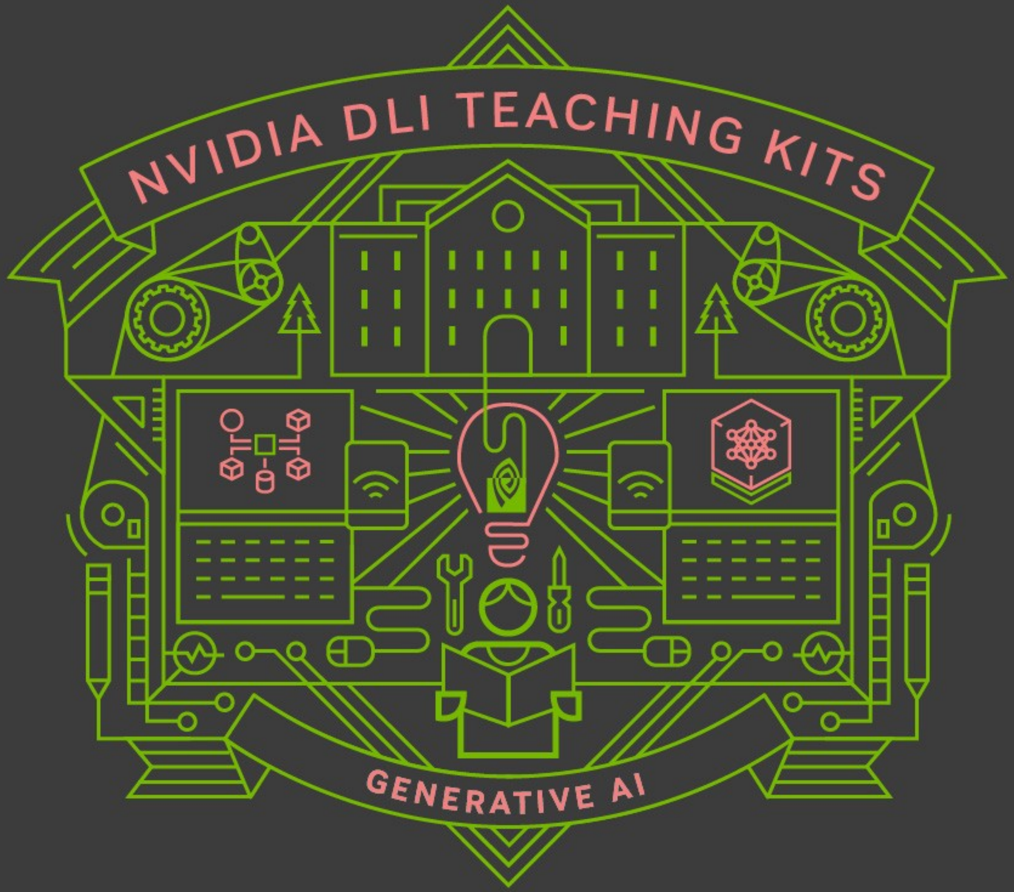
Lecture 7.1 - Pre-training, continued pre-training, and task training

Generative AI Teaching Kit



This lecture

- Autoregressive Training
- Pre-training LLMs
- Instruction Fine-tuning LLMs
- Continued Pre-training LLMs

Autoregressive Training

Making use of unlabeled data and learning to speak

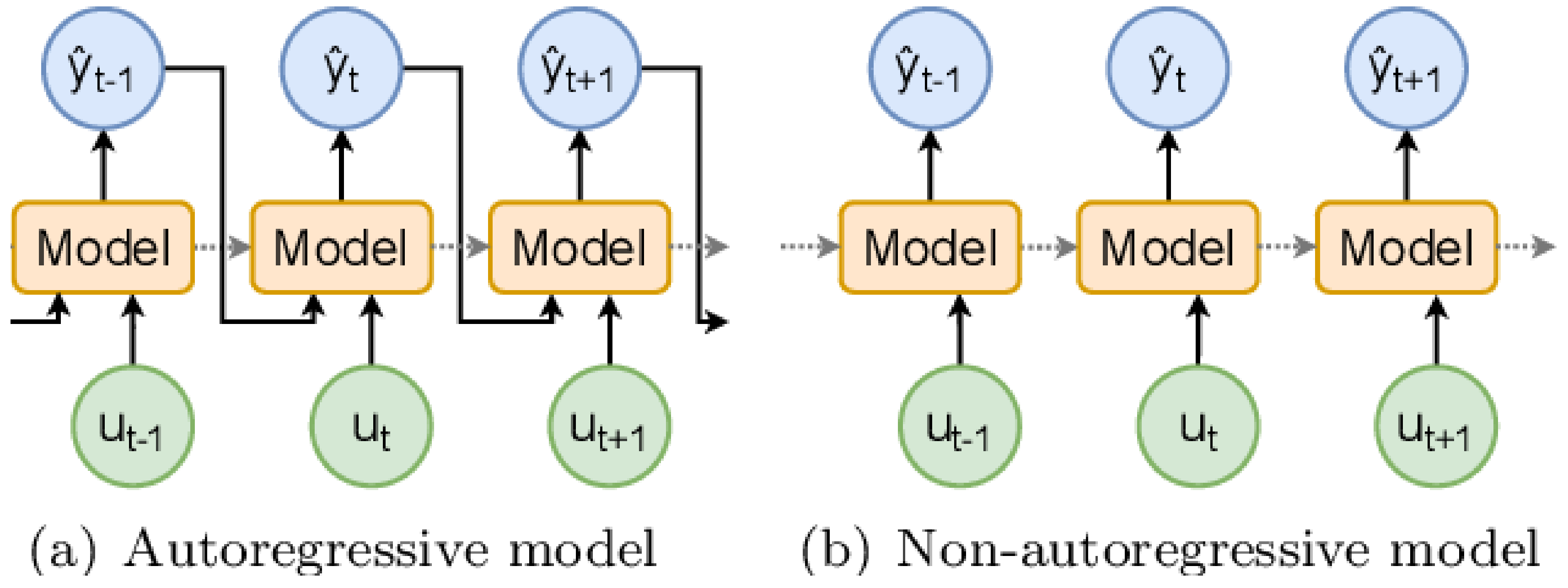
What are Autoregressive models?

In an autoregressive model, each value in a sequence is predicted based on the prior values.

Each value is predicted conditionally based on the previous values.

Autoregressive models are particularly powerful for sequential data and are fundamental to large language models (LLMs) like GPT (Generative Pre-trained Transformer), which predict tokens in a sequence of text.

The Autoregressive nature of LLMs explains a number of key advantages and challenges



Predicting the next token

LLMs, such as GPT generate text token by token, where each token prediction depends on all previously generated tokens.

For example, given the sequence:

“The cat is …”

The model computes a probability distribution over the possible next tokens (e.g., "sleeping," "jumping," etc.). Each token prediction is made by conditioning on the previous tokens in the sequence.

This prediction is probabilistic. The model does not simply predict the next token deterministically. Instead, it computes a probability distribution over all possible next tokens
The next token is then sampled from this probability distribution, making the process both sequential and probabilistic.

$$P(blue|The, cat, is) = .1$$
$$P(black|The, cat, is) = .85$$
$$P(green|The, cat, is) = .05$$

$$f(\texttt{<start>}) = \begin{pmatrix} \mathbf{P(The) = .5} \\ P(cat) = .1 \\ P(is) = .2 \\ P(blue) = .05 \\ P(black) = .1 \\ P(green) = .04 \\ P(\texttt{<end>}) = .01 \end{pmatrix}$$

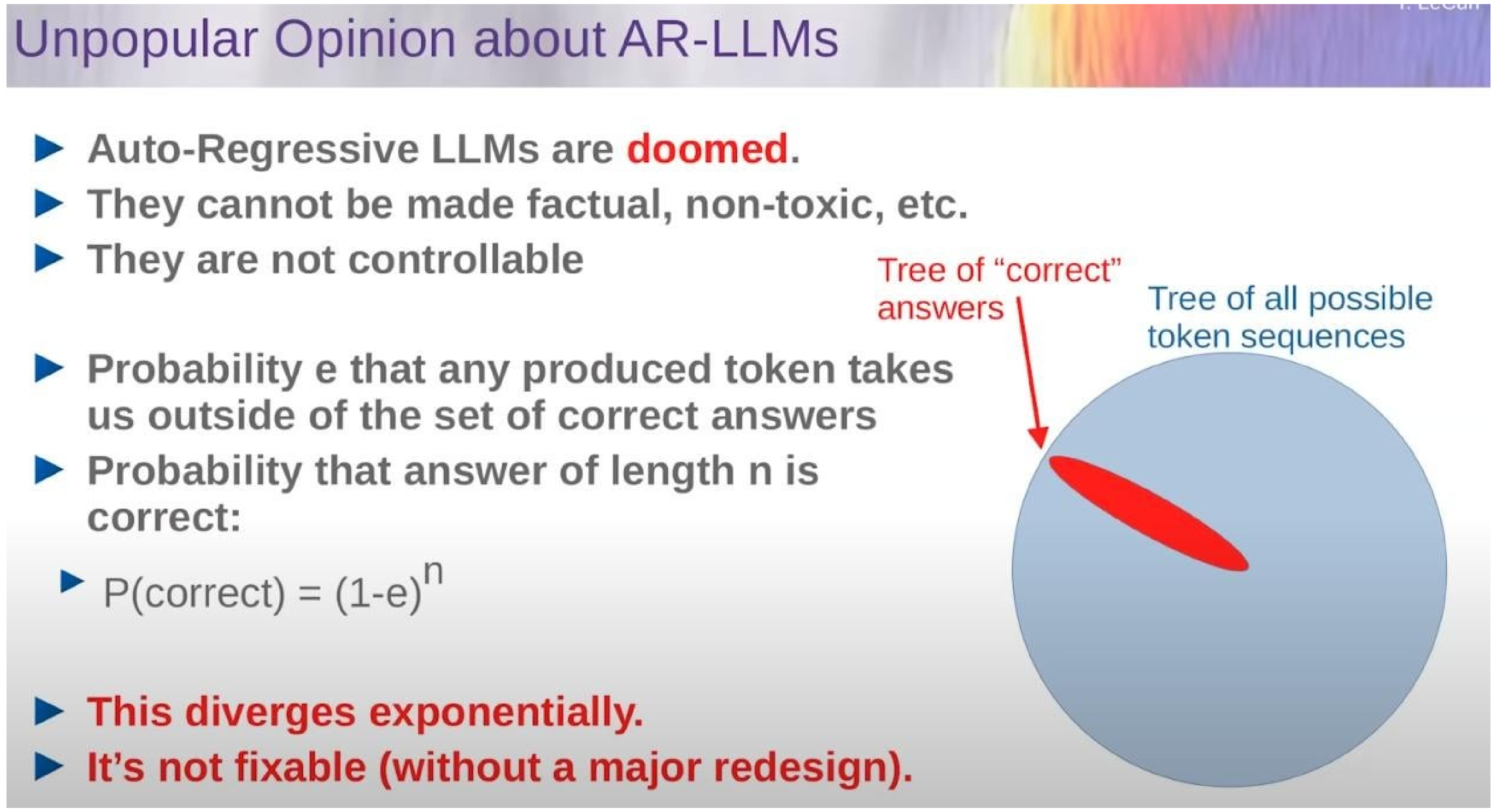
Autoregression and Hallucination

Hallucination:
“LLMs generating text that sounds plausible but is factually incorrect or nonsensical.”

Relation to Autoregression:

Token-by-token generation: Each token depends on previous ones. A small error can propagate, leading to hallucinations.

Probabilistic predictions: Models sample from a probability distribution, which can lead to selecting less likely tokens, increasing hallucination risk.



Unpopular Opinion about AR-LLMs

- Auto-Regressive LLMs are **doomed**.
- They cannot be made factual, non-toxic, etc.
- They are not controllable
- Probability e that any produced token takes us outside of the set of correct answers
- Probability that answer of length n is correct:
  - P(correct) = $(1-e)^n$
- **This diverges exponentially.**
- **It’s not fixable (without a major redesign).**

Tree of “correct” answers
Tree of all possible token sequences

Training Autoregressive Models for Language Modeling

Data Preparation for Autoregressive Training:

The data preparation step is crucial because the model needs to learn to predict the next token based on previous tokens in a sequence. Here's how it works:

Sequence Construction:

The tokenized text is split into sequences of fixed lengths, often referred to as the context window. For instance, if the context window is 512 tokens, the text is chunked into sequences of 512 tokens.

Creating Input-Output Pairs:

The key to autoregressive training is predicting the next token. For each sequence, we create input-output pairs where:

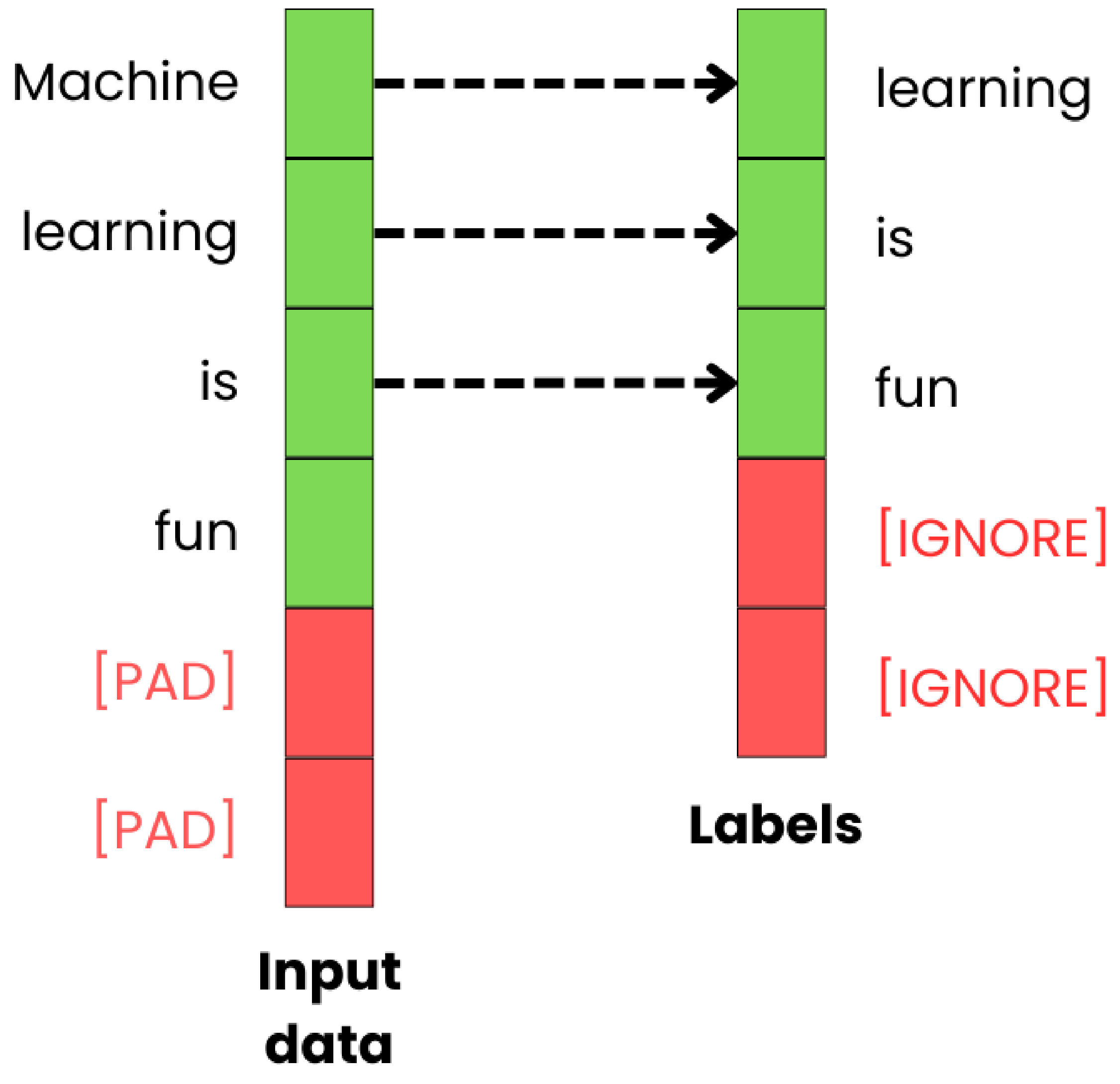
Input: The first N tokens.

Output: The N+1st token. This is a shifted copy of the sequence:

Input: [125, 894, 45, 204, 10] → Output: [894, 45, 204, 10, 356] The input is what the model sees, and the output is what the model is trained to predict.

Batching:

These input-output pairs are then grouped into batches for efficient training. The model processes multiple sequences in parallel, where each batch contains a set of input-output pairs.



Pre-training

Creating our Language Model from scratch

Training a deep learning model

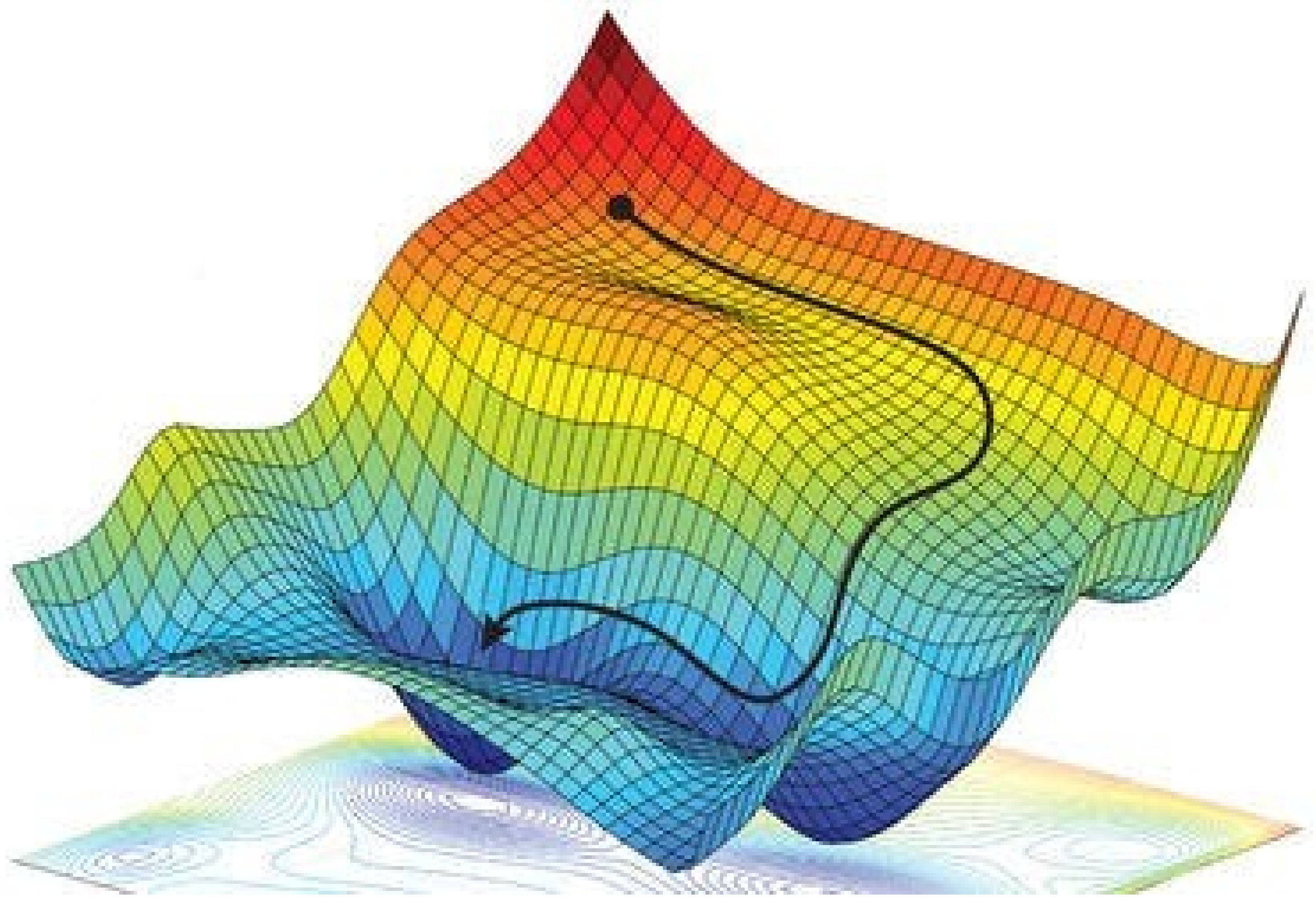
Creating a well performing deep learning model is all about the loss.

This “loss” is the difference between the predicted value output of the model, and some true/known value that comprises the target of our training data.

The model is trained by varying all the weights such that this loss is minimized as we traverse the loss landscape.

Various gradient decent methods, most of them based on Stochastic Gradient Descent are available, but they largely depend on 1) the current weight value, and 2) the gradient of the weights with respect to the loss.

We continue to vary the weights in training until this loss reaches a state where we decide to stop further training.


Loss landscape

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

A simple gradient descent update

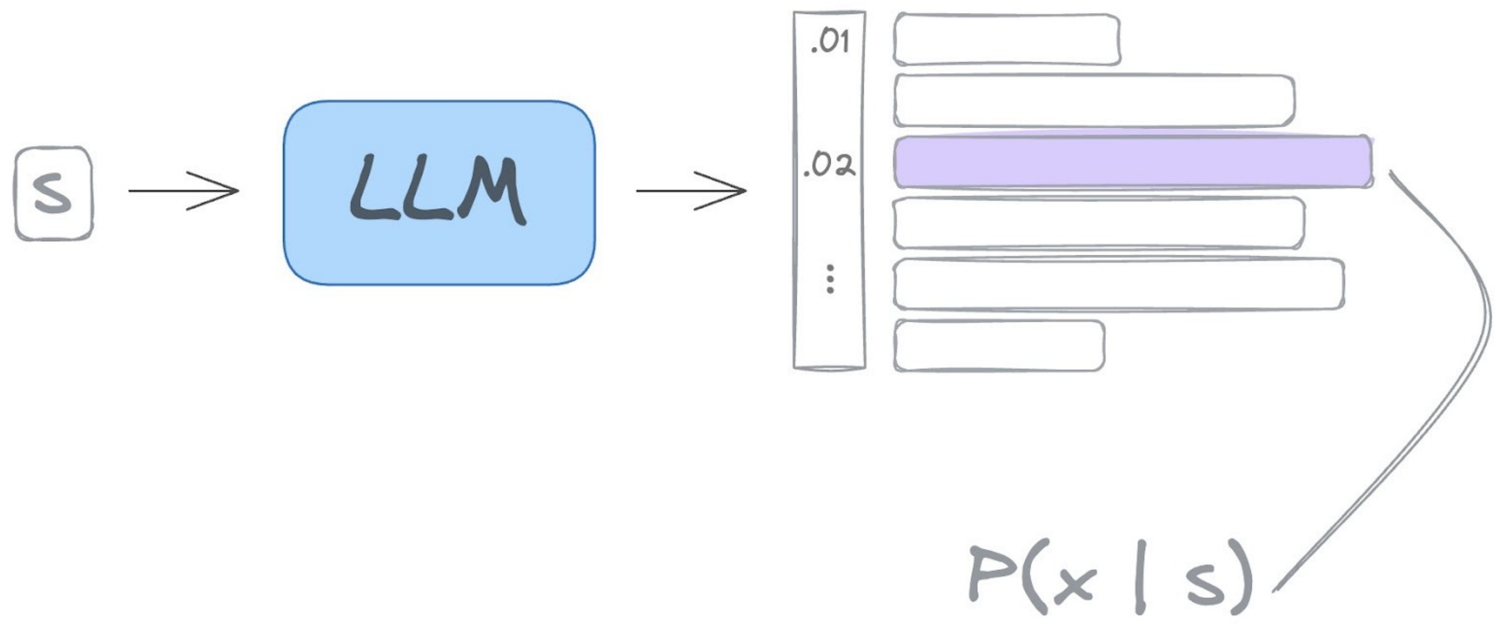
Language Model Training

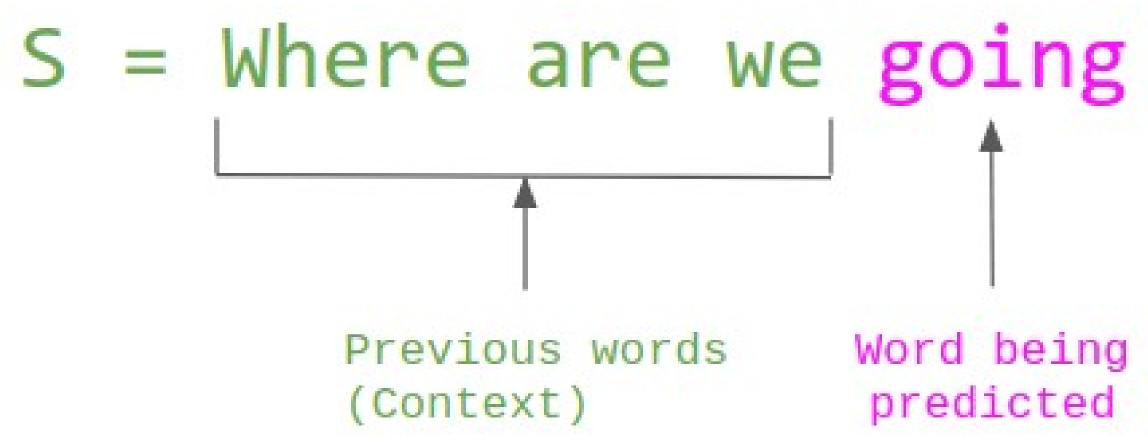
Language Models are those which attempt to predict the missing token of a given input.

For Decoder-type models, this is **the next token**.

For Encoder-type models, this can be some **intermediate token that has been masked**.

Either way, Language Models use the vocabulary they have, to select the **right token**.





Preparing an LLM for Pre-training

To get a model ready for pre-training, we need to set up the following:

1. The training data must be collected and curated
2. The data must be in the right format and ready to be loaded in batches
3. The model architecture must be constructed and loaded into memory
4. All algorithm configurations set to appropriate values
5. The compute infrastructure established
6. Training strategy configured to ensure the hardware and data are being most effectively utilized

Let’s look at these one-by-one

1 - The training data must be collected and curated

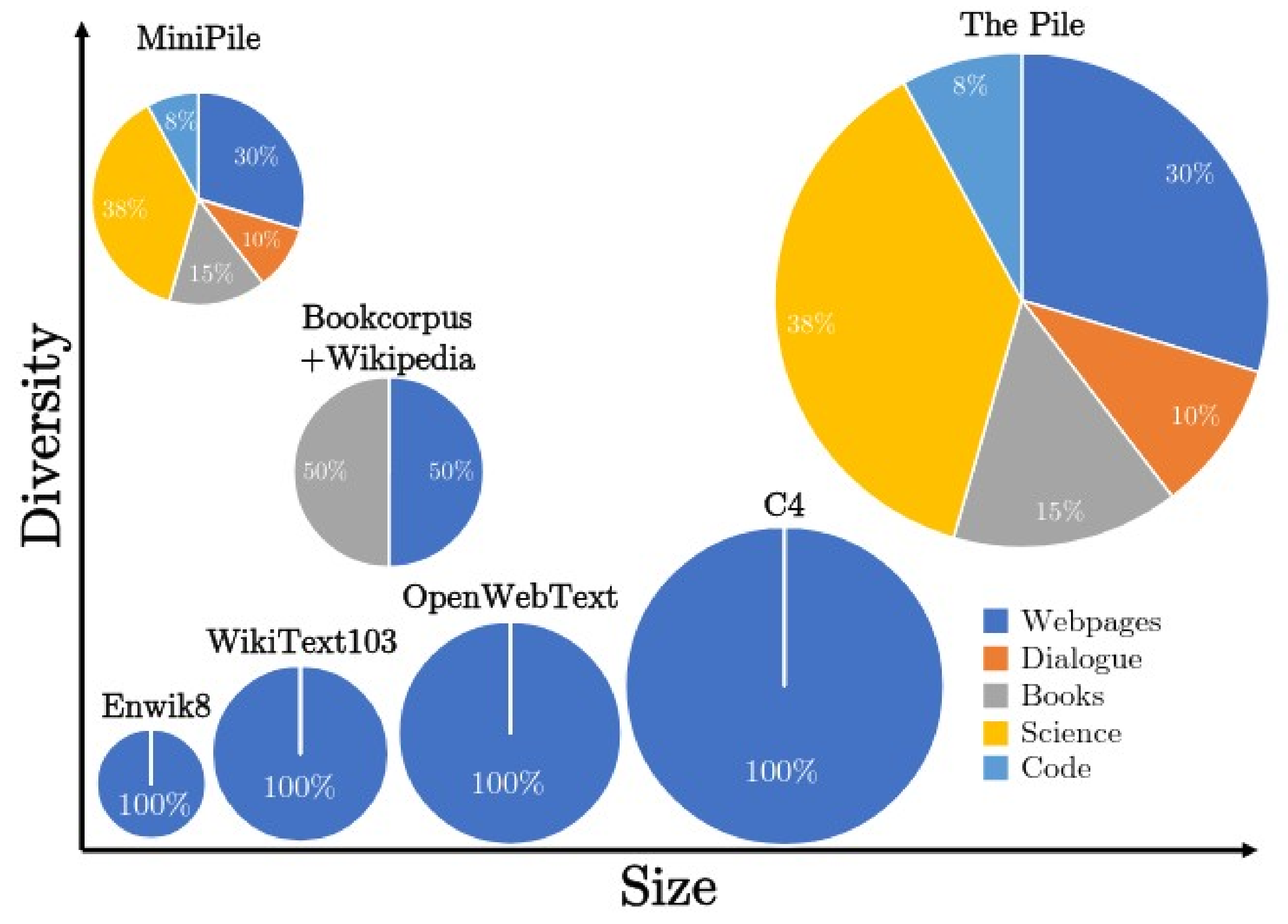
By crawling the open web, we can create massive datasets of all of the digital knowledge stored online.

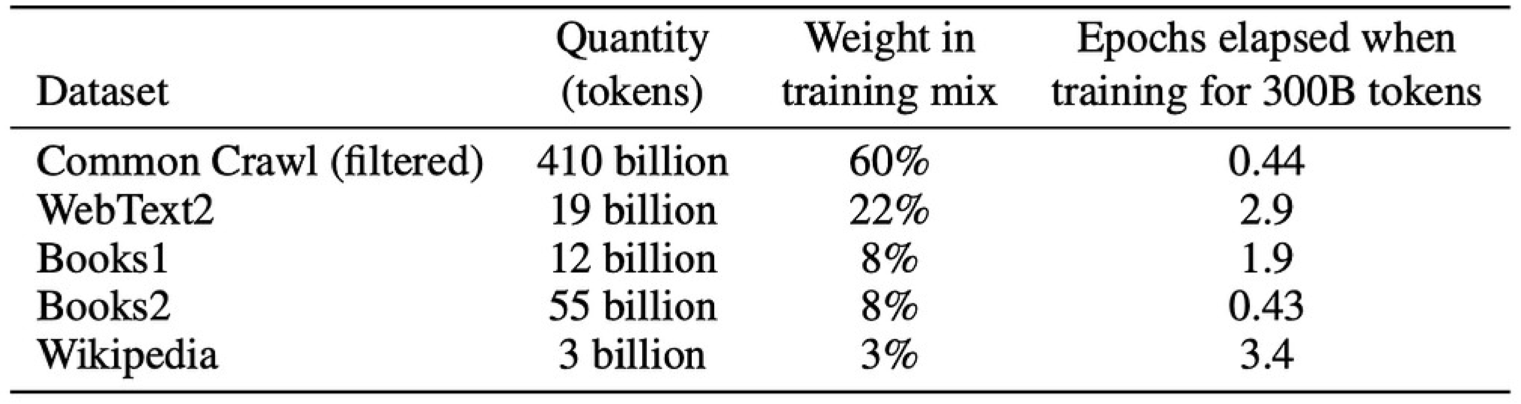
The issues arise when finding what site to use and what not to use.

The CommonCrawl project was started over a decade ago to find all addressable links.

Wikipedia also contains a huge repository of human language information

Getting the best data, though, is still an art rather than a science as the terabytes of raw text are impossible to manually clean and curate.





Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

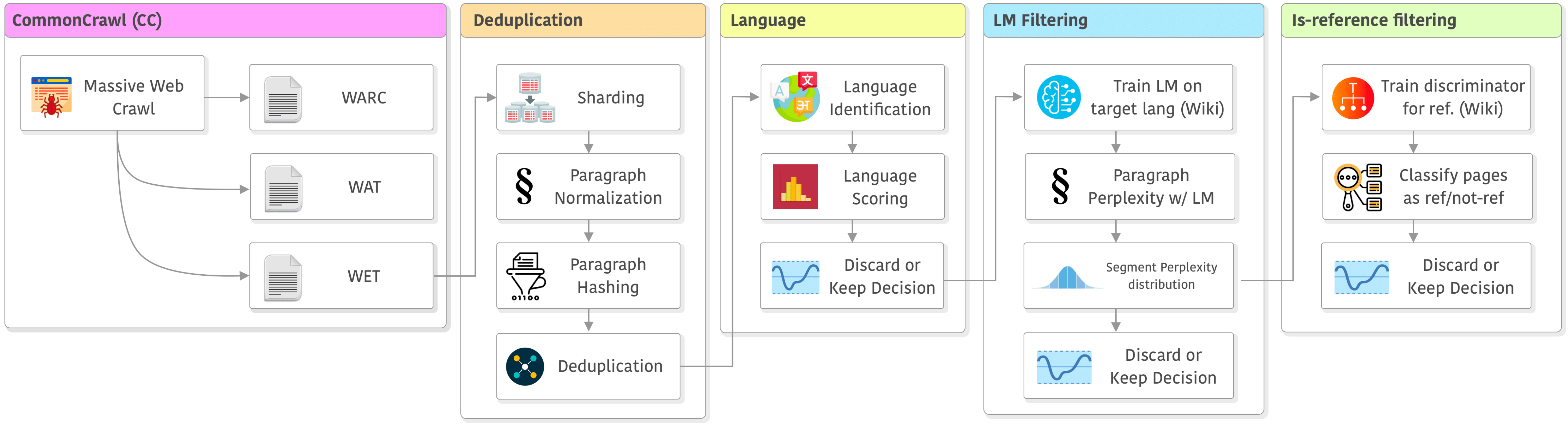
Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

2 – Data Formatting for Training

Once collected, the data must be processed before being useful for training.

A typical workflow consists of:

- Gather the raw data
- Removing duplicated data (deduplication)
- Filtering for language (e.g. only English)
- Filtering for harmful or specific content
- Remove unknown tokens
- Format data into training and testing datasets



3 – Selecting Model Architecture

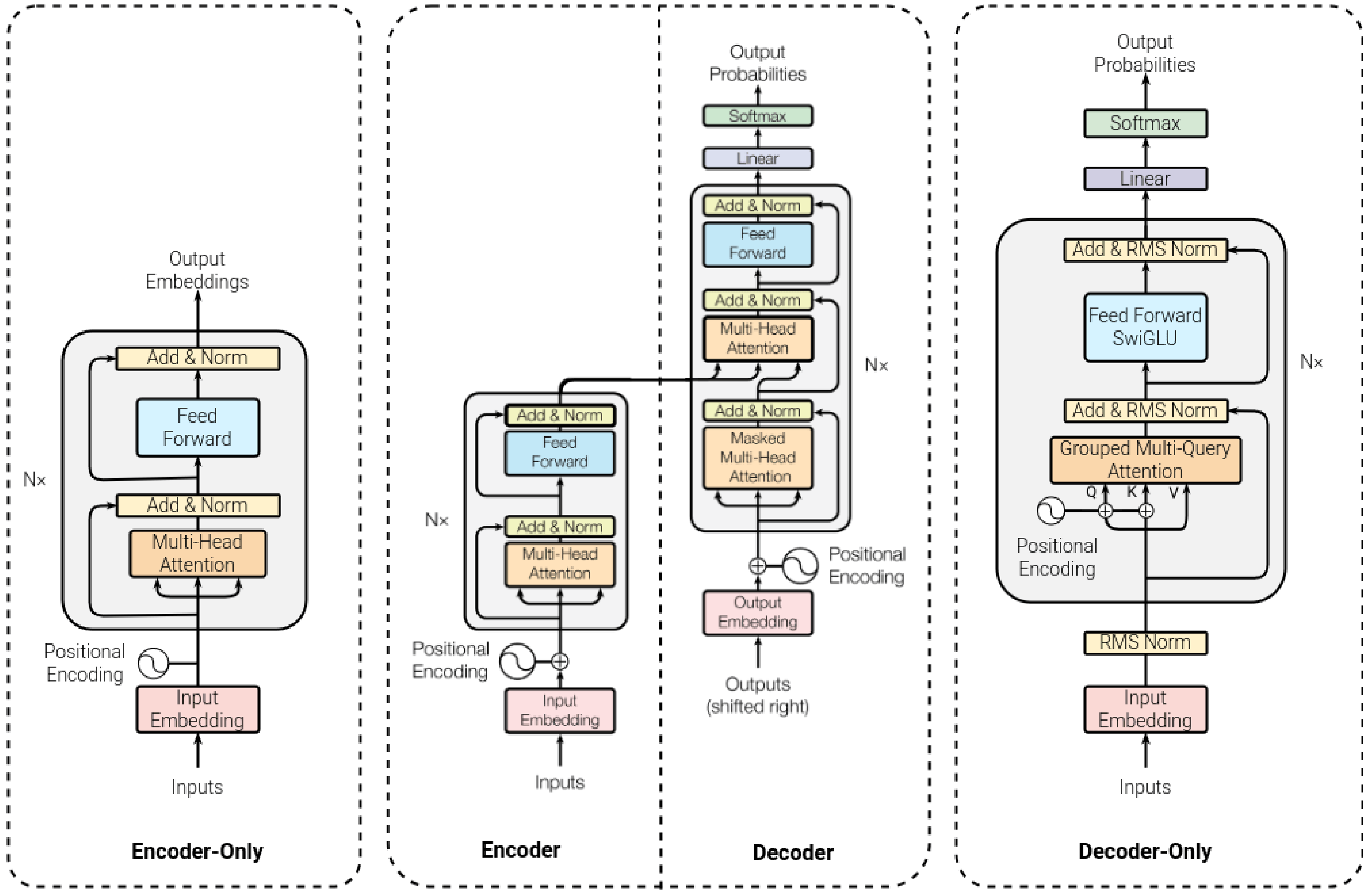
Depending on the desired task for which the LLM will be used, a specific architecture will need to be programmed using libraries such as PyTorch, TensorFlow, or Jax.

There are now several libraries that help with predefined architectures which can leverage the developments of the community

```python
import torch
from torch.nn import functional as F
from pytorch_pretrained_bert import GPT2Tokenizer, GPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

text = tokenizer.encode("Unicorns are a fascinating")
input, past = torch.tensor([output]), None
for _ in range(20):
    logits, past = model(input, past=past)
    input = torch.multinomial(F.softmax(logits[:, -1]), 1)
    text.append(input.item())
```



BERT
(Devlin et al., 2018)

Original Transformer
(Vaswani et al., 2017)

LLaMA
(Touvron et al., 2023)

4 – Training Algorithms

Once we have the data and the model architecture, we are ready to setup the training loop.

In this loop we will have some practical choices to make:

Data Loader

- Batch Size
- Shuffle
- Micro batch size

Loss Function

- Cross Entropy / Negative Log Likelihood
- MSE/MAE

Optimizer

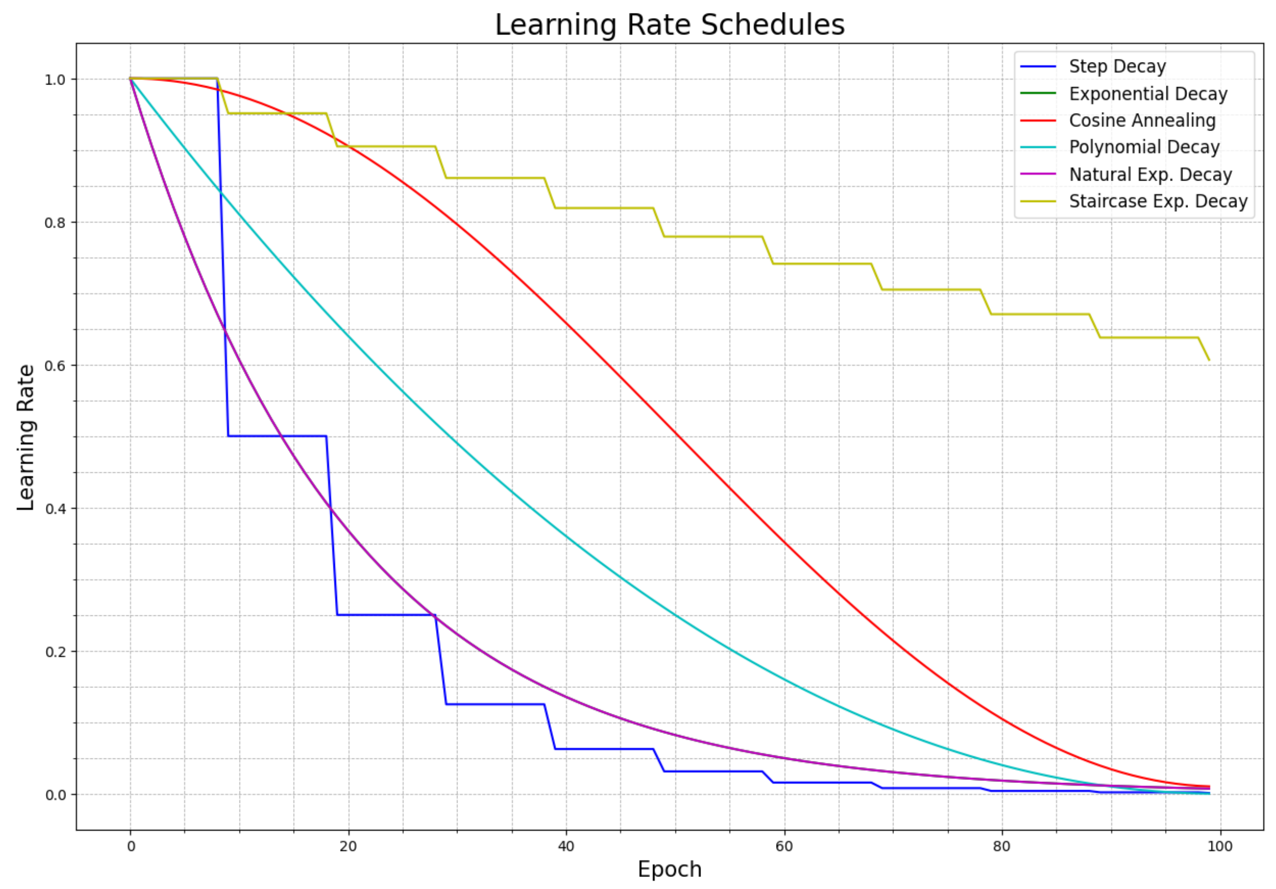
- Stochastic Gradient Descent
- ADAM
- RMSProp

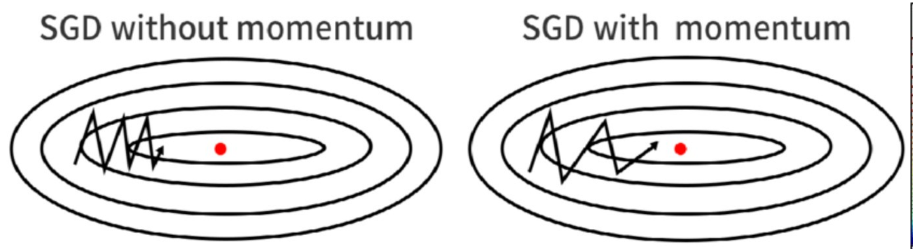
Learning Rate Schedule

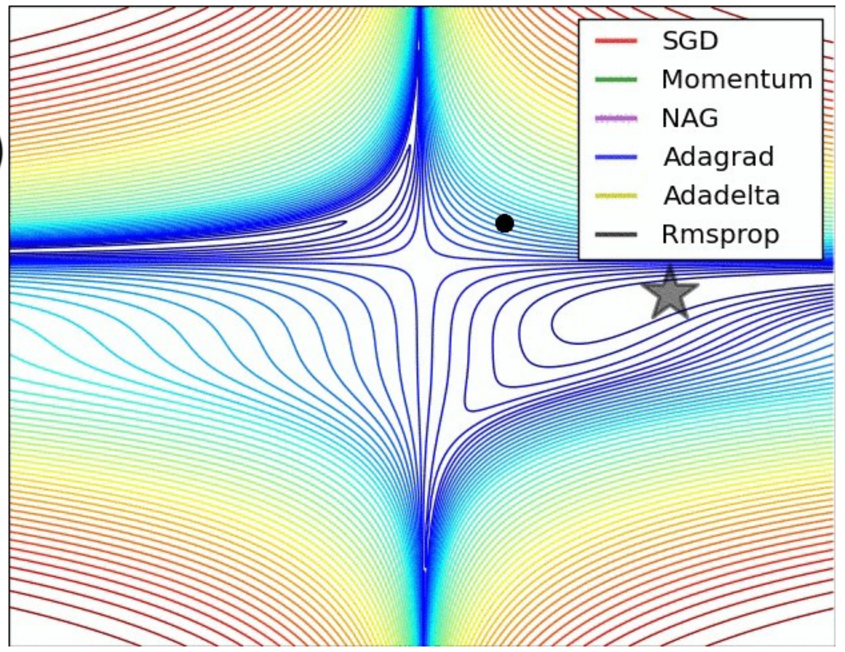
- Constant
- Cosine

Training Length

- Epochs, Batches, Tokens







5 – Compute Infrastructure for Training

- **Single Node**: Training happens on a single machine, using one or multiple CPUs/GPUs. This setup is simpler and typically used for smaller datasets and models.
- **Multi Node**: Involves distributing training across multiple machines (nodes). This is useful for training large models or datasets that can't fit into a single machine's memory. It requires communication between nodes, often managed by distributed computing frameworks like Horovod or PyTorch Distributed.

CPU, GPU, TPU Providers and Configurations:

- **CPU**: Central Processing Units (CPUs) are versatile but slower for deep learning tasks. Suitable for small models or when GPU/TPU resources are limited.
- **GPU**: Graphics Processing Units (GPUs) are highly efficient for parallel processing tasks like matrix operations, making them the go-to for deep learning. Configurations can vary from single to multiple GPUs.
- **TPU**: Tensor Processing Units (TPUs) are specialized hardware designed by Google for accelerating deep learning, especially for TensorFlow. TPUs are extremely fast but require specific configuration.

Cloud vs. Local:

- **Cloud**: Cloud providers (e.g., AWS, Google Cloud, Azure) offer scalable resources for training, allowing easy access to multiple GPUs/TPUs. This is ideal for large-scale training but comes with cost considerations.
- **Local**: Training on your own hardware (CPUs/GPUs) may be cost-effective but is limited by available resources. Local setups are good for experimentation and small-scale models but may struggle with large datasets or models.

CPU
- Small models
- Small datasets
- Useful for design space exploration

GPU
- Medium-to-large models, datasets
- Image, video processing
- Application on CUDA or OpenCL

TPU
- Matrix computations
- Dense vector processing
- No custom TensorFlow operations

FPGA
- Large datasets, models
- Compute intensive applications
- High performance, high perf./cost ratio

6 - Training Strategy

Data Parallelism:

DDP (Distributed Data Parallel): A standard method for splitting data across multiple GPUs/nodes, where each device processes a portion of the dataset, and gradients are averaged across all devices during backpropagation.

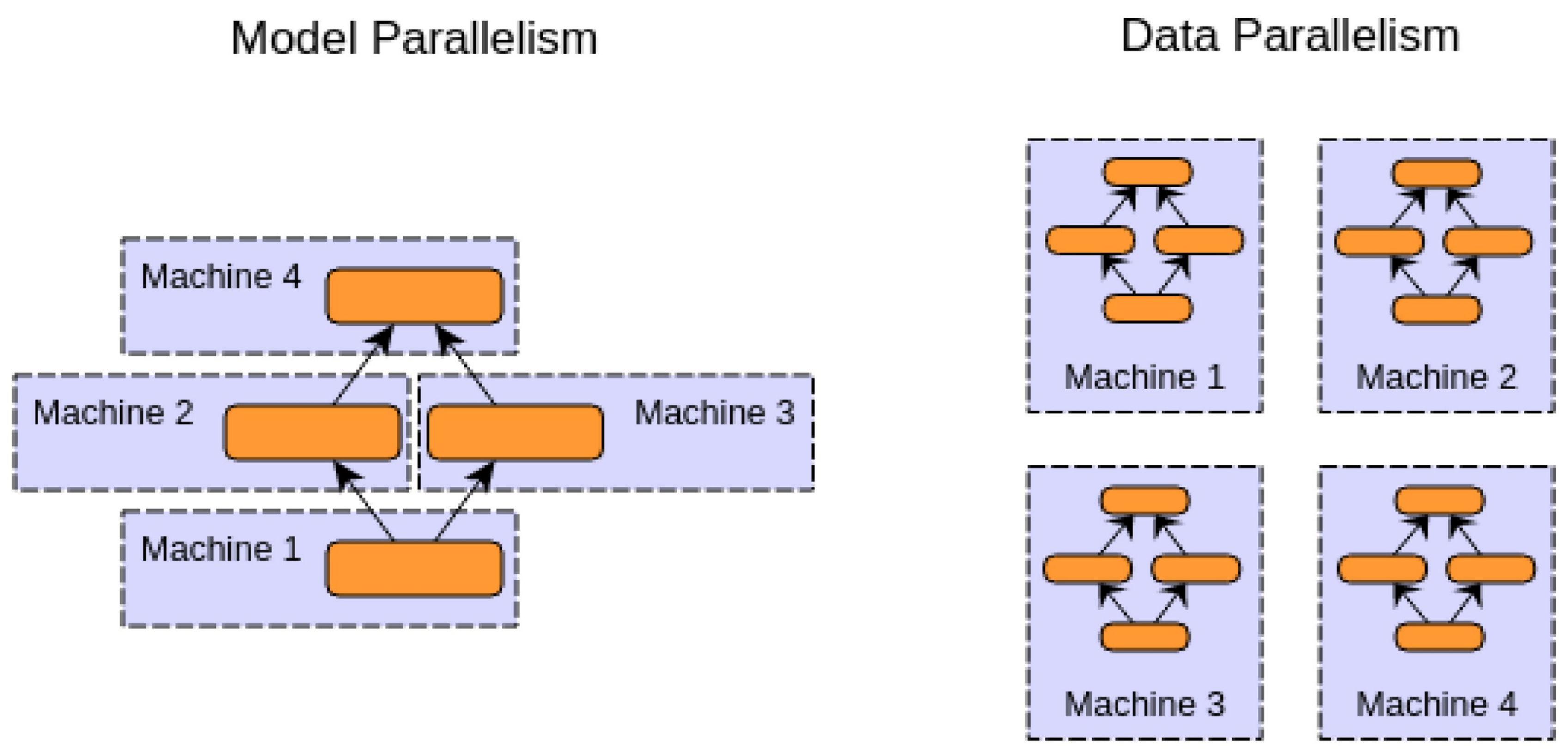
DeepSpeed: A framework designed for large-scale model training that optimizes memory and computation efficiency. It includes optimizations like zero redundancy, gradient accumulation, and mixed precision training.

FSDP (Fully Sharded Data Parallel): Shards (splits) both model weights and optimizer states across GPUs to minimize memory usage, enabling training of extremely large models on fewer GPUs.

Model Parallelism:

Pipeline Parallelism: Instead of replicating the entire model on each GPU, the model is split into layers and distributed across different devices. Each device processes a part of the forward and backward pass, working like an assembly line.

Tensor Parallelism: Model tensors (such as weight matrices) are split across multiple GPUs, allowing each GPU to handle a subset of the computations. This technique is commonly used for large-scale matrix operations in models like transformers

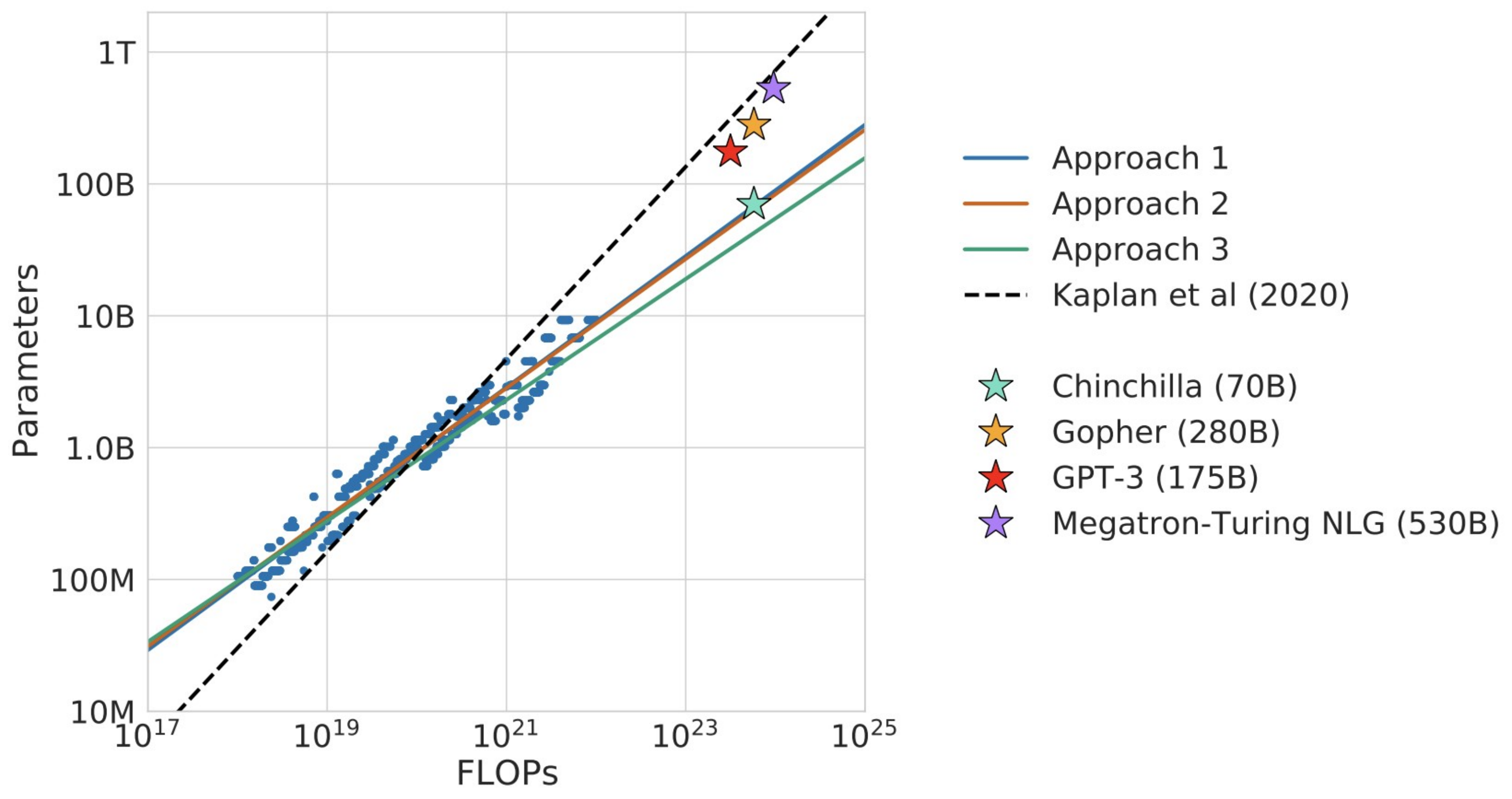


How much to train?

Chinchilla is a state-of-the-art language model proposed by DeepMind, known for optimizing model size and training duration based on the scaling laws of dee learning. Here's a breakdown of the key concepts and graphs:
Chinchilla Scaling Laws

Key Insight: For optimal performance, the number of parameters in the model should scale proportionally with the amount of data used for training. Previous large models (like GPT-3) were over-parameterized relative to the data availabl

Chinchilla's Adjustment: Reduces the number of parameters but trains on significantly more data to strike a better balance between model size and data size.



Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

Are we done? Limitations of base models.

Shallow Understanding:

Base models, like GPT, predict the next word based on patterns, not true comprehension. They generate plausible text without deep understanding.

No Task-Specific Knowledge:

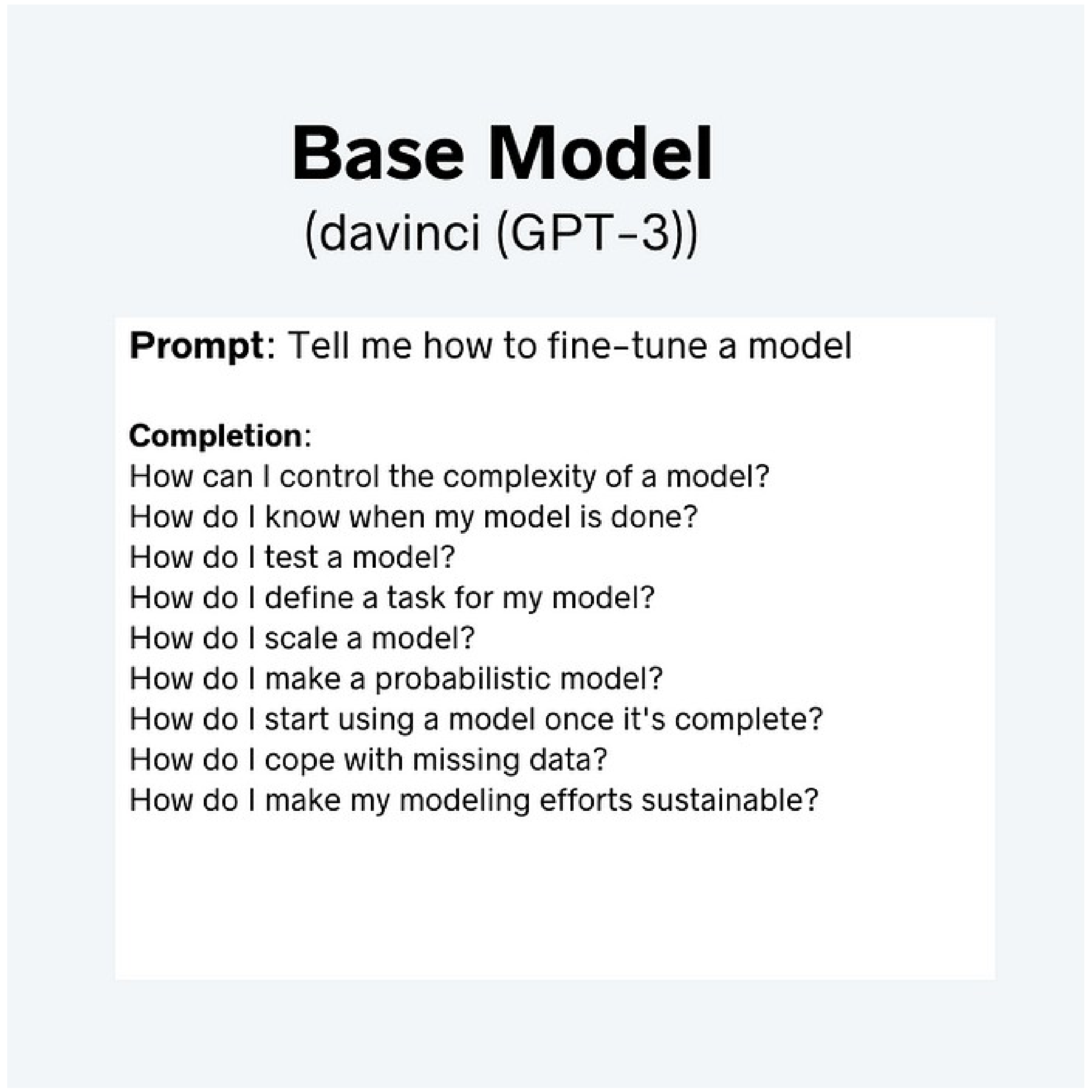
They struggle with complex tasks like reasoning, question-answering, or structured outputs because they aren't fine-tuned for specific tasks.

No Long-Term Planning:

Next-token models can't plan or keep track of long sequences. They may lose coherence in long tasks (like storytelling) because they don't retain a memory of earlier steps.

Limited in Complex Dependencies:

They handle local coherence but struggle with tasks requiring structure or logic over longer spans (e.g., math, formal reasoning).



Base Model

(davinci (GPT-3))

Prompt: Tell me how to fine-tune a model

Completion:
How can I control the complexity of a model?
How do I know when my model is done?
How do I test a model?
How do I define a task for my model?
How do I scale a model?
How do I make a probabilistic model?
How do I start using a model once it's complete?
How do I cope with missing data?
How do I make my modeling efforts sustainable?

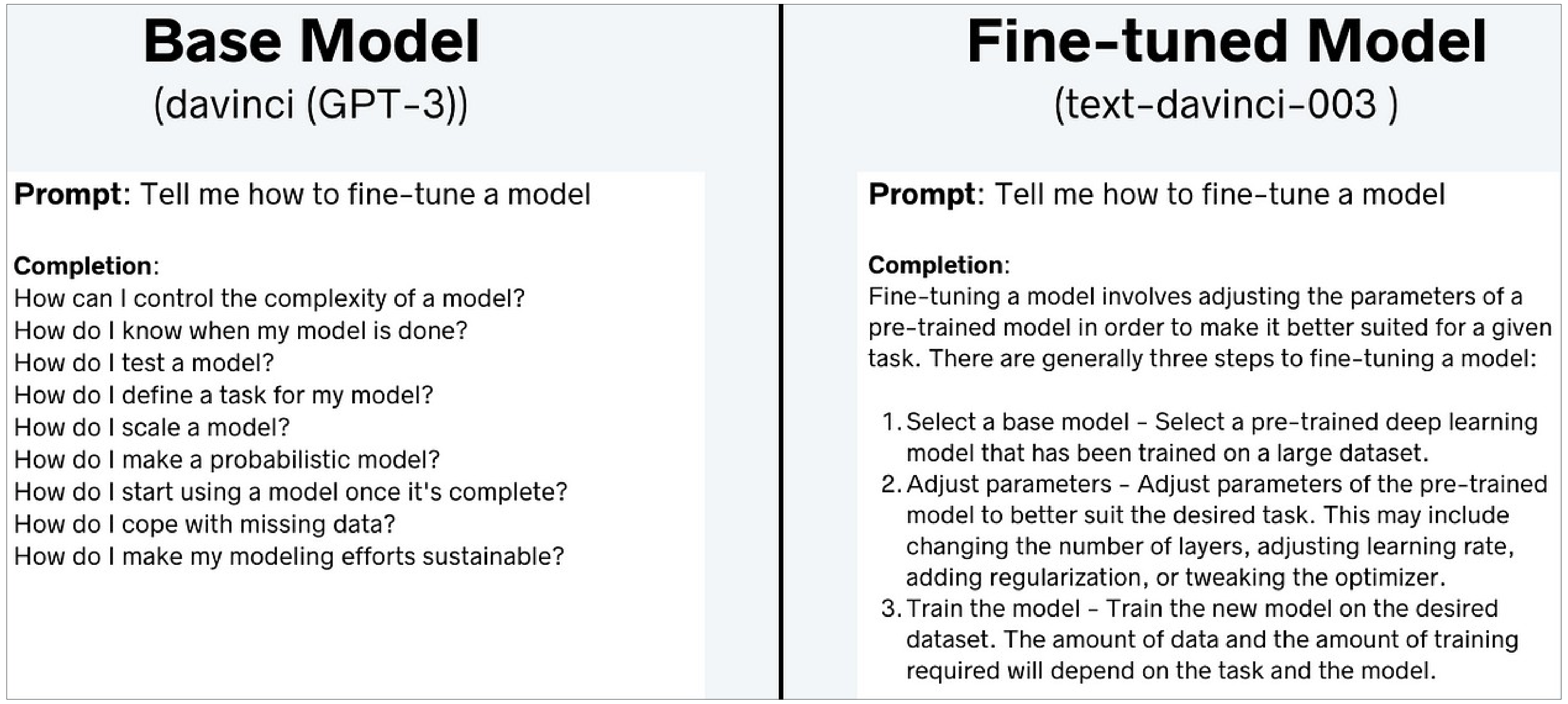
Instruction Fine-tuning

Beyond the next token

How useful are pre-trained models?

While base models can be used with techniques such as few shot learning and other in-context learning methods, these quickly become limitations

In every case where a task would be needed, a collection of few shot examples would also be needed.



Base Model
(davinci (GPT-3))

Prompt: Tell me how to fine-tune a model

Completion:
How can I control the complexity of a model?
How do I know when my model is done?
How do I test a model?
How do I define a task for my model?
How do I scale a model?
How do I make a probabilistic model?
How do I start using a model once it's complete?
How do I cope with missing data?
How do I make my modeling efforts sustainable?

Fine-tuned Model
(text-davinci-003)

Prompt: Tell me how to fine-tune a model

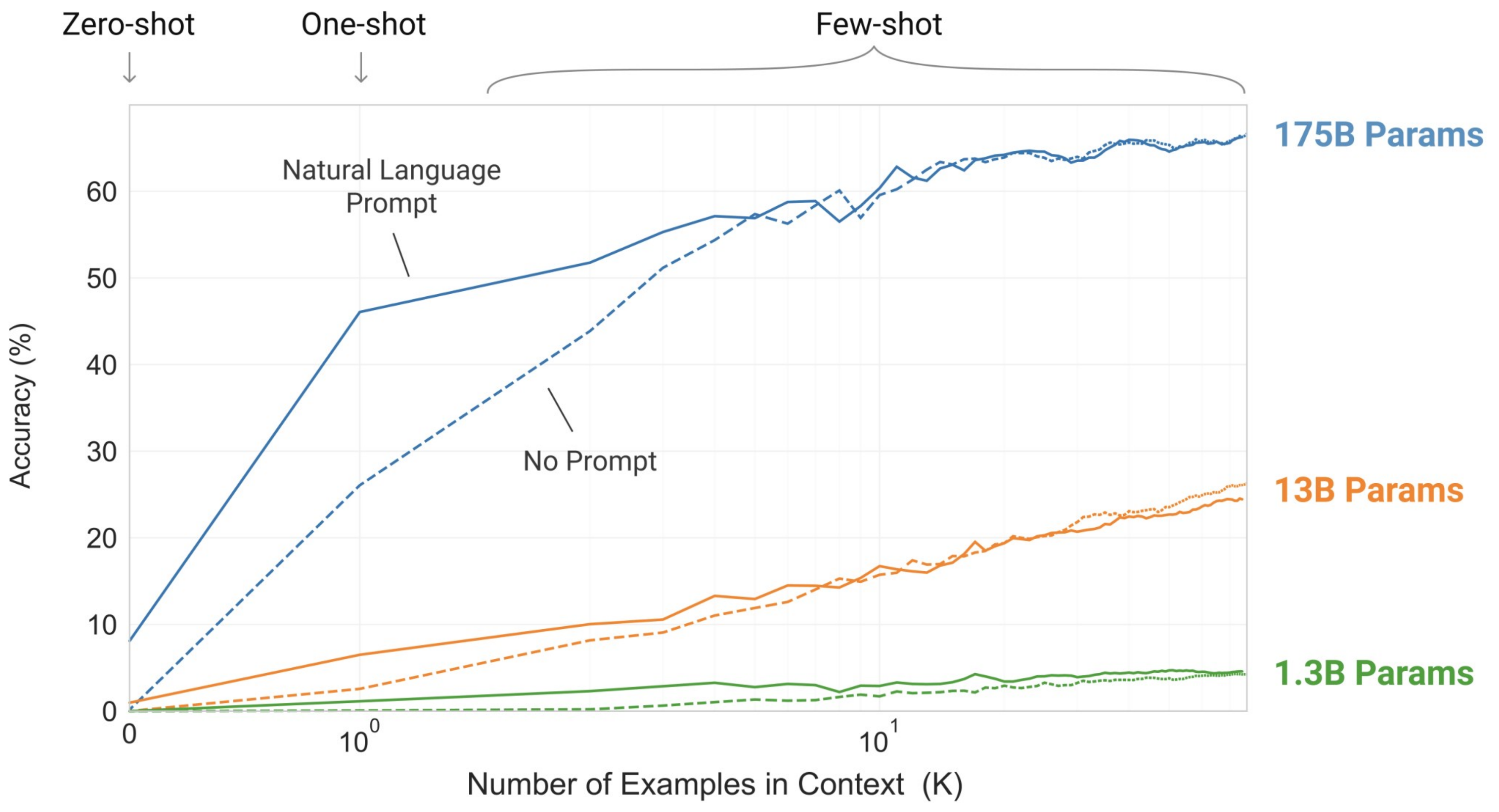
Completion:
Fine-tuning a model involves adjusting the parameters of a pre-trained model in order to make it better suited for a given task. There are generally three steps to fine-tuning a model:

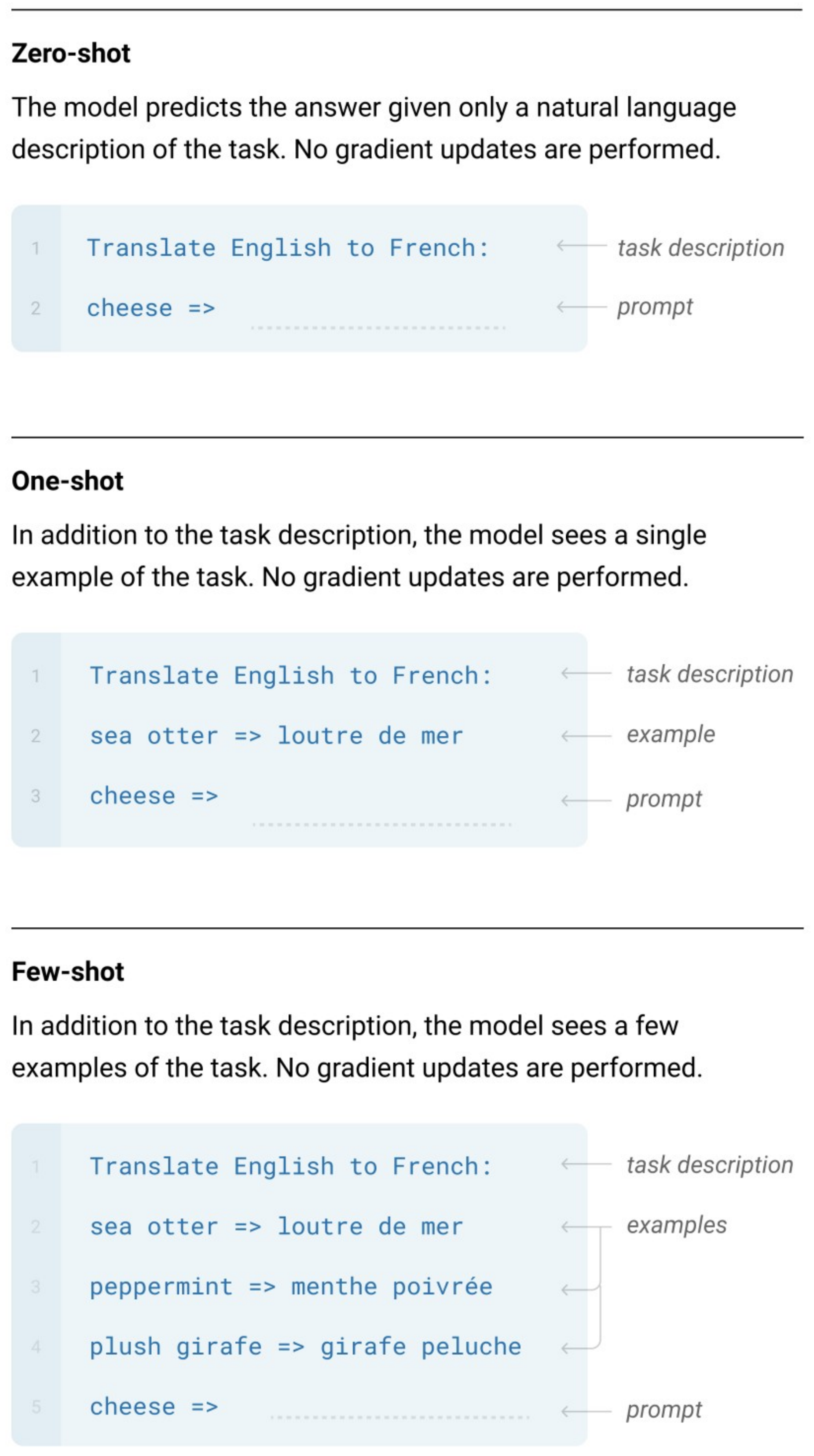
1. Select a base model - Select a pre-trained deep learning model that has been trained on a large dataset.
2. Adjust parameters - Adjust parameters of the pre-trained model to better suit the desired task. This may include changing the number of layers, adjusting learning rate, adding regularization, or tweaking the optimizer.
3. Train the model - Train the new model on the desired dataset. The amount of data and the amount of training required will depend on the task and the model.

Few shot learning

GPT 3 showed the world that LLMs could perform tasks beyond what it was trained on.

This ability to learn from the input context, know as in-context learning, can be used to leverage the inherent abilities and knowledge the model gained through pre-training.





Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:
cheese =>
```

One-shot

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

```
Translate English to French:
sea otter => loutre de mer
cheese =>
```

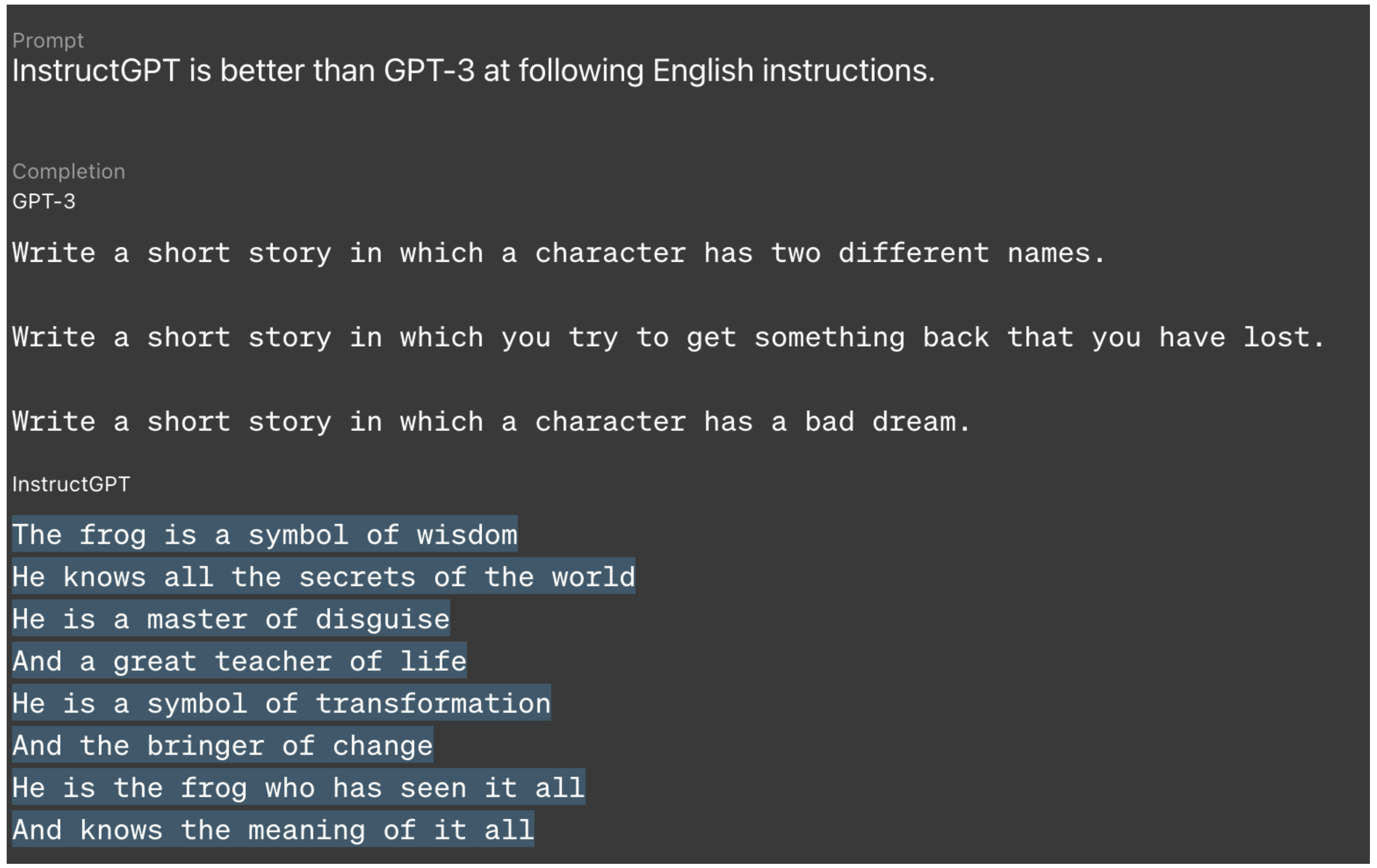
Few-shot

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

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Instruction following

We can use this ability of the model to utilize in-context learning to change how the model behaves. By training the model to learn how to respond to input instructions with response pairs, the model can learn how to respond to general queries.



Prompt
InstructGPT is better than GPT-3 at following English instructions.

Completion
GPT-3

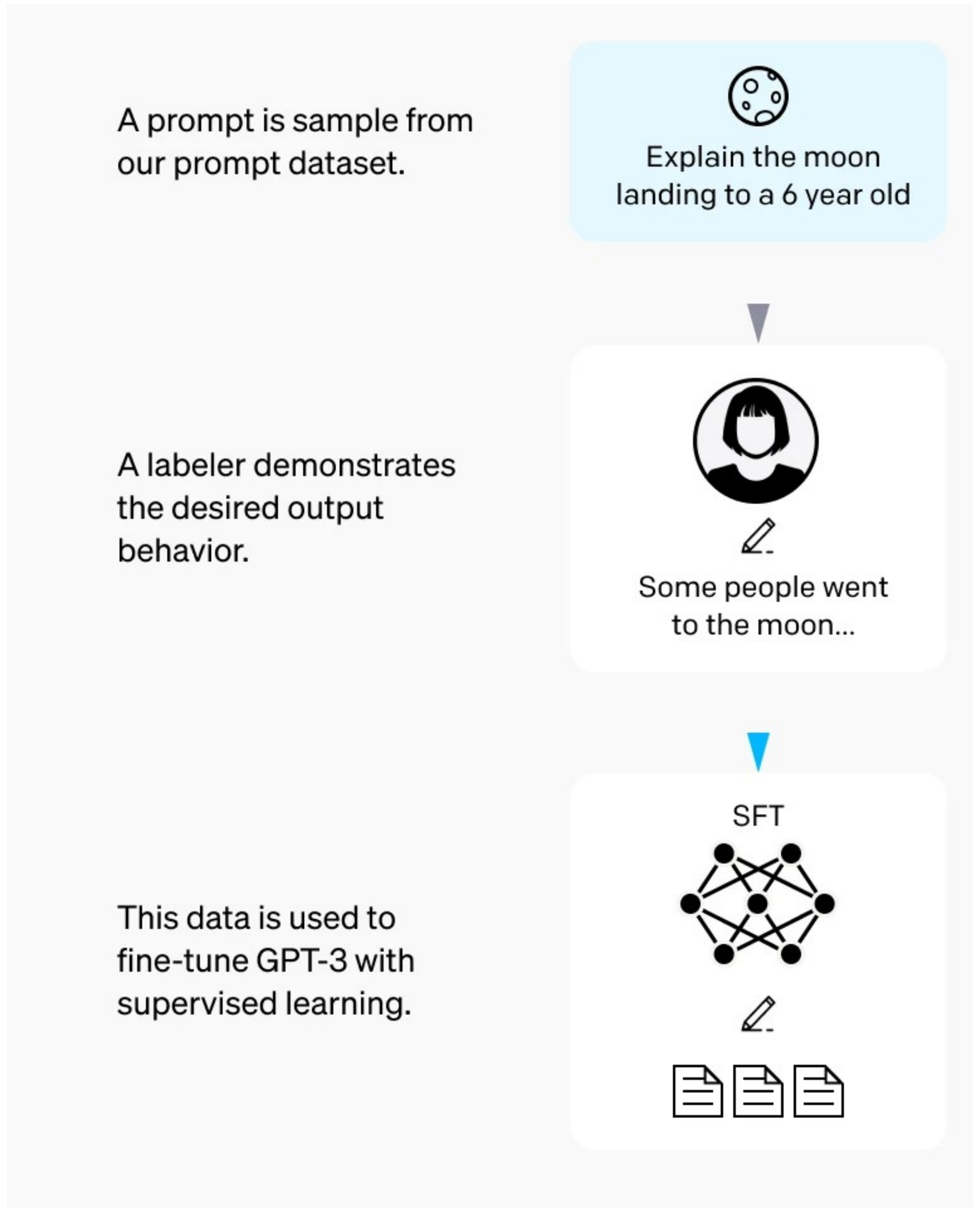
```
Write a short story in which a character has two different names.

Write a short story in which you try to get something back that you have lost.

Write a short story in which a character has a bad dream.
```

InstructGPT

```
The frog is a symbol of wisdom
He knows all the secrets of the world
He is a master of disguise
And a great teacher of life
He is a symbol of transformation
And the bringer of change
He is the frog who has seen it all
And knows the meaning of it all
```


A prompt is sample from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT

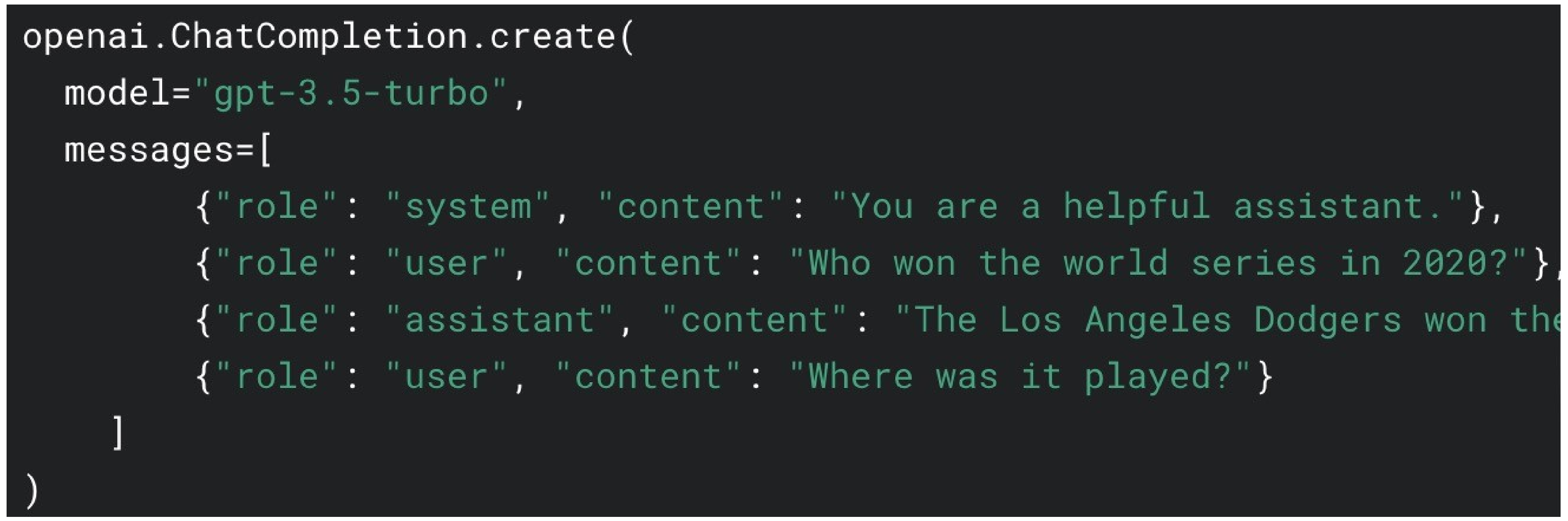
The chat structure

What is the best format to train models with?

Currently, the structure used to build training data and to interact with the LLM APIs is based on the “chat structure”. Here a series of roles and their respective content is stored in a messages array.

This allows for the system, assistant, and user roles to be stored in the same place, and the content of the interaction can be progressively added.

This provides a consistent format to both train and use the models as they react to user input.



```
openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the
        {"role": "user", "content": "Where was it played?"}
    ]
)
```

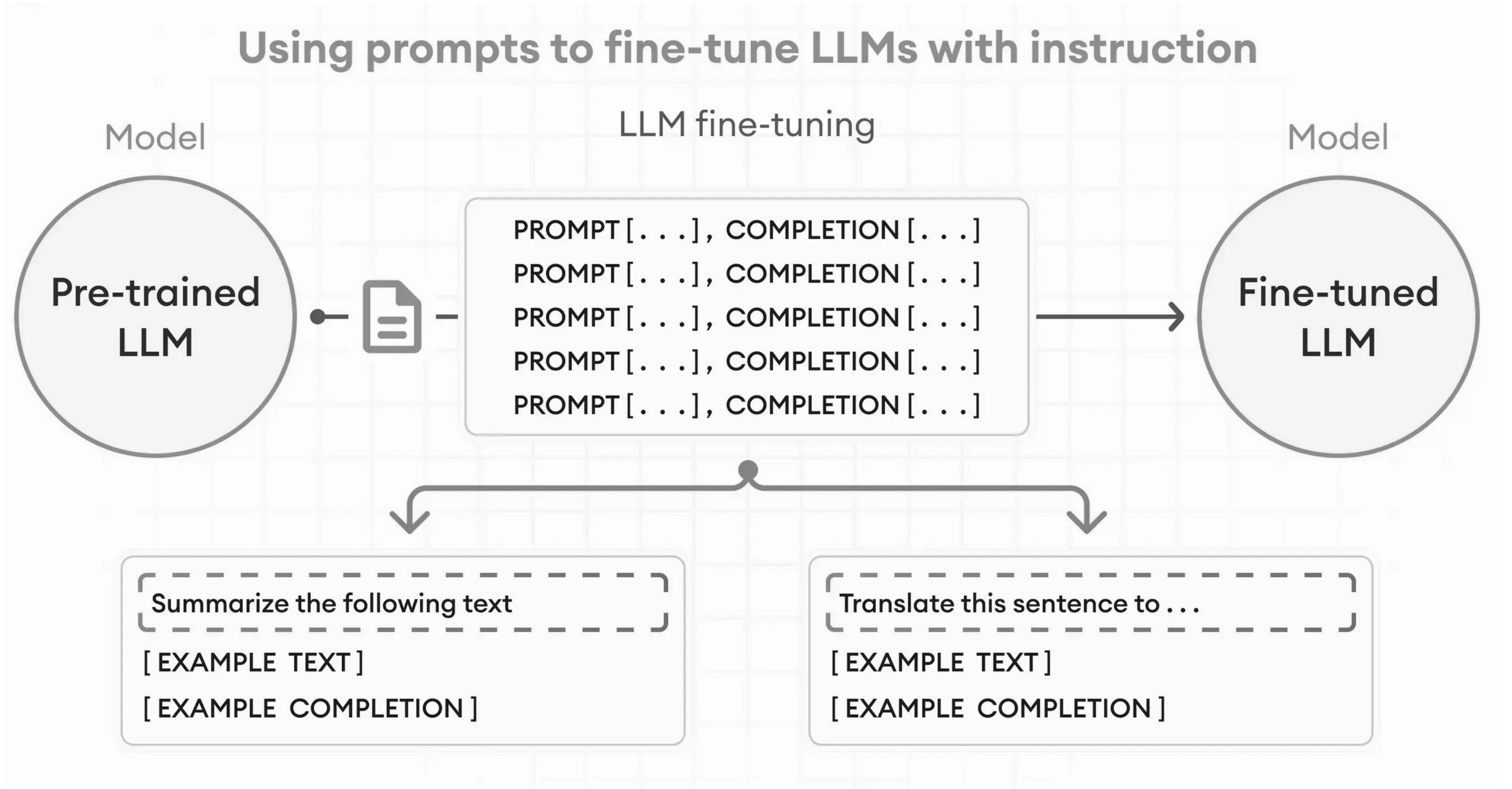
IFT Data Preparation

Generating prompt-response pairs requires more intentional curation than for pre-training.

The data can be generated synthetically with another LLM that has already been trained to create queries from raw data.

Or, more commonly or when starting from scratch, a human-generated dataset will need to be created with a variety of query-response pairs.

These pairs can be simple input-output or they can also include additional context to the query, e.g. Summarize this email into 5 bullet points.



Using prompts to fine-tune LLMs with instruction
Model
LLM fine-tuning
Model
Pre-trained LLM
PROMPT [. . .], COMPLETION [. . .]
PROMPT [. . .], COMPLETION [. . .]
PROMPT [. . .], COMPLETION [. . .]
PROMPT [. . .], COMPLETION [. . .]
PROMPT [. . .], COMPLETION [. . .]
Fine-tuned LLM
Summarize the following text
[EXAMPLE TEXT]
[EXAMPLE COMPLETION]
Translate this sentence to . . .
[EXAMPLE TEXT]
[EXAMPLE COMPLETION]

The model will be trained to produce the specific output, not predict the next token. This changes how the model behaves.

Limits of IFT for domain adaptation – Going deeper?

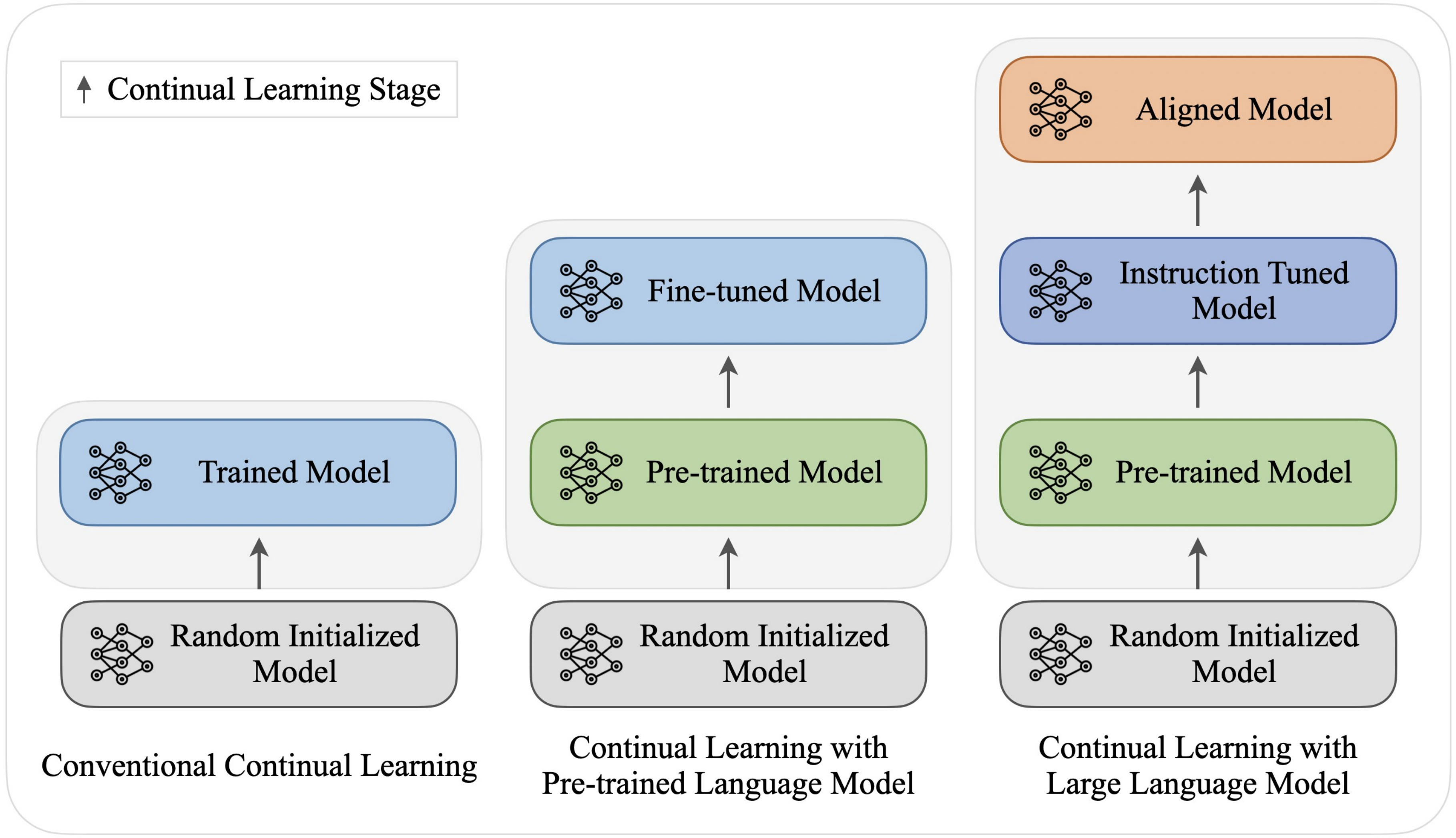
When a model has been trained for IFT, it will have new abilities to respond to a users' query.

However, one issue that can arise is a **lack of domain adaptation**.

If the model was trained on a broadly trained base model, the instruction fine-tuning data often just changes the behavior of the model, rather than teaching is any domain-specific behavior.

If we need to instill domain-specific information, or align the model, how can we do this without starting from scratch?

Ans: Continued Pre-training



DARTMOUTH ENGINEERING NVIDIA

Continued Pre-training

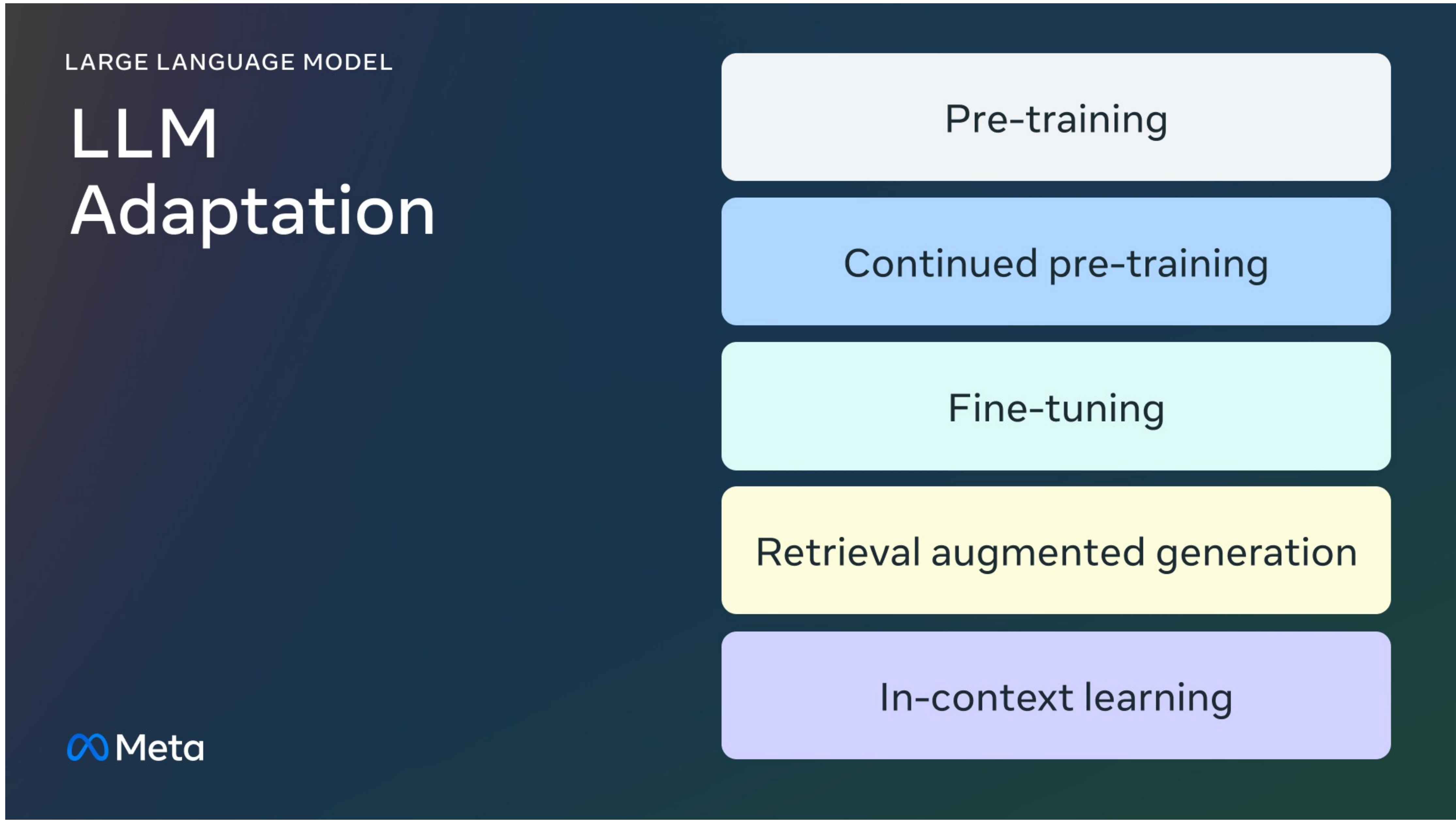
Focusing domain understanding

Why do Continued Pre-Training?

Building an LLM to understand language and respond to queries can be a challenging task.

Using the breadth of of data from the web can teach the model to have vast knowledge and linguistic skills, but focusing on the nuances of specific fields can be limited, particularly for smaller models.

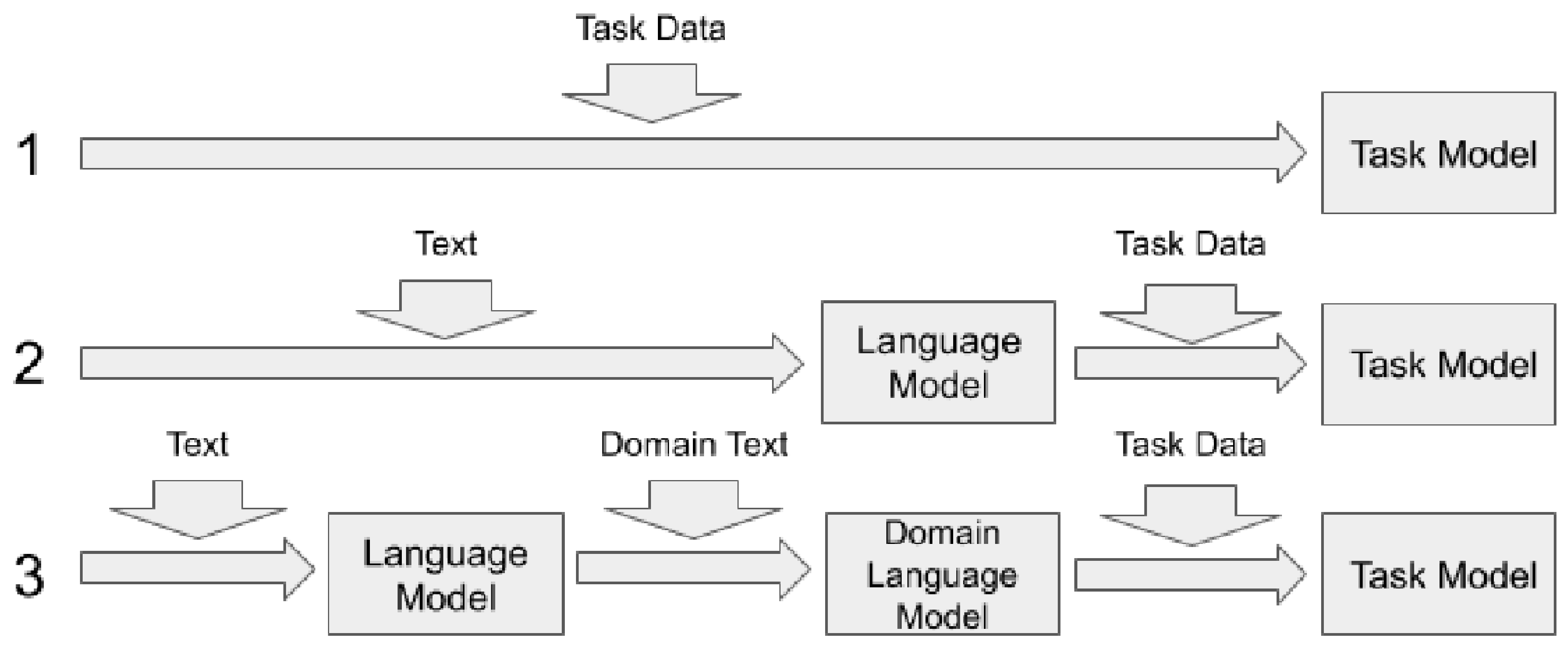
Continued Pre-Training (CPT) can help address this issue by training a base model on domain-specific data to tune the model weights to align with a specific area of interest.



What does it mean to *continue* Pre-training?

Unlike Instruction Fine-Tuning (IFT) where the goal is to change how the model responds by training it on pairs of inputs and outputs, in CPT, a base model is trained in the exact same manner as pre-training but for a much shorter amount of time and with a custom and smaller dataset.

The hope is that this new CPT-base model that results will be more attuned to specific domain tasks, e.g. legal document analysis.



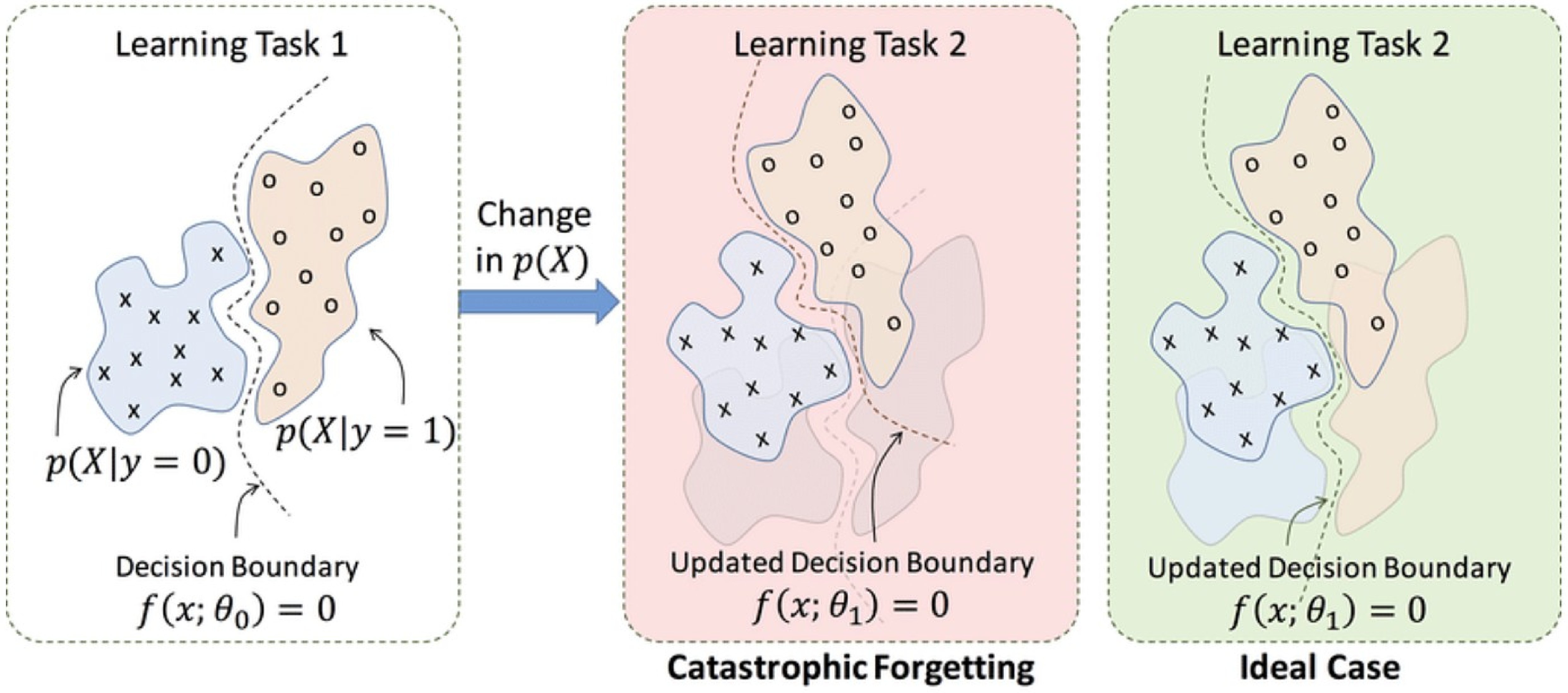
1. Pre-train a model on task data: typically poor performance.

2. IFT a base mode: moderate performance on domain-specific tasks

3. CPT and IFT a model for domain-specific task: best results

CPT Data and Catastrophic Forgetting

While focusing a model to learn a specific domain, there is a risk that the model will forget about other, potentially important, pieces of information. This is known as **catastrophic forgetting**, and can only be mediated with a very carefully selected data mixture to preserve whatever desired knowledge from the base model.



CPT and IFT

In most use cases, a CPT model on its own is not very useful, in the same way that a base model is not.

A CPT model will still only predict the next token.

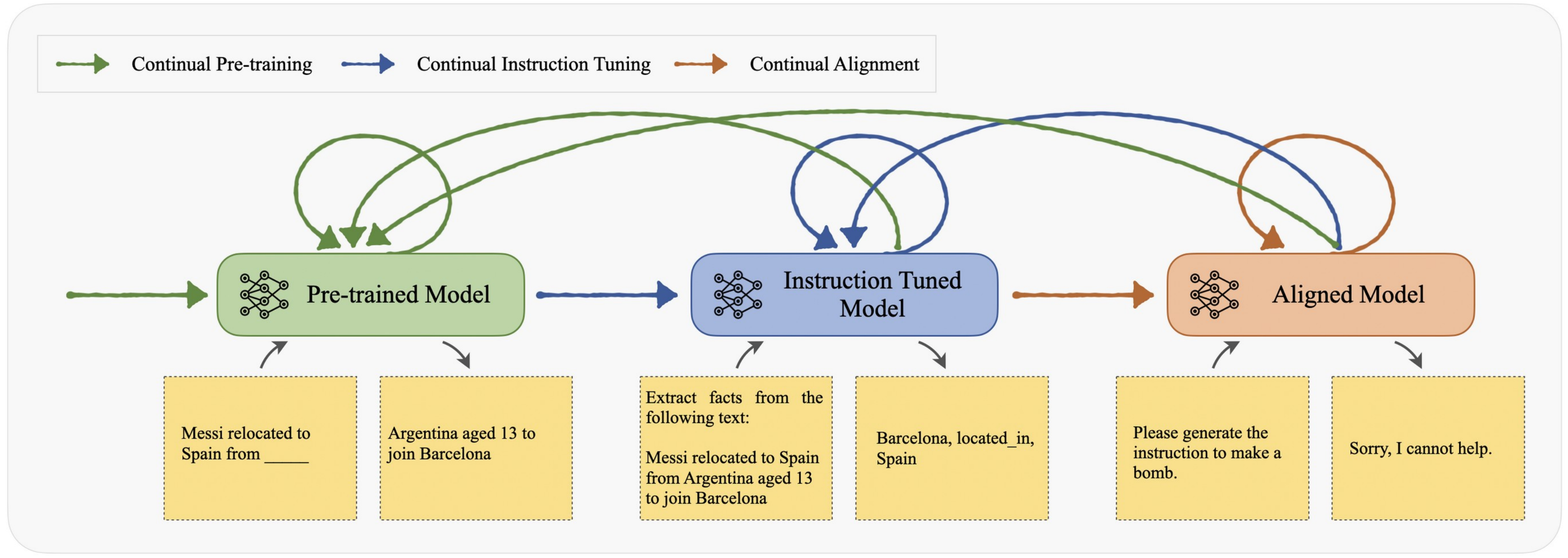
The typical workflow is to:

1) Test a model on a particular set of domain-specific tasks
2) If the model achieves sufficient performance, stop.

3) If not, take the underlying base model and perform CPT on domain-specific data

4) Then perform IFT on the new CPT version of the base model

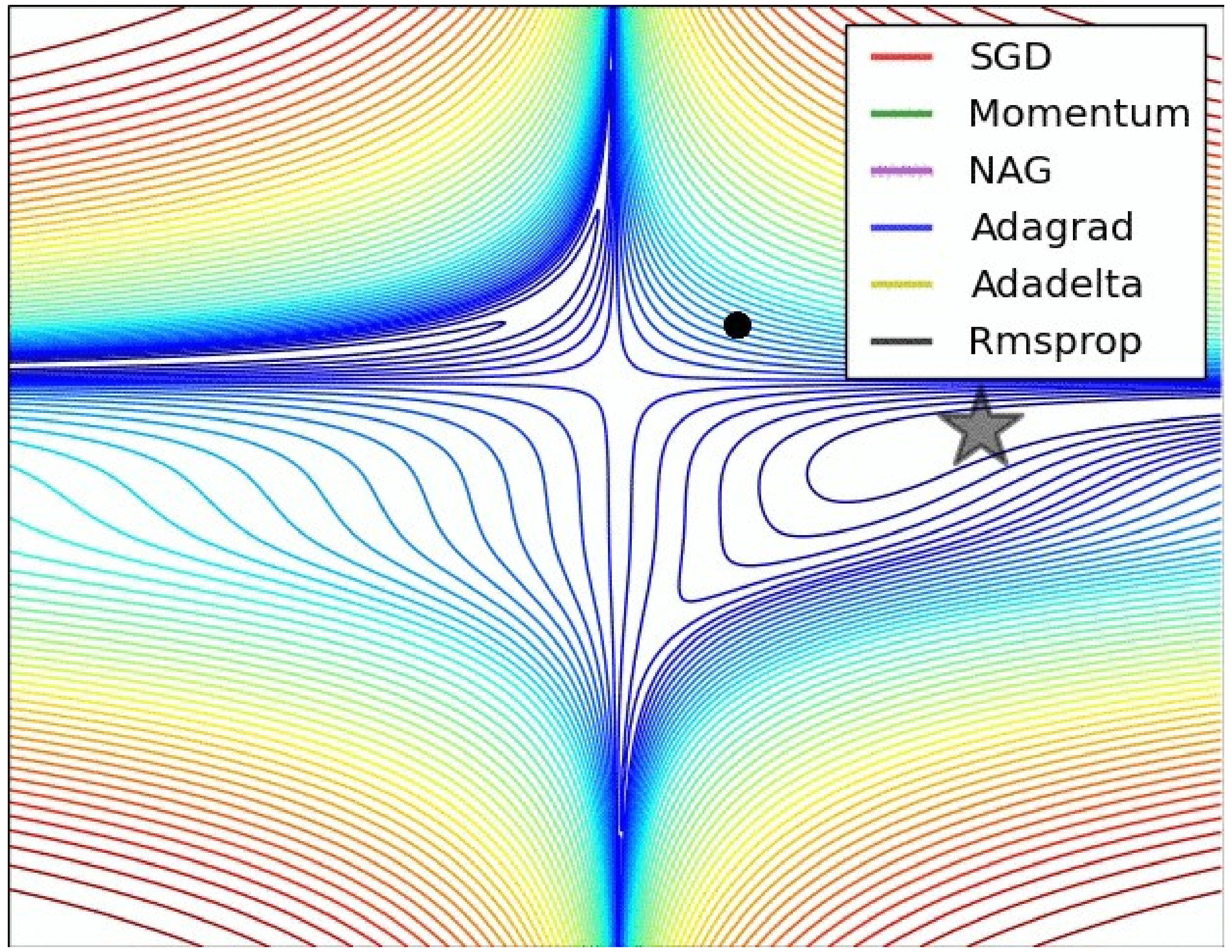
5) Collect more data for each stage as needed and repeat the process until sufficient performance is achieved.


Continual Pre-training
Continual Instruction Tuning
Continual Alignment
Pre-trained Model
Instruction Tuned Model
Aligned Model
Messi relocated to Spain from _____
Argentina aged 13 to join Barcelona
Extract facts from the following text: Messi relocated to Spain from Argentina aged 13 to join Barcelona
Barcelona, located_in, Spain
Please generate the instruction to make a bomb.
Sorry, I cannot help.

Wrap Up

Training LLMs

- Today we discussed the different ways to train LLMs
- We introduced the idea of LLMs as Auto Regressive models
- Pre-training was presented as the method to train these models on massive datasets of text
- Instruction Fine-tuning was discussed as a solution to these models not inherently being able to response to a query
- Continued pre-training was also covered as a means to domain tune an existing base model.



DARTMOUTH ENGINEERING

Thank you!