Deep Learning Practice - NLP

Adapting to Downstream Tasks:

Fine-tuning, Prompting, Instruction Tuning and Preference tuning

Mitesh M. Khapra

AI4Bharat, Department of Computer Science and Engineering, IIT Madras

Pre-Training

We trained the GPT-2 model with the CLM (Causal Language Modelling) training objective

Minimize

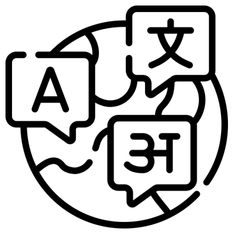
$$\boxed{\mathscr{L} = -\sum_{\mathbb{D}} \sum_{i=1}^{T} y_i \log(\hat{y}_i))}$$

However, how can we adapt it to different downstream tasks ?

Classification

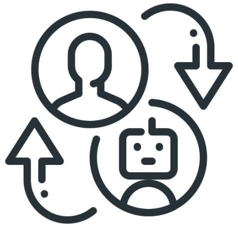
sentiment, NER,..

Text generation

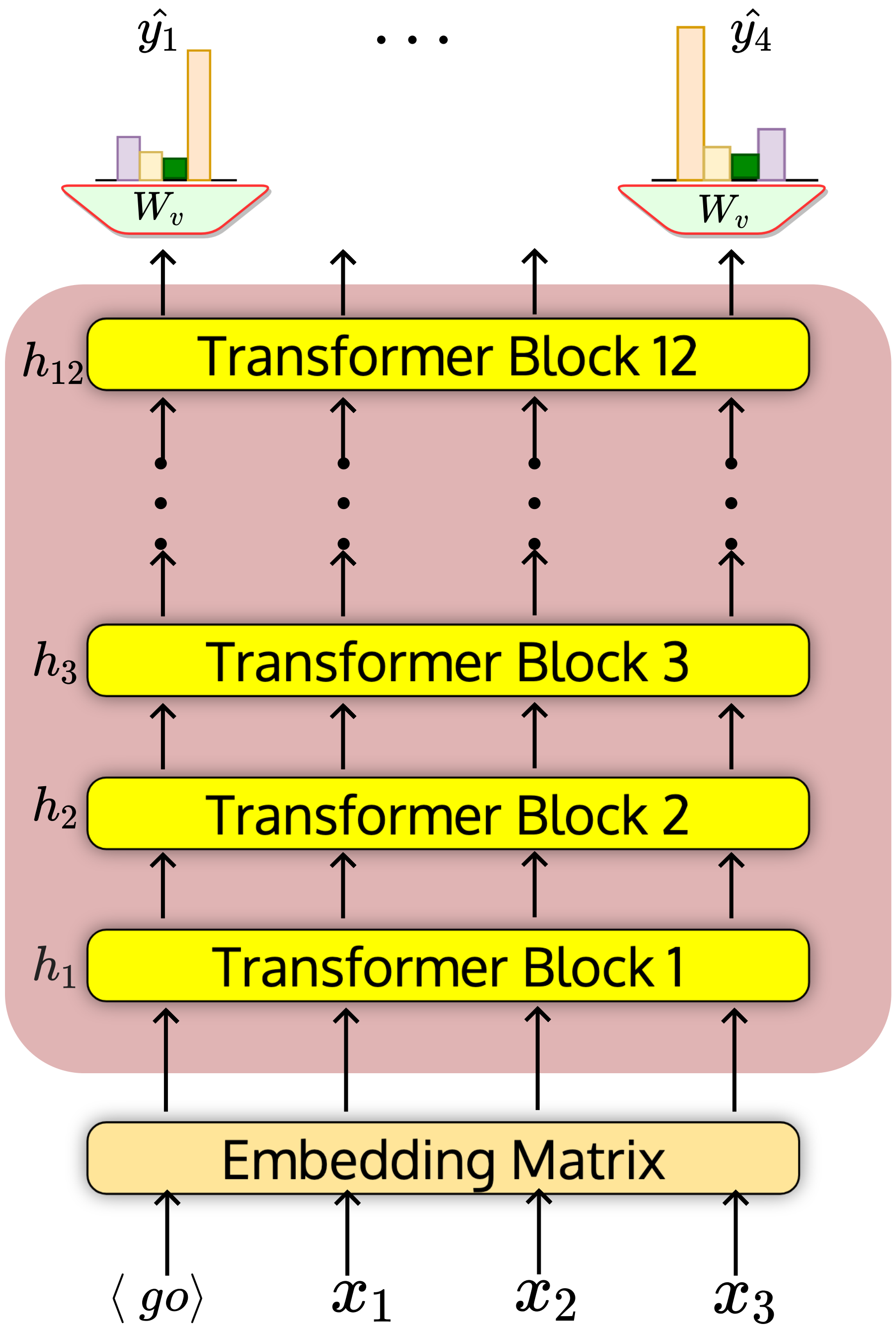


QA, translate, summarize

Conversation

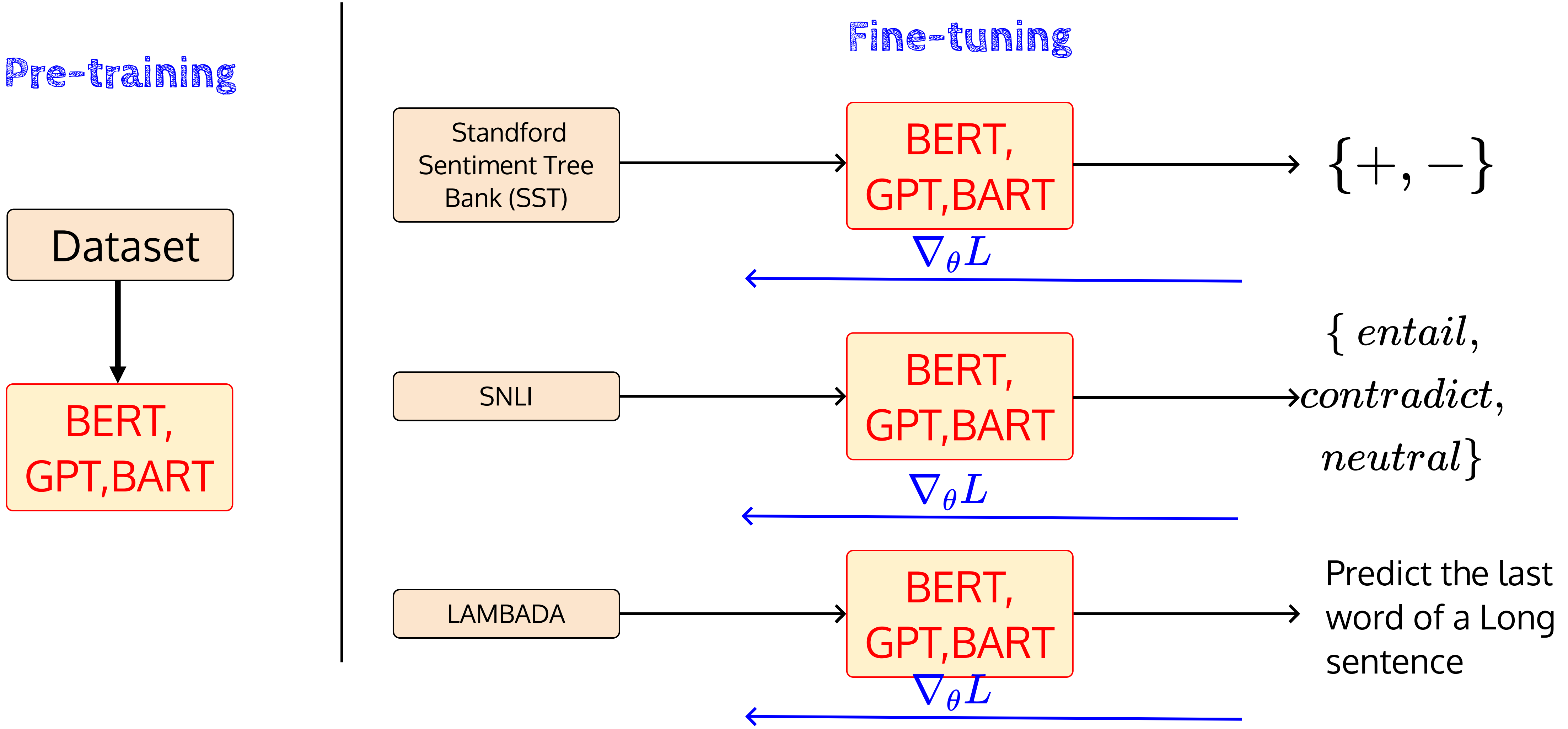


Chat bot



One approach for using the pre-trained models for downstream tasks is to independently fine-tune the parameters of the model for each task

That is, we make a copy of the pre-trained model for each task and fine-tune it on the dataset specific to that task



Fine-tuning for Text Classfication

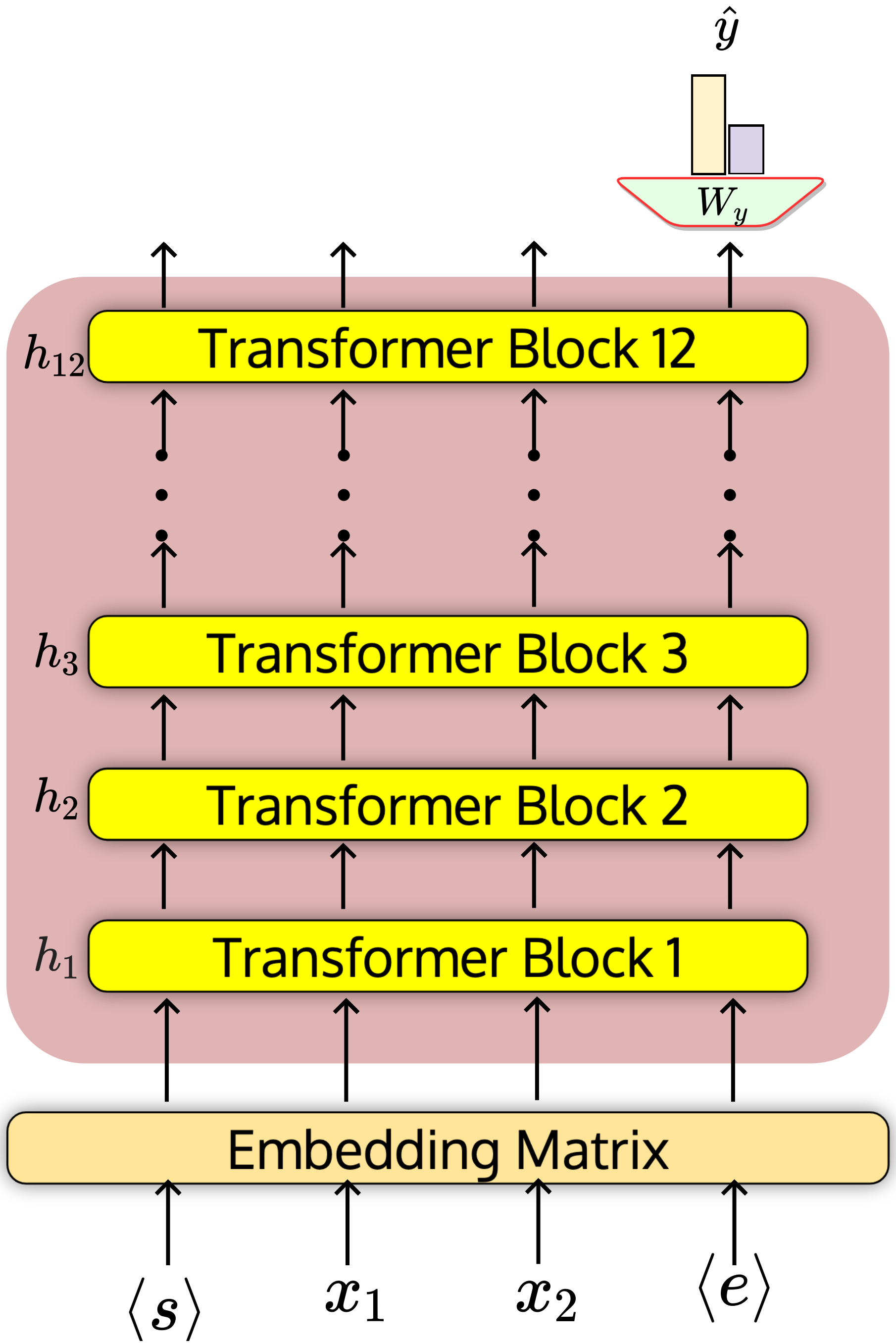
Fine-tuning involves adapting a model for various downstream tasks (with a minimal or no change in the architecture)

Each sample in a labelled data set $\mathcal{C}$ consists of a sequence of tokens $x_1, x_2, \cdots, x_m$ with the label y

Initialize the parameters with the parameters learned by solving the pre-trianing objective

At the input side, add additional tokens based on the type of downstream task. For example, start $\langle s \rangle$ and end $\langle e \rangle$ tokens for classification tasks

At the output side, replace the pre-training LM head with the classification head (a linear layer W_y)



Fine-tuning for Text Classfication

Now our objective is to predict the label of the input sequence

$$\hat{y} = P(y|x_1, \cdots, x_m) = softmax(W_y h_l^m)$$

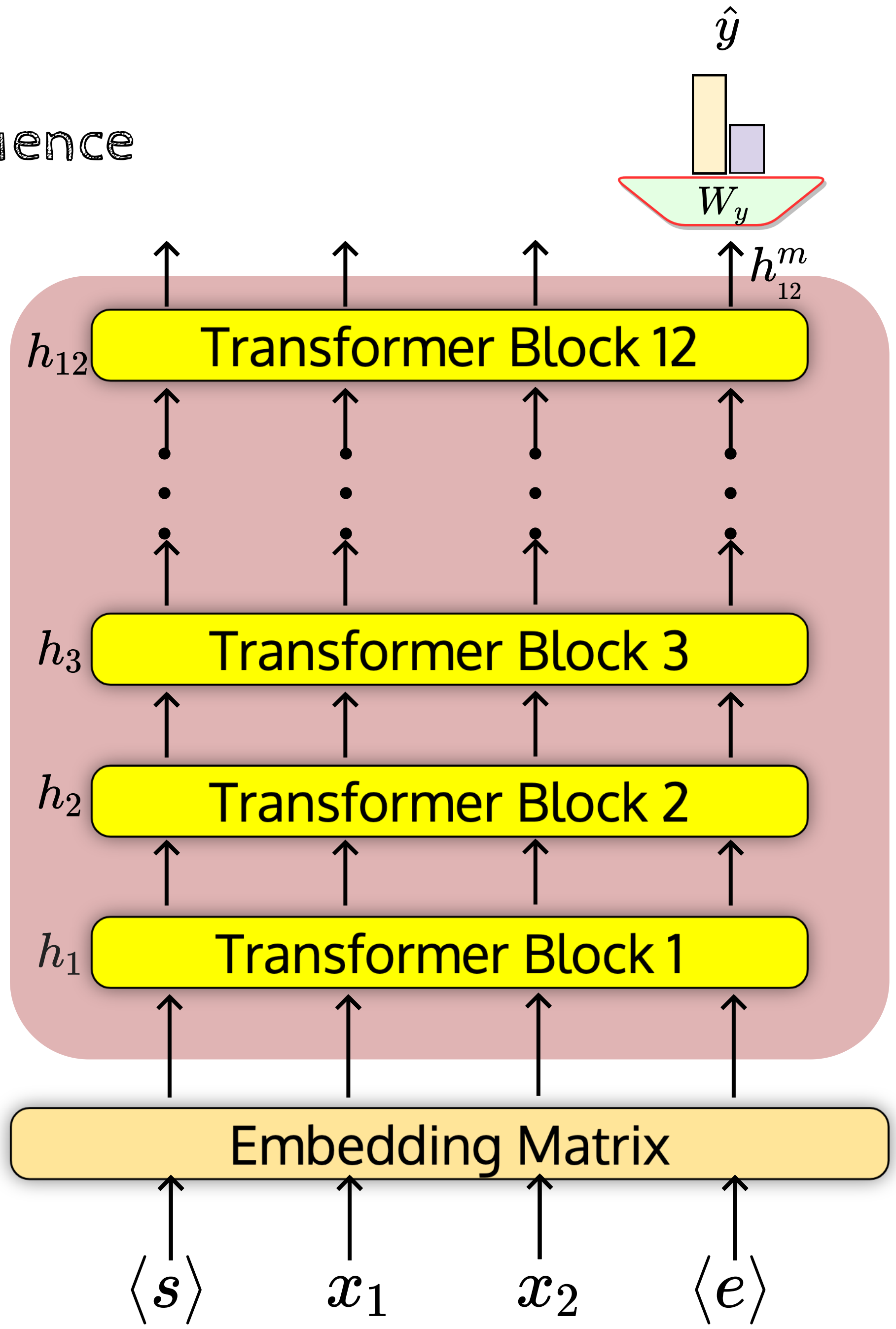
Note that we take the output representation at the last time step of the last layer h_l^m.

It makes sense as the entire sentence is encoded only at the last time step due to causal masking.

Then we can minimize the following objective

$$\boxed{\mathscr{L} = -\sum_{(x,y)} \log(\hat{y}_i)}$$

Note that W_y is randomly intialized. Padding or truncation is applied if the length of input sequence is less or greater than the context length



Fine-tuning for Text Classfication

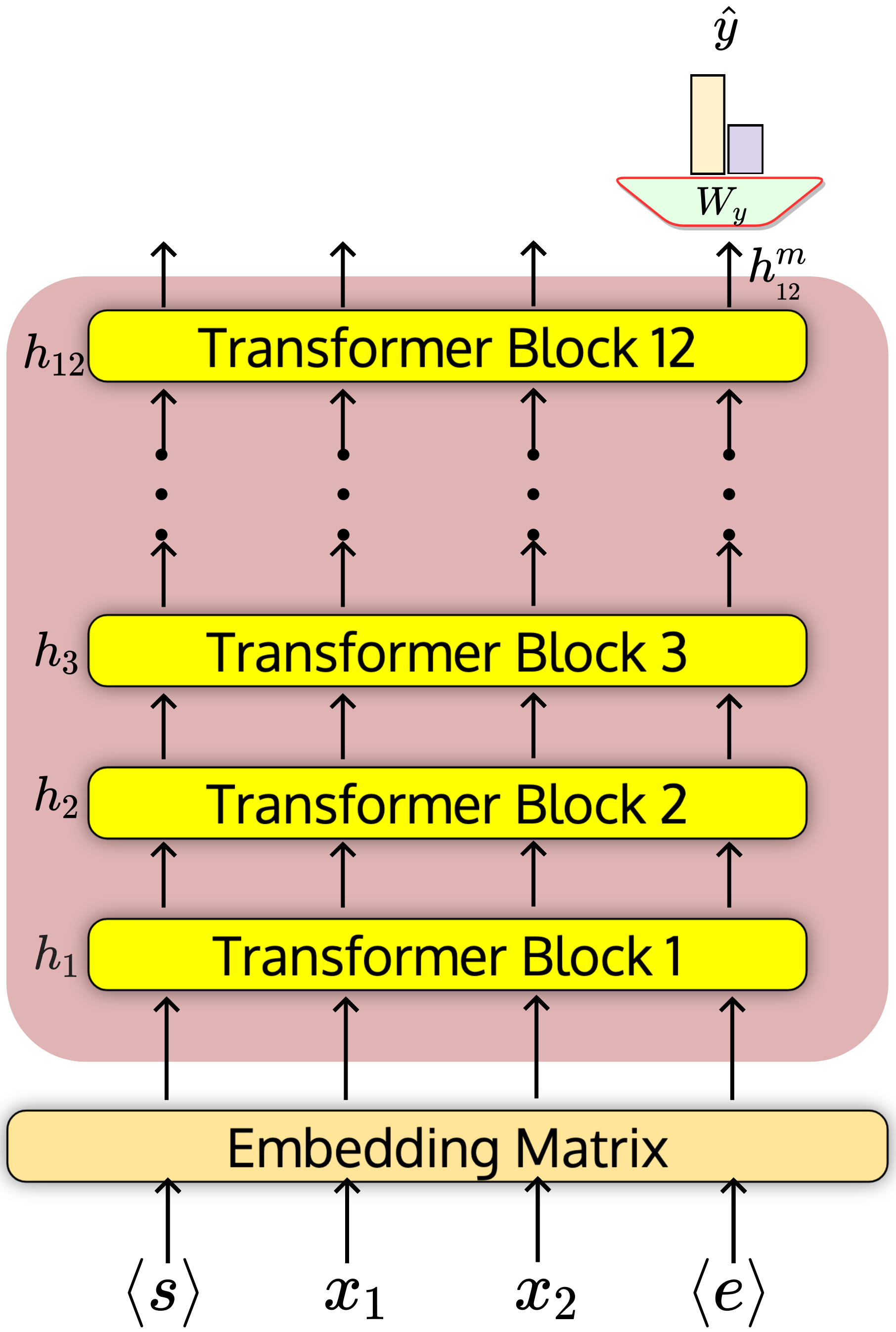
We can freeze the pre-trained model parameters and randomly initialize the weights of the classification head (W_y) while training the model

In this case, the pre-trained model acts as a feature extractor and the classification head act as a simple classifier.

The other option is to train all the parameters of the model which is called full fine-tuning

In general, the latter approach provides better performance on downstream tasks than the former.

However, there is a catch.



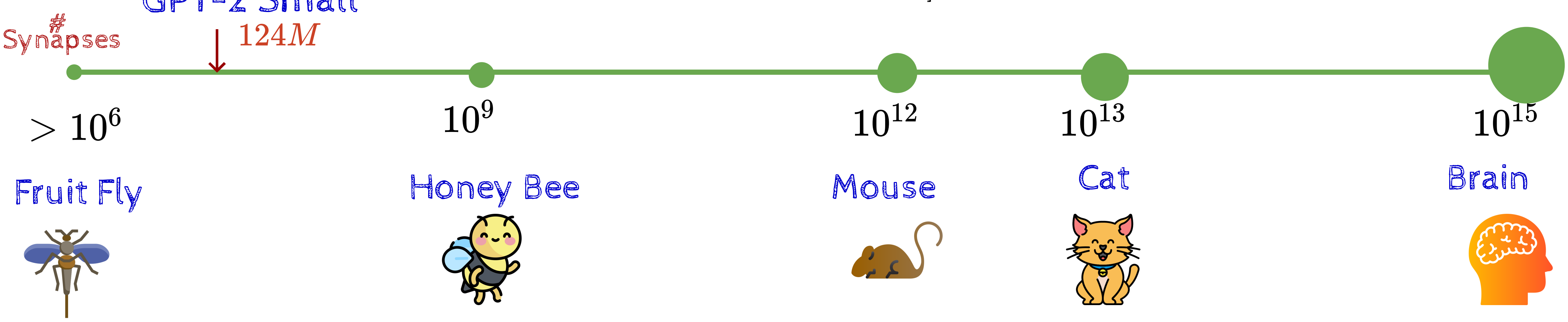
GPT-2
SMALL

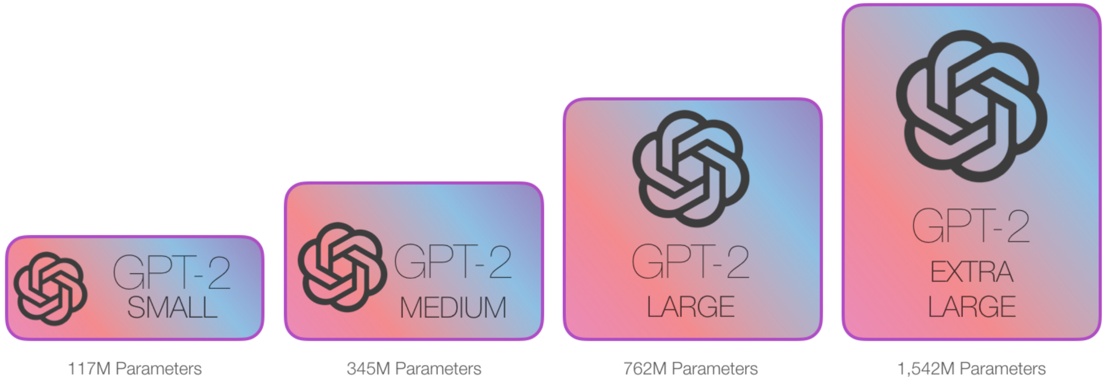
117M Parameters

(T4 has 16 GB of Memory)

We have trained GPT-2 small that has about 124 million parameters

It requires at least 3 to 4 GB of GPU memory to train the model with a batch of size 1 (assuming 4 bytes per parameter and adam optimizer)

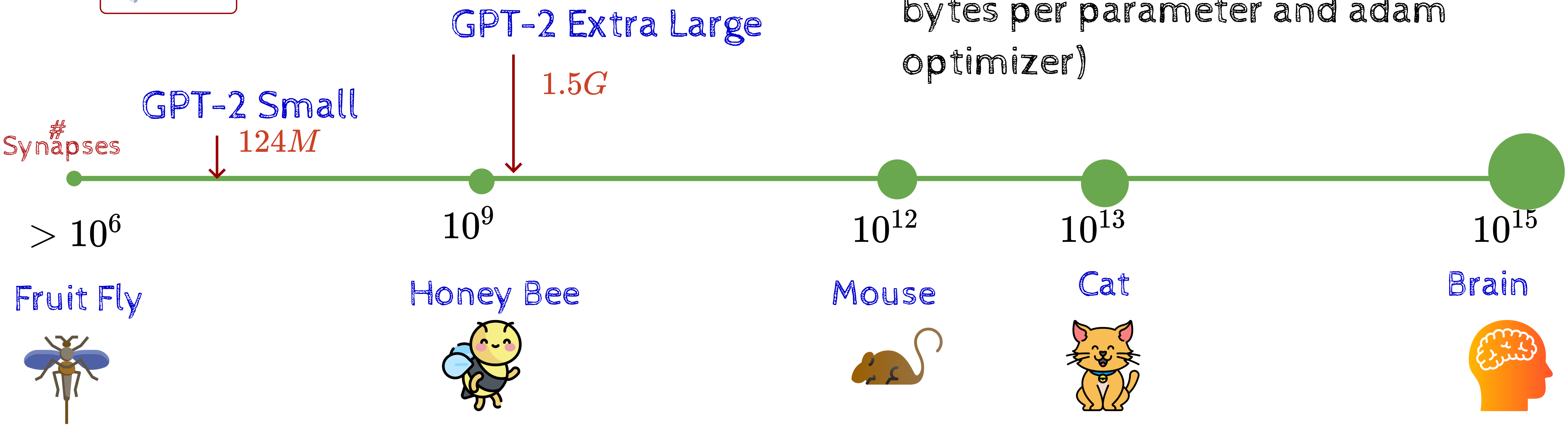


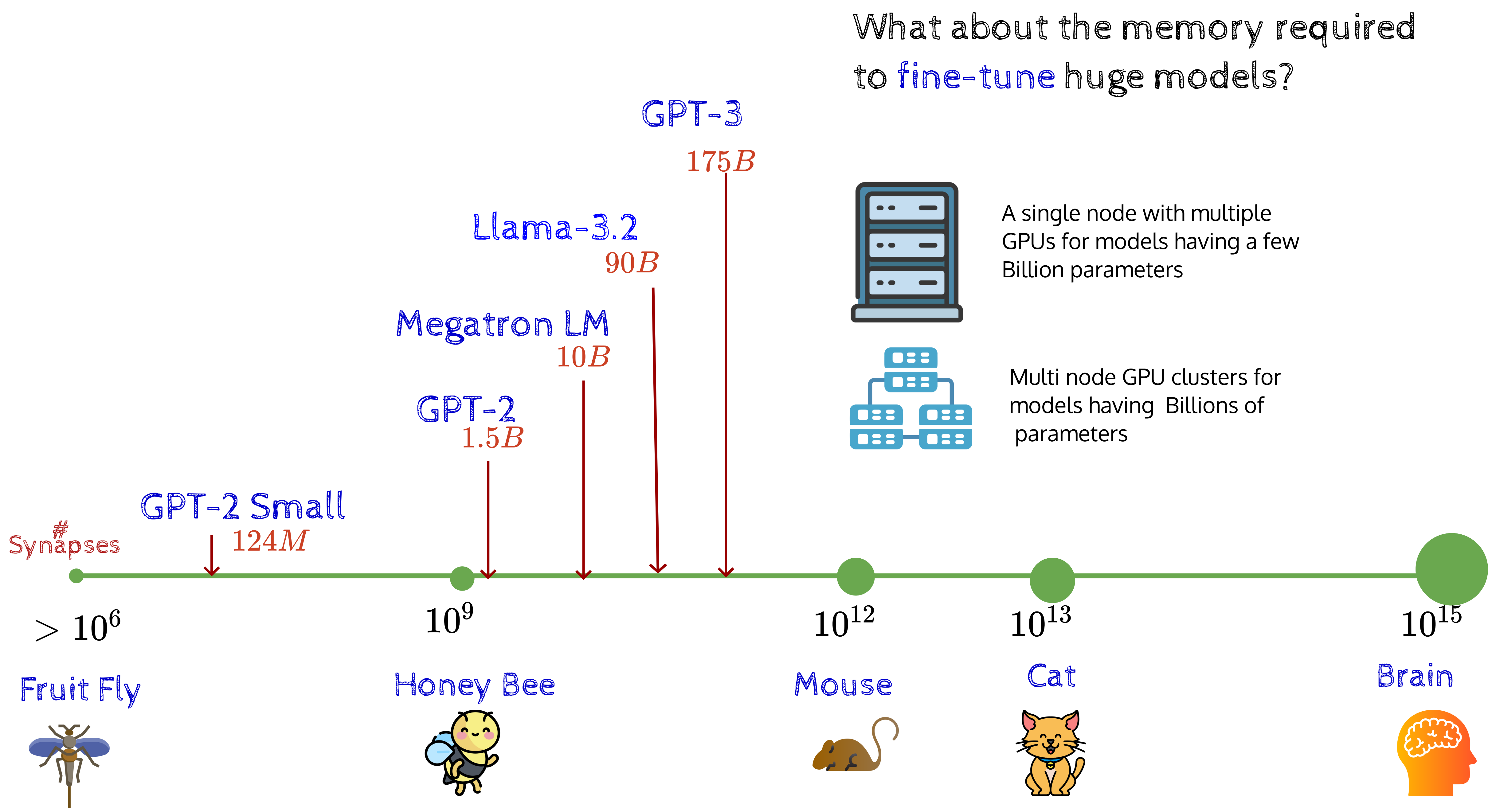


What about the memory required to fine-tune GPT-2 Extra Large?

It requires at least 22 to 24 GB of GPU memory to train the model with a batch of size 1 (assuming 4 bytes per parameter and adam optimizer)

(V100 has 32 GB of Memory)




What about the memory required to fine-tune huge models?
GPT-3
175B
Llama-3.2
90B
Megatron LM
10B
GPT-2
1.5B
GPT-2 Small
124M
Synapses
> 10^6
10^9
10^12
10^13
10^15
Fruit Fly
Honey Bee
Mouse
Cat
Brain
A single node with multiple GPUs for models having a few Billion parameters
Multi node GPU clusters for models having Billions of parameters

Suppose that we have a single V100 GPU

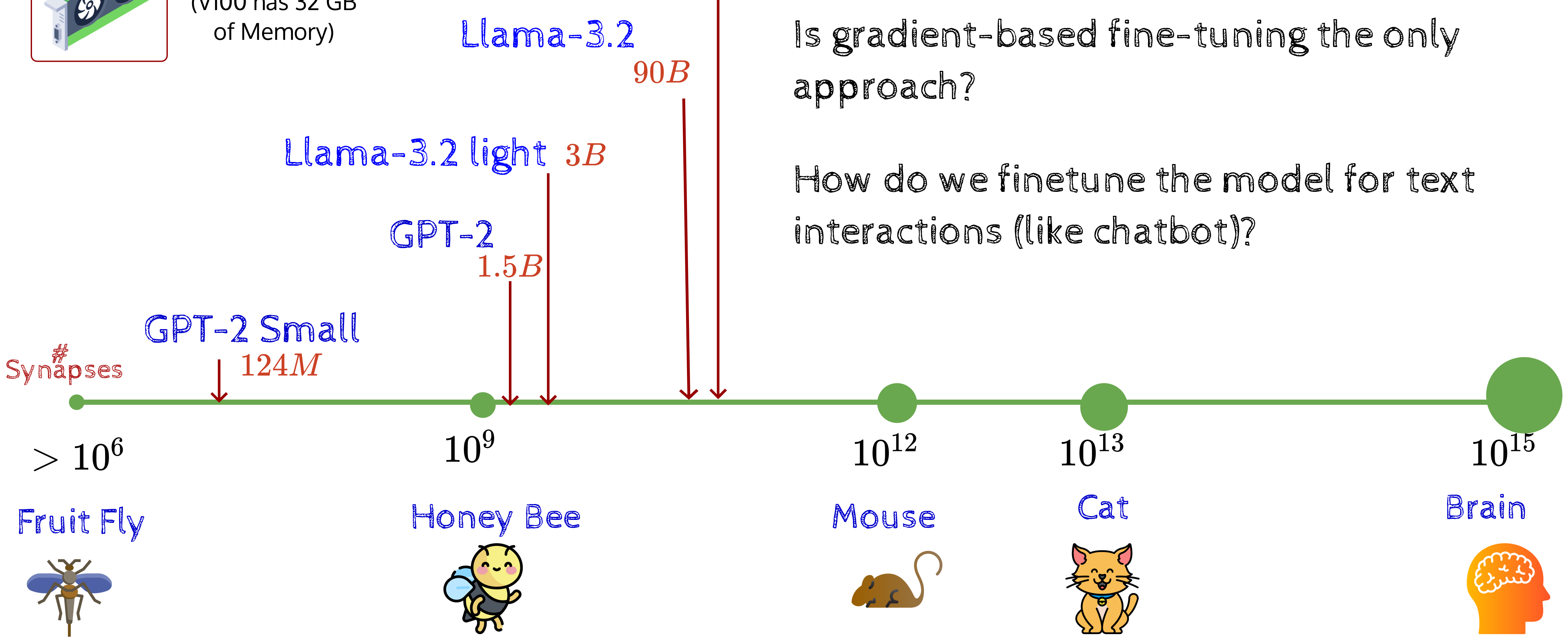
(V100 has 32 GB of Memory)

Can we at least fine-tune a 3B parameters model which requires about 48 GB of Memory?

If yes, what if we do not have enough samples for supervised fine-tuning?

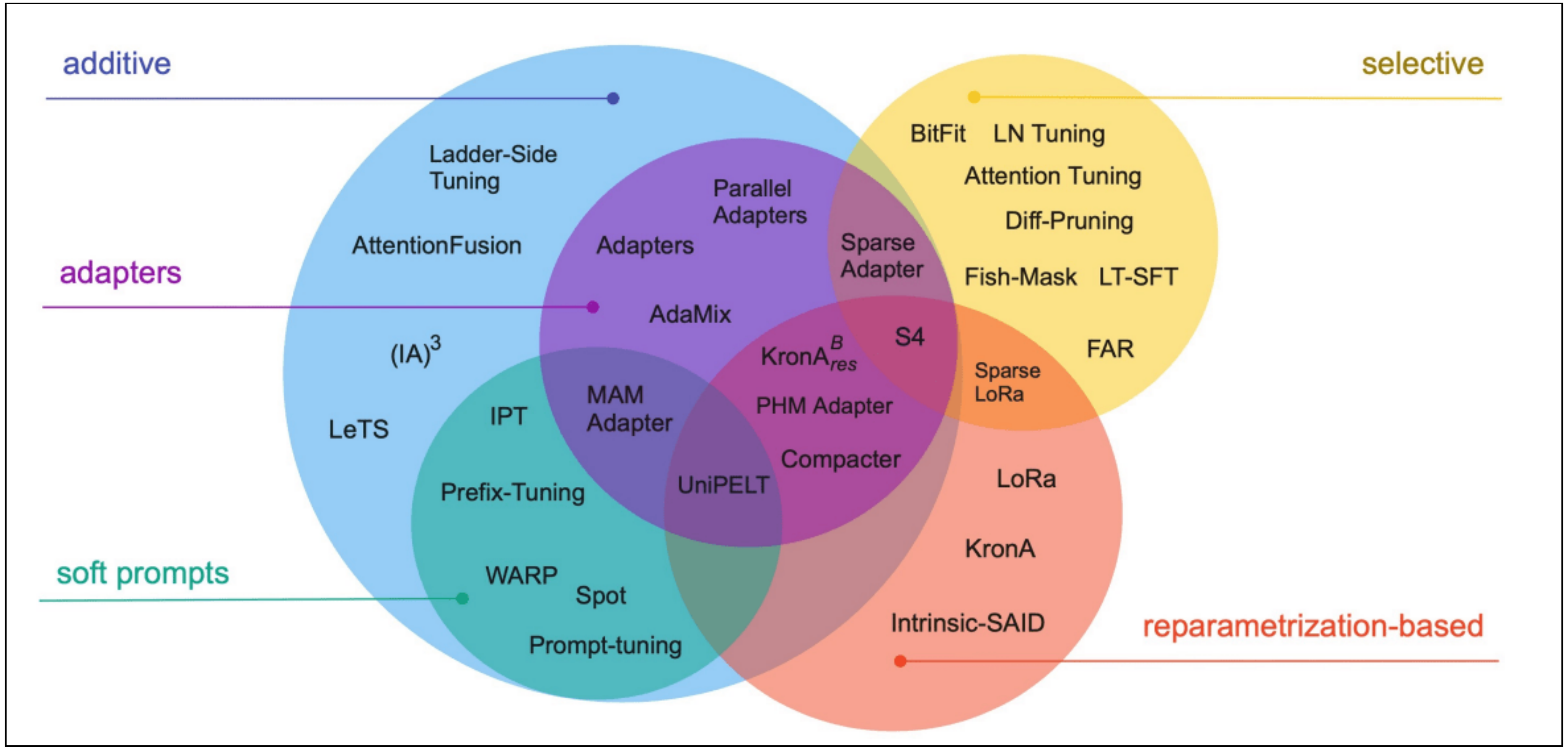
Is gradient-based fine-tuning the only approach?

How do we finetune the model for text interactions (like chatbot)?



Can we at least fine-tune 3B parameters models which require about 48 GB of Memory?

Many approaches have been developed to tackle the memory requirement to fine-tune large models.



Of these Parameter Efficient Fine Tuning (PEFT) techniques, LoRa, QLoRa and AdaLoRa are most commonly used (optionally, in combination with quantization)

Emerging Abilities

Fine-tuning large models is costly, as it still requires thousands of samples to perform well in a downstream task [Ref].

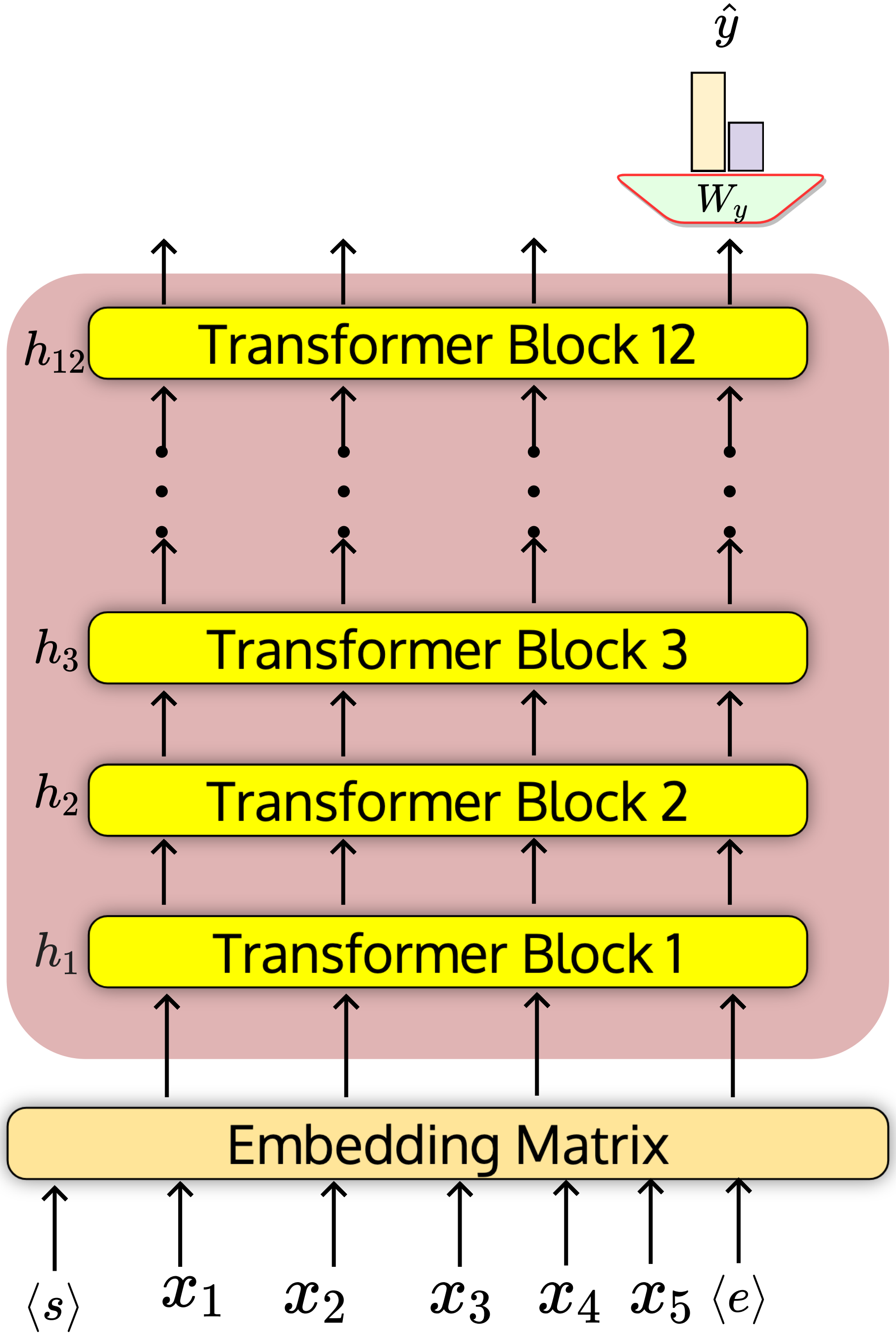
Some tasks may not have enough labelled samples

Moreover, this is not how humans adapt to different tasks once they understand the language.

We can give them a book to read and **"prompt"** them to summarize it or find an answer for a specific question.

That is, we do not need **"supervised fine-tuning"** at all (for most of the tasks)

In a nutshell, we want a single model that learns to do multiple tasks with zero or few examples (instead of thousands)!



Can we adapt a pre-trained language model for downstream tasks **without** any explicit supervision (called **zero-shot transfer**)?

Yes, with a simple tweak to the input text!

Note that, the prompts (or instructions) are words. Therefore,we just need to change the single task (LM) formulation

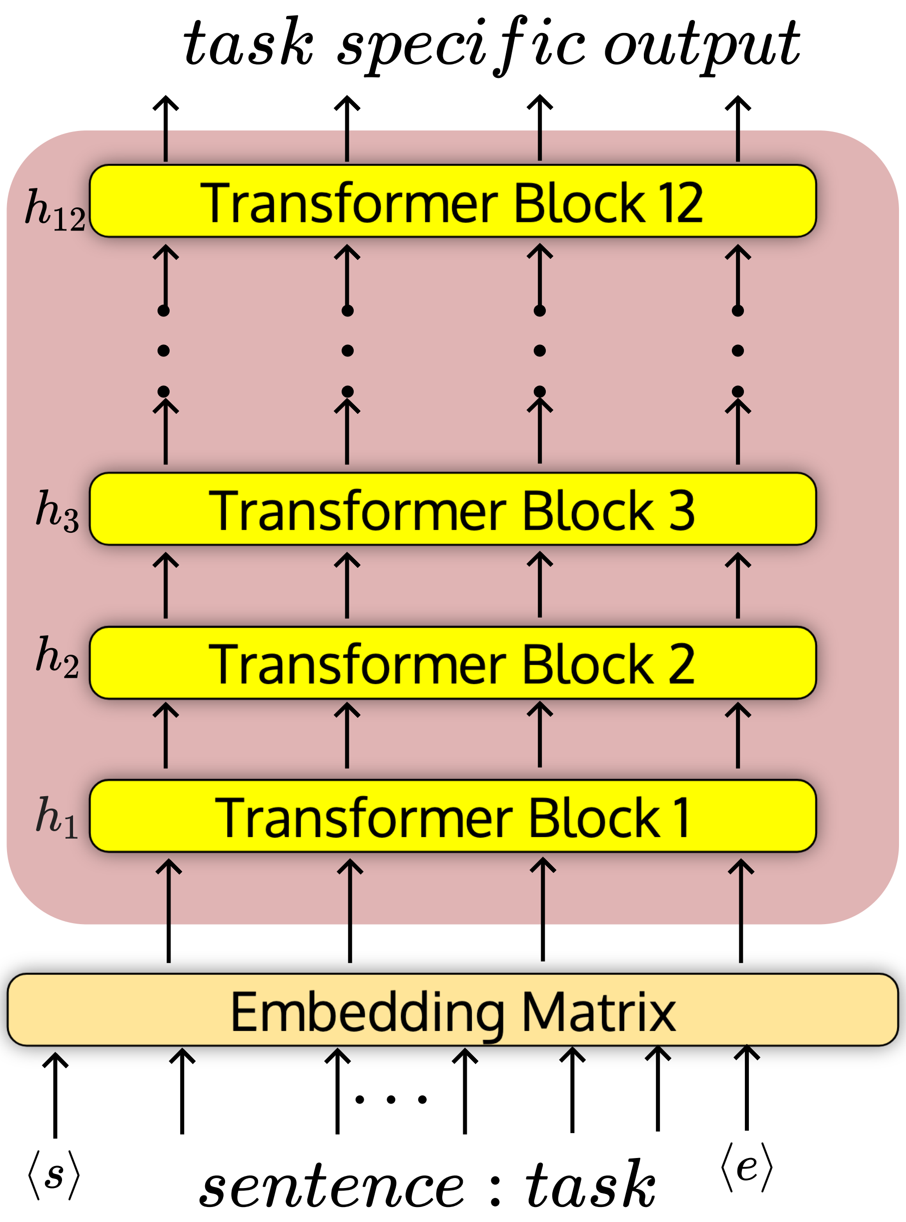
$$p(output|input)$$

to multi task

$$p(output|input, task)$$

where the **task** is just an instruction in plain words that is prepended (appended) to the input sequence **during inference**

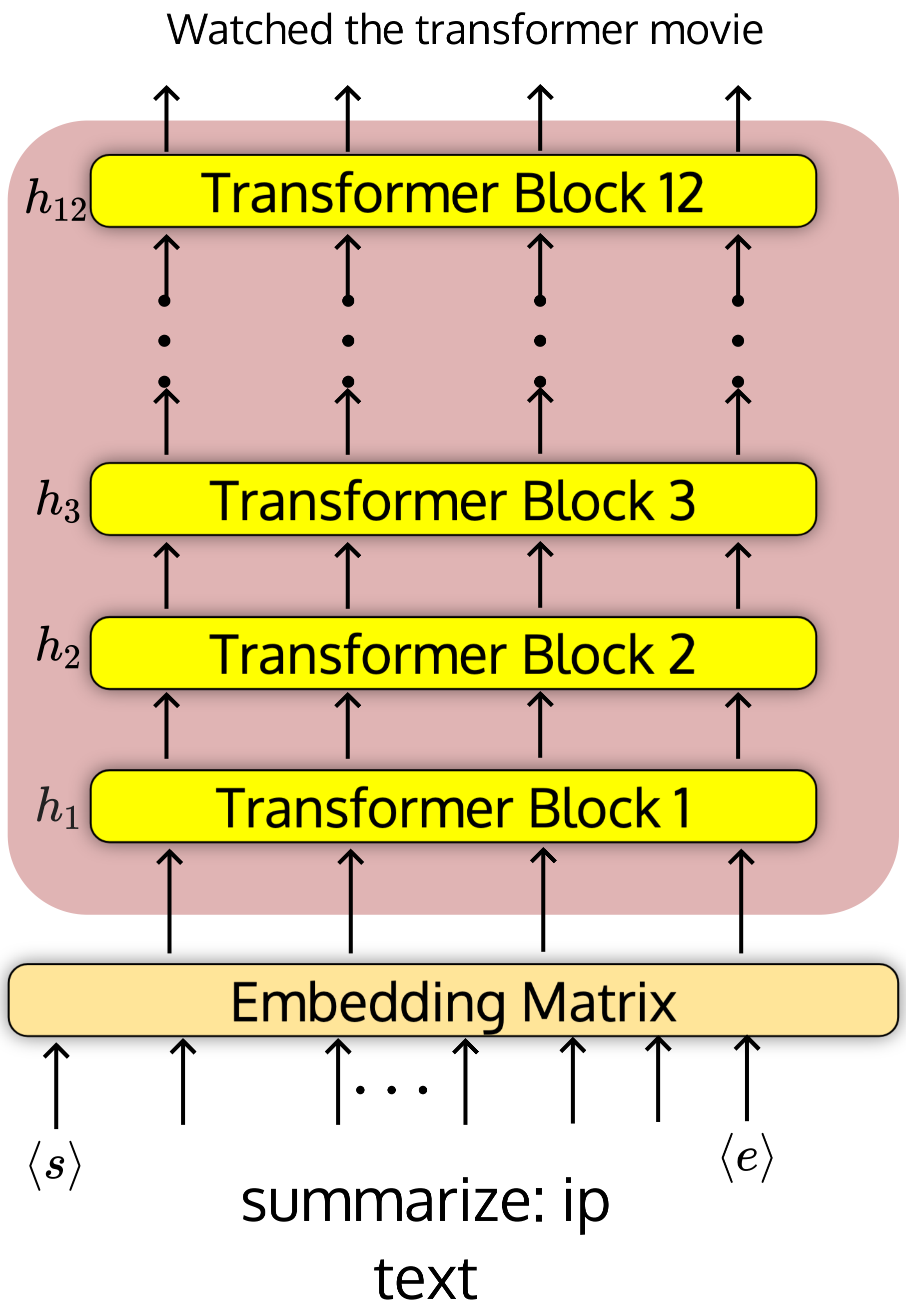
Surprisingly, this **induces** a model to output a task specific response for the same input



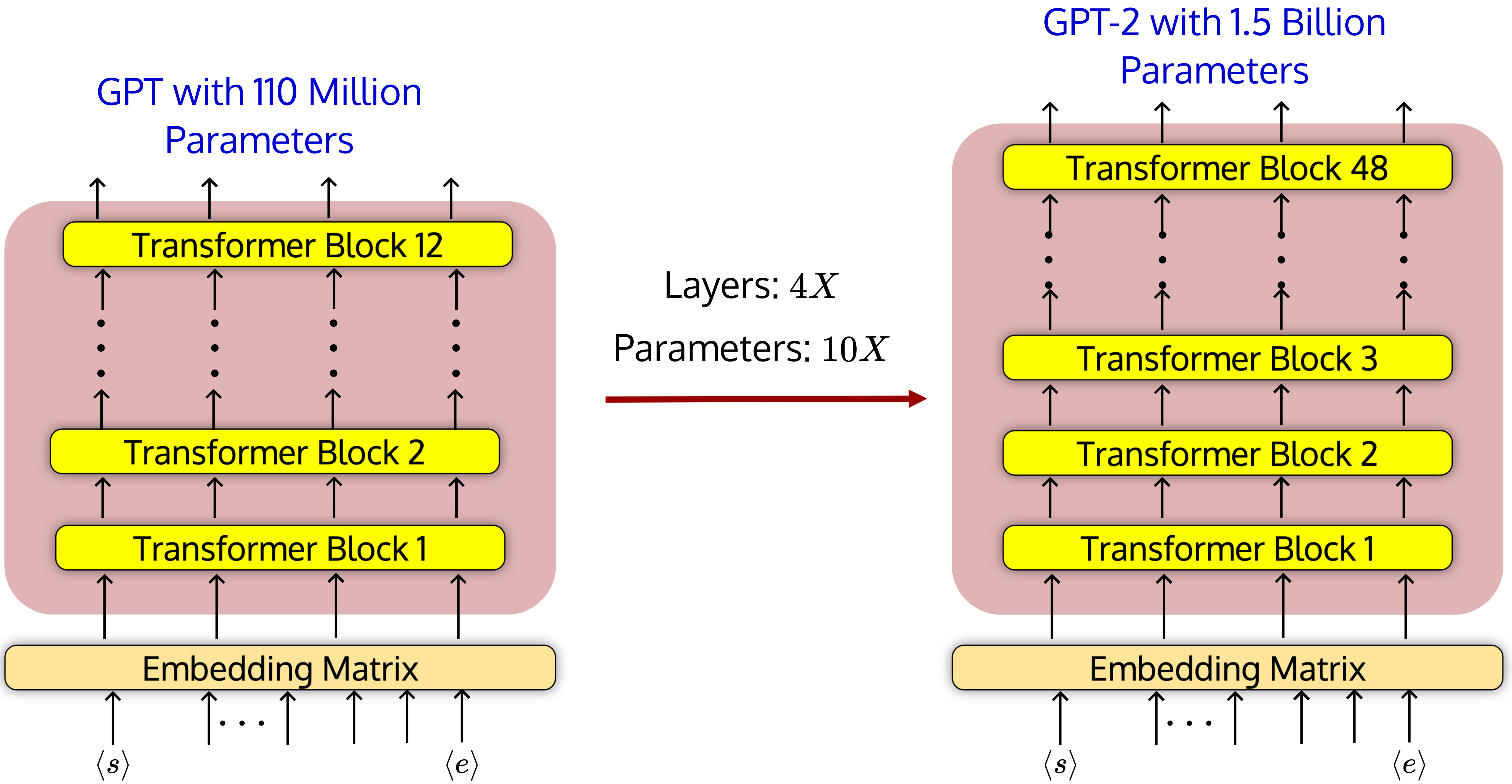
For example,

Task: summarize (or TL;DR:)

Input text: I enjoyed watching the movie transformer along with my .....



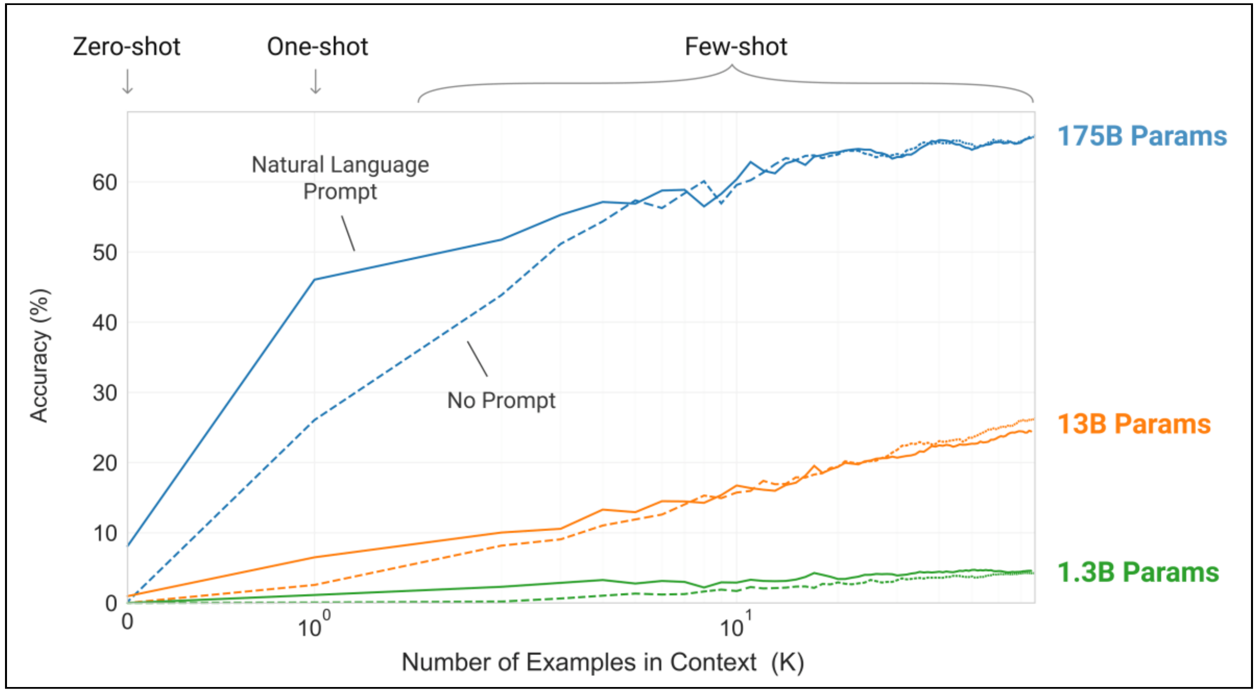
To get a good performance, we need to scale up both the model size and the data size



Pushing the limits: 1.5B to 175B

The ability to learn from a few examples improves as model size increases

For certain tasks, the performance is comparable to that achieved through full task-specific fine-tuning.



Since the model learns a new task from samples within the context window, this approach is called 'in-context' learning.

This remarkable ability enables the adaptation of the model to downstream tasks without the need for gradient-based fine-tuning.

Adaptation happens on-the-fly in inference mode (which consumes far less memory)

Prompting

Since adaptation occurs during inference, there is no need to share model weights for fine-tuning.

This approach enables the model's deployment across a variety of use cases through simple API calls (making it more accessible)

This new ability paved the way for fine-tuning the model for specific tasks by Prompting

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

Text: I enjoyed watching the transformers movie.
Sentiment:
```

```
positive
```

There are many ways of prompting the model. For example,

1. Zero-shot
2. Few-shot (in-context)
3. Chain of Thought
4. Prompt Chaining

However, there is a catch

```
guide the user to reach the destination

Text:how to reach Marina Beach from IIT
Madras
```

```
by train
```

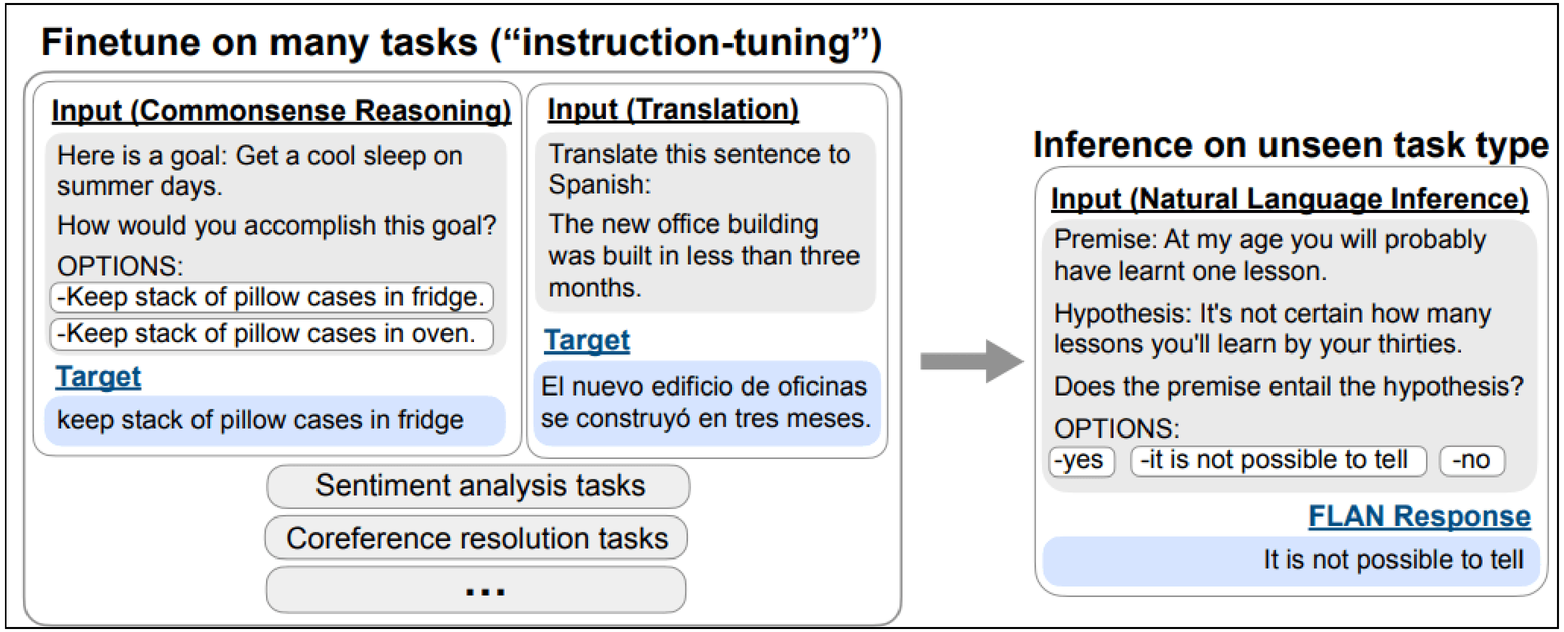
The model is unable to follow the user's intent and instead just completes the text coherently

Instruction Tuning

Zero-shot learning performance is often poor, despite the model size (say, GPT 3 175B), especially in following the user intent (instructions).

How do we improve the zero-shot learning performance?

Fine-tune the model on the instructions (one approach is to reformat the available datasets into instruction sets)



Refer to the FLAN paper for more details

Preference Tuning via RLHF

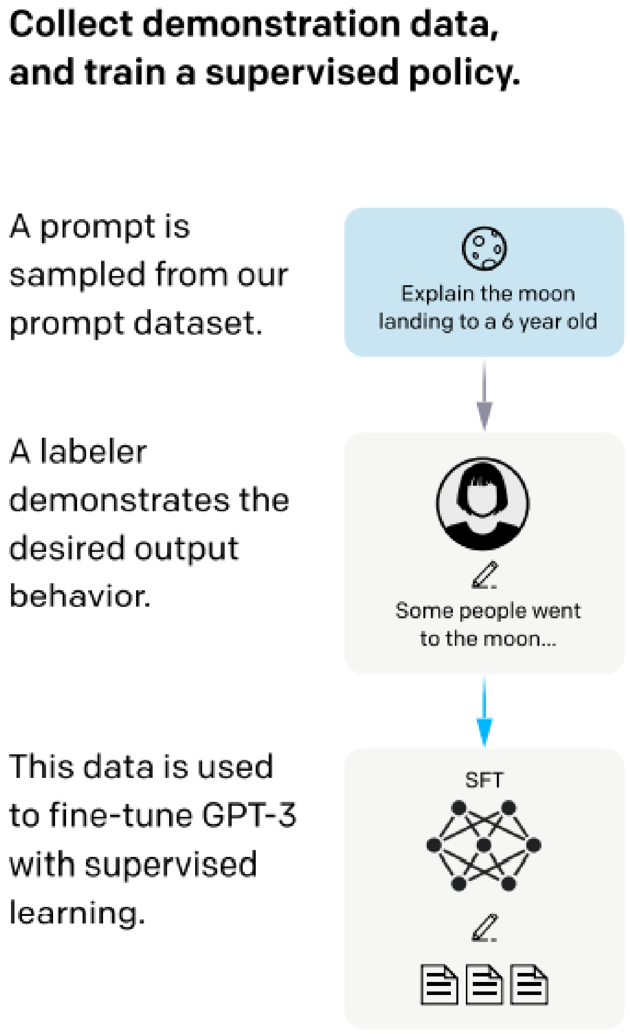
"Making language models bigger does not inherently make them better at following a user's intent." -[Instruct GPT paper]

Language Modelling objective is "misaligned" with user intent

Therefore, we must fine-tune the model to align with the user intent.

This requires human labelled demonstrations for a collection of prompts (a labour intensive task)

Use these collections to fine-tune (Supervised Fine Tuning , SFT) the model using Reinforcement Leanring from Human Feedback (RLHF)



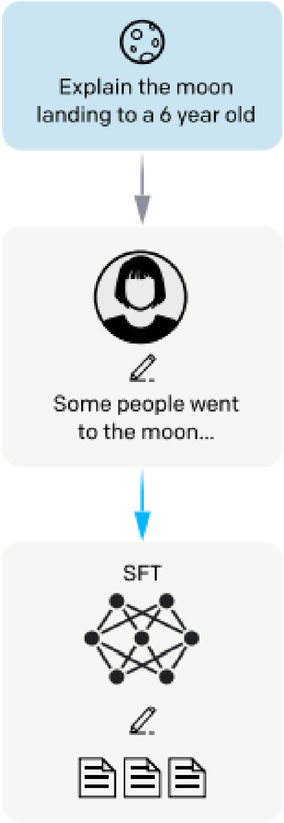
Preference Tuning via RLHF

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

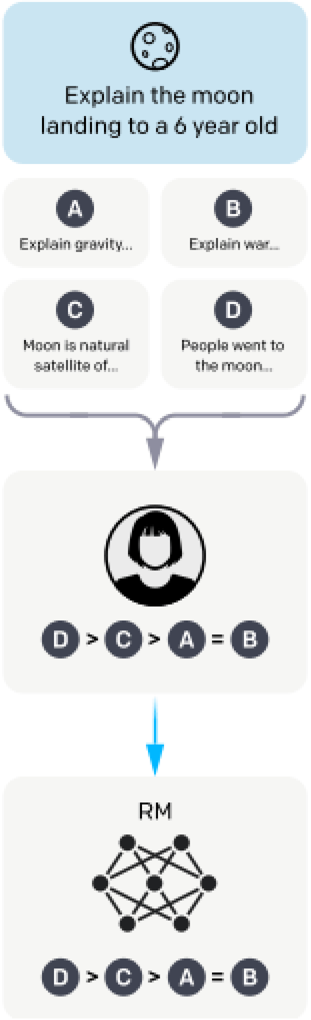


Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



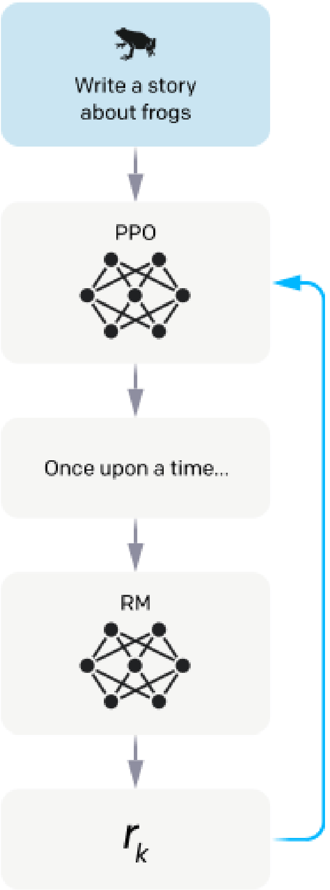
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



An imagined example

Text: Guide me on how to reach Marina Beach from IIT Madras → GPT → by train

Text: Guide me on how to reach Marina Beach from IIT Madras → Instruct GPT →

To reach **Marina Beach** from **IIT Madras**, here's a step-by-step guide:

By Public Transport (Bus):

1. **Start from IIT Madras Main Gate (Sardar Patel Road)**:
   - Exit through the **IIT Main Gate** on Sardar Patel Road.
2. **Take a Bus to Anna Square/MGR Memorial**:
   - You can take a bus from the bus stop near the Main Gate.
   - Look for buses heading towards **Anna Square**, **MGR Memorial**, or **Marina Beach** (Route numbers may include 21G, 21L, 19B, etc.).
   - Ask the conductor to confirm if it stops at **Marina Beach**.
   - It should take approximately 30-45 minutes, depending on traffic.

*Actual response from ChatGPT

Suppose that we have a single V100 GPU

(V100 has 32 GB of Memory)

Can we at least fine-tune 3B parameters models which require about 48 GB of Memory?

Yes

If yes, what if we do not have enough samples for supervised fine-tuning?

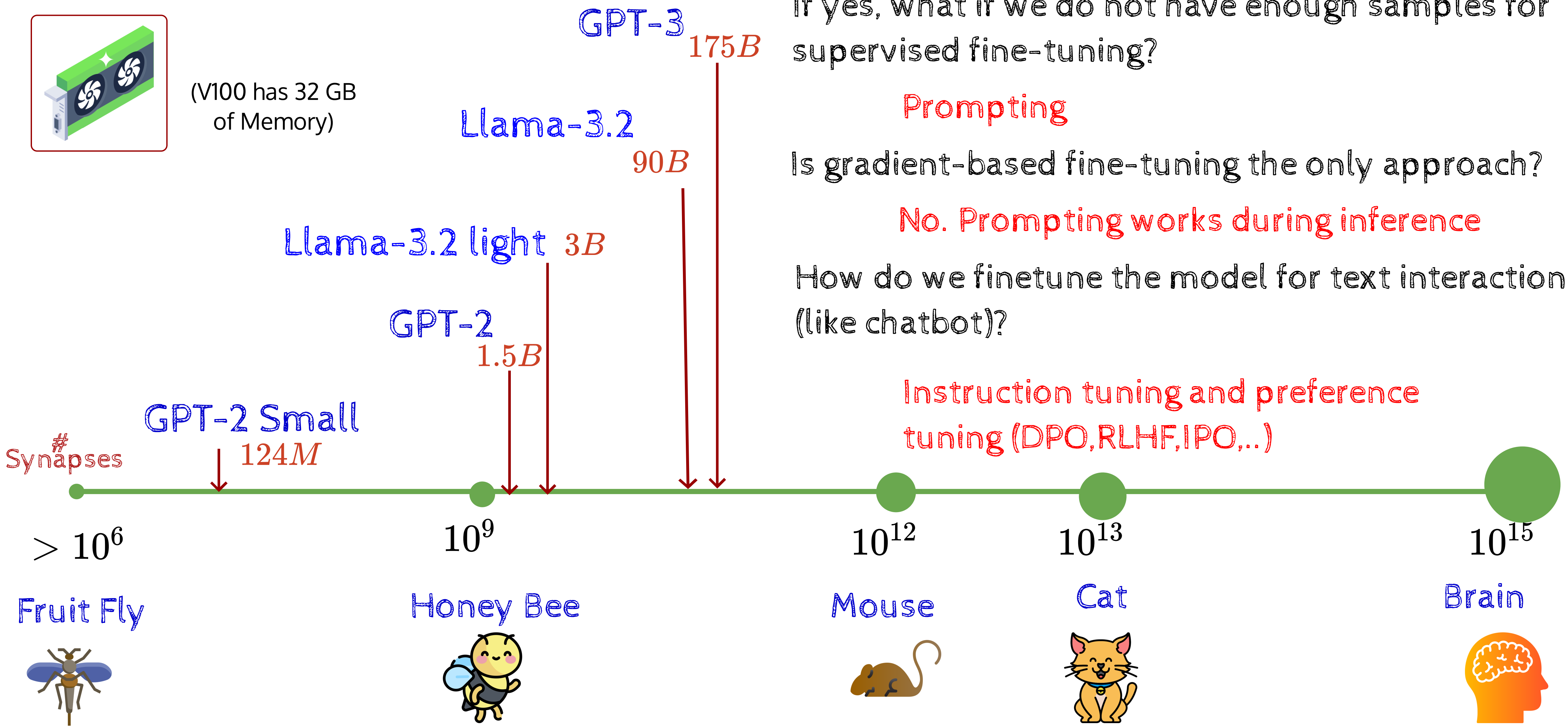
Prompting

Is gradient-based fine-tuning the only approach?

No. Prompting works during inference

How do we finetune the model for text interactions (like chatbot)?

Instruction tuning and preference tuning (DPO,RLHF,IPO,..)



Now we try grouping the approaches broadly into two categories

Fine-tuning

Supervised Fine-tuning (SFT)

Task-specific Full Fine-tuning

(task agnostic) Instruction-tuning

Preference Tuning: RLHF, DPO, IPO, KTO..

Memory efficient fine-tuning*

1. PEFT (Parameter Efficient Fine-tuning): LoRA, QLoRA, AdaLora

2. Quantization

*general techniques to reduce memory requirement, suitable for any fine-tuning schemes

Prompt tuning

Zero-shot,

few-shot

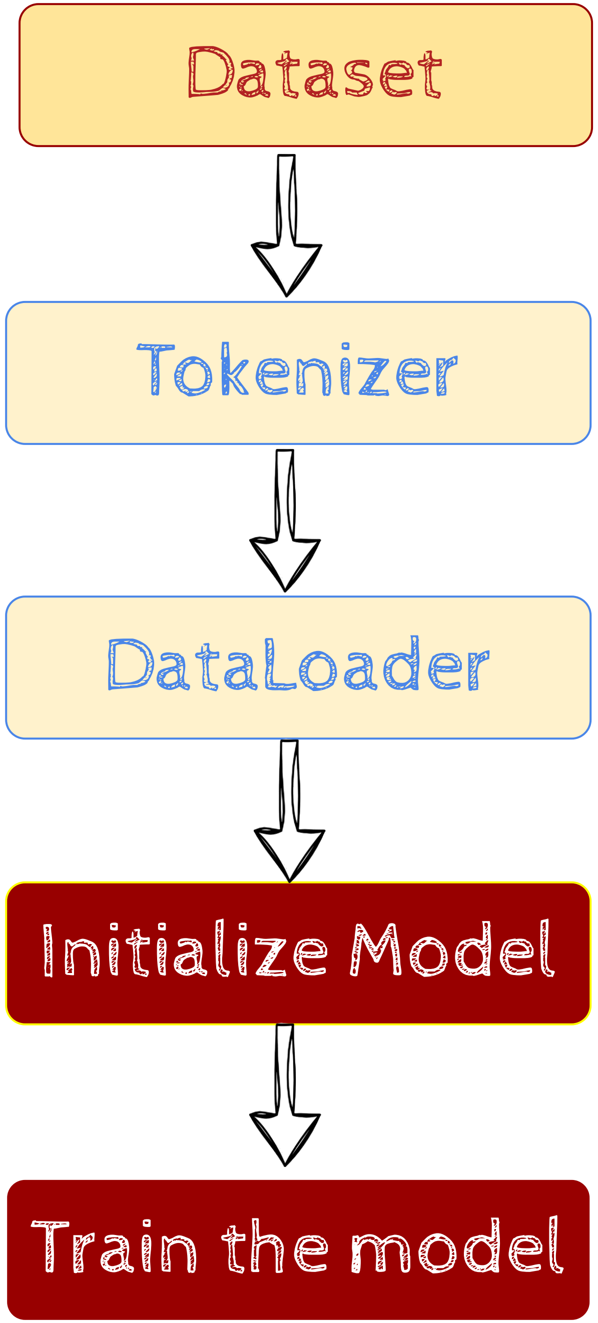
Chain of Thought

Prompt chaining

Meta prompting

⋮

The list of modules we used so far




```python
from datasets import load_dataset
```

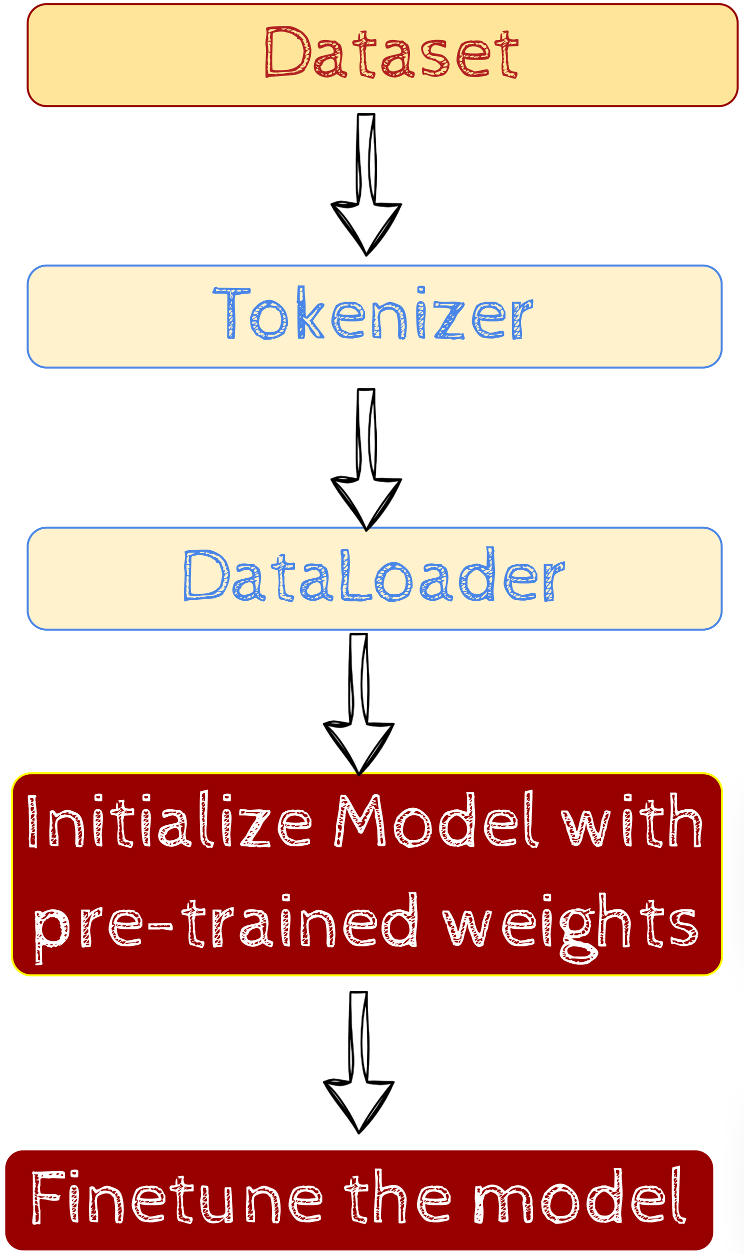
```python
from transformers import AutoTokenizer
```

```python
from transformers import DataCollatorForLanguageModeling
```

```python
from transformers import GPT2Config, GPT2LMHeadModel
```

```python
from transformers import TrainingArguments, Trainer
```

Let's dive in



Additional Modules:

1. peft,
2. trl,SFTTrainer (for preference tuning)
3. bitsandbytes (quantization)
4. Unsloth (for single-gpu, 2.5x faster training)

```
from transformers import GPT2ForSequenceClassification
model = GPT2ForSequenceClassification.from_pretrained()
```

```
from transformers import TrainingArguments, Trainer
from peft import LoraConfig
```