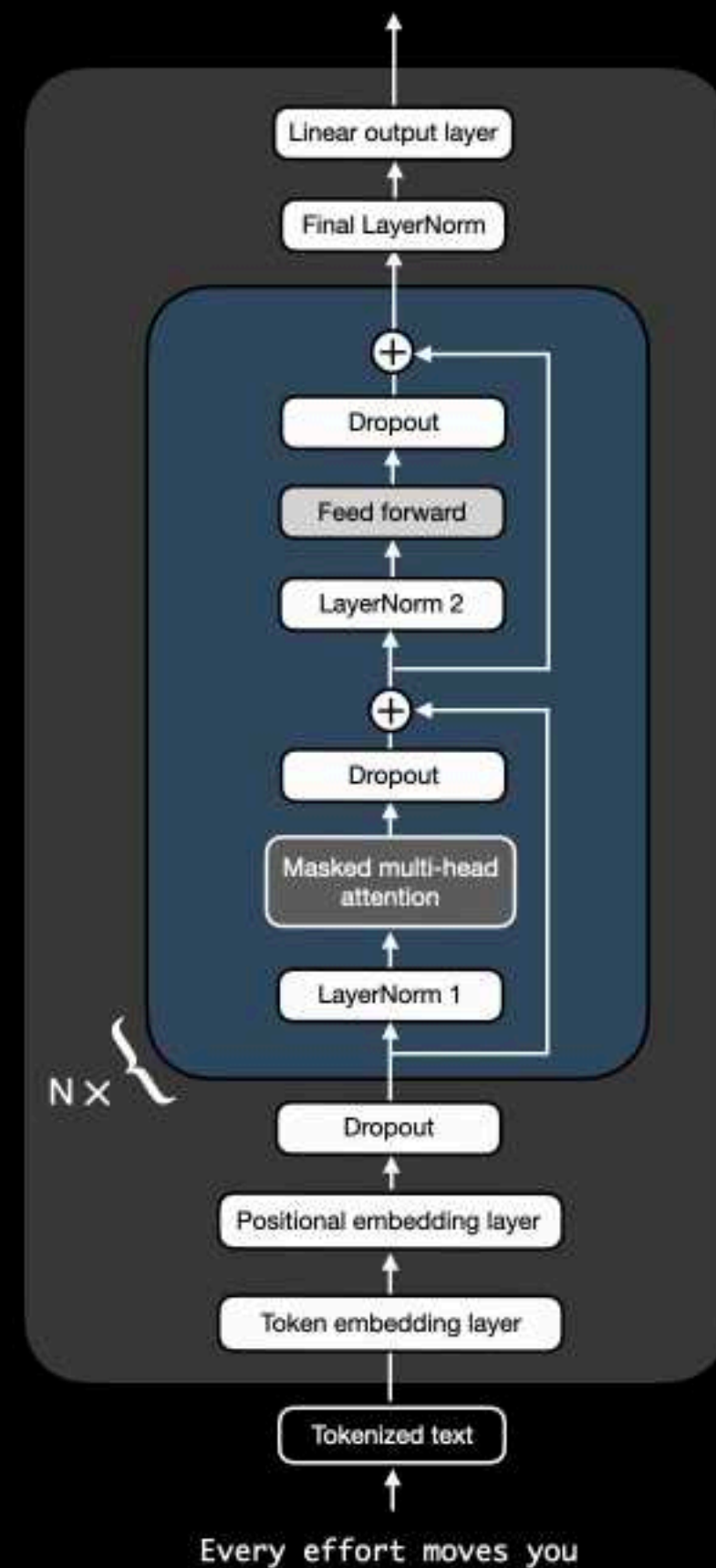


# Developing an LLM: Building, Training, Finetuning



Building an LLM

Foundation model

Classifier

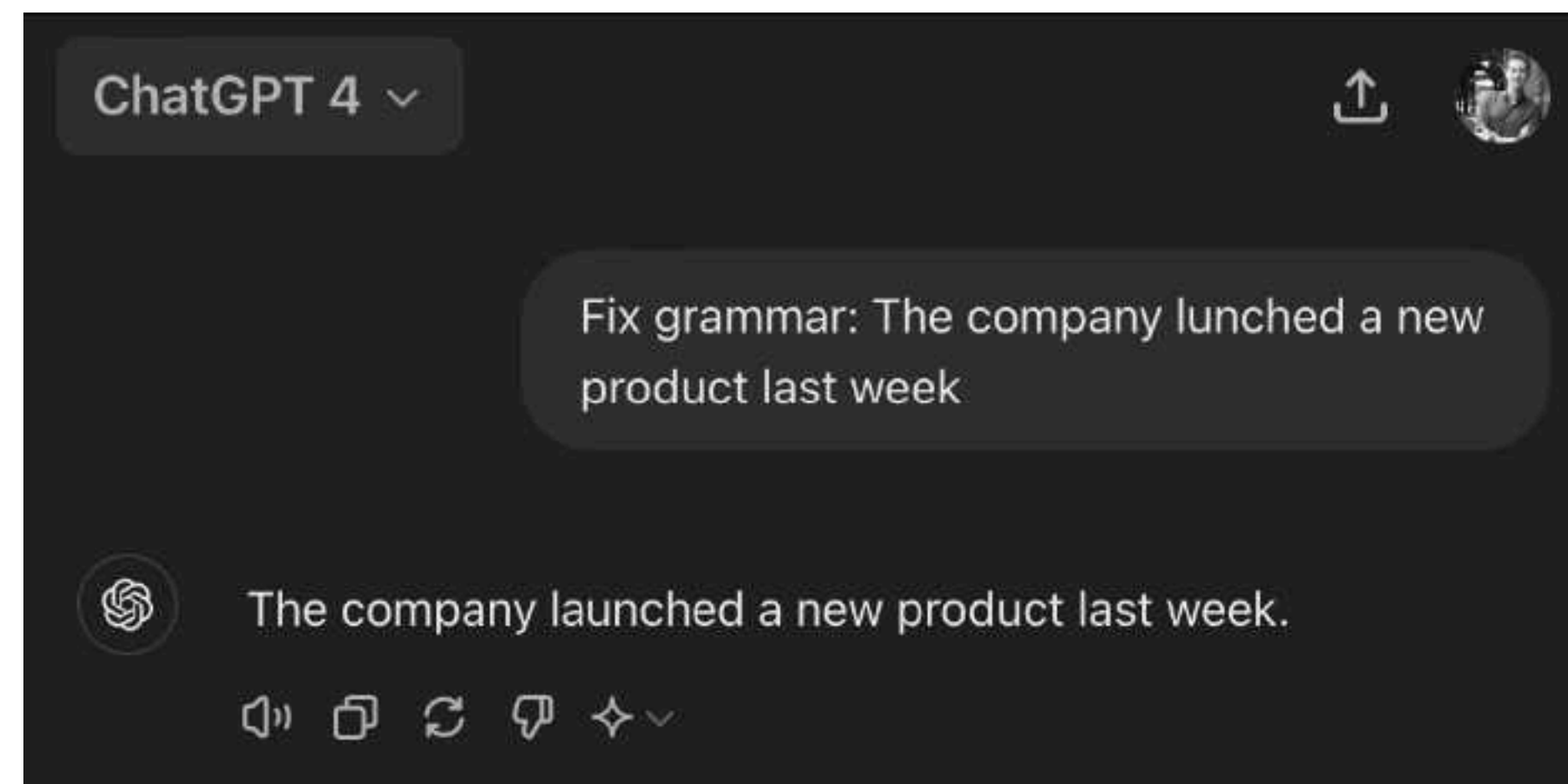
Personal assistant

Dataset with class labels

Instruction dataset

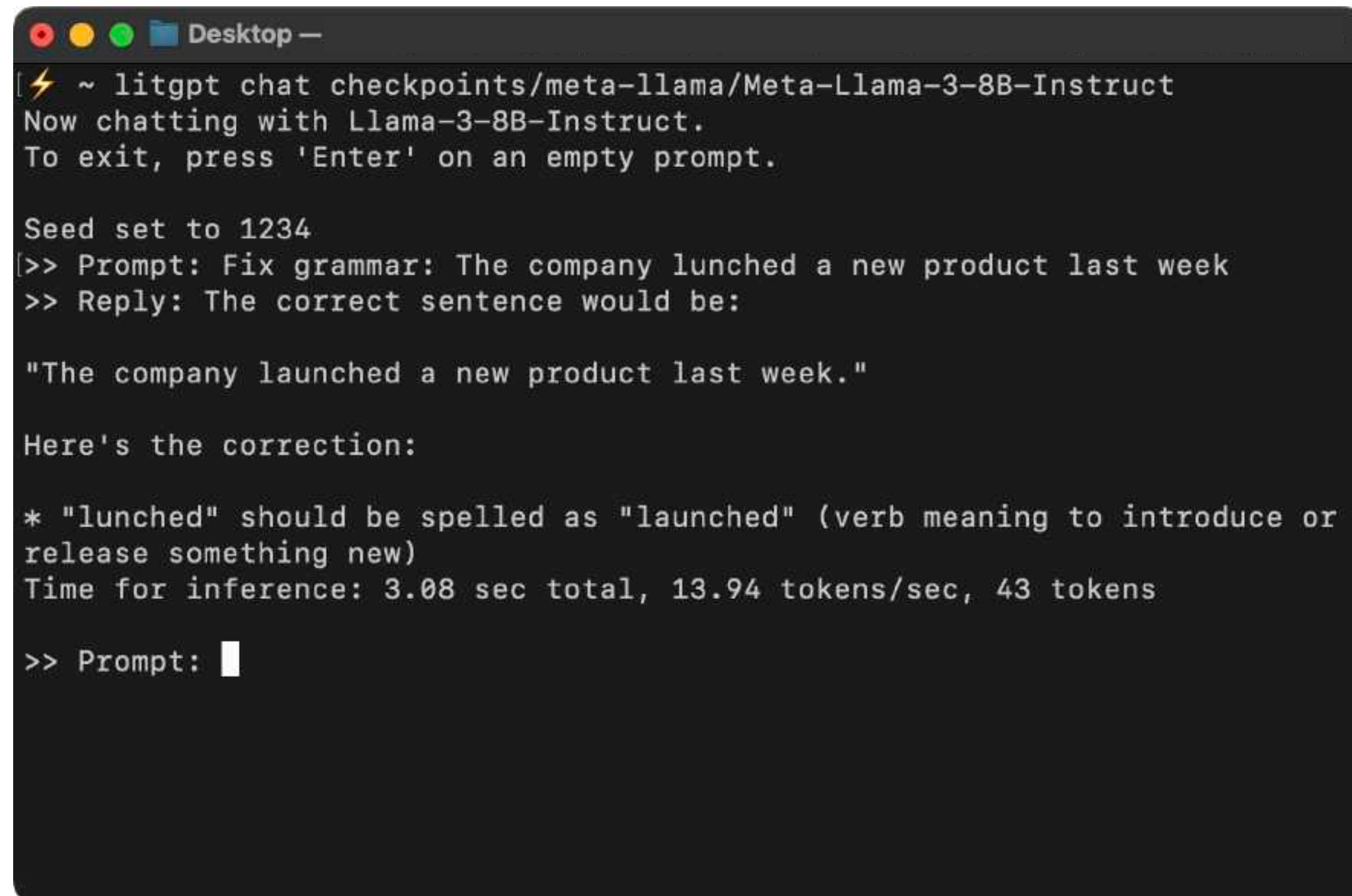
# Using Large Language Models (LLMs)

# Using Large Language Models (LLMs)



1) Via public & proprietary services

# Using Large Language Models (LLMs)

A terminal window titled "Desktop" showing the execution of the litgpt chat command. The prompt is "Fix grammar: The company lunched a new product last week". The model's reply is "The correct sentence would be: 'The company launched a new product last week.'". It also provides a correction: "\* 'lunched' should be spelled as 'launched' (verb meaning to introduce or release something new)". The inference time is 3.08 seconds for 43 tokens.

```
Desktop —
[⚡ ~ litgpt chat checkpoints/meta-llama/Meta-Llama-3-8B-Instruct
Now chatting with Llama-3-8B-Instruct.
To exit, press 'Enter' on an empty prompt.

Seed set to 1234
[>> Prompt: Fix grammar: The company lunched a new product last week
>> Reply: The correct sentence would be:

"The company launched a new product last week."

Here's the correction:

* "lunched" should be spelled as "launched" (verb meaning to introduce or
release something new)
Time for inference: 3.08 sec total, 13.94 tokens/sec, 43 tokens

>> Prompt: █
```

## 2) Running a (custom) LLM locally

<https://github.com/Lightning-AI/litgpt>

# Using Large Language Models (LLMs)

## 3) Deploying a (custom) LLM

and using an LLM via a private API

<https://lightning.ai/lightning-ai/studios/litgpt-serve>

```
Desktop —
~ litgpt serve checkpoints/meta-llama/Meta-Llama-3-8B-Instruct
File '/home/zeus/miniconda3/envs/cloudspace/lib/python3.10/site-packages/litserve/python_client.py' copied to '/teamspace/studios/this_studio/client.py'
INFO: Started server process [56909]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: 127.0.0.1:50206 - "POST /predict HTTP/1.1" 200 OK
INFO: 127.0.0.1:50228 - "POST /predict HTTP/1.1" 200 OK

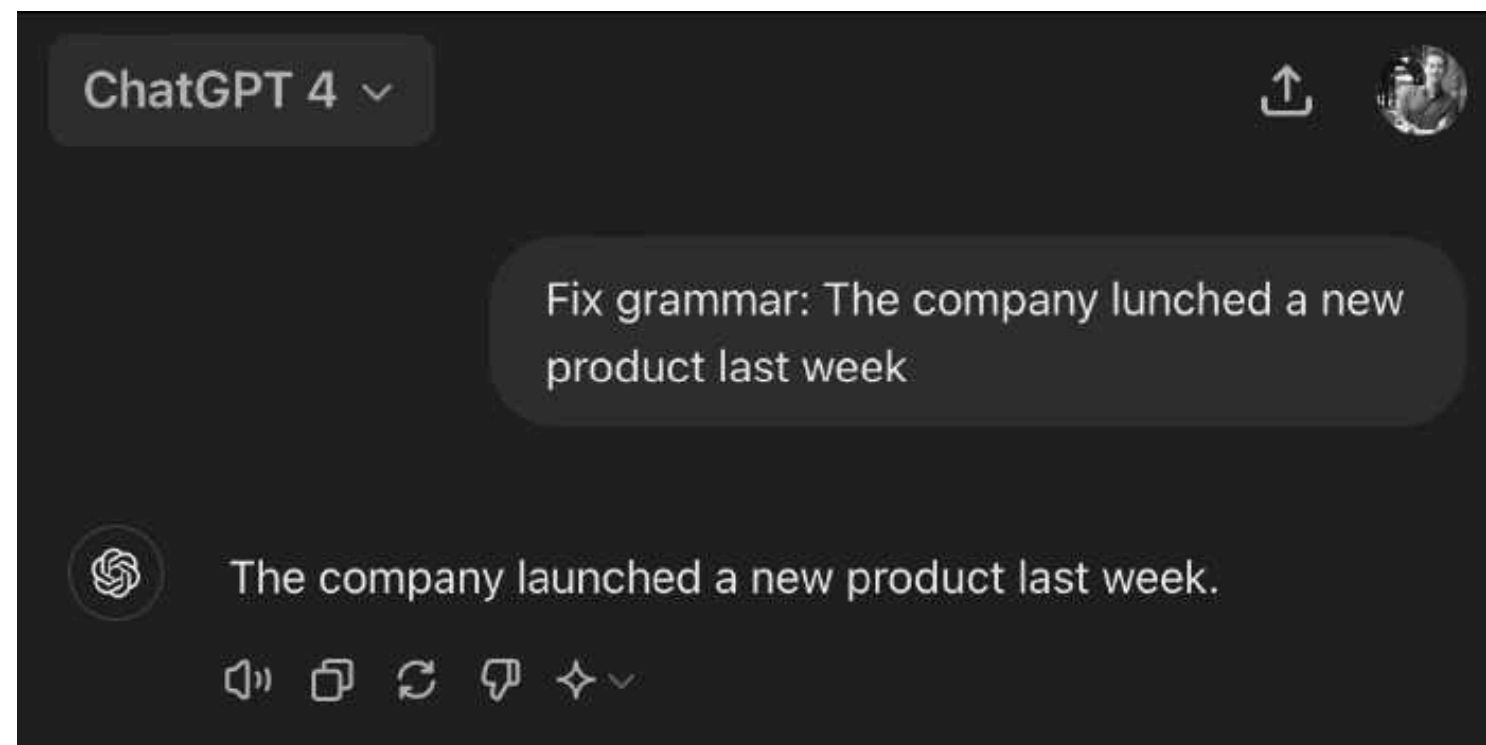
In [1]: import requests, json

In [2]: response = requests.post(
...:     "http://127.0.0.1:8000/predict",
...:     json={"prompt": "Fix grammar: The company lunched a new
...: product last week"}
...: )

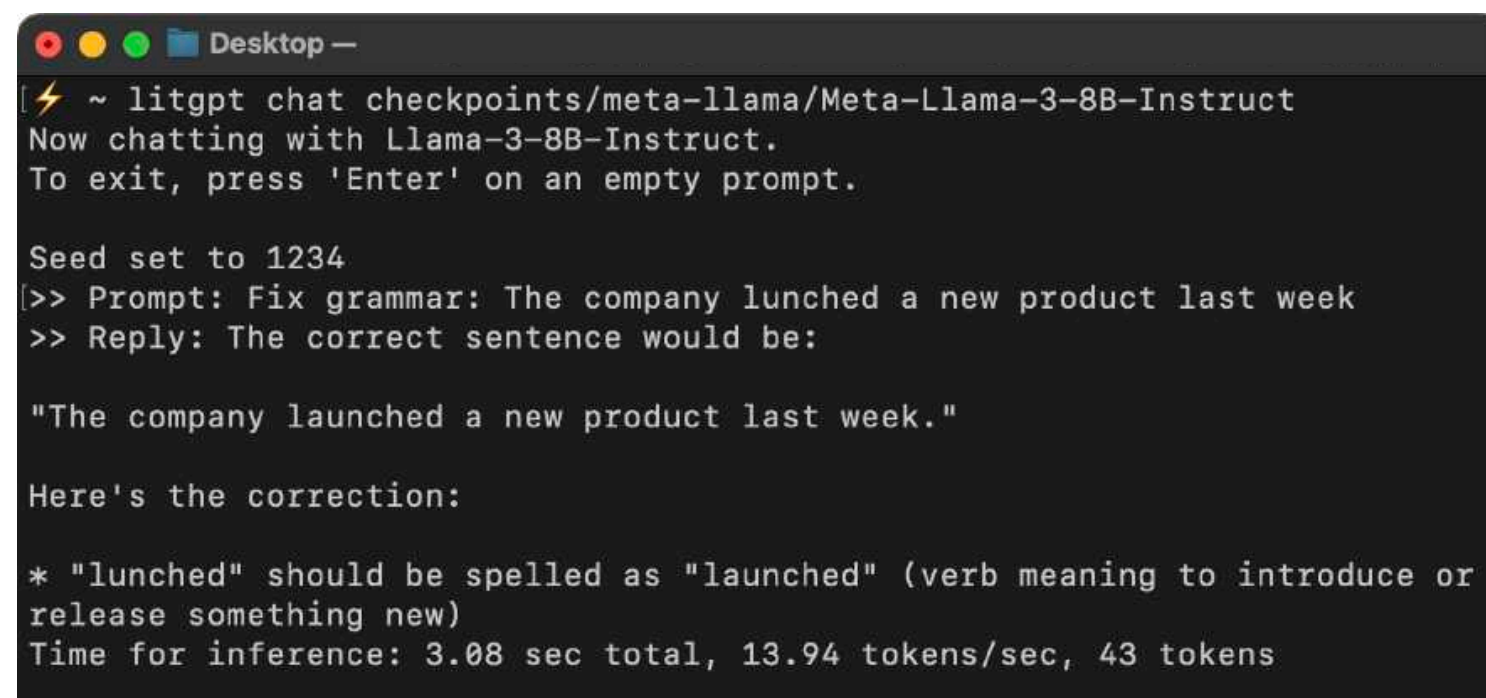
In [3]: print(response.json()["output"])
system
You are a helpful assistant.
user
Fix grammar: The company lunched a new product last week
assistant
The correct grammar is:
The company launched a new product last week.
The verb "lunched" is incorrect, and the correct verb to use in this context is "launched".

In [4]:
```

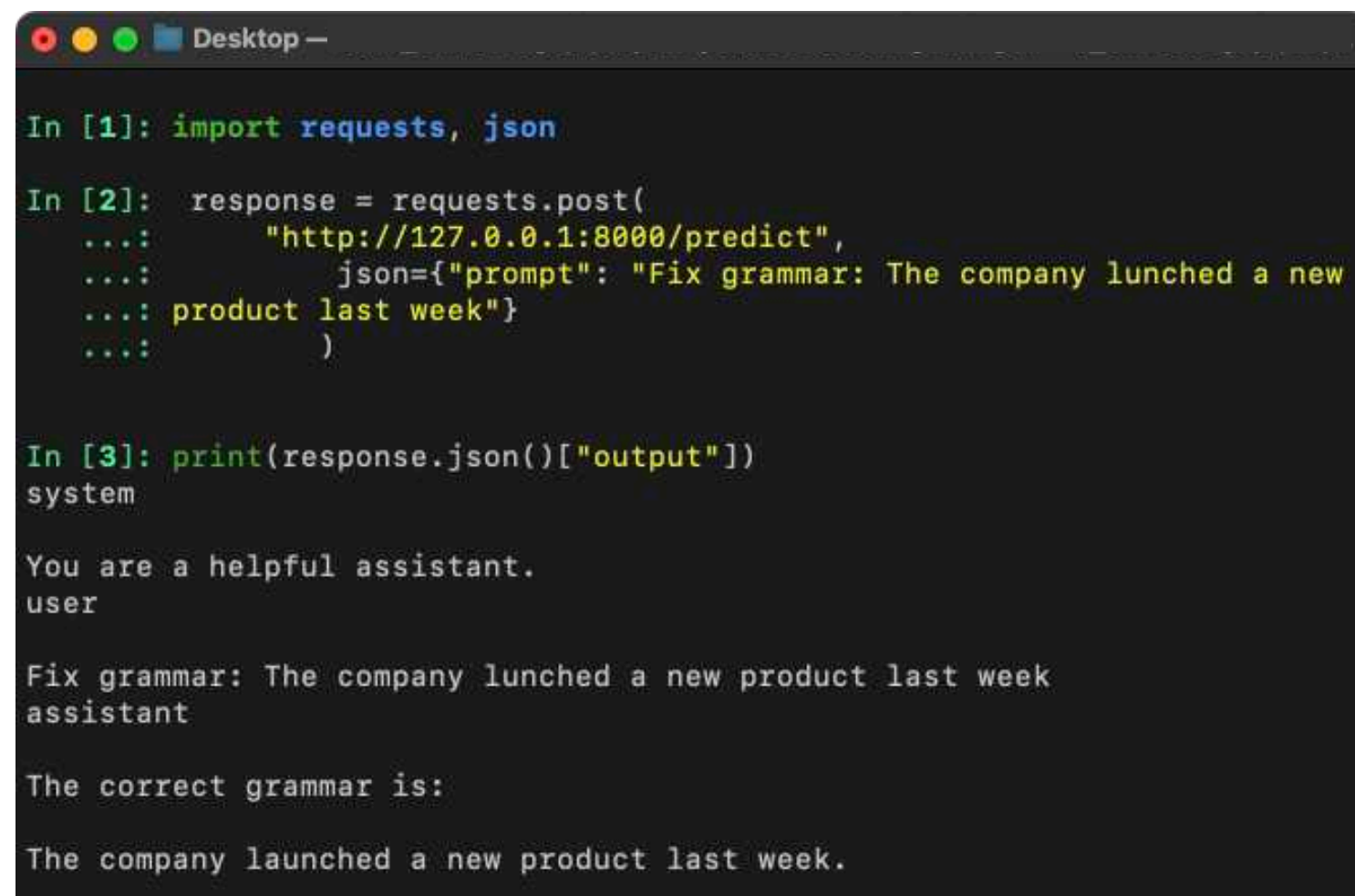




1) Via public & proprietary services



2) Running a (custom) LLM locally



3) Deploying a (custom) LLM  
& using an LLM via a private API

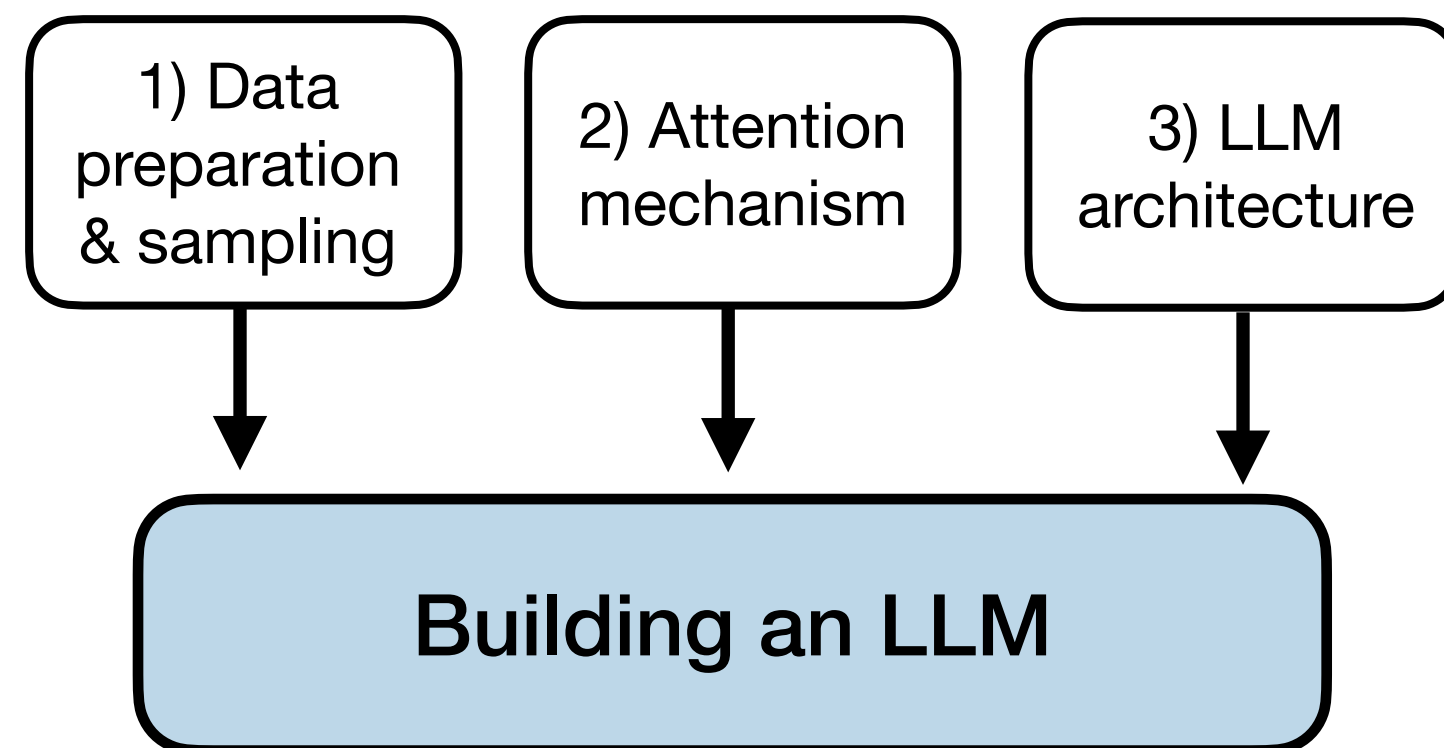
Different use cases &  
trade-offs

(I use all of them)

# **What goes into developing an LLM like this?**

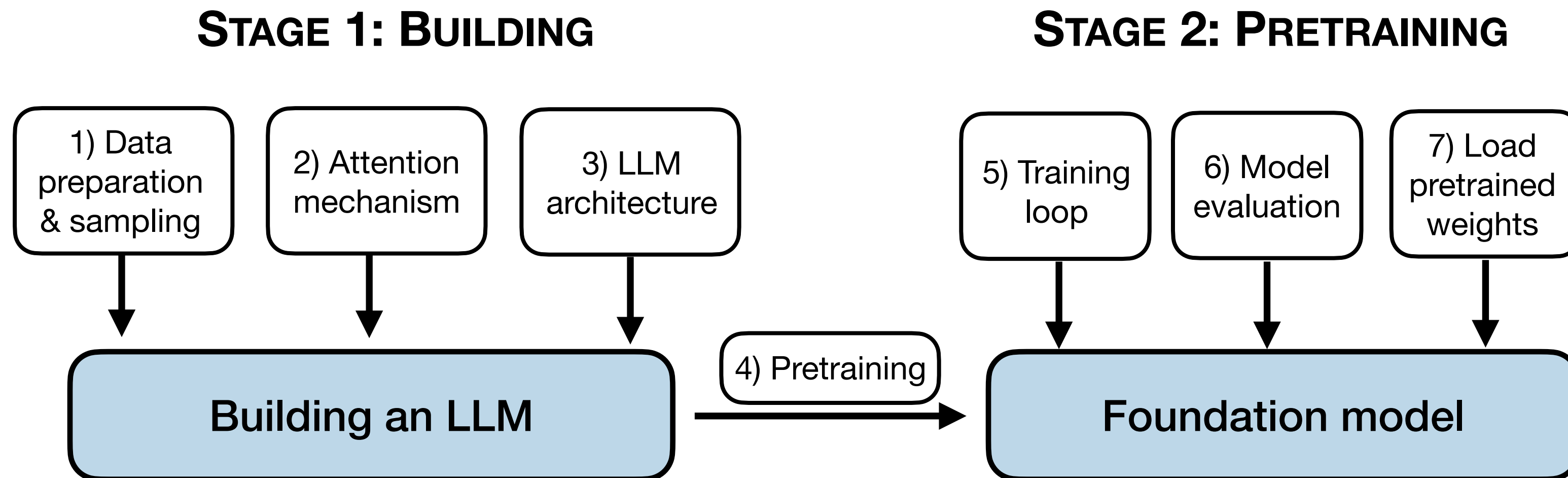
# Developing an LLM

## STAGE 1: BUILDING

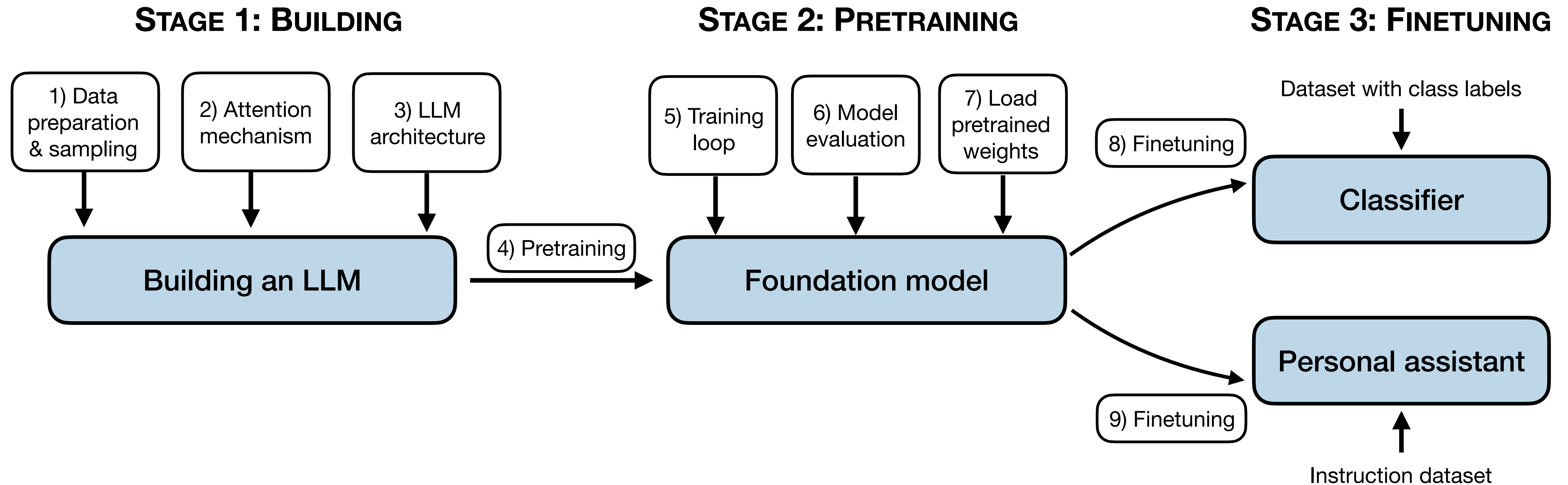


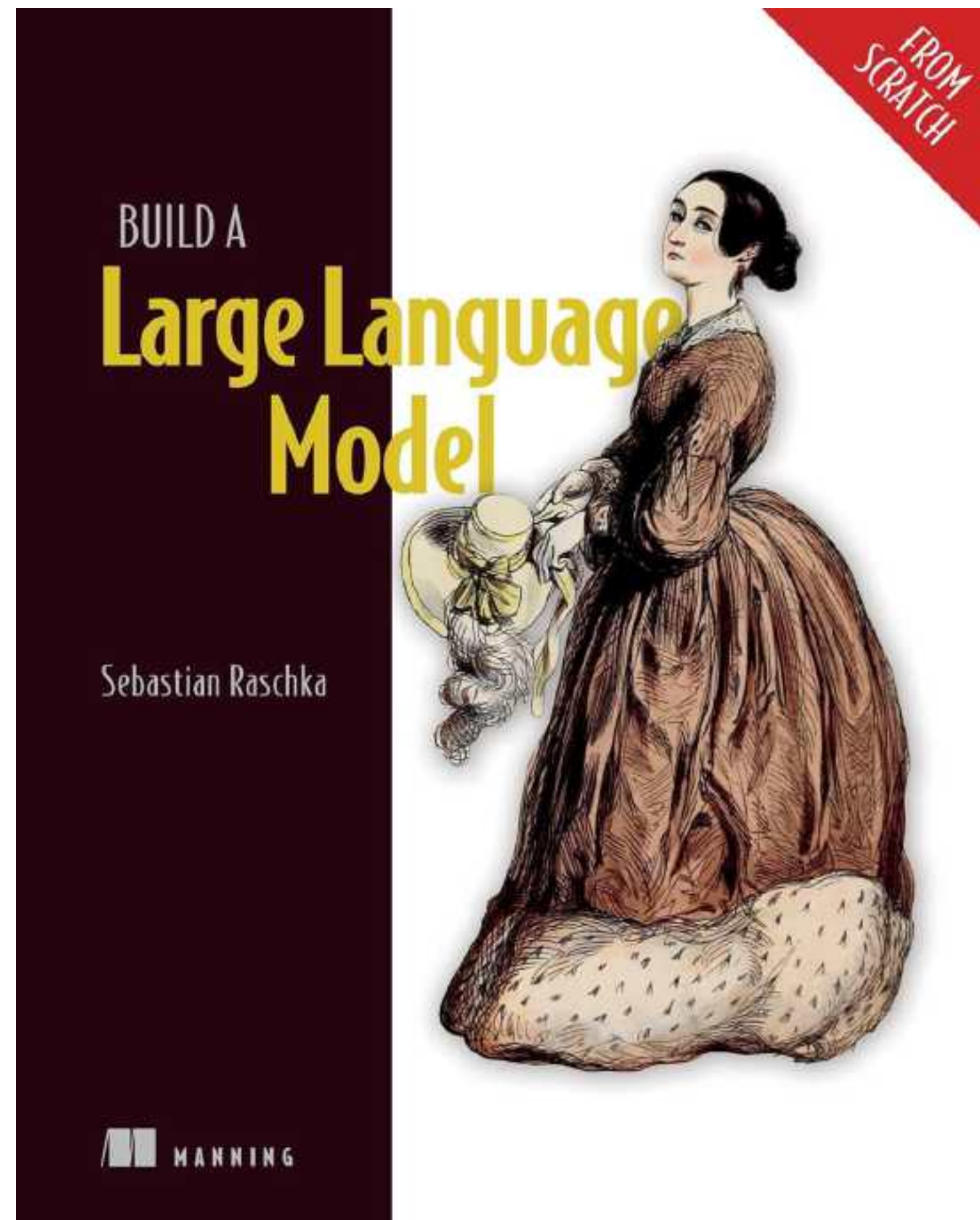


# Developing an LLM



# Developing an LLM





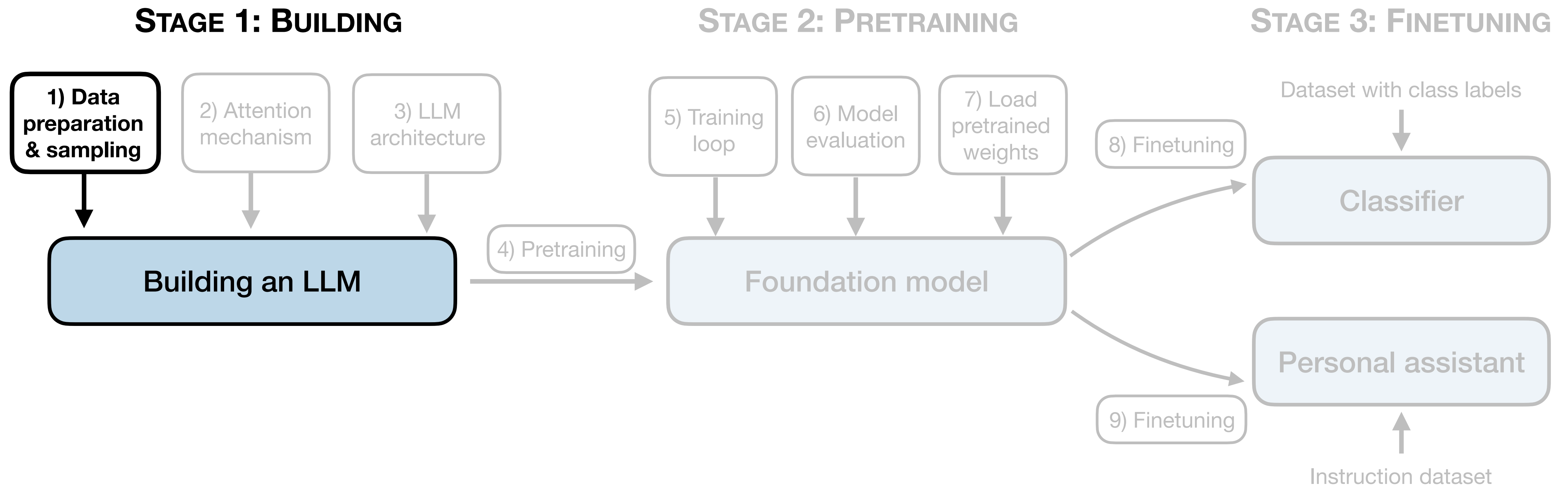
<https://mng.bz/M96o>

<https://github.com/rasbt/LLMs-from-scratch>

**(Most figure source)**

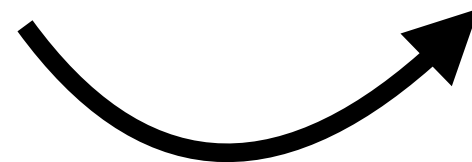
# Stage 1: Building

# Let's start with the dataset!





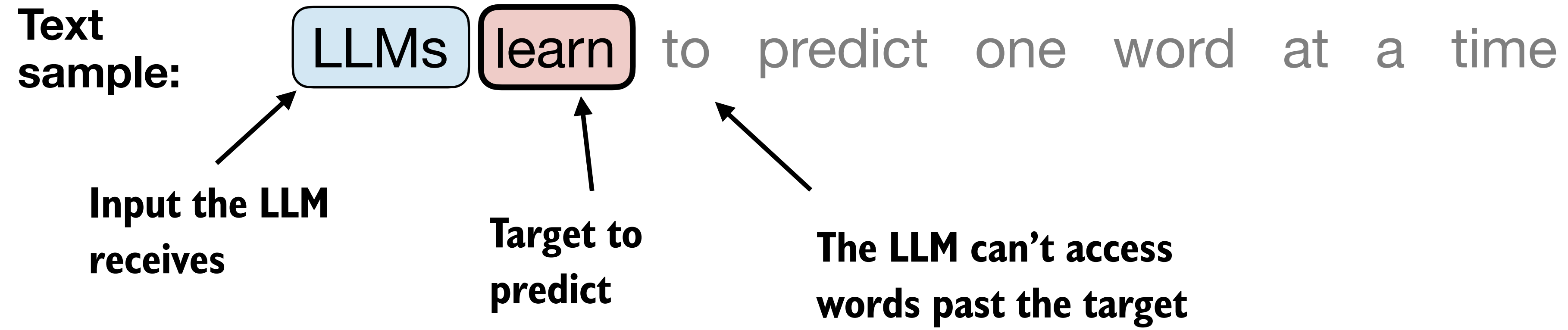
**The model is simply (pre)trained  
to predict the next word**



# Next word (/token) prediction

**Text  
sample:**

LLMs learn to predict one word at a time



**Sample 1**

LLMs

learn

to predict one word at a time

**Sample 2**

LLMs

learn

to

predict one word at a time



**Sample 1**

LLMs learn to predict one word at a time

**Sample 2**

LLMs learn to predict one word at a time

**Sample 3**

LLMs learn to predict one word at a time

**Sample 4**

LLMs learn to predict one word at a time

**Sample 5**

LLMs learn to predict one word at a time

**Sample 6**

LLMs learn to predict one word at a time

**Sample 7**

LLMs learn to predict one word at a time

**Sample 8**

LLMs learn to predict one word at a time

# Batching

Sample text

"In the heart of the city stood the old library, a relic from a bygone era. Its stone walls bore the marks of time, and ivy clung tightly to its facade ..."

Tensor  
containing  
the inputs

```
x = tensor([[ "In",      "the",      "heart", "of" ],  
            [ "the",      "city",      "stood", "the" ],  
            [ "old",      "library", ",",      "a" ],  
            [ ... ]])
```

# Batching

Sample text

"In the heart of the city stood the old library, a relic from a bygone era. Its stone walls bore the marks of time, and ivy clung tightly to its facade ..."

Tensor  
containing  
the inputs

```
x = tensor([[ "In",      "the",      "heart", "of" ],  
            [ "the",      "city",      "stood", "the" ],  
            [ "old",      "library", ",",    "a" ],  
            [ ... ]])
```

# Batching

Sample text

"In the heart of the city stood the old library, a relic from a bygone era. Its stone walls bore the marks of time, and ivy clung tightly to its facade ..."

Tensor  
containing  
the inputs

```
x = tensor([[ "In",      "the",      "heart", "of"    ],
            [ "the",    "city",    "stood", "the"   ],
            [ "old",    "library", ",",    "a"     ],
            [ ...]])
```

# Batching

Sample text

"In the heart of the city stood the old library, a relic from a bygone era. Its stone walls bore the marks of time, and ivy clung tightly to its facade ..."

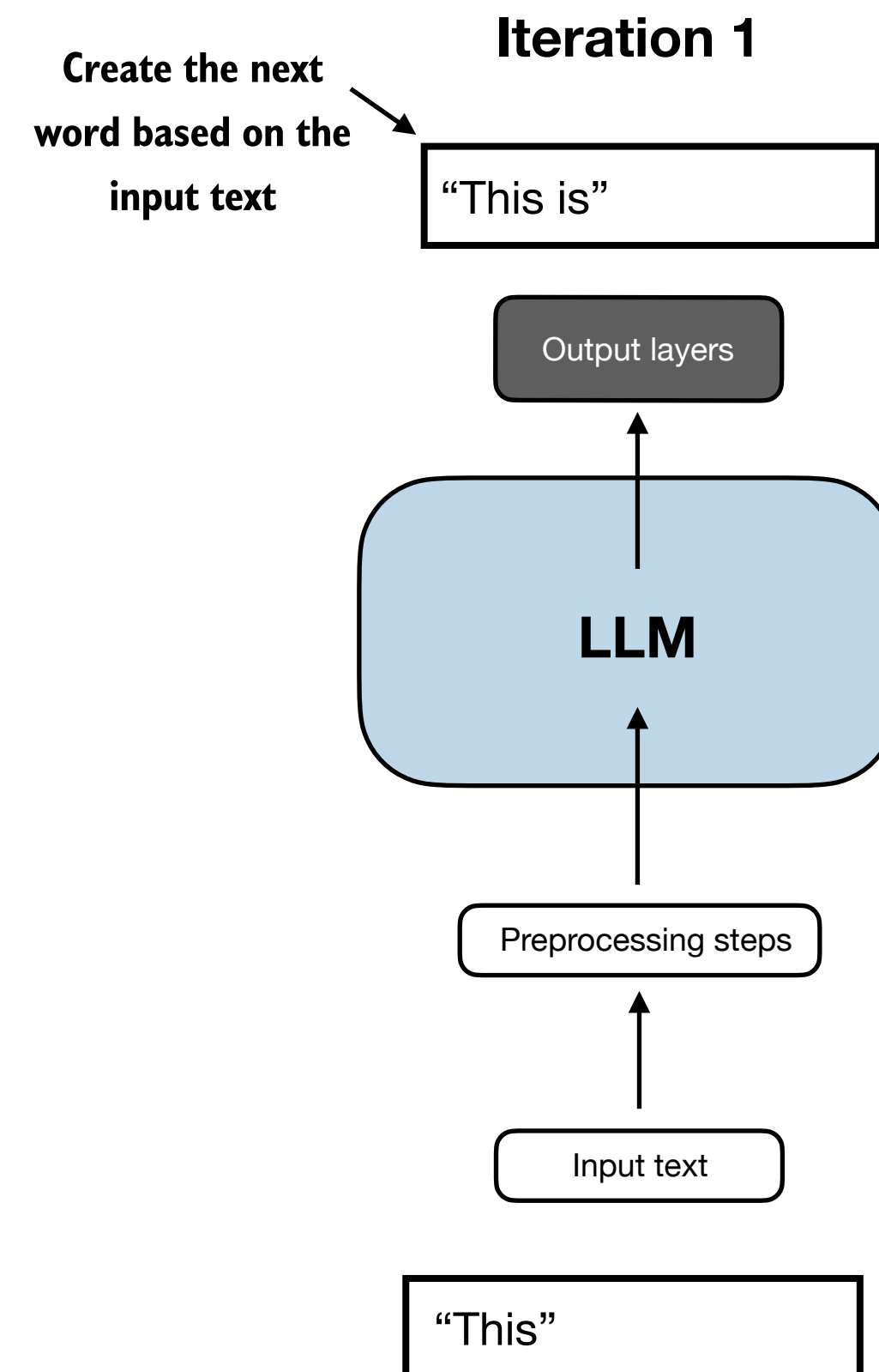
Tensor  
containing  
the inputs

```
x = tensor([[ "In",      "the",      "heart", "of"      ],  
            [ "the",      "city",      "stood", "the"     ],  
            [ "old",      "library", ",",      "a"       ],  
            [ ...]])
```

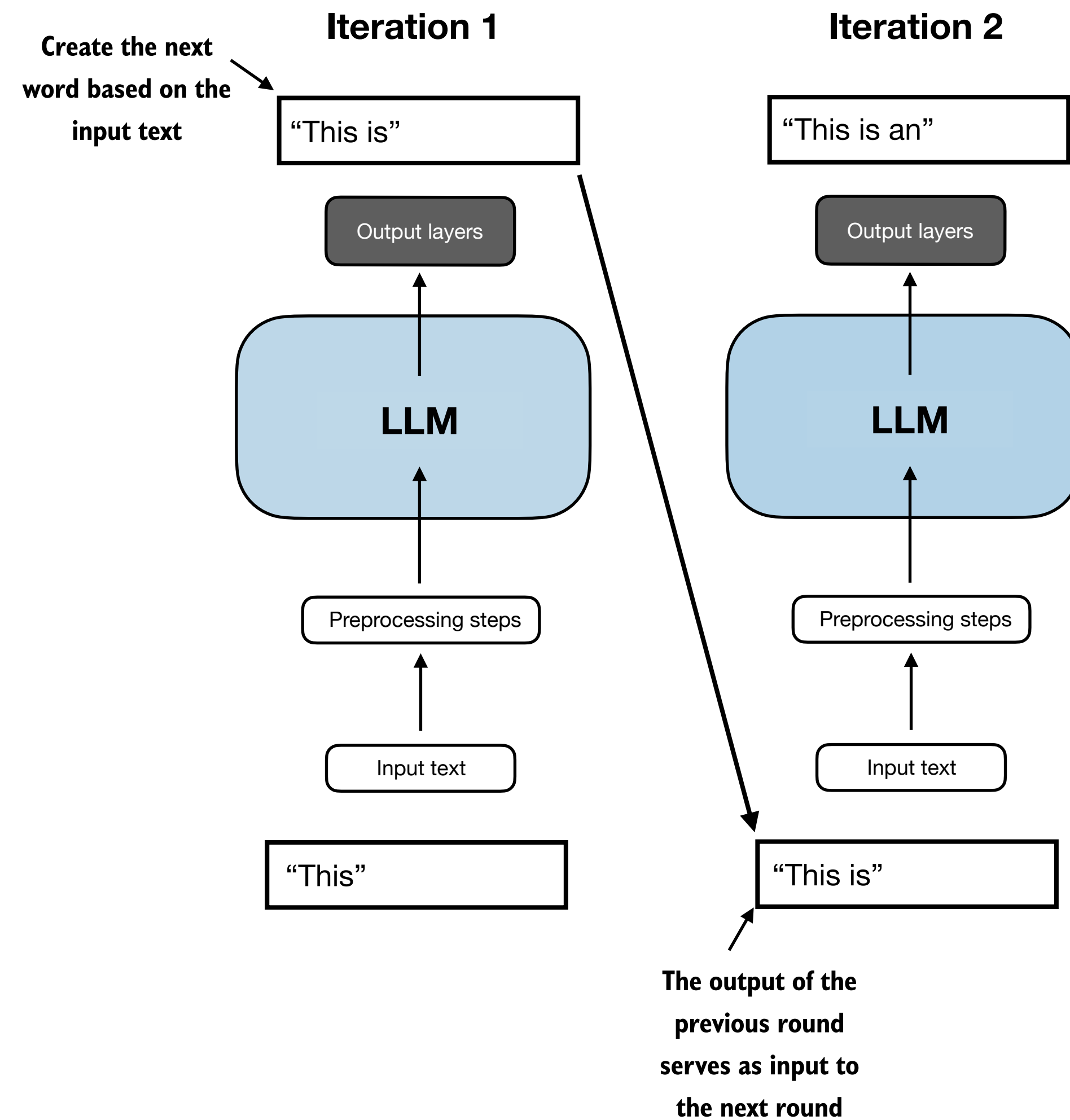
(Common input lengths are >1024)



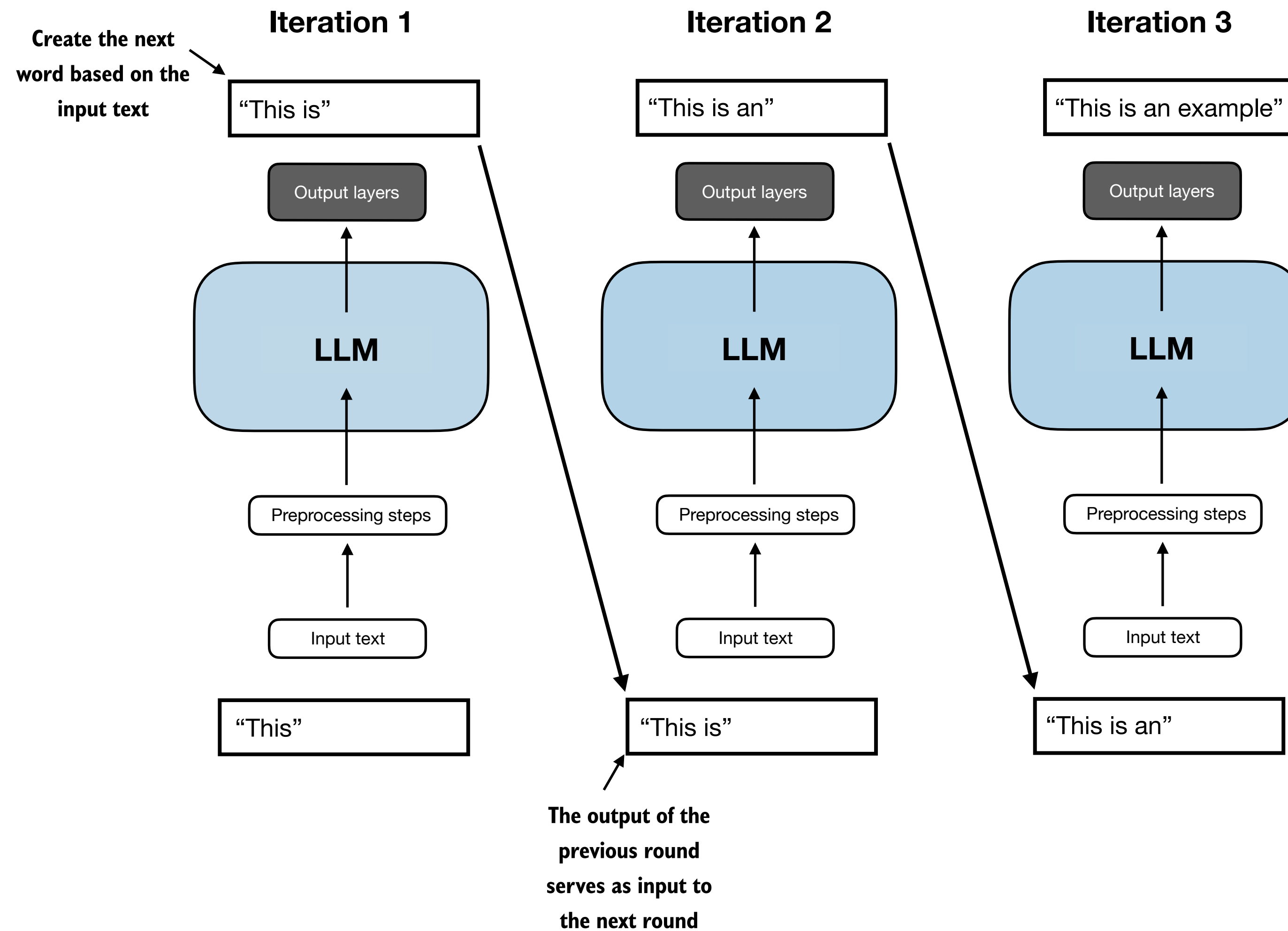
# How do LLMs generate multi-word outputs?



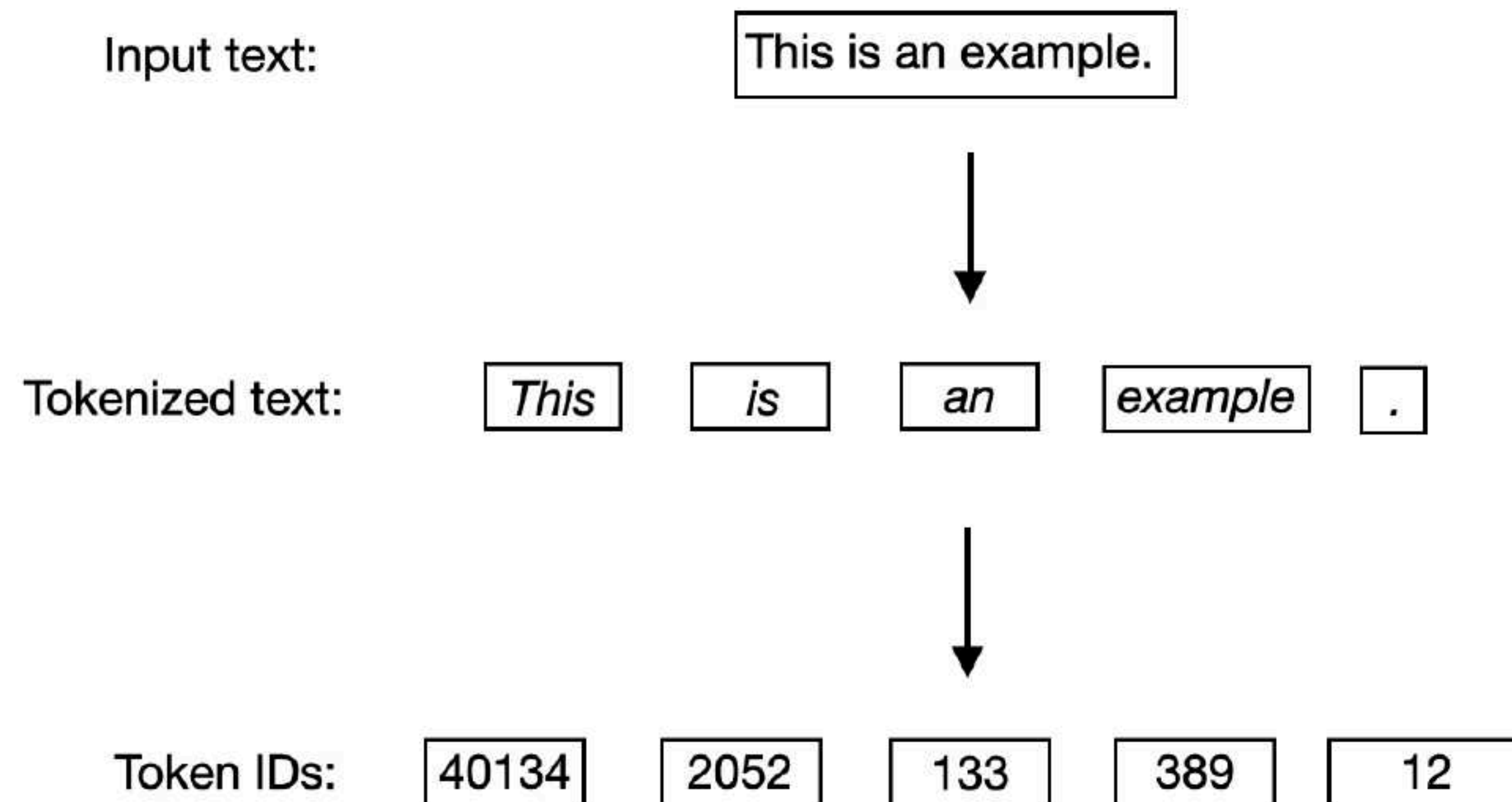
# How do LLMs generate multi-word outputs?

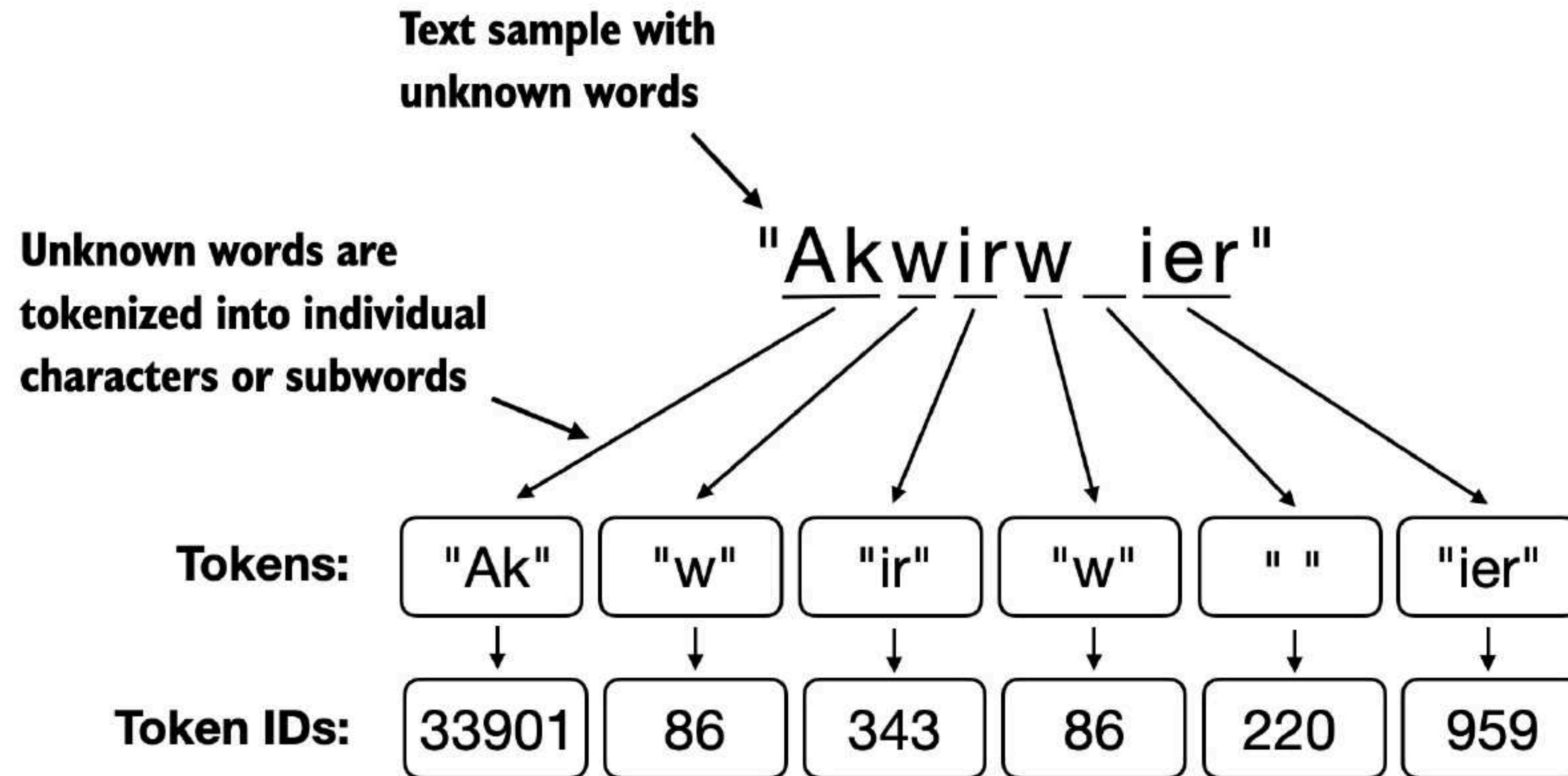


# How do LLMs generate multi-word outputs?



# There's one more thing: tokenization







# The GPT-3 dataset was 499 billion tokens

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

Language Models are Few-Shot Learners (2020), <https://arxiv.org/abs/2005.14165>

# Llama 1 was trained on 1.4T tokens

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

LLaMA: Open and Efficient Foundation Language Models (2023), <https://arxiv.org/abs/2302.13971>

# Llama 2 was trained on 2T tokens

“Our training corpus includes a new mix of data from publicly available sources, which does not include data from Meta’s products or services. We made an effort to remove data from certain sites known to contain a high volume of personal information about private individuals. We trained on 2 trillion tokens of data as this provides a good performance–cost trade-off, up-sampling the most factual sources in an effort to increase knowledge and dampen hallucinations.”

Llama 2: Open Foundation and Fine-Tuned Chat Models (2023), <https://arxiv.org/abs/2307.09288>

# Llama 3 was trained on 15T tokens

“To train the best language model, the curation of a large, high-quality training dataset is paramount. In line with our design principles, we invested heavily in pretraining data. Llama 3 is pretrained on over 15T tokens that were all collected from publicly available sources.”

Introducing Meta Llama 3: The most capable openly available LLM to date (2024), <https://ai.meta.com/blog/meta-llama-3/>

# Quantity vs quality

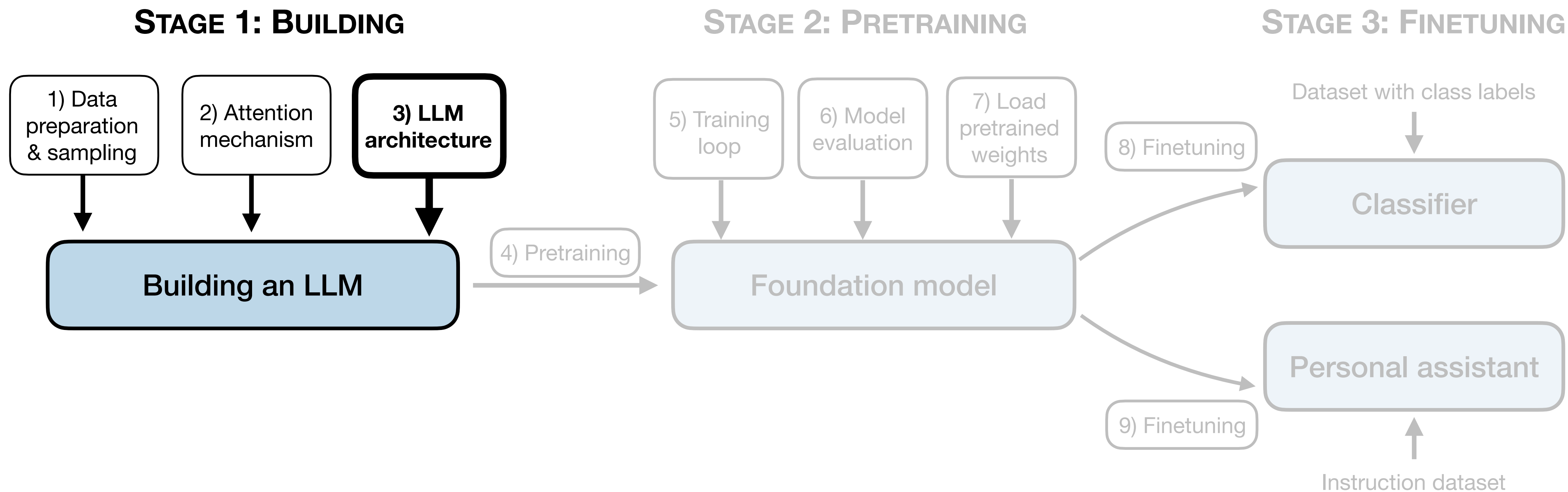
“we mainly focus on the **quality of data** for a given scale. We try to calibrate the training data to be closer to the “data optimal” regime for small models. In particular, we filter the publicly available web data to contain the correct level of “knowledge” and keep more web pages that could potentially improve the “reasoning ability” for the model. As an example, **the result of a game in premier league in a particular day might be good training data for frontier models, but we need to remove such information to leave more model capacity for “reasoning”** for the mini size models.

Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone (2024), <https://arxiv.org/abs/2404.14219>

# **What goes into developing an LLM like this?**

# LLM architectures

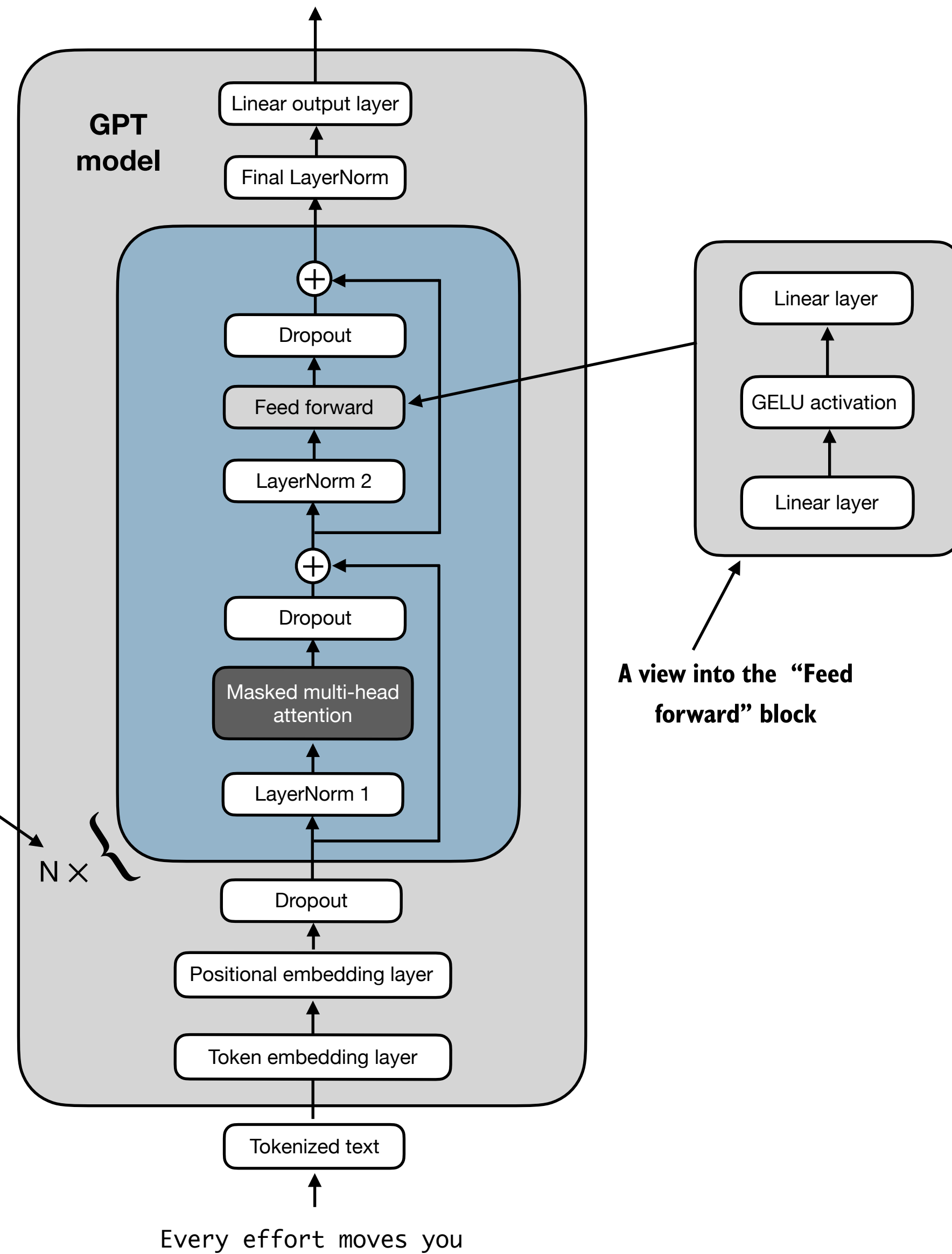
# Implementing the architecture





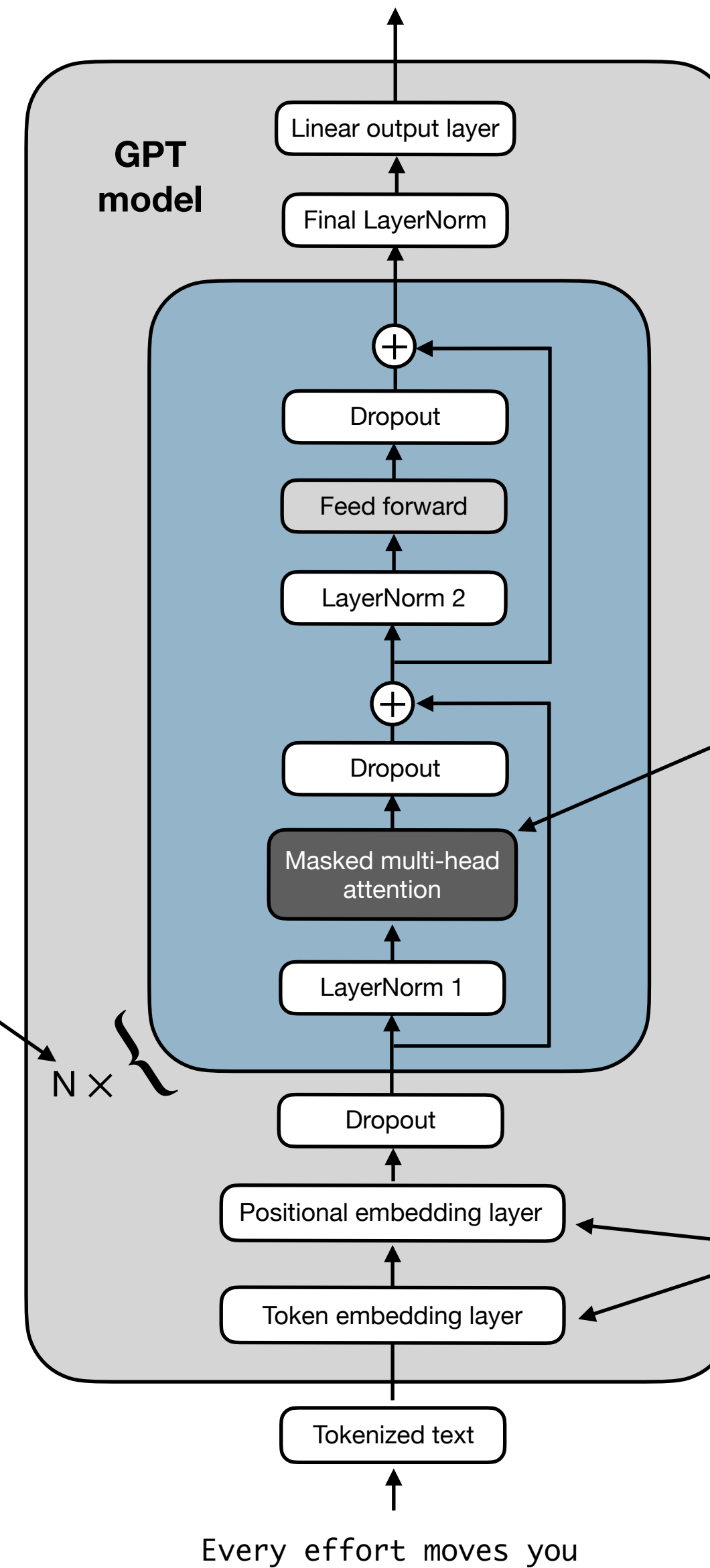
# The original GPT model

Repeat this transformer block  $N$  times



- Total number of parameters:**
- 124 M in "gpt2-small"
  - 355 M in "gpt2-medium"
  - 774 M in "gpt2-large"
  - 1558 M in "gpt2-xl"

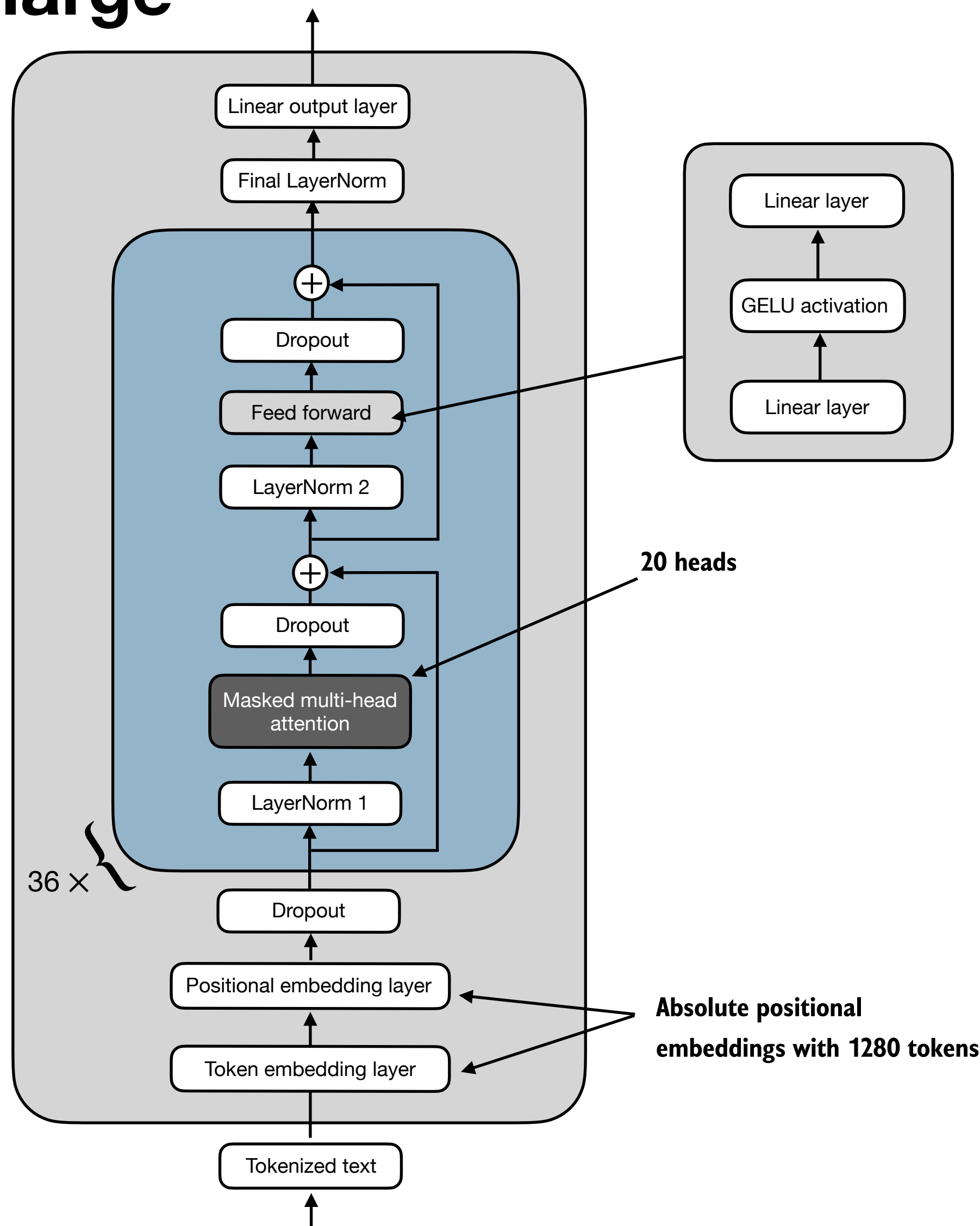
- Repeat this transformer block:**
- 12 × in "gpt2-small"
  - 24 × in "gpt2-medium"
  - 36 × in "gpt2-large"
  - 48 × in "gpt2-xl"



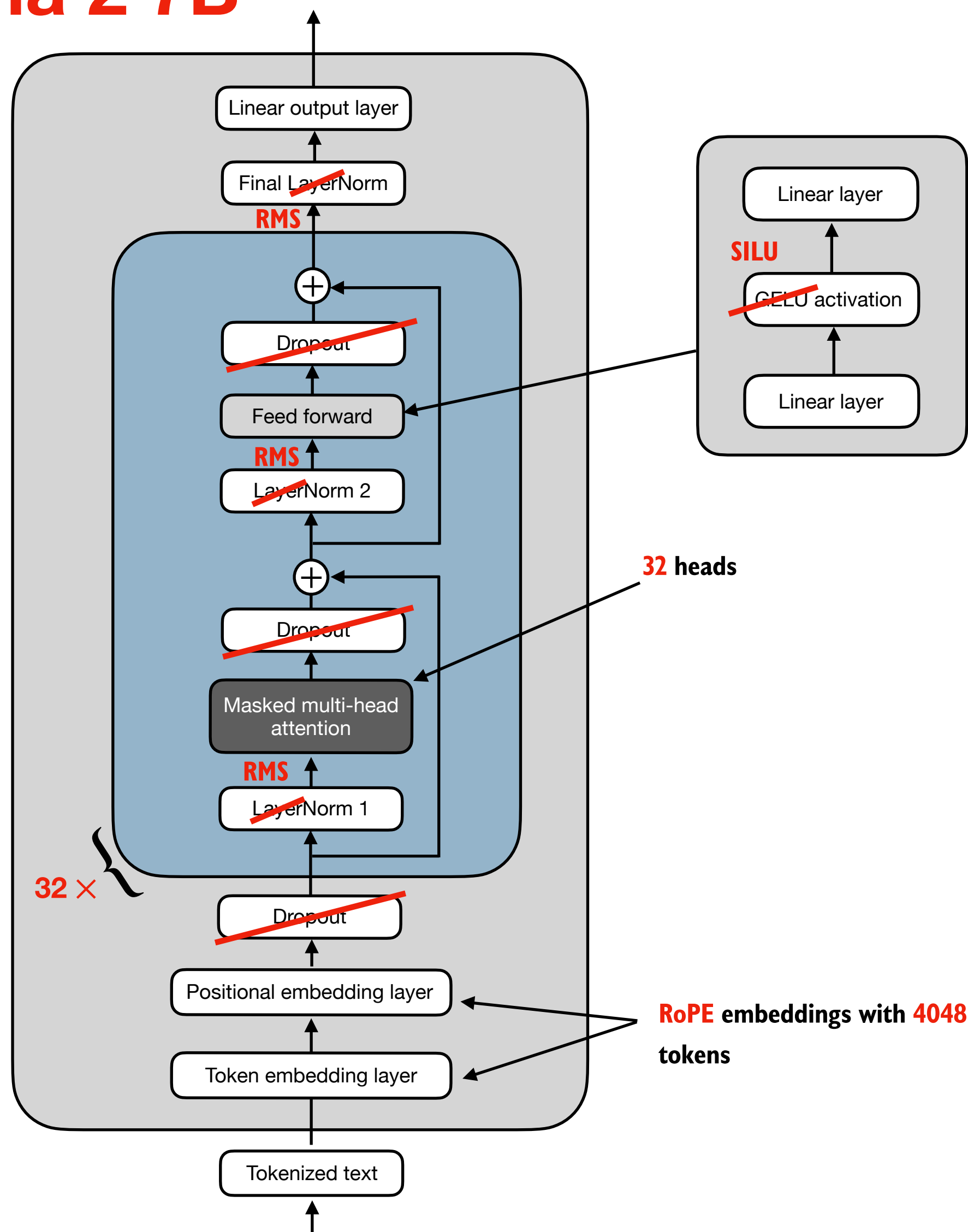
- Number of heads in multi-head attention:**
- 12 in "gpt2-small"
  - 16 in "gpt2-medium"
  - 20 in "gpt2-large"
  - 25 in "gpt2-xl"

- Embedding dimensions:**
- 768 in "gpt2-small"
  - 1024 in "gpt2-medium"
  - 1280 in "gpt2-large"
  - 1600 in "gpt2-xl"

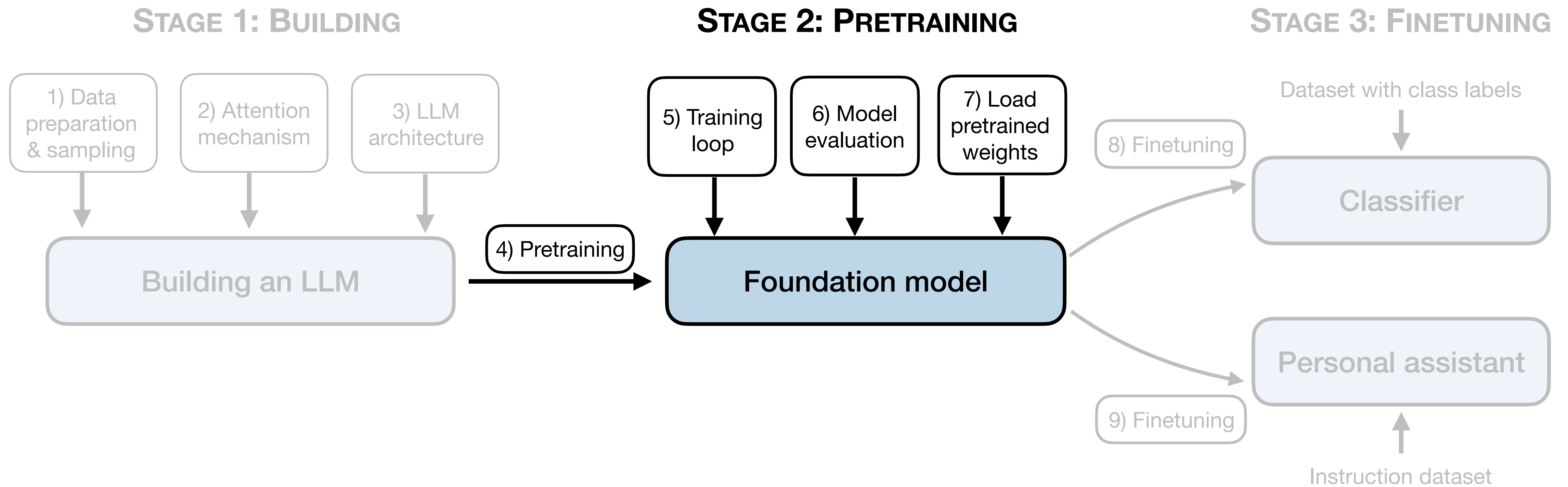
# GPT-2 “large”



# Llama 2 7B

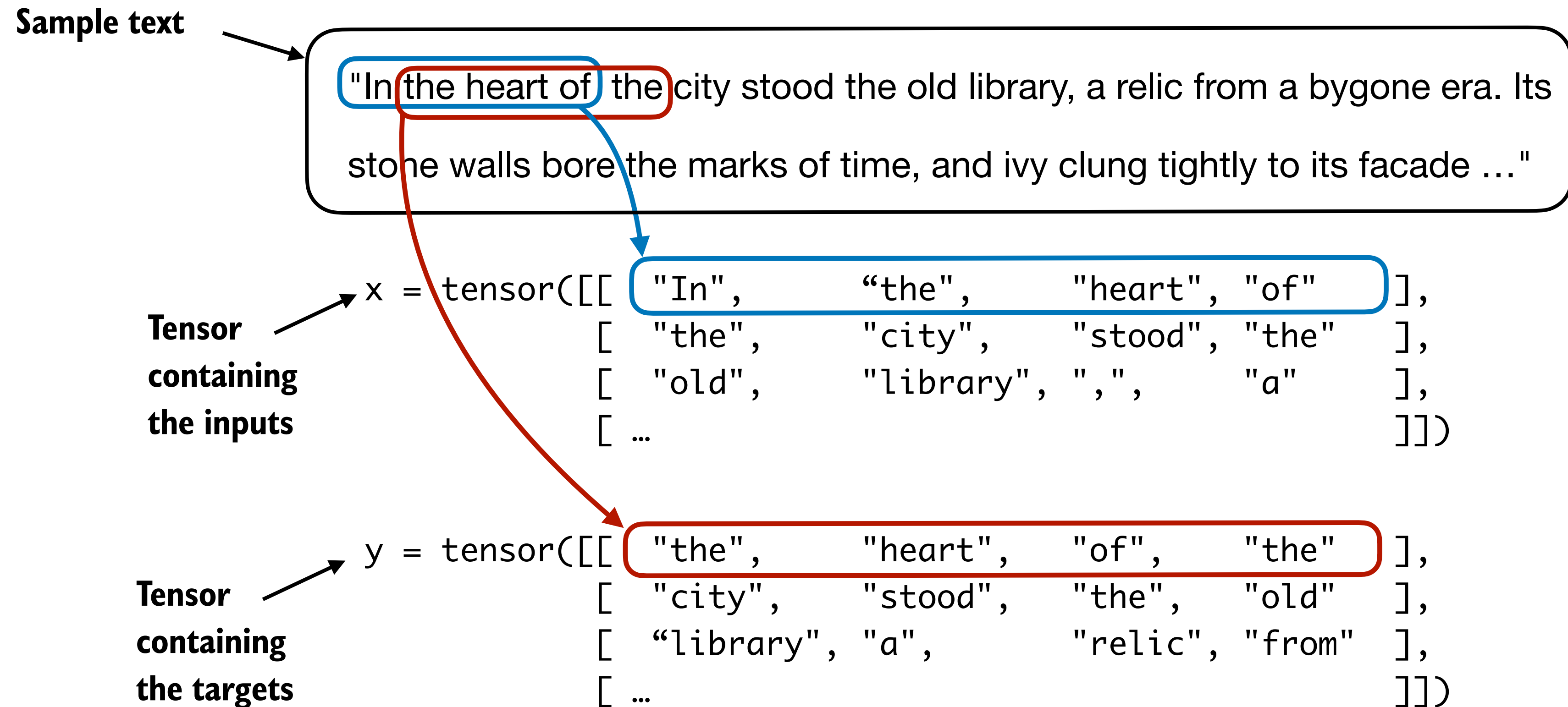


# Stage 2: Pretraining

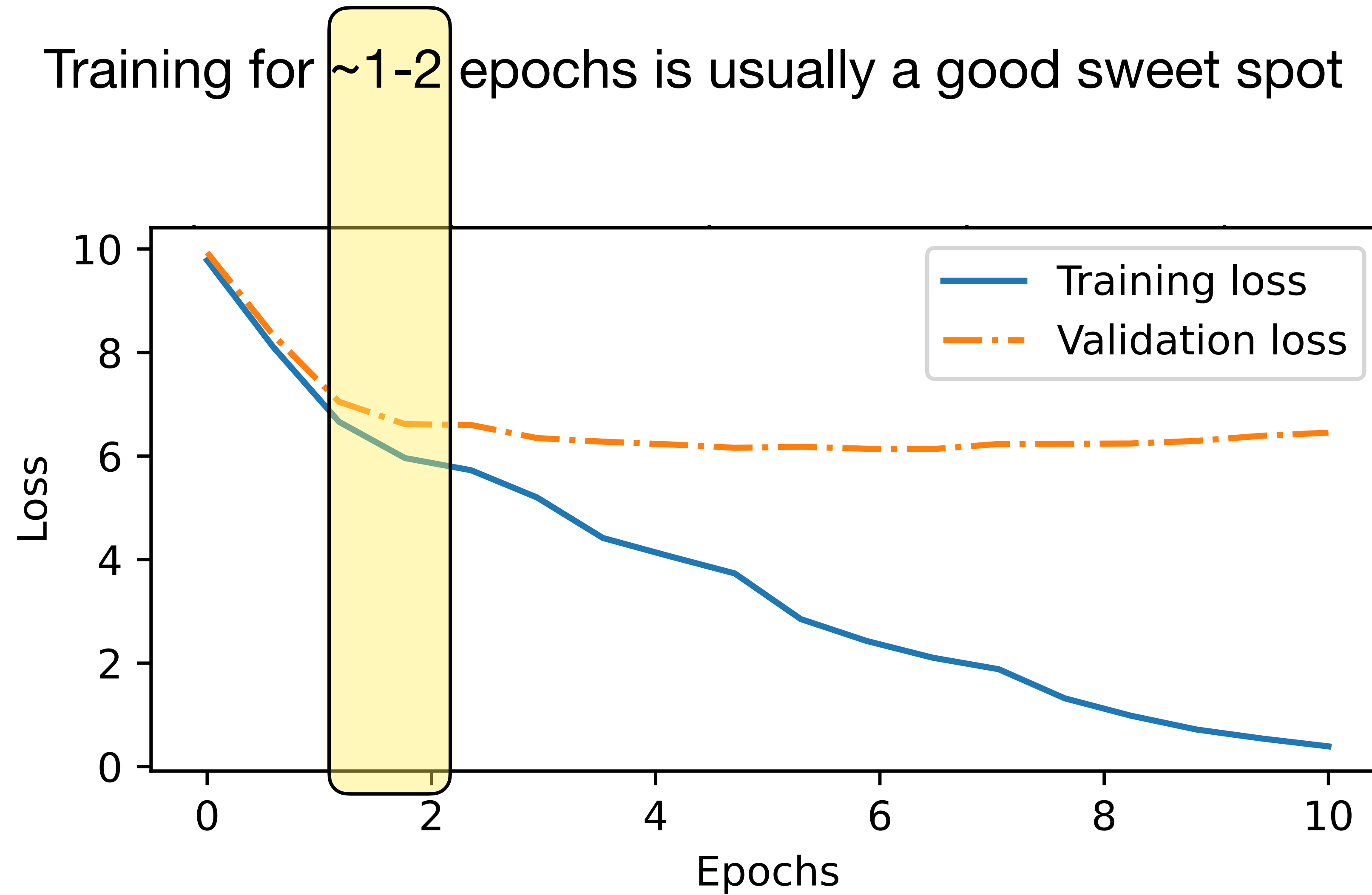


# Pretty standard deep learning training loop

# Labels are the inputs shifted by +1



Training for ~1-2 epochs is usually a good sweet spot






[← Featured](#)

3.41 K views155 runsShare

# Pretrain LLMs - TinyLlama 1.1B

 Lightning AI  
Published January 8, 2024

Open in Studio

[Overview](#)[Files](#)

## Pretrain LLM - TinyLlama 1.1B


Use this Studio to pretrain your own 1B LLM (TinyLlama). Click "Run" to run this Studio on cloud GPUs with Lightning Studio. What's included in this Studio:

- Model architecture
- Pretraining recipe
- 1.2 TB dataset optimized for fast data loading
- Code written in PyTorch and Lightning Fabric

Click "Open template" to run your own copy of this Studio.

### In-depth walkthrough

This 90 minute video does a deep dive without edits, skipping no details of pretraining an LLM using this Studio. It shows how to start at 1B on 1 GPU and scale it to 3B across 16 H100 GPUs and more. By the end of this video, you'll have the tools to know how to trade-off time vs size vs compute.



**Pretrain TinyLlama 1.1B**

Machine: (1 × A10G) GPU  
License: Apache-2.0  
Get Studio badge

TrainingText

**Pretrain LLM - TinyLlama 1.1B**

- In-depth walkthrough
- About pretraining TinyLlama
- Meet TinyLlama

**Train TinyLlama**

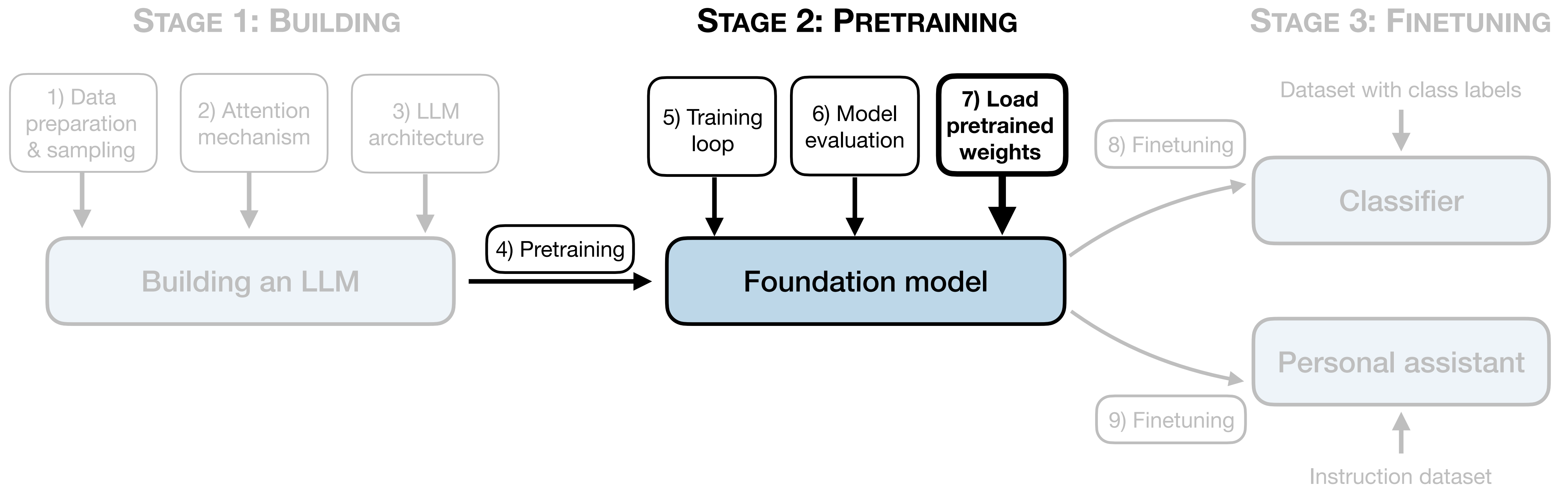
- Train It in a Studio
- Speed It up on Multiple Machines
- Locate the Logs and Checkpoints
- Resume Training
- Convert Checkpoints

**Evaluate TinyLlama**

- Conclusion
- FAQ

Parameters	1.1 billion
Context size	2048 tokens
Training data	<a href="#">SlimPajama</a> (893 GB) <a href="#">Starcoder</a> (290 GB)
Combined dataset size	~ 950 billion tokens
Total tokens during training	3 trillion (3 epochs)
Time to complete training	~ 4 weeks with 64 A100 GPUs
Model FLOPs Utilization (MFU)	55%

<https://lightning.ai/lightning-ai/studios/pretrain-llms-tinyllama-1-1b>



# Loading pretrained weights

## 🔗 Choose from 20+ LLMs

LitGPT has **custom, from-scratch** implementations of [20+ LLMs](#) without layers of abstraction:

Model	Model size	Author	Reference
Llama 3	8B, 70B	Meta AI	<a href="#">Meta AI 2024</a>
Llama 2	7B, 13B, 70B	Meta AI	<a href="#">Touvron et al. 2023</a>
Code Llama	7B, 13B, 34B, 70B	Meta AI	<a href="#">Rozière et al. 2023</a>
Mixtral MoE	8x7B	Mistral AI	<a href="#">Mistral AI 2023</a>
Mistral	7B	Mistral AI	<a href="#">Mistral AI 2023</a>
CodeGemma	7B	Google	<a href="#">Google Team, Google Deepmind</a>
...	...	...	...

<https://github.com/Lightning-AI/litgpt>

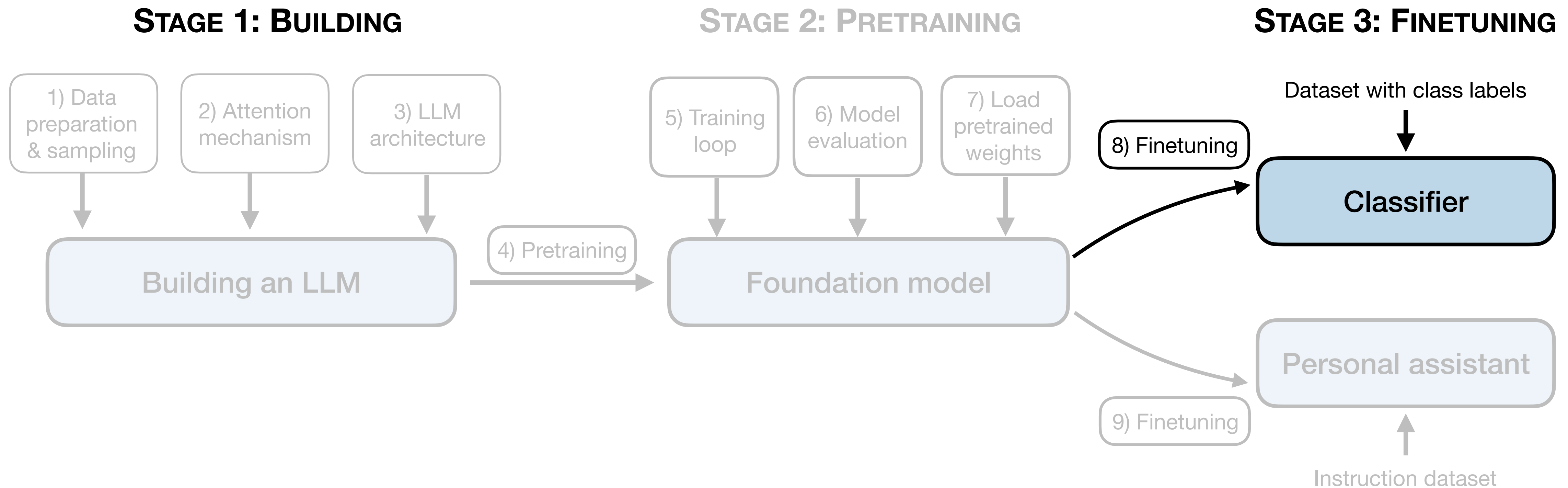


# LitGPT

```
# litgpt [action] [model]
litgpt download meta-llama/Meta-Llama-3-8B-Instruct
litgpt chat meta-llama/Meta-Llama-3-8B-Instruct
litgpt finetune meta-llama/Meta-Llama-3-8B-Instruct
litgpt pretrain meta-llama/Meta-Llama-3-8B-Instruct
litgpt serve meta-llama/Meta-Llama-3-8B-Instruct
```

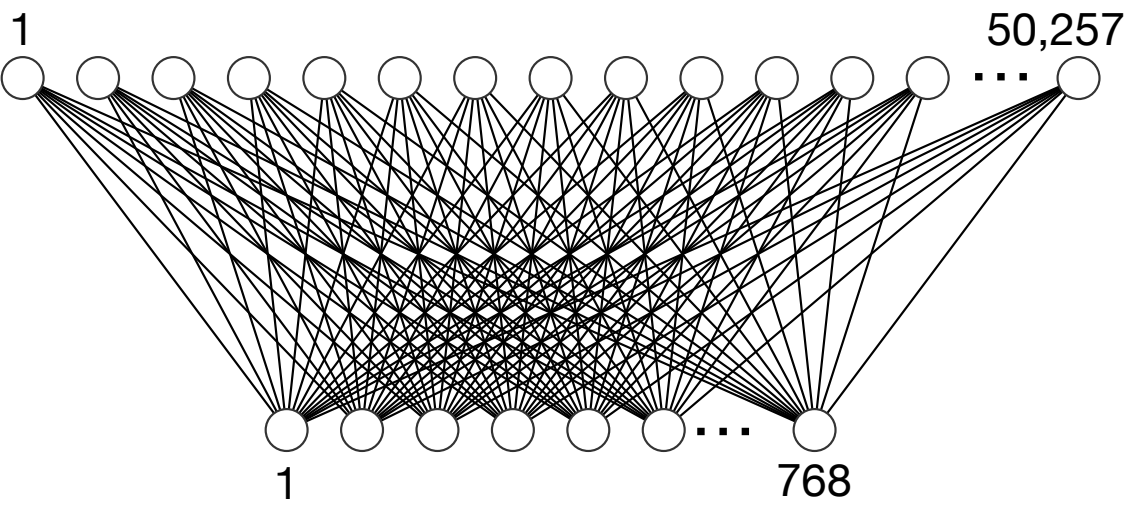
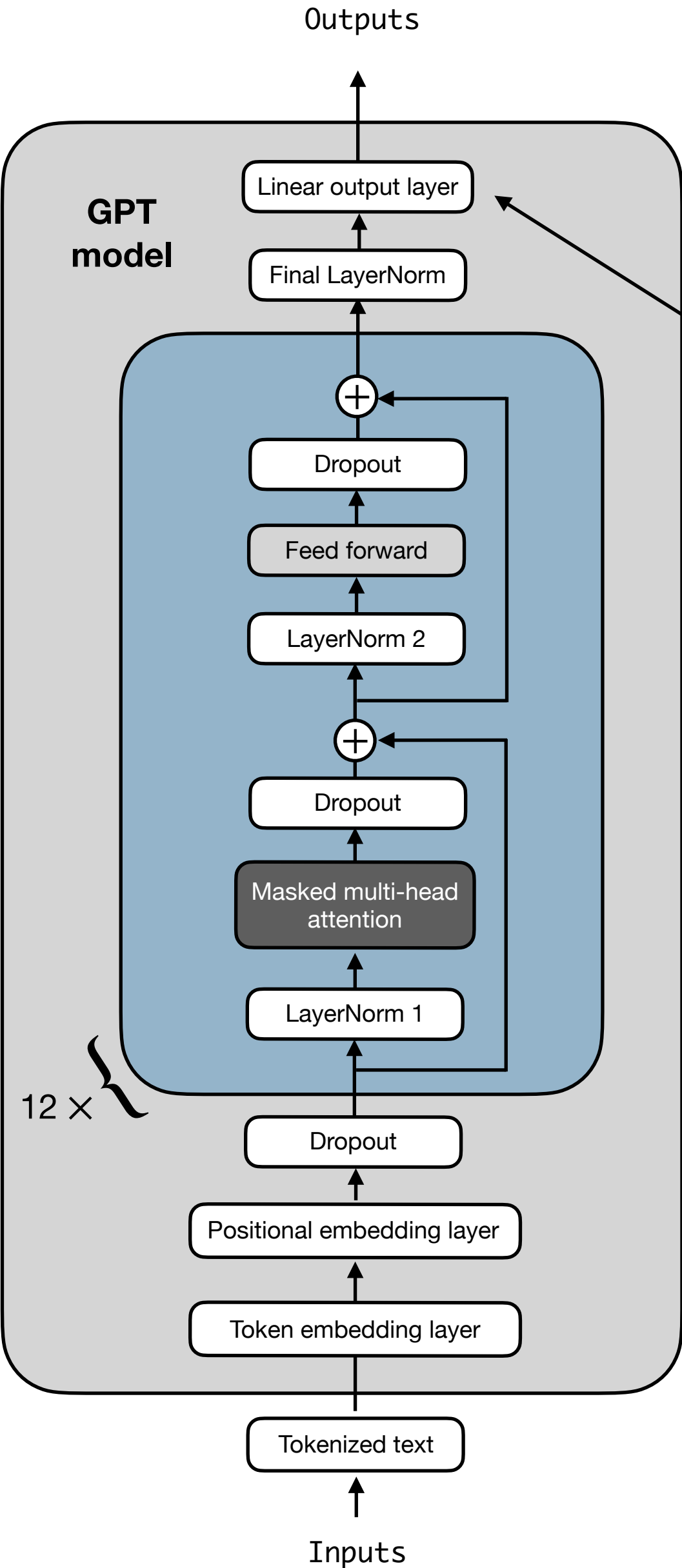
<https://github.com/Lightning-AI/litgpt>

# Stage 3: Finetuning



	Label	Text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...

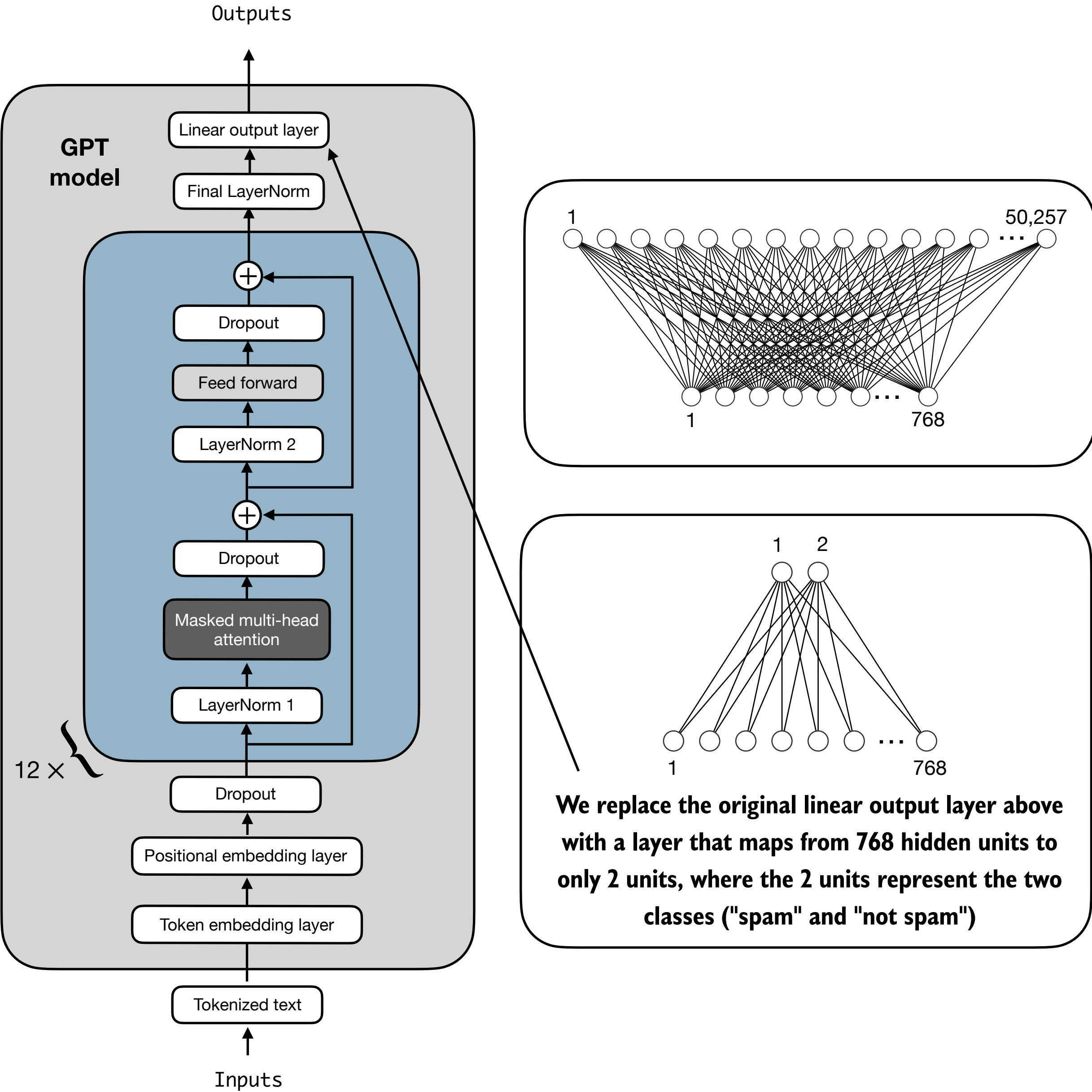
# Replace output layer



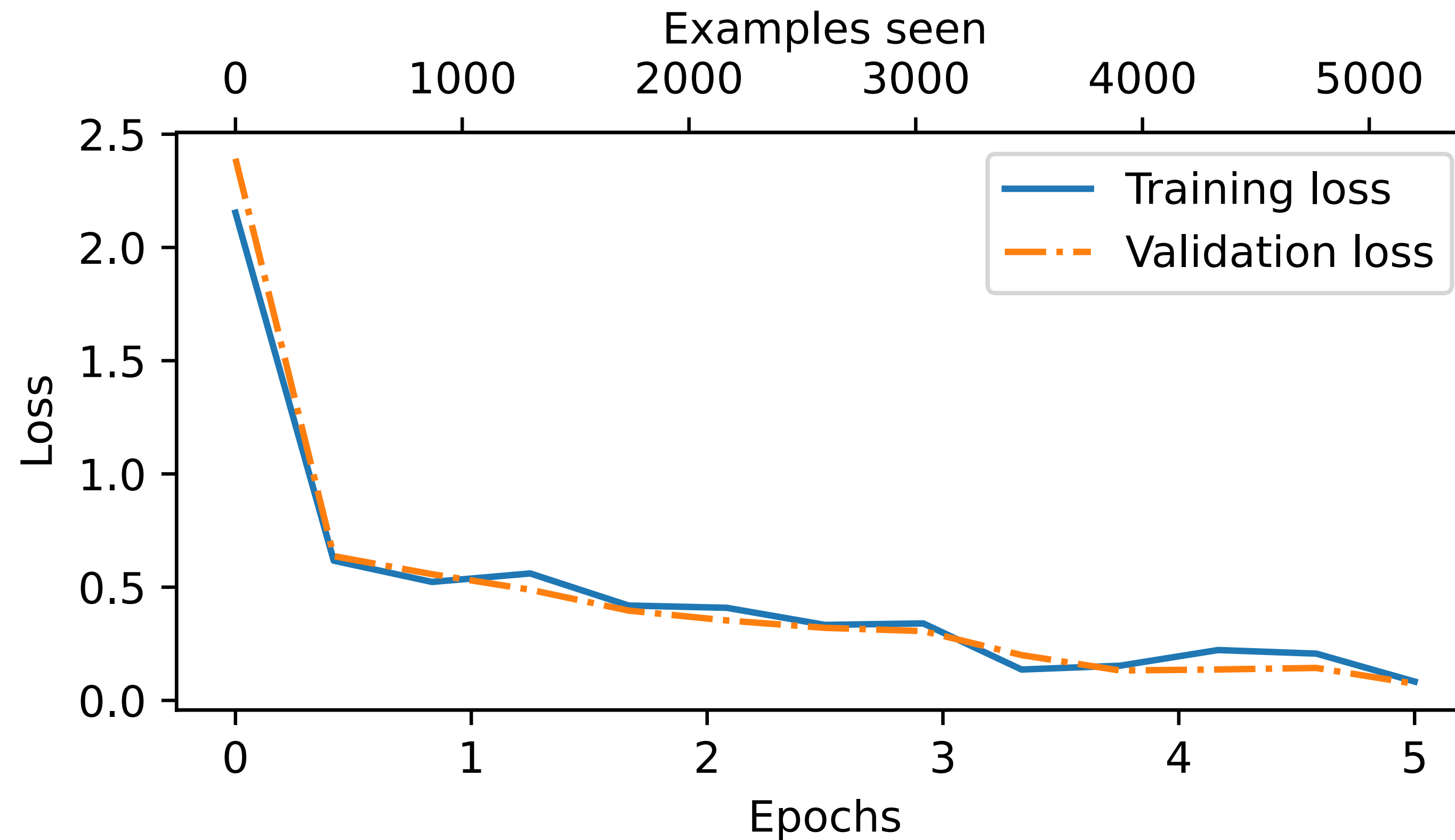
**The original linear output layer  
maps 768 hidden units to 50,257 units  
(the number of tokens in the vocabulary)**



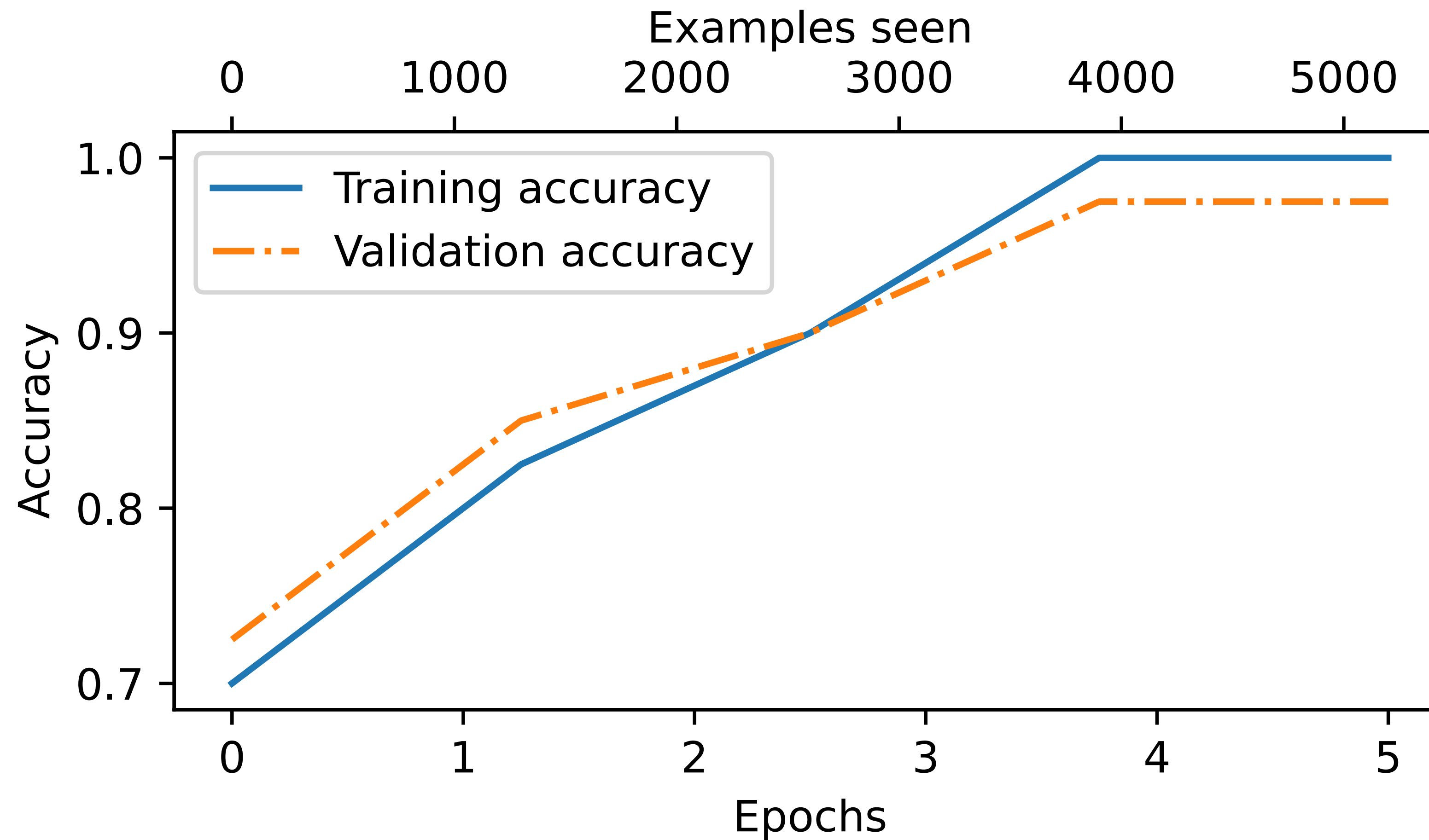
# Replace output layer



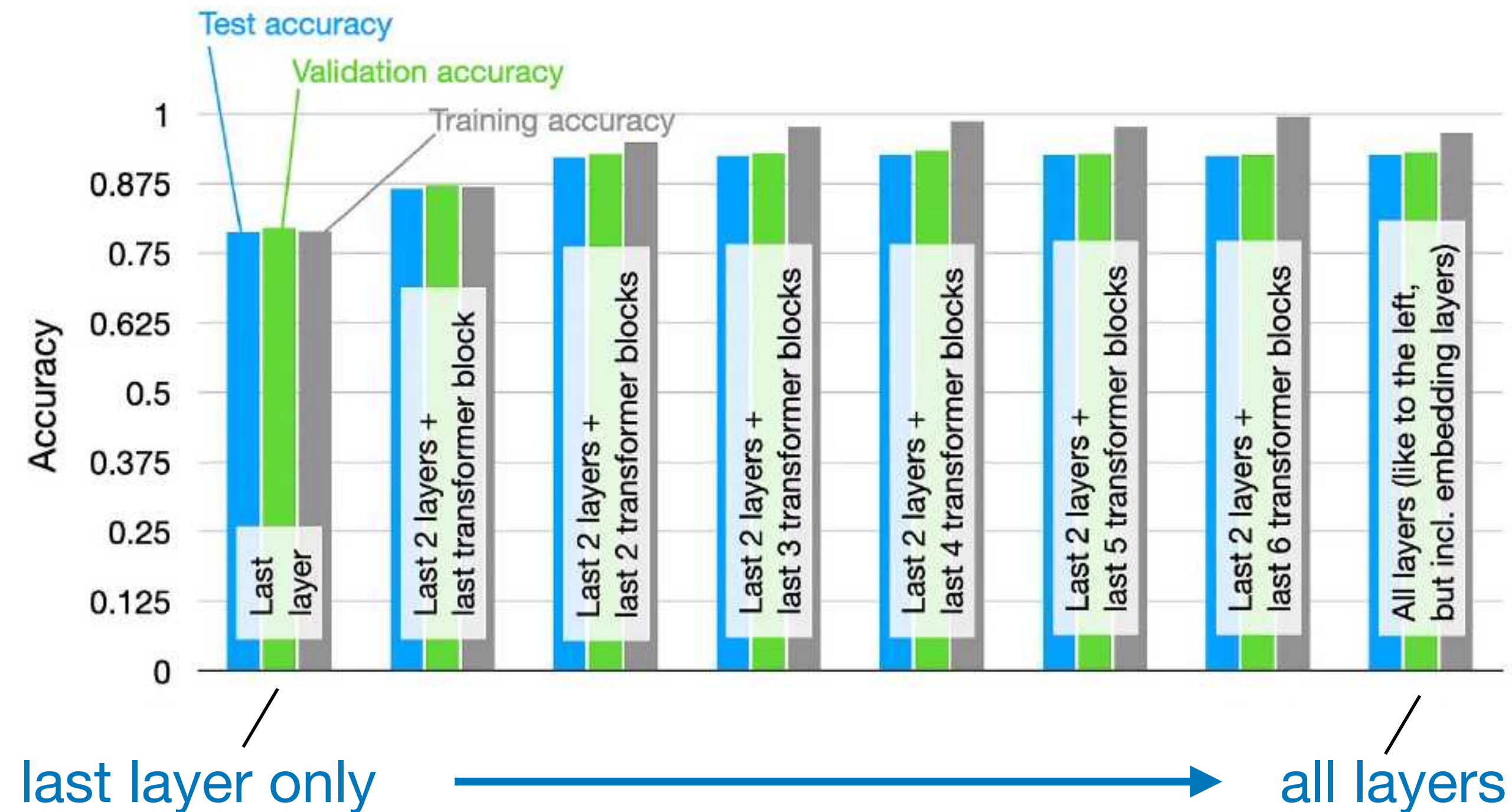
# Track loss values as usual



# In addition, look at task performance



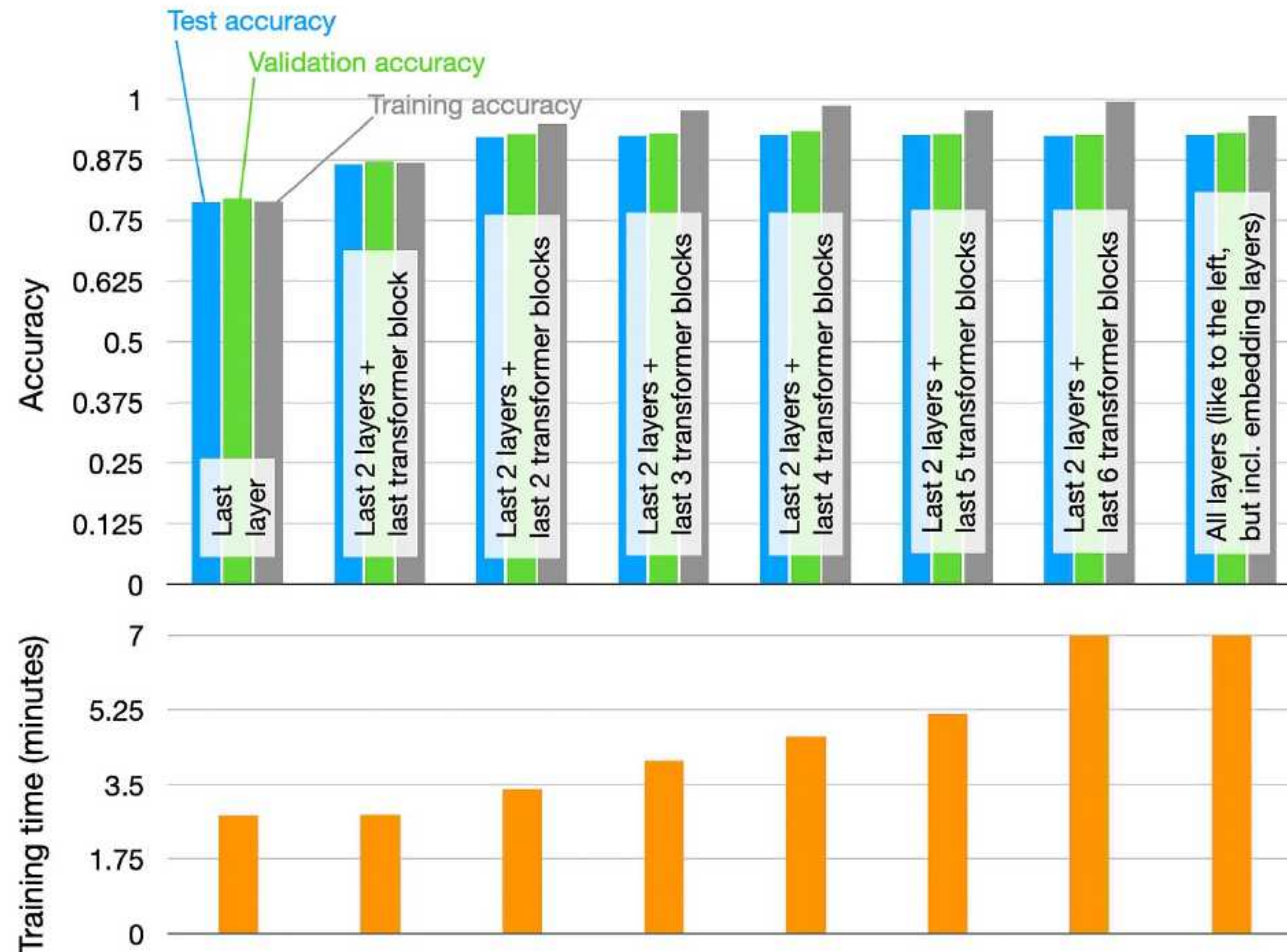
# We don't need to finetune all layers



<https://magazine.sebastianraschka.com/p/finetuning-large-language-models>

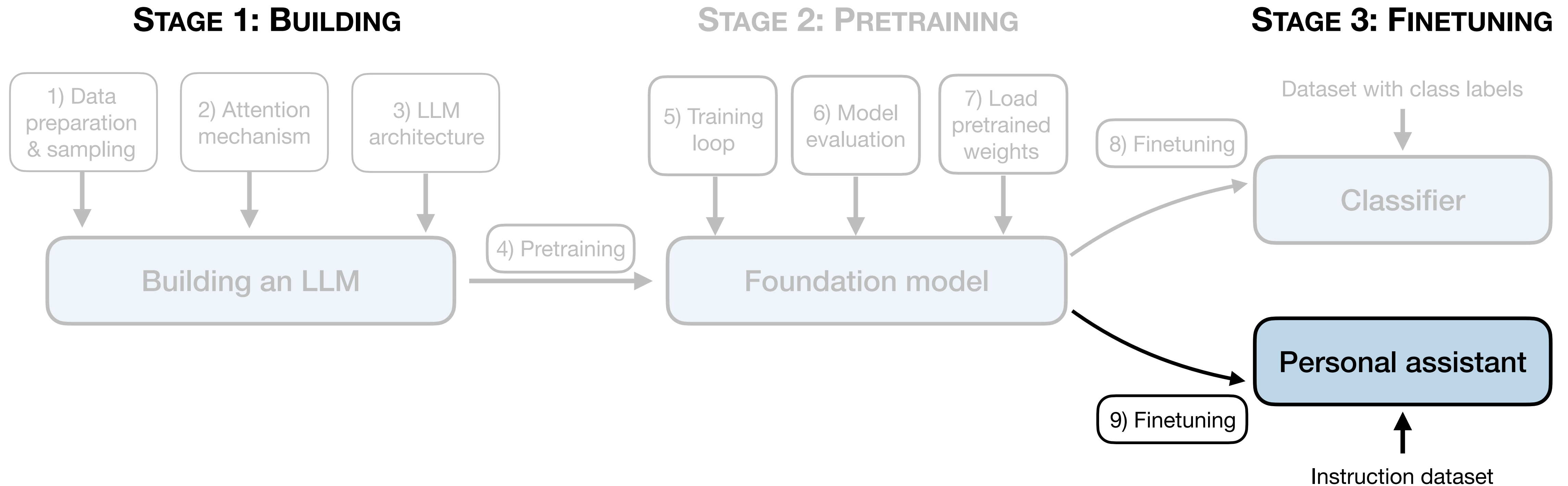


# Training more layers takes more time



<https://magazine.sebastianraschka.com/p/finetuning-large-language-models>

# Instruction finetuning



# Instruction finetuning datasets

```
{  
  "instruction": "Rewrite the following sentence using passive voice.",  
  "input": "The team achieved great results.",  
  "output": "Great results were achieved by the team."  
},
```

```
{  
  "instruction": "Rewrite the following sentence using passive voice.",  
  "input": "The team achieved great results.",  
  "output": "Great results were achieved by the team."  
},
```



**Apply prompt style template (for example, Alpaca-style)**

Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:  
Rewrite the following sentence using passive voice.

### Input:  
The team achieved great results.

### Response:  
Great results were achieved by the team.



**Pass to LLM for supervised instruction finetuning**





**Model input**

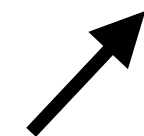


```
Below is an instruction that describes a task. Write a response  
that appropriately completes the request.
```

```
### Instruction:  
Rewrite the following sentence using passive voice.
```

```
### Input:  
The team achieved great results.
```

```
### Response:  
Great results were achieved by the team.
```



**Model response**

# Bonus: Preference tuning

**Input Prompt:**

**"What are the key features to look for when purchasing a new laptop?"**

## **Input Prompt:**

**"What are the key features to look for when purchasing a new laptop?"**

### **Answer 1: Technical Response**

**"When purchasing a new laptop, focus on key specifications such as the processor speed, RAM size, storage type (SSD vs. HDD), and battery life. The processor should be powerful enough for your software needs, and sufficient RAM will ensure smooth multitasking. Opt for an SSD for faster boot times and file access. Additionally, screen resolution and port types are important for connectivity and display quality."**

## **Input Prompt:**

**"What are the key features to look for when purchasing a new laptop?"**

### **Answer 1: Technical Response**

**"When purchasing a new laptop, focus on key specifications such as the processor speed, RAM size, storage type (SSD vs. HDD), and battery life. The processor should be powerful enough for your software needs, and sufficient RAM will ensure smooth multitasking. Opt for an SSD for faster boot times and file access. Additionally, screen resolution and port types are important for connectivity and display quality."**

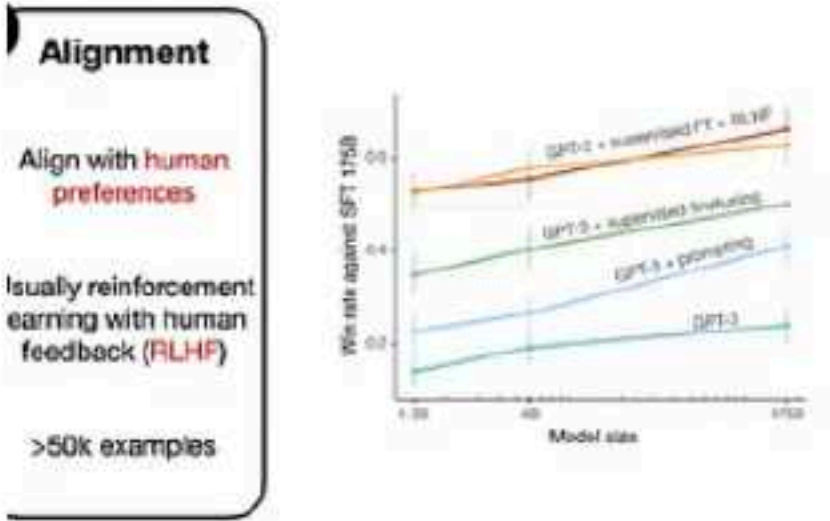
### **Answer 2: User-Friendly Response**

**"When looking for a new laptop, think about how it fits into your daily life. Choose a lightweight model if you travel frequently, and consider a laptop with a comfortable keyboard and a responsive touchpad. Battery life is crucial if you're often on the move, so look for a model that can last a full day on a single charge. Also, make sure it has enough USB ports and possibly an HDMI port to connect with other devices easily."**

# LLM Training: RLHF and Its Alternatives

I frequently reference a process called Reinforcement Learning with Human Feedback (RLHF) when discussing LLMs, whether ...

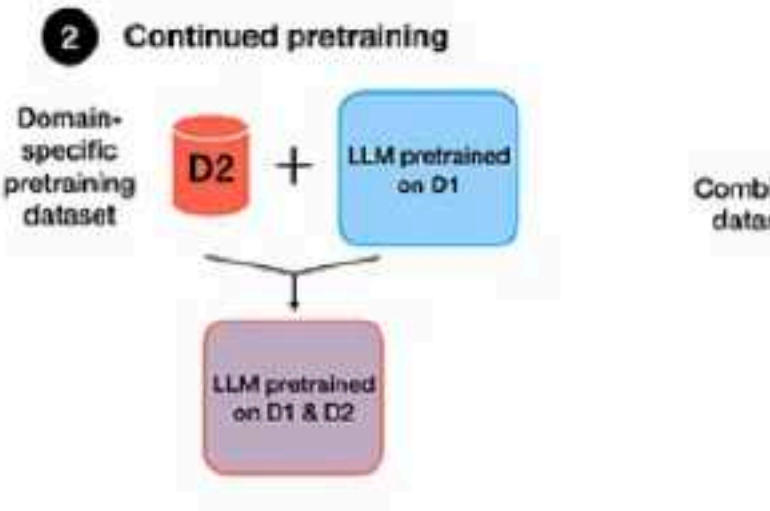
SEP 10, 2023 • SEBASTIAN RASCHKA, PHD



# Tips for LLM Pretraining and Evaluating Reward Models

Discussing AI Research Papers in March 2024

MAR 31 • SEBASTIAN RASCHKA, PHD



<https://magazine.sebastianraschka.com/p/llm-training-rlhf-and-its-alternatives>

# Evaluating LLMs

# MMLU and others

Rank	Model	MMLU Average ↑ (%)	Paper
1	Gemini Ultra ~1760B	90	Gemini: A Family of Highly Capable Multimodal Models
2	GPT-4o	88.7	GPT-4 Technical Report
3	Claude 3 Opus (5-shot, CoT)	88.2	The Claude 3 Model Family: Opus, Sonnet, Haiku
4	Claude 3 Opus (5-shot)	86.8	The Claude 3 Model Family: Opus, Sonnet, Haiku
5	Leeroo (5-shot)	86.64	Leeroo Orchestrator: Elevating LLMs Performance Through Model
6	GPT-4 (few-shot)	86.4	GPT-4 Technical Report
7	Gemini Ultra (5-shot)	83.7	Gemini: A Family of Highly Capable Multimodal Models
8	Claude 3 Sonnet (5-shot, CoT)	81.5	The Claude 3 Model Family: Opus, Sonnet, Haiku



# MMLU

MMLU = Measuring Massive Multitask Language Understanding (2020), <https://arxiv.org/abs/2009.03300>

Multiple-choice questions from diverse subjects

```
input = ("Which character is known for saying,  
        'To be, or not to be, that is the question'?  
Options:  
A) Macbeth, B) Othello,  
C) Hamlet, D) King Lear.")
```

```
model_answer = model(input)
```

```
correct_answer = "C) Hamlet"
```

```
score += model_answer == correct_answer
```

```
# total_score = score / num_examples * 100%
```

# LM Evaluation Harness

```
litgpt evaluate checkpoints/microsoft/phi-2/ \
  --batch_size 4 \
  --tasks "hellaswag,truthfulqa_mc2,mmlu" \
  --out_dir evaluate_model/
```

The resulting output is as follows:

```
...
|-----|-----|-----|-----:|-----|-----:|---|-----:|
...
|truthfulqa_mc2          |      2|none |      0|acc      |0.4656|±  |0.0164|
|hellaswag              |      1|none |      0|acc      |0.2569|±  |0.0044|
|                        |      |none |      0|acc_norm|0.2632|±  |0.0044|

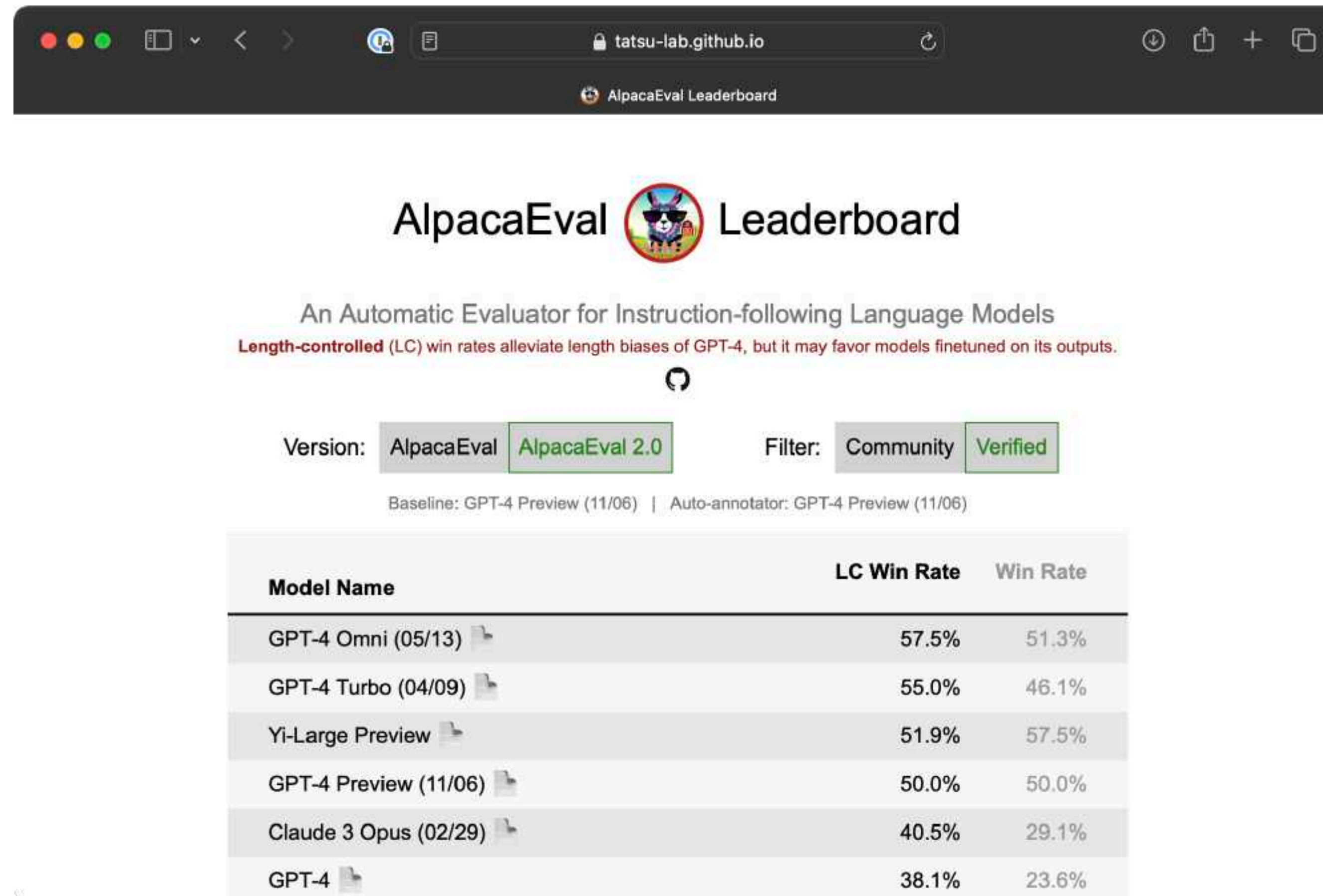
|      Groups      |Version|Filter|n-shot|Metric|Value | |Stderr|
|-----|-----|-----|-----:|-----|-----:|---|-----:|
|mmlu              |N/A    |none  |      0|acc    |0.2434|±  |0.0036|
| - humanities    |N/A    |none  |      0|acc    |0.2578|±  |0.0064|
| - other         |N/A    |none  |      0|acc    |0.2401|±  |0.0077|
| - social_sciences|N/A    |none  |      0|acc    |0.2301|±  |0.0076|
| - stem          |N/A    |none  |      0|acc    |0.2382|±  |0.0076|
```

<https://github.com/EleutherAI/lm-evaluation-harness>

<https://github.com/Lightning-AI/litgpt/blob/main/tutorials/evaluation.md>

# AlpacaEval

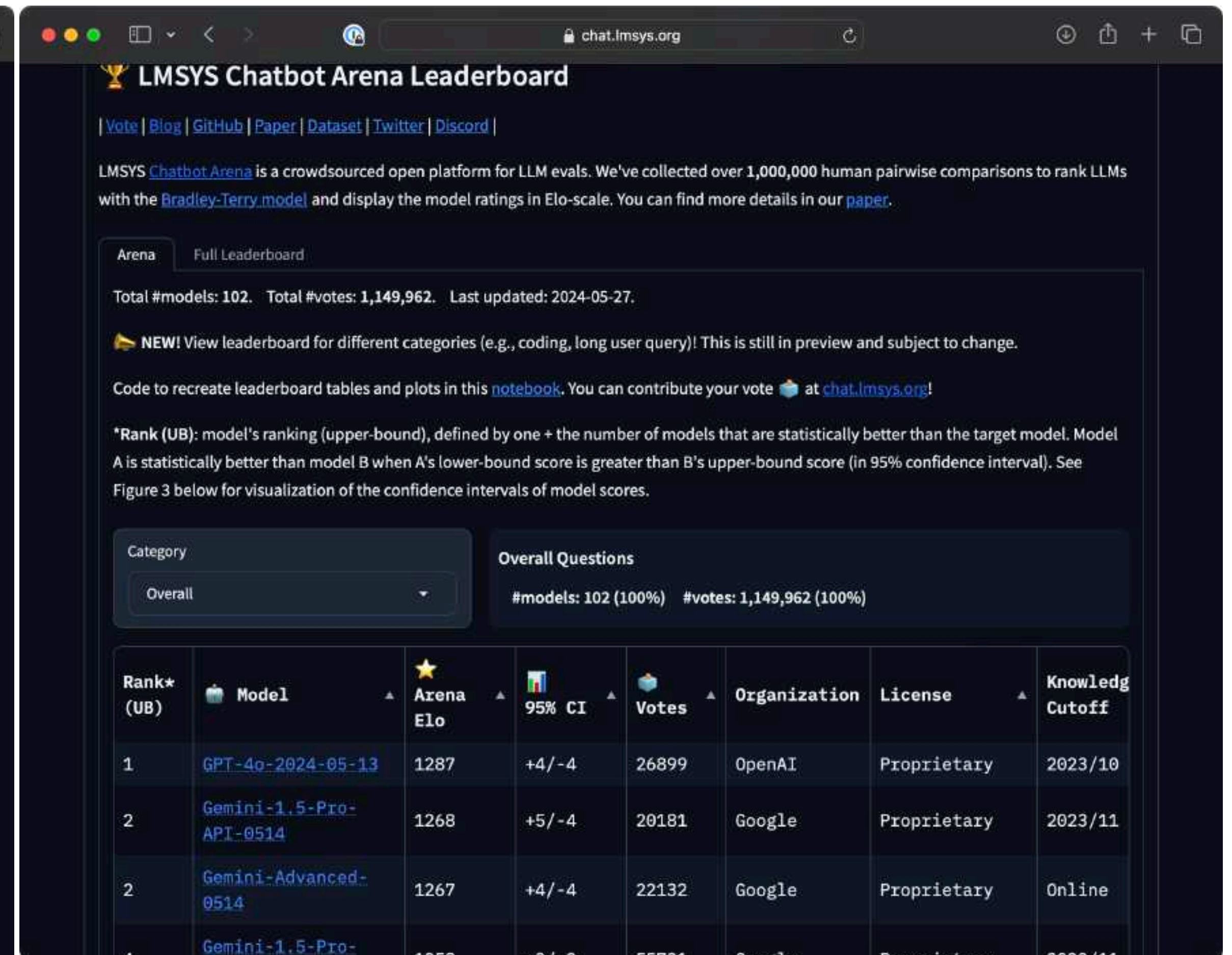
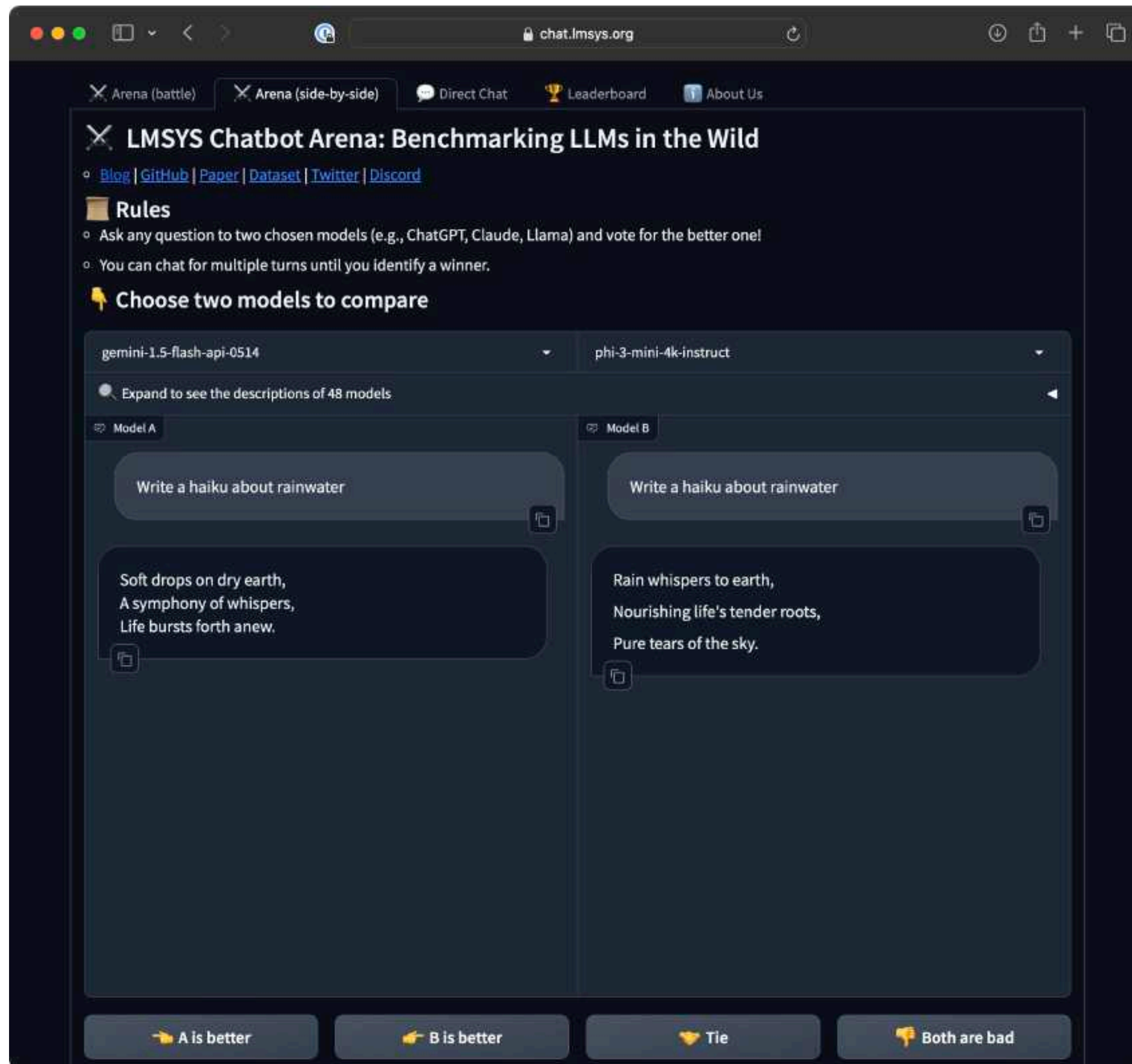
Compare to response by GPT-4 Preview using a GPT-4 based auto-annotator





# LMSYS ChatBot Arena

## LLM community comparison



Screenshots from <https://chat.lmsys.org/>



# GPT-4 scoring

```
from tqdm import tqdm

def generate_model_scores(json_data, json_key, client):
    scores = []
    for entry in tqdm(json_data, desc="Scoring entries"):
        prompt = (
            f"Given the input `{format_input(entry)}` "
            f"and correct output `{entry['output']}`, "
            f"score the model response `{entry[json_key]}` "
            f"on a scale from 0 to 100, where 100 is the best score. "
            f"Respond with the number only."
        )
        score = run_chatgpt(prompt, client)
        try:
            scores.append(int(score))
        except:
            continue

    return scores
```

```
In [10]: for model in ("model 1 response", "model 2 response"):

    scores = generate_model_scores(json_data, model, client)
    print(f"\n{model}")
    print(f"Number of scores: {len(scores)} of {len(json_data)}")
    print(f"Average score: {sum(scores)/len(scores):.2f}\n")
```

Scoring entries: 100% | ██████████ 100/100 [01:09<00:00, 1.44it/s]

```
model 1 response
Number of scores: 100 of 100
Average score: 74.04
```

```
Scoring entries: 100% | 100/100 [01:08<00:00, 1.46it/s]
```

```
model 2 response
Number of scores: 100 of 100
Average score: 56.72
```

[https://github.com/rasbt/LLMs-from-scratch/blob/main/ch07/03\\_model-evaluation/llm-instruction-eval-openai.ipynb](https://github.com/rasbt/LLMs-from-scratch/blob/main/ch07/03_model-evaluation/llm-instruction-eval-openai.ipynb)

# Rules of thumb

# Rules of thumb

**Pretraining from scratch → Expensive, almost never necessary**

# Rules of thumb

Pretraining from scratch → Expensive, almost never necessary

**Continued pretraining → Add new knowledge**



# Rules of thumb

Pretraining from scratch → Expensive, almost never necessary

Continued pretraining → Add new knowledge

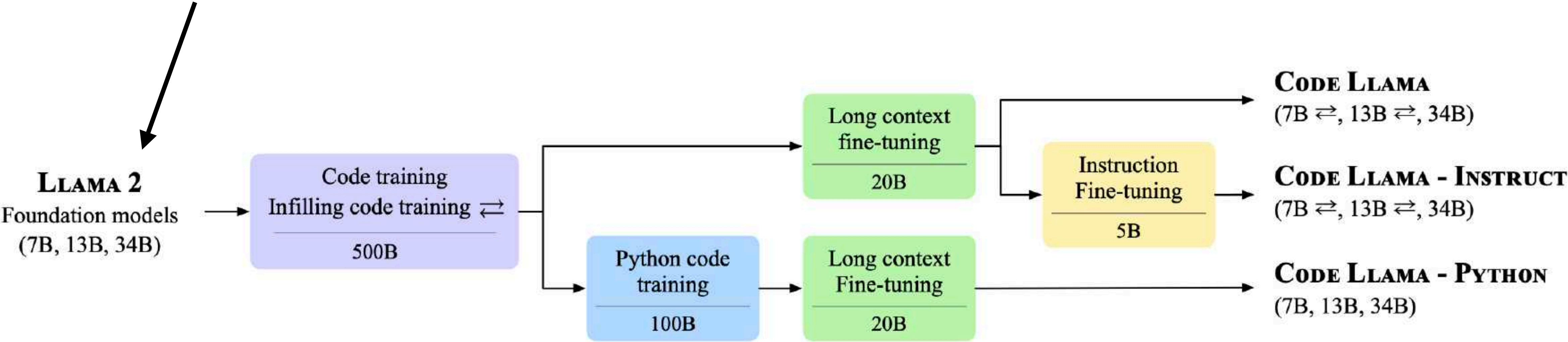
**Finetuning** → **Special usecase, follow instructions**

# Rules of thumb

Pretraining from scratch	→	Expensive, almost never necessary
Continued pretraining	→	Add new knowledge
Finetuning	→	Special usecase, follow instructions
<b>Preference finetuning</b>	<b>→</b>	<b>Improve helpfulness+safety if developing a chatbot</b>

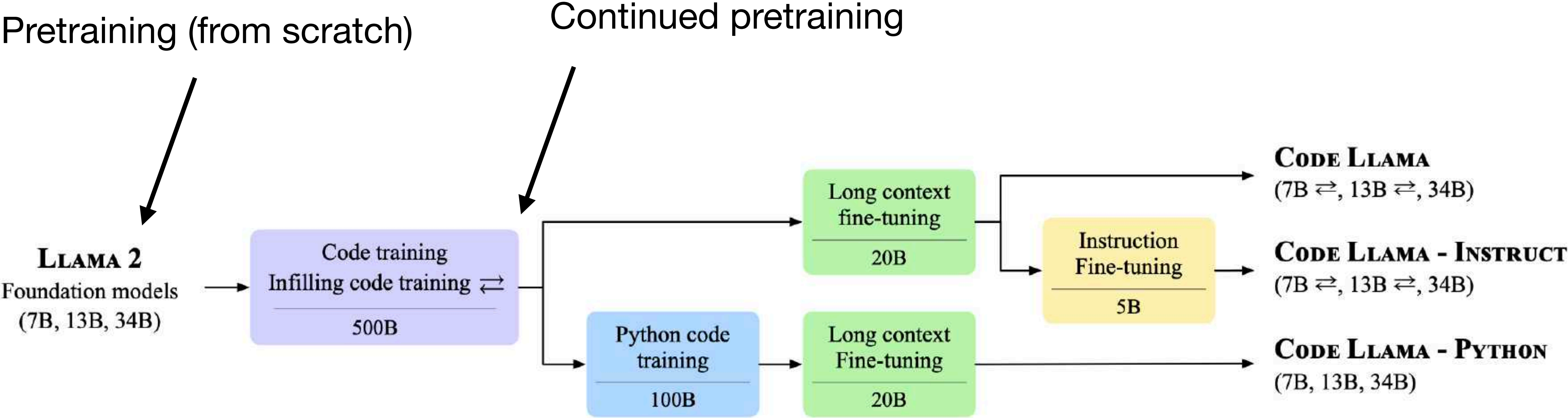
# CodeLlama example

Pretraining (from scratch)



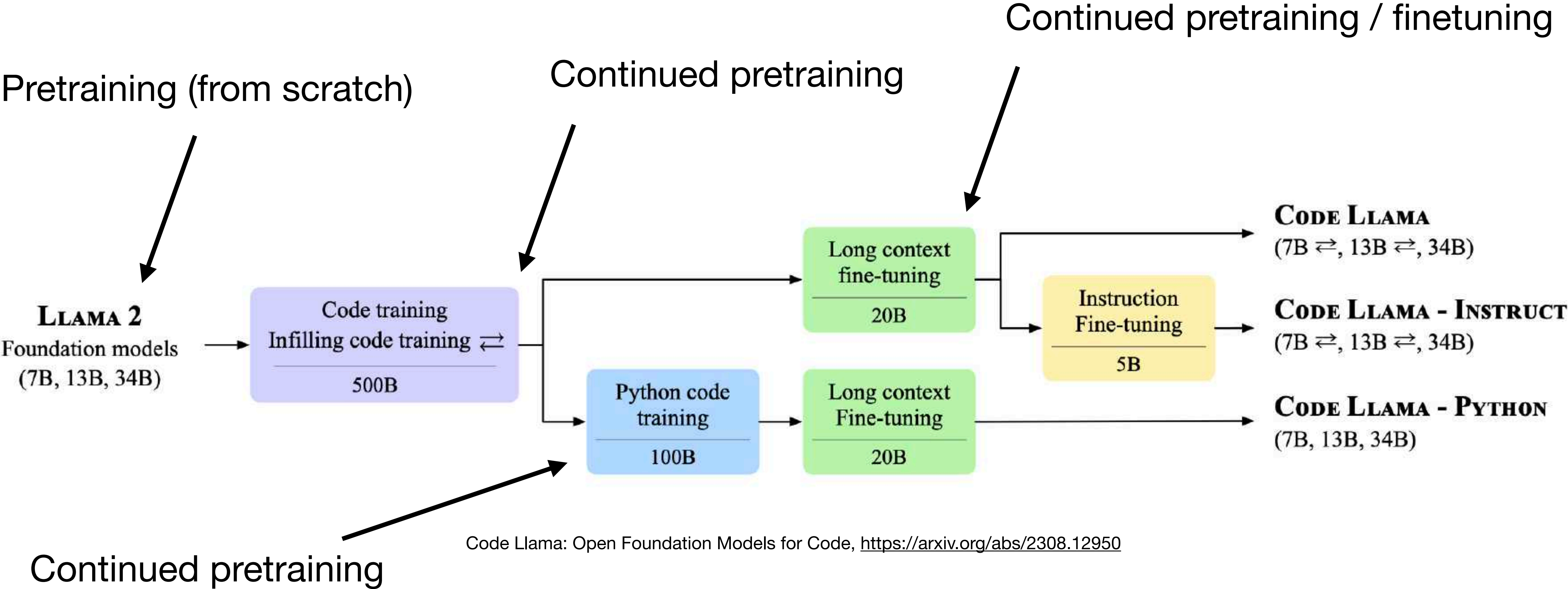
Code Llama: Open Foundation Models for Code, <https://arxiv.org/abs/2308.12950>

# CodeLlama example



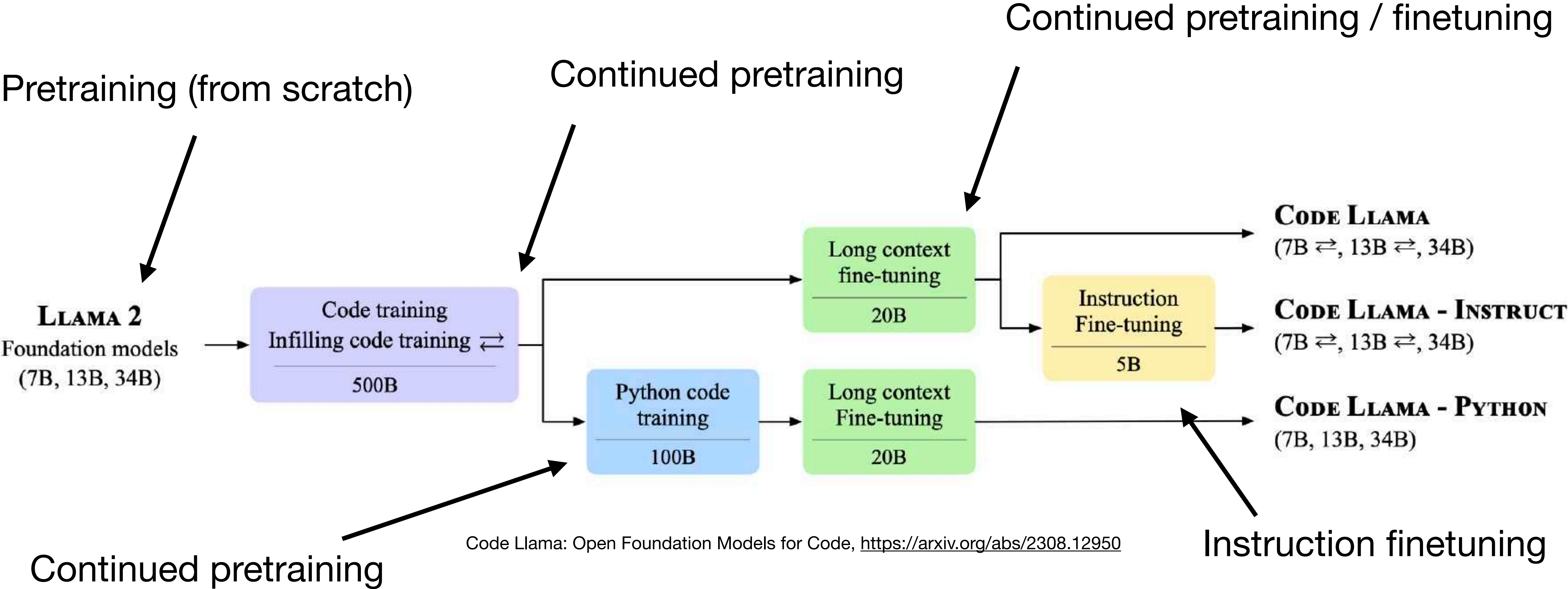
Code Llama: Open Foundation Models for Code, <https://arxiv.org/abs/2308.12950>

# CodeLlama example

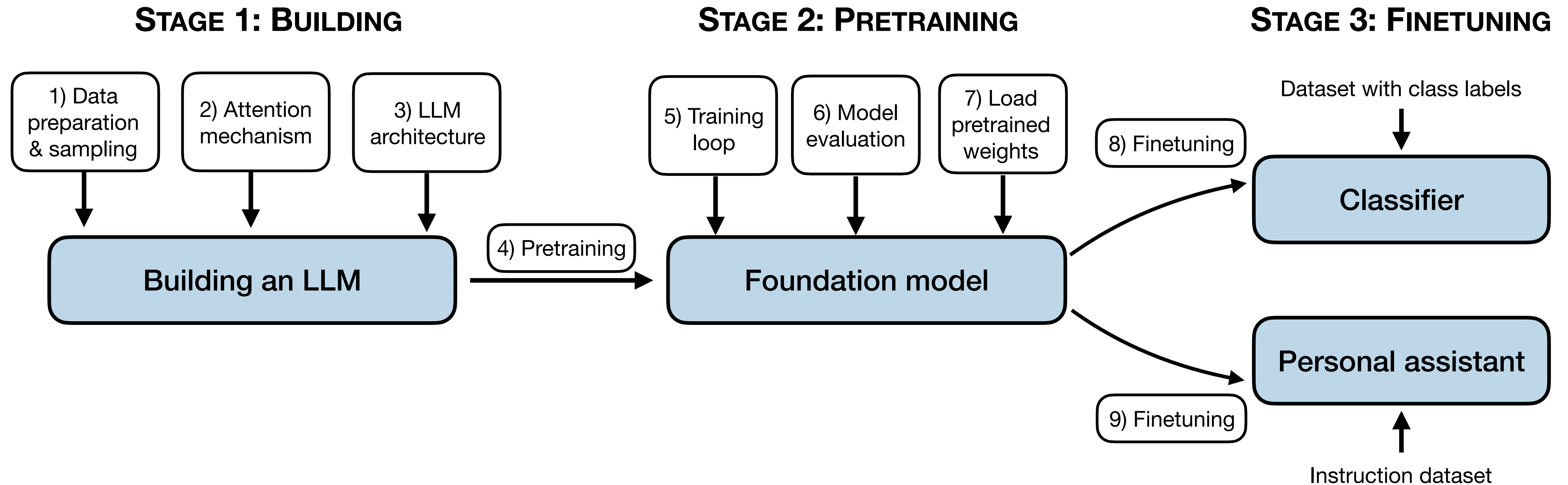


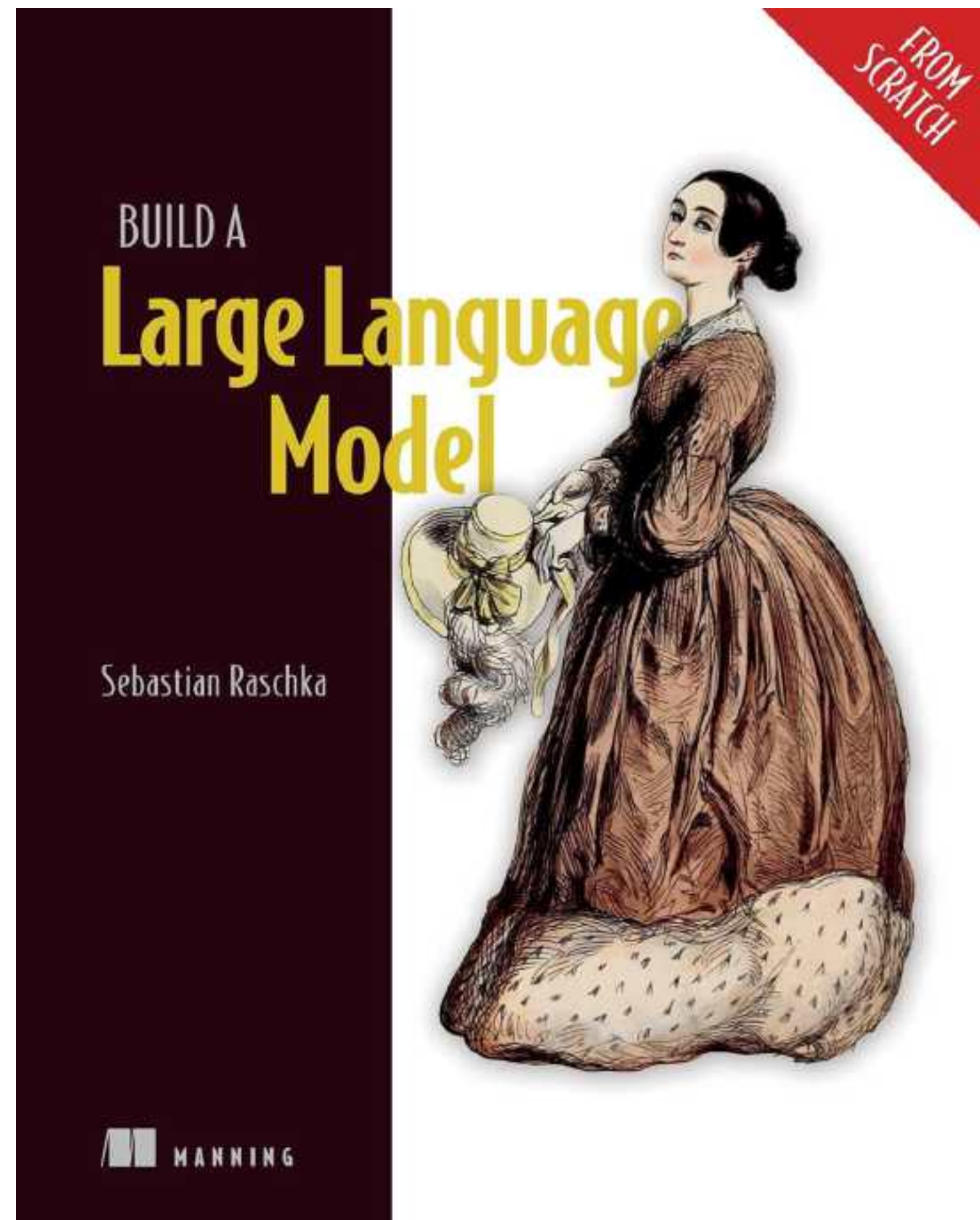


# CodeLlama example



# Developing an LLM





<https://mng.bz/M96o>

<https://sebastianraschka.com/books/>



## Simple. Powerful.

Zero setup. Persistent. Always ready.

Studio marries the simplicity of a **local development experience** with the power of **1,000s of cloud GPUs**, unlimited storage and multiplayer collaboration.



- ⚡ No environment setup.
- <> Code in the browser or connect your local IDE.
- ⚙️ **Switch from CPU to GPU with zero environment changes.**
- 🌐 Host and share AI apps. Streamlit. Gradio. React JS.
- 👥 Code together.
- 📁 Infinite storage. Upload, share files and connect S3 buckets.

<https://lightning.ai/>



Lightning AI

Source

Lightning AI Public

Explore

★ Featured

✓ Trending

🕒 Recent

🗃 All studios

👤 My studios

Educational

📖 Blogs

📄 Papers

📖 Tutorials

Workflows

🗃 Data processing

🔊 Endpoints

🔄 Training

🚀 Serving

⚙ Other

Model types

🔊 Audio

🖼 Image

🌟 Multimodal

💬 Text

📊 Tabular

HomeStudio templatesAgentsTeamspacesCommunityDocs

RAG 102

★ Featured

Chat with Documents

Document Chat Assistant using RAG

aniket

269

6.54 K

Improve LLMs via Proxy-Tuning

★ Featured

Improve LLMs With Proxy-Tuning

sebastian

47

7.88 K

Embed Wikipedia

★ Featured

Embed English Wikipedia under 5 dollars

thomasgridai

26

2.73 K

Finetune Hugging Face BERT

★ Featured

Finetune Hugging Face BERT with PyTorch Lig...

JG justin

97

1.98 K

Ingest documents (text, pdf, markdown, docx) in a vector database for Retrieval Augmented Generation (RAG)

Document Search and Retrieval using RAG

aniket

676

7.10 K

Data streaming benchmarks for ImageNet

★ Featured

Benchmark cloud data-loading libraries

thomasgridai

23

1.05 K

SlimPajama & Starcoder

★ Featured

Prepare the TinyLlama 1T token dataset

thomasgridai

38

1.64 K

LoRA from Scratch

★ Featured

Code LoRA from Scratch

sebastian

229

24.66 K

Optimized Inference API for Mistral 7B with vLLM

★ Featured

Optimized LLM inference API for Mistral 7B usi...

aniket

50

7.63 K

Sebastian Raschka

Building LLMs

# Contact

 @rasbt     in/sebastianraschka

 <https://sebastianraschka.com/contact/>

 <https://lightning.ai>

# Slides

 <https://sebastianraschka.com/pdf/slides/2024-build-llms.pdf>