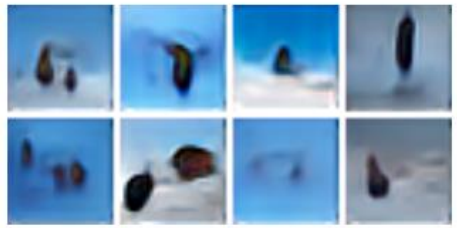
# Multimodal Generative AI 2025
# Fine-tuning Foundation Models


VISTA
École Polytechnique

# Evolution of Multimodal Generative AI

# Evolution of Multimodal Generative AI

A stop sign is flying in blue skies.



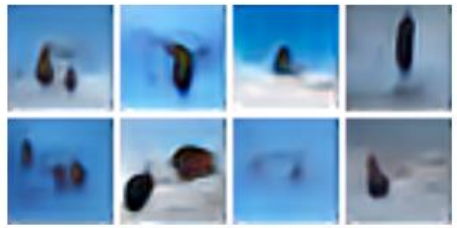
A herd of elephants flying in the blue skies.

A toilet seat sits open in the grass field.

A person skiing on sand clad vast desert.

# Evolution of Multimodal Generative AI

A stop sign is flying in blue skies.

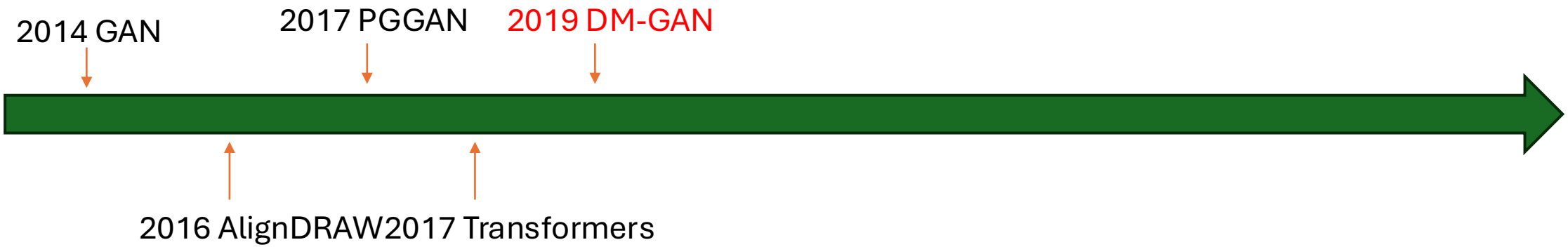


A herd of elephants flying in the blue skies.

A toilet seat sits open in the grass field.

A person skiing on sand clad vast desert.

# Evolution of Multimodal Generative AI

# Evolution of Multimodal Generative AI

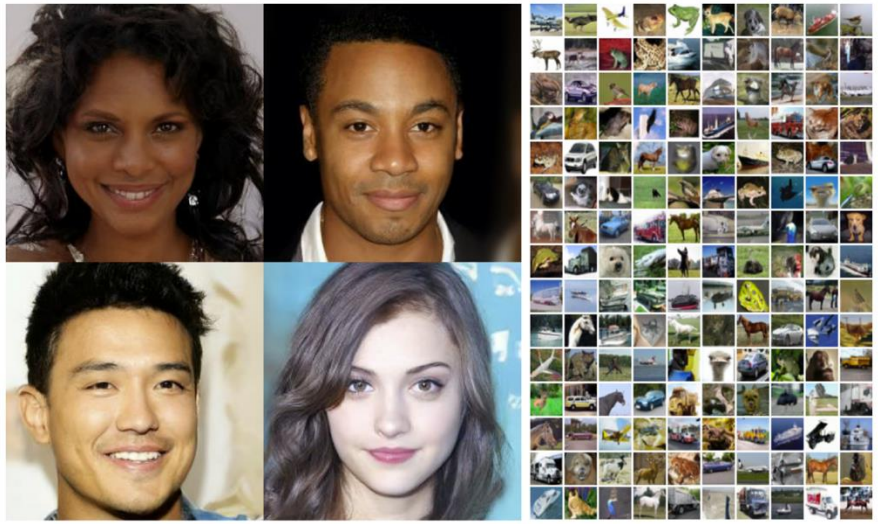
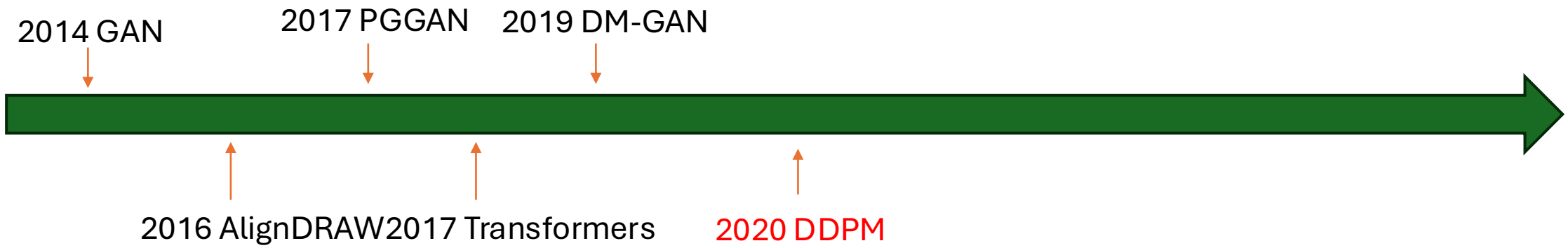
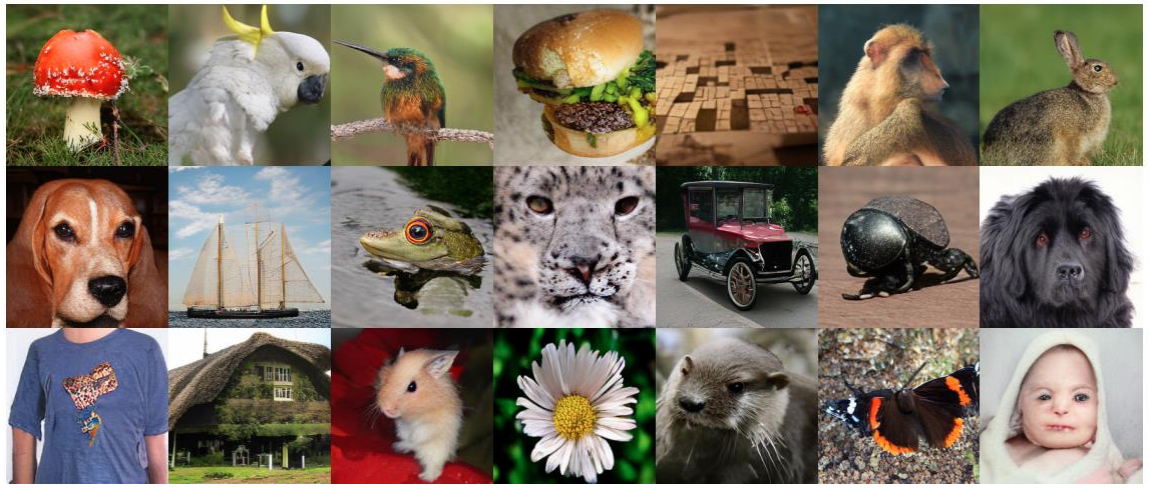


Figure 1: Generated samples on CelebA-HQ $256 \times 256$ (left) and unconditional CIFAR10 (right)



# Evolution of Multimodal Generative AI



2014 GAN

2017 PGGAN

2019 DM-GAN(T2I)

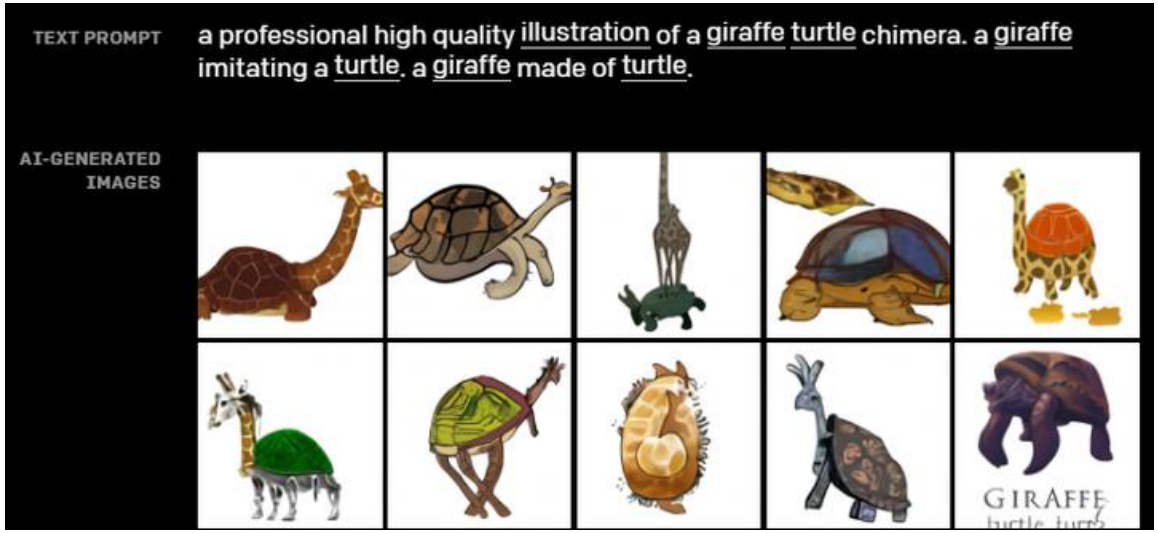
2021 VQ-GAN

2016 AlignDRAW

2017 Transformers

2020 DDPM

# Evolution of Multimodal Generative AI




DALLE-1

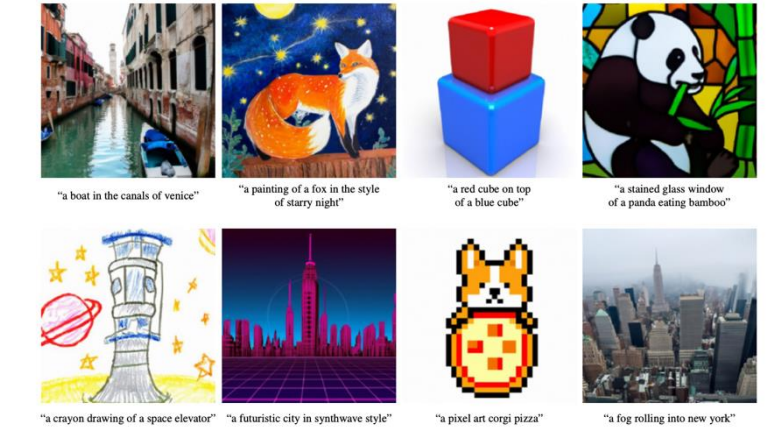



*Figure 1.* Selected samples from GLIDE using classifier-free guidance. We observe that our model can produce photorealistic images with shadows and reflections, can compose multiple concepts in the correct way, and can produce artistic renderings of novel concepts. For random sample grids, see Figure 17 and 18.

GLIDE

2014 GAN

2017 PGGAN

2019 DM-GAN(T2I)

2021 VQ-GAN

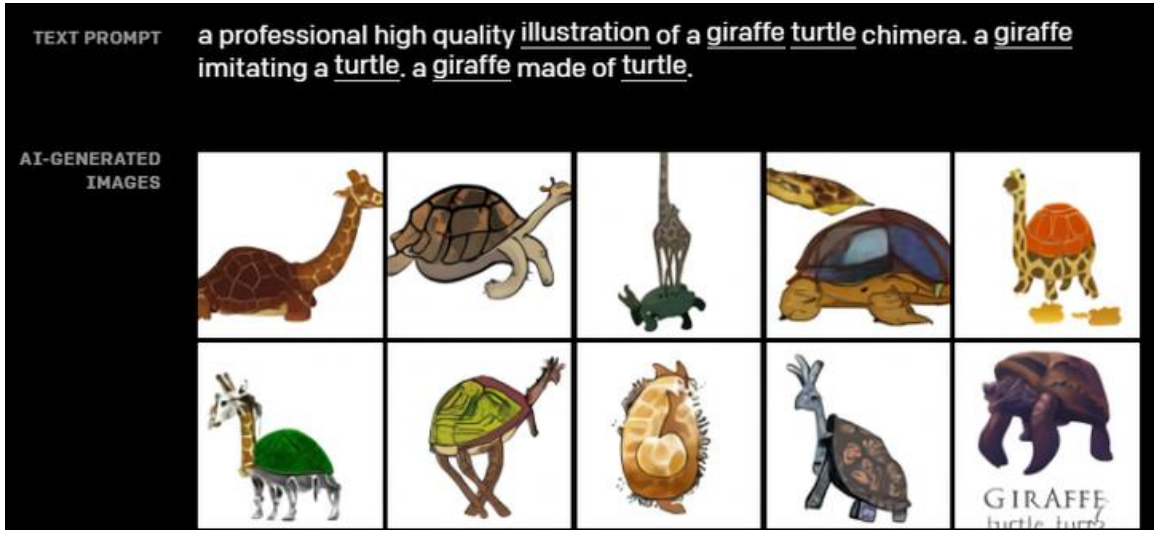
2022 DALLE, GLIDE

2016 AlignDRAW

2017 Transformers

2020 DDPM

# Evolution of Multimodal Generative AI




DALLE-1

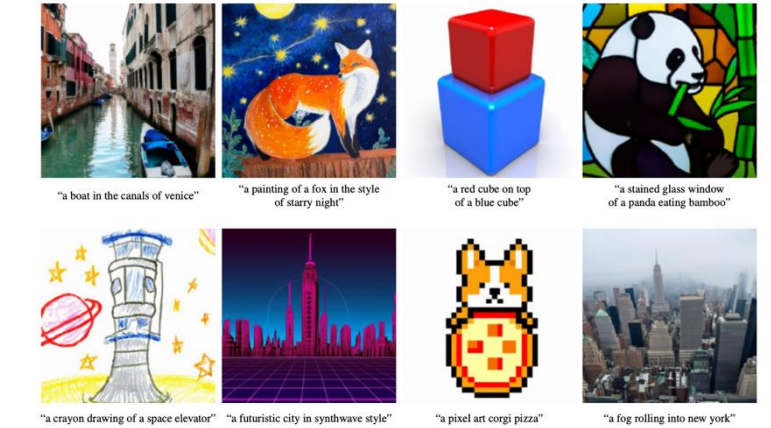



*Figure 1.* Selected samples from GLIDE using classifier-free guidance. We observe that our model can produce photorealistic images with shadows and reflections, can compose multiple concepts in the correct way, and can produce artistic renderings of novel concepts. For random sample grids, see Figure 17 and 18.

GLIDE

2014 GAN

2017 PGGAN

2019 DM-GAN(T2I)

2021 VQ-GAN

2022 DALLE, GLIDE

2016 AlignDRAW

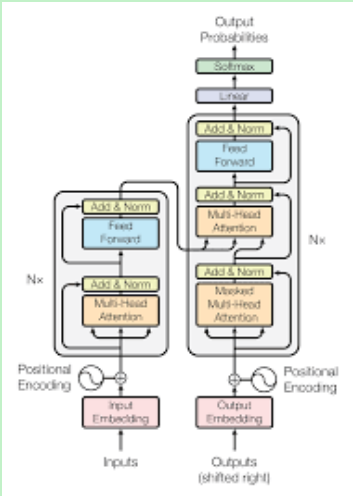
2017 Transformers

2020 DDPM

2022-2023 Dalle-3, Midjourney, StableDiffusion

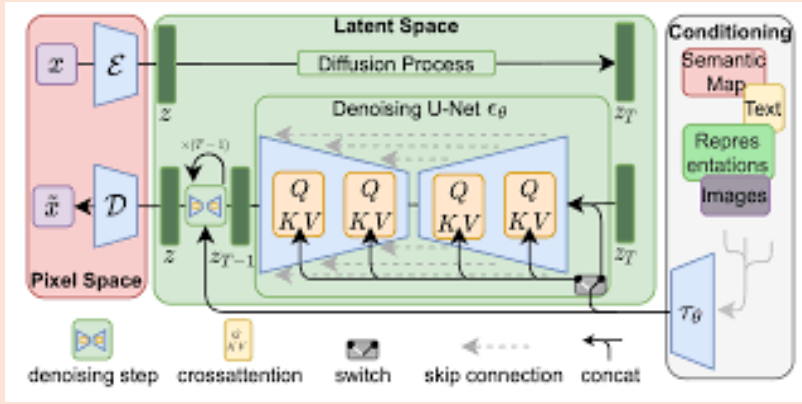
# Popular Generative AI architectures

## Transformers



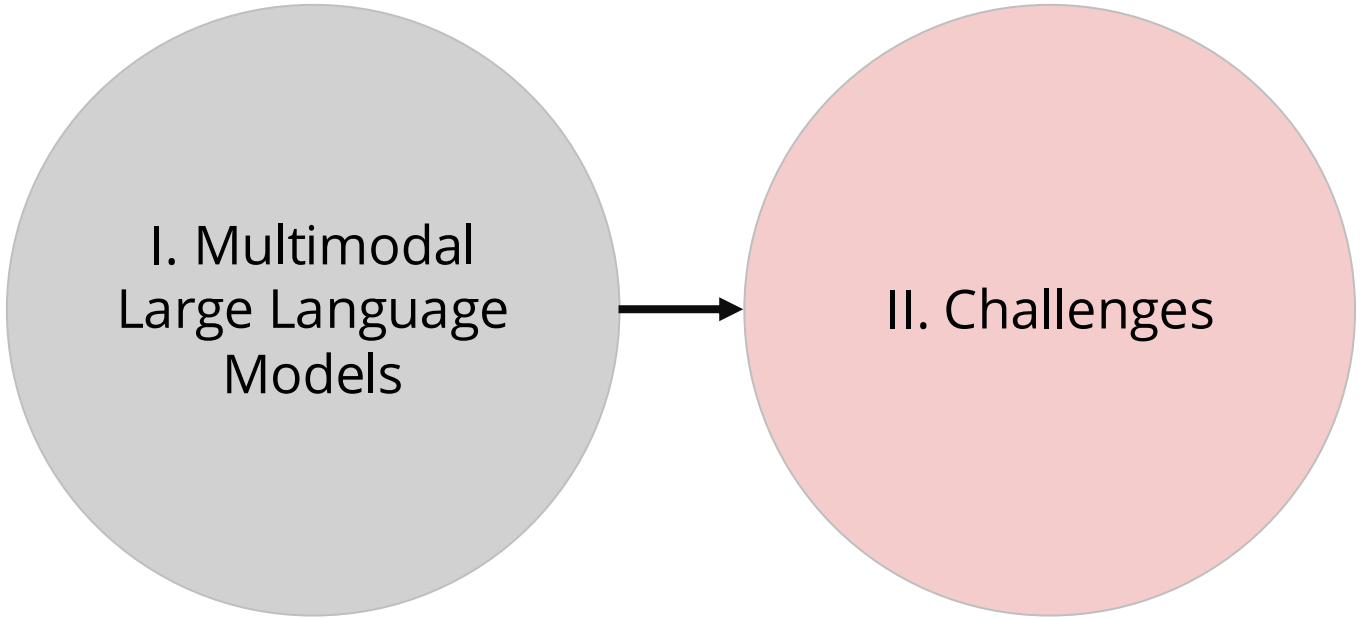

Use: Text, code, tokens, sequential data
Examples: GPT series, LLaMA, Claude

## Diffusion Models

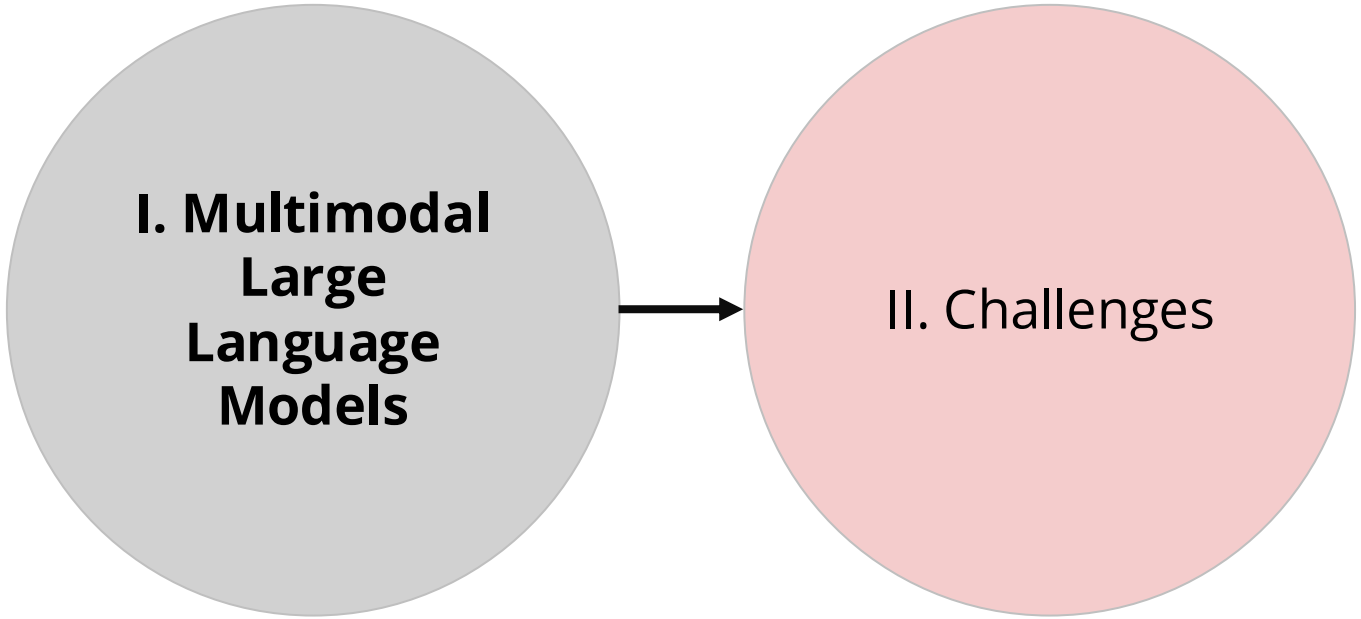



Use: Image, audio, continuous data
Examples: Stable Diffusion, DALL-E

# Generative AI



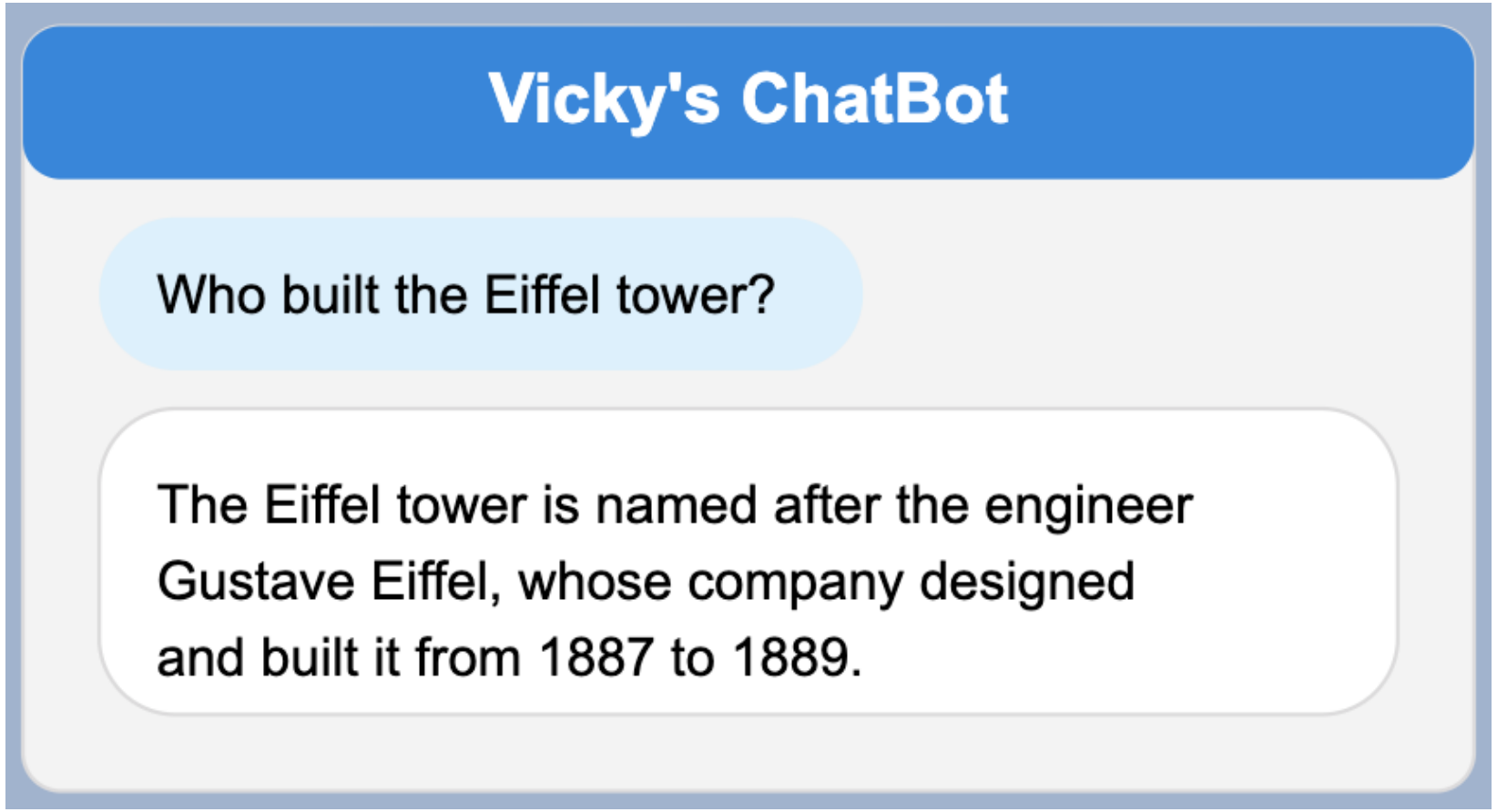
# Generative AI



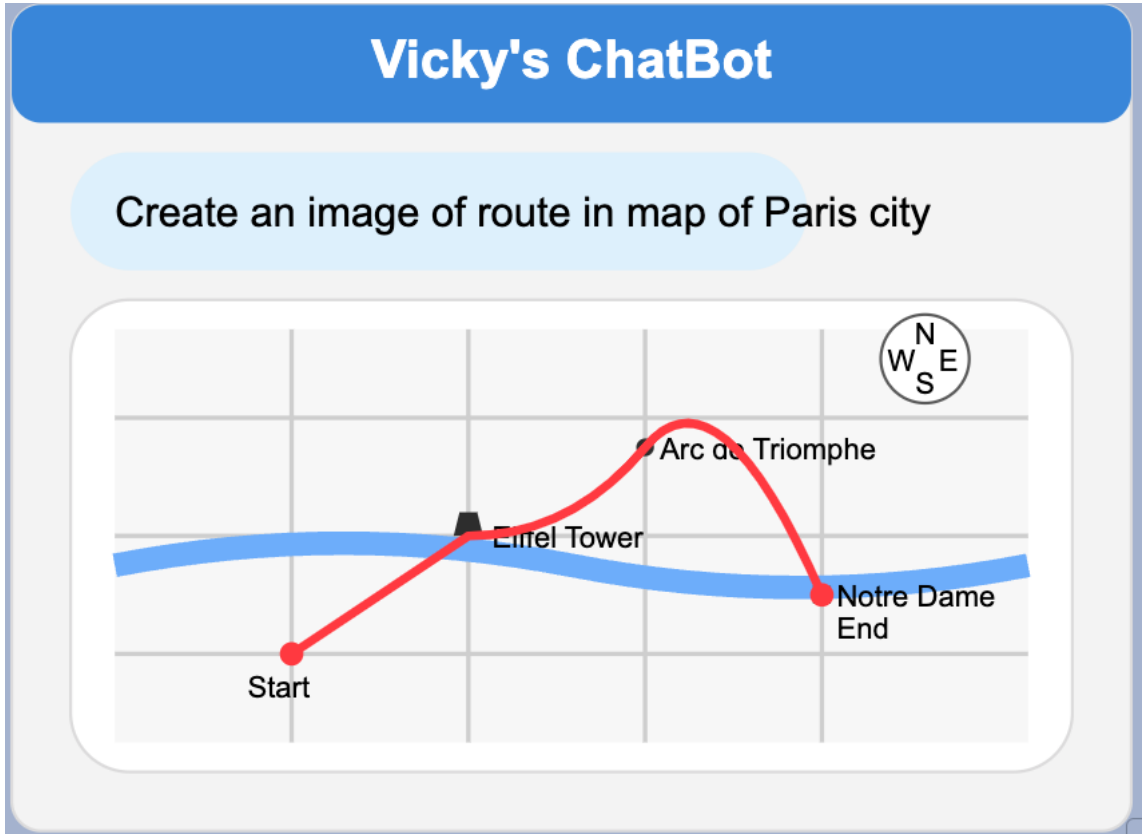
# I. Multimodal Large Language Models

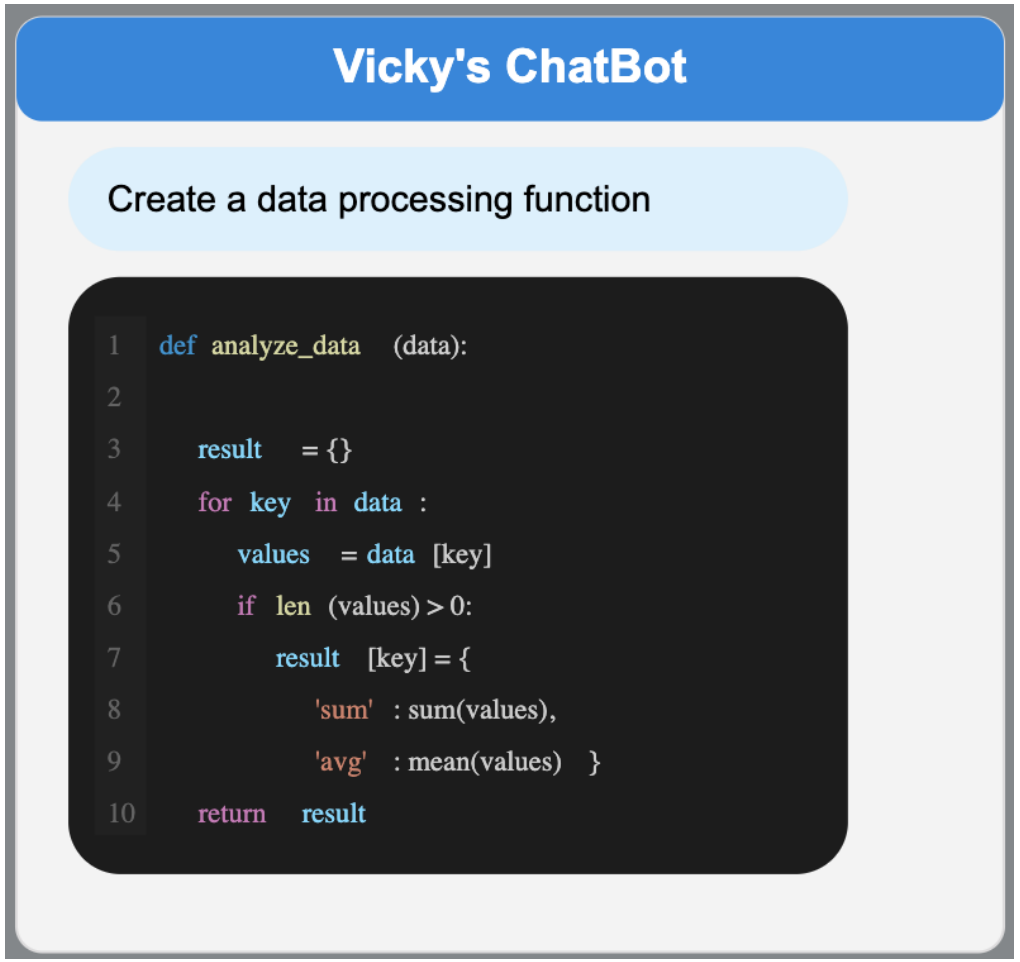
- **Introduction to LLMs**
- Transformers & Self-attention
- Towards unification of fields
- Vision-Language Models

# Large Language Models (LLMs)

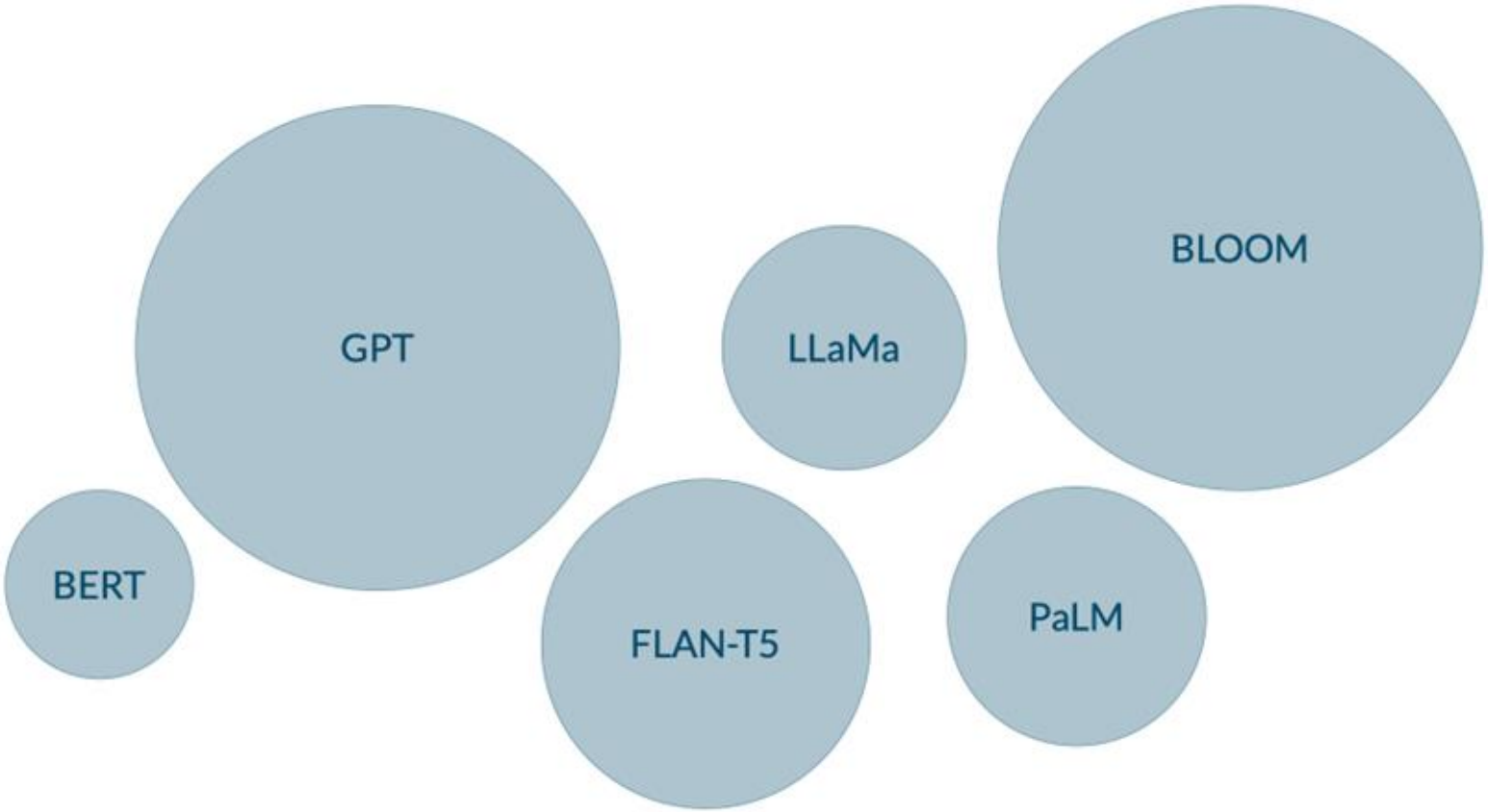


# Large Language Models (LLMs)

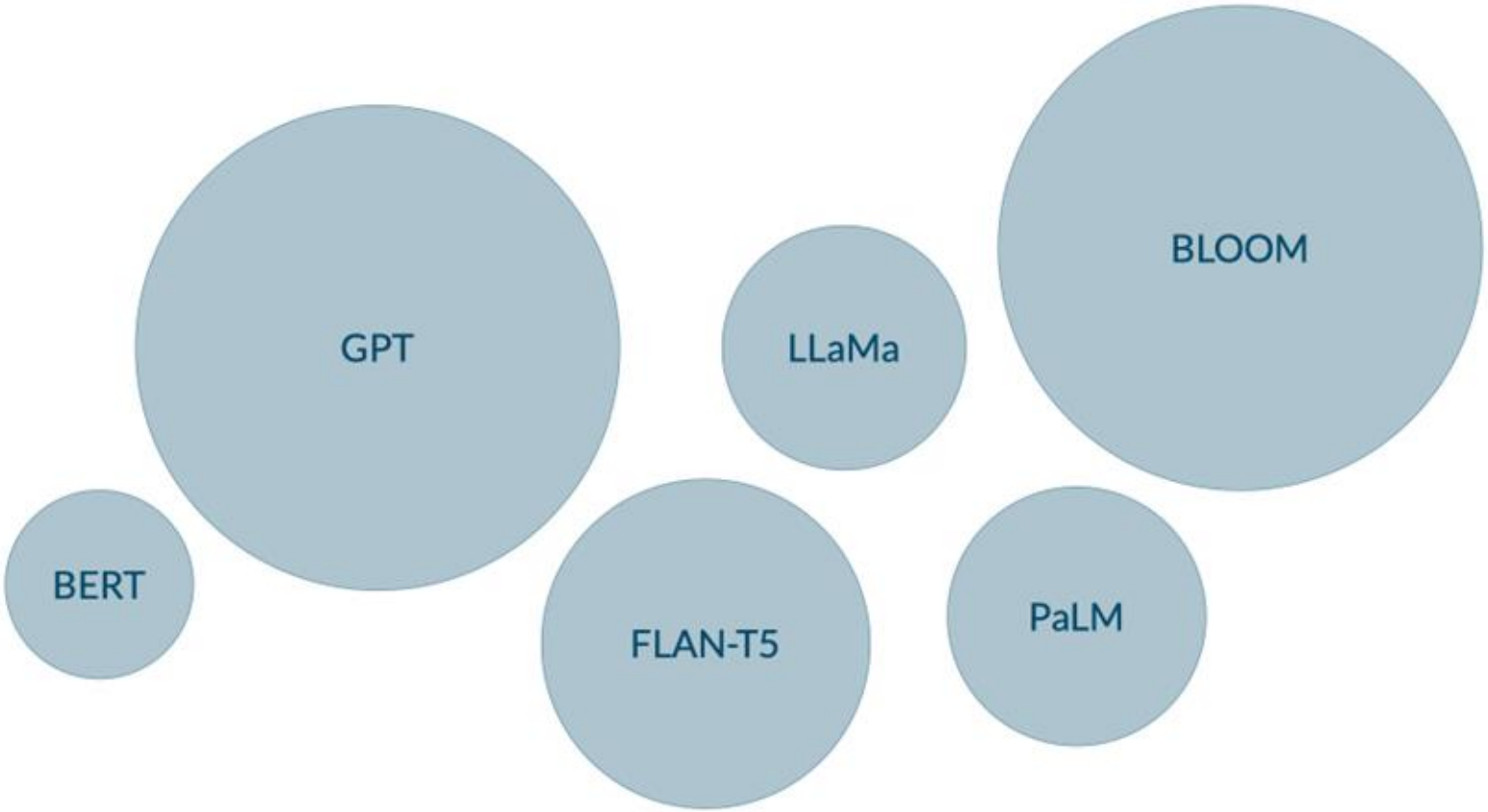


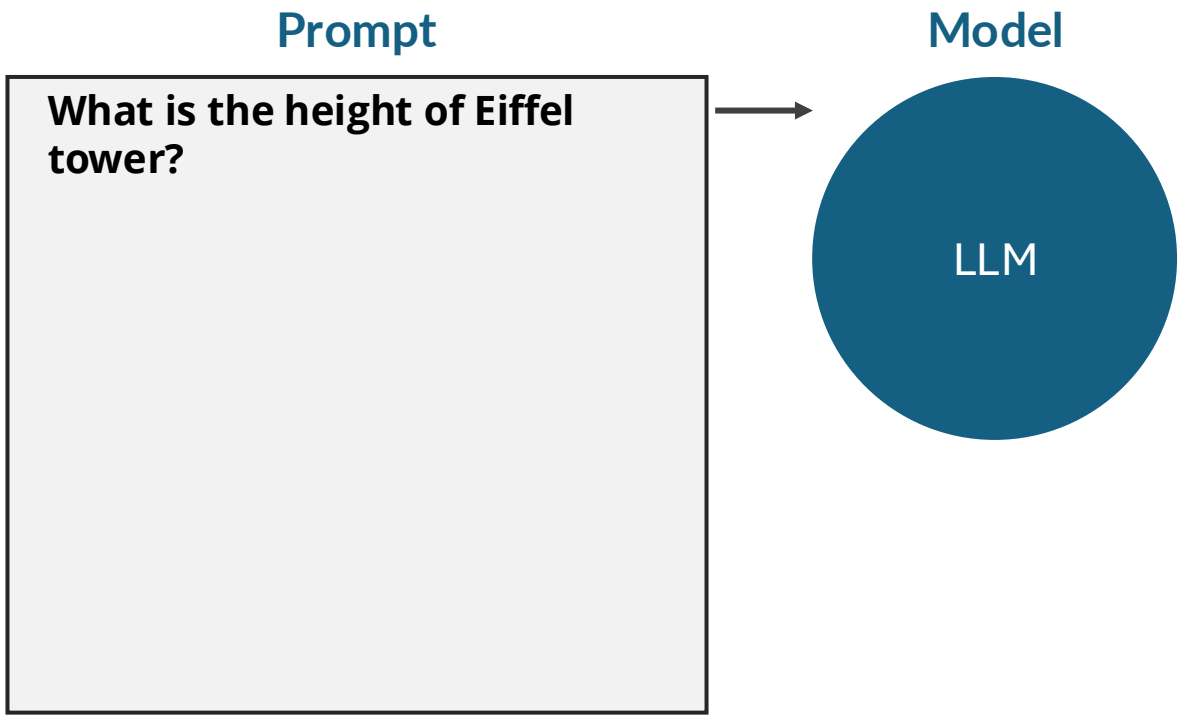
# Large Language Models (LLMs)



# Large Language Models (LLMs)

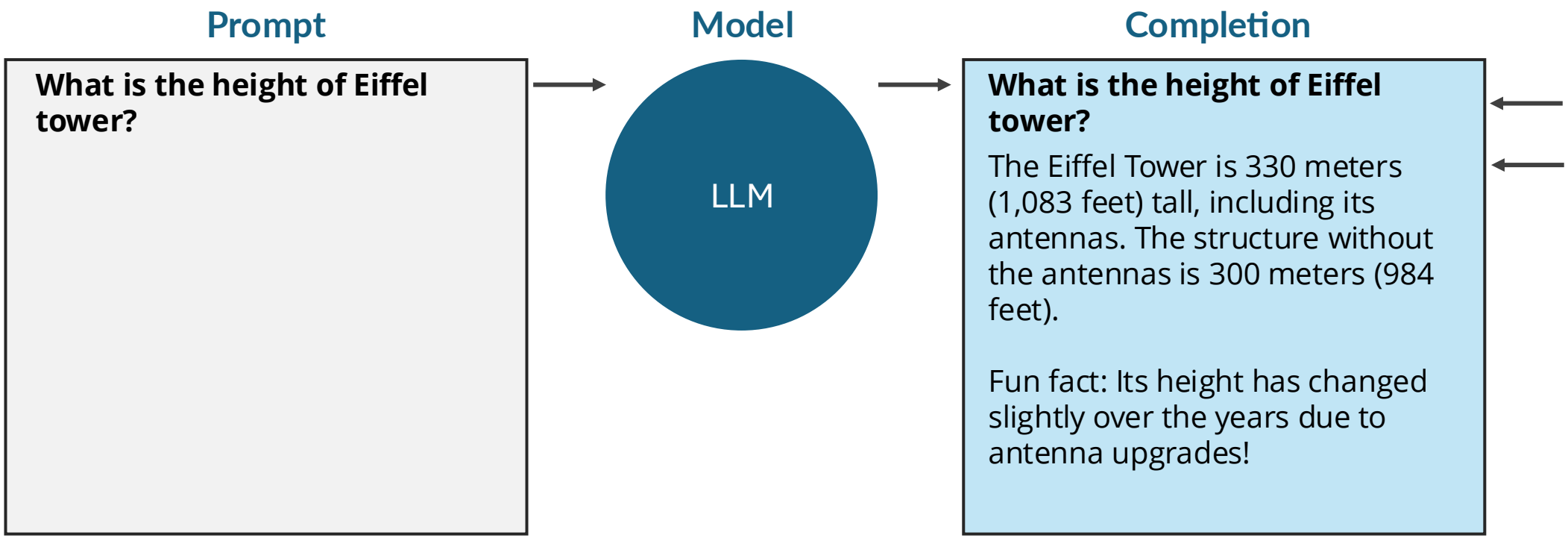


# Prompts and completions



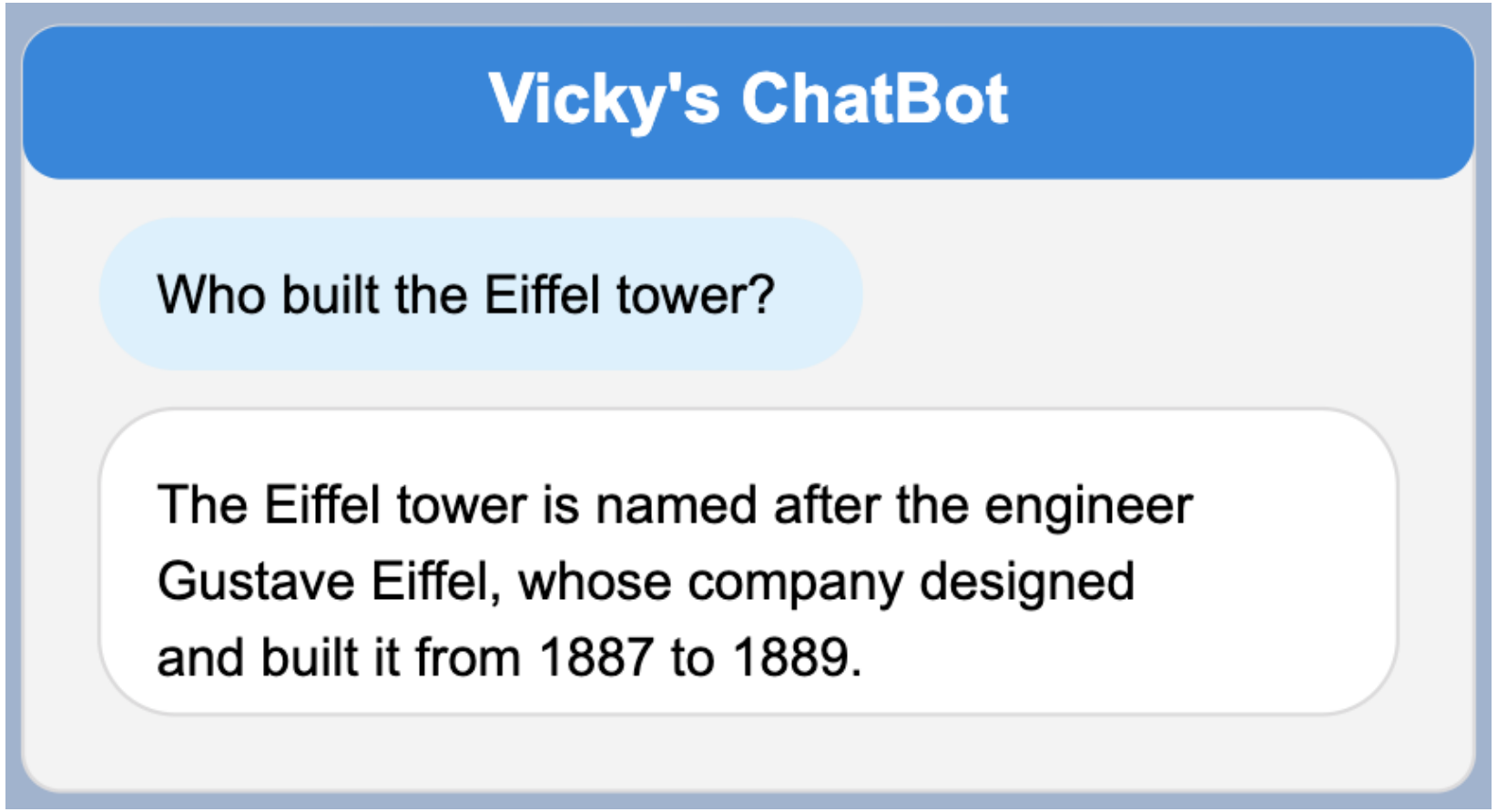

Context window: typically a few 1000 words

# Prompts and completions



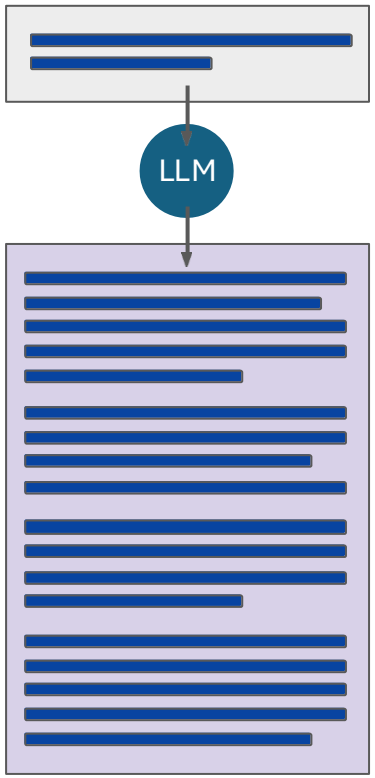

Context window: typically a few 1000 words

# Use cases & Tasks

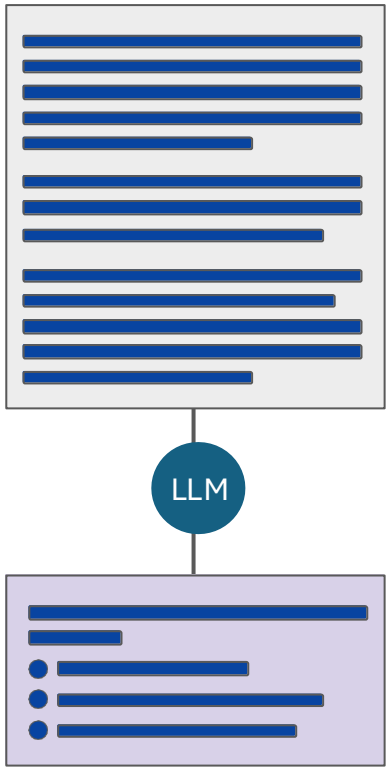


# LLM use cases & tasks

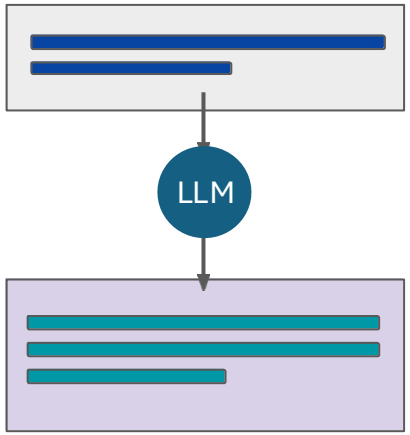
Essay Writing




Summarization

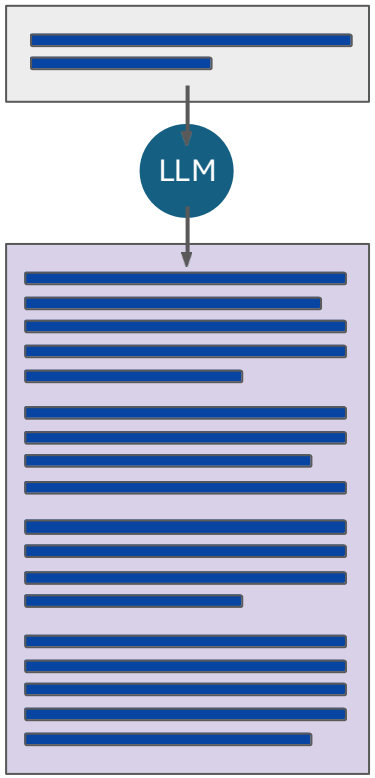



Translation

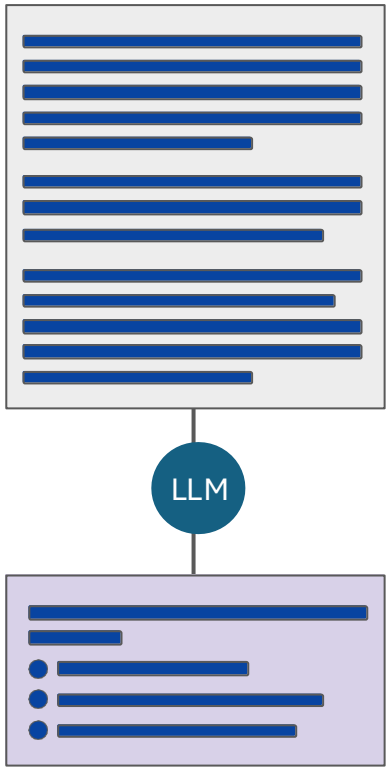


# LLM use cases & tasks

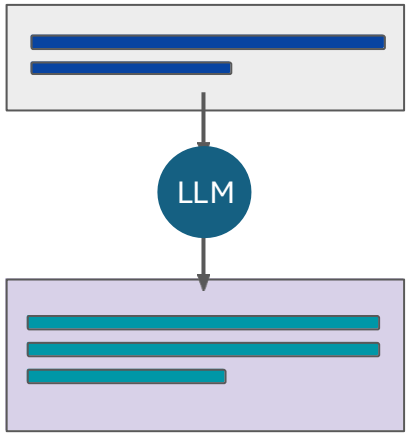
Essay Writing



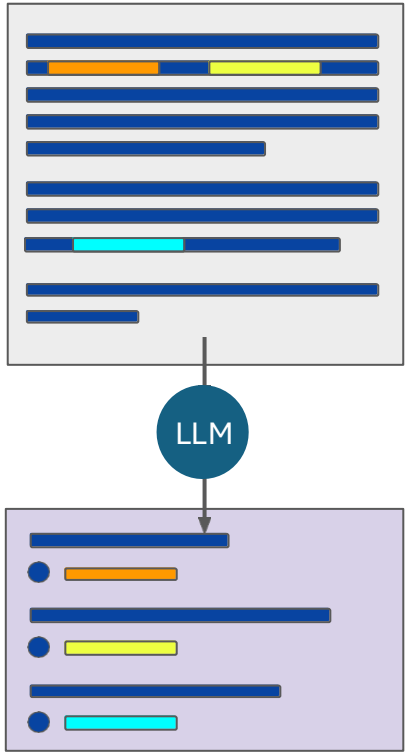

Summarization



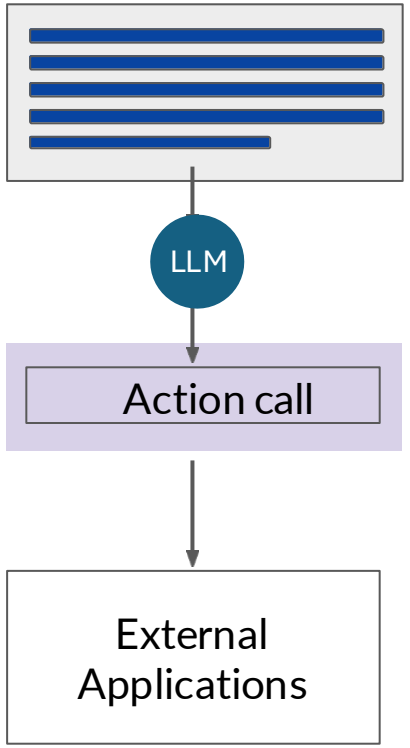

Translation




Information retrieval




Invoke APIs and actions



# I. Multimodal Large Language Models

- Introduction to LLMs
- **Transformers & Self-attention**
- Towards unification of fields
- Vision-Language Models

# Generative AI exists because of the **Transformer**

# Transformers

## Attention Is All You Need


**Ashish Vaswani*** Google Brain avaswani@google.com

**Noam Shazeer*** Google Brain noam@google.com

**Niki Parmar*** Google Research nikip@google.com

**Jakob Uszkoreit*** Google Research usz@google.com

**Llion Jones*** Google Research llion@google.com

**Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu

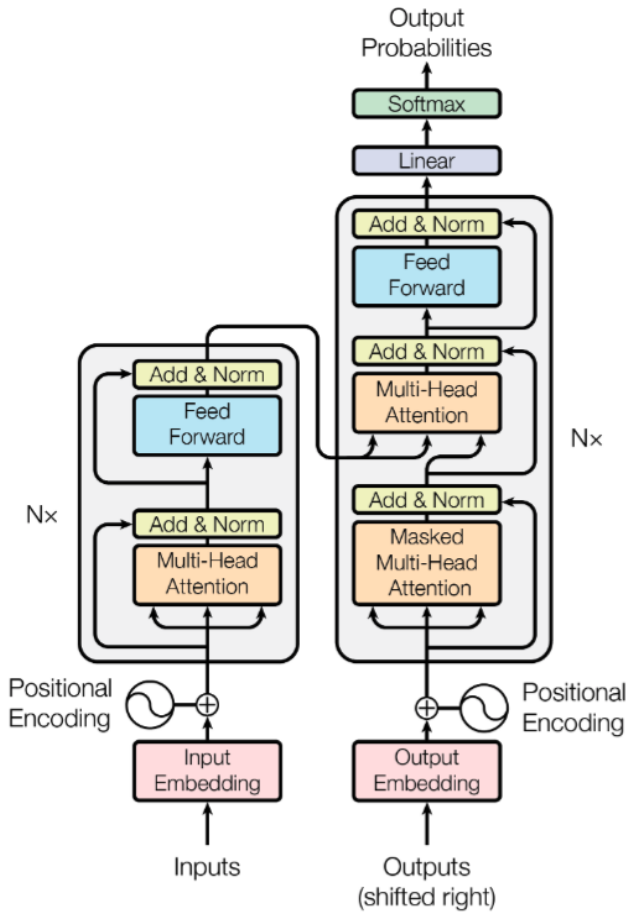
**Łukasz Kaiser*** Google Brain lukaszkaiser@google.com

**Illia Polosukhin*** ‡ illia.polosukhin@gmail.com



### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to

# Transformers

- Scale efficiently
- Parallel process
- Attention to input meaning

**Attention Is All You Need**

**Ashish Vaswani*** Google Brain avaswani@google.com
**Noam Shazeer*** Google Brain noam@google.com
**Niki Parmar*** Google Research nikip@google.com
**Jakob Uszkoreit*** Google Research usz@google.com
**Llion Jones*** Google Research llion@google.com
**Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu
**Łukasz Kaiser*** Google Brain lukaszkaiser@google.com
**Illia Polosukhin*** ‡ illia.polosukhin@gmail.com

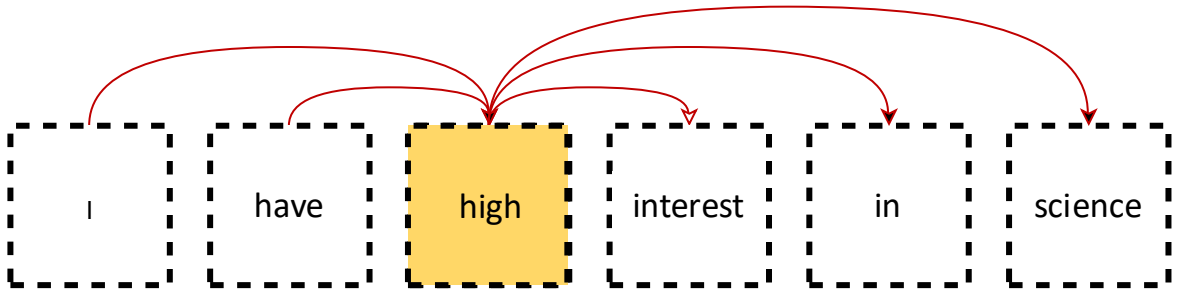
**Abstract**

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to

# Self-attention

# Transformers: Self-attention

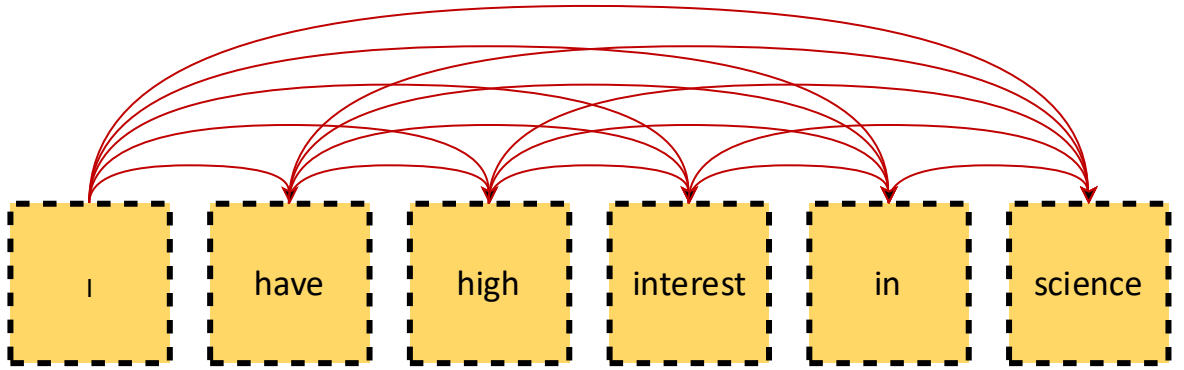
- Self-attention looks at each word in the text and decides which others are most important to understanding its meaning



# Transformers: Self-attention

- With self-attention, the transformer computes all the words in a sentence at the same time

- Capturing this context gives LLMs far more sophisticated capabilities to parse language

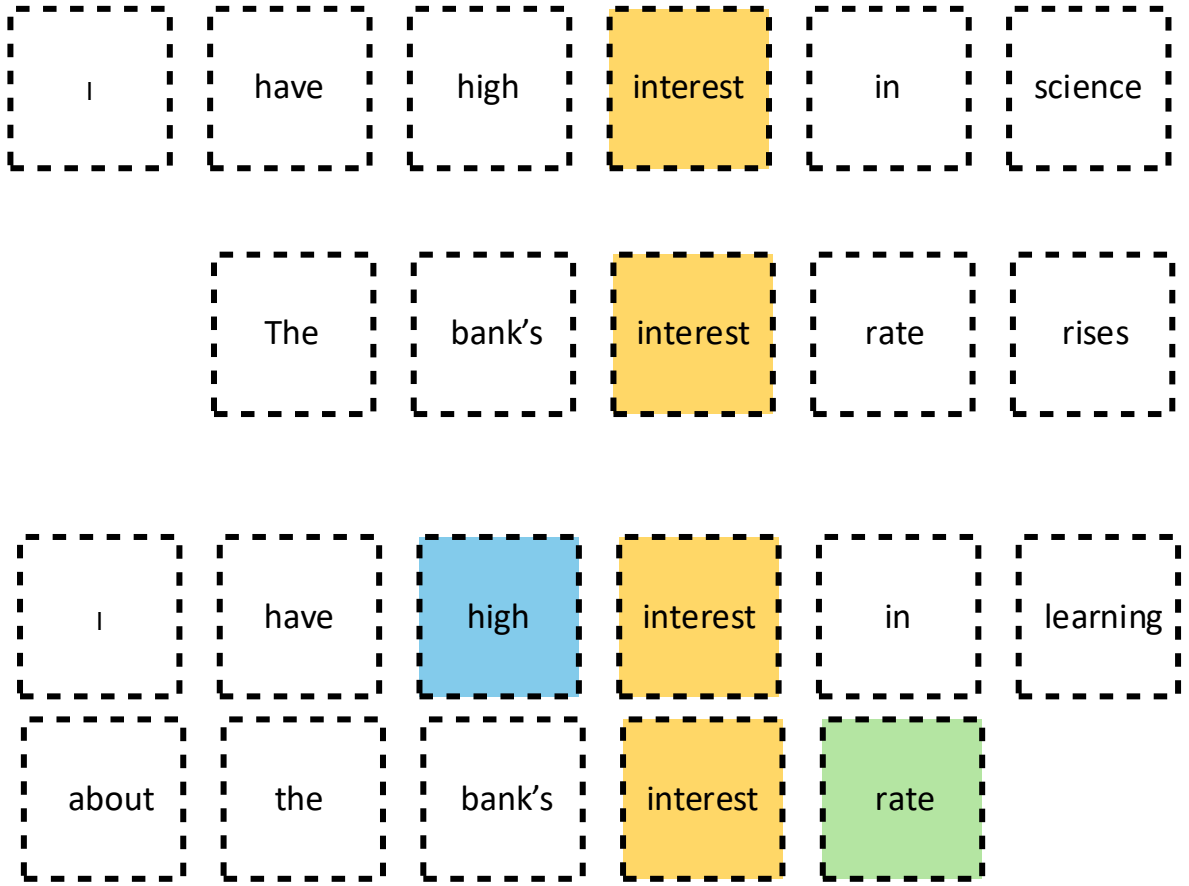


# Transformers: Self-attention

- *Interest*: noun
  opinion about science

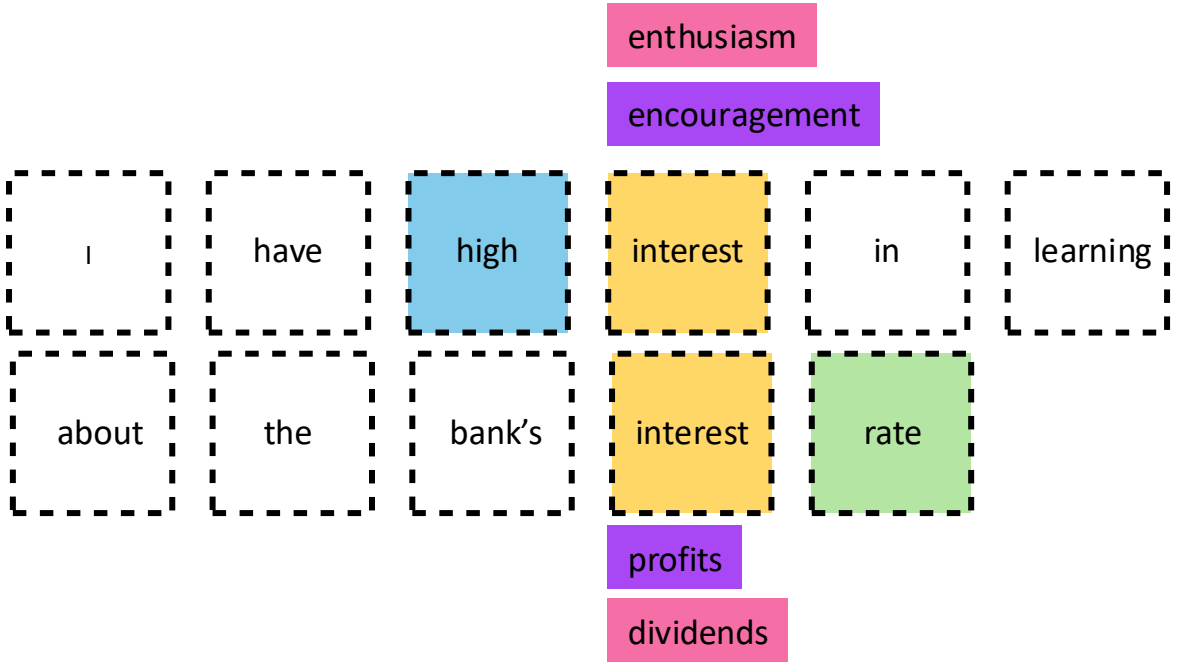
- *Interest*: noun
  financial aspect

- Combination



# Transformers: Self-attention

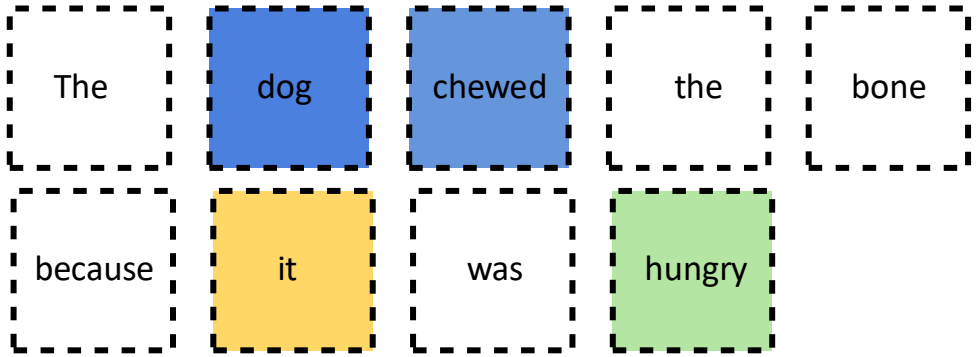
- Text generation
- Words are not always interchangeable
- It depends on the context



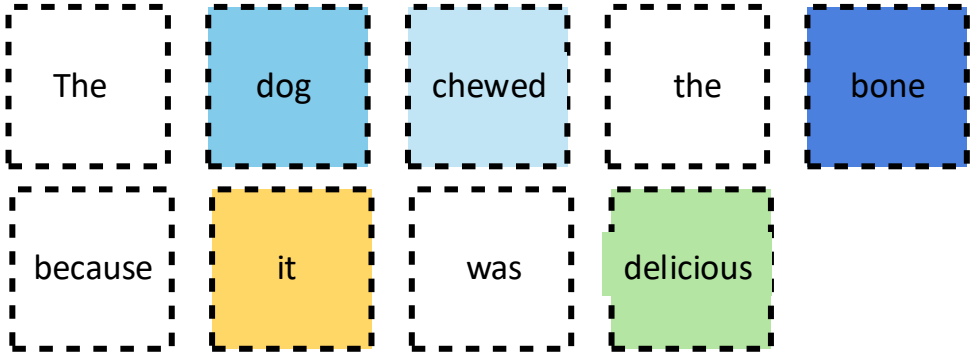

E.g. If a summary of this sentence was produced, you wouldn't have enthusiasm used when you were writing about interest rates

# Transformers: Self-attention

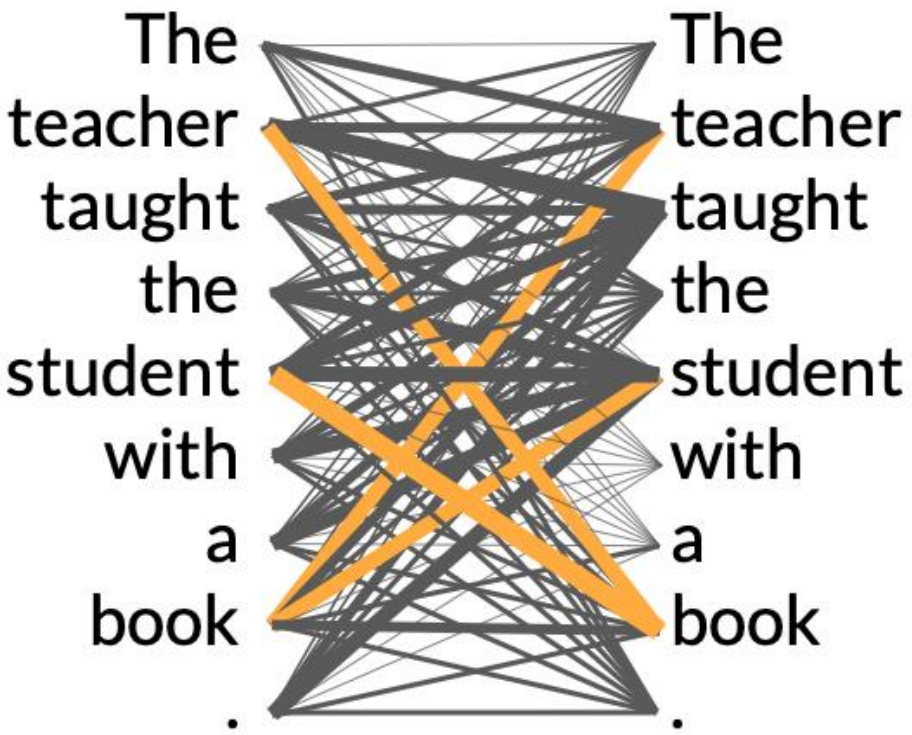
- *it* → referring to *dog*



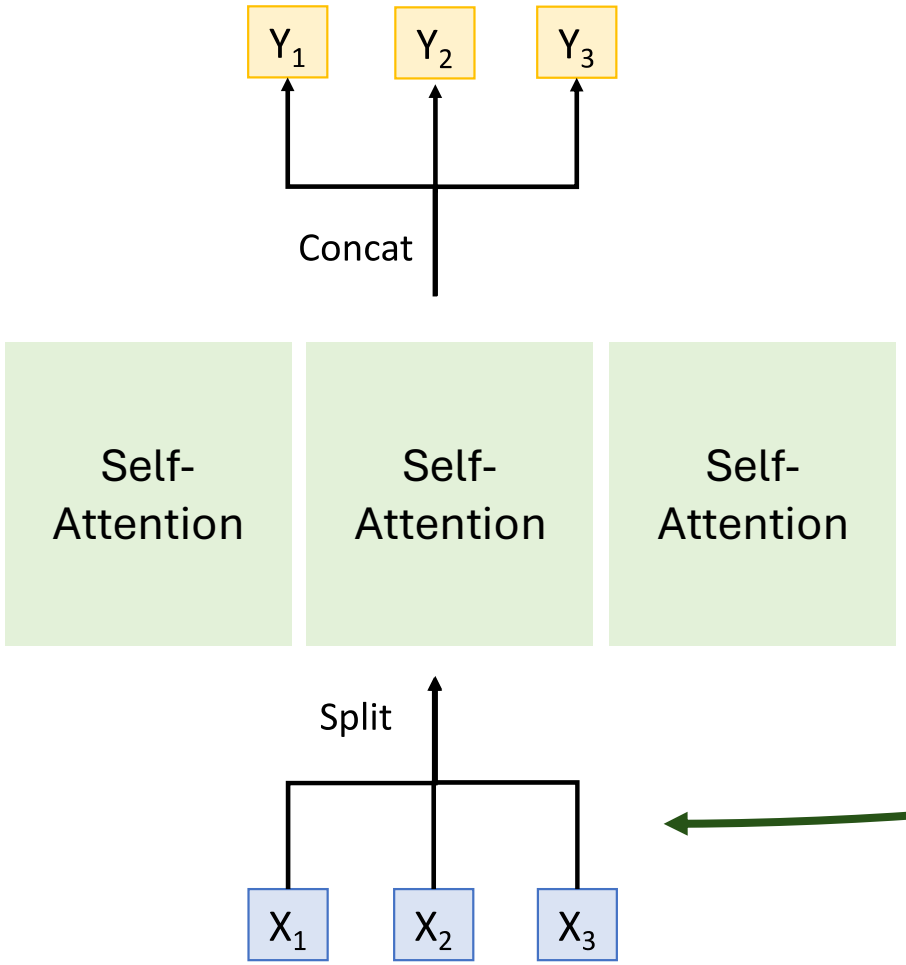

- *it* → referring to *bone*



# Transformers: Self-attention



# Multi-head self-attention



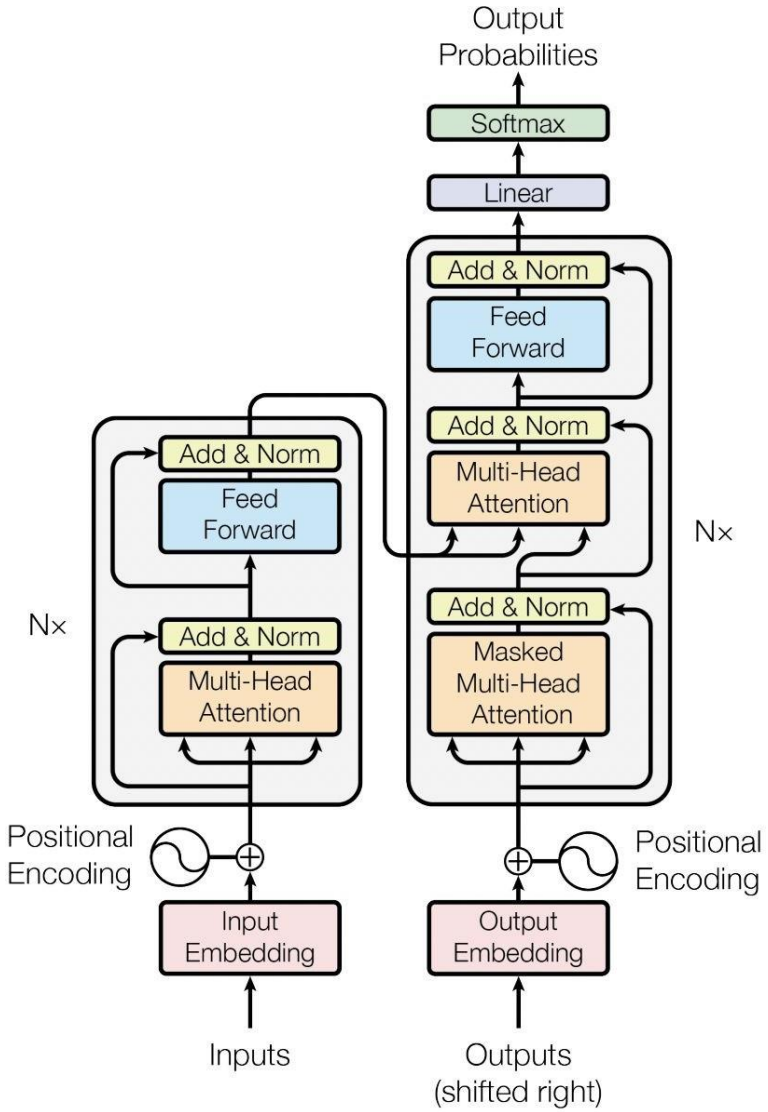

Choose number of heads h to run self-attention independently, in parallel to diversify learning

→

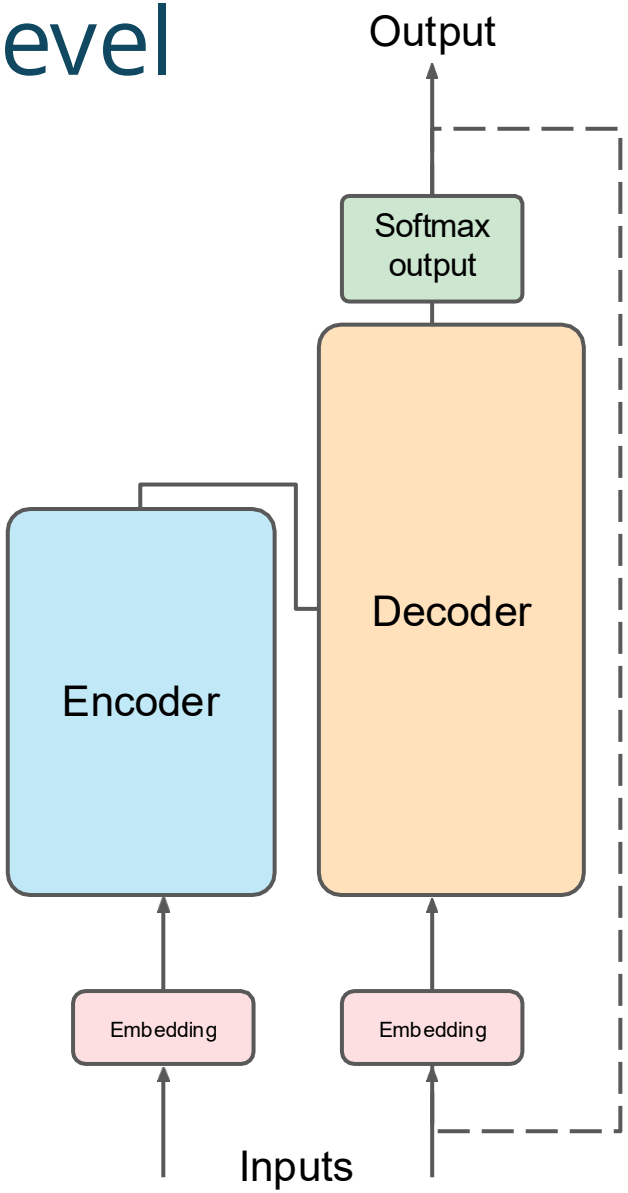
Input X, split into h chucks

# Transformers: High-level

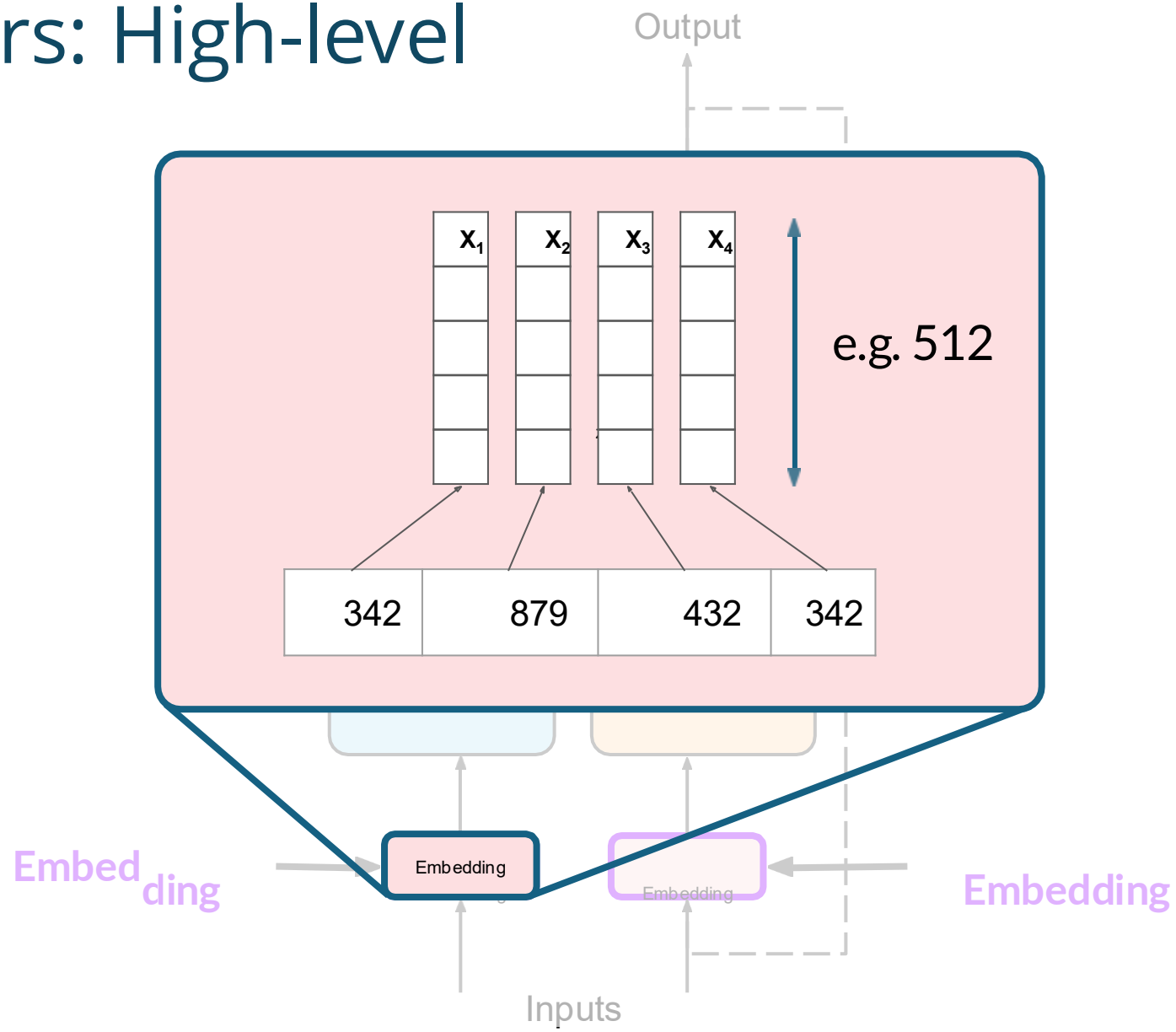
# Transformers: High-level



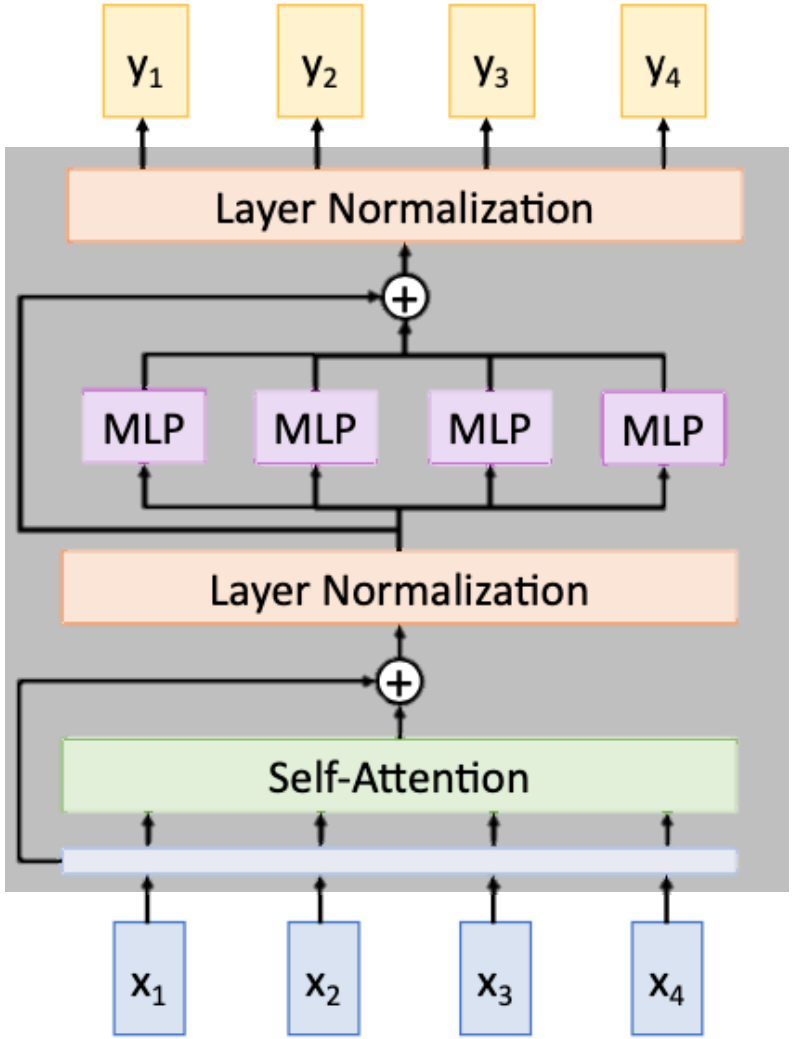
# Transformers: High-level



# Transformers: High-level



# Transformers



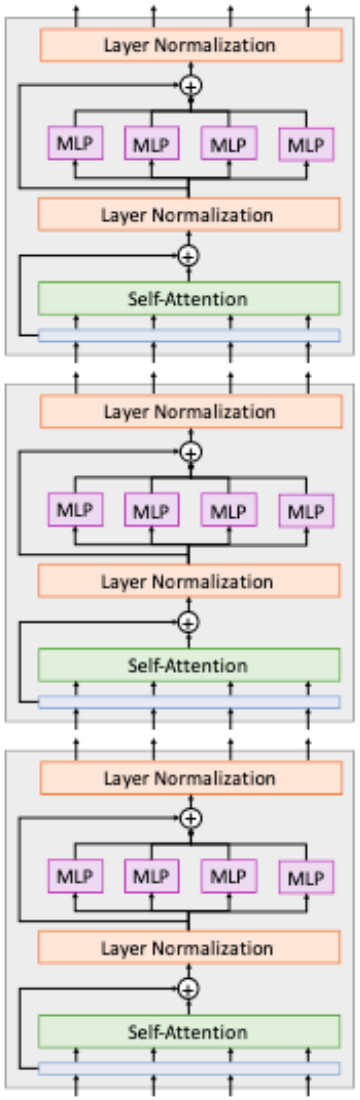

**Input**: set of vectors x
**Output**: set of vectors y

Self-attention is the only interaction between vectors

Layer normalization and MLP operate independently per vector

(+) highly scalable
(+) highly parallelizable

# Transformers




**Transformer**: sequence of transformer blocks

**Hyper-parameters: Attention is all you need**

12 blocks, $D_Q = 512$, 6 heads

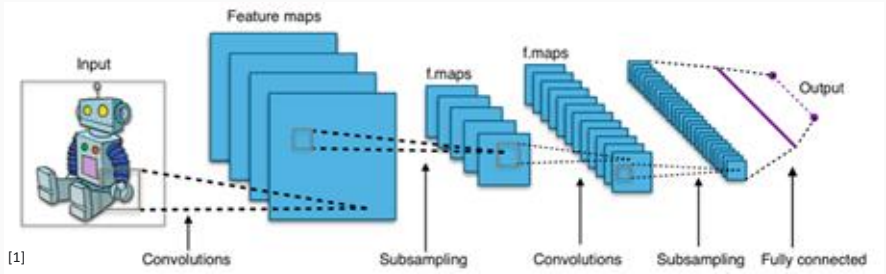
# I. Multimodal Large Language Models

- Introduction to LLMs
- Transformers & Self-attention
- **Towards unification of fields**
- Vision-Language Models

# Towards unification of fields

## Computer Vision

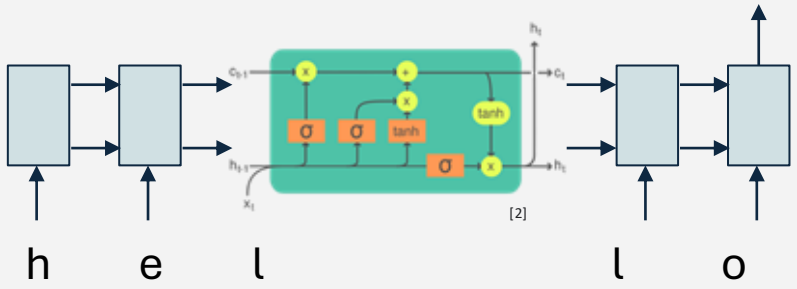
Convolutional NNs (+ResNets)




[1]

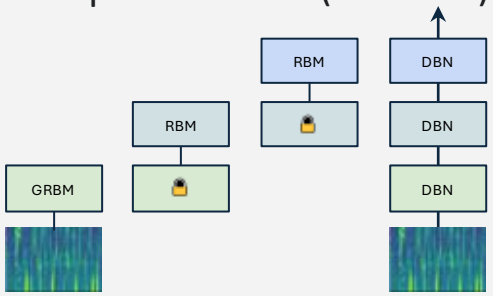
## Natural Lang. Proc.

Recurrent NNs (+LSTMs)



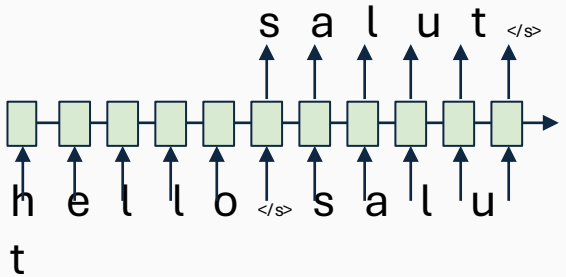

## Speech

Deep Belief Nets (+non-DL)



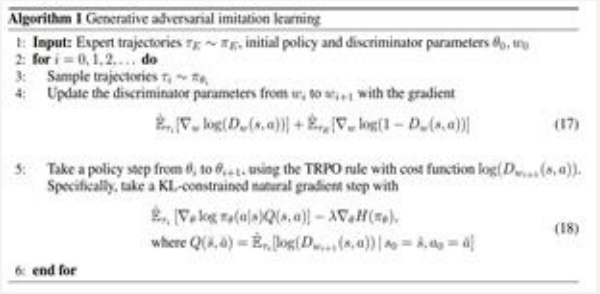

## Translation

Seq2Seq




## RL

BC/GAIL



**Algorithm 1** Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters $\theta_0, w_0$
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \quad (17)$$

5: Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$
$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \quad (18)$$

6: **end for**

[1] CNN image CC-BY-SA by Aphex34 for Wikipedia https://commons.wikimedia.org/wiki/File:Typical_cnn.png
[2] RNN image CC-BY-SA by GChe for Wikipedia https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg

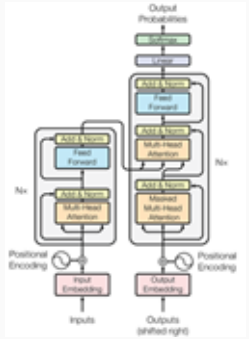
Slide credit: Lucas Beyer

The Transformer's takeover:

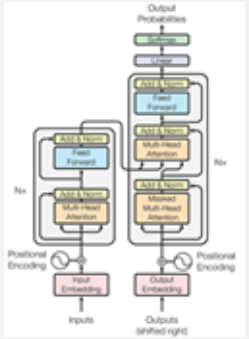
# One community at a time

# Next time: Transformers

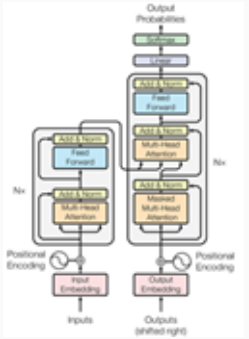
Computer Vision



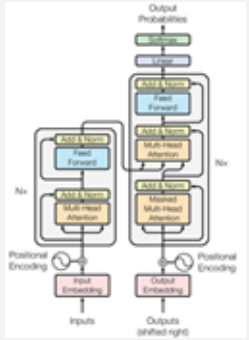
Natural Lang. Proc.



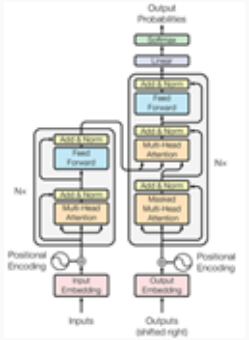
Reinf. Learning



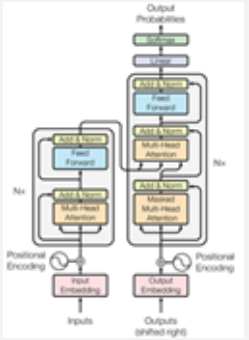
Speech



Translation



Graphs/Science



Slide credit: Lucas Beyer

# Vision Transformers

# Vision Transformers

1. Split an image into patches

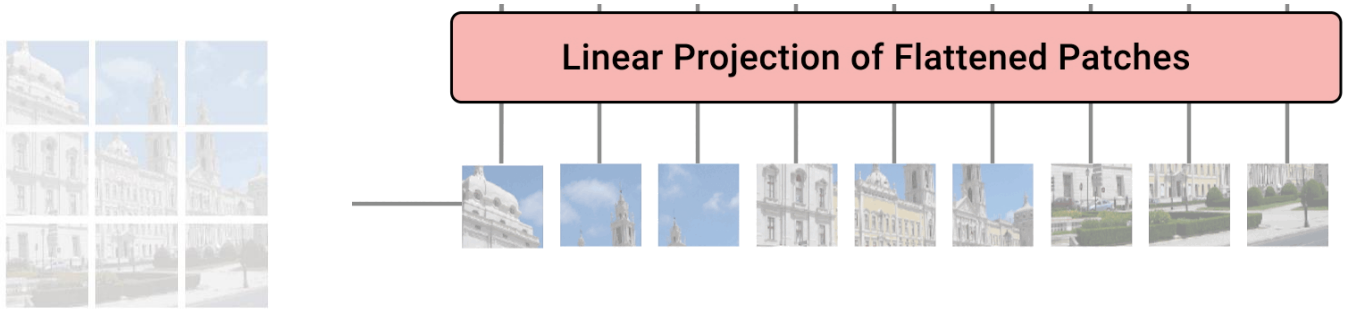
# Vision Transformers

1. Split an image into patches
2. Flatten the patches



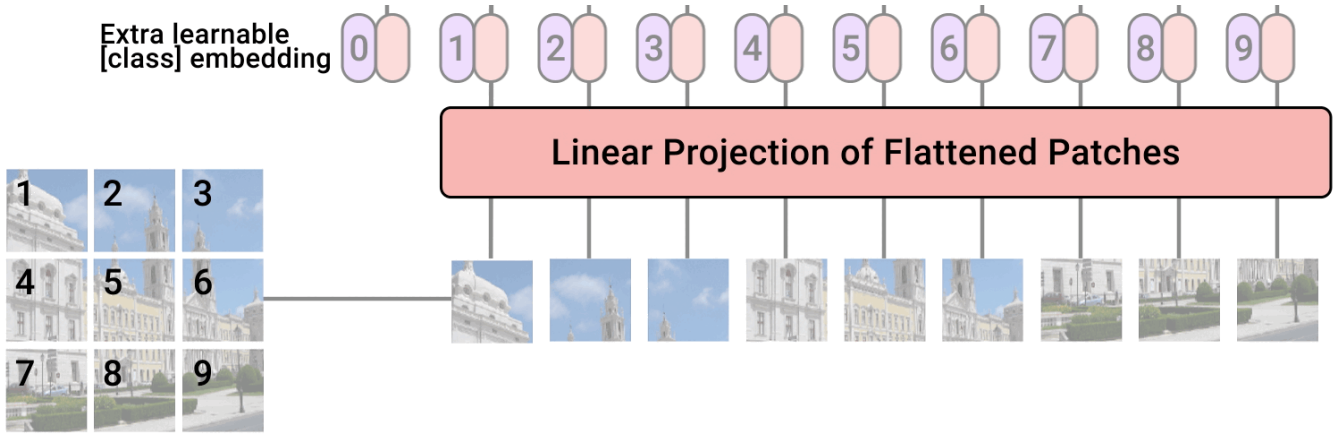
# Vision Transformers

1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the flattened patches



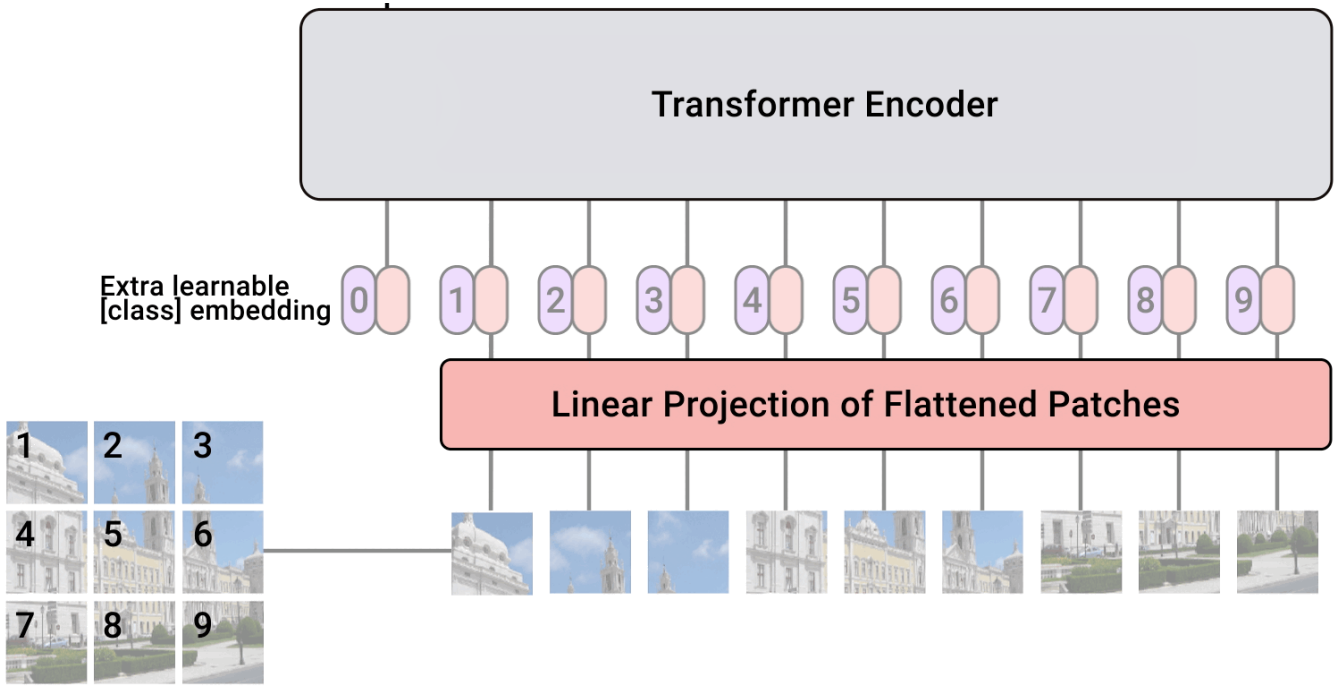
# Vision Transformers

1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the flattened patches
4. Add positional embeddings

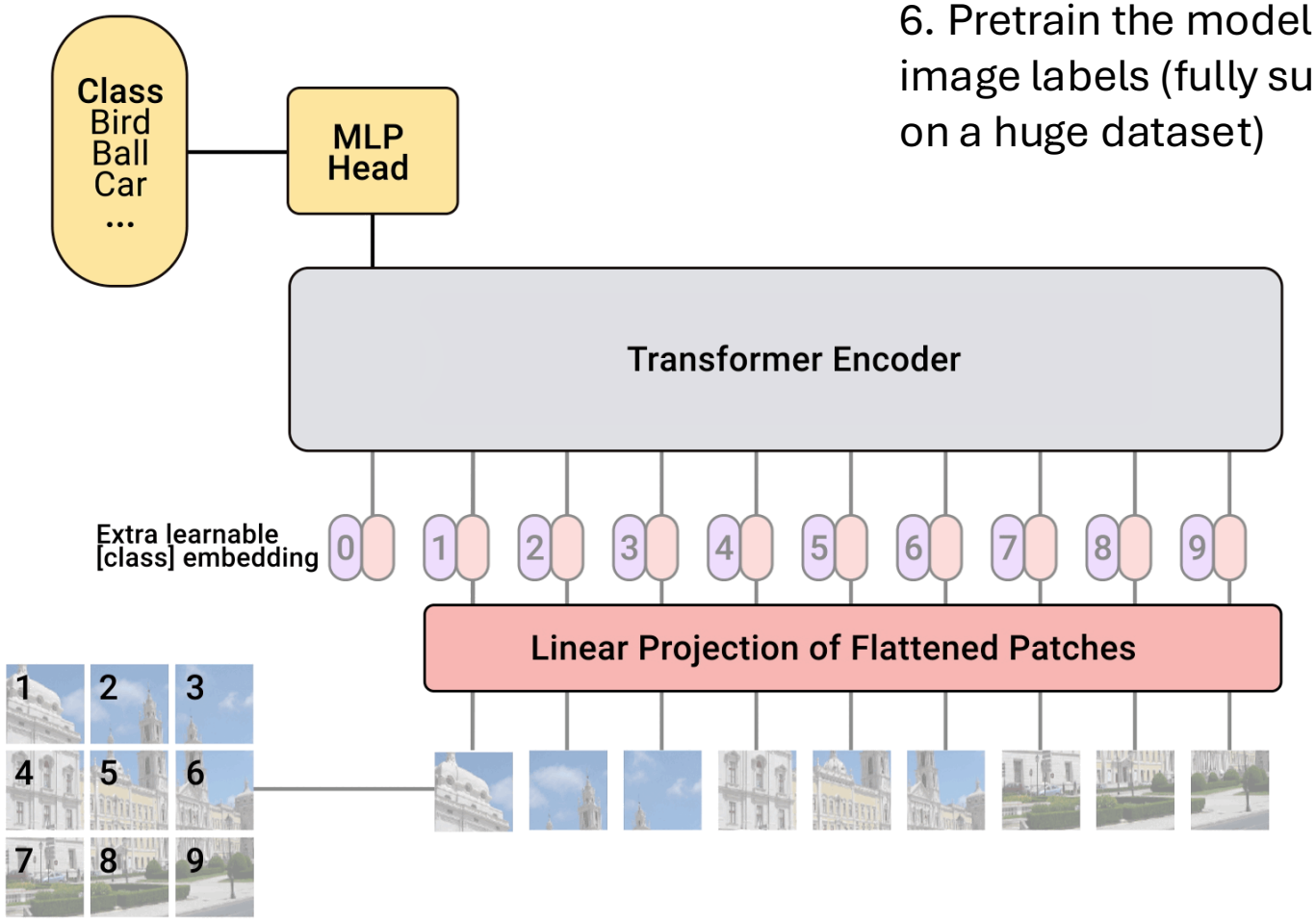


# Vision Transformers

5. Feed the sequence as an input to a standard transformer encoder



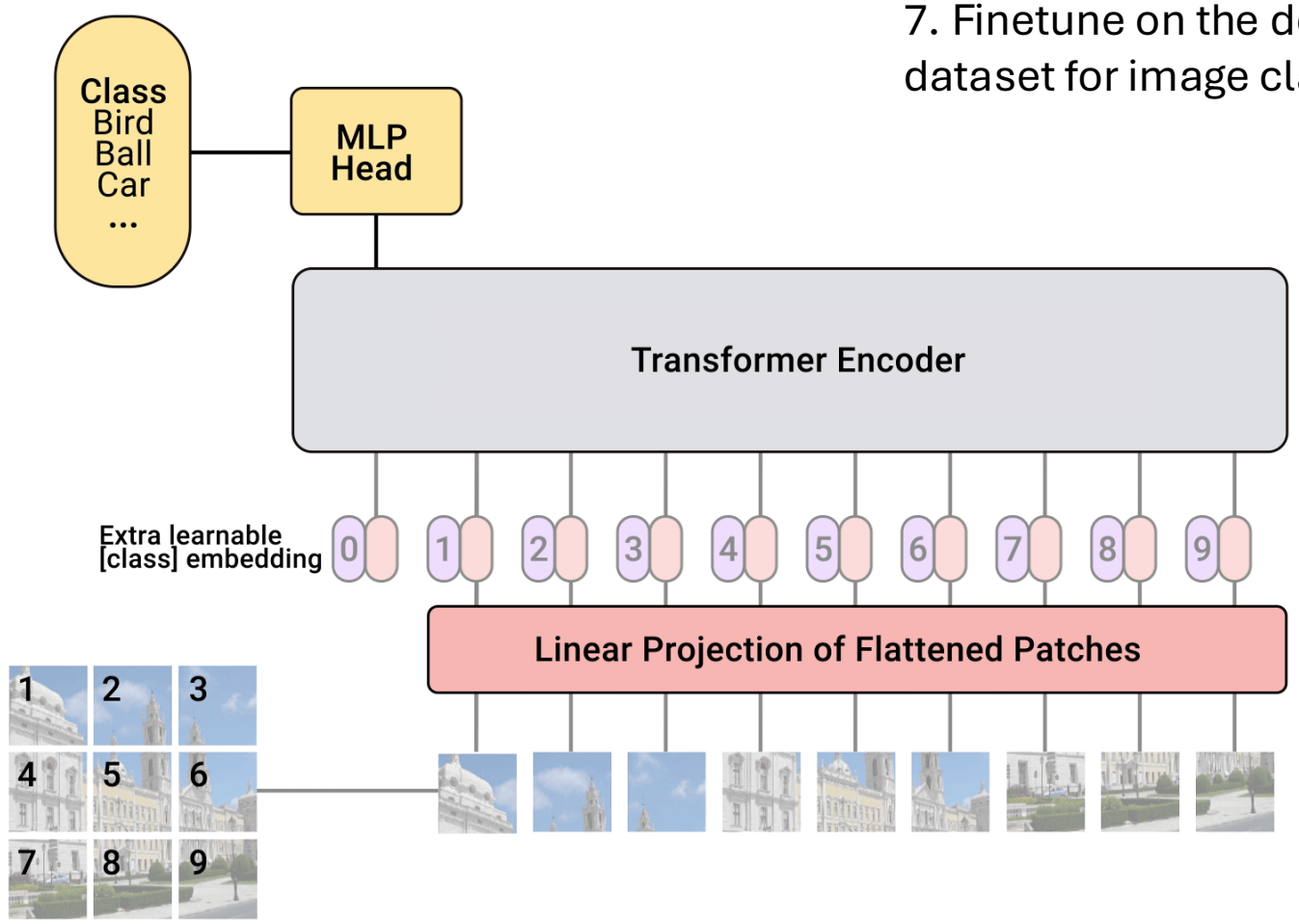
# Vision Transformers




6. Pretrain the model with image labels (fully supervised on a huge dataset)

# Vision Transformers

7. Finetune on the downstream dataset for image classification



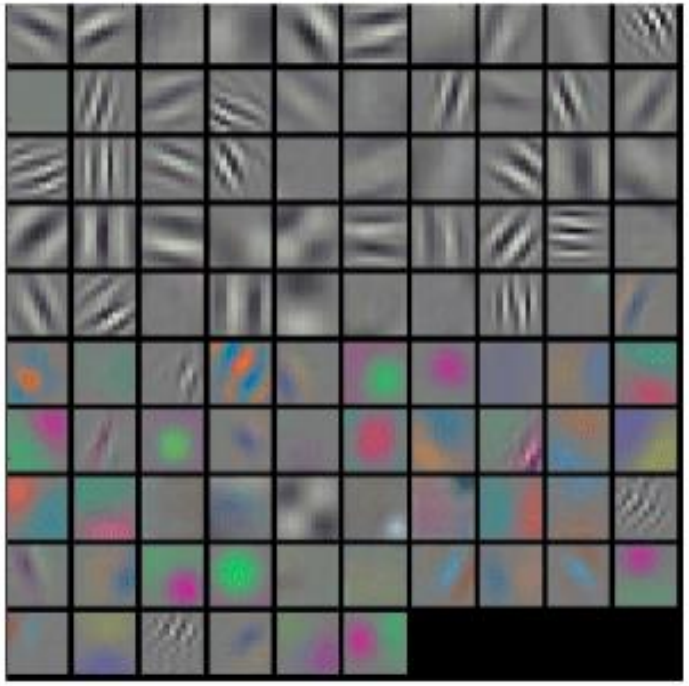
# Vision Transformers

# Vision Transformers

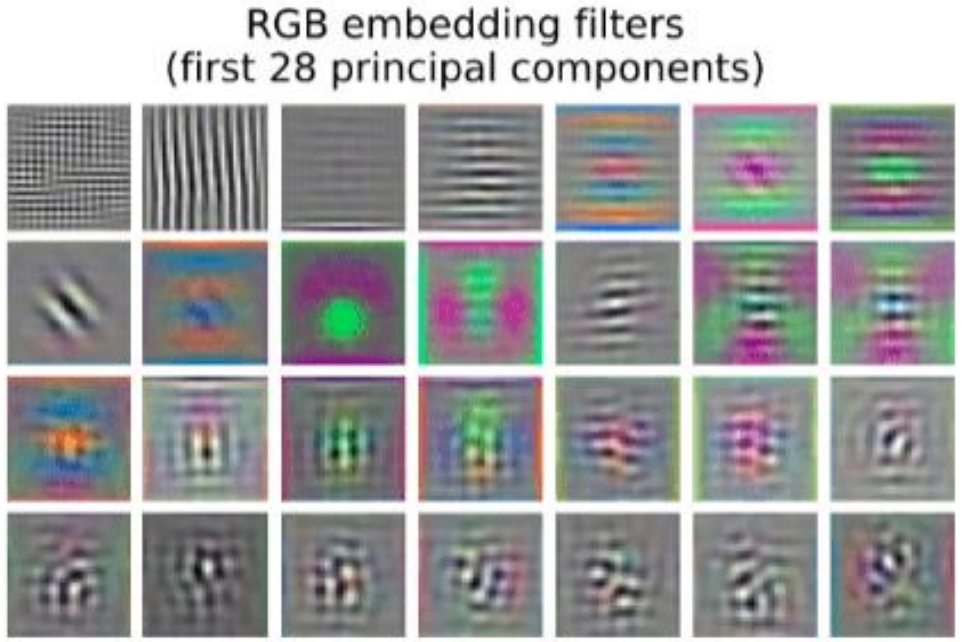
| Model | Layers | Hidden size $D$ | MLP size | Heads | Params |
|---|---|---|---|---|---|
| ViT-Base | 12 | 768 | 3072 | 12 | 86M |
| ViT-Large | 24 | 1024 | 4096 | 16 | 307M |
| ViT-Huge | 32 | 1280 | 5120 | 16 | 632M |

# Vision Transformers

Alexnet 1st conv filters



ViT 1st linear embedding filters



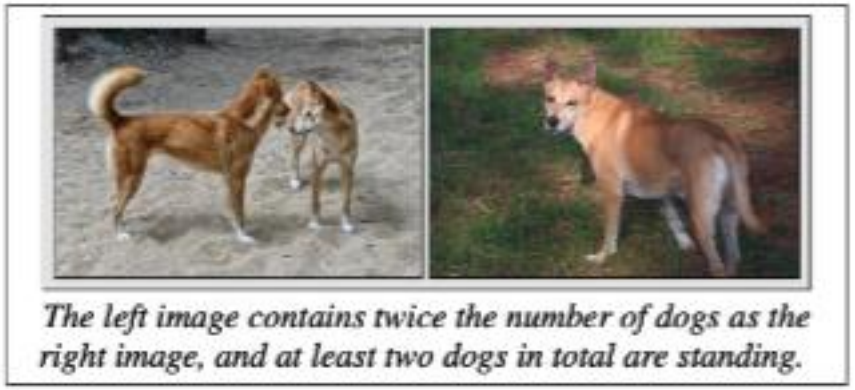
# I. Multimodal Large Language Models

- Introduction to LLMs
- Transformers & Self-attention
- Towards unification of fields
- **Vision-Language Models**

# Vision-Language Tasks

# Visual Reasoning

- Given an image (or a pair of images) determine if some natural language statement about the image(s) is true or false

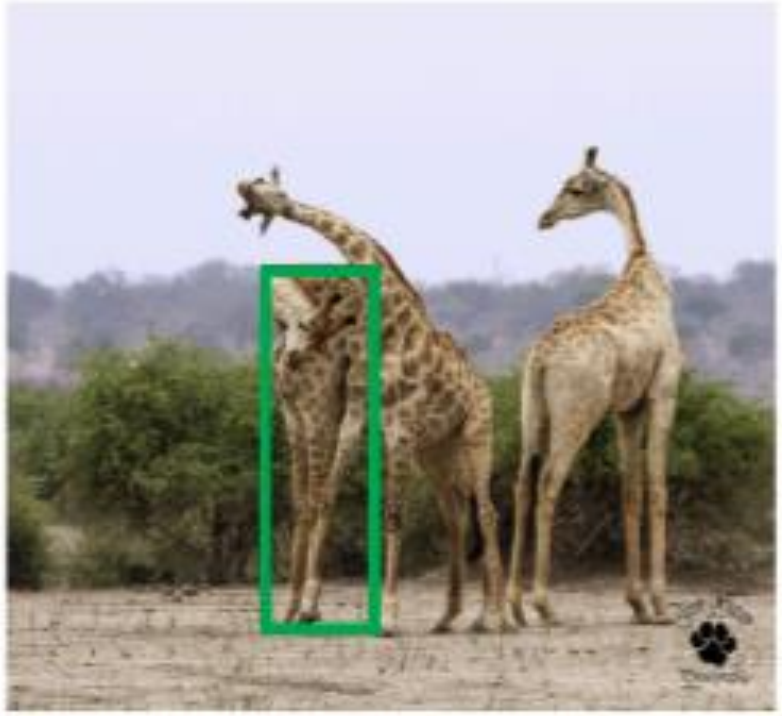


*The left image contains twice the number of dogs as the right image, and at least two dogs in total are standing.*

*One image shows exactly two brown acorns in back-to-back caps on green foliage.*

# Visual Grounding

- Locate an object in some image given a natural language description



RefCOCO:
1. giraffe on left
2. first giraffe on left

RefCOCO+:
1. giraffe with lowered head
2. giraffe head down

RefCOCOg:
1. an adult giraffe scratching its back with its horn
2. giraffe hugging another giraffe

# Visual Question Answering

- Given an image (or images), respond to arbitrary, potentially open-ended questions about the content.

# Visual Caption Generation

- Create natural language descriptions of content of some image

**Ground Truth Caption:** A little boy runs away from the approaching waves of the ocean.

**Generated Caption:** A young boy is running on the beach.

**Ground Truth Caption:** A brunette girl wearing sunglasses and a yellow shirt.

**Generated Caption:** A woman in a black shirt and sunglasses smiles.

# Early works on VLMs

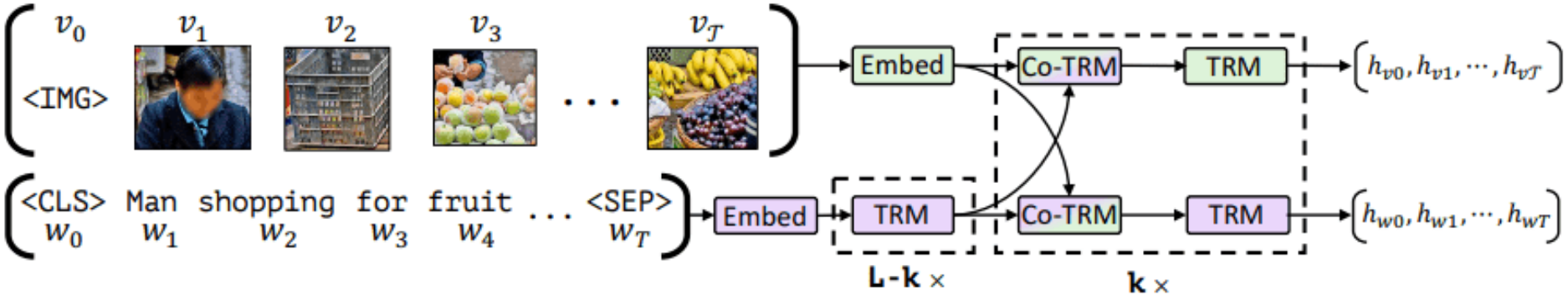
# Early works on VLMs

- **Idea**: process language and images at the same time with a transformer-like architecture

- Two categories:
  - Two-stream models
    - process text and images using two separate modules
      e.g., ViLBERT, LXMERT
  - Single-stream models
    - Encode both modalities within the same module
      e.g., VisualBERT, VL-BERT, UNITER

# Two-stream models
# ViLBERT

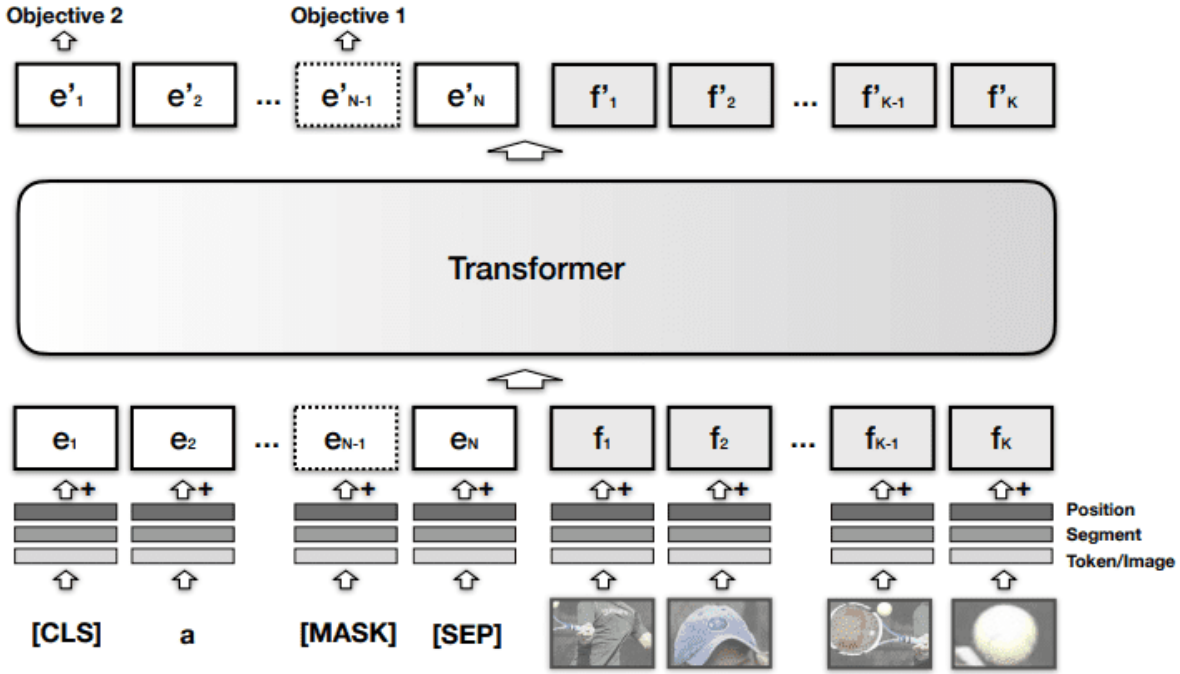
- **Text** stream (purple)
  - the weights are set by pretraining the model on standard text corpus
- **Image** stream (green): Object detection pipeline (Faster R-CNN)
- **Training**: dataset of image-text pairs
- **Objective**: understand the relationship between text and images



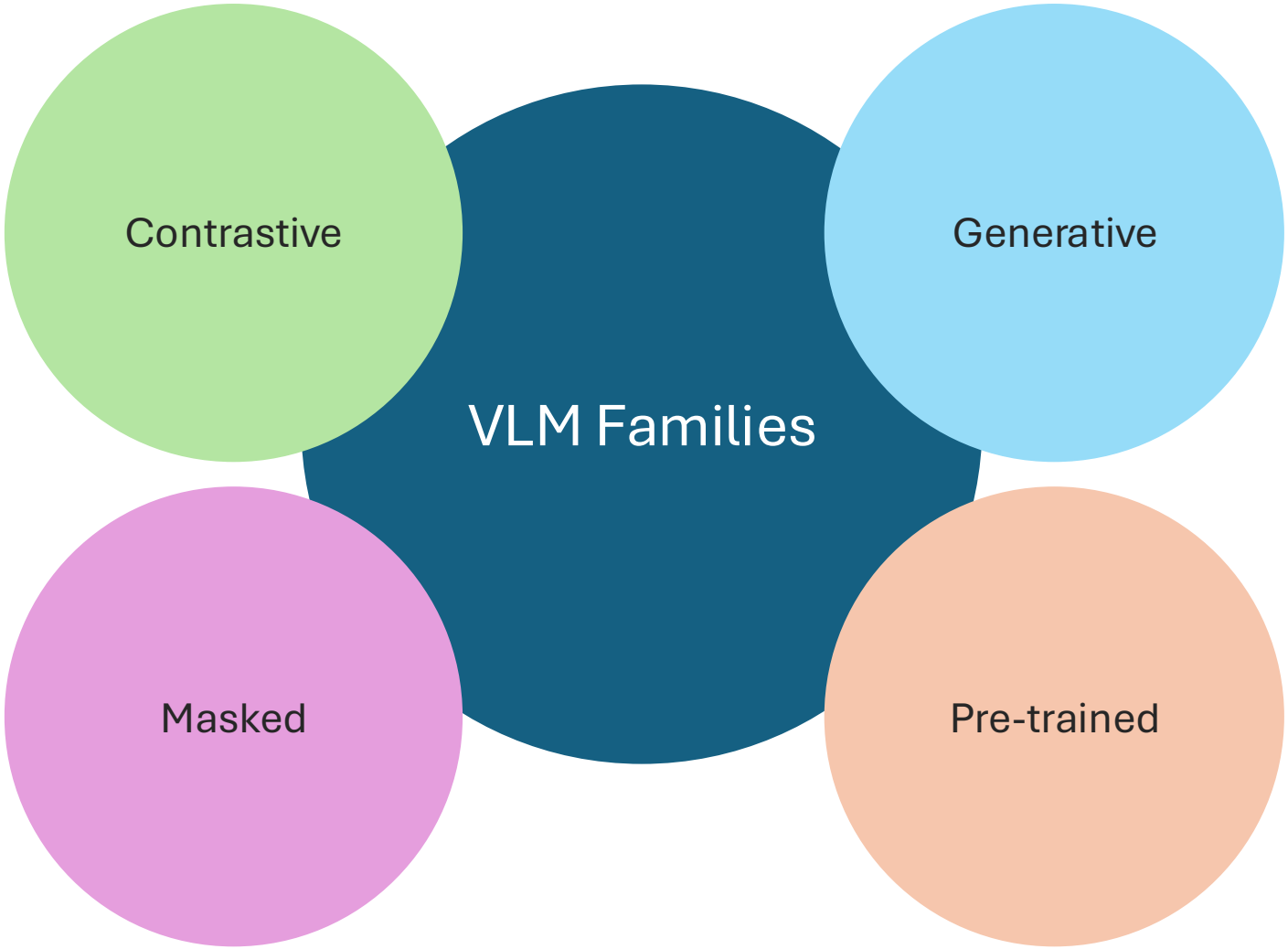
# Single-stream models
# VisualBERT

- VisualBERT: combines image + language with transformer
- Uses self-attention to discover alignments between them

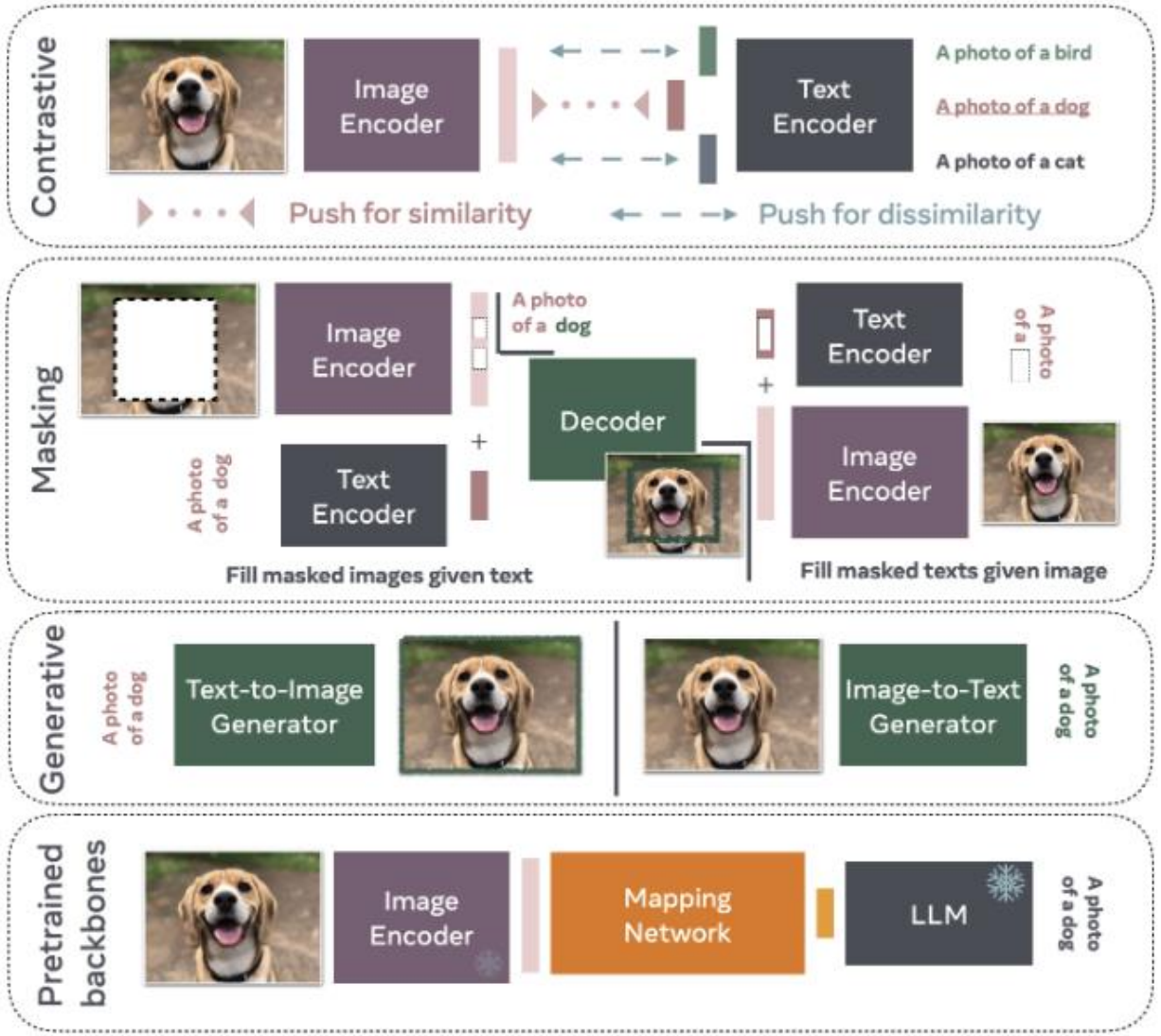
- Add visual embedding to BERT:
  - A visual feature representation of the region produced by a CNN
  - A segment embedding that distinguishes image from text embeddings
  - A positional embedding to align regions with words if provided in the input



# Modern work on VLMs



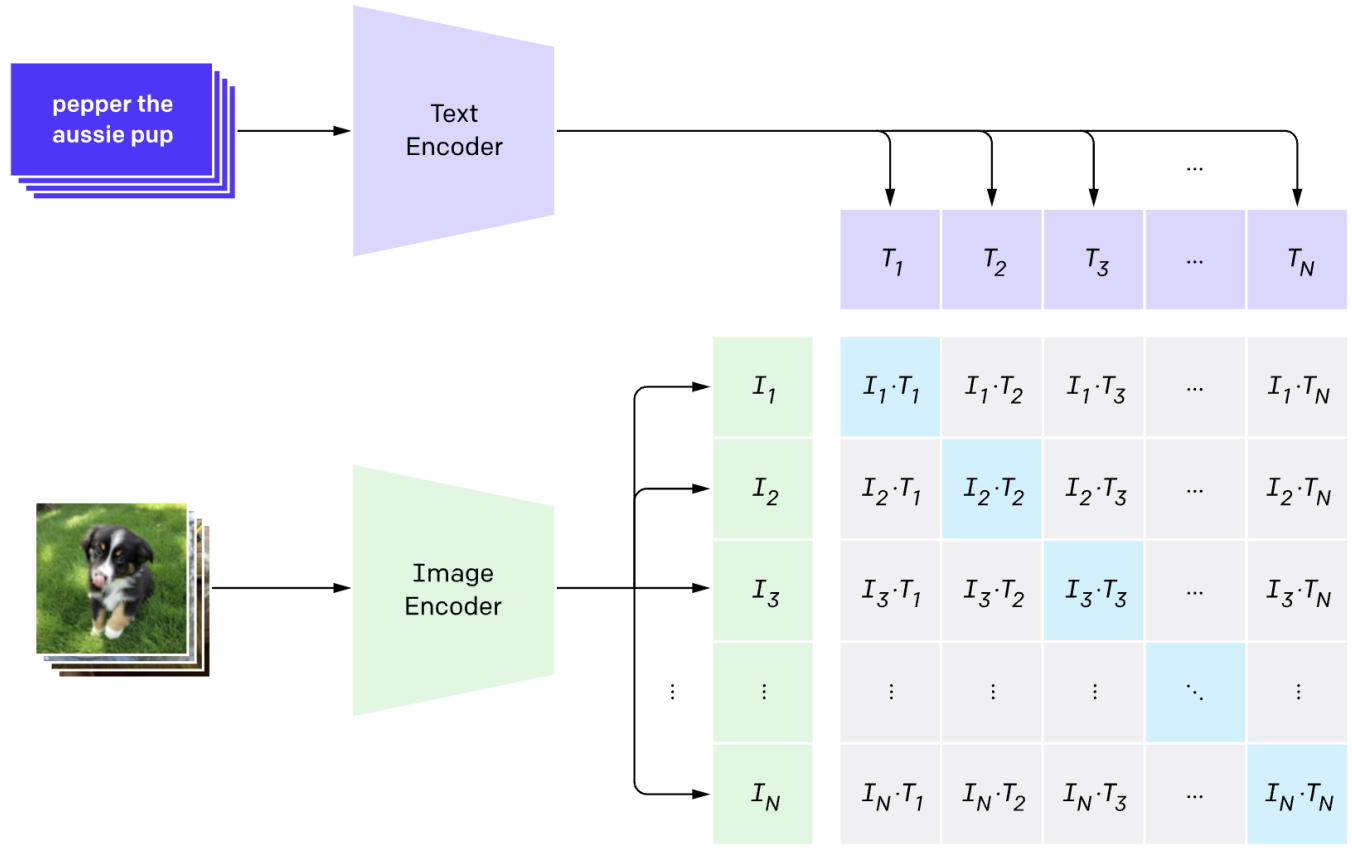
# VLM Families



# 1. Contrastive learning

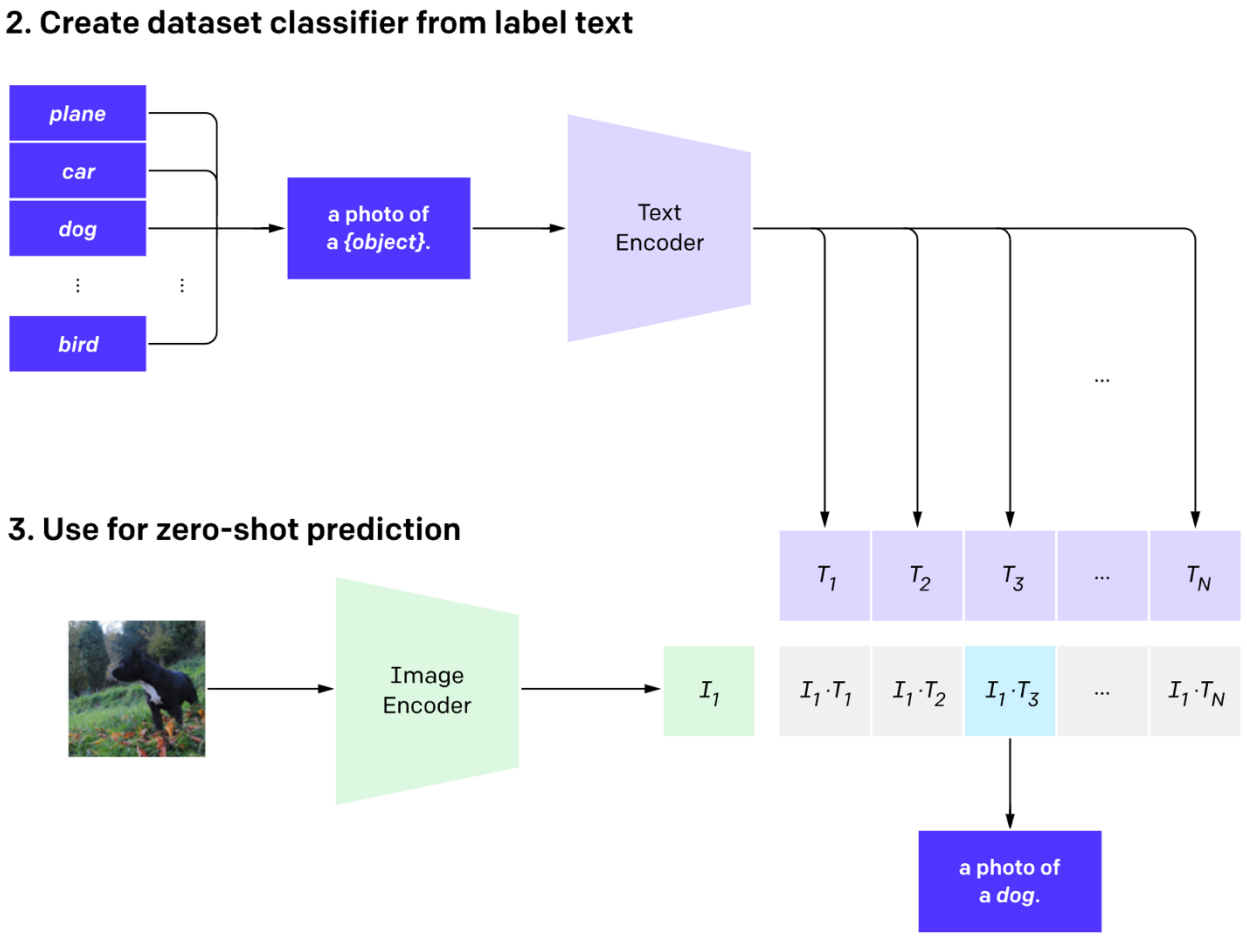
# CLIP

5 Januray 2021
12+k citations in 3 years



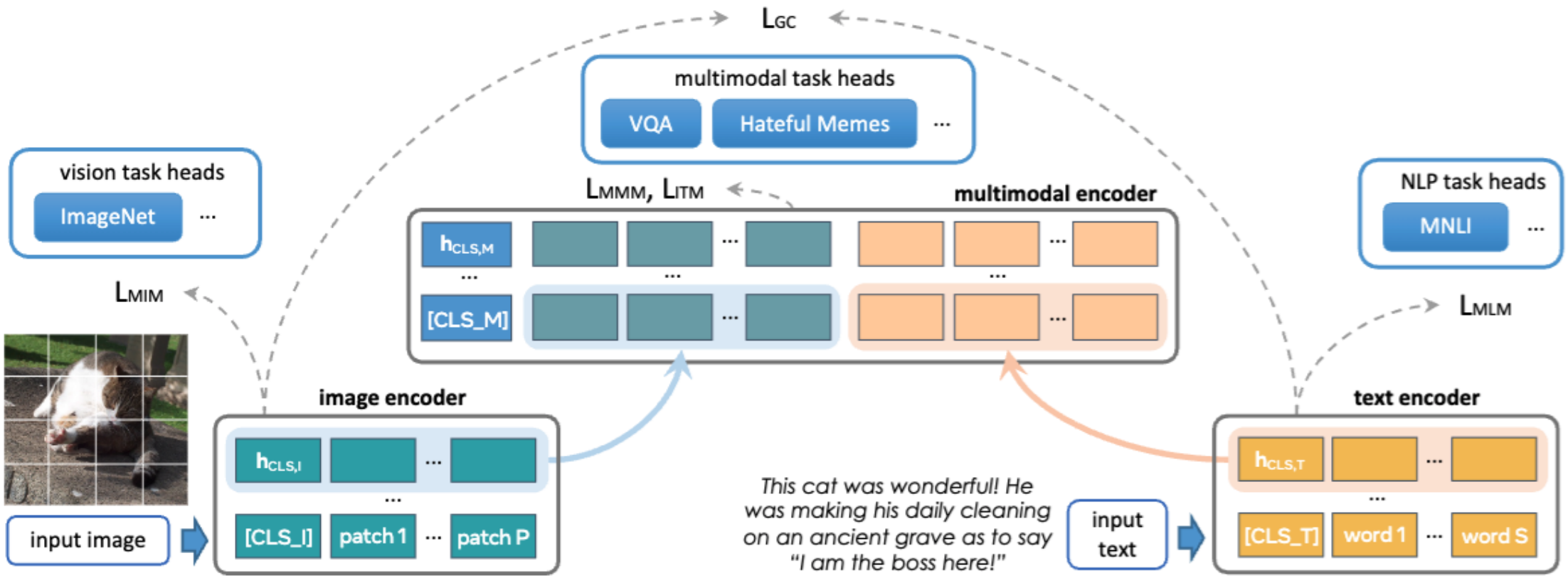
5 Januray 2021
12+k citations in 3 years

# CLIP



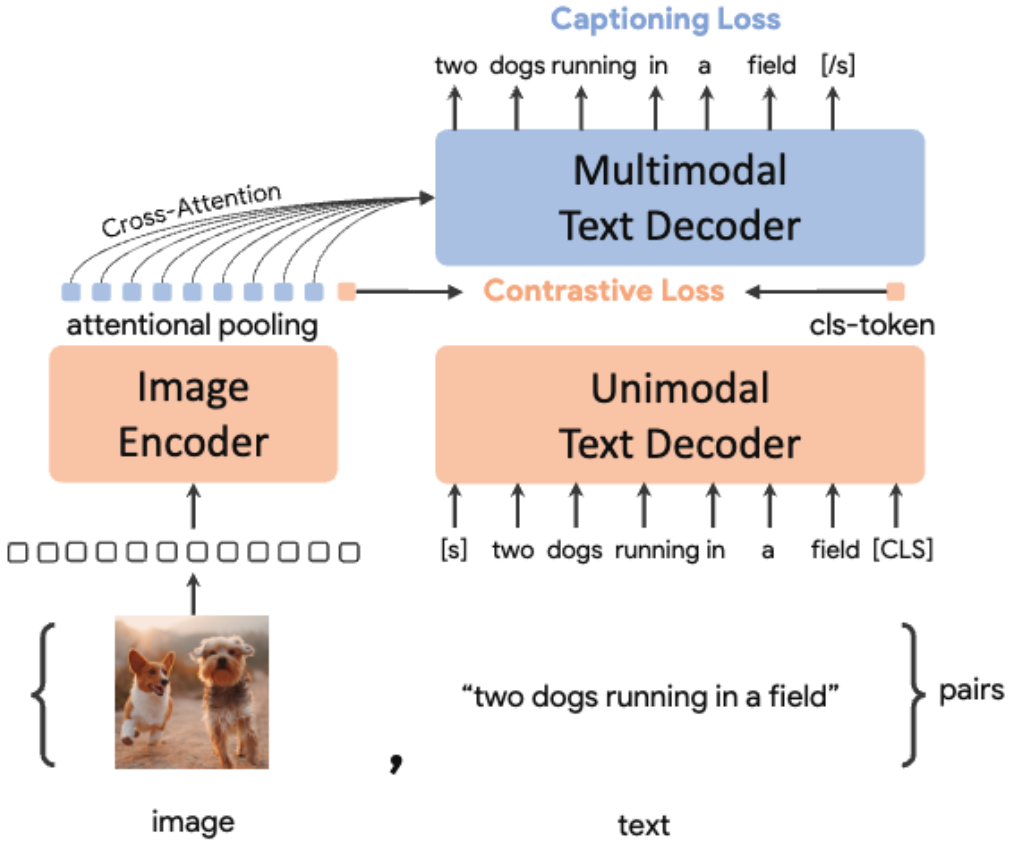
# 2. Masking

# Foundational Language And Vision Alignment (FLAVA)



# 3. Generative

# Learning a text generator: Contrastive Captioner (CoCa)

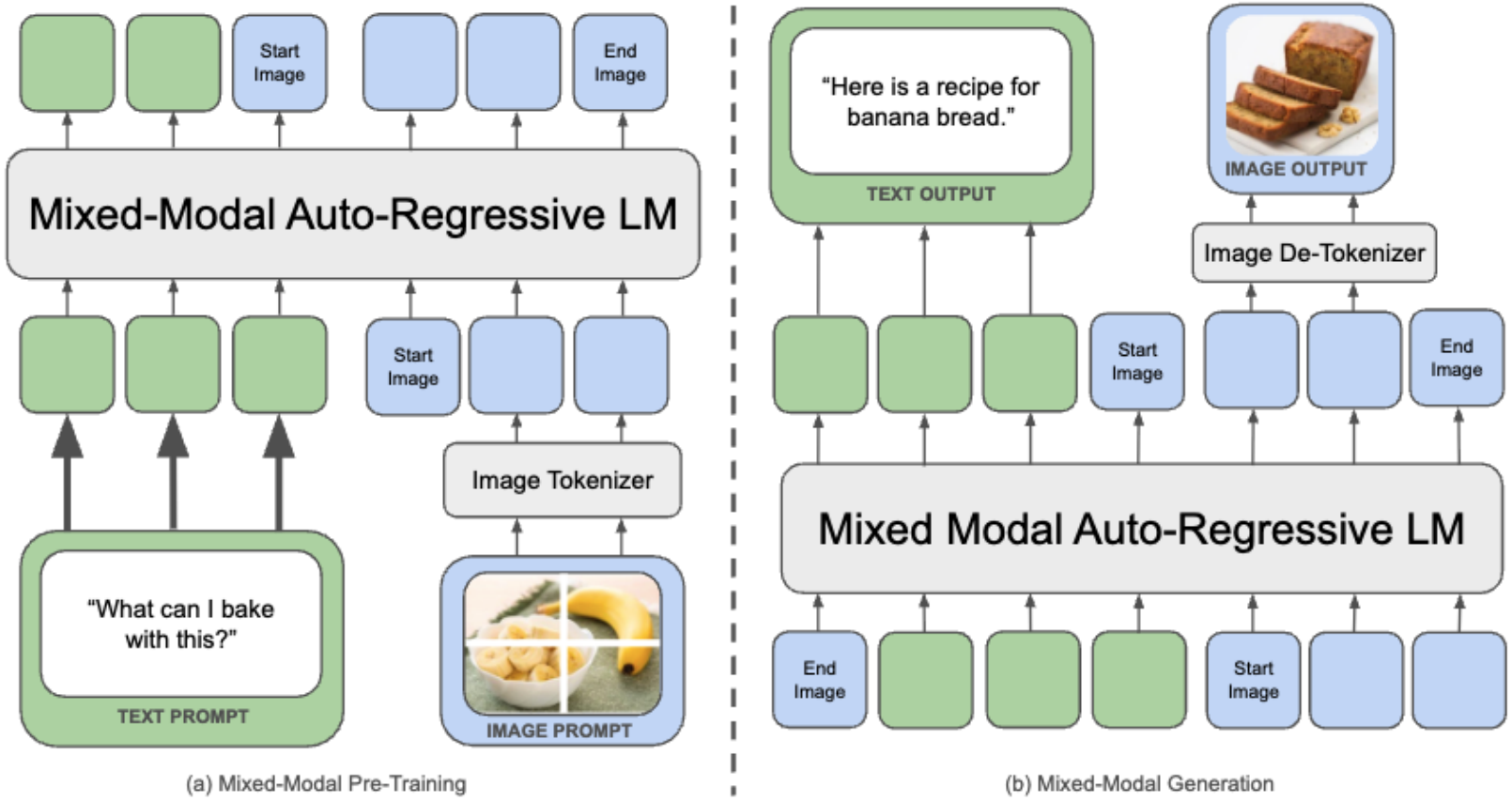



**Algorithm 1** Pseudocode of Contrastive Captioners architecture.

```
# image, text.ids, text.labels, text.mask: paired {image, text} data
# con_query: 1 query token for contrastive embedding
# cap_query: N query tokens for captioning embedding
# cls_token_id: a special cls_token_id in vocabulary

def attentional_pooling(features, query):
  out = multihead_attention(features, query)
  return layer_norm(out)

img_feature = vit_encoder(image) # [batch, seq_len, dim]
con_feature = attentional_pooling(img_feature, con_query) # [batch, 1, dim]
cap_feature = attentional_pooling(img_feature, cap_query) # [batch, N, dim]

ids = concat(text.ids, cls_token_id)
mask = concat(text.mask, zeros_like(cls_token_id)) # unpad cls_token_id
txt_embs = embedding_lookup(ids)
unimodal_out = lm_transformers(txt_embs, mask, cross_attn=None)
multimodal_out = lm_transformers(
    unimodal_out[:, :-1, :], mask, cross_attn=cap_feature)
cls_token_feature = layer_norm(unimodal_out)[:, -1:, :] # [batch, 1, dim]

con_loss = contrastive_loss(con_feature, cls_token_feature)
cap_loss = softmax_cross_entropy_loss(
    multimodal_out, labels=text.labels, mask=text.mask)
```

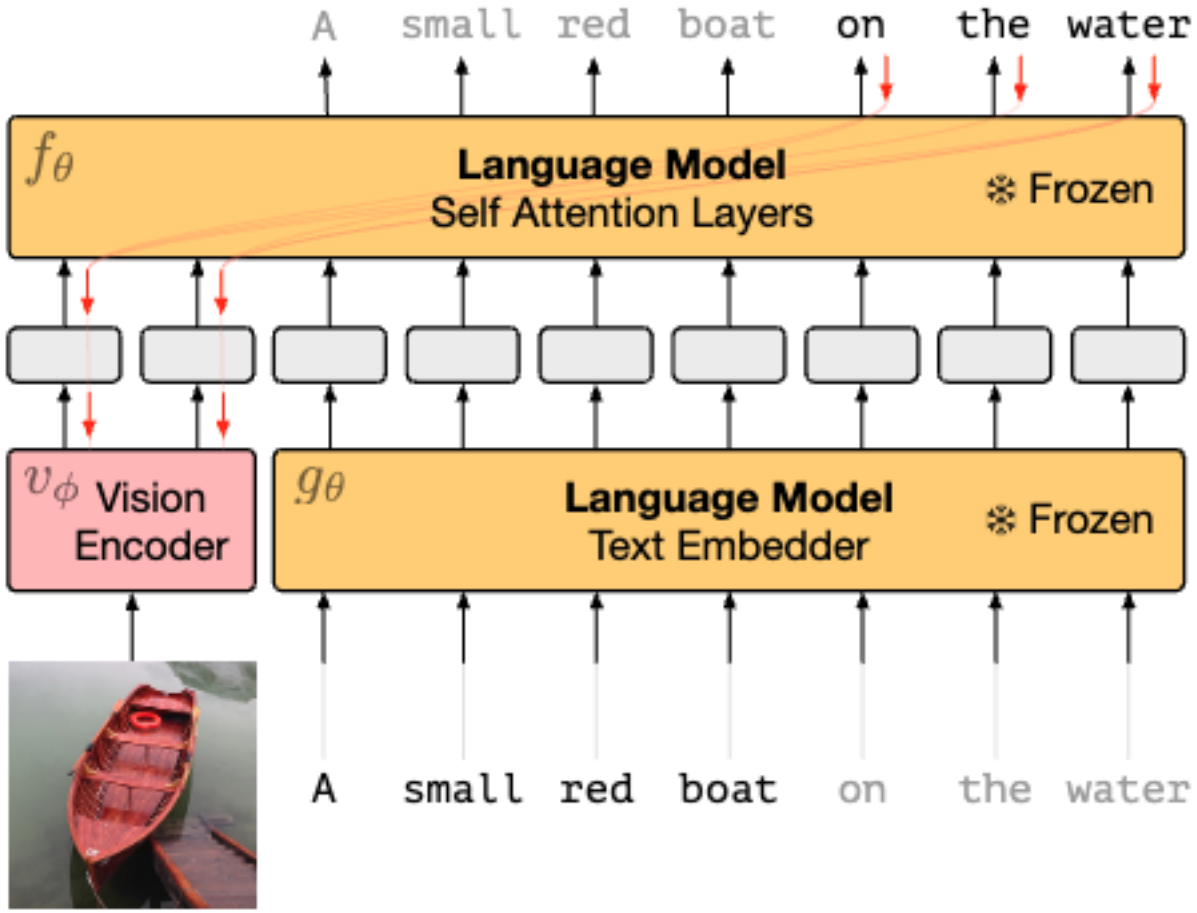
`vit_encoder`: vision transformer based encoder; `lm_transformer`: language-model transformers.

# Chameleon

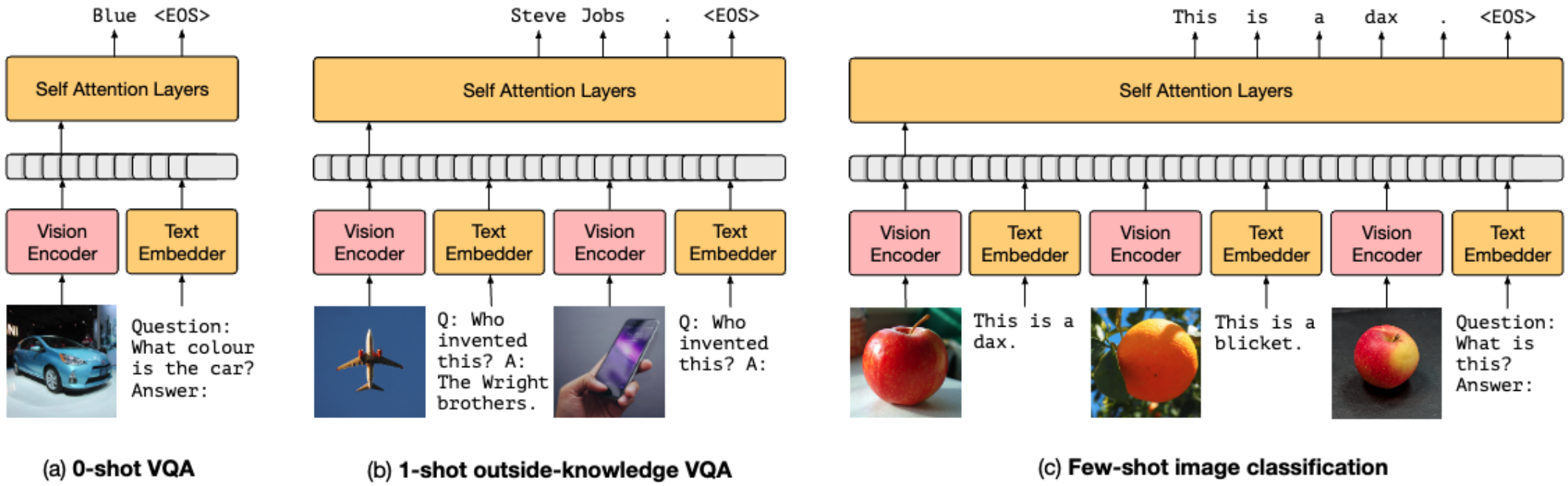

Start Image
End Image
Mixed-Modal Auto-Regressive LM
Start Image
Image Tokenizer
"What can I bake with this?"
TEXT PROMPT
IMAGE PROMPT
(a) Mixed-Modal Pre-Training
"Here is a recipe for banana bread."
TEXT OUTPUT
IMAGE OUTPUT
Image De-Tokenizer
Start Image
End Image
Mixed Modal Auto-Regressive LM
End Image
Start Image
(b) Mixed-Modal Generation

# 4. Pre-trained

# Frozen


A small red boat on the water
$f_\theta$
Language Model
Self Attention Layers
Frozen
$v_\phi$ Vision Encoder
$g_\theta$
Language Model
Text Embedder
Frozen
A small red boat on the water

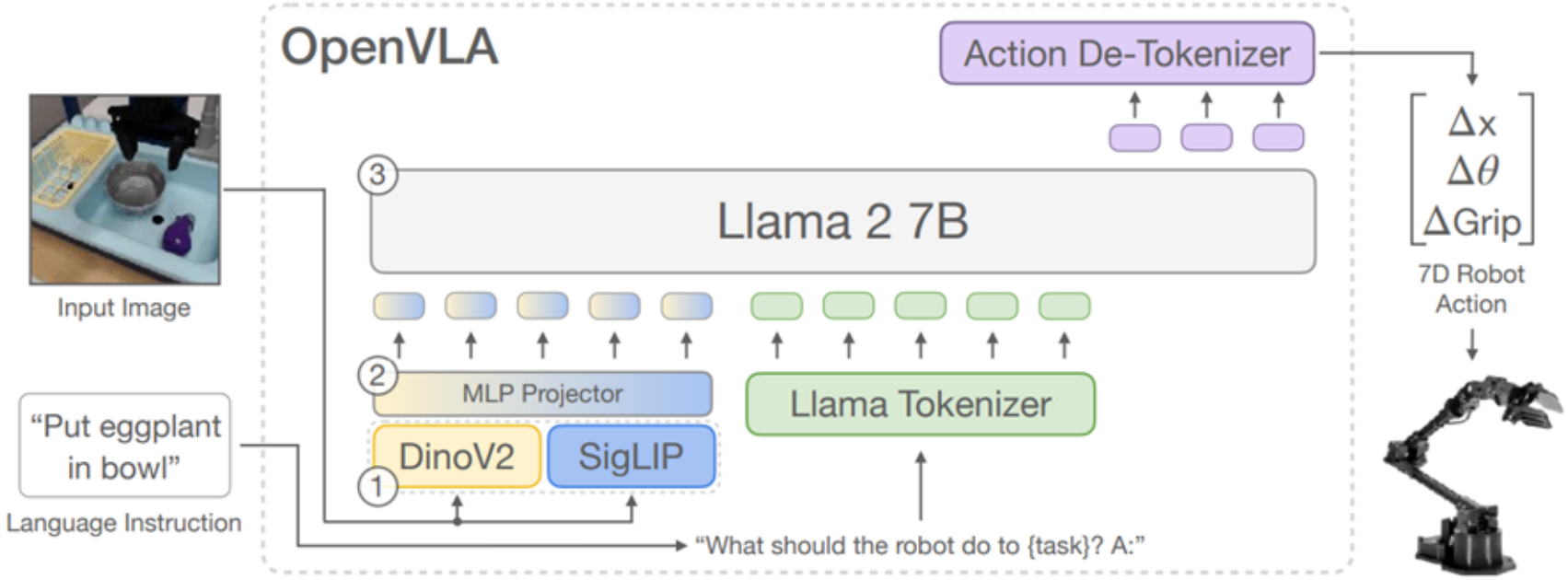
# Frozen




(a) **0-shot VQA** (b) **1-shot outside-knowledge VQA** (c) **Few-shot image classification**

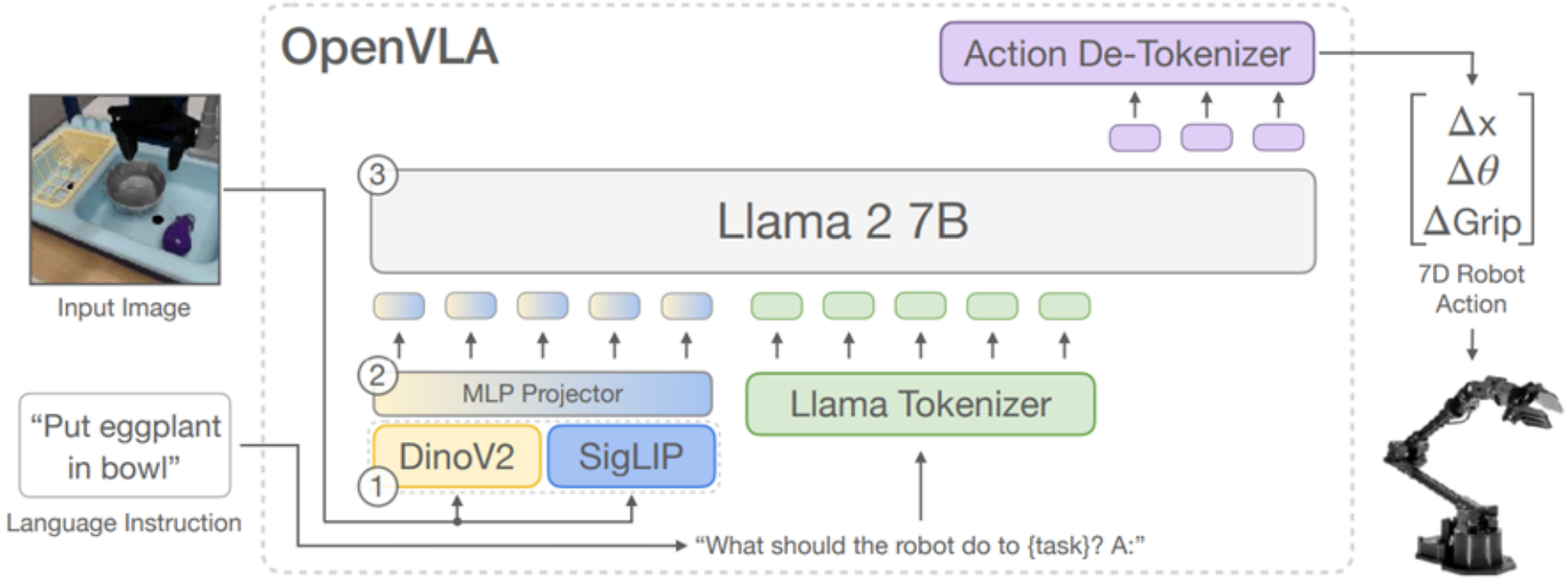
# From VLMs to VLAs: Vision-Language Action models

# Vision-Language-Action models (VLA)




OpenVLA: An Open-Source Vision-Language-Action Model, Kim et al., Arxiv 2024

# Vision-Language-Action models (VLA)



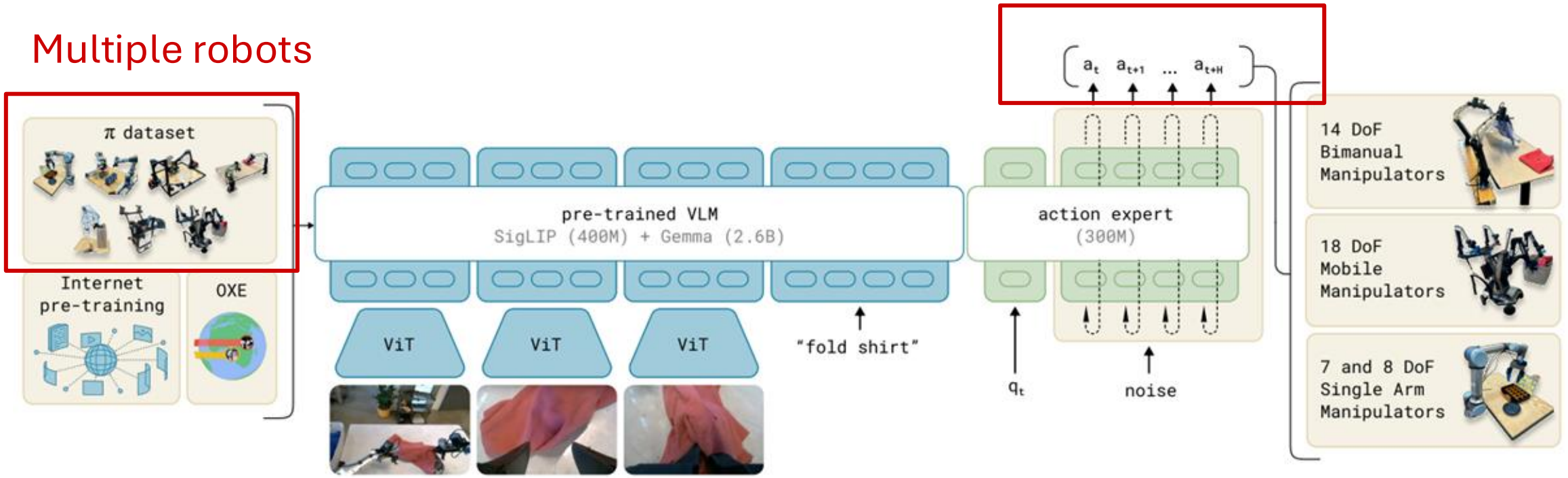

They curated a dataset of 970k robot manipulation trajectories and trained OpenVLA on a cluster of 64 A100 GPUs for 15 days.

OpenVLA: An Open-Source Vision-Language-Action Model, Kim et al., Arxiv 2024

# Vision-Language-Action models (VLA)

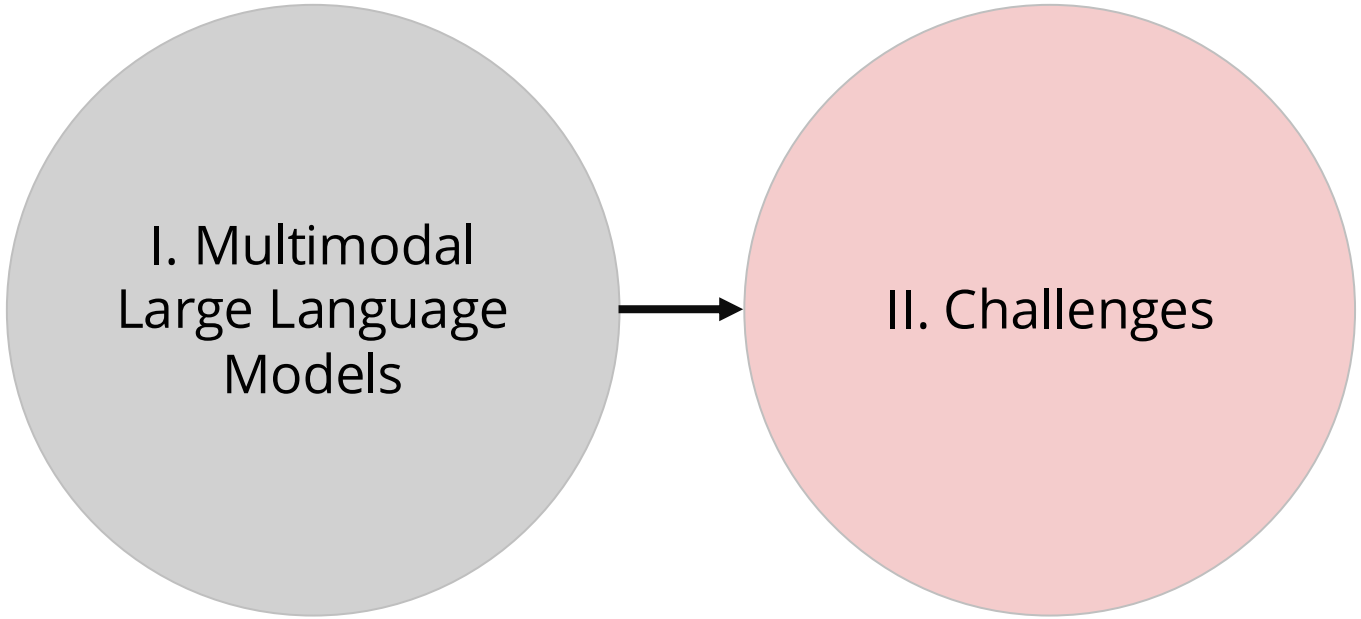
Continuous actions via diffusion

Multiple robots

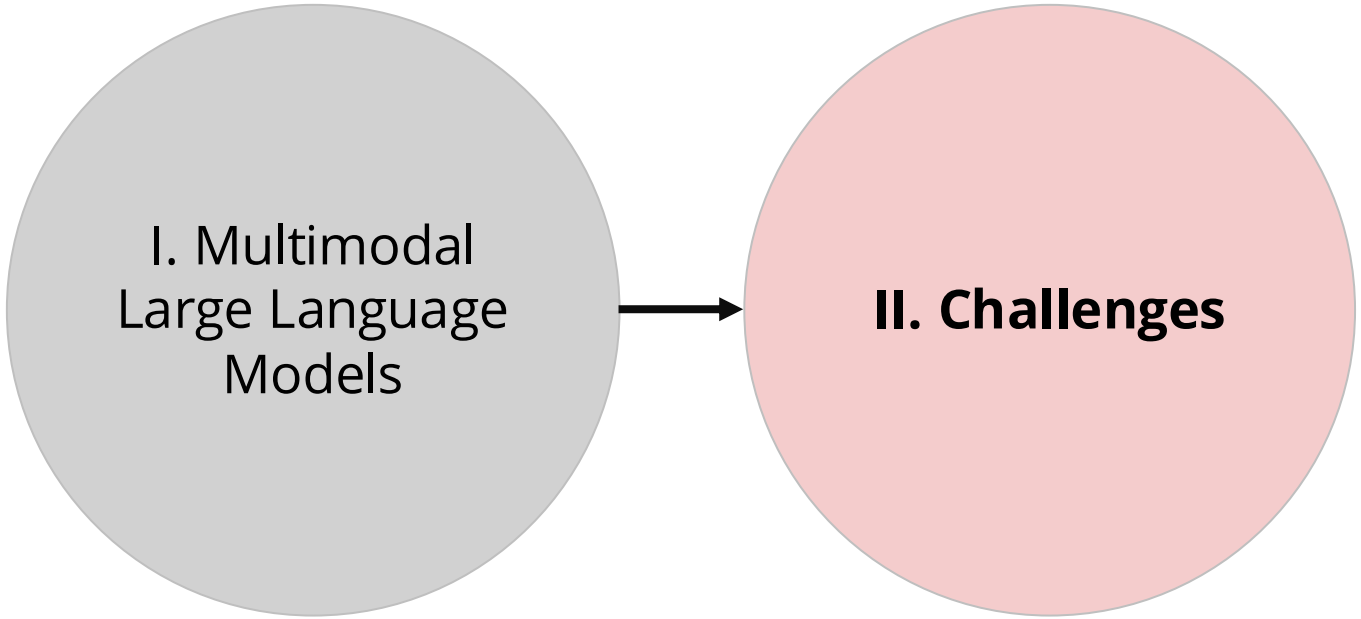



π0: A Vision-Language-Action Flow Model for General Robot Control, Black et al., Arxiv 2025

# Generative AI



# Generative AI



# II. Challenges

1. **Definition:** Define usecase

2. **Select:** Foundation Model to use or pretrain

3. **Adapt:** Foundation Model

# II. Challenges

1. **Definition:** Define usecase

2. **Select:** Foundation Model to use or pretrain

3. **Adapt:** Foundation Model

# II. Challenges

1. **Definition:** Define usecase

2. **Select:** Foundation Model to use or pretrain

3. **Adapt:** Foundation Model

# II. Challenges

1. **Definition:** Define usecase

2. **Select:** Foundation Model to use or pretrain

3. **Adapt:** Foundation Model

# II. Challenges

**1. Definition: Define usecase**

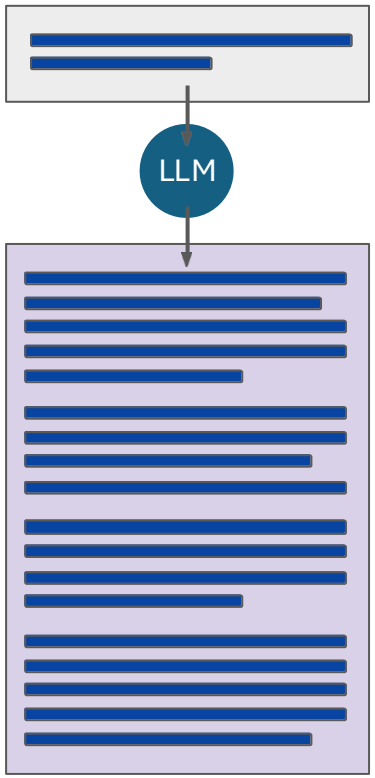
2. **Select:** Foundation Model to use or pretrain

3. **Adapt:** Foundation Model

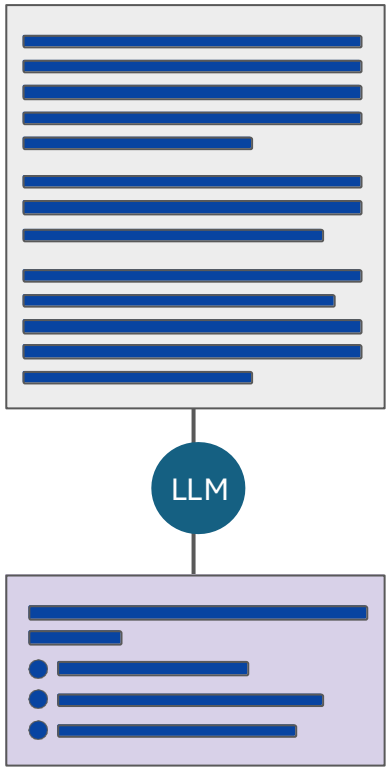
# Scope

# Good at many tasks?

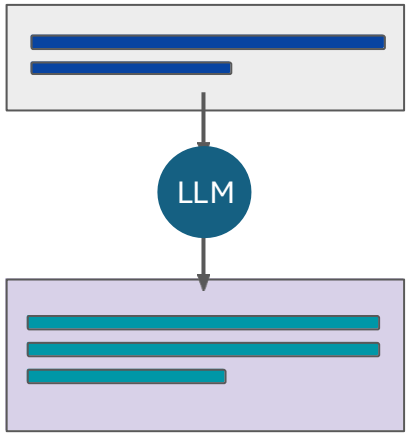
Essay Writing



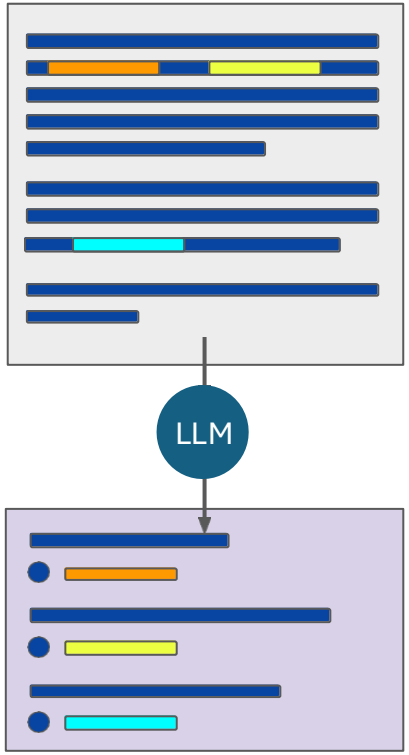

Summarization



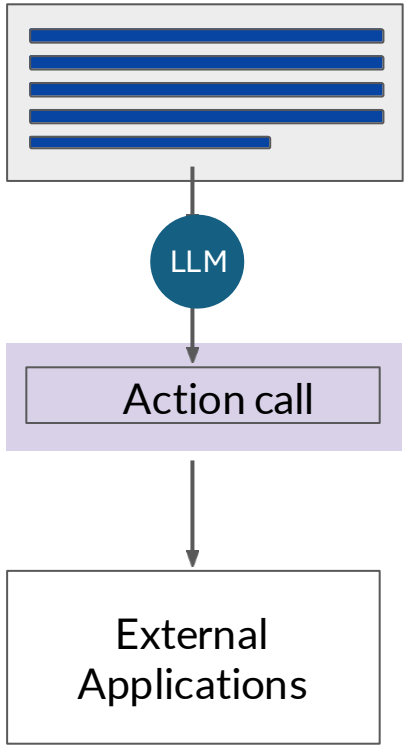

Translation



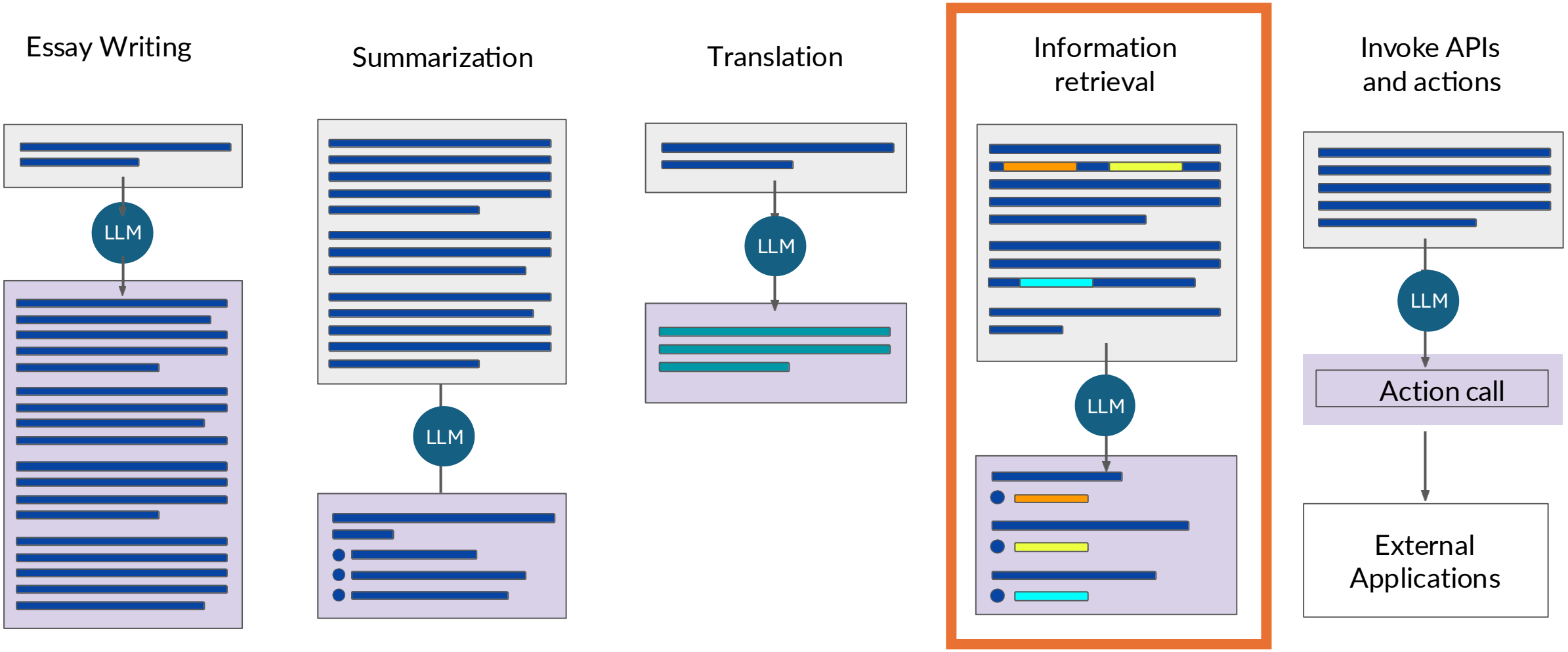

Information retrieval




Invoke APIs and actions



# Or a single task?



# II. Challenges

**1. Definition:** Define usecase

**2. Select: Foundation Model to use or pretrain**

3. **Adapt:** Foundation Model

# II. Challenges

## 2. Select: Foundation Model to use or pretrain

- Choose an existing model or pretrain your own
- Scaling
  - Challenges
  - Cost
  - Scaling laws
- Pre-training for domain adaptation

created with chatGPT

# II. Challenges

## 2. Select: Foundation Model to use or pretrain

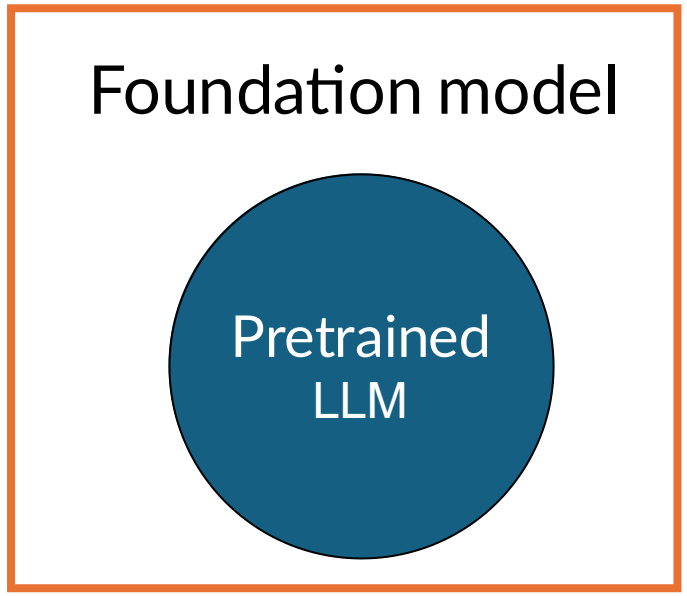
- **Choose an existing model or pretrain your own**

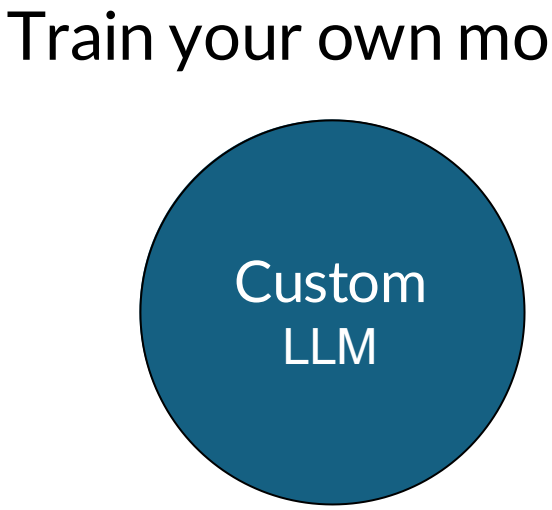
- Scaling
  - Challenges
  - Cost
  - Scaling laws

- Pre-training for domain adaptation

created with chatGPT

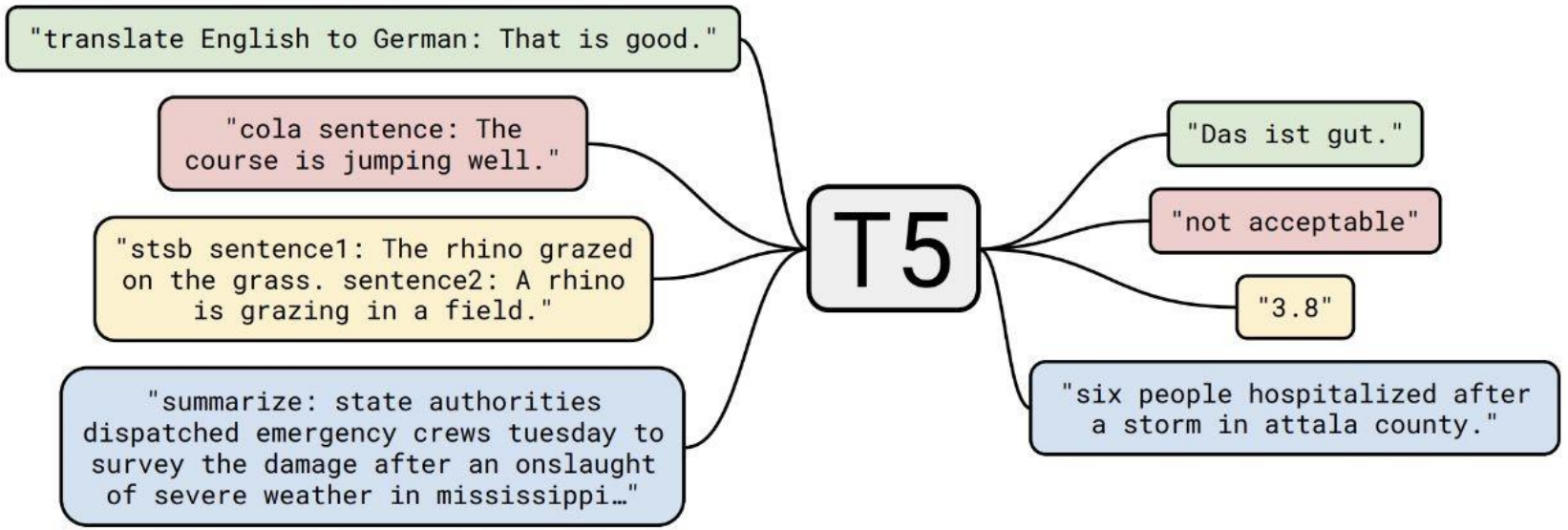
# Considerations for choosing a model



# Model hubs

## Model Card for T5 Large




## Table of Contents

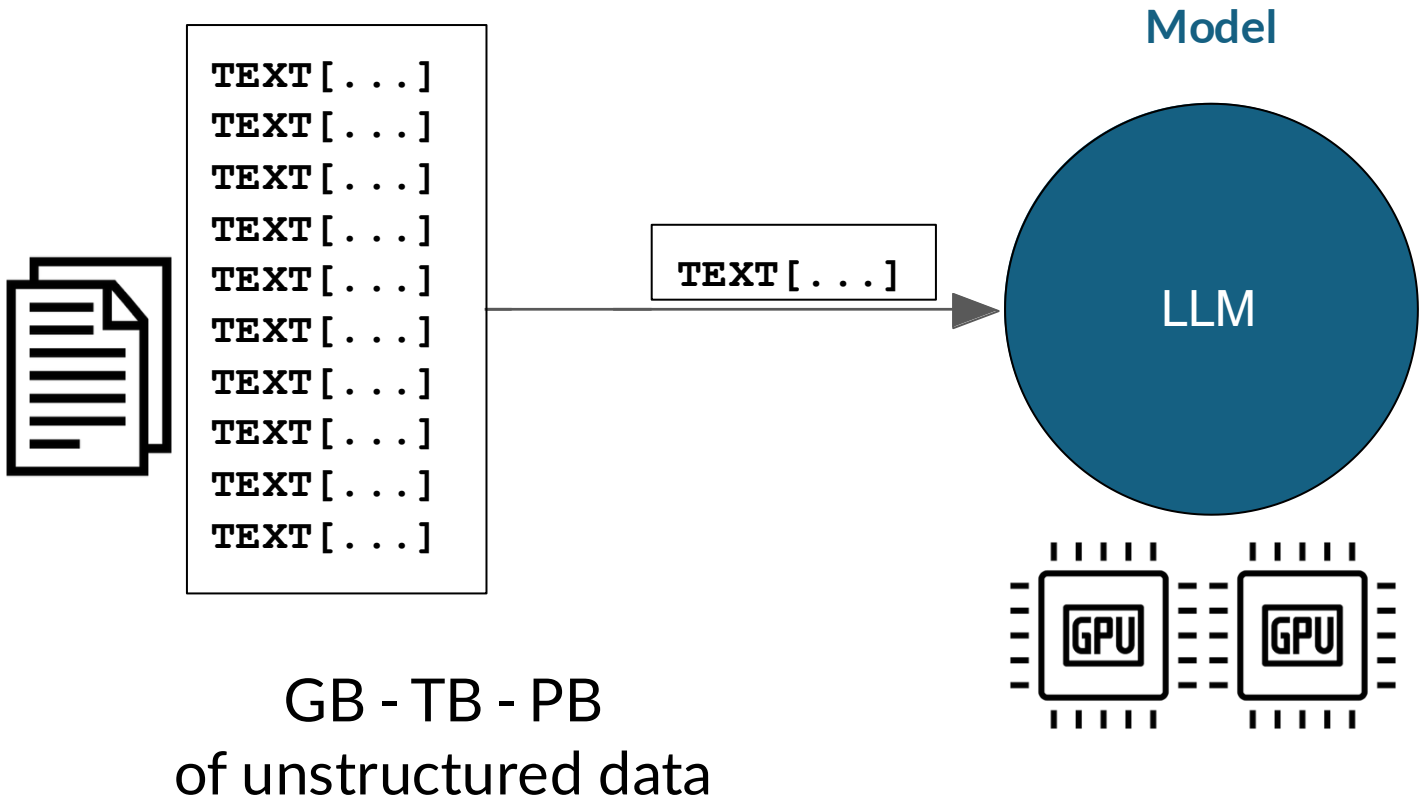
# Considerations for choosing a model

Foundation model


Train your own model

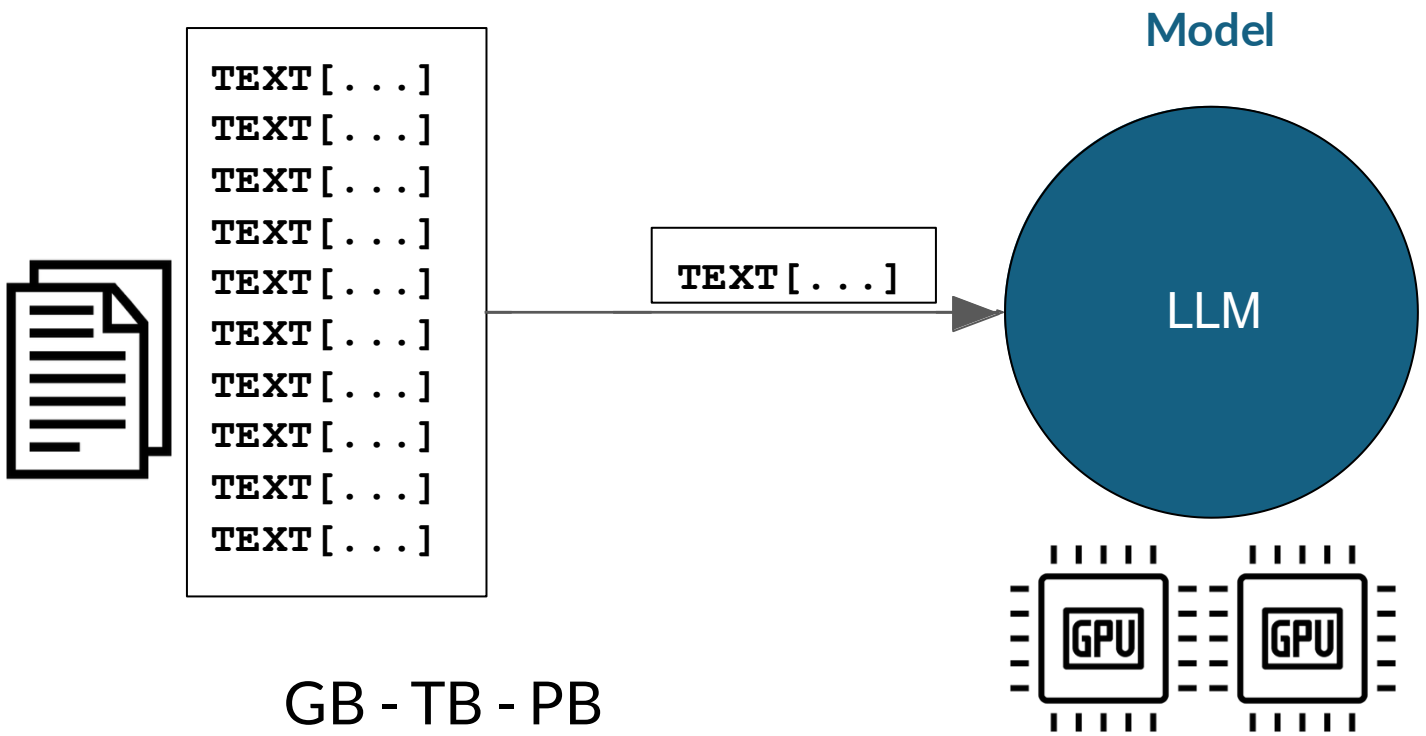
# Pre-training objectives

# LLM pre-training at a high level




GB - TB - PB
of unstructured data

# LLM pre-training at a high level

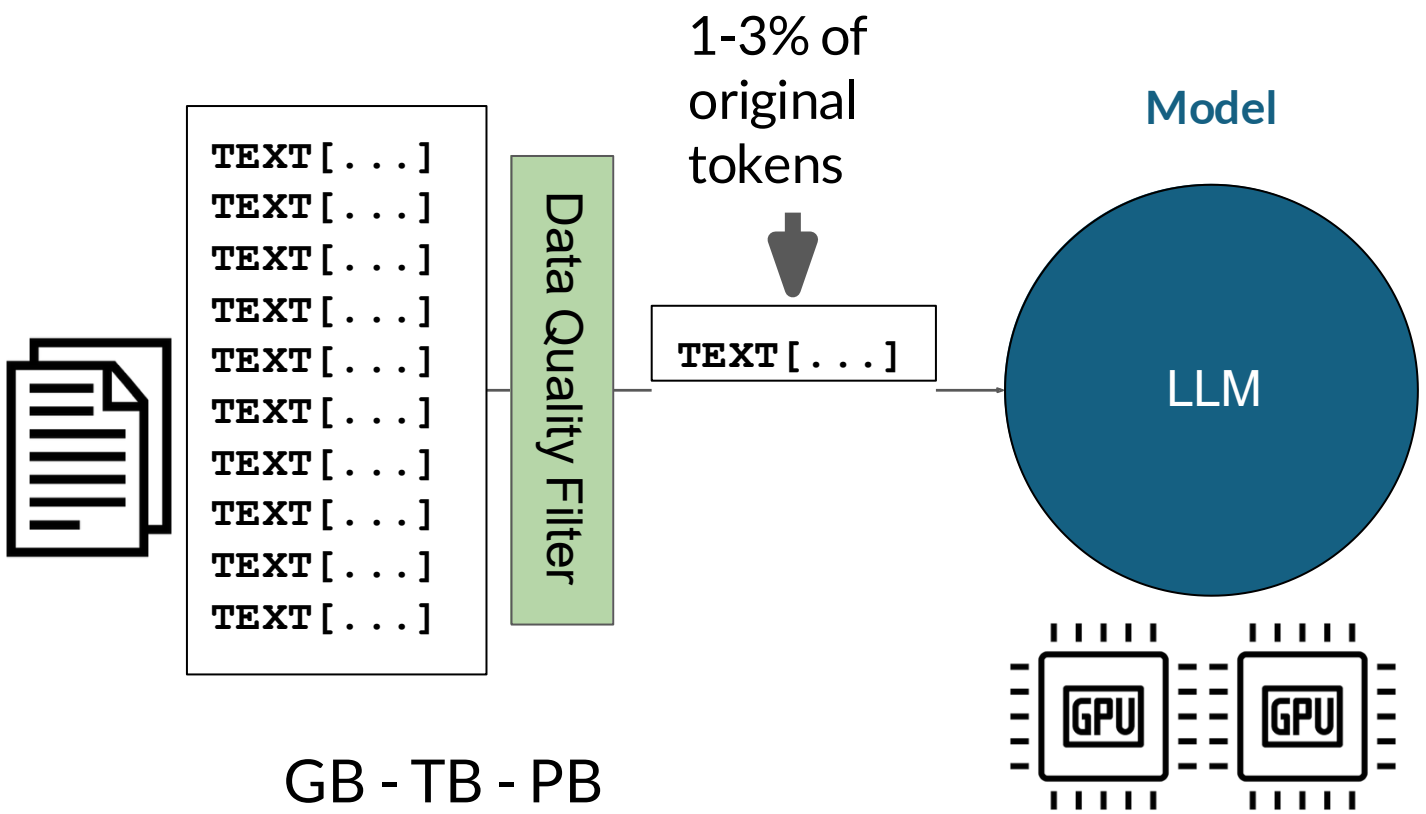



GB - TB - PB
of unstructured data

| Token String | Token ID | Embedding / Vector Representation |
|---|---|---|
| '_The' | 37 | [-0.0513, -0.0584, 0.0230, ...] |
| '_teacher' | 3145 | [-0.0335, 0.0167, 0.0484, ...] |
| '_teaches' | 11749 | [-0.0151, -0.0516, 0.0309, ...] |
| '_the' | 8 | [-0.0498, -0.0428, 0.0275, ...] |
| '_student' | 1236 | [-0.0460, 0.0031, 0.0545, ...] |
| ... | ... | ... |

Vocabulary

# LLM pre-training at a high level

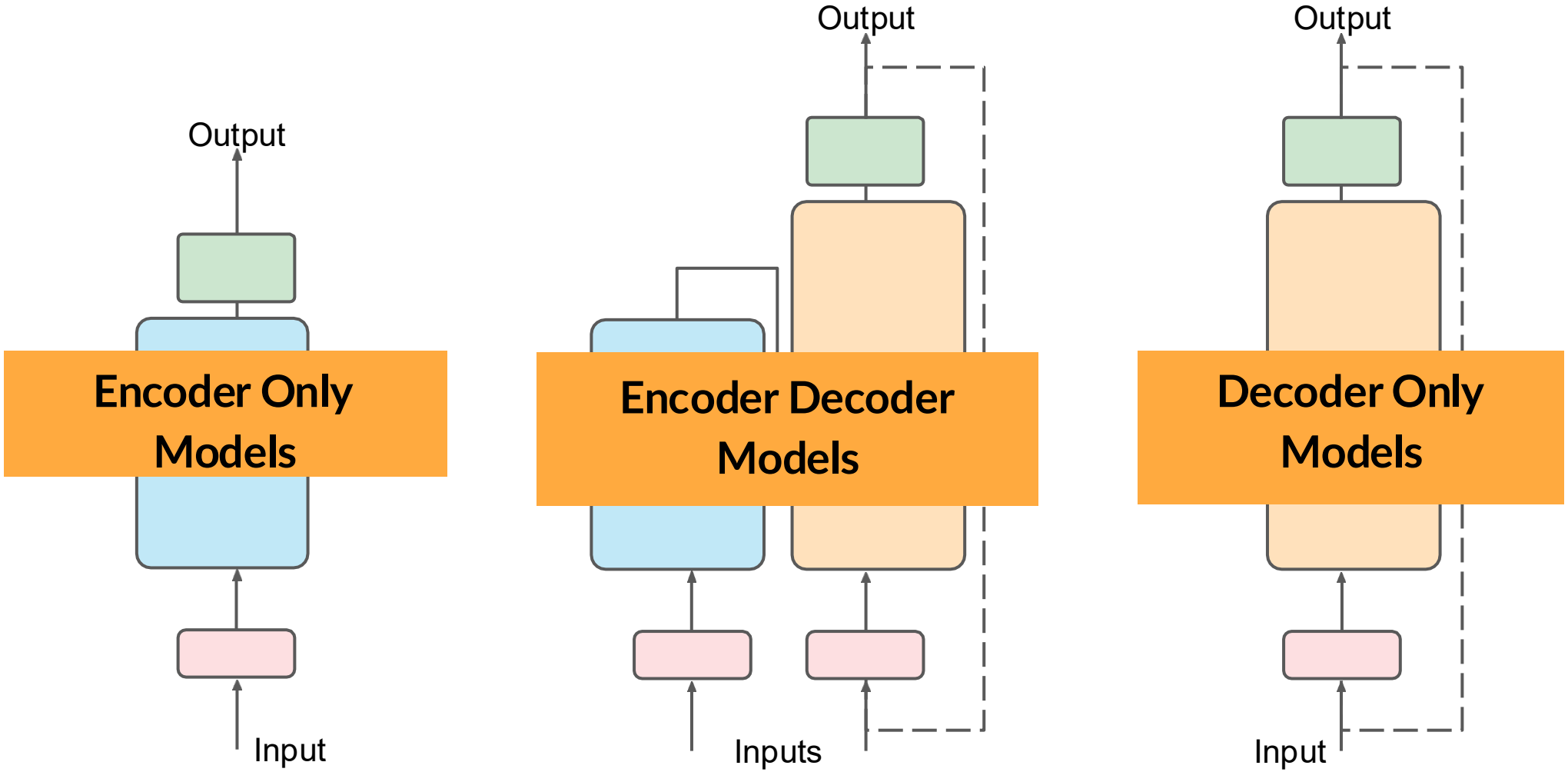



GB - TB - PB
of unstructured data

| Token String | Token ID | Embedding / Vector Representation |
|---|---|---|
| '_The' | 37 | [-0.0513, -0.0584, 0.0230, ...] |
| '_teacher' | 3145 | [-0.0335, 0.0167, 0.0484, ...] |
| '_teaches' | 11749 | [-0.0151, -0.0516, 0.0309, ...] |
| '_the' | 8 | [-0.0498, -0.0428, 0.0275, ...] |
| '_student' | 1236 | [-0.0460, 0.0031, 0.0545, ...] |
| ... | ... | ... |

Vocabulary

# Transformers



# Autoencoding models

Original text

**The teacher teaches the student.**

[...]

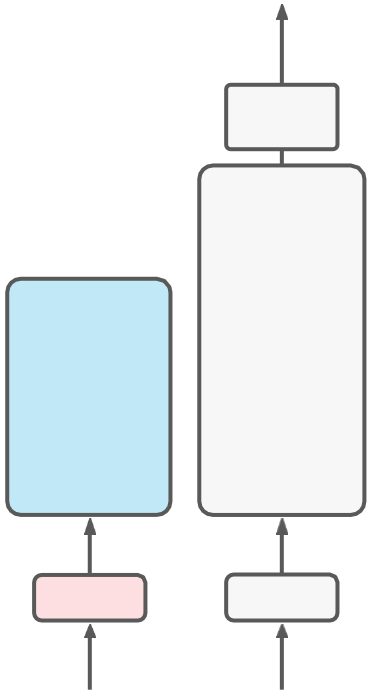
Masked Language Modeling (MLM)

| The | teacher | **<MASK>** | the | student |
|---|---|---|---|---|

Encoder-only LLM

Objective: Reconstruct text ("denoising")

| The | teacher | **teaches** | the | student |
|---|---|---|---|---|



Bidirectional context

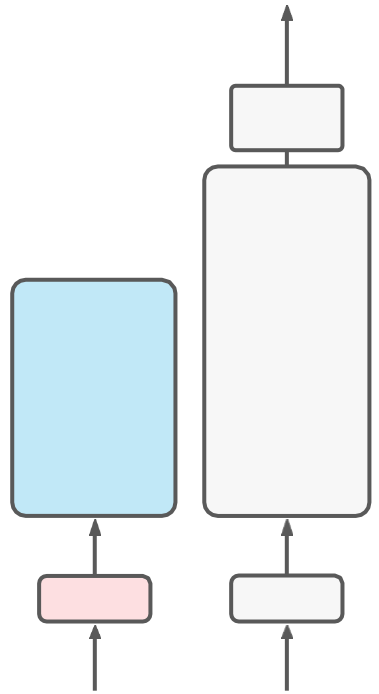
# Autoencoding models

Good use cases:

- Sentiment analysis
- Named entity recognition
- Word classification

Example models:

- BERT
- ROBERTA



# Autoregressive models

Original text

**The teacher teaches the student.**

[...]

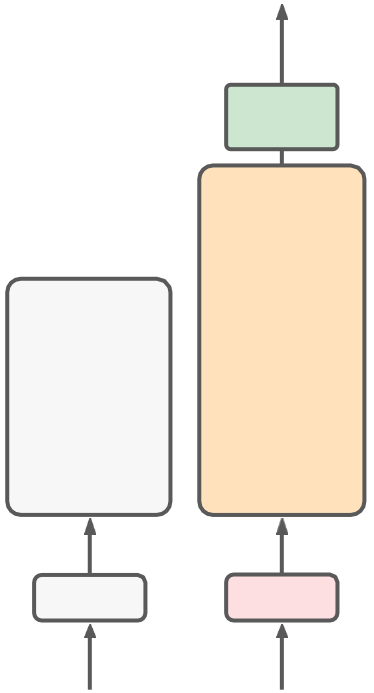
Causal Language Modeling (CLM)

| The | teacher | ? |
|---|---|---|

Decoder-only LLM

Objective: Predict next token

| The | **teacher** | **teaches** |
|---|---|---|



Unidirectional context

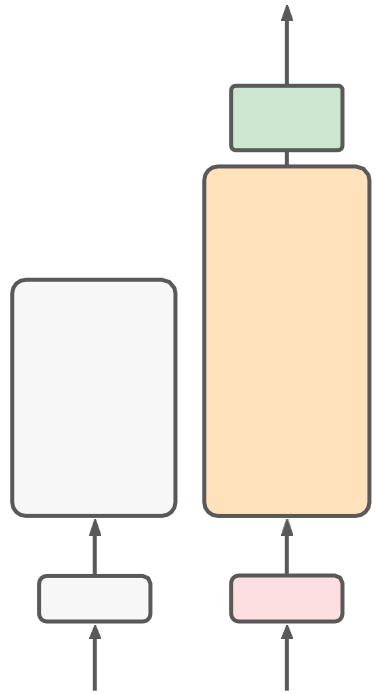
# Autoregressive models

Good use cases:

- Text generation
- Other emergent behavior
  - Depends on model size

Example models:

- GPT
- BLOOM



# Sequence-to-sequence models

Span Corruption

| | | | | |
|---|---|---|---|---|
| The | teacher | **<MASK>** | **<MASK>** | student |
| The | teacher | **<X>** | | student |

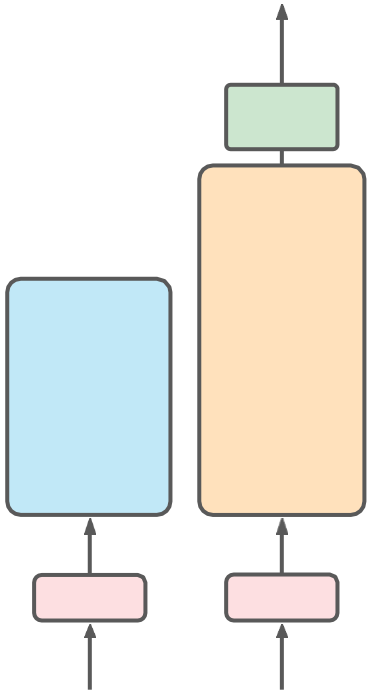
Original text

**The teacher teaches the student.**

[...]

Encoder-Decoder LLM

Sentinel token



Objective: Reconstruct span

| | | |
|---|---|---|
| **<x>** | **teaches** | **the** |

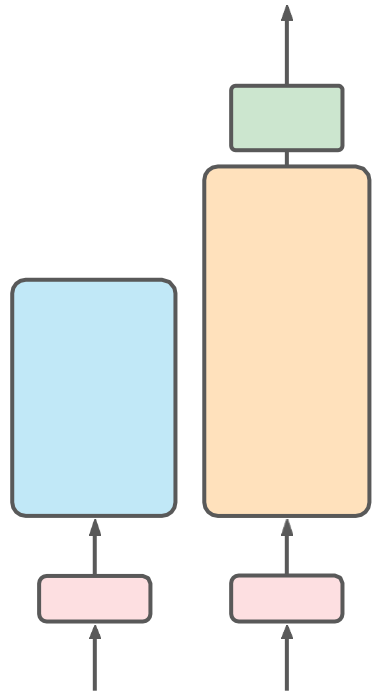
# Sequence-to-sequence models

Good use cases:

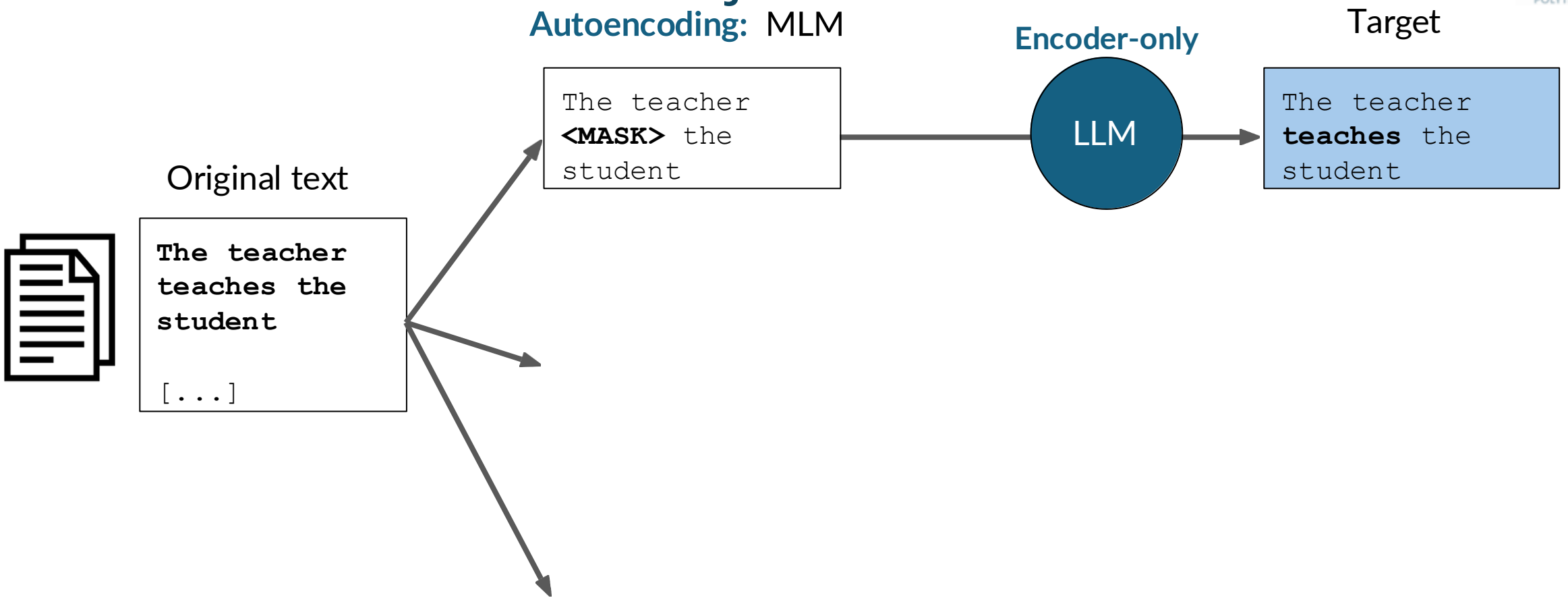
- Translation
- Text summarization
- Question answering

Example models:

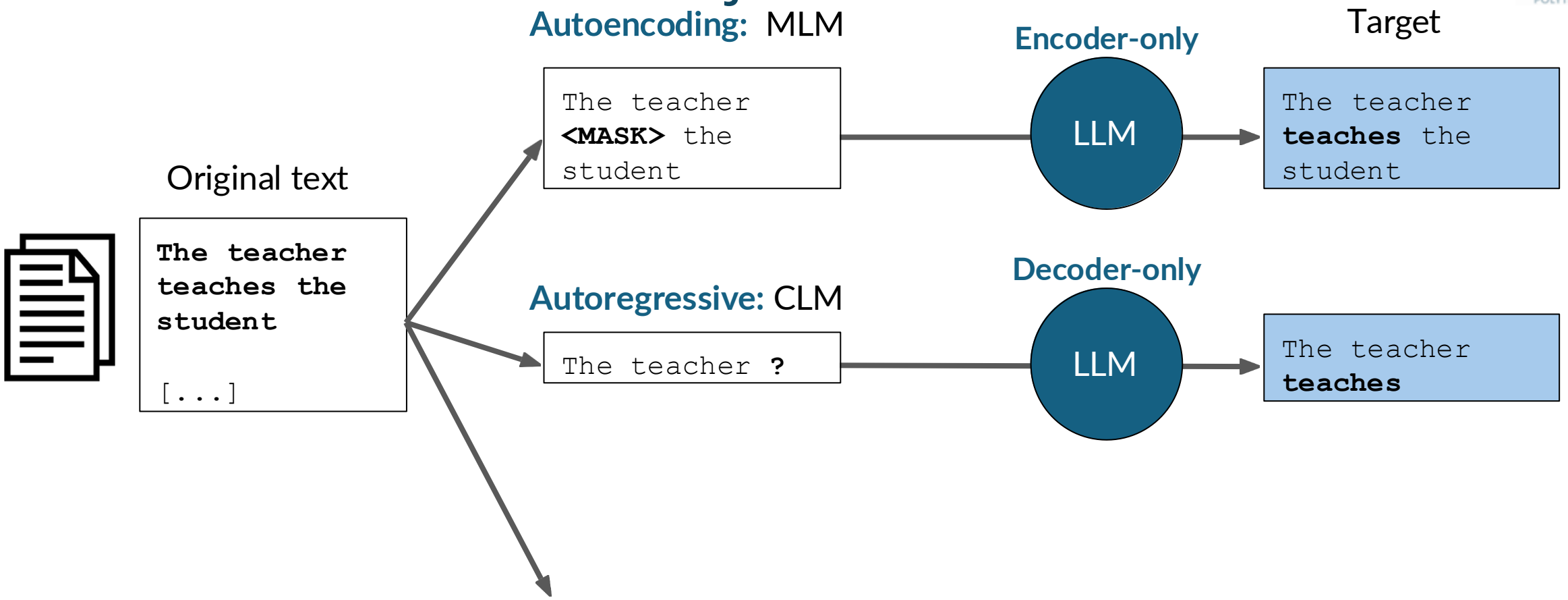
- T5
- BART



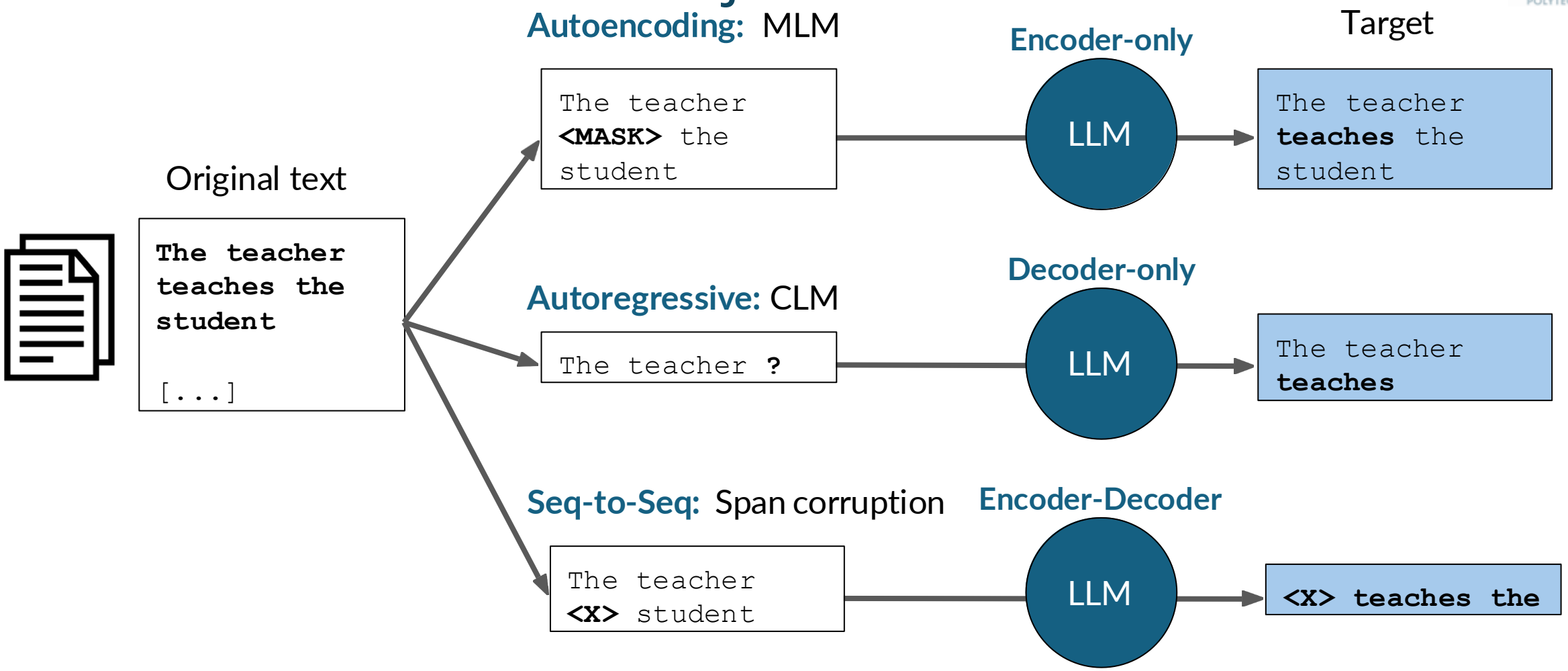
# Summary: Model architectures and pre-training objectives



# Summary: Model architectures and pre-training objectives



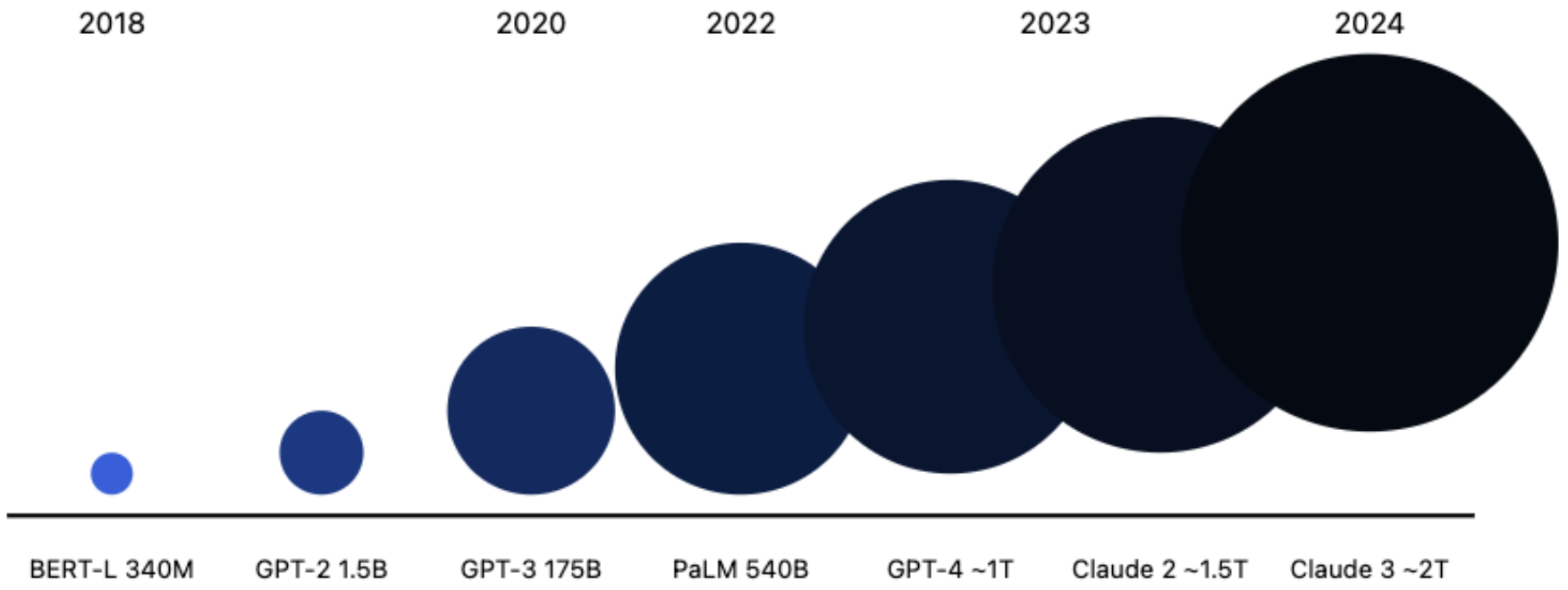
# Summary: Model architectures and pre-training objectives



# Model size vs. time

Growth powered by:

- Introduction of transformer
- Access to massive datasets
- More powerful compute resources



# II. Challenges

## 2. Select: Foundation Model to use or pretrain

- Choose an existing model or pretrain your own

- **Scaling**
  - Challenges
  - Cost
  - Scaling laws

- Pre-training for domain adaptation

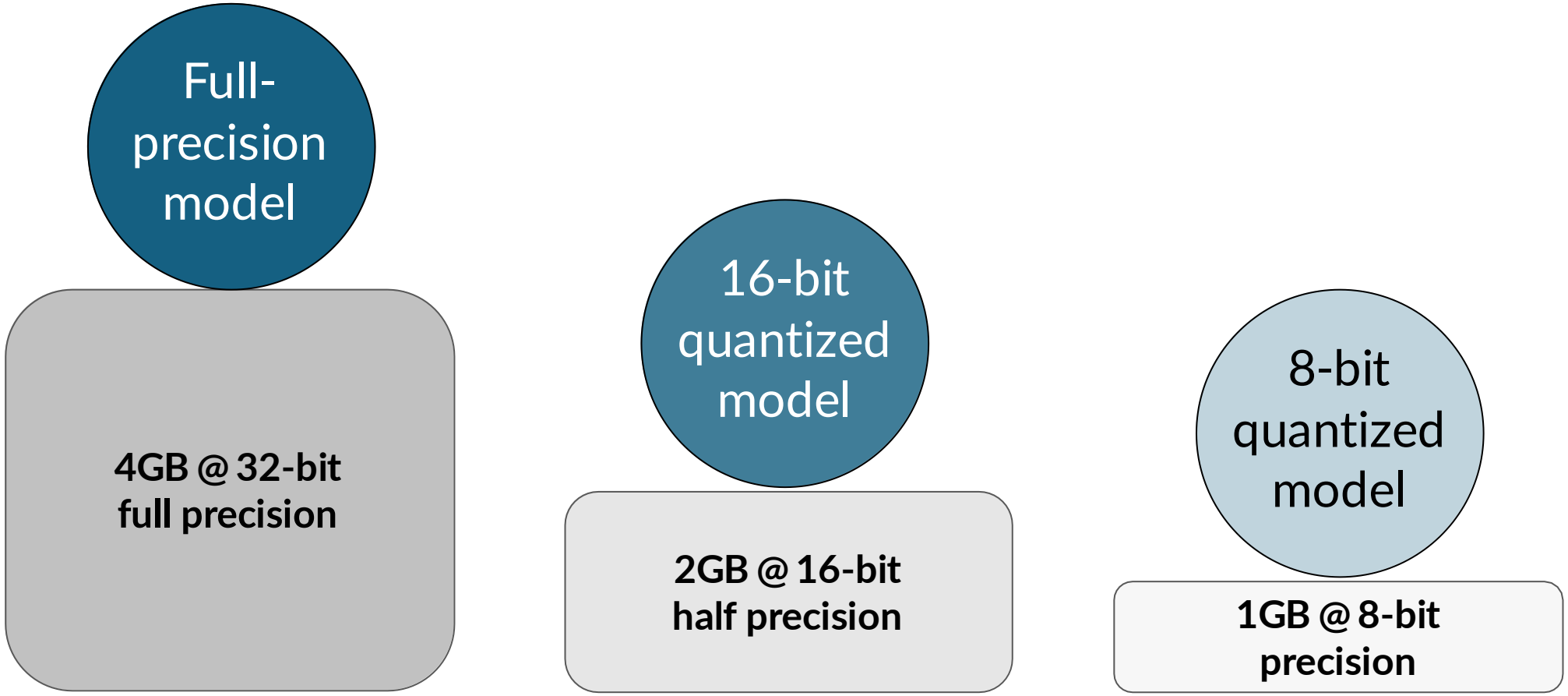
created with chatGPT

# Challenges

# Compute…

```
OutOfMemoryError: CUDA out of memory.
```

# Approximate GPU RAM needed to store 1B parameters




Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, https://github.com/facebookresearch/bitsandbytes

# GPU RAM needed to train larger models

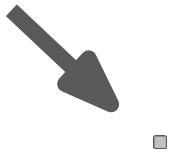
**1B param model**

**175B param model**

**4,200 GB @ 32-bit full precision**

**500B param model**

**12,000 GB @ 32-bit full precision**



# GPU RAM needed to train larger models

**As model sizes get larger, you will need to split your model across multiple GPUs for training**

**1B param model**

**4,200 GB @ 32-bit full precision**

**175B param model**

**500B param model**

**12,000 GB @ 32-bit full precision**

# Scaling-up Transformers

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |
| XLNet Large | 24 | 1024 | 16 | ~340M | 126GB | 512xTPUv3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160GB | 1024xV100 GPU (1 day) |

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |
| XLNet Large | 24 | 1024 | 16 | ~340M | 126GB | 512xTPUv3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160GB | 1024xV100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40GB | |

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |
| XLNet Large | 24 | 1024 | 16 | ~340M | 126GB | 512xTPUv3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160GB | 1024xV100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40GB | |
| Megatron-LM | 72 | 2072 | 32 | 8.3B | 174GB | 512xV100 GPU (9 days) |
| Turing NLG | 78 | 4256 | 28 | 17B | ? | 256xV100 GPU |

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |
| XLNet Large | 24 | 1024 | 16 | ~340M | 126GB | 512xTPUv3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160GB | 1024xV100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40GB | |
| Megatron-LM | 72 | 2072 | 32 | 8.3B | 174GB | 512xV100 GPU (9 days) |
| Turing NLG | 78 | 4256 | 28 | 17B | ? | 256xV100 GPU |

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |
| XLNet Large | 24 | 1024 | 16 | ~340M | 126GB | 512xTPUv3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160GB | 1024xV100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40GB | |
| Megatron-LM | 72 | 2072 | 32 | 8.3B | 174GB | 512xV100 GPU (9 days) |
| Turing NLG | | | 28 | 17B | ? | 256xV100 GPU |

~350k euros!

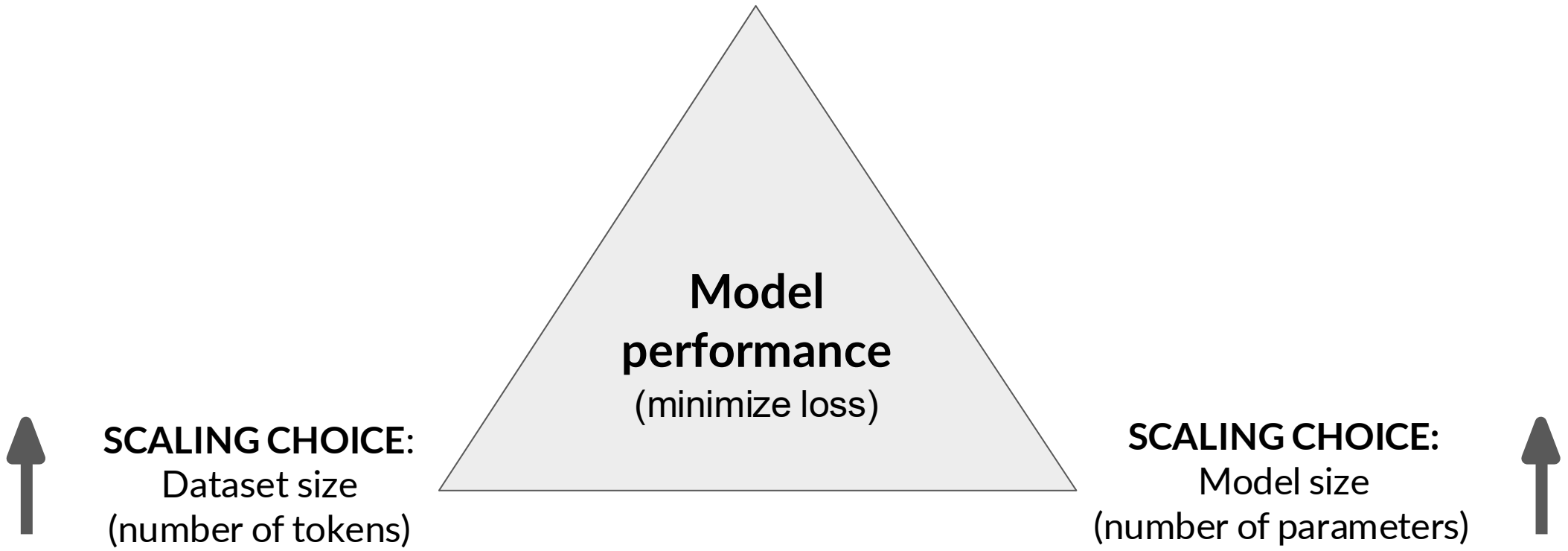
# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |
| XLNet Large | 24 | 1024 | 16 | ~340M | 126GB | 512xTPUv3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160GB | 1024xV100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40GB | |
| Megatron-LM | 72 | 2072 | 32 | 8.3B | 174GB | 512xV100 GPU (9 days) |
| Turing NLG | 78 | 4256 | 28 | 17B | ? | 256xV100 GPU |
| GPT-3 | 96 | 12288 | 96 | 175B | 694GB | |

# Scaling laws

# Scaling choices for pre-training

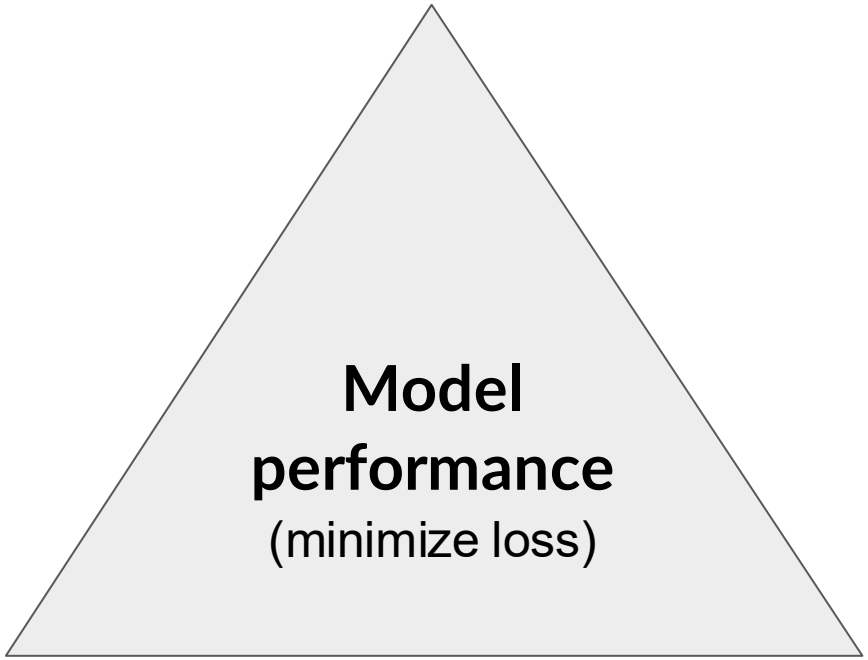
Goal: **maximize model performance**



# Scaling choices for pre-training

Goal: **maximize model performance**

**CONSTRAINT:**
Compute budget
(GPUs, training time, cost)



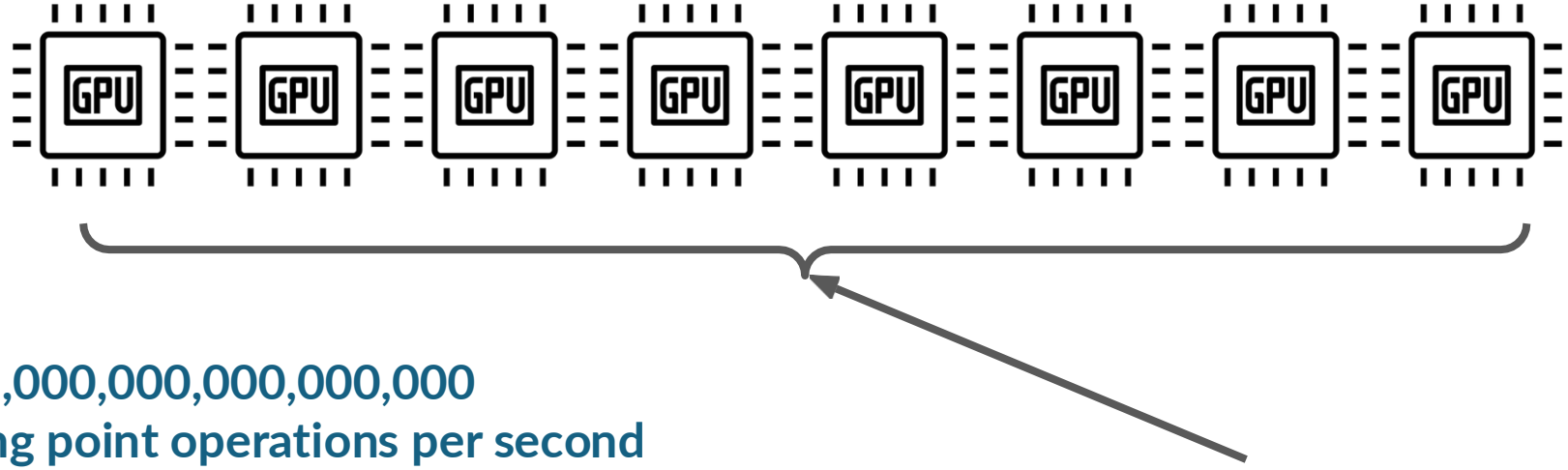

**SCALING CHOICE:**
Dataset size
(number of tokens)

**SCALING CHOICE:**
Model size
(number of parameters)

# Compute budget for training LLMs

1 “petaflop/s-day” =
# floating point operations performed at rate of 1 petaFLOP per second for one day

NVIDIA V100s




**Note: 1 petaFLOP/s = 1,000,000,000,000,000 (one quadrillion) floating point operations per second**

1 petaflop/s-day is these chips running at full efficiency for 24 hours

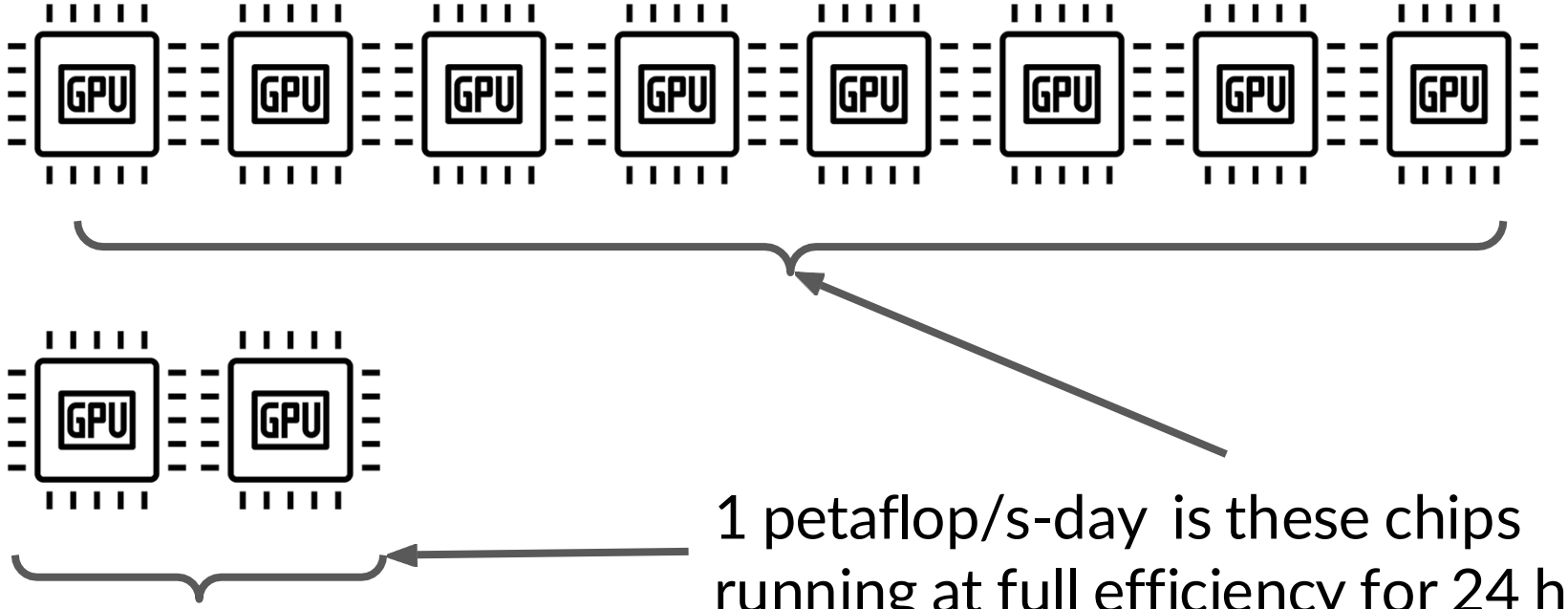
# Compute budget for training LLMs

1 “petaflop/s-day” =
# floating point operations performed at rate of 1 petaFLOP per second for one day

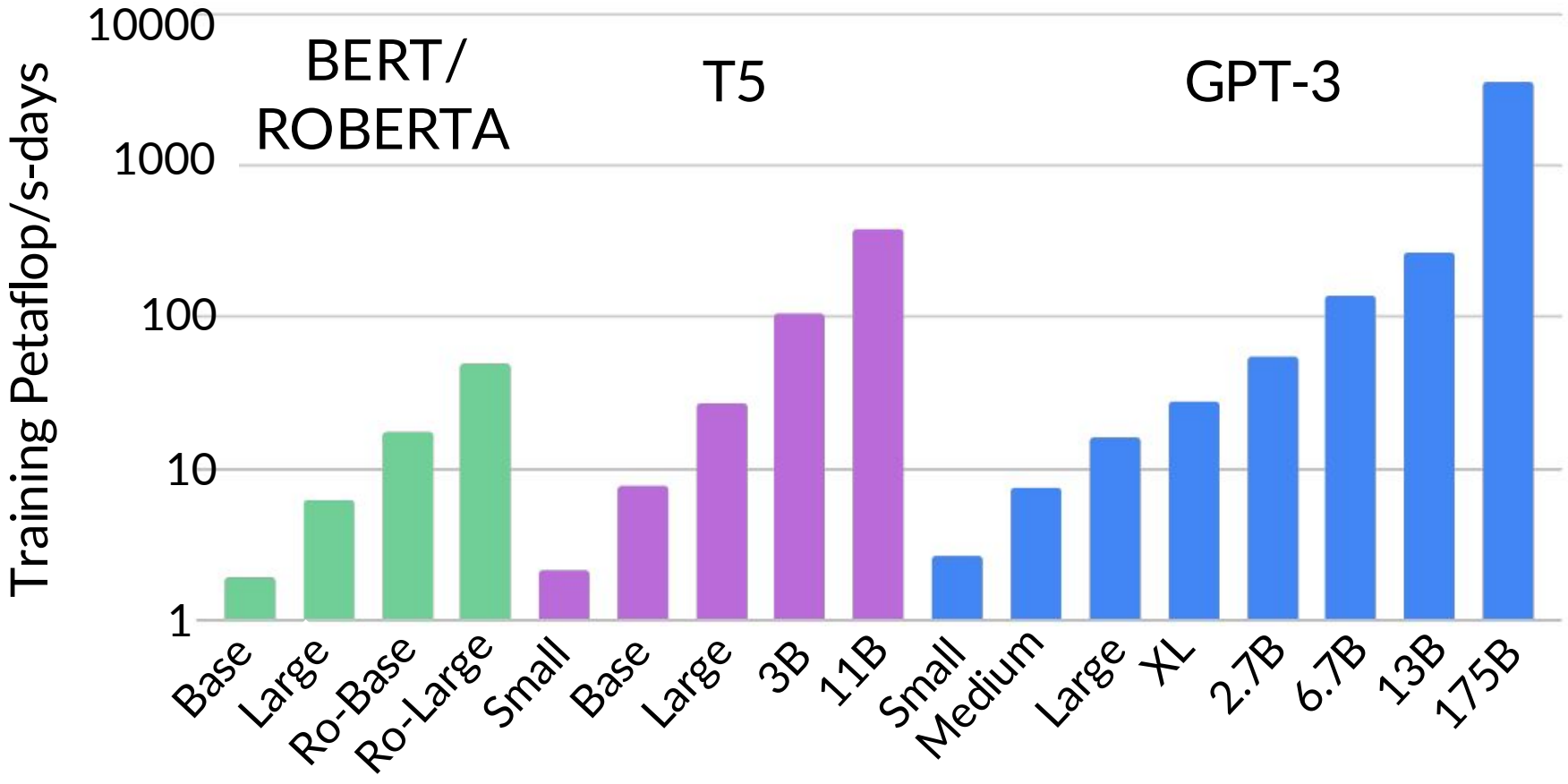
NVIDIA V100s

OR

NVIDIA A100s

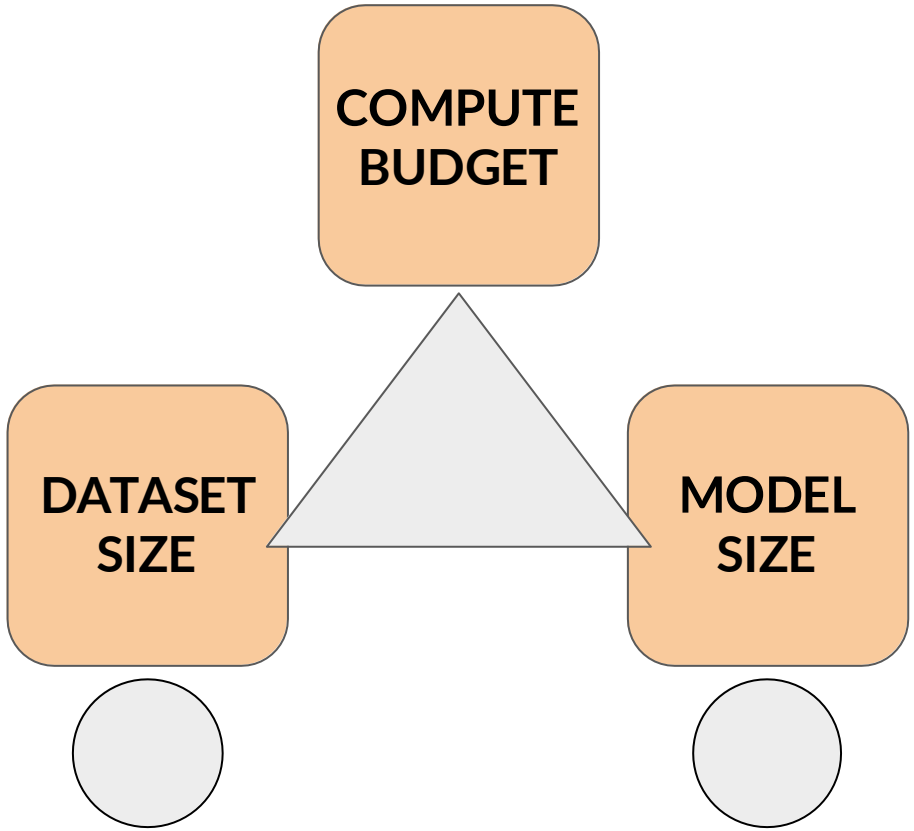


# Number of petaflop/s-days to pre-train various LLMs



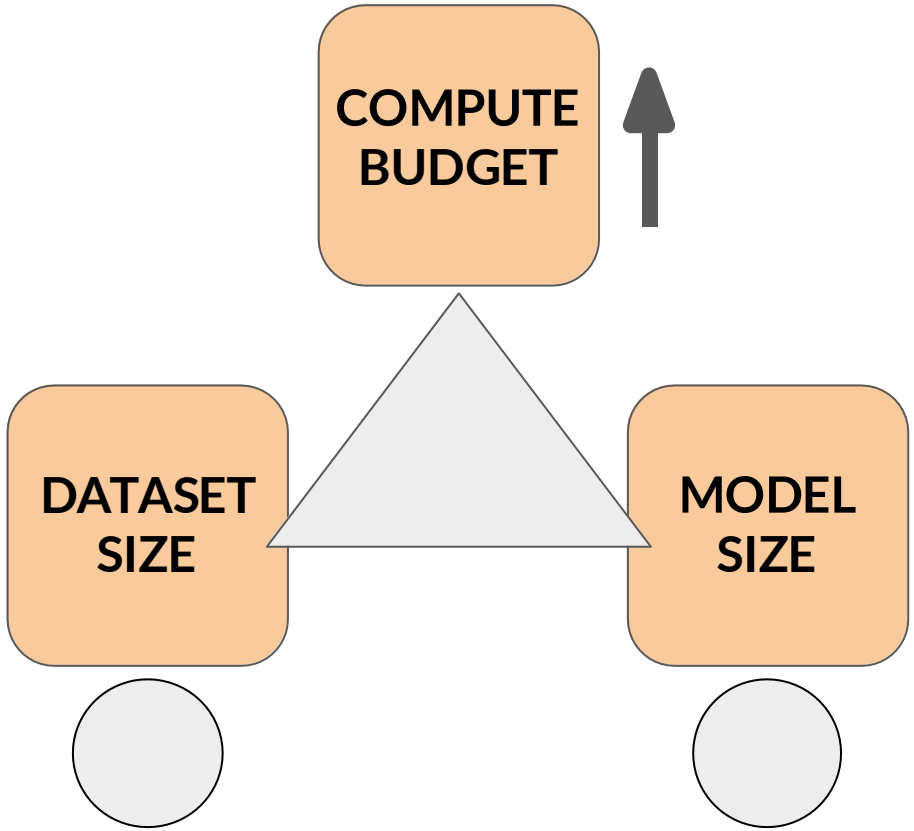

Source: Brown et al. 2020, “Language Models are Few-Shot Learners”

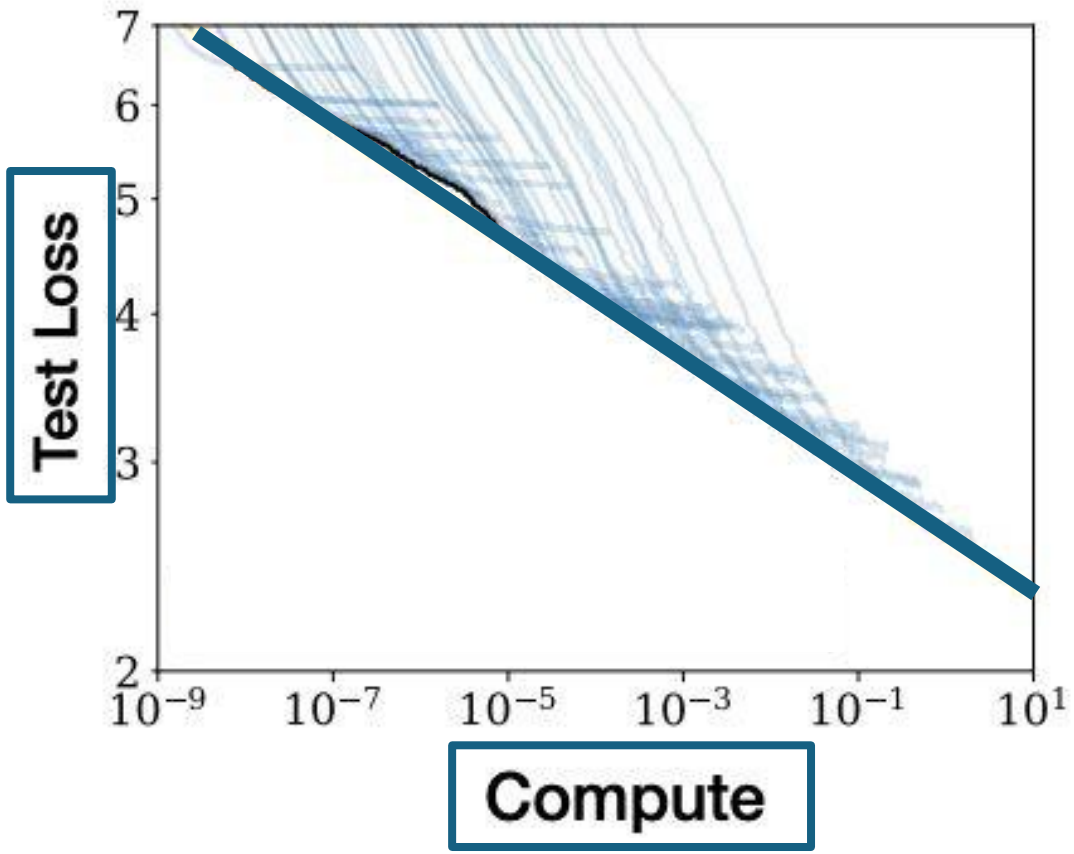
# Compute budget vs. model performance




Source: Kaplan et al. 2020, “Scaling Laws for Neural Language Models”

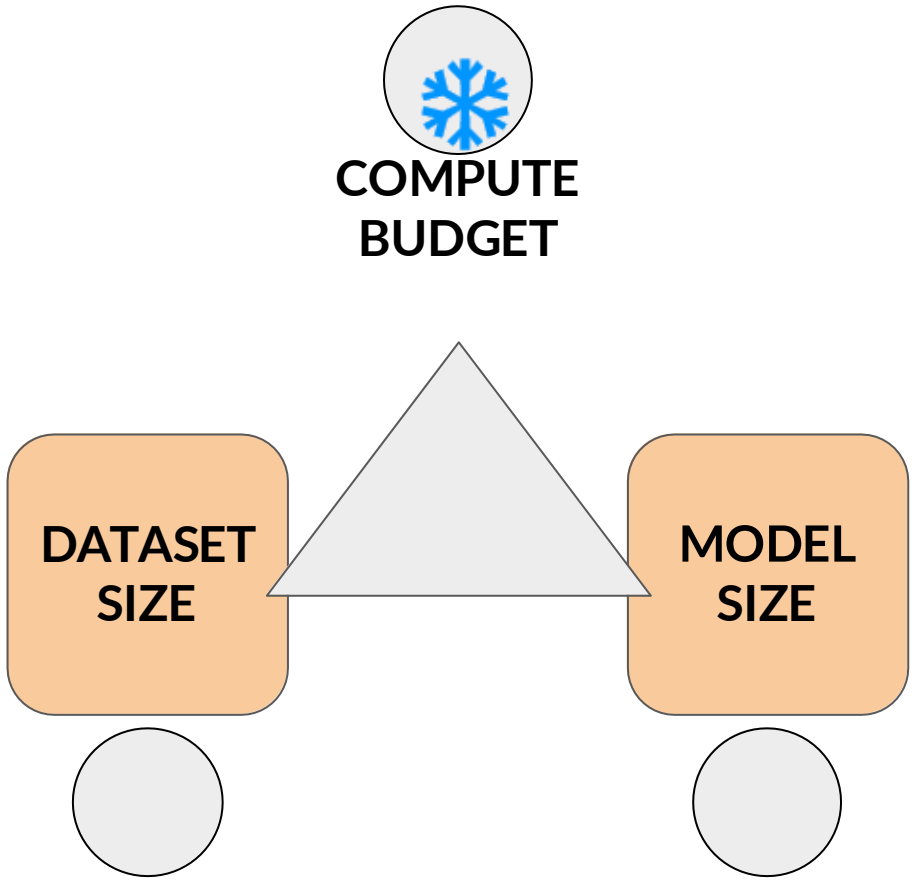
# Compute budget vs. model performance




Source: Kaplan et al. 2020, “Scaling Laws for Neural Language Models”

# Dataset size and model size vs. performance ❄

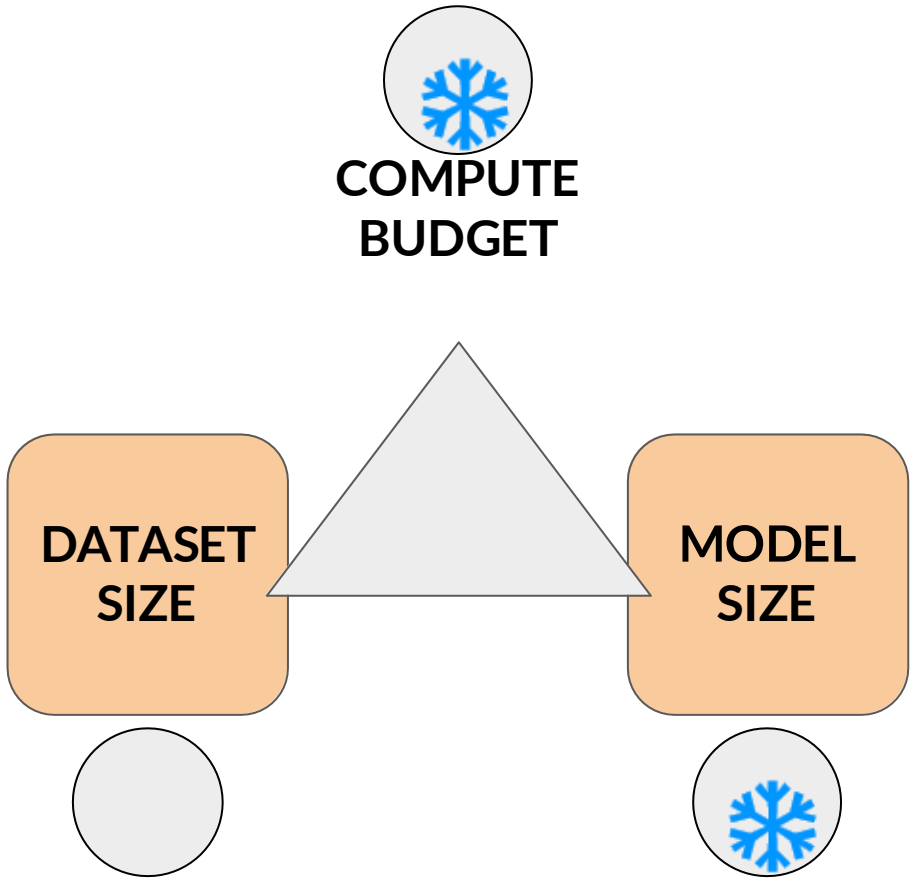


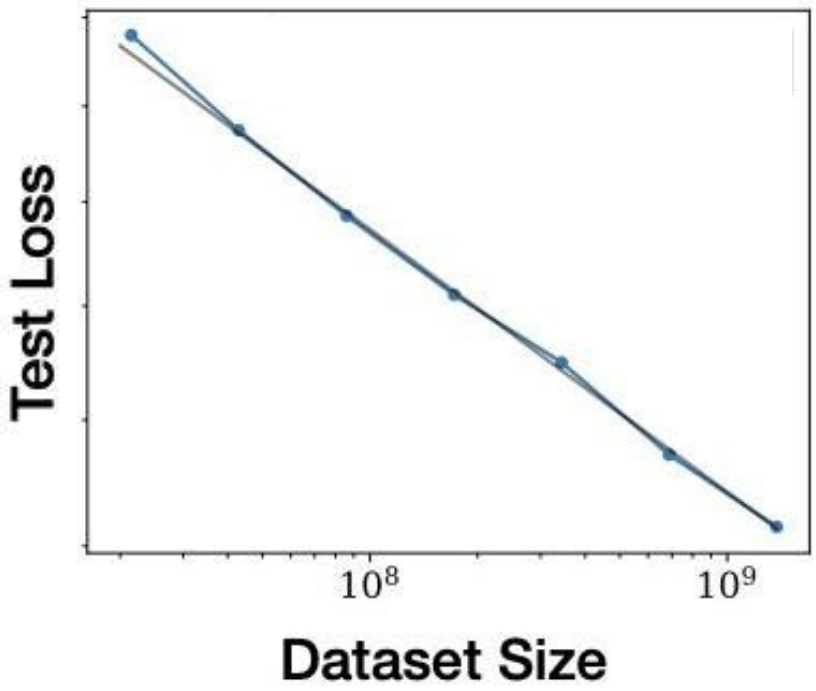

Compute resource constraints

- Hardware
- Project timeline
- Financial budget

Source: Kaplan et al. 2020, “Scaling Laws for Neural Language Models”

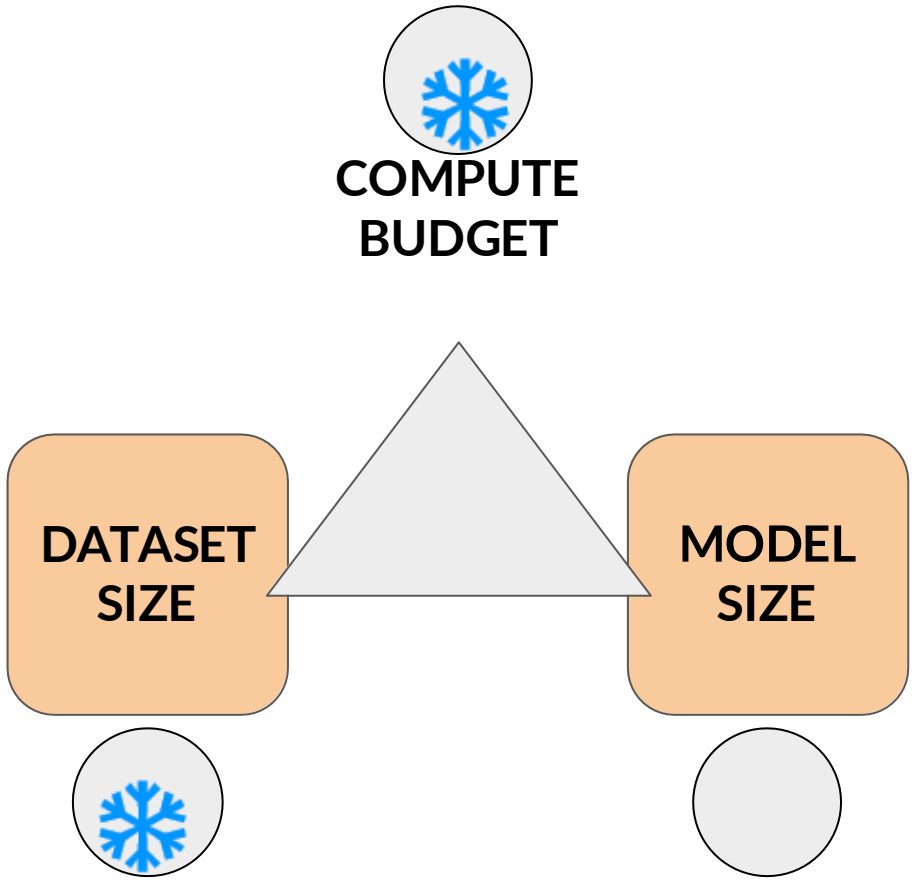
# Dataset size and model size vs. performance ❄

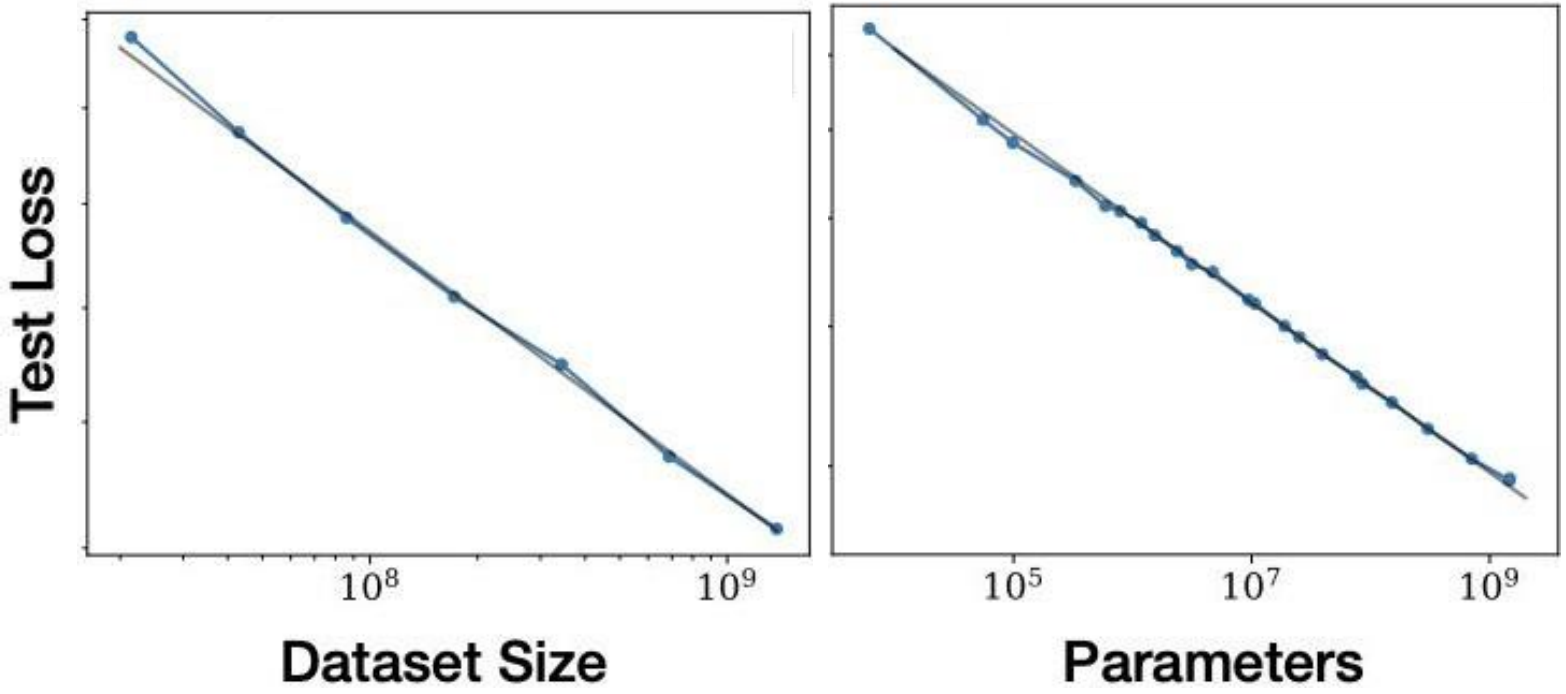



Source: Kaplan et al. 2020, “Scaling Laws for Neural Language Models”

# Dataset size and model size vs. performance ❄




Source: Kaplan et al. 2020, “Scaling Laws for Neural Language Models”

# Chinchilla paper

## Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

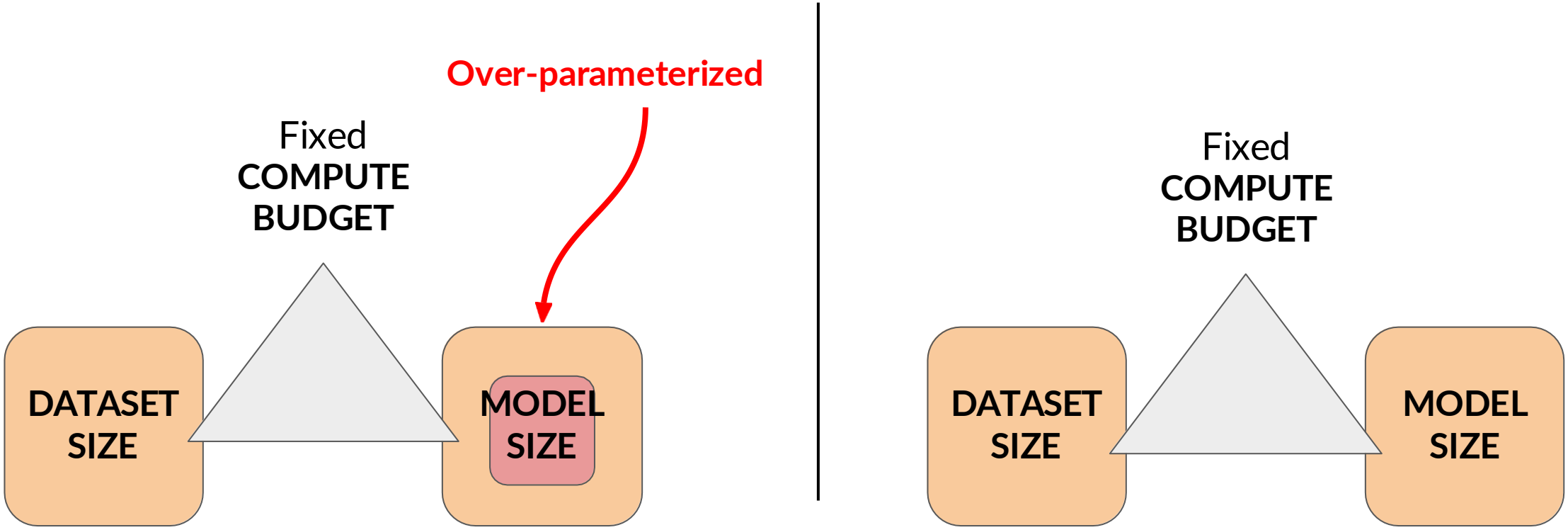
*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Jordan et al. 2022

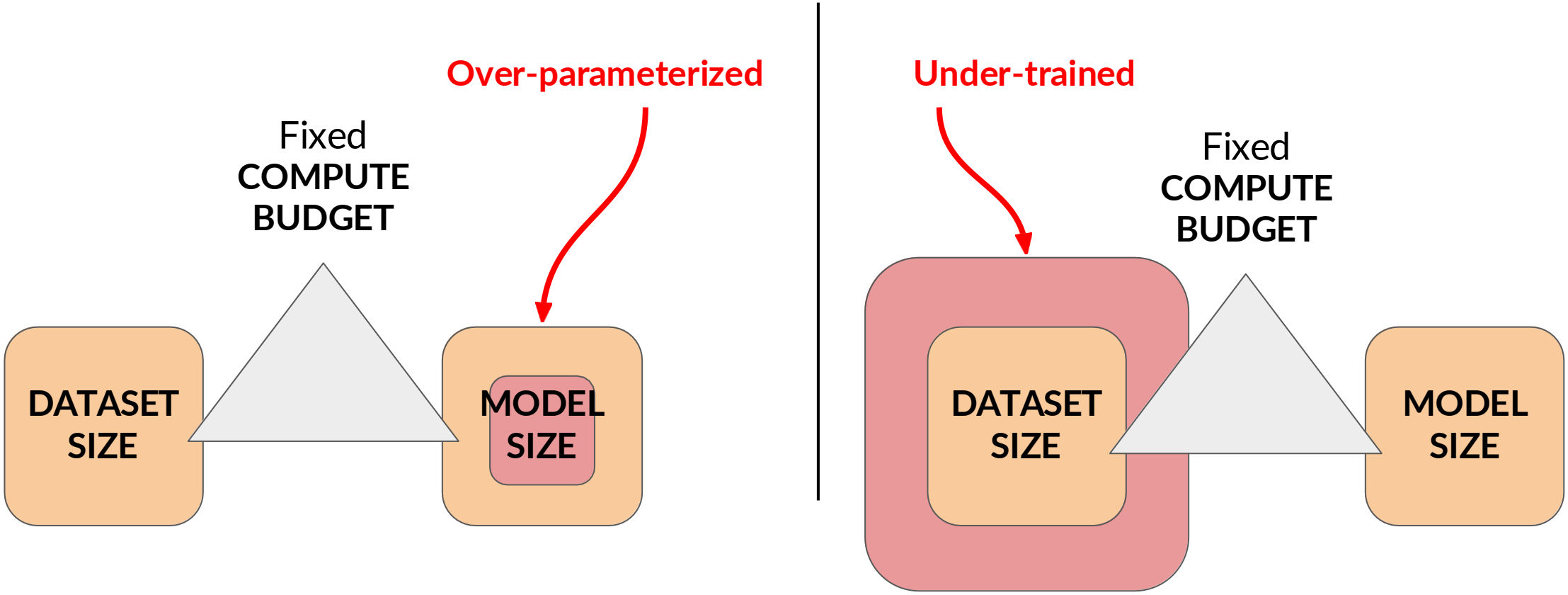
# Compute optimal models

- Very large models may be **over-parameterized**



# Compute optimal models

- Very large models may be **over-parameterized** and **under-trained**
- Smaller models trained on more data could perform as well as large models



# Chinchilla scaling laws for model + dataset size

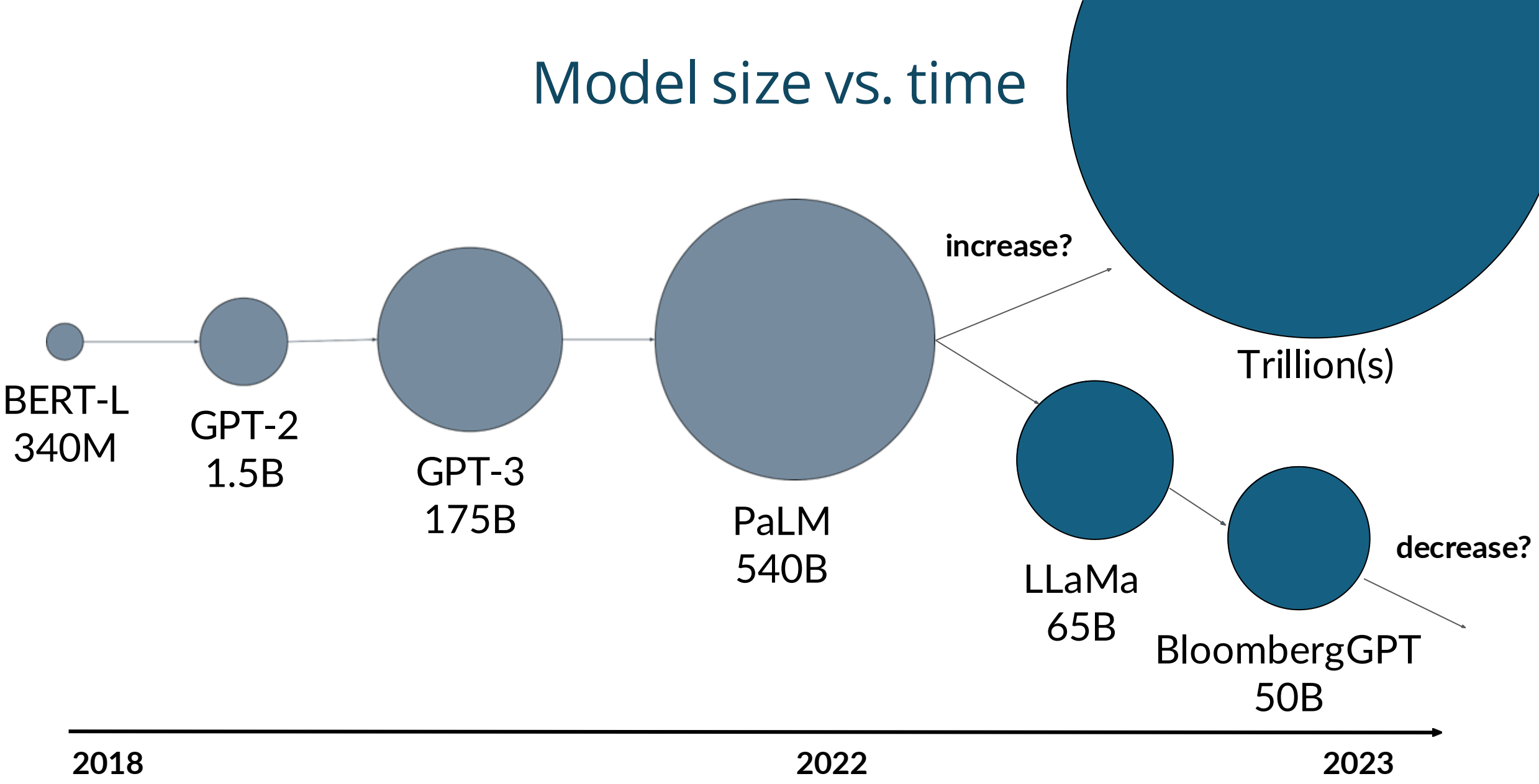
| Model | # of parameters | Compute-optimal* # of tokens (~20x) | Actual # tokens |
|---|---|---|---|
| Chinchilla | 70B | ~1.4T | 1.4T |
| LLaMA-65B | 65B | ~1.3T | 1.4T |
| GPT-3 | 175B | ~3.5T | 300B |
| OPT-175B | 175B | ~3.5T | 180B |
| BLOOM | 176B | ~3.5T | 350B |

Compute optimal training datasize is ~**20x** number of parameters

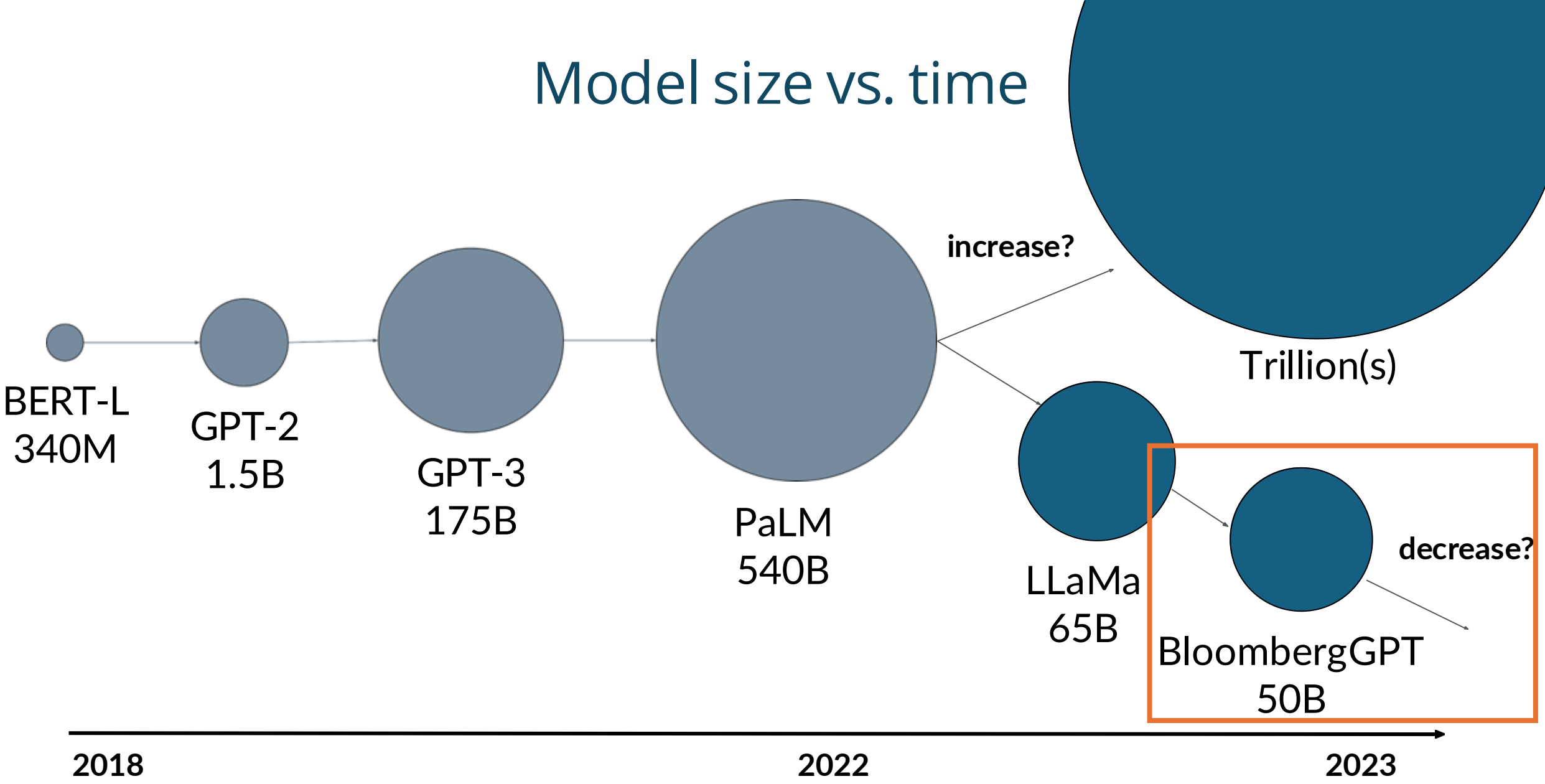
* assuming models are trained to be compute-optimal per Chinchilla paper

Sources: Hoffmann et al. 2022, "Training Compute-Optimal Large Language Models"
Touvron et al. 2023, "LLaMA: Open and Efficient Foundation Language Models"

# Model size vs. time



# Model size vs. time



# II. Challenges

## 2. Select: Foundation Model to use or pretrain

- Choose an existing model or pretrain your own
- Scaling
  - Challenges
  - Cost
  - Scaling laws
- **Pre-training for domain adaptation**

created with chatGPT

# Pre-training for domain adaptation

## Legal language

**The prosecutor had difficulty proving mens rea, as the defendant seemed unaware that his actions were illegal.**

**The judge dismissed the case, citing the principle of res judicata as the issue had already been decided in a previous trial.**

**Despite the signed agreement, the contract was invalid as there was no consideration exchanged between the parties.**

# Pre-training for domain adaptation

## Legal language

**The prosecutor had difficulty proving mens rea, as the defendant seemed unaware that his actions were illegal.**

**The judge dismissed the case, citing the principle of res judicata as the issue had already been decided in a previous trial.**

**Despite the signed agreement, the contract was invalid as there was no consideration exchanged between the parties.**

## Medical language

**After a strenuous workout, the patient experienced severe myalgia that lasted for several days.**

**After the biopsy, the doctor confirmed that the tumor was malignant and recommended immediate treatment.**

**Sig: 1 tab po qid pc & hs**

Take one tablet by mouth four times a day, after meals, and at bedtime.

# Solution →
# Pre-training for domain adaptation

# BloombergGPT: domain adaptation for finance

## BloombergGPT: A Large Language Model for Finance

**Shijie Wu[1,*], Ozan İrsoy[1,*], Steven Lu[1,*], Vadim Dabravolski[1], Mark Dredze[1,2], Sebastian Gehrmann[1], Prabhanjan Kambadur[1], David Rosenberg[1], Gideon Mann[1]**
[1] **Bloomberg, New York, NY USA**
[2] **Computer Science, Johns Hopkins University, Baltimore, MD USA**
gmann16@bloomberg.net

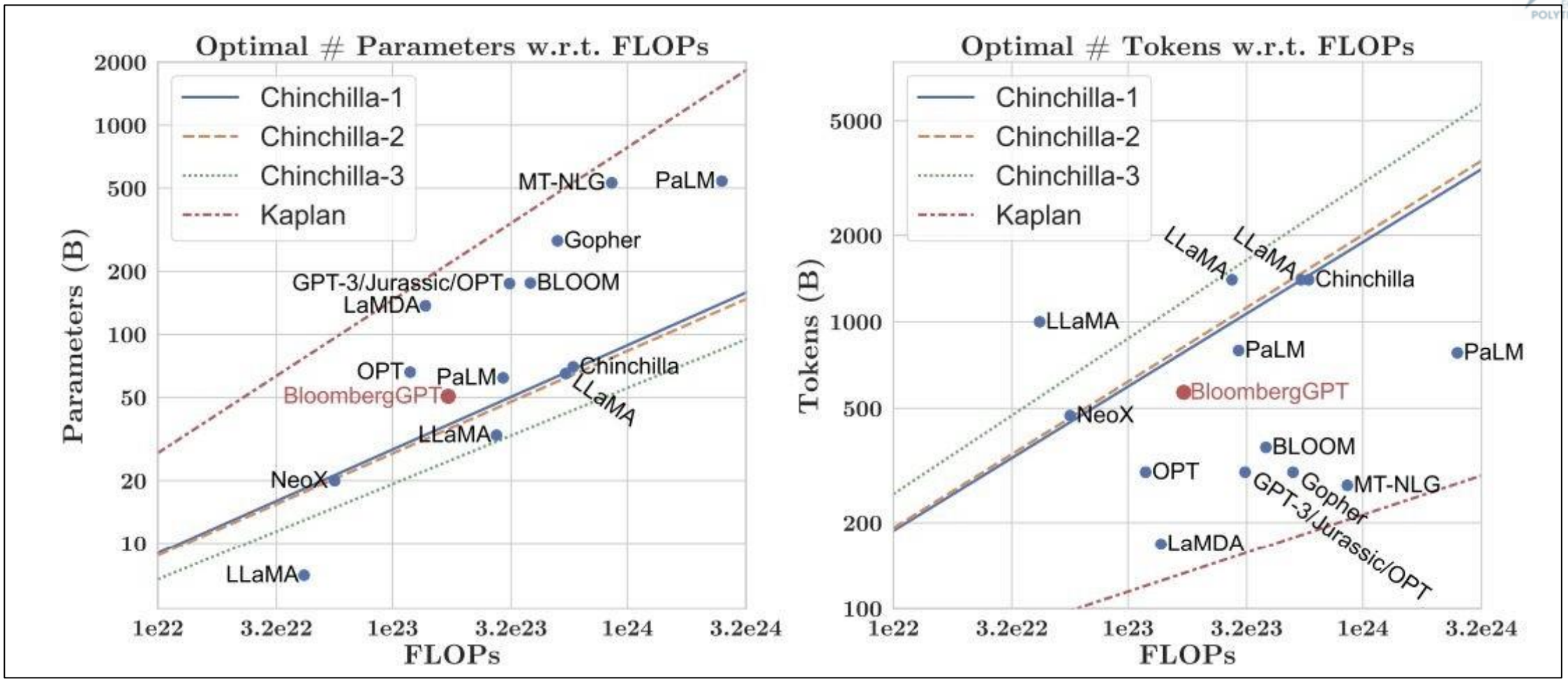
### Abstract

The use of NLP in the realm of financial technology is broad and complex, with applications ranging from sentiment analysis and named entity recognition to question answering. Large Language Models (LLMs) have been shown to be effective on a variety of tasks; however, no LLM specialized for the financial domain has been reported in literature. In this work, we present BloombergGPT, a 50 billion parameter language model that is trained on a wide range of financial data. We construct a 363 billion token dataset based on Bloomberg's extensive data sources, perhaps the largest domain-specific dataset yet, augmented with 345 billion tokens from general purpose datasets. We validate BloombergGPT on standard LLM benchmarks, open financial benchmarks, and a suite of internal benchmarks that most accurately reflect our intended usage. Our mixed dataset training leads to a model that outperforms existing models on financial tasks by significant margins without sacrificing performance on general LLM benchmarks. Additionally, we explain our modeling choices, training process, and evaluation methodology. As a next step, we plan to release training logs (Chronicles) detailing our experience in training BloombergGPT.

~51% **Financial (Public & Private)**

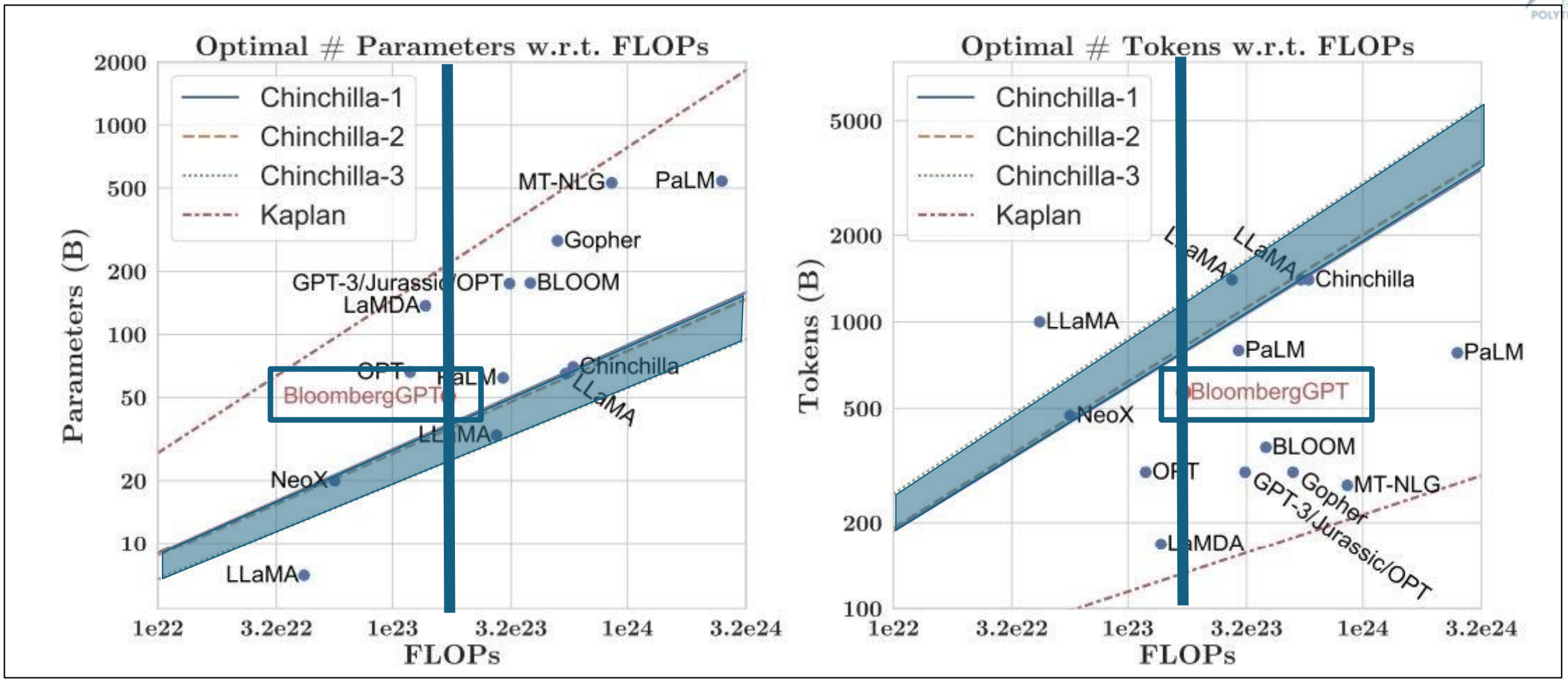
~49% **Other (Public)**

# BloombergGPT relative to other LLMs




Source: Wu et al. 2023, "BloombergGPT: A Large Language Model for Finance"

# BloombergGPT relative to other LLMs




Source: Wu et al. 2023, "BloombergGPT: A Large Language Model for Finance"

# II. Challenges

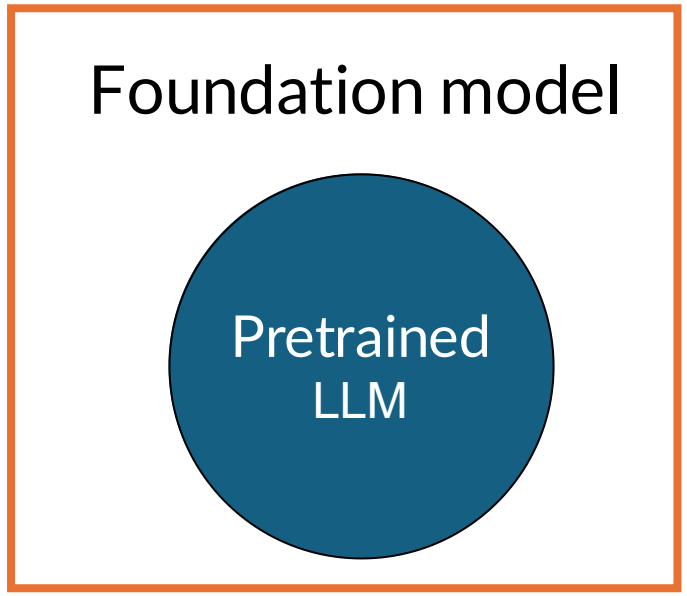
**SUMMARY**

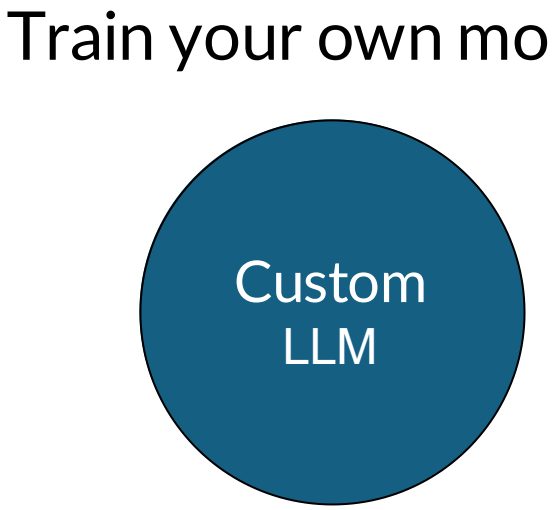
**2. Select: Foundation Model to use or pretrain**

- Choose an existing model or pretrain your own
- Scaling
  - Challenges
  - Cost
  - Scaling laws
- Pre-training for domain adaptation

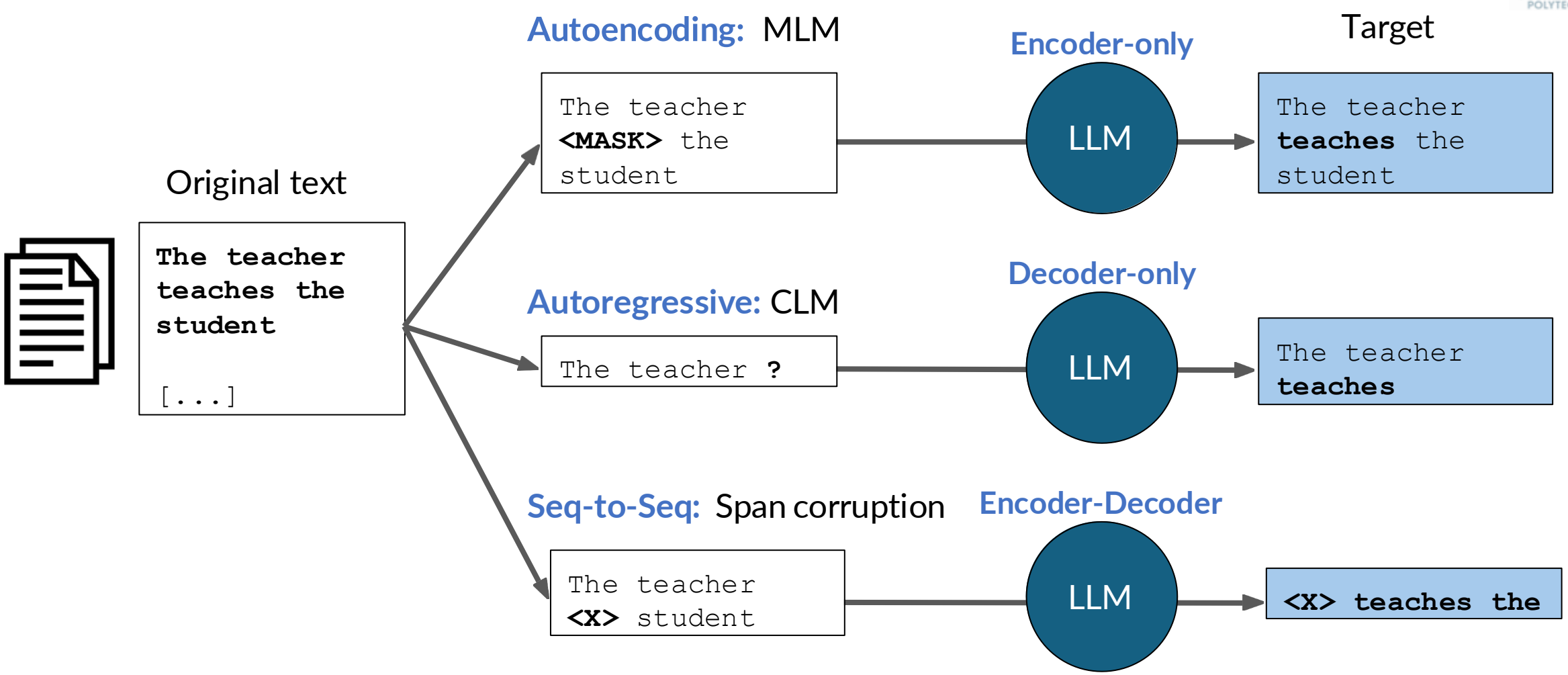
created with chatGPT

# Considerations for choosing a model



# Model architectures and pre-training objectives



# Compute…

```
OutOfMemoryError: CUDA out of memory.
```

# GPU RAM needed to train larger models

**As model sizes get larger, you will need to split your model across multiple GPUs for training**

**1B param model**

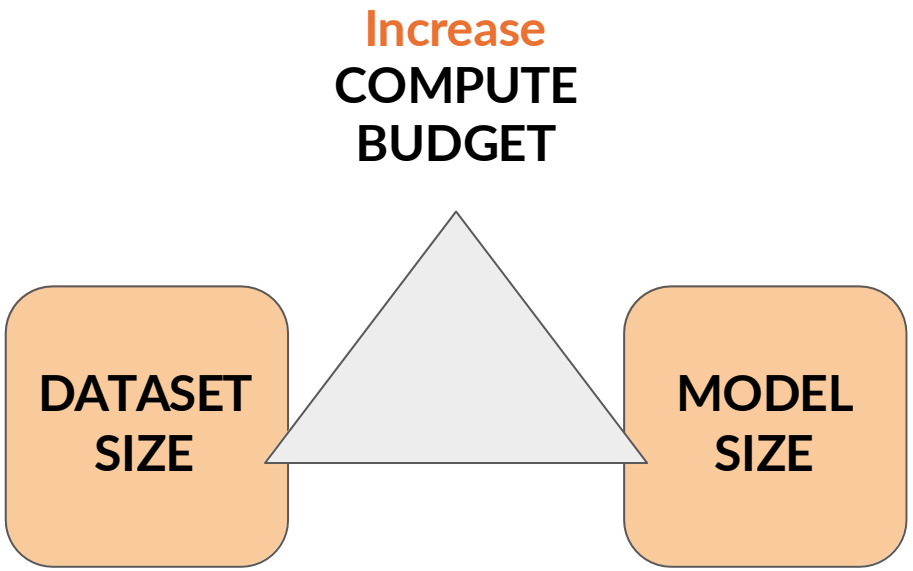
**4,200 GB @ 32-bit full precision**

**175B param model**

**500B param model**

**12,000 GB @ 32-bit full precision**

# Increase compute budget → increase performance?

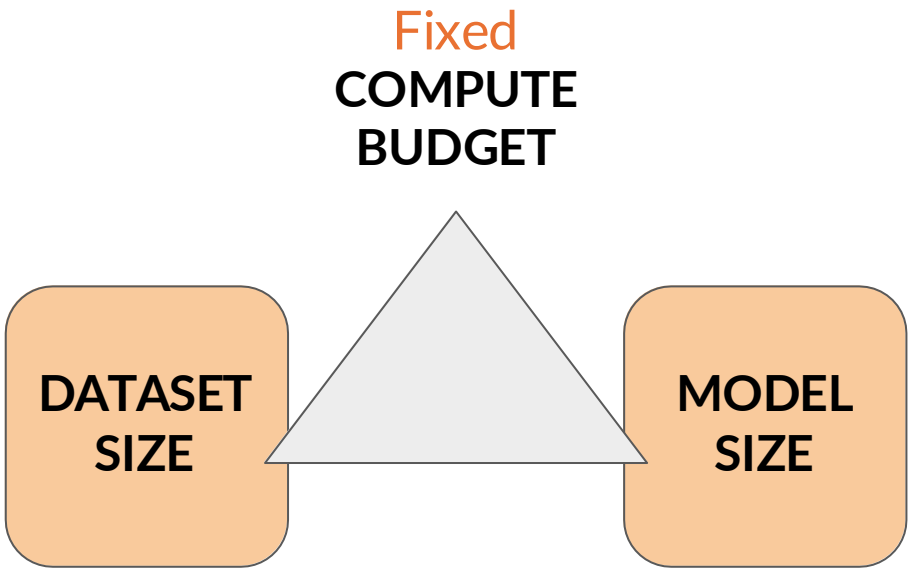


# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer Base | 12 | 512 | 8 | 65M | | 8xP100 (12h) |
| Transformer Large | 12 | 1024 | 16 | 213M | | 8xP100 (12h) |
| Bert Base | 12 | 768 | 12 | 110M | 13GB | |
| Bert Large | 24 | 1024 | 16 | 340M | 13GB | |
| XLNet Large | 24 | 1024 | 16 | ~340M | 126GB | 512xTPUv3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160GB | 1024xV100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40GB | |
| Megatron-LM | 72 | 2072 | 32 | 8.3B | 174GB | 512xV100 GPU (9 days) |
| Turing NLG | | | 28 | 17B | ? | 256xV100 GPU |

~350k euros!

# Chinchilla scaling laws for model + dataset size




| Model | # params | Compute-optimal* # of tokens (~20x) | Actual tokens |
|---|---|---|---|
| Chinchilla | 70B | ~1.4T | 1.4T |
| LLaMA-65B | 65B | ~1.3T | 1.4T |
| GPT-3 | 175B | ~3.5T | 300B |
| OPT-175B | 175B | ~3.5T | 180B |
| BLOOM | 176B | ~3.5T | 350B |

Compute optimal training datasize is **~20x** number of parameters

* assuming models are trained to be compute-optimal per Chinchilla paper

Sources: Hoffmann et al. 2022, "Training Compute-Optimal Large Language Models"
Touvron et al. 2023, "LLaMA: Open and Efficient Foundation Language Models"

# Pre-training for domain adaptation

## Legal language

**The prosecutor had difficulty proving mens rea, as the defendant seemed unaware that his actions were illegal.**

**The judge dismissed the case, citing the principle of res judicata as the issue had already been decided in a previous trial.**

**Despite the signed agreement, the contract was invalid as there was no consideration exchanged between the parties.**

## Medical language

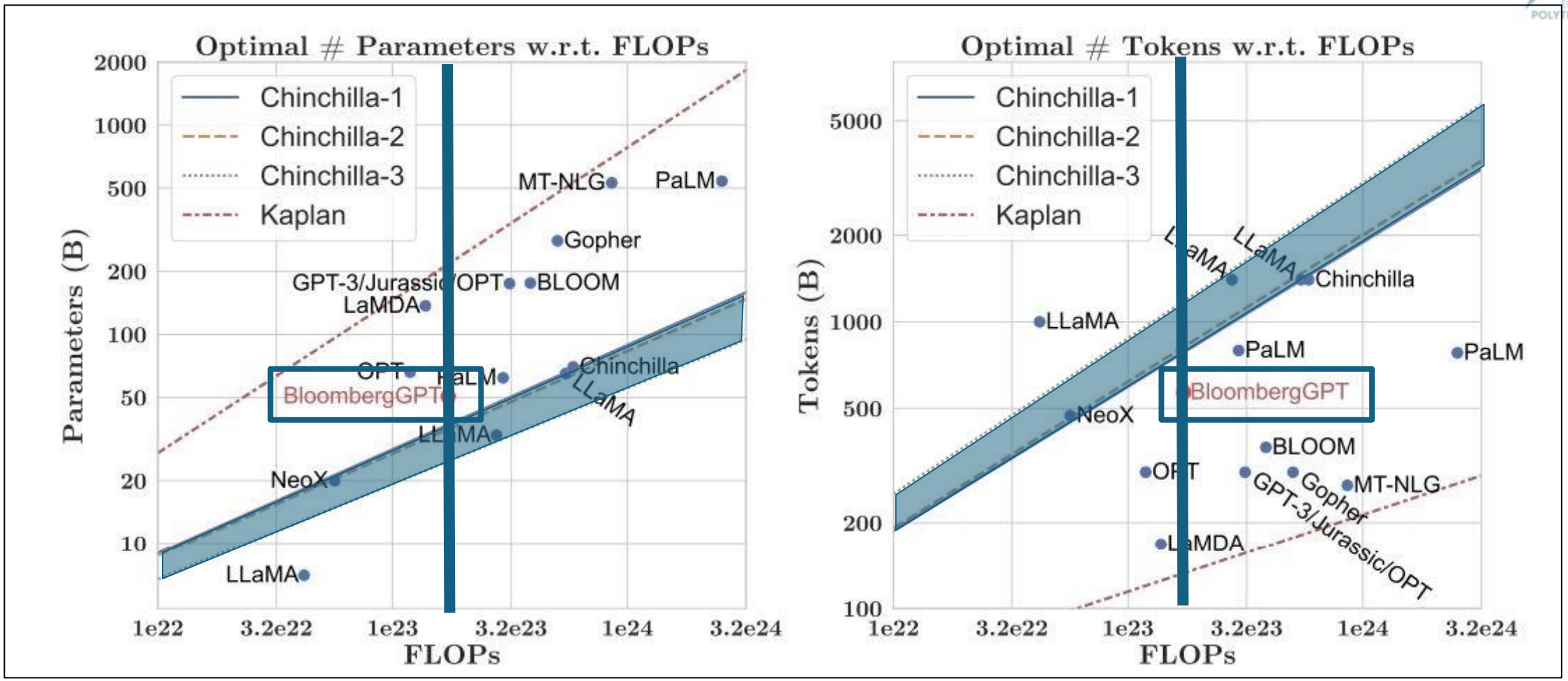
**After a strenuous workout, the patient experienced severe myalgia that lasted for several days.**

**After the biopsy, the doctor confirmed that the tumor was malignant and recommended immediate treatment.**

**Sig: 1 tab po qid pc & hs**

Take one tablet by mouth four times a day, after meals, and at bedtime.

# BloombergGPT relative to other LLMs




Source: Wu et al. 2023, "BloombergGPT: A Large Language Model for Finance"

# II. Challenges

**1. Definition:** Define usecase

**2. Select:** Foundation Model to use or pretrain

3. **Adapt:** Foundation Model

# II. Challenges

**1. Definition:** Define usecase

**2. Select:** Foundation Model to use or pretrain

**3. Adapt: Foundation Model**

# II. Challenges

## 3. Adapt: Foundation Model

- Prompting & Prompt Engineering
- Fine-tuning
  - Instruction fine-tuning
  - Fine-tuning on a single task
  - Fine-tuning on multiple tasks
  - Parameter efficient fine-tuning (PEFT)

created with chatGPT

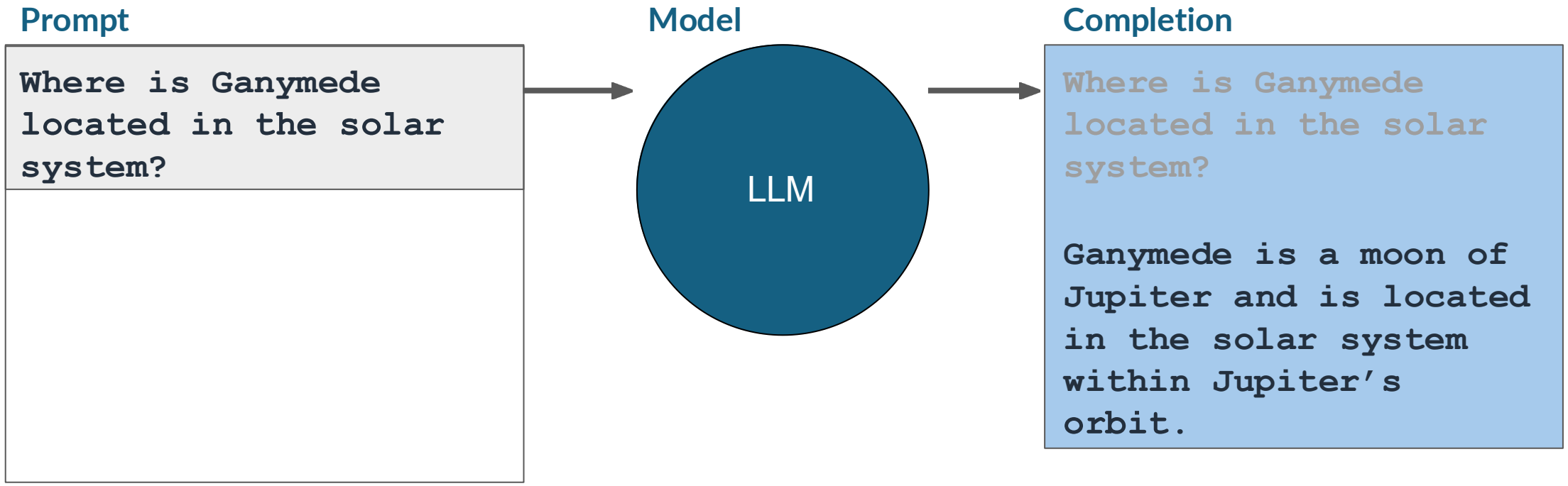
# II. Challenges

## 3. Adapt: Foundation Model

- **Prompting & Prompt Engineering**
- Fine-tuning
  - Instruction fine-tuning
  - Fine-tuning on a single task
  - Fine-tuning on multiple tasks
  - Parameter efficient fine-tuning (PEFT)

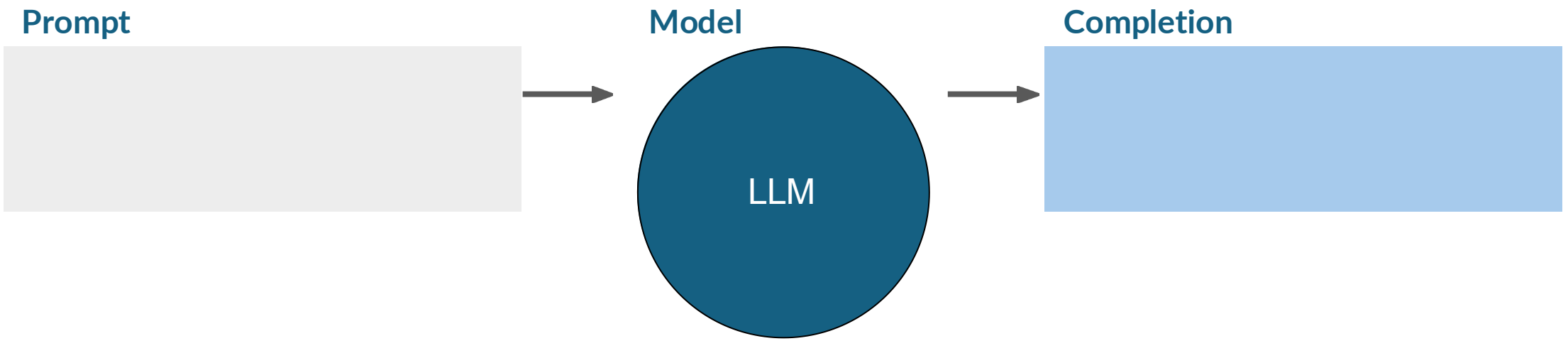
created with chatGPT

# Prompting and prompt engineering

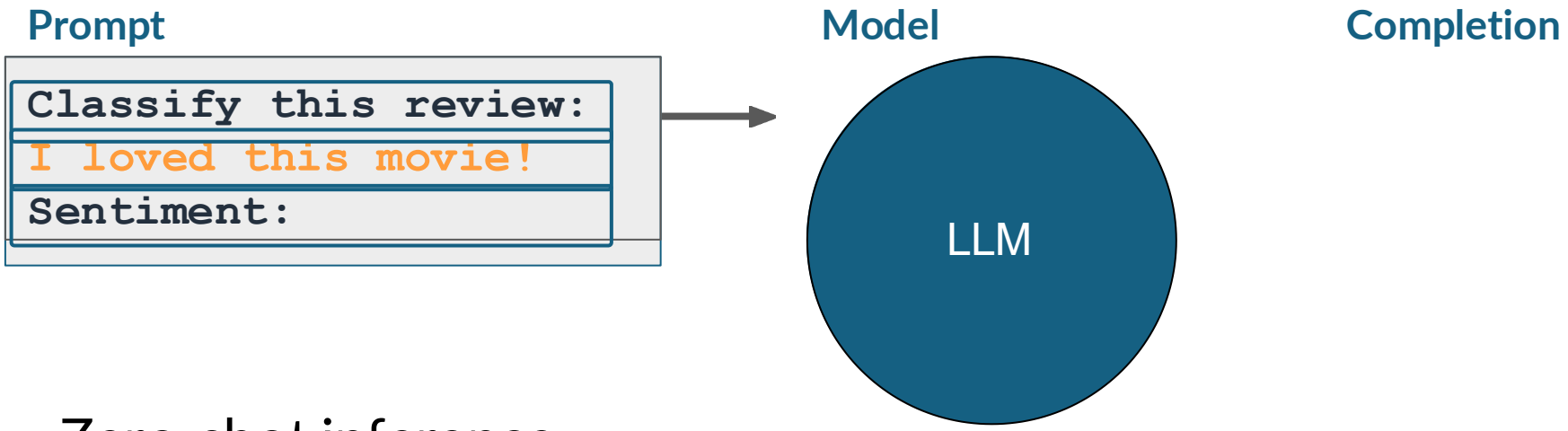



**Context window**: typically a few thousand words

# In-context learning (ICL) - zero shot inference

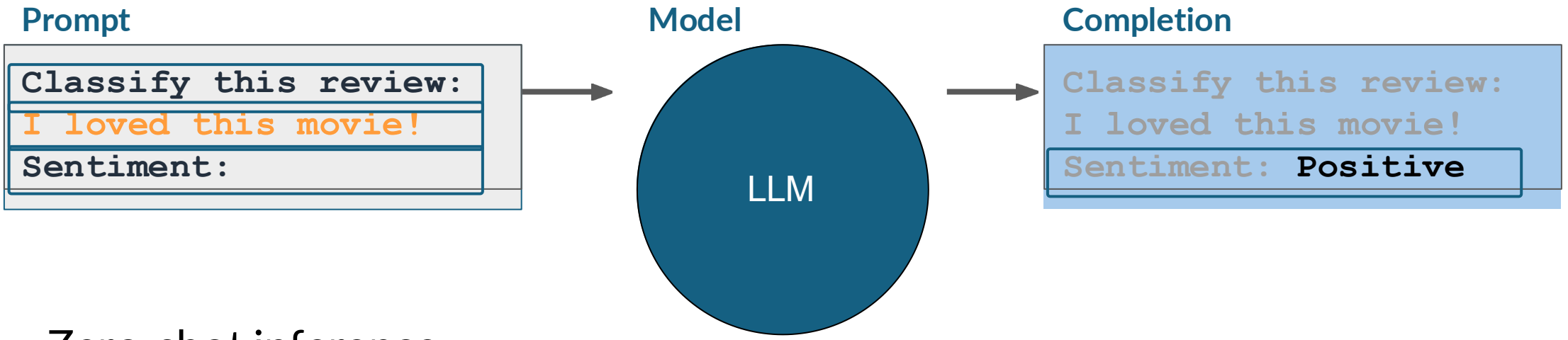


# In-context learning (ICL) - zero shot inference



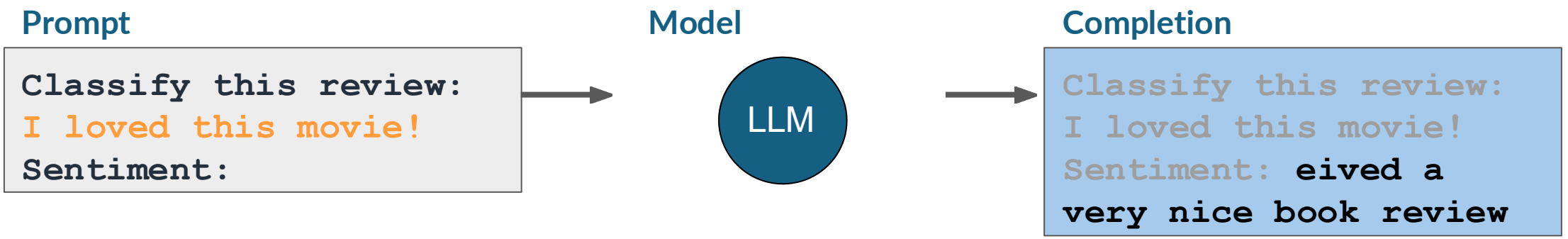

Zero-shot inference

# In-context learning (ICL) - zero shot inference




Zero-shot inference

# In-context learning (ICL) - zero shot inference



# In-context learning (ICL) - one shot inference

**Prompt**

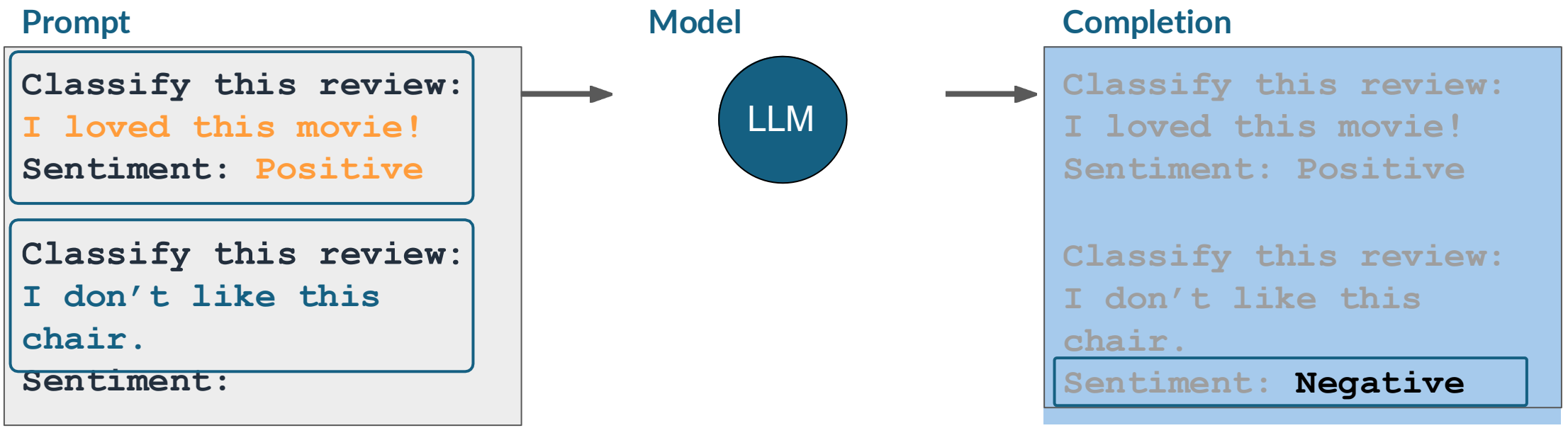
```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment:
```

**Model**

LLM

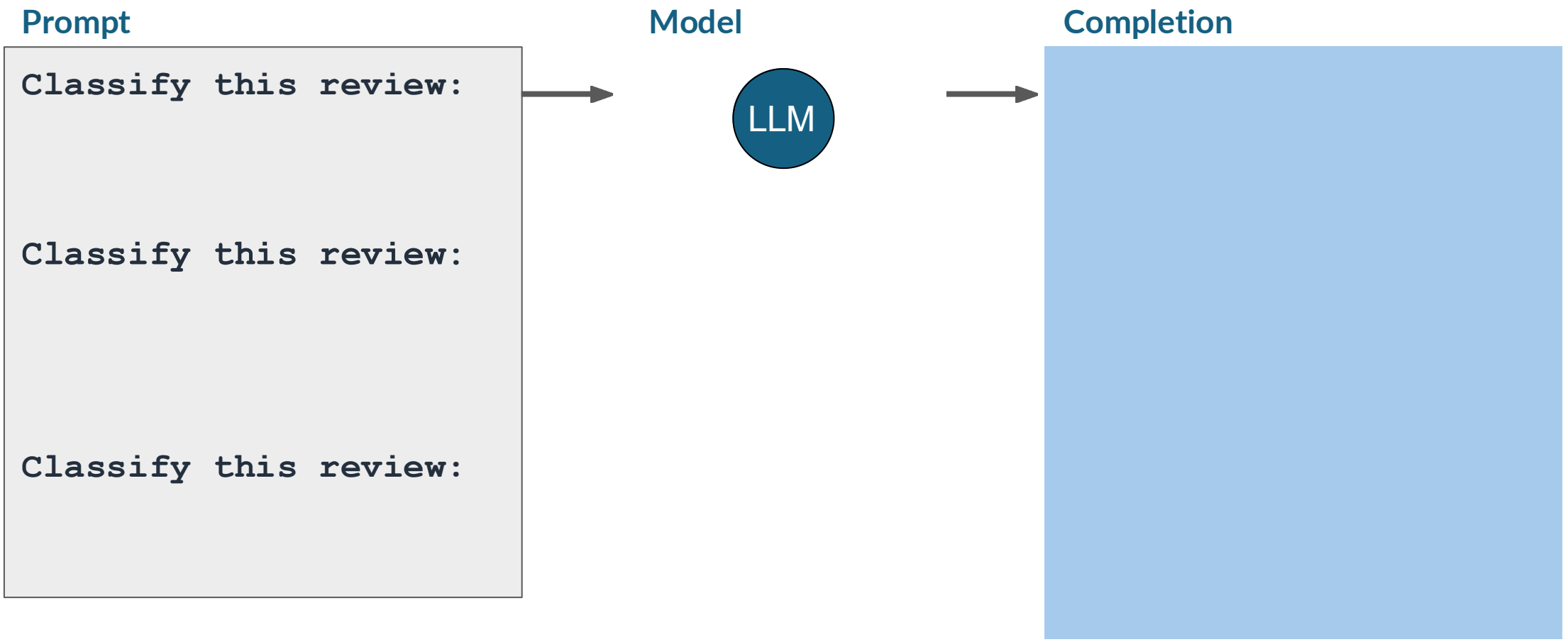
**Completion**

# In-context learning (ICL) - one shot inference

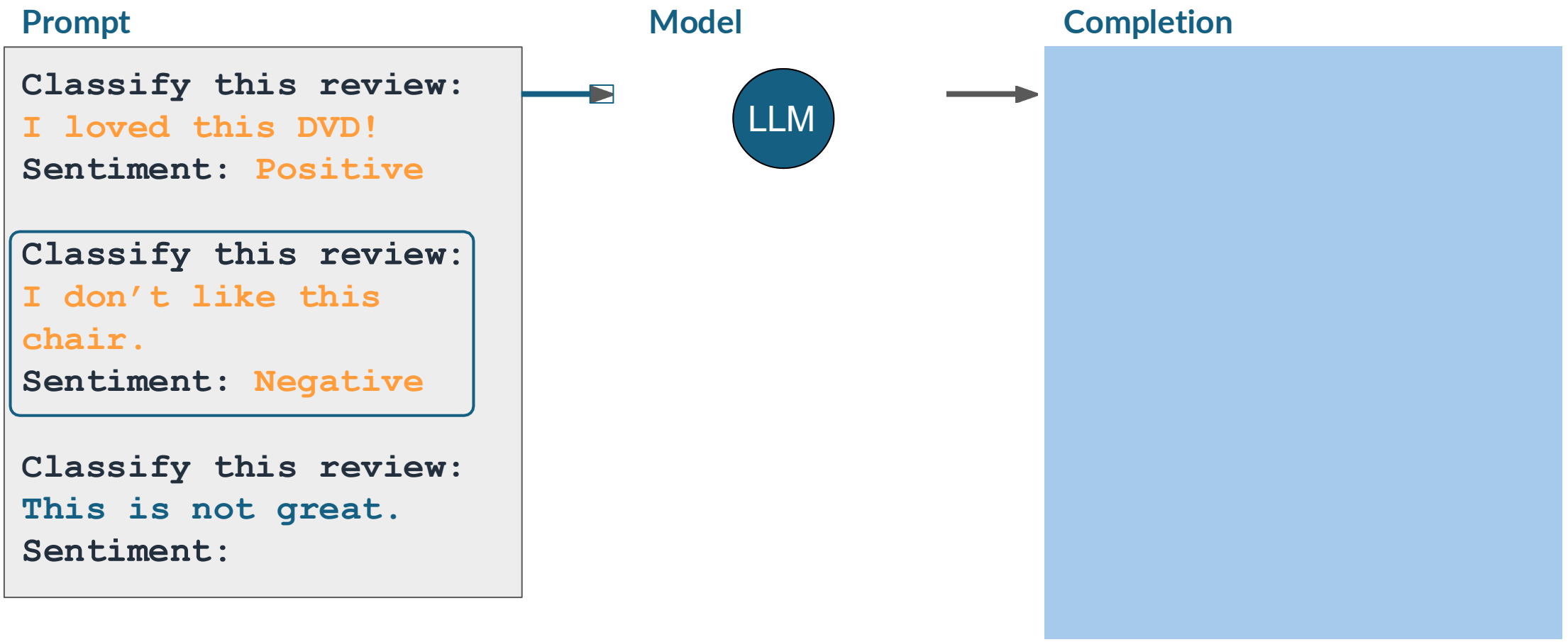



One-shot inference

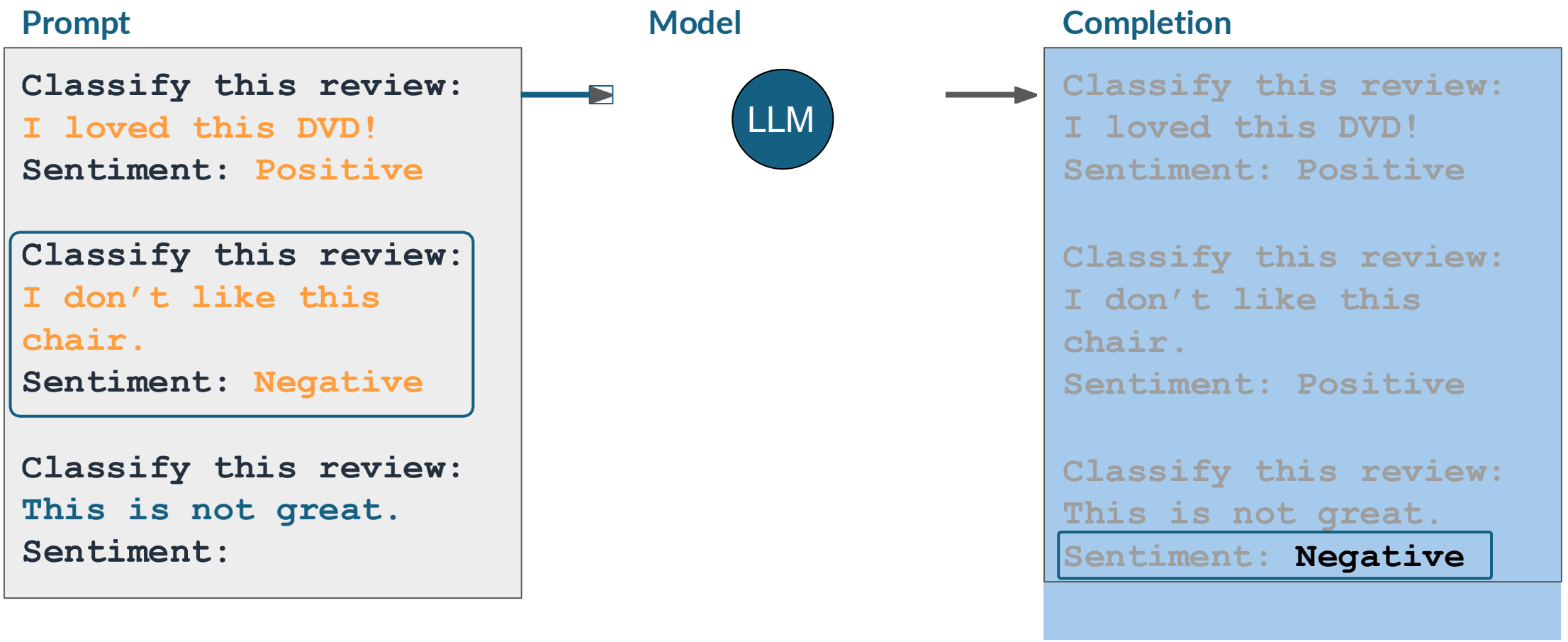
# In-context learning (ICL) - few shot inference



# In-context learning (ICL) - few shot inference



# In-context learning (ICL) - few shot inference



# Summary: In-context learning (ICL)

**Prompt // Zero Shot**

```
Classify this review:
I loved this movie!
Sentiment:
```

**Prompt // One Shot**

```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment:
```

**Prompt // Few Shot** >5 or 6 examples

```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment: Negative

Classify this review:
Who would use this
product?
Sentiment:
```

**Context Window**
(few thousand words)

# II. Challenges

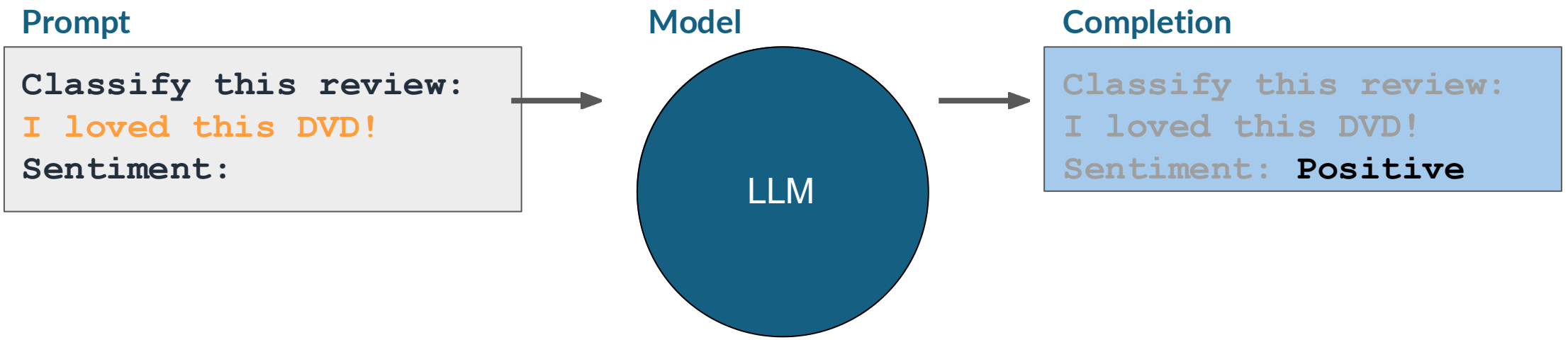
## 3. Adapt: Foundation Model

- Prompting & Prompt Engineering
- **Fine-tuning**
  - Instruction fine-tuning
  - Fine-tuning on a single task
  - Fine-tuning on multiple tasks
  - Parameter efficient fine-tuning (PEFT)

created with chatGPT

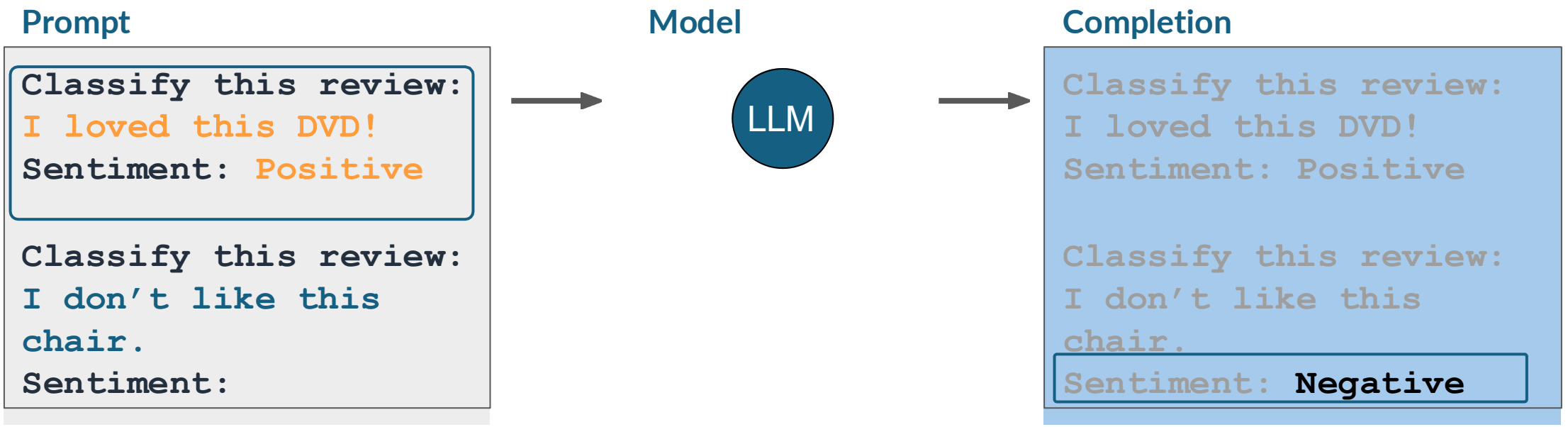
# Fine-tuning

# In-context learning (ICL) - zero shot inference



# In-context learning (ICL) - zero shot inference

# In-context learning (ICL) - one/few shot inference




One-shot or Few-shot Inference

# Limitations of in-context learning

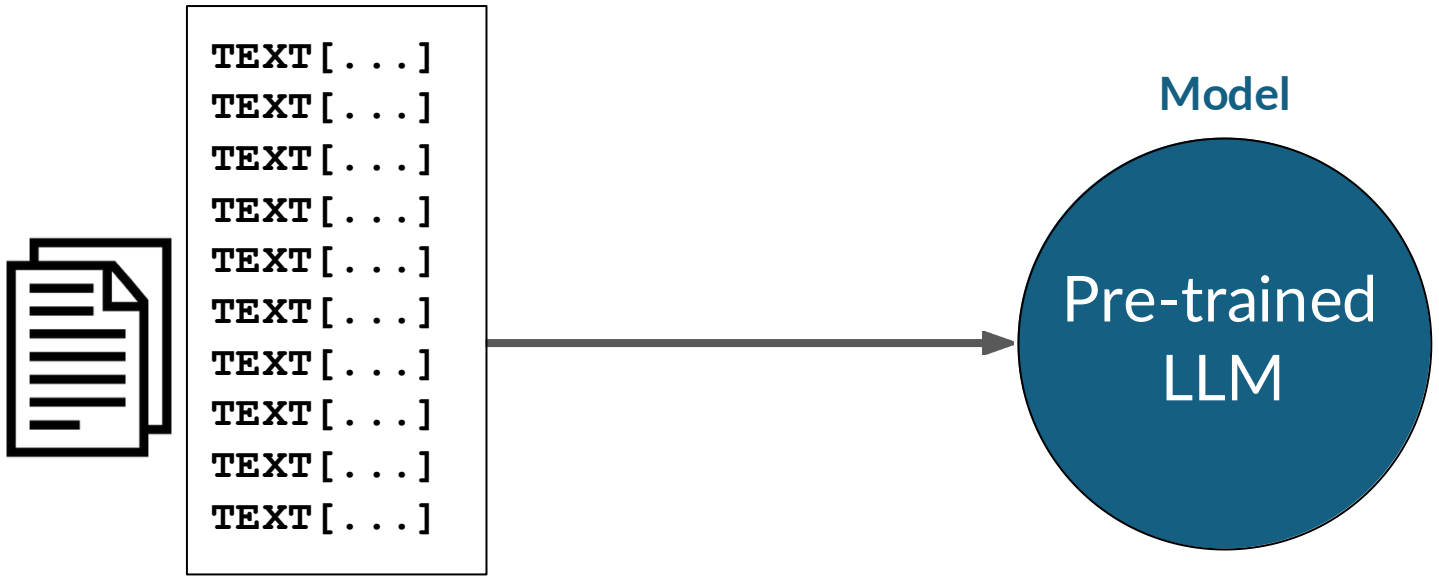
```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this chair.
Sentiment: Negative

Classify this review:
This sofa is so ugly.
Sentiment: Negative

Classify this review:
Who would use this product?
Sentiment:
```

Even with multiple examples

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

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

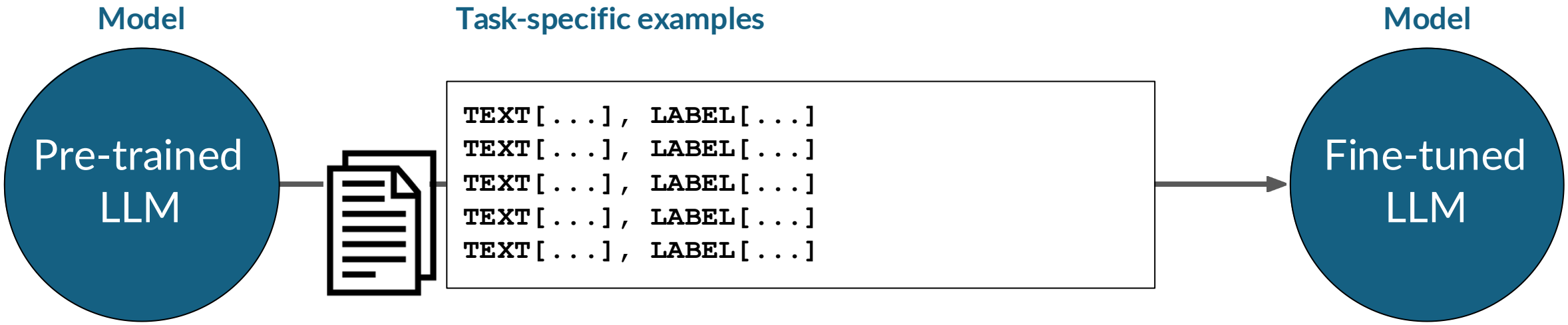
# LLM fine-tuning at a high level




GB - TB - PB
of unstructured textual data

**LLM pre-training**

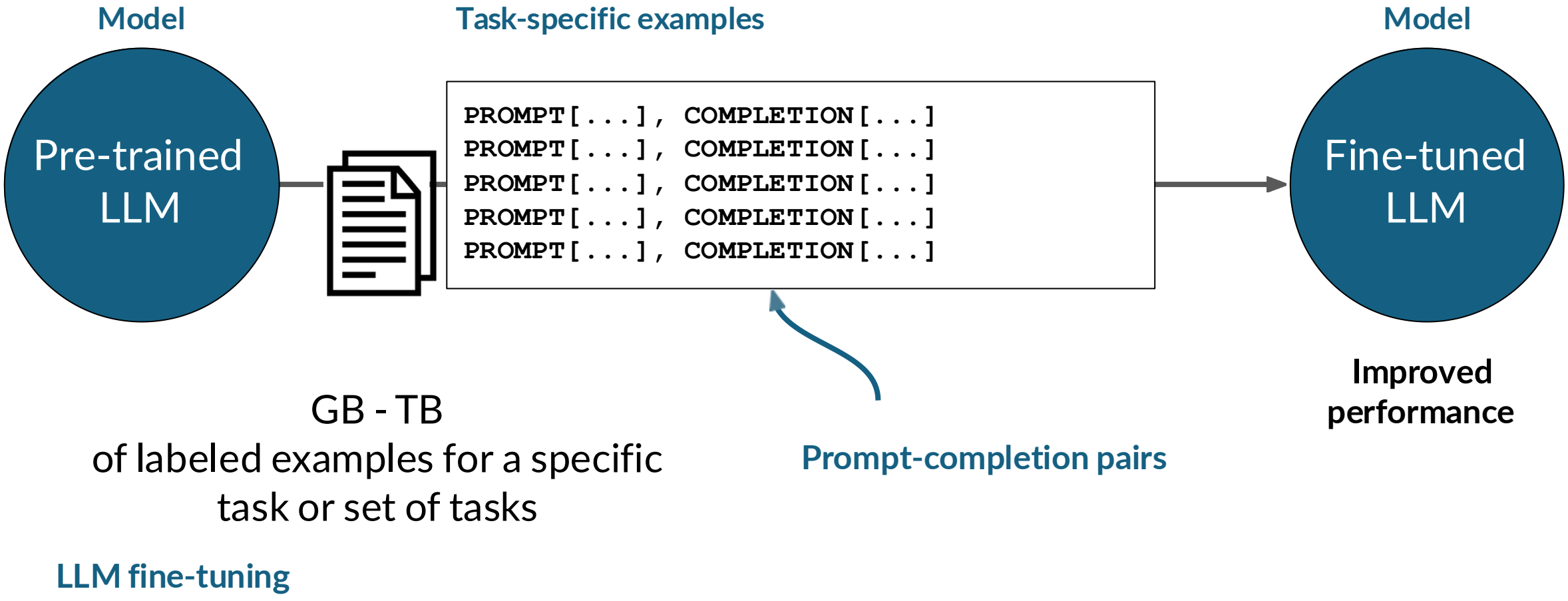
# LLM fine-tuning at a high level




GB - TB
of labeled examples for a specific task or set of tasks

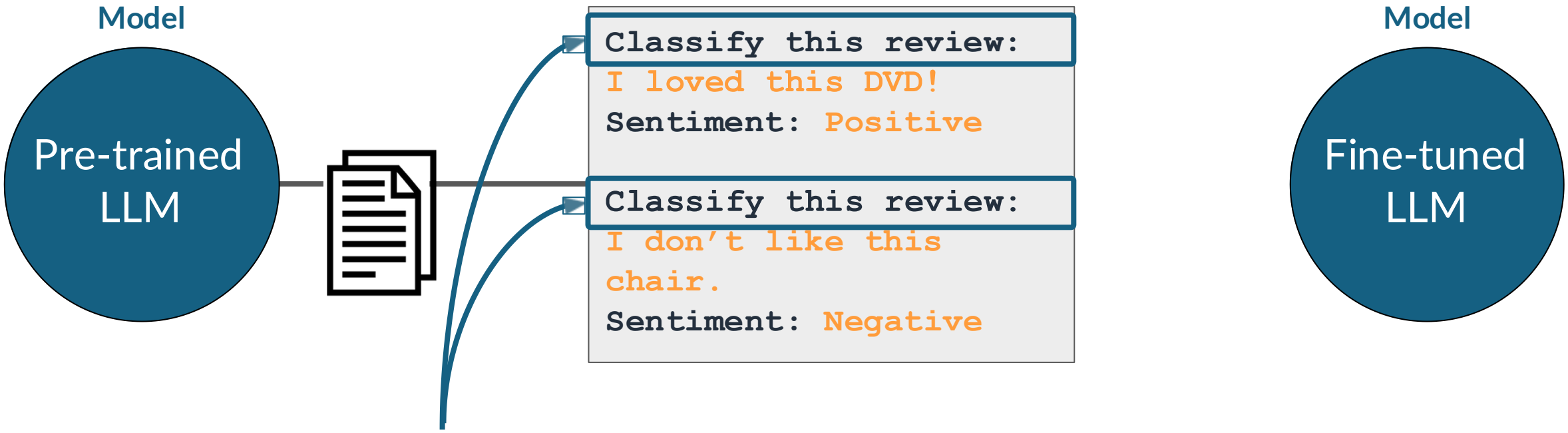
**LLM fine-tuning**

# LLM fine-tuning at a high level



# Instruction fine-tuning

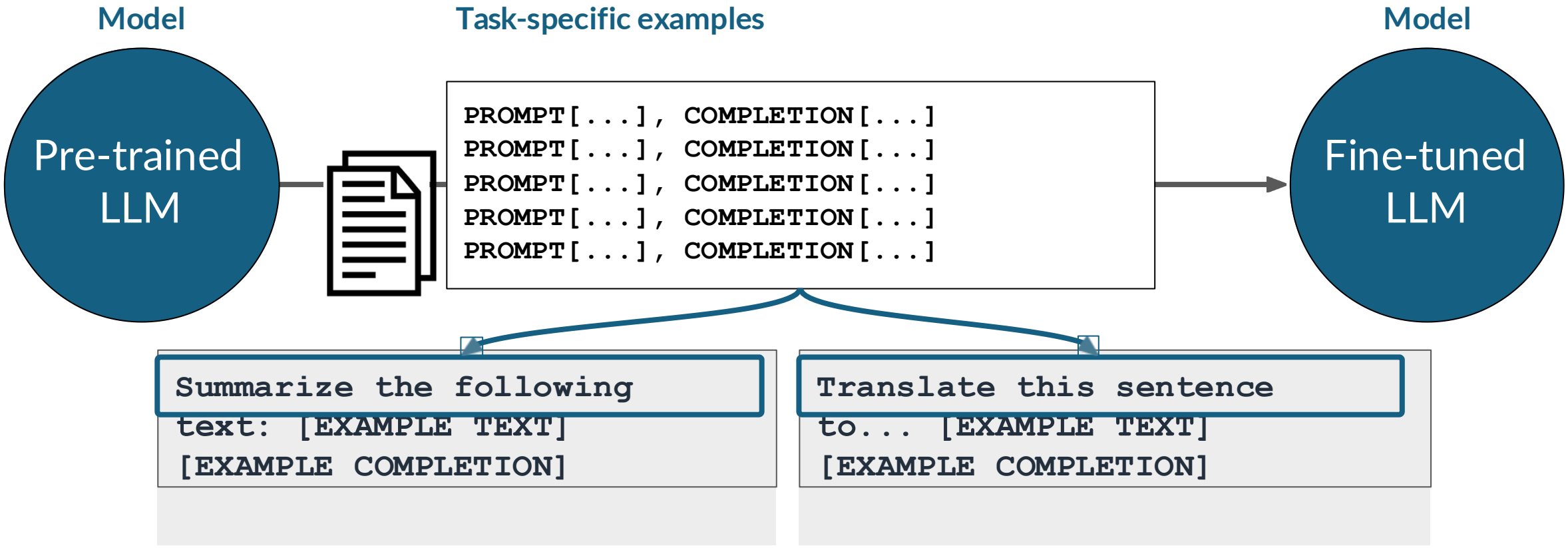
# Using prompts to fine-tune LLMs with instruction




**Each prompt/completion pair includes a specific "instruction" to the LLM**

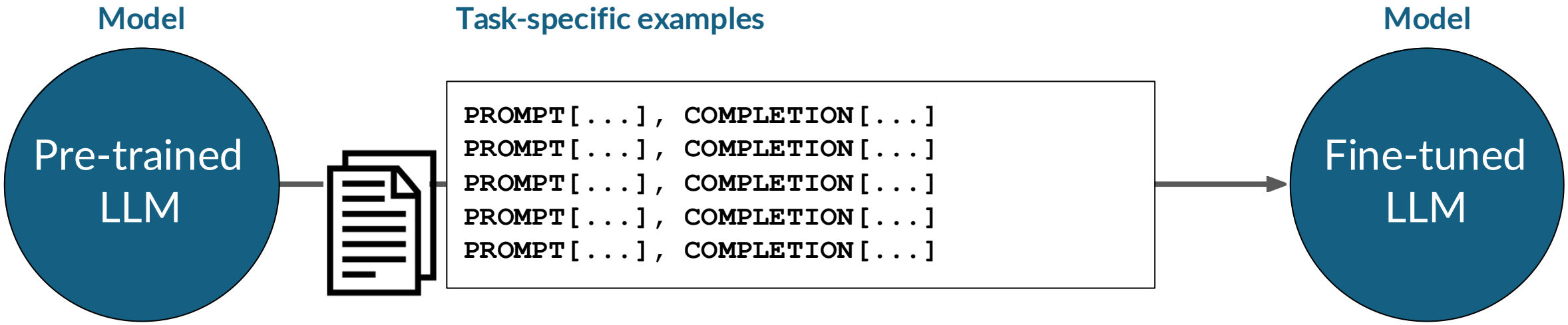
**LLM fine-tuning**

# Using prompts to fine-tune LLMs with instruction




**LLM fine-tuning**

# Using prompts to fine-tune LLMs with instruction




## LLM fine-tuning

# Sample prompt instruction templates

- To store and process all the gradients, optimizers, …, full fine-tuning / Instruction fine-tuning requires:
  - memory
  - compute budget

→ updates all parameters

- How to instruction tune?
  - Start from datasets!

# LLM fine-tuning process

**Prepared instruction dataset**

**Training splits**

```
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
```

**Training**

```
PROMPT[...], COMPLETION[...]
...
```

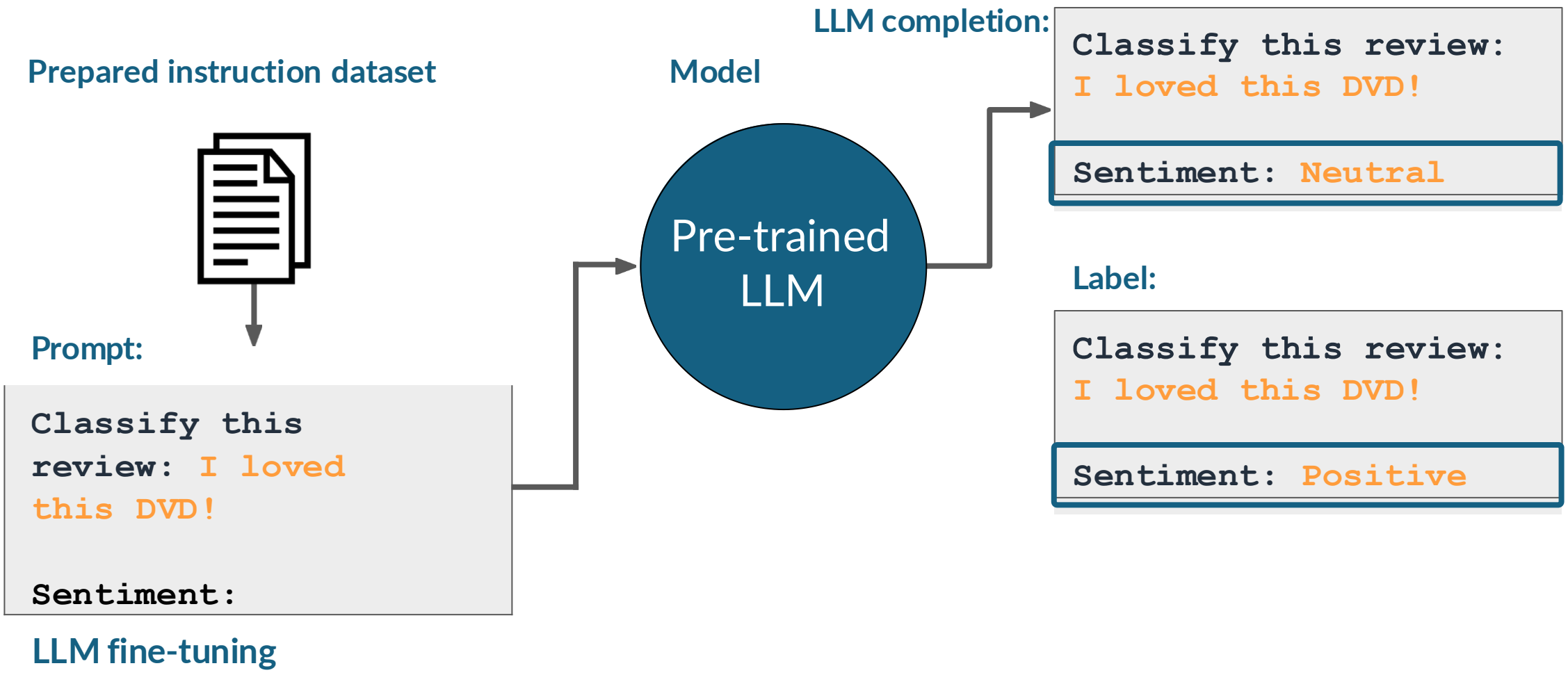
**Validation**

```
PROMPT[...], COMPLETION[...]
...
```

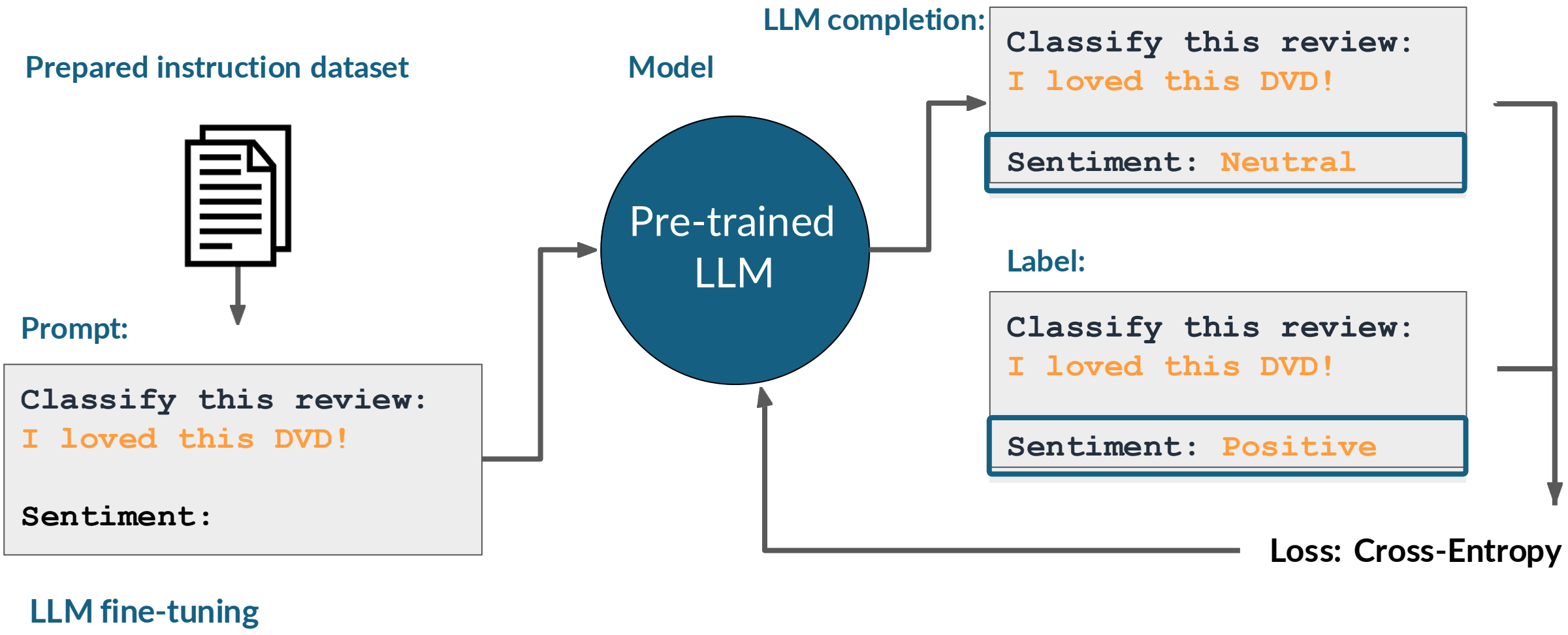
**Test**

**LLM fine-tuning**

# LLM fine-tuning process



# LLM fine-tuning process



# LLM fine-tuning process

**Prepared instruction dataset**

**Training splits**

```
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
```

**Training**

```
PROMPT[...], COMPLETION[...]
...
```

**Validation**

**validation_accuracy**

```
PROMPT[...], COMPLETION[...]
...
```

**Test**

**LLM fine-tuning**

# LLM fine-tuning process

**Prepared instruction dataset**

**Training splits**

```
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
```

**Training**

```
PROMPT[...], COMPLETION[...]
...
```

**Validation**

```
PROMPT[...], COMPLETION[...]
...
```

**Test**

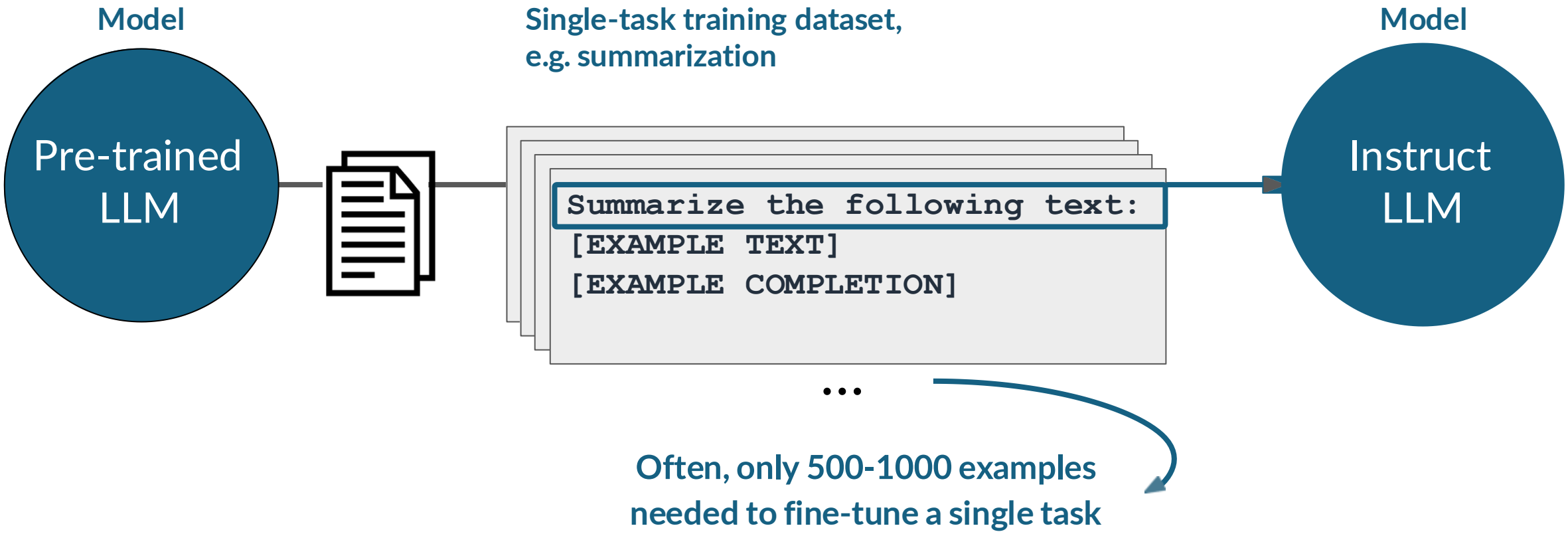
**test_accuracy**

**LLM fine-tuning**

# LLM fine-tuning process

# Fine-tuning on a single task

# Fine-tuning on a single task



# Catastrophic forgetting

- Fine-tuning can significantly increase the performance of a model on a specific task…

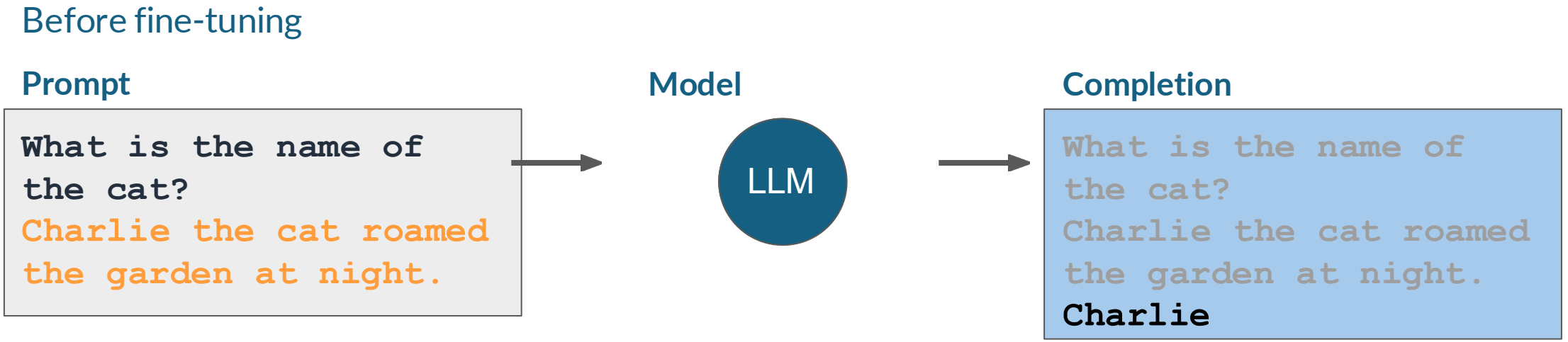
Before fine-tuning

# Catastrophic forgetting

- Fine-tuning can significantly increase the performance of a model on a specific task…

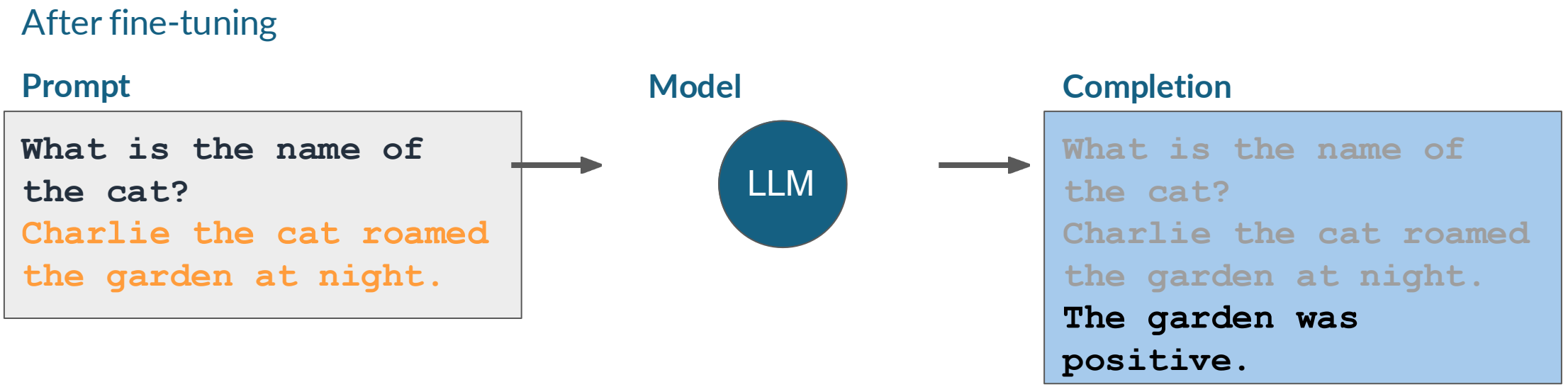
# Catastrophic forgetting

- …but can lead to reduction in ability on other tasks



# Catastrophic forgetting

- …but can lead to reduction in ability on other tasks

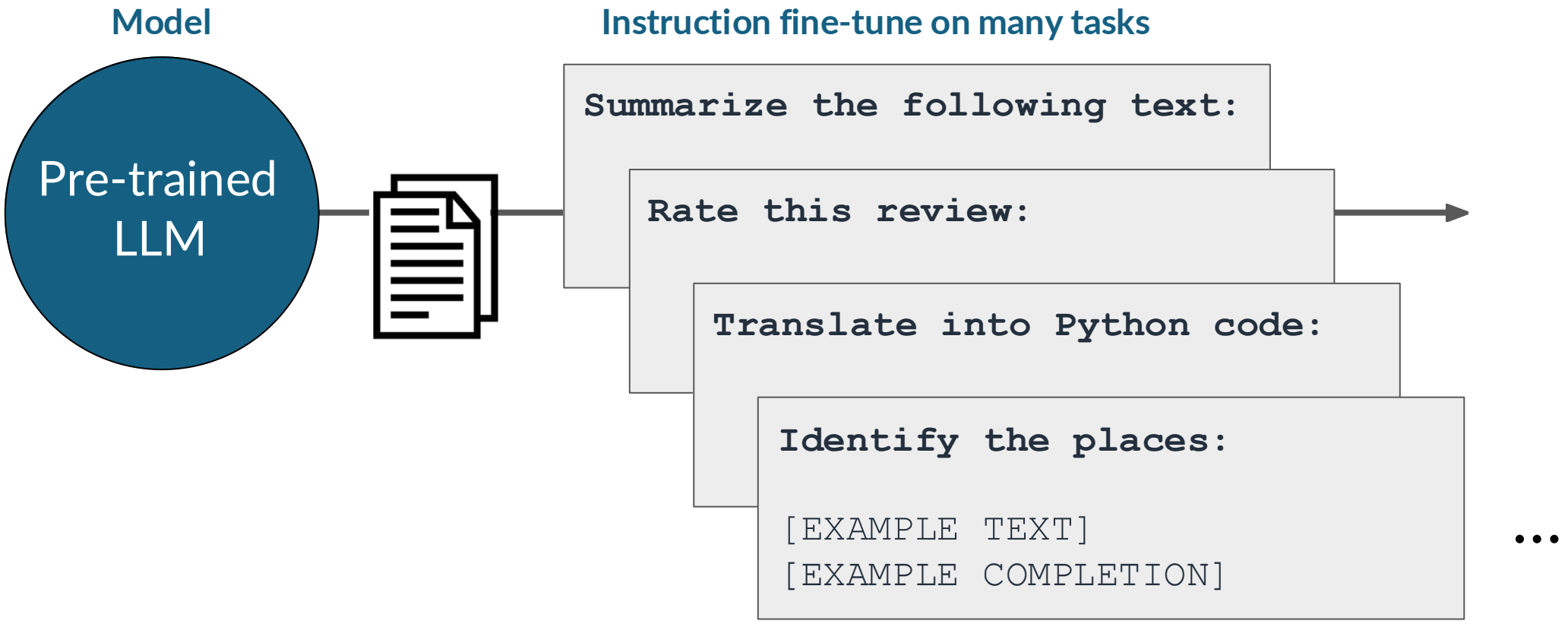


# How to avoid catastrophic forgetting

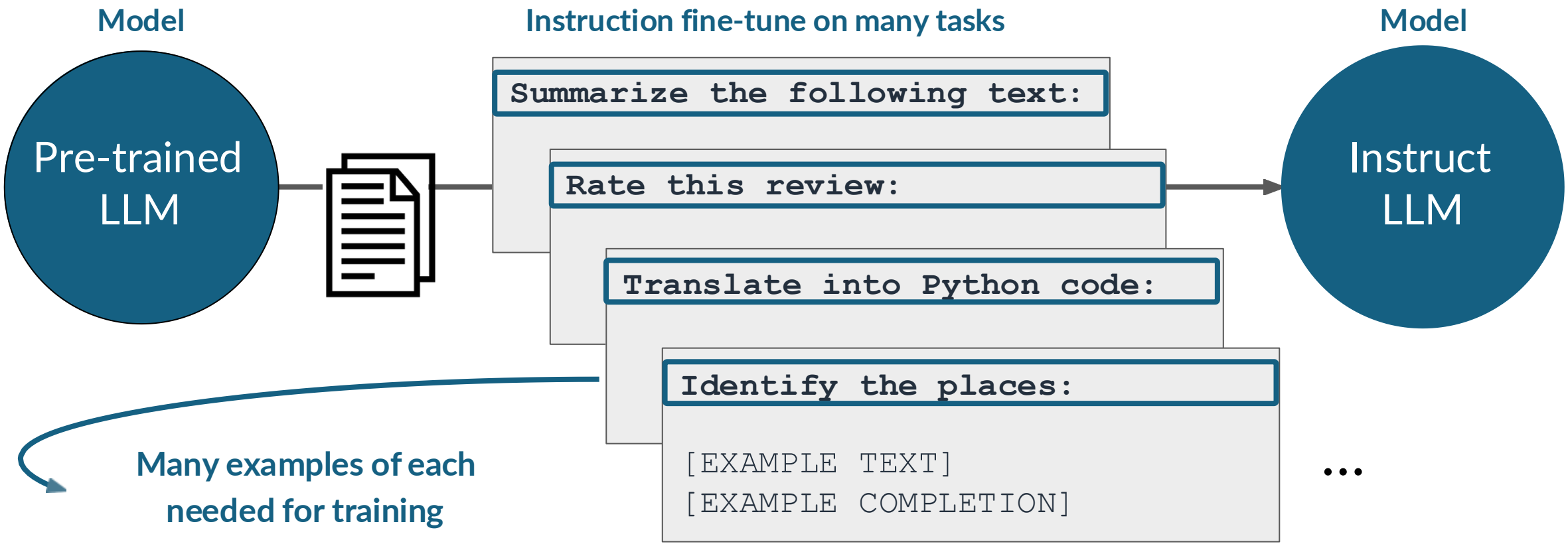
- First note that you might not have to!
- Fine-tune on **multiple tasks** at the same time
- Consider **Parameter Efficient Fine-tuning** (PEFT)

# Multi-task, instruction fine-tuning

# Multi-task, instruction fine-tuning



# Multi-task, instruction fine-tuning



# II. Challenges

## 3. Adapt: Foundation Model

- Prompting & Prompt Engineering
- **Fine-tuning**
  - Instruction fine-tuning
  - Fine-tuning on a single task
  - Fine-tuning on multiple tasks
  - **Parameter efficient fine-tuning (PEFT)**
    - LoRA
    - Prompt tuning

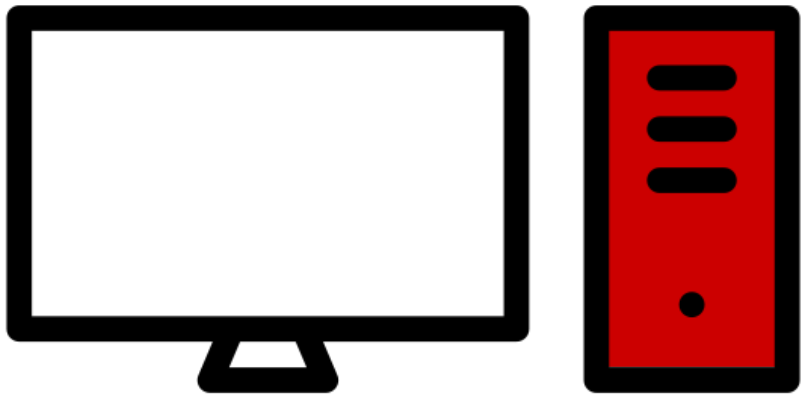
created with chatGPT

# Parameter efficient fine-tuning (PEFT)

# Full fine-tuning of LLMs is challenging

- Computationally intensive
- Store additional items (hundreds of GBs)
- Memory allocation

LLM



Temp memory

Forward Activations

Gradients

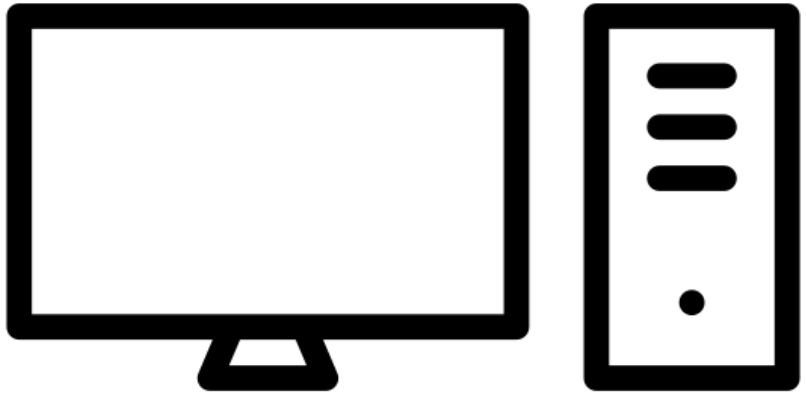
Optimizer states

Trainable Weights

12-20x weights

# Parameter efficient fine-tuning (PEFT)

Small number of trainable layers

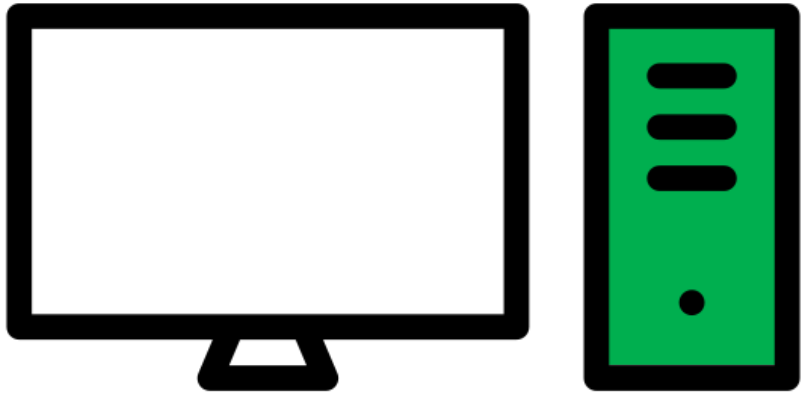



LLM with most layers frozen

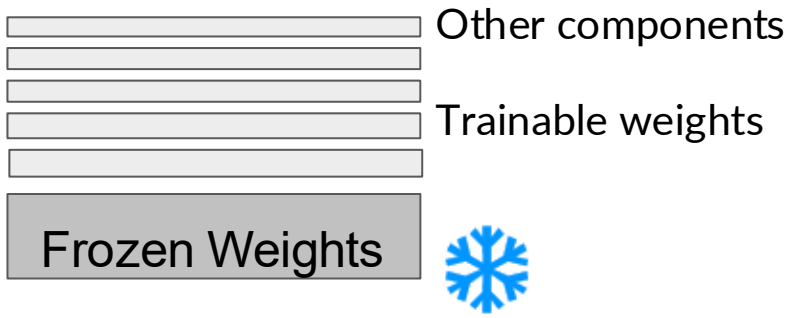
# Parameter efficient fine-tuning (PEFT)

New trainable layers

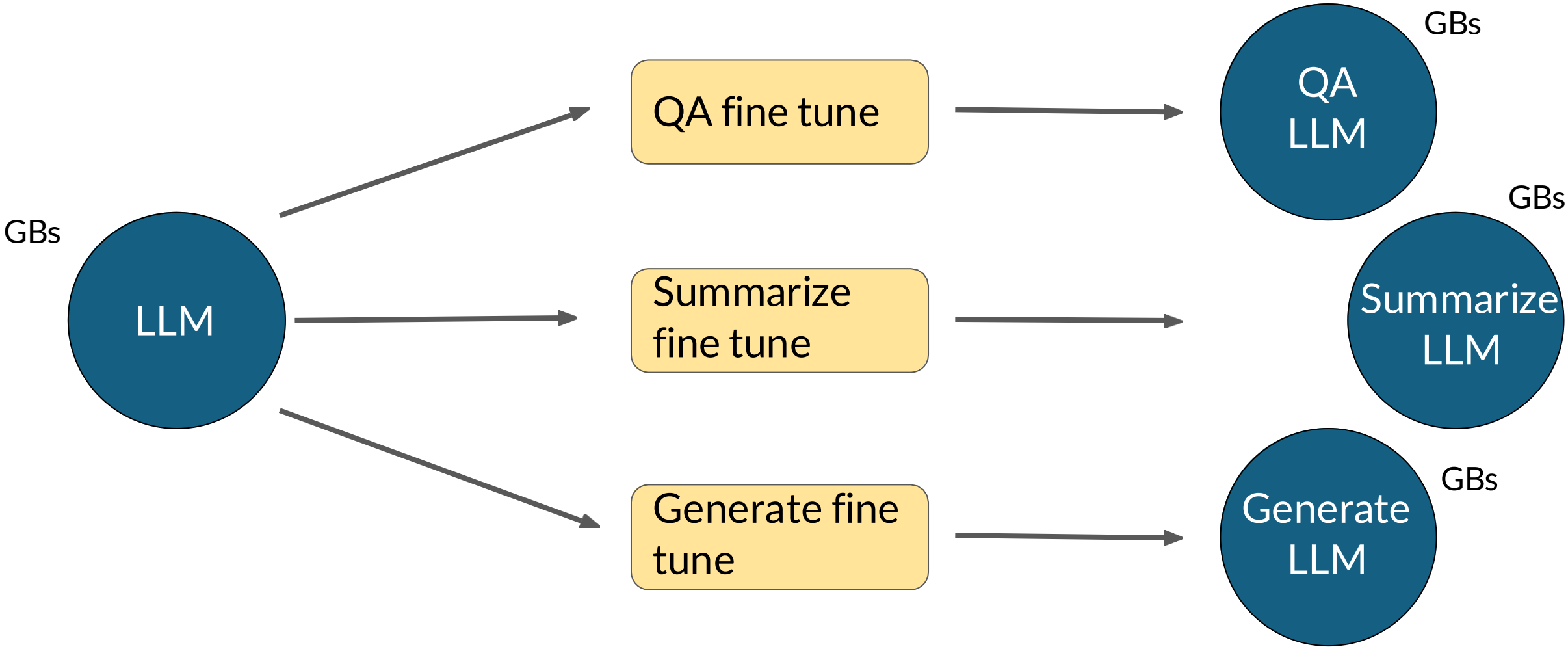

LLM with additional layers for PEFT



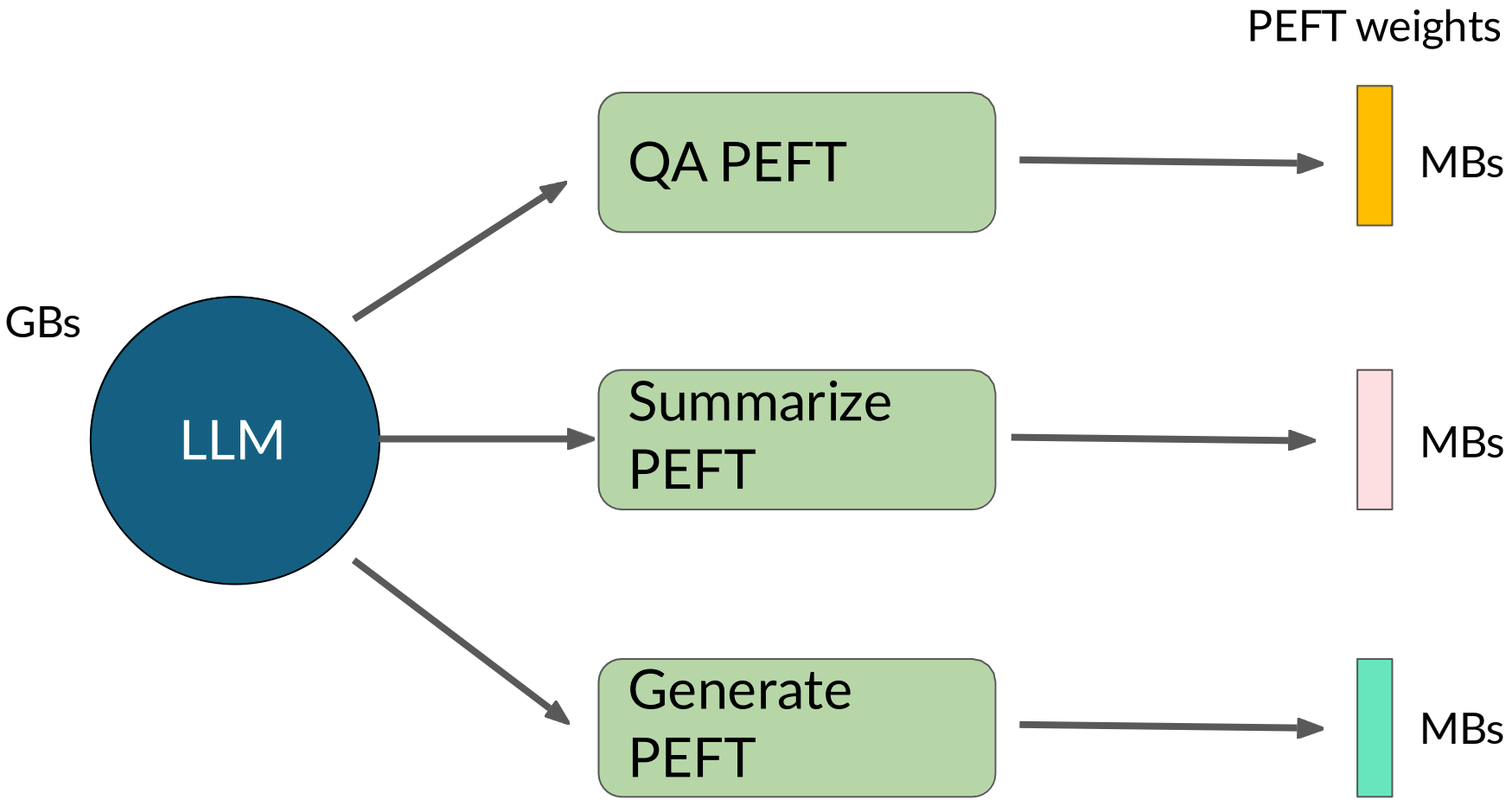
Less prone to catastrophic forgetting



# Full fine-tuning creates full copy of original LLM per task



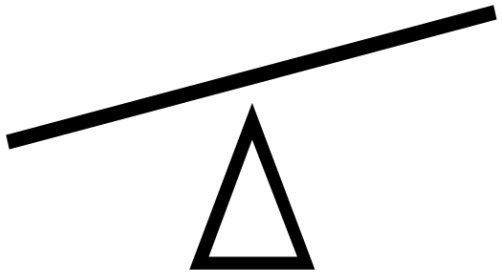
# PEFT fine-tuning saves space and is flexible



# PEFT Trade-offs

Parameter Efficiency

Memory Efficiency



Training Speed

Model Performance

Inference Costs

# Categories of PEFT methods

Source: Lialin et al. 2023, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning”,

# PEFT methods

Selective

**Select** subset of initial LLM parameters to fine-tune

Source: Lialin et al. 2023, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning”,

# PEFT methods

| Selective | Reparameterization |
| --- | --- |
| **Select** subset of initial LLM parameters to fine-tune | **Reparameterize** model weights using a low-rank representation<br><br>**LoRA** |

Source: Lialin et al. 2023, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning”,

# PEFT methods

| Selective | Reparameterization | Additive |
| --- | --- | --- |
| **Select** subset of initial LLM parameters to fine-tune | **Reparameterize** model weights using a low-rank representation<br><br>**LoRA** | **Add** trainable layers or parameters to model |

Source: Lialin et al. 2023, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning”,

# PEFT methods

| Selective | Reparameterization | Additive |
| --- | --- | --- |
| **Select** subset of initial LLM parameters to fine-tune | **Reparameterize** model weights using a low-rank representation<br><br>**LoRA** | **Add** trainable layers or parameters to model<br><br>Adapters |

Source: Lialin et al. 2023, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning”,

# PEFT methods

| Selective | Reparameterization | Additive |
| --- | --- | --- |
| **Select** subset of initial LLM parameters to fine-tune | **Reparameterize** model weights using a low-rank representation<br><br>**LoRA** | **Add** trainable layers or parameters to model<br>Adapters<br>Soft Prompts<br>**Prompt Tuning** |

Source: Lialin et al. 2023, "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning",

# II. Challenges

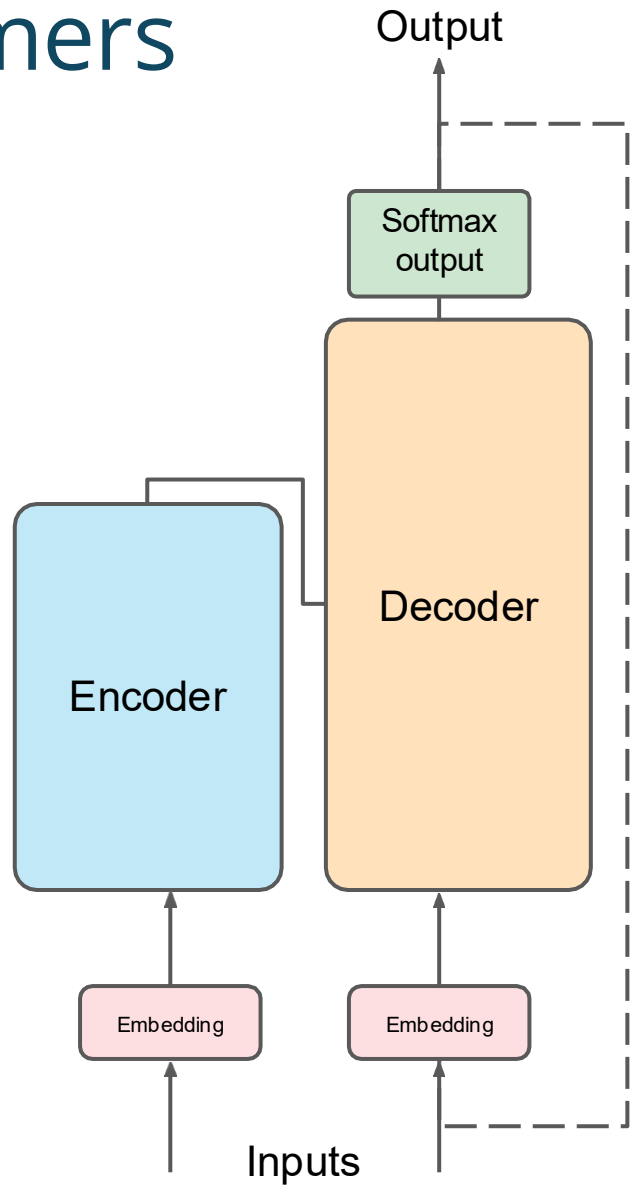
## 3. Adapt: Foundation Model

- Prompting & Prompt Engineering
- **Fine-tuning**
  - Instruction fine-tuning
  - Fine-tuning on a single task
  - Fine-tuning on multiple tasks
  - **Parameter efficient fine-tuning (PEFT)**
    - **LoRA**
    - Prompt tuning

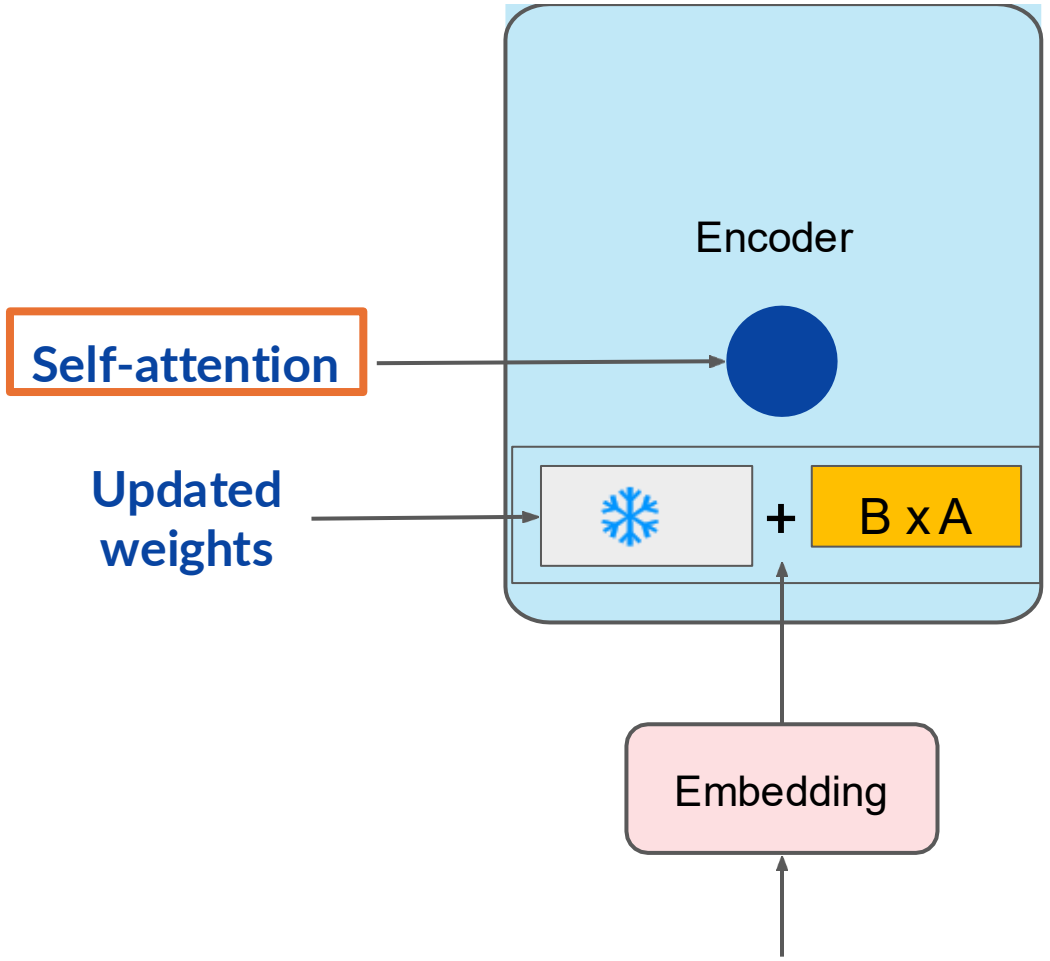
created with chatGPT

# Low-Rank Adaptation of Large Language Models (LoRA)

# Reminder: Transformers



# LoRA: Low Rank Adaption of LLMs

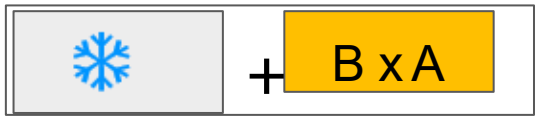



1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

Steps to update model for inference:

1. Matrix multiply the low rank matrices


2. Add to original weights

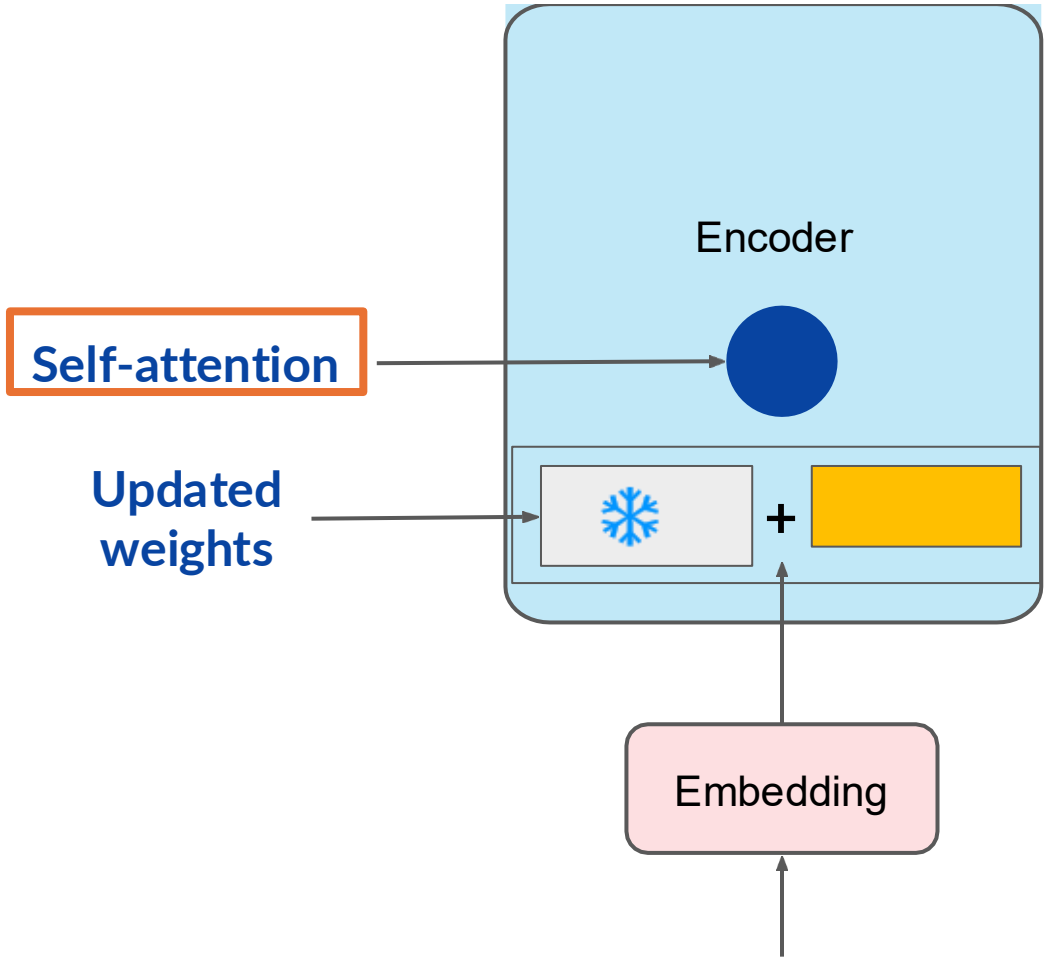


# Example: base Transformer as reference

- Use the base Transformer model by Vaswani et al. 2017:
  - Transformer weights have dimensions $d$ x $k$ = 512 x 64
  - So 512 x 64 = 32,768 trainable parameters

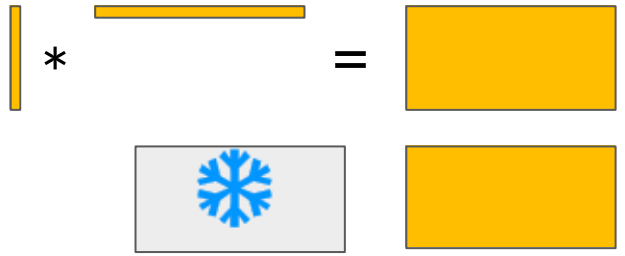
- In LoRA with rank $r$ = 8:
  - $A$ has dimensions $r$ x $k$ = 8 x 64 = 512 parameters
  - B has dimension $d$ x $r$ = 512 x 8 = 4,096 trainable parameters
  - **86% reduction in parameters to train!**

# LoRA: Low Rank Adaption of LLMs

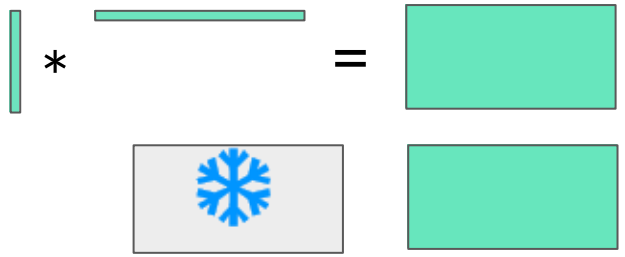



1. Train different rank decomposition matrices for different tasks
2. Update weights before inference

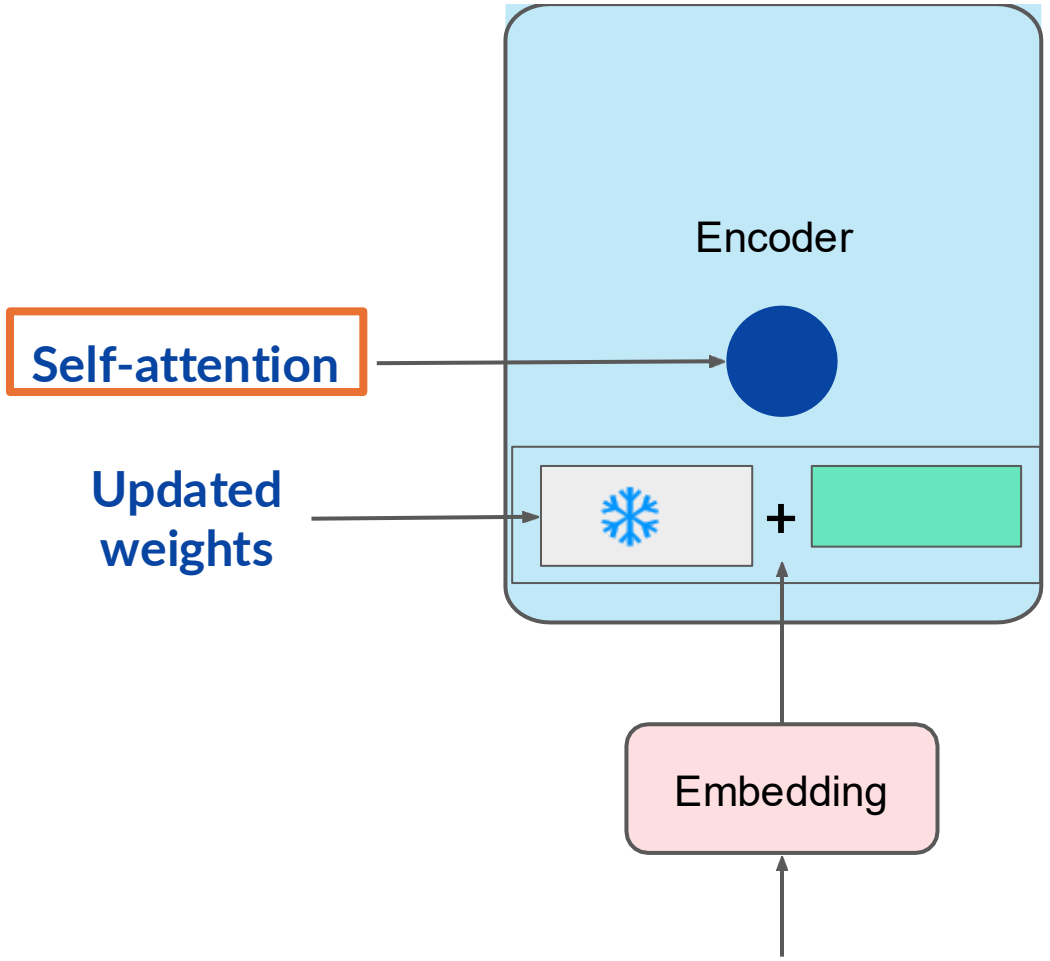
Task A




Task B

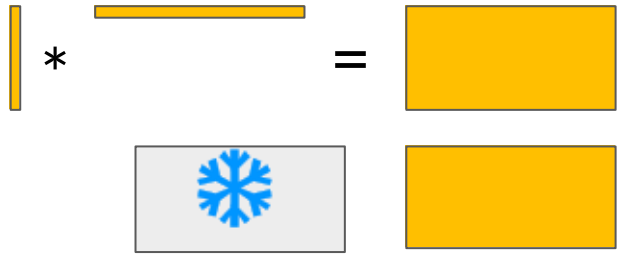


# LoRA: Low Rank Adaption of LLMs

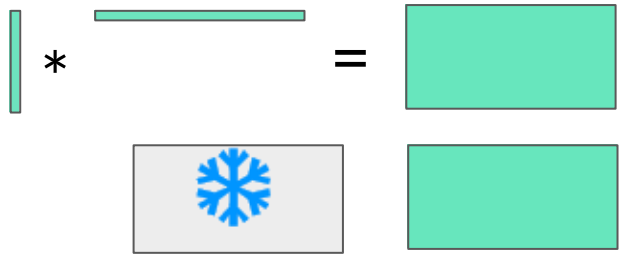



1. Train different rank decomposition matrices for different tasks
2. Update weights before inference

Task A




Task B



# Choosing the LoRA rank

| Rank $r$ | val_loss | BLEU | NIST | METEOR | ROUGE_L | CIDEr |
|---|---|---|---|---|---|---|
| 1 | 1.23 | 68.72 | 8.7215 | 0.4565 | 0.7052 | 2.4329 |
| 2 | 1.21 | 69.17 | 8.7413 | 0.4590 | 0.7052 | 2.4639 |
| 4 | 1.18 | **70.38** | **8.8439** | **0.4689** | 0.7186 | **2.5349** |
| 8 | 1.17 | 69.57 | 8.7457 | 0.4636 | **0.7196** | 2.5196 |
| 16 | **1.16** | 69.61 | 8.7483 | 0.4629 | 0.7177 | 2.4985 |
| 32 | **1.16** | 69.33 | 8.7736 | 0.4642 | 0.7105 | 2.5255 |
| 64 | **1.16** | 69.24 | 8.7174 | 0.4651 | 0.7180 | 2.5070 |
| 128 | **1.16** | 68.73 | 8.6718 | 0.4628 | 0.7127 | 2.5030 |
| 256 | **1.16** | 68.92 | 8.6982 | 0.4629 | 0.7128 | 2.5012 |
| 512 | **1.16** | 68.78 | 8.6857 | 0.4637 | 0.7128 | 2.5025 |
| 1024 | 1.17 | 69.37 | 8.7495 | 0.4659 | 0.7149 | 2.5090 |

- Effectiveness of higher rank appears to plateau
- Relationship between rank and dataset size needs more empirical data

Source: Hu et al. 2021, "LoRA: Low-Rank Adaptation of Large Language Models"

# II. Challenges

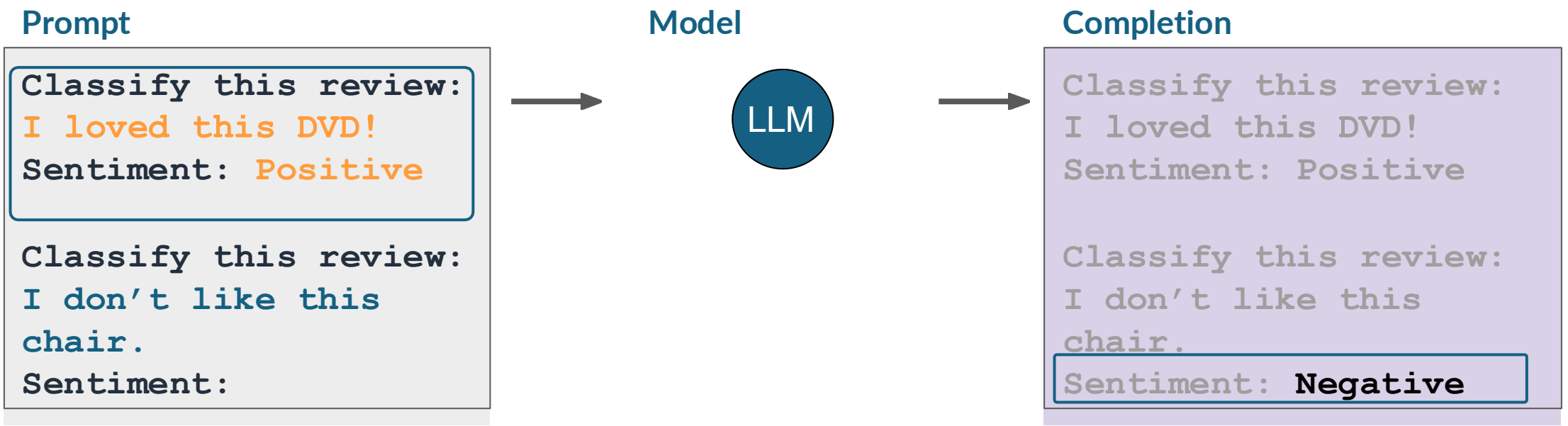
## 3. Adapt: Foundation Model

- Prompting & Prompt Engineering
- **Fine-tuning**
  - Instruction fine-tuning
  - Fine-tuning on a single task
  - Fine-tuning on multiple tasks
  - **Parameter efficient fine-tuning (PEFT)**
    - LoRA
    - **Prompt tuning**

created with chatGPT

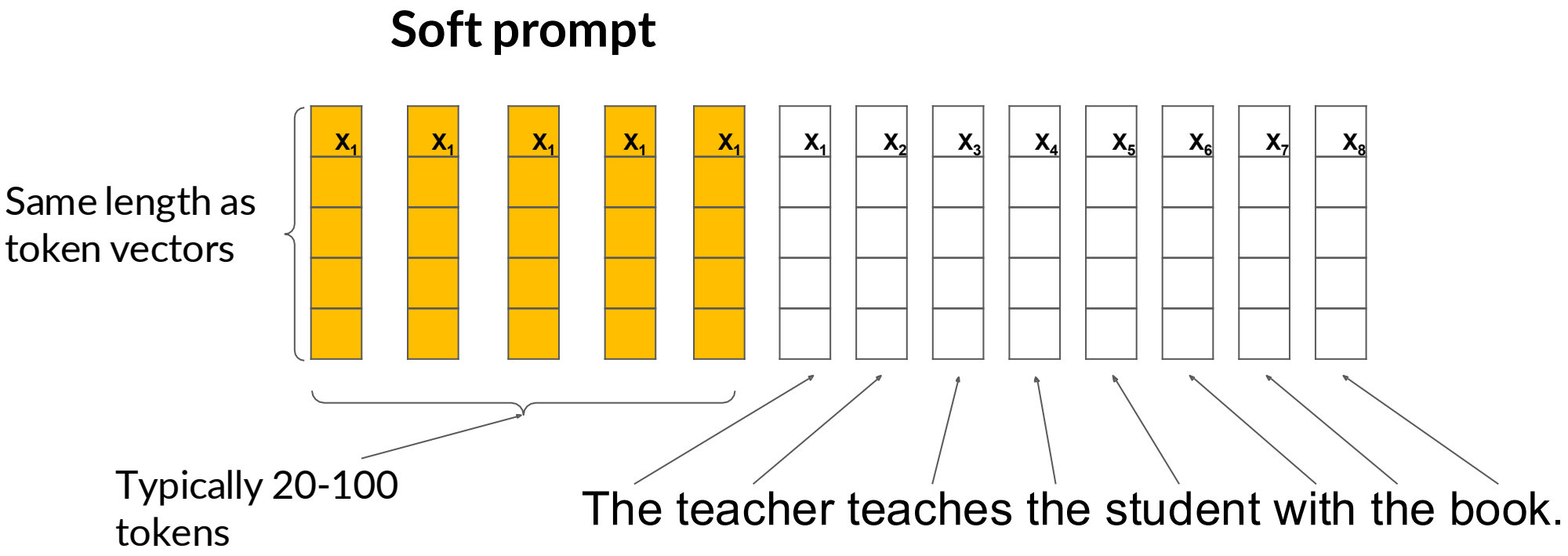
# Prompt Tuning with soft prompts (not prompt engineering)

# Prompt tuning is **not** prompt engineering!

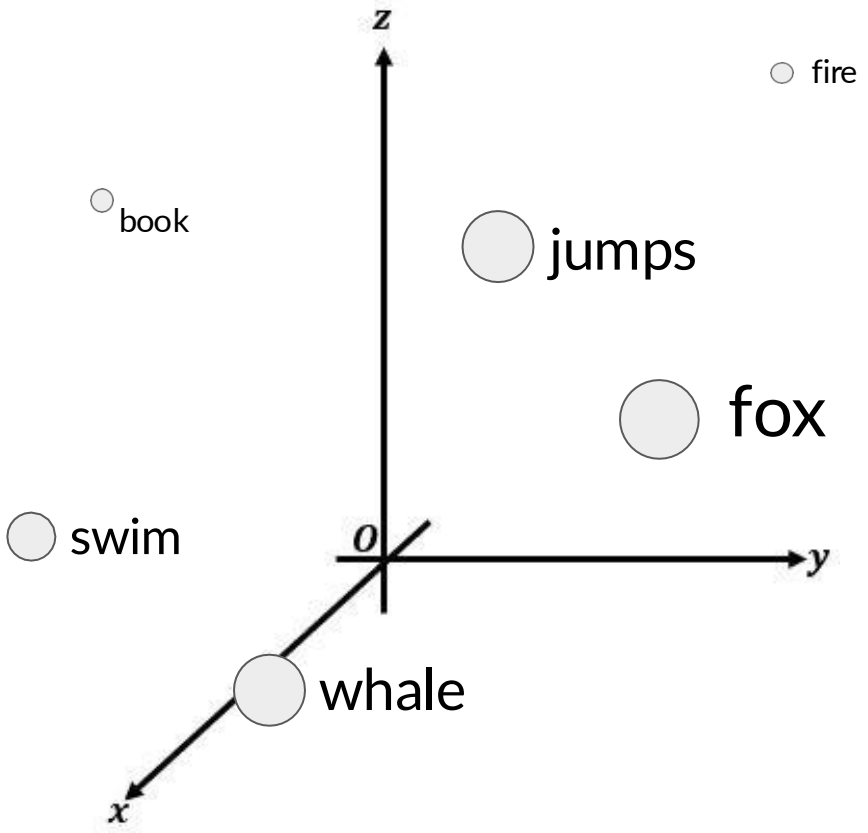



One-shot or Few-shot Inference

# Prompt tuning adds trainable "soft prompt" to inputs



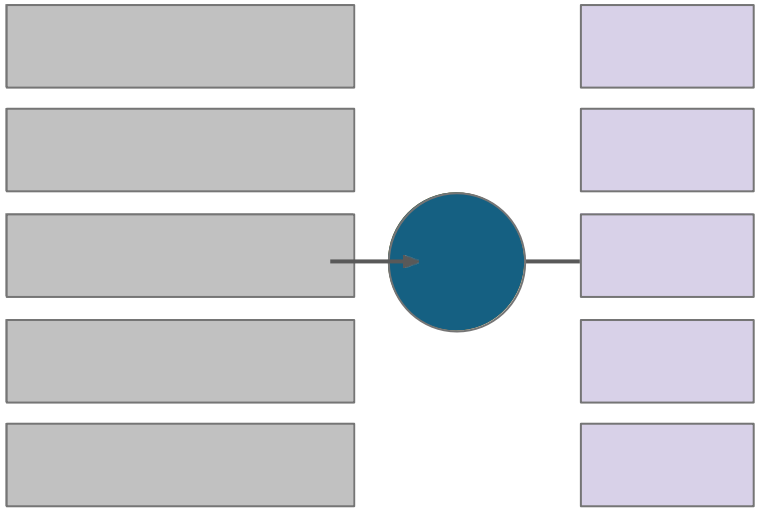
# Soft prompts




Embeddings of each token exist at unique point in multi-dimensional space

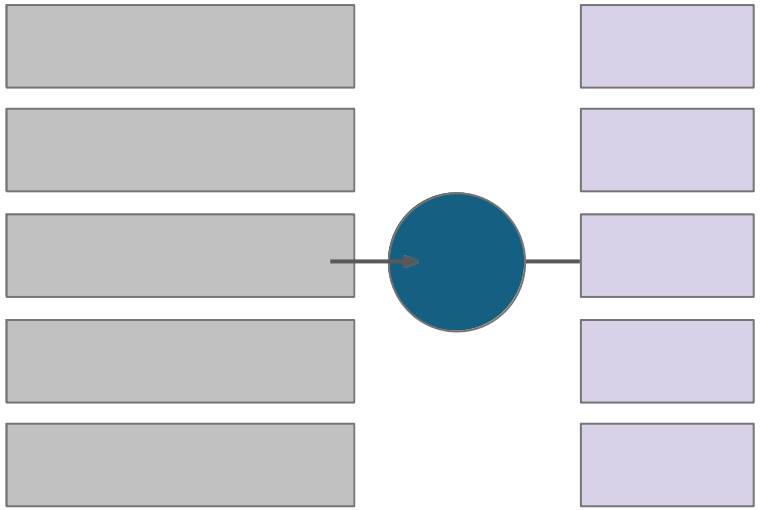
# Full Fine-tuning vs prompt tuning

Weights of model updated during training



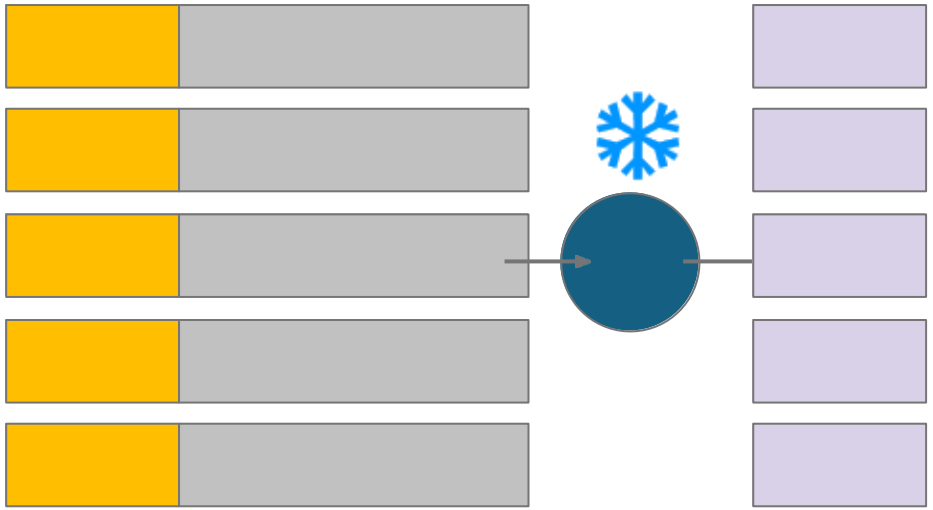
# Full Fine-tuning vs prompt tuning

Weights of model updated during training



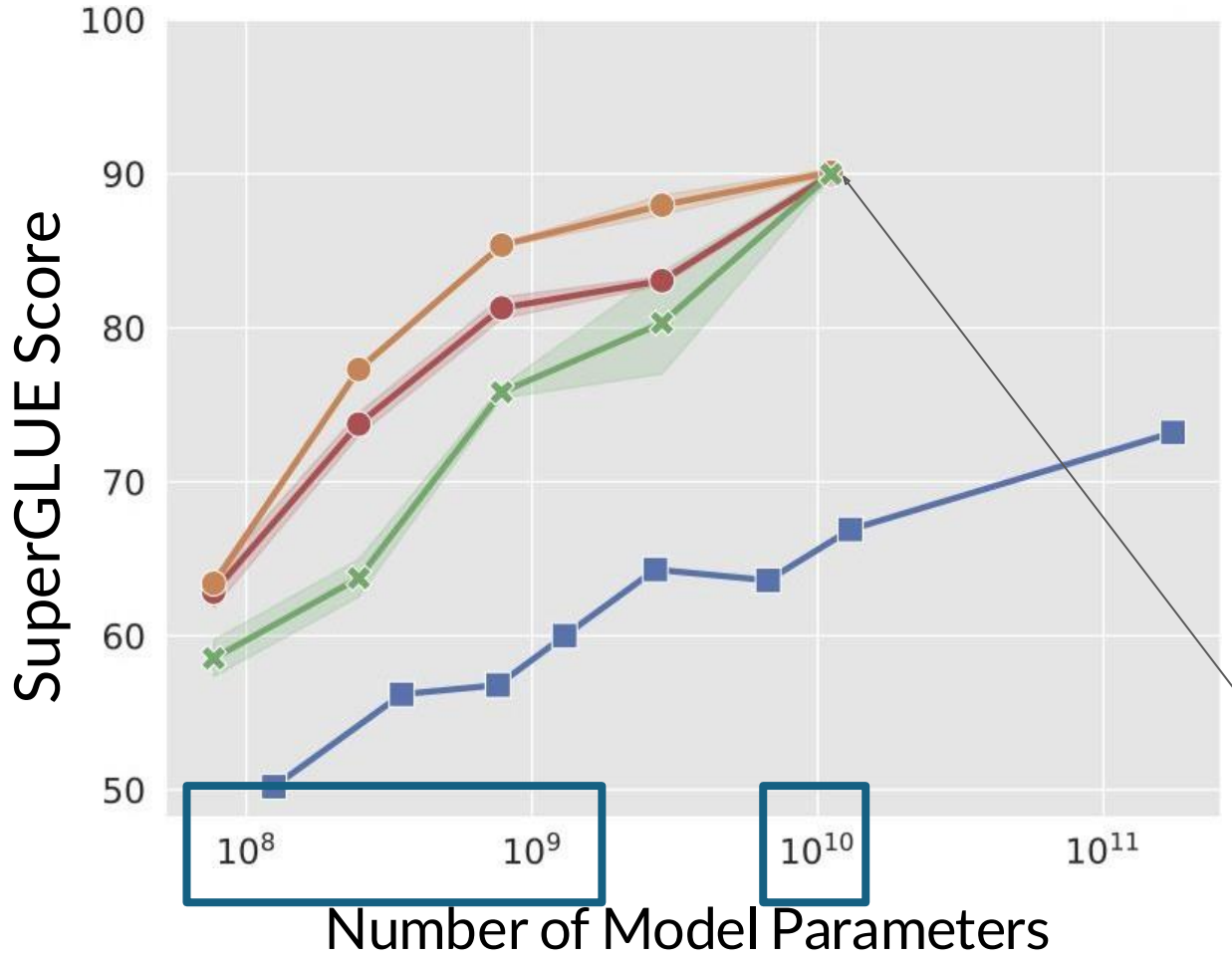
Millions to Billions of parameter updated

Weights of model frozen and soft prompt trained



10K - 100K of parameters updated

# Performance of prompt tuning

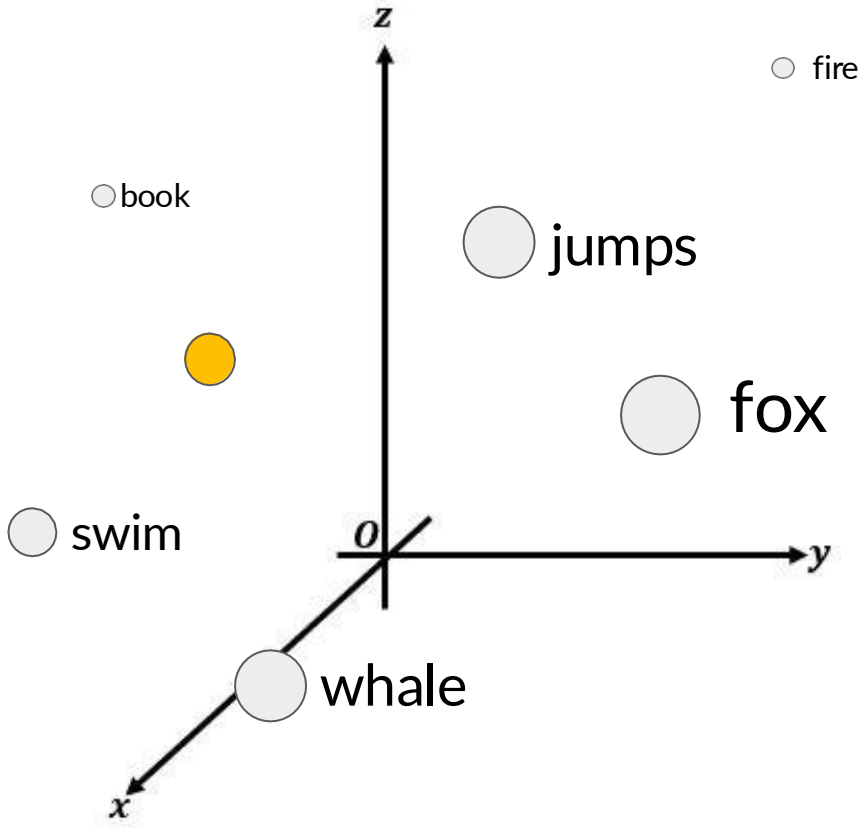



- Full Fine-tuning
- Multi-task Fine-tuning
- Prompt tuning
- Prompt engineering

Prompt tuning can be as effective as full Fine-tuning for larger models!

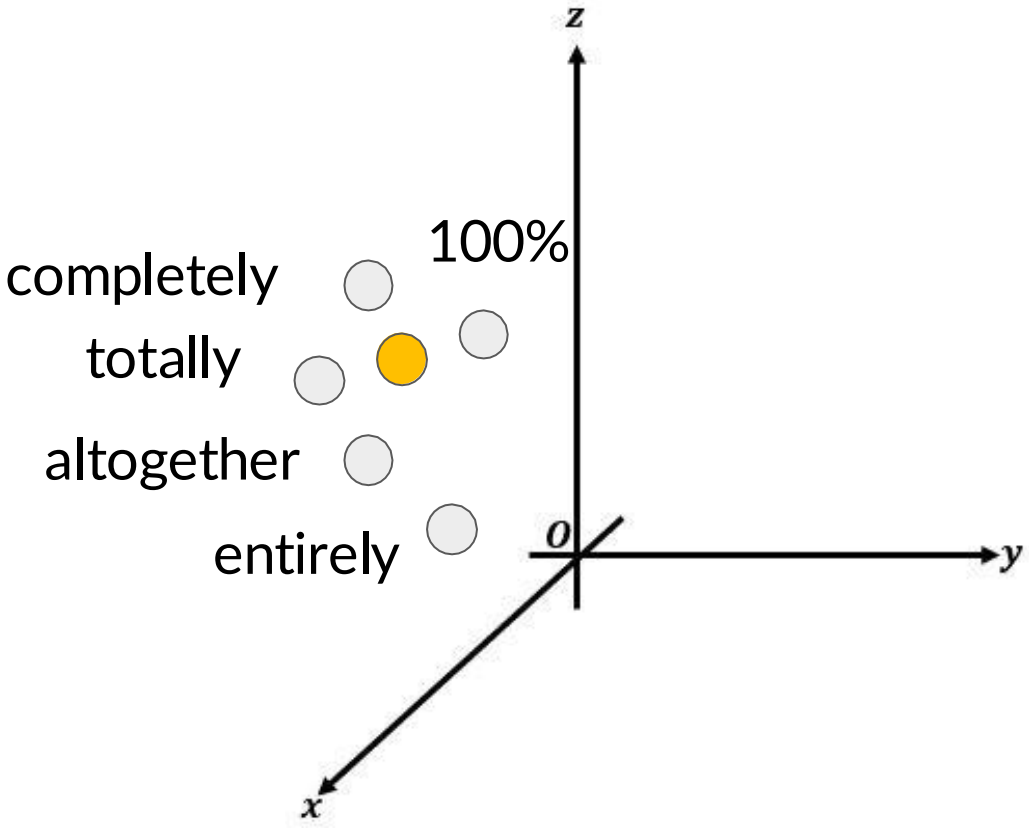
Source: Lester et al. 2021, "The Power of Scale for Parameter-Efficient Prompt Tuning"

# Interpretability of soft prompts




Trained soft-prompt embedding does not correspond to a known token…

# Interpretability of soft prompts




...but nearest neighbors form a semantic group with similar meanings.

# II. Challenges

## Summary
## 3. Adapt: Foundation Model

- Prompting & Prompt Engineering
- Fine-tuning
  - Instruction fine-tuning
  - Fine-tuning on a single task
  - Fine-tuning on multiple tasks
  - Parameter efficient fine-tuning (PEFT)

created with chatGPT

# Summary: Prompt engineering with In-context learning (ICL)

**Prompt // Zero Shot**

```
Classify this review:
I loved this movie!
Sentiment:
```

**Prompt // One Shot**

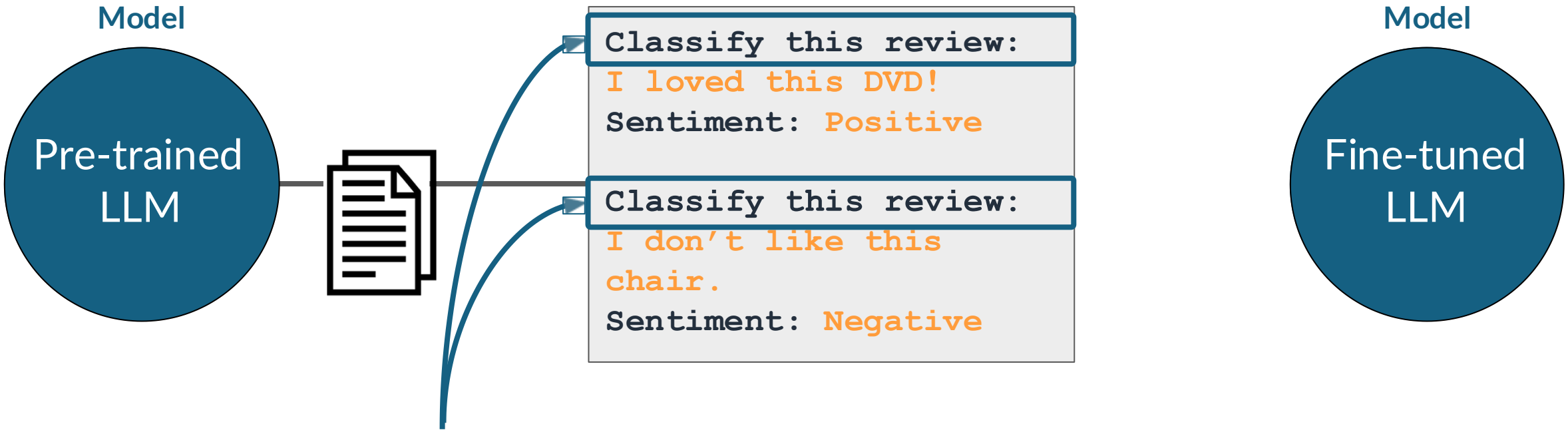
```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment:
```

**Prompt // Few Shot** >5 or 6 examples

```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment: Negative

Classify this review:
Who would use this
product?
Sentiment:
```

**Context Window**
(few thousand words)

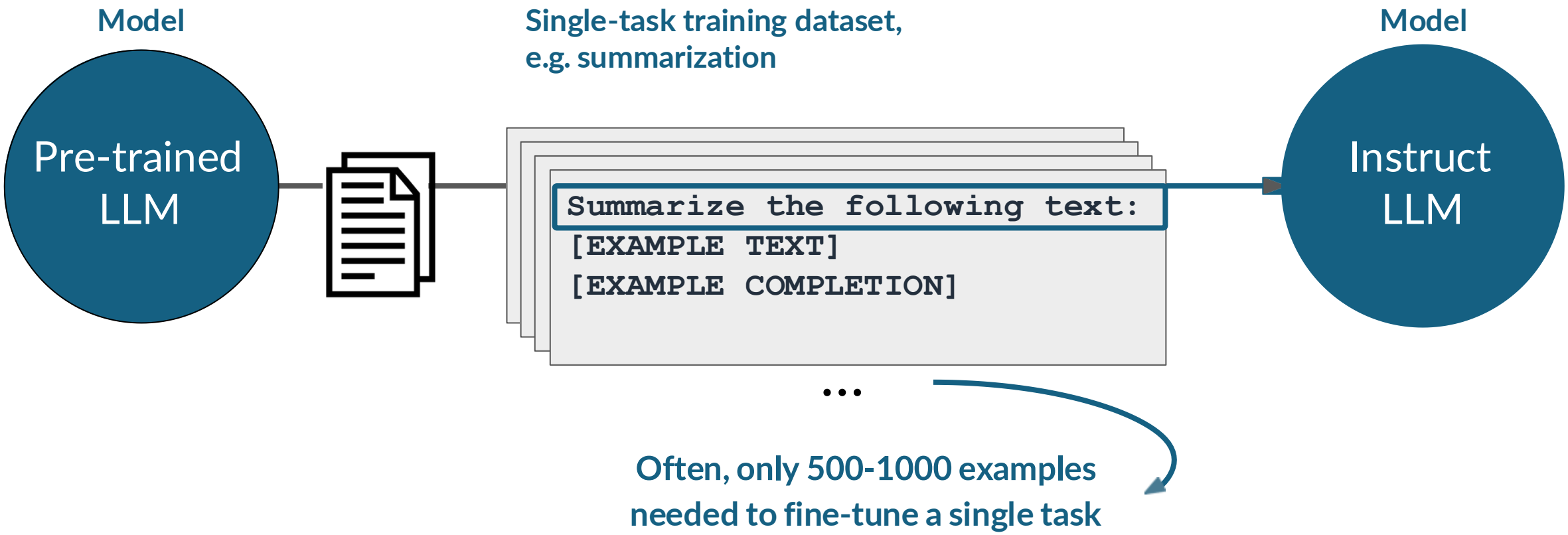
# Summary: fine-tune LLMs w/ instructions




**Each prompt/completion pair includes a specific "instruction" to the LLM**

**LLM fine-tuning**

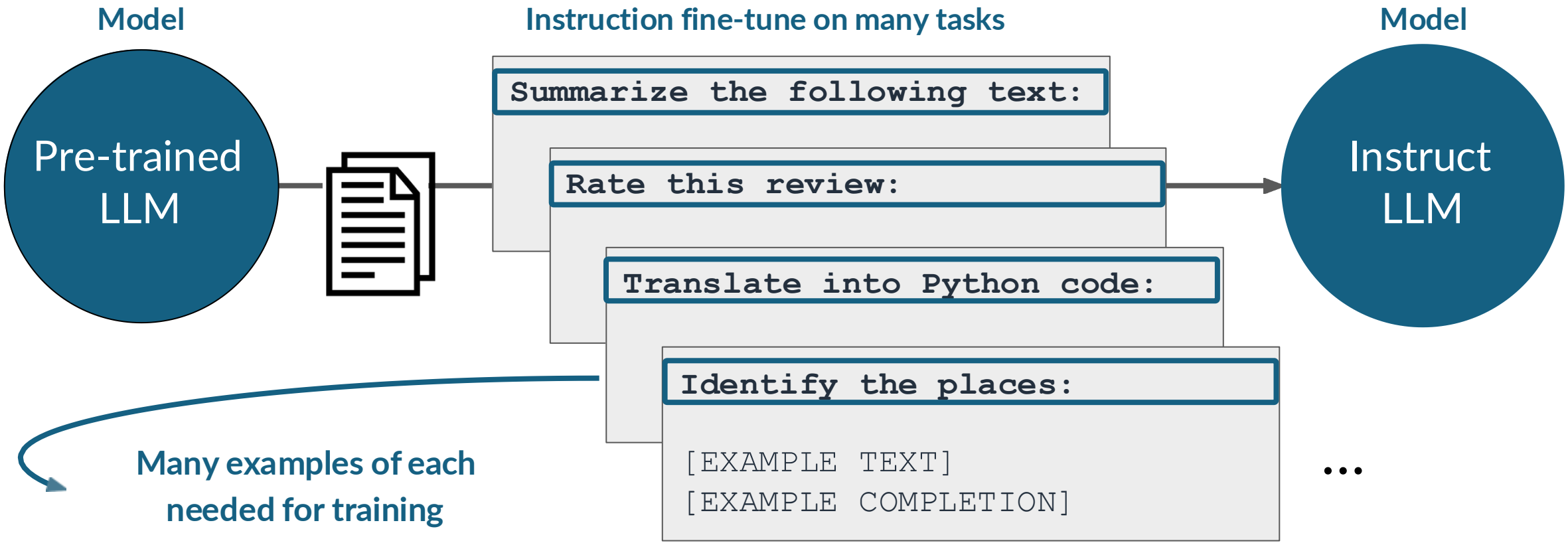
# Summary: Fine-tuning on a single task



# How to avoid catastrophic forgetting

- First note that you might not have to!
- Fine-tune on **multiple tasks** at the same time
- Consider **Parameter Efficient Fine-tuning** (PEFT)

# Summary: Multi-task, instruction fine-tuning



# Summary: PEFT methods

| Selective | Reparameterization | Additive |
| --- | --- | --- |
| **Select** subset of initial LLM parameters to fine-tune | **Reparameterize** model weights using a low-rank representation<br><br>**LoRA** | **Add** trainable layers or parameters to model<br>Adapters<br>Soft Prompts<br>**Prompt Tuning** |

Source: Lialin et al. 2023, "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning",

# Thank you

Slides adapted from various sources: [Intro to Large Language Models, Andrej Karpathy, Executive Education Polytechnique, Udemy, Deeplearning.ai, Stanford University CS231n, Financial Times, New York Times, Hi!Paris summer school 2023]