing the quantization constants
  - Dequantize the constants
  - Dequantize the parameters using dequantized constants
- Unified Memory Paging/Paged Optimisers

Normalisation

1.Normalization

The weights of the model are first normalized to have zero mean and unit variance. This ensures that the weights are distributed around zero and fall within a certain range.

Quantization

Step 1: Original Weights

Assume we have a small block of weights:

$$W_{\text{original}} = \{0.35, -1.2, 2.7, -0.45\}$$

Step 2: Determine Quantization Range

Find the minimum and maximum values:

$$W_{\min} = -1.2, \quad W_{\max} = 2.7$$

Step 3: Calculate Scale Factor and Zero Point

1. **Calculate Scale Factor:**

   - The scale factor is calculated to map the range of weights to the range of 4-bit integers (0 to 15).
   - The range of the weights is:

$$\text{Range} = W_{\max} - W_{\min} = 2.7 - (-1.2) = 3.9$$

   - The scale factor is:

$$\text{Scale Factor} = \frac{\text{Range}}{15} = \frac{3.9}{15} \approx 0.26$$

Quantization

2. **Calculate Zero Point:**

   - The zero point is the value that maps to the minimum weight:

$$\text{Zero Point} = \text{round}\left(\frac{-W_{\text{min}}}{\text{Scale Factor}}\right) = \text{round}\left(\frac{1.2}{0.26}\right) \approx 5$$

4. **Quantize the Parameters:**

- Using the scale factor and zero point, we convert the original values to 4-bit integers:

$$W_{\text{quantized}} = \text{round}\left(\frac{W_{\text{original}}}{\text{Scale Factor}} + \text{Zero Point}\right)$$

Applying this to each weight:

$$0.35 \rightarrow \text{round}\left(\frac{0.35}{0.26} + 5\right) = \text{round}(1.346 + 5) = 6$$

$$-1.2 \rightarrow \text{round}\left(\frac{-1.2}{0.26} + 5\right) = \text{round}(-4.615 + 5) = 0$$

$$2.7 \rightarrow \text{round}\left(\frac{2.7}{0.26} + 5\right) = \text{round}(10.385 + 5) = 15$$

$$-0.45 \rightarrow \text{round}\left(\frac{-0.45}{0.26} + 5\right) = \text{round}(-1.731 + 5) = 3$$

- So, the quantized values are:

$$W_{\text{quantized}} = \{6, 0, 15, 3\}$$

5. **Dequantize the Parameters:**

- To convert the quantized values back to their approximate original values, use the scale factor and zero point:

$$W_{\text{dequantized}} = (W_{\text{quantized}} - \text{Zero Point}) \times \text{Scale Factor}$$

Applying this to each quantized weight:

$$6 \rightarrow (6 - 5) \times 0.26 = 1 \times 0.26 = 0.26$$
$$0 \rightarrow (0 - 5) \times 0.26 = -5 \times 0.26 = -1.3$$
$$15 \rightarrow (15 - 5) \times 0.26 = 10 \times 0.26 = 2.6$$
$$3 \rightarrow (3 - 5) \times 0.26 = -2 \times 0.26 = -0.52$$

$$W_{\text{original}} = \{0.35, -1.2, 2.7, -0.45\}$$

- The dequantized values approximate the original values, but with some loss of precision:

$$W_{\text{dequantized}} = \{0.26, -1.3, 2.6, -0.52\}$$

Double Quantization

Step 1: First Level Quantization (Recap)

We already quantized our original parameters:

1. **Original Parameters (High Precision):**

$$W_{\text{original}} = \{0.35, -1.2, 2.7, -0.45\}$$

2. **Determine Quantization Range:**

$$W_{\min} = -1.2, \quad W_{\max} = 2.7$$

3. **Calculate Scale Factor and Zero Point:**

$$\text{Scale Factor} = 0.26, \quad \text{Zero Point} = 5$$

4. **Quantize the Parameters:**

$$W_{\text{quantized}} = \{6, 0, 15, 3\}$$

Double Quantization

Step 2: Quantizing the Quantization Constants

Now, we'll apply a second level of quantization to the scale factor and zero point.

1. **Quantization Constants:**
    - Scale Factor: 0.26
    - Zero Point: 5
2. **Determine Range for Constants:**
    - Assume we need to store the scale factor and zero point with even lower precision. Let's represent these using 4-bit integers again.

Double Quantization

3. **Quantize the Scale Factor:**

- Suppose the possible range for the scale factor (0.26 in this case) is from 0 to 1.
- We divide this range into 16 steps (for 4-bit):

$$\text{Scale Factor Steps} = \frac{1-0}{15} = 0.0667$$

- Calculate quantized value for the scale factor:

$$\text{Quantized Scale Factor} = \text{round}\left(\frac{0.26}{0.0667}\right) = \text{round}(3.9) = 4$$

Double Quantization

4. **Quantize the Zero Point:**

- Assume the range for the zero point (5) is from 0 to 10.
- We divide this range into 16 steps (for 4-bit):

$$\text{Zero Point Steps} = \frac{10 - 0}{15} = 0.6667$$

- Calculate quantized value for the zero point:

$$\text{Quantized Zero Point} = \text{round}\left(\frac{5}{0.6667}\right) = \text{round}(7.5) = 8$$

Double Quantization

Step 3: Dequantize the Constants

To use these quantized constants, we need to dequantize them back:

1. **Dequantize the Scale Factor:**

$$\text{Dequantized Scale Factor} = 4 \times 0.0667 = 0.2668$$

2. **Dequantize the Zero Point:**

$$\text{Dequantized Zero Point} = 8 \times 0.6667 = 5.3336$$

Double Quantization

QLoRA Steps

Step 4: Dequantize the Parameters Using Dequantized Constants

Now, use the dequantized constants to dequantize the parameters:

1. **Dequantize the Parameters:**

$$W_{\text{dequantized}} = (W_{\text{quantized}} - \text{Dequantized Zero Point}) \times \text{Dequantized Scale Factor}$$

Applying this to each quantized weight:

$$\begin{aligned}
6 &\rightarrow (6 - 5.3336) \times 0.2668 = 0.6664 \times 0.2668 = 0.1778 \\
0 &\rightarrow (0 - 5.3336) \times 0.2668 = -5.3336 \times 0.2668 = -1.423 \\
15 &\rightarrow (15 - 5.3336) \times 0.2668 = 9.6664 \times 0.2668 = 2.579 \\
3 &\rightarrow (3 - 5.3336) \times 0.2668 = -2.3336 \times 0.2668 = -0.6225
\end{aligned}$$

The dequantized values are:

$$W_{\text{dequantized}} = \{0.1778, -1.423, 2.579, -0.6225\}$$

Paged Optimisers/Unified Memory paging

Paged optimizers are designed to efficiently manage memory usage, especially when training large models with long sequence lengths. They leverage memory paging techniques and NVIDIA's unified memory to avoid out-of-memory (OOM) errors and optimize performance

Paged Optimisers

Memory Challenges in Training LLMs

Training large language models involves handling massive amounts of data and numerous parameters, which can lead to significant memory usage. The main memory challenges include:

- **Memory Spikes**: These occur during gradient calculations, especially with long sequences, causing temporary peaks in memory usage.
- **Gradient Checkpointing**: This technique saves intermediate activations at specific points to reduce memory usage but can still cause spikes.

How Paged Optimizers Work

1. **Unified Memory**: Paged optimizers utilize NVIDIA's unified memory, which **allows a GPU to use both its own VRAM and the system RAM**. This approach provides a larger memory pool, reducing the risk of OOM errors when the GPU's VRAM is insufficient.

2. **Paging System**: Similar to how operating systems manage memory, paged optimizers use a **paging system to dynamically allocate and deallocate memory**. When processing a mini-batch, only the required portions of the model and data are loaded into the GPU memory, while the rest remain in the system RAM.

How Paged Optimizers Help

1. Avoiding Memory Spikes: By **spreading the memory usage more evenly between the GPU memory and system RAM**, paged optimizers help in mitigating memory spikes. This approach ensures that large models and long sequences can be processed without causing sudden peaks in memory usage that could lead to OOM errors.

2. Efficiency in Training: The optimizer **dynamically pages data in and out of GPU memory, ensuring that the GPU is not overwhelmed** by large memory demands at any given time. This dynamic management makes it possible to handle larger models or longer sequences than what would typically fit into the GPU's VRAM alone.

Benefits of Paged Optimizers

- **Increased Model Size**: Allows for training larger models than what the GPU VRAM alone would permit.
- **Handling Long Sequences**: Enables processing of longer sequences without running into OOM errors.
- **Efficient Memory Use**: Optimizes memory allocation dynamically, improving training efficiency.

Retrieval-Augmented Generation (RAG) with Large Language Models

Courtesy:, deeplearning.ai , https://arxiv.org/abs/2005.11401

RAG, short for Retrieval-Augmented Generation, helps large language models by giving them access to relevant information during text generation. This allows them to be more accurate and informative, especially for tasks that require real-world knowledge.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis[†‡], **Ethan Perez**[⋆],

Aleksandra Piktus[†], **Fabio Petroni**[†], **Vladimir Karpukhin**[†], **Naman Goyal**[†], **Heinrich Küttler**[†],

Mike Lewis[†], **Wen-tau Yih**[†], **Tim Rocktäschel**[†‡], **Sebastian Riedel**[†‡], **Douwe Kiela**[†]

[†]Facebook AI Research; [‡]University College London; [⋆]New York University;
plewis@fb.com

Abstract

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages per token. We fine-tune and evaluate our models on a wide range of knowledge-intensive NLP tasks and set the state of the art on three open domain QA tasks, outperforming parametric seq2seq models and task-specific retrieve-and-extract architectures. For language generation tasks, we find that RAG models generate more specific, diverse and factual language than a state-of-the-art parametric-only seq2seq baseline.

What is RAG?

Large language models (LLMs) have undoubtedly changed the way we interact with information. However, they come with their fair share of limitations as to what we can ask of them. Base LLMs (ex. Llama-2-70b, gpt-4, etc.) **are only aware of the information that they've been trained on** and will fall short when we require them to know information beyond that. **Retrieval augmented generation (RAG)** based LLM applications address this exact issue and **extend the utility of LLMs to our specific data sources.**

RAG- steps

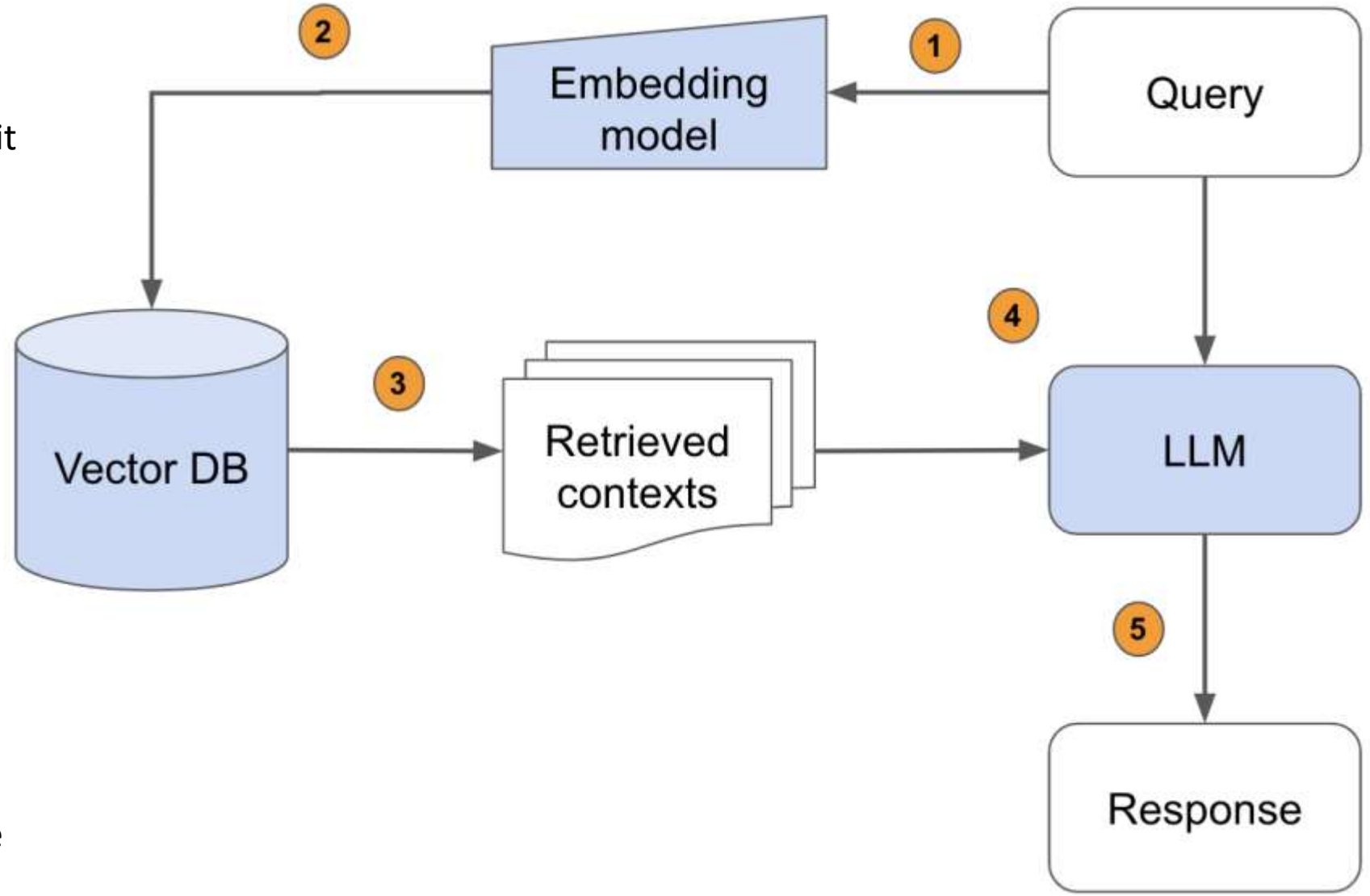
1. Pass the query to the embedding model to semantically represent it as an embedded query vector.

2. Pass the embedded query vector to our vector DB.

3. Retrieve the top-k relevant contexts – measured by distance between the query embedding and all the embedded chunks in our knowledge base.

4. Pass the query text and retrieved context text to our LLM.

5. The LLM will generate a response using the provided content.



The integration of RAG into LLMs

The integration of RAG into LLMs involves two main components: the retriever and the generator.

Retriever: The retriever takes the input query, converts it into a vector using the query encoder, and then finds the most similar document vectors in the corpus. The documents associated with these vectors are then passed to the generator.

Generator : The generator in a RAG-LLM setup is a large transformer model, such as GPT3.5, GPT4, Llama2, Falcon, PaLM, and BERT. The generator takes the input query and the retrieved documents, and generates a response.

Training RAG-LLM Models

Training a RAG-LLM model involves fine-tuning both the retriever and the generator on a question-answering dataset. The retriever is trained to retrieve documents that are relevant to the input query, while the generator is trained to generate accurate responses based on the input query and the retrieved documents.

- During training, pre-trained weights $W_0 \in \mathbb{R}^{d \times k}$ is fixed

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

$$B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}, r \ll \min(d, k)$$

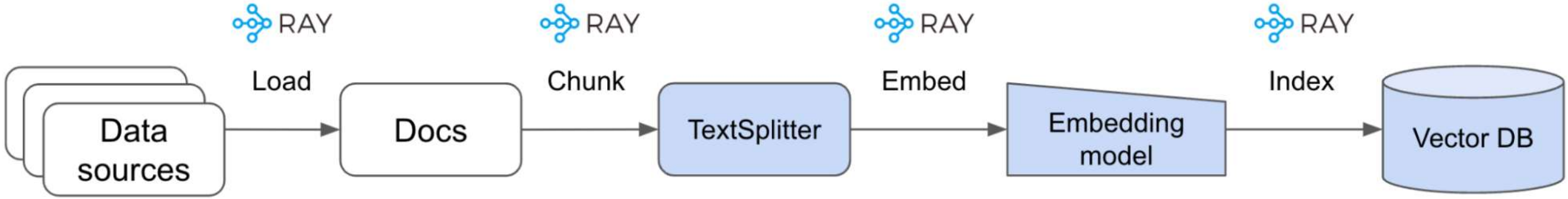
- LoRA can be applied to any weights in a deep neural network
- In the LoRA paper, the authors considered three weights related to self-attention

$$W_k, W_q, W_v \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$$

Vector DB creation

Before we can start building our RAG application, we need to first create our vector DB that will contain our processed data sources.

Load data: We're going to then load our docs contents into a Dataset so that we can perform operations at scale on them (ex. embed, index, etc.).



What is the optimal rank?

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "Lora: Low-rank adaptation of large language models." arXiv preprint arXiv:2106.09685 (2021).

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm$0.5%)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm$0.1%)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "Lora: Low-rank adaptation of large language models." arXiv preprint arXiv:2106.09685 (2021).

- Subspace spanned by the top i singular vectors of U_{r8} is contained in the subspace spanned by the top j singular vectors of U_{r64}
- ΔW has a strong correlation with W
- ΔW amplifies some features that are already in W but not emphasized in W
- ΔW with a larger rank tends to pick up more directions already emphasized in W

- Intrinsic SAID
  - Structure-Aware Intrinsic Dimension (SAID)
  - An objective function's intrinsic dimensionality describes the **minimum dimension needed to solve the optimization problem** it defines to some precision level
  - Intrinsic dimensionality of a pre-trained LLM (or LM): **The number of free parameters required to closely approximate the optimization problem to be solved during fine-tuning of a model for a downstream task**.
    - Intrinsic dimension is the lowest dimensional subspace in which one can optimize the original function to within a certain level of approximation error

$\theta^D = [\theta_0, \theta_1, \ldots, \theta_m]$ Set of D parameters that parameterize some model $f(., \theta)$

Generally, re-parameterize in the lower-dimensional d-dimensions

$$\theta^D = \theta_0^D + P(\theta^d)$$

$P : \mathbb{R}^d \rightarrow \mathbb{R}^D$ Linear projection

θ_0^D is the original model parameterization

SAID

$$\theta_i^D = \theta_{0,i}^D + \lambda_i P\left(\theta^{d-m}\right)_i$$

m is the number of layers

i represents a layer

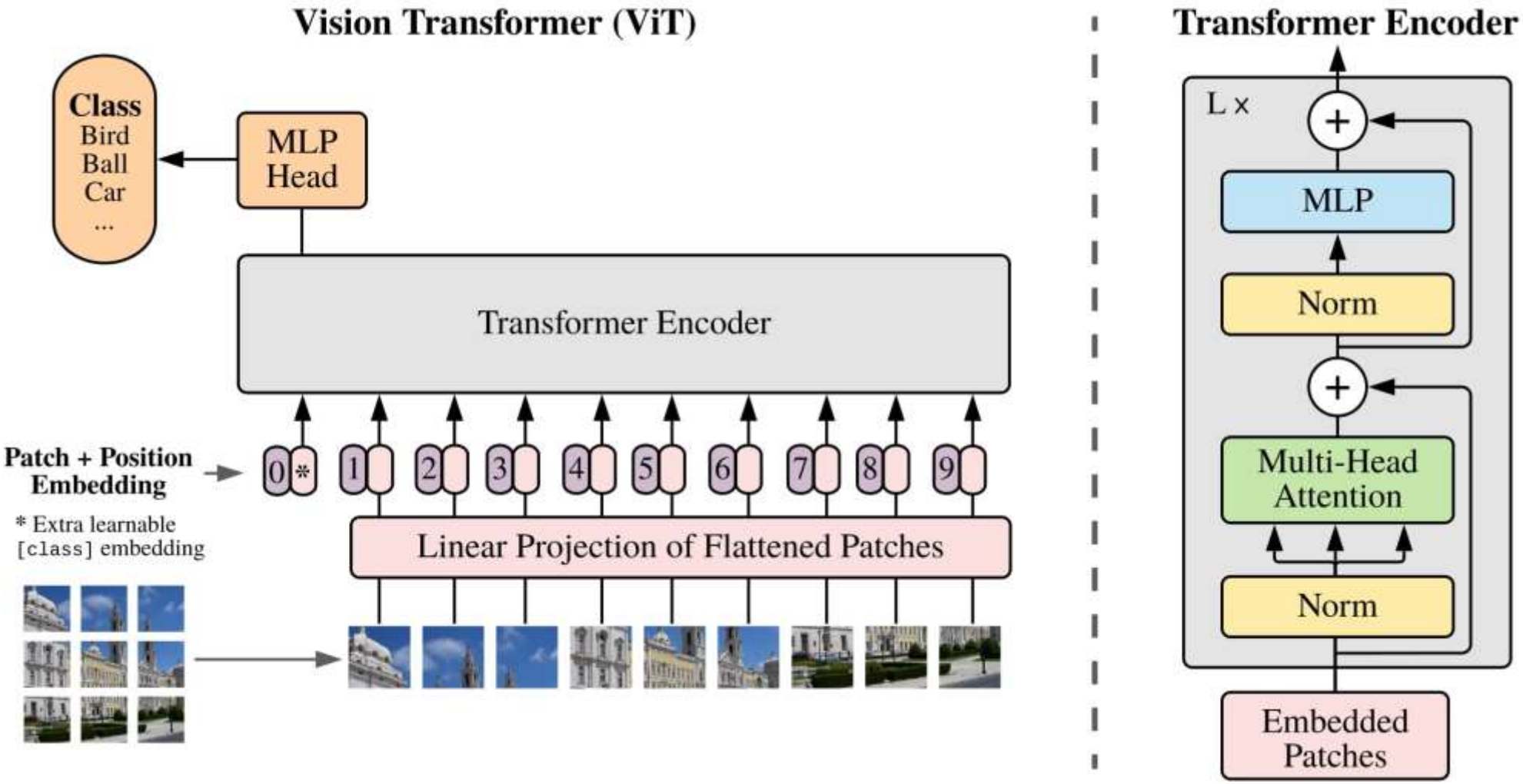
λ is a scaling parameter

- Sample code

- https://medium.com/@siddharth.vij10/prompt-engineering-llama-2-chat-cot-react-few-shot-self-critiq-fbf3bbf6688f
- https://arxiv.org/pdf/2401.06866v1
- https://arxiv.org/pdf/2402.07927
- https://medium.com/decodingml/why-you-need-to-pay-attention-to-llm-prompt-templates-51808189a2e4
- https://github.com/ksm26/Prompt-Engineering-with-Llama-2

- https://medium.com/@dillipprasad60/qlora-explained-a-deep-dive-into-parametric-efficient-fine-tuning-in-large-language-models-llms-c1a4794b1766
- https://medium.com/@marketing_novita.ai/step-by-step-tutorial-on-integrating-retrieval-augmented-generation-rag-with-large-language-7c509cddf4ac
- https://www.youtube.com/watch?v=pov3pLFMOPY
- https://medium.com/@hayagriva99999/lora-and-qlora-an-efficient-approach-to-fine-tuning-large-models-under-the-hood-948468424cd6#:~:text=QLoRA%20extends%20LoRA%20to%20enhance,memory%20demands%20and%20faster%20calculations.

- https://realpython.com/build-llm-rag-chatbot-with-langchain/
- https://www.anyscale.com/blog/a-comprehensive-guide-for-building-rag-based-llm-applications-part-1


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP
Head
Transformer Encoder
Patch + Position
Embedding
* Extra learnable
[class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head
Attention
Norm
Embedded
Patches

The total architecture is called Vision Transformer (ViT in short). Let's examine it step by step.

1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the flattened patches
4. Add positional embeddings
5. Feed the sequence as an input to a standard transformer encoder
6. Pretrain the model with image labels (fully supervised on a huge dataset)
7. Finetune on the downstream dataset for image classification

Thank You

Namah Shivaya

- https://medium.com/@shravankoninti/decoding-strategies-of-all-decoder-only-models-gpt-631faa4c449a#:~:text=In%20the%20previous%20blog%20we,steps%20and%20it%20generates%20outputs.
- https://medium.com/@shravankoninti/generative-pretrained-transformer-gpt-4b94d017a3f9
- https://medium.com/@shravankoninti/generative-pretrained-transformer-gpt-pre-training-fine-tuning-different-use-case-c93bb0553ffc
- https://medium.com/@shravankoninti/generative-pretrained-transformer-gpt-4b94d017a3f9
- https://medium.com/@YanAIx/step-by-step-into-gpt-70bc4a5d8714
- https://jalammar.github.io/illustrated-gpt2/
- https://learn.deeplearning.ai/courses/finetuning-large-language-models/lesson/1/introduction