CS 668 Applied Large Language Models
Spring Semester, 2026

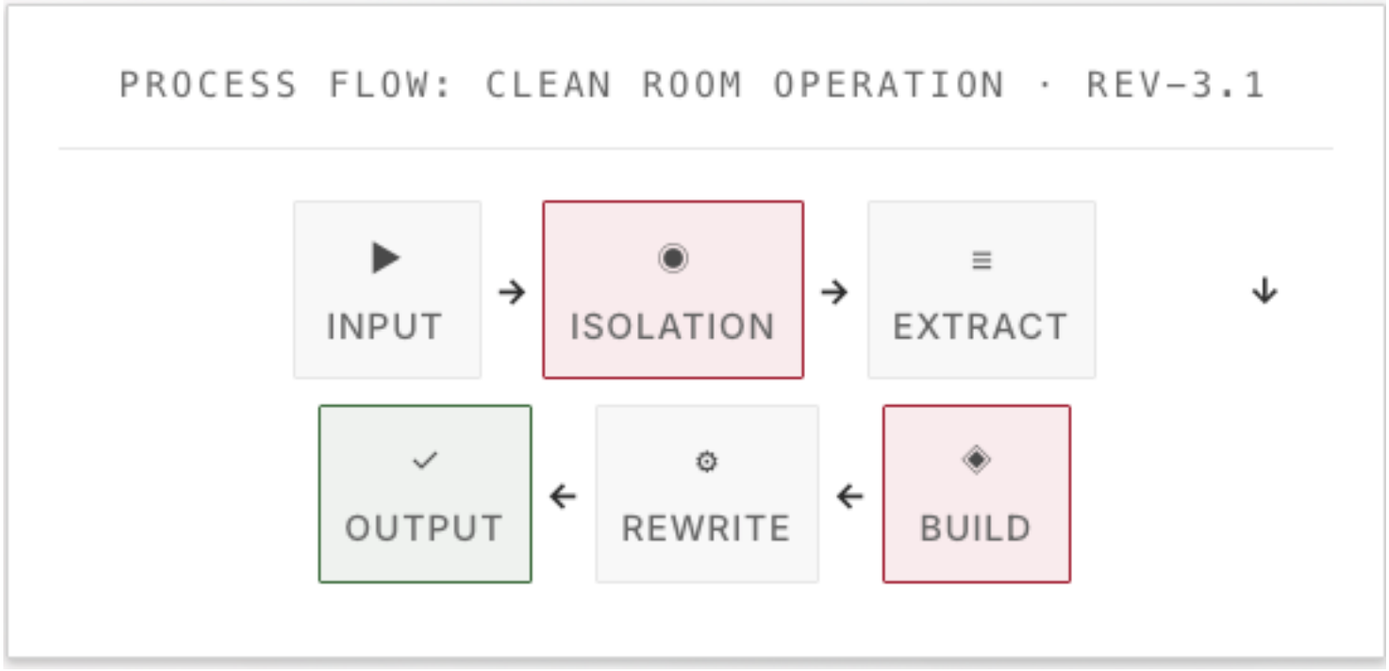
Doc 18 Fine-Tuning, Hooks, Continually Pre-train
Mar 12, 2026

MALUS - Clean Room as a Service

https://malus.sh

Our proprietary AI systems have never seen the original source code.

They independently analyze documentation, API specifications, and public interfaces to recreate functionally equivalent software from scratch.



MALUS - Clean Room as a Service

https://malus.sh

HOW PRICING WORKS

$0.01/KB

Every package is priced by its **unpacked size on npm**. We look up each dependency in your `package.json`, measure the size in kilobytes, and charge **$0.01 per KB**. That's it.

```
per package = max( $0.01, size_kb × $0.01 )
order total = max( $0.50, sum of all packages )
```

$0.50 minimum applies per order (Stripe processing floor). No base fee.

LIVE PRICES (FETCHED FROM OUR API)

Package	npm Size	Compute	If Ordered Alone
is-number	9 KB	$0.09	**$0.50***
left-pad	9 KB	$0.09	**$0.50***

CodeSpeak

https://codespeak.dev

CodeSpeak is a next-generation programming language powered by LLMs

You write a concise spec, codespeak build generates code

When you change the spec, codespeak build translates a diff in the spec → a diff in the code

Coming Soon

Turning Code into Specs

CodeSpeak Spec

EmlConverter

Converts RFC 5322 email files (.eml) to Markdown using Python's built-in `email` module.

Accepts

`.eml` extension or `message/rfc822` MIME type.

Output Structure
1. **Headers section**: From, To, Cc, Subject, Date as `**Key:** value` pairs
2. **Body**: plain text preferred; if only HTML, convert to markdown
3. **Attachments section** (if any): list with filename, MIME type, human-readable size

Parsing Requirements
- Decode RFC 2047 encoded headers (e.g., `=?UTF-8?B?...?=`)
- Decode body content (base64, quoted-printable)
- Handle multipart: walk parts, prefer `text/plain` over `text/html`
- For `message/rfc822` parts: recursively format as quoted nested message
- Extract attachment metadata without decoding attachment content

CodeSpeak Spec to Code

- Decode RFC 2047 encoded headers (e.g., `=?UTF-8?B?...?=`)

```
    def _decode_header_value(value: Optional[str]) -> str:
        """Decode RFC 2047 encoded header values."""
        if value is None:
            return ""
        decoded_parts: List[str] = []
        for part, charset in decode_header(value):
            if isinstance(part, bytes):
                if charset:
                    try:
                        decoded_parts.append(part.decode(charset))
                    except (UnicodeDecodeError, LookupError):
                        decoded_parts.append(part.decode("utf-8", errors="replace"))
                else:
                    decoded_parts.append(part.decode("utf-8", errors="replace"))
            else:
                decoded_parts.append(part)
    return "".join(decoded_parts)
```

Why ATMs didn’t kill bank teller jobs

https://davidoks.blog/p/why-the-atm-didnt-kill-bank-teller

JD Vance

> And the example I always give is the bank teller in the 1970s. There were very stark predictions of thousands, hundreds of thousands of bank tellers going out of a job. Poverty and commiseration.

What actually happens is we have more bank tellers today than we did when the ATM was created, but they’re doing slightly different work

Why ATMs didn’t kill bank teller jobs

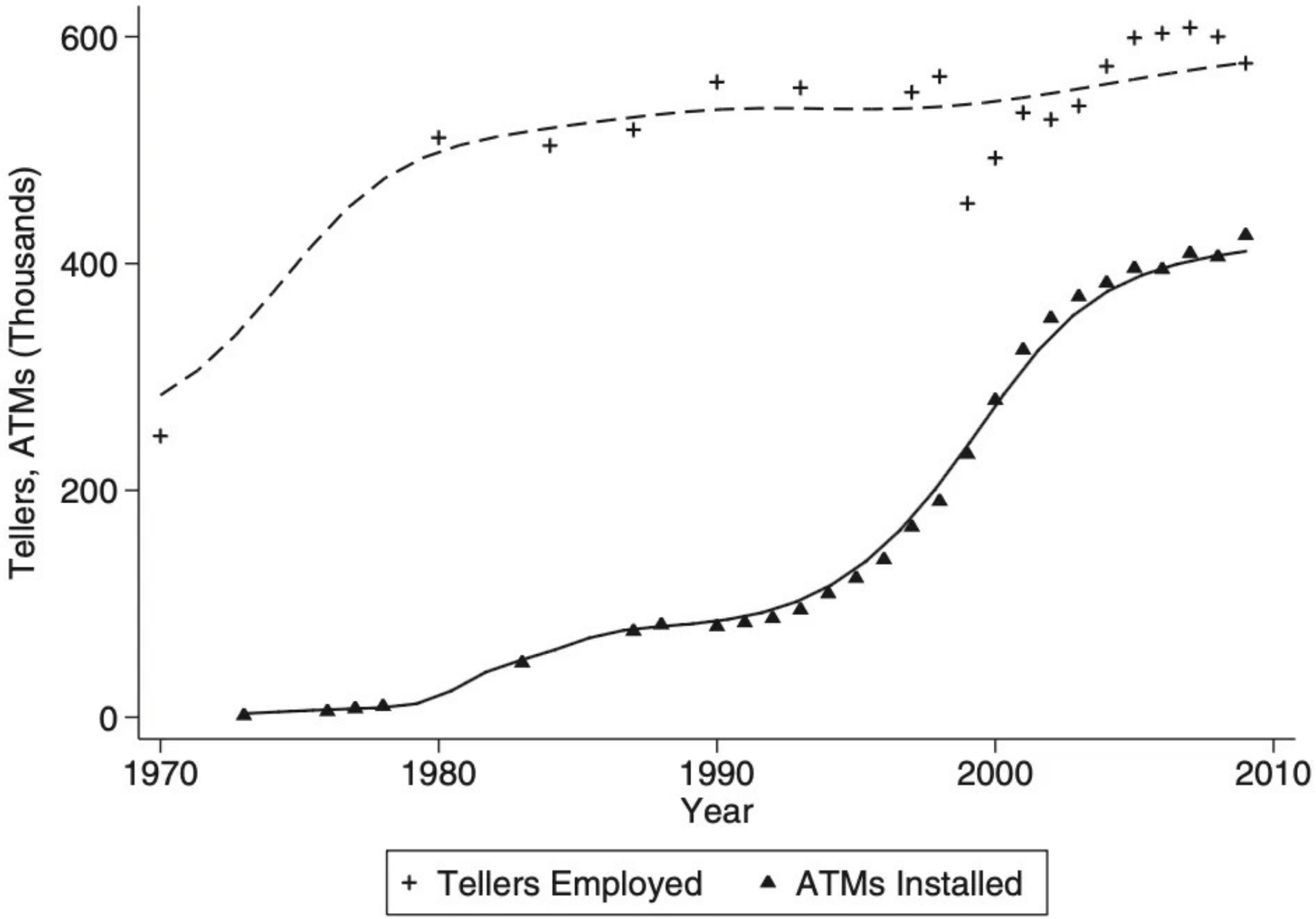
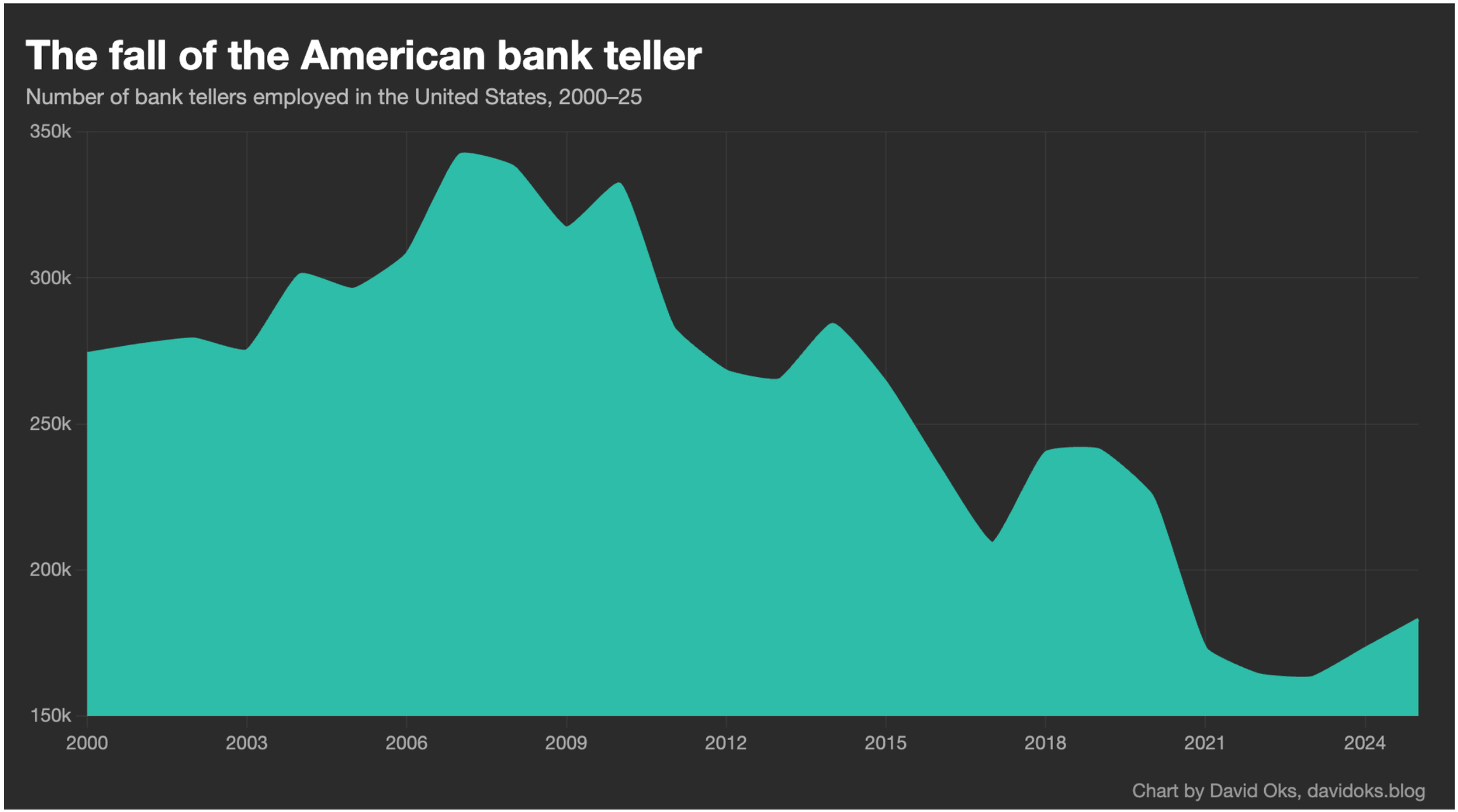


Figure 7.1. Adoption of automated teller machines did not reduce teller jobs.

Why ATMs didn’t kill bank teller jobs

The huge decline in bank teller employment that we’ve seen over the last 15-odd years is mainly a story about iPhones and what they made possible.



Why ATMs didn’t kill bank teller jobs

humans were getting expensive in the 1950s and ‘60s, so everyone wanted to reduce the human component, and so in that period you saw the rise of supermarkets and discount stores, where the whole innovation is getting the stuff yourself.

So in the 1950s and ‘60s, the goal of every single business that employed humans was to find ways to replace humans with machines

First, by reducing the cost of operating a bank branch, ATMs indirectly increased the demand for tellers: the number of tellers per branch fell by more than a third between 1988 and 2004, but the number of urban bank branches (also encouraged by a wave of bank deregulation allowing more branches) rose by more than 40 percent.

Why ATMs didn’t kill bank teller jobs

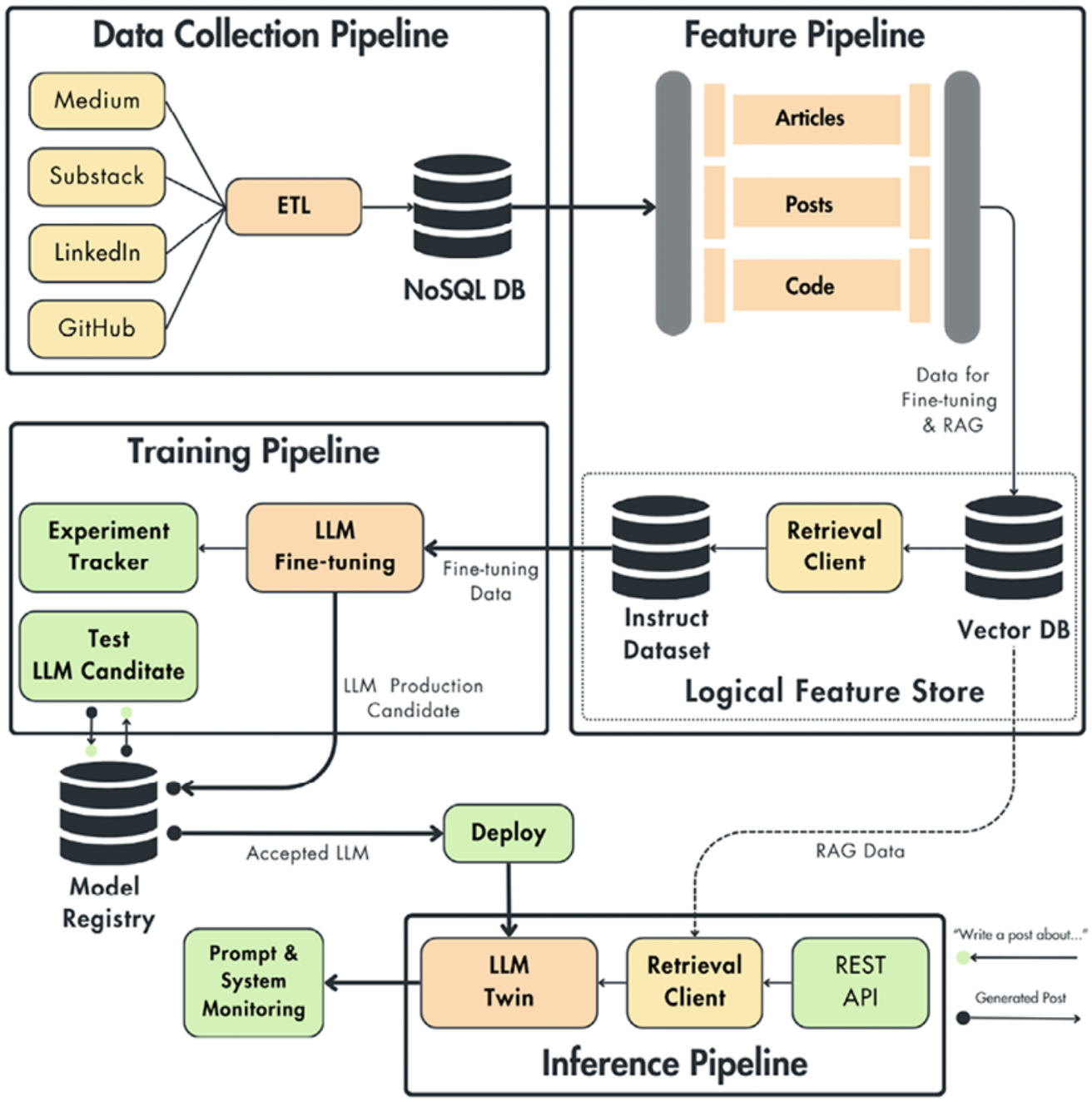
The mobile banking vision was simple: the banking customers of the future would do all their banking via their banks’ mobile apps.

The lesson is worth stating plainly. The ATM tried to do the teller’s job better, faster, cheaper; it tried to fit capital into a labor-shaped hole; but the iPhone made the teller’s job irrelevant. One automated tasks within an existing paradigm, and the other created a new paradigm in which those tasks simply didn’t need to exist at all. And it is paradigm replacement, not task automation, that actually displaces workers—and, conversely, unlocks the latent productivity within any technology.

The real productivity gains from AI—and the real threat of labor displacement—will come not from the “drop-in remote worker,” but from something like Dwarkesh Patel’s vision of the fully-automated firm.

Book & Papers

LLM Engineer’s Handbook, Lusztin, Labonne



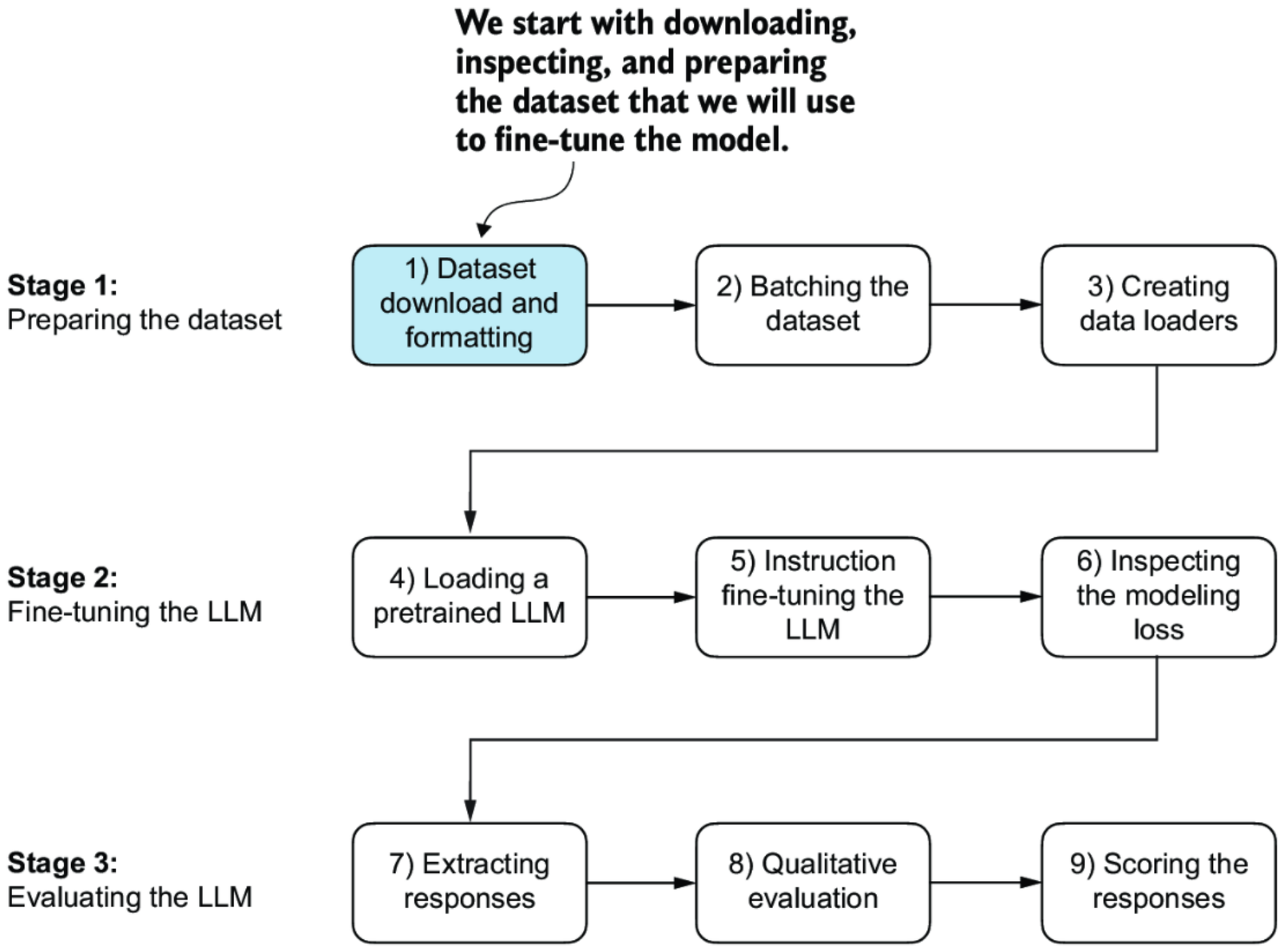
Book & Papers

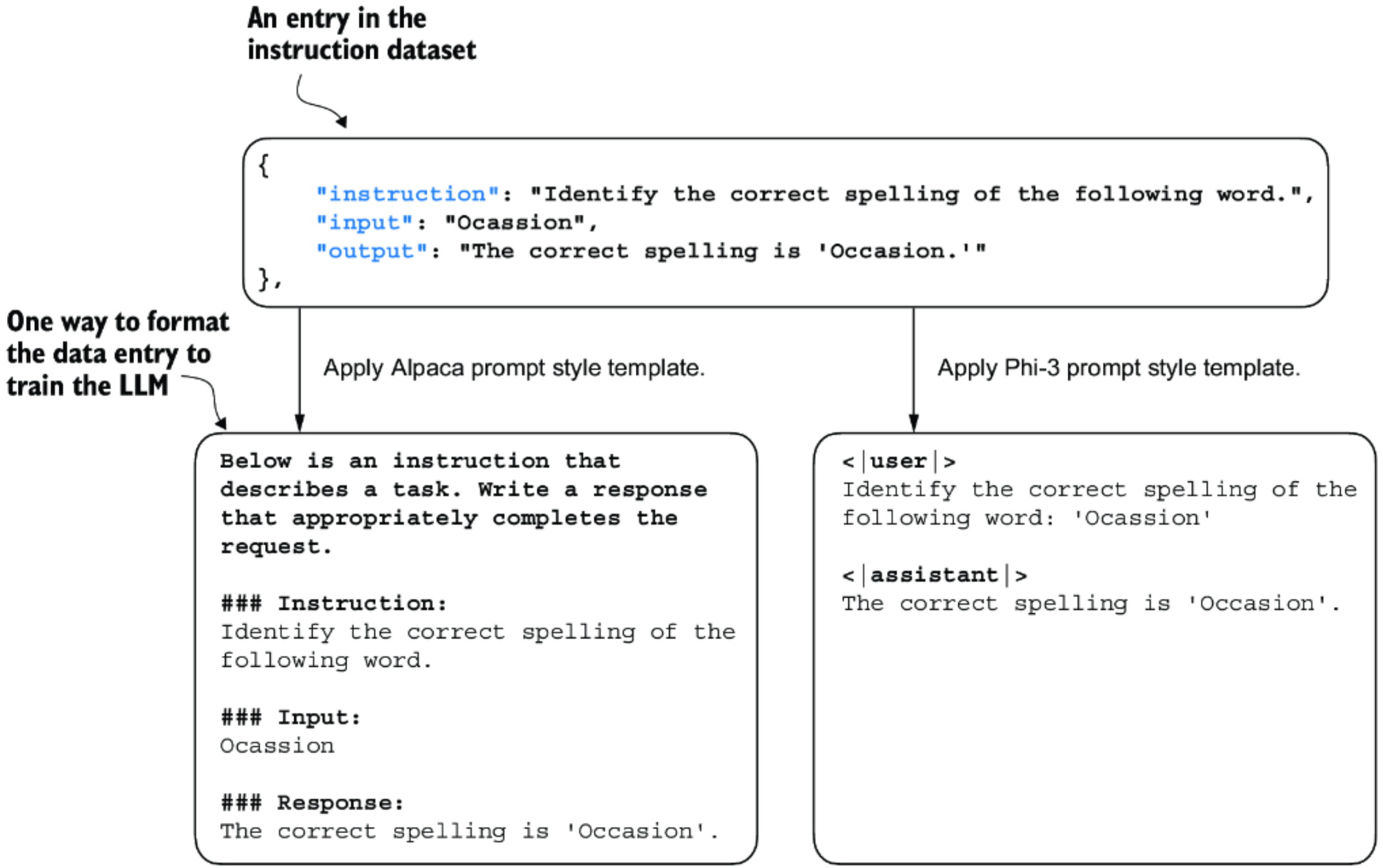
Hands-On Large Language Models

Hacker News

https://news.ycombinator.com

Instruction Fine-Tuning




An entry in the instruction dataset
{
"instruction": "Identify the correct spelling of the following word.",
"input": "Ocassion",
"output": "The correct spelling is 'Occasion.'"
},
One way to format the data entry to train the LLM
Apply Alpaca prompt style template.
Apply Phi-3 prompt style template.
Below is an instruction that describes a task. Write a response that appropriately completes the request.
Instruction:
Identify the correct spelling of the following word.
Input:
Ocassion
Response:
The correct spelling is 'Occasion'.
<|user|>
Identify the correct spelling of the following word: 'Ocassion'
<|assistant|>
The correct spelling is 'Occasion'.

Alpaca format

instruction: This is the instruction or prompt given to the language model.

input:

output:

```
{
  "instruction": "Write a short story about a cat who goes on an adventure.",
  "input": "",
  "output": "Whiskers twitched, Jasper the cat crept through the tall grass. He was on a mission,
to find the legendary catnip patch..."
  }


  {
    "instruction": "human instruction (required)",
    "input": "human input (optional)",
    "output": "model response (required)",
    "system": "system prompt (optional)",
    "history": [
      ["human instruction in the first round (optional)", "model response in the first round (optional)"
      ["human instruction in the second round (optional)", "model response in the second round
(optional)"]]}
```

Phi-3

<|system|>: Role for the model to assume. Ie. Python developer,

<|user|>: This is where you provide your actual prompt or question to the model.

<|assistant|>: This is where the model will generate its response.

<|system|> You are a helpful and informative AI assistant. <|end|>
<|user|> What are the main causes of climate change? <|end|>
<|assistant|>

Sample Data

{'instruction': 'Identify the correct spelling of the following word.',
'input': 'Ocassion',
‘output': "The correct spelling is 'Occasion.'"}

{'instruction': "What is an antonym of 'complicated'?",
'input': '',
‘output': "An antonym of 'complicated' is 'simple'."}

Formating

```
def format_input(entry):
    instruction_text = (
        f"Below is an instruction that describes a task. "
        f"Write a response that appropriately completes the request."
        f"\n\n### Instruction:\n{entry['instruction']}"
    )

    input_text = f"\n\n### Input:\n{entry['input']}" if entry["input"] else ""

    return instruction_text + input_text
```

Sample Formated Request

```
data = {'instruction': 'Identify the correct spelling of the following word.',
'input': 'Ocassion',
‘output': "The correct spelling is 'Occasion.'"}
```

```
model_input = format_input(data)
desired_response = f"\n\n### Response:\n{data['output']}"

print(model_input + desired_response)
```

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

### Instruction:
Identify the correct spelling of the following word.

### Input:
Ocassion

### Response:
The correct spelling is 'Occasion.'
```

InstructionDataset

```
import torch
from torch.utils.data import Dataset

class InstructionDataset(Dataset):
    def __init__(self, data, tokenizer):
        self.data = data

        self.encoded_texts = []
        for entry in data:
            instruction_plus_input = format_input(entry)
        response_text = f"\n\n### Response:\n{entry['output']}"
        full_text = instruction_plus_input + response_text
        self.encoded_texts.append(
            tokenizer.encode(full_text)
        )

def __getitem__(self, index):
    return self.encoded_texts[index]

def __len__(self):
    return len(self.data)
```

Preparing the Batchs

Tokenize data

Padding

- Each batch can be a different length

Adding targets

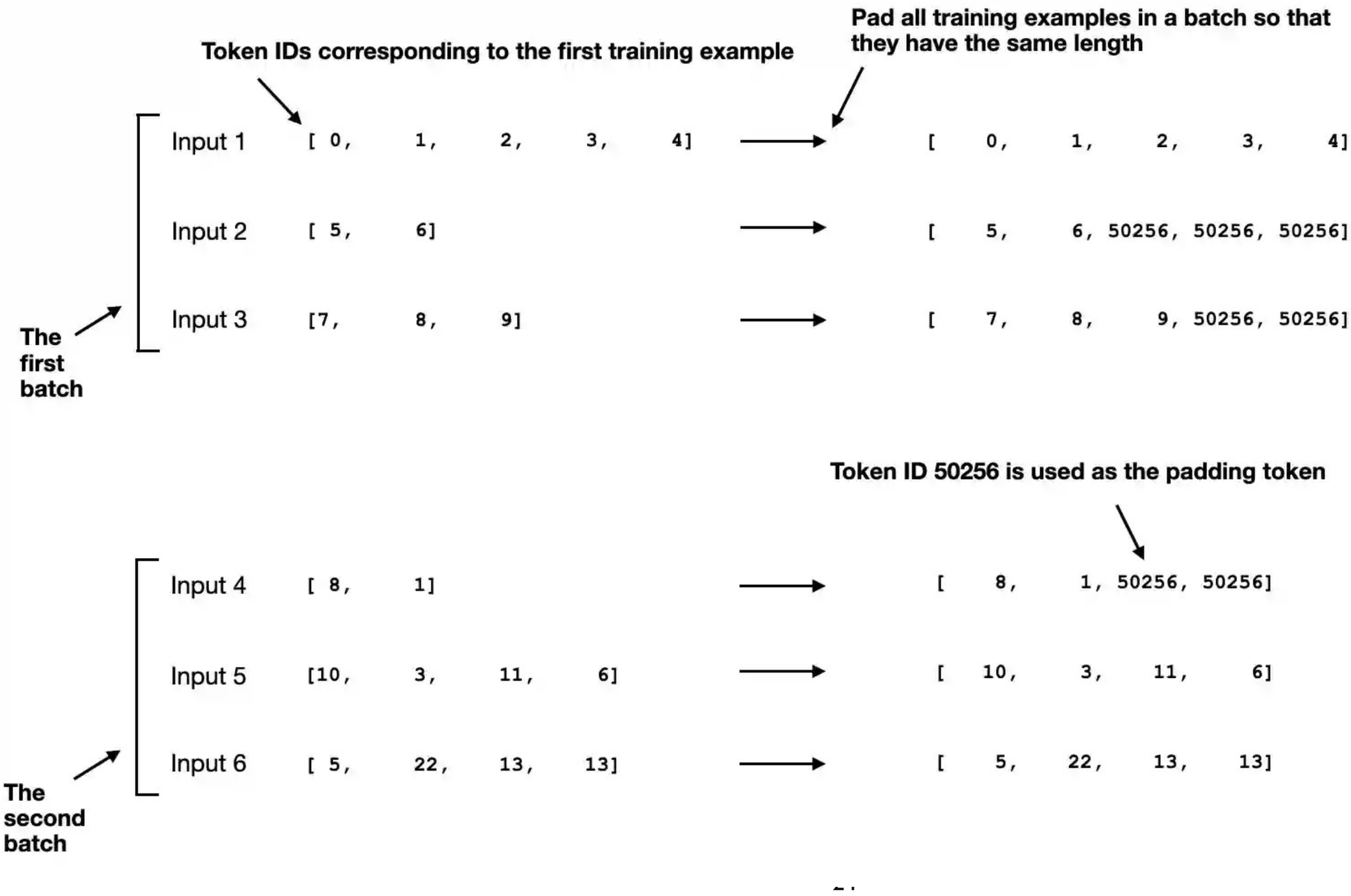
Maximum length

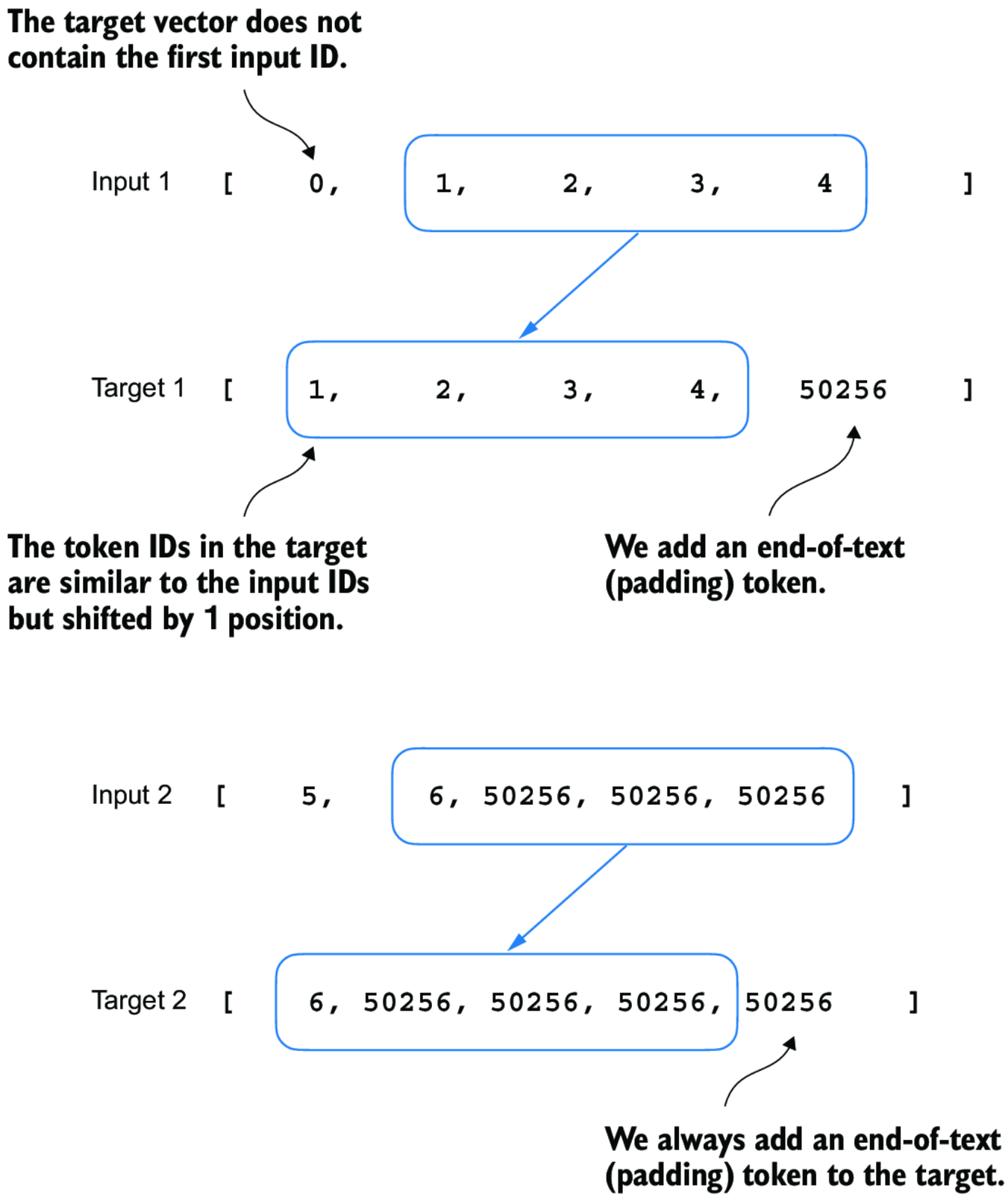
Ignore Index

- So loss function ignores padding
- cross_entropy ignores examples with label -100

Padding

Pad each patch individually so different lengths



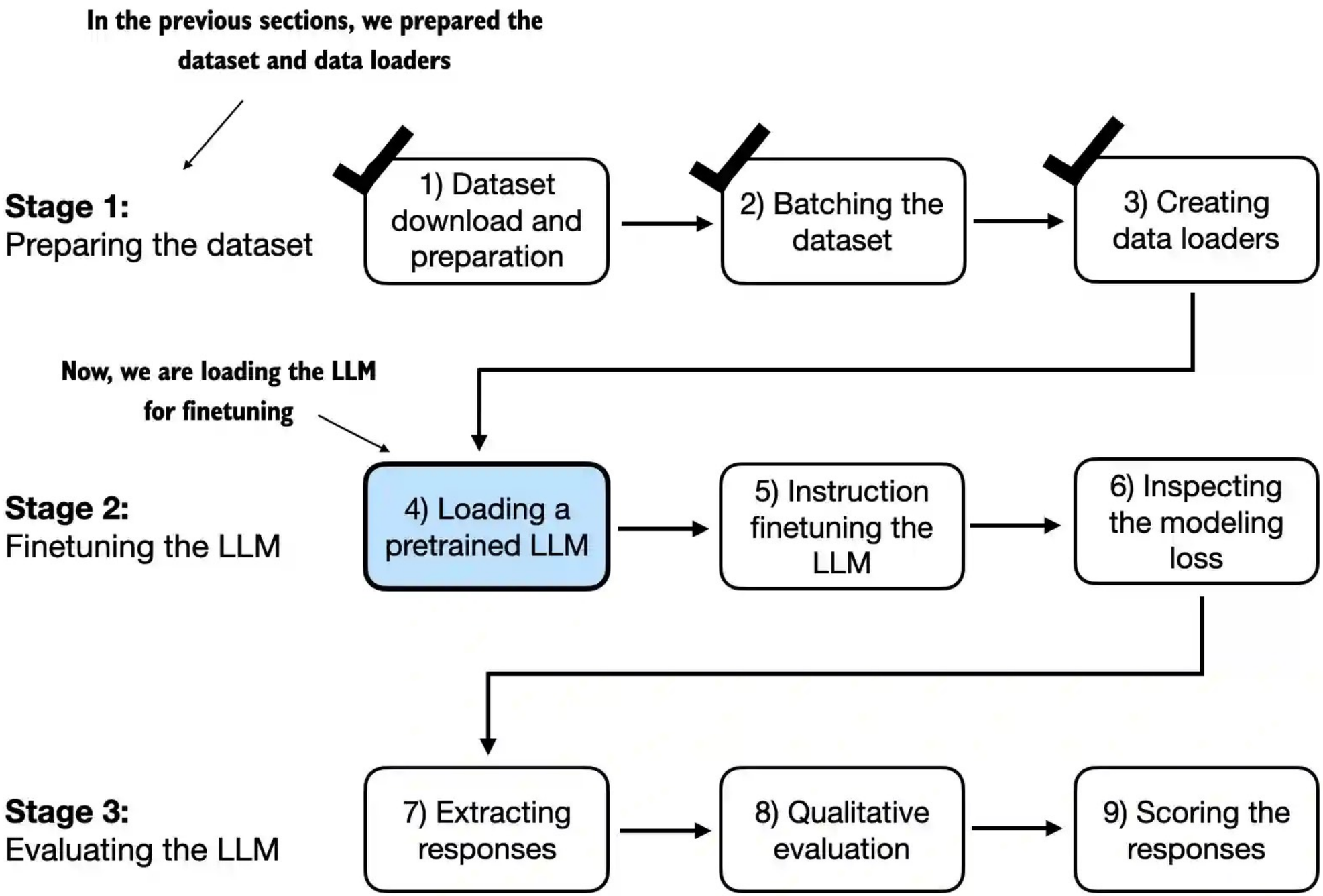

The target vector does not contain the first input ID.
Input 1 [0, 1, 2, 3, 4]
Target 1 [1, 2, 3, 4, 50256]
The token IDs in the target are similar to the input IDs but shifted by 1 position.
We add an end-of-text (padding) token.
Input 2 [5, 6, 50256, 50256, 50256]
Target 2 [6, 50256, 50256, 50256, 50256]
We always add an end-of-text (padding) token to the target.

Some Functional Magic

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

from functools import partial

customized_collate_fn = partial(
    custom_collate_fn,
    device=device,
    allowed_max_length=1024
)


customized_collate_fn(batch) =
    custom_collate_fn(batch, allowed_max_length=1024, device=device):
```

Dataset & Loader

```
rom torch.utils.data import DataLoader

num_workers = 0
batch_size = 8

torch.manual_seed(123)

train_dataset = InstructionDataset(train_data, tokenizer)
train_loader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    collate_fn=customized_collate_fn,
    shuffle=True,
    drop_last=True,
    num_workers=num_workers
)
```


In the previous sections, we prepared the dataset and data loaders
Stage 1:
Preparing the dataset
1) Dataset download and preparation
2) Batching the dataset
3) Creating data loaders
Now, we are loading the LLM for finetuning
Stage 2:
Finetuning the LLM
4) Loading a pretrained LLM
5) Instruction finetuning the LLM
6) Inspecting the modeling loss
Stage 3:
Evaluating the LLM
7) Extracting responses
8) Qualitative evaluation
9) Scoring the responses

```
from gpt_download import download_and_load_gpt2
from previous_chapters import GPTModel, load_weights_into_gpt

BASE_CONFIG = {
    "vocab_size": 50257,     # Vocabulary size
    "context_length": 1024,  # Context length
    "drop_rate": 0.0,        # Dropout rate
    "qkv_bias": True         # Query-key-value bias
}

model_configs = {
    "gpt2-small (124M)": {"emb_dim": 768, "n_layers": 12, "n_heads": 12},
    "gpt2-medium (355M)": {"emb_dim": 1024, "n_layers": 24, "n_heads": 16},
    "gpt2-large (774M)": {"emb_dim": 1280, "n_layers": 36, "n_heads": 20},
    "gpt2-xl (1558M)": {"emb_dim": 1600, "n_layers": 48, "n_heads": 25},
}

CHOOSE_MODEL = "gpt2-medium (355M)"
BASE_CONFIG.update(model_configs[CHOOSE_MODEL])

model_size = CHOOSE_MODEL.split(" ")[-1].lstrip("(").rstrip(")")
settings, params = download_and_load_gpt2(
    model_size=model_size,
    models_dir="gpt2"
)

model = GPTModel(BASE_CONFIG)
load_weights_into_gpt(model, params)
model.eval();
```

```
def calc_loss_loader(data_loader, model, device, num_batches=None):
    total_loss = 0.
    if len(data_loader) == 0:
        return float("nan")
    elif num_batches is None:
        num_batches = len(data_loader)
    else:
        # Reduce the number of batches to match the total number of batches in the data loader
        # if num_batches exceeds the number of batches in the data loader
        num_batches = min(num_batches, len(data_loader))
for i, (input_batch, target_batch) in enumerate(data_loader):
    if i < num_batches:
        loss = calc_loss_batch(input_batch, target_batch, model, device)
        total_loss += loss.item()
    else:
        break
return total_loss / num_batches
```

```
def train_model_simple(model, train_loader, val_loader, optimizer, device, num_epochs,
                       eval_freq, eval_iter, start_context, tokenizer):
    train_losses, val_losses, track_tokens_seen = [], [], []
    tokens_seen, global_step = 0, -1

    # Main training loop
    for epoch in range(num_epochs):
        model.train()  # Set model to training mode

        for input_batch, target_batch in train_loader:
            optimizer.zero_grad()  # Reset loss gradients from previous batch iteration
            loss = calc_loss_batch(input_batch, target_batch, model, device)
            loss.backward()  # Calculate loss gradients
            optimizer.step()  # Update model weights using loss gradients
            tokens_seen += input_batch.numel()
            global_step += 1

            if global_step % eval_freq == 0:
                train_loss, val_loss = evaluate_model(
                    model, train_loader, val_loader, device, eval_iter)
                train_losses.append(train_loss)
                val_losses.append(val_loss)
                track_tokens_seen.append(tokens_seen)
                print(f"Ep {epoch+1} (Step {global_step:06d}): "
                      f"Train loss {train_loss:.3f}, Val loss {val_loss:.3f}")

        generate_and_print_sample(
            model, tokenizer, device, start_context
        )

    return train_losses, val_losses, track_tokens_seen
```

```
import time

start_time = time.time()

torch.manual_seed(123)

optimizer = torch.optim.AdamW(model.parameters(), lr=0.00005, weight_decay=0.1)

num_epochs = 2

train_losses, val_losses, tokens_seen = train_model_simple(
    model, train_loader, val_loader, optimizer, device,
    num_epochs=num_epochs, eval_freq=5, eval_iter=5,
    start_context=format_input(val_data[0]), tokenizer=tokenizer
)

end_time = time.time()
execution_time_minutes = (end_time - start_time) / 60
print(f"Training completed in {execution_time_minutes:.2f} minutes.")
```

Initial losses

Training loss: 3.8390236377716063

Validation loss: 3.761903762817383

Ep 1 (Step 000000): Train loss 2.637, Val loss 2.626

Ep 1 (Step 000005): Train loss 1.174, Val loss 1.102

Ep 1 (Step 000010): Train loss 0.872, Val loss 0.945

Ep 1 (Step 000015): Train loss 0.856, Val loss 0.906

Ep 1 (Step 000020): Train loss 0.776, Val loss 0.881

Ep 1 (Step 000025): Train loss 0.753, Val loss 0.859

Ep 1 (Step 000030): Train loss 0.798, Val loss 0.836

Ep 2 (Step 000220): Train loss 0.299, Val loss 0.650

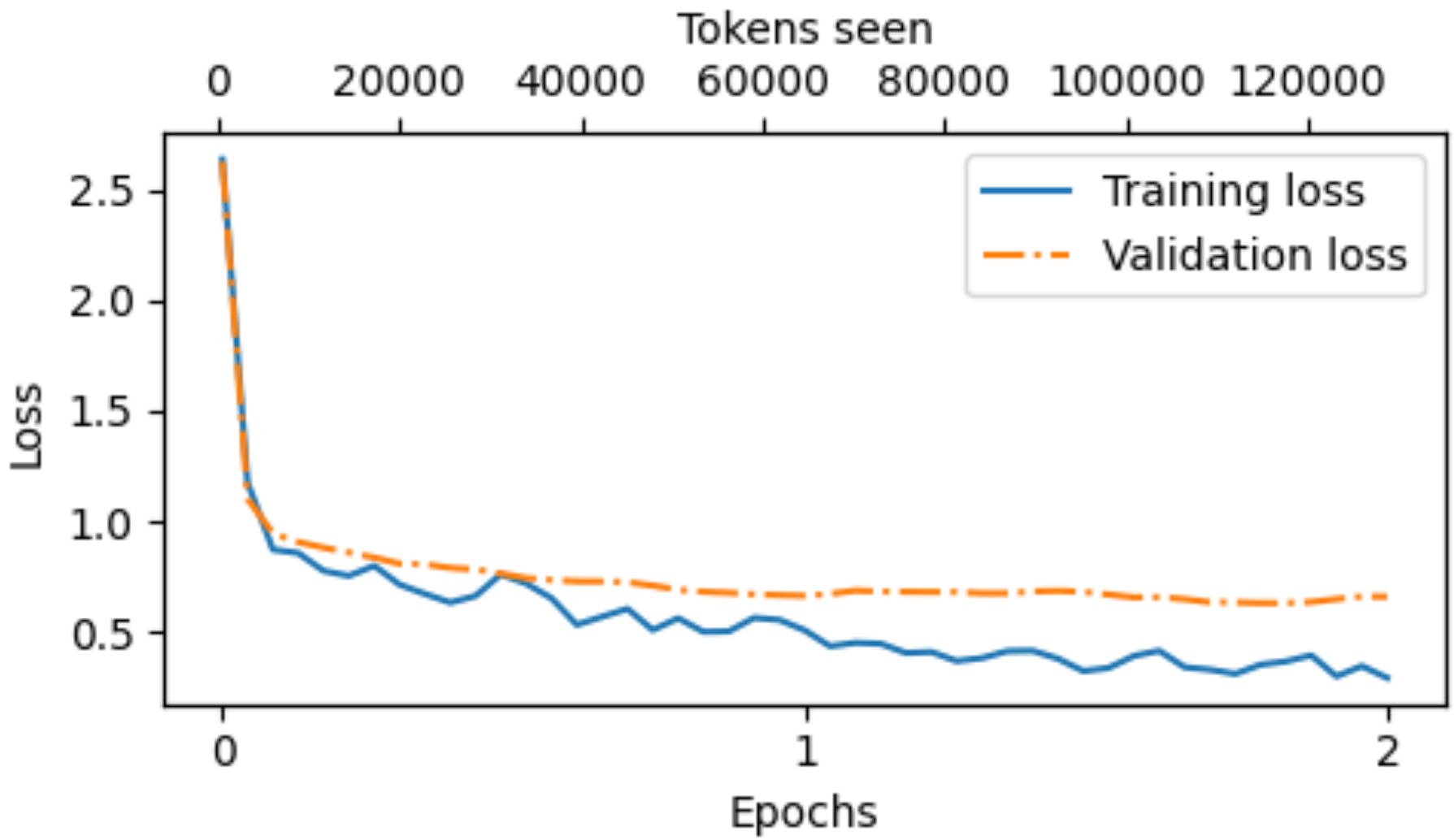
Ep 2 (Step 000225): Train loss 0.348, Val loss 0.662

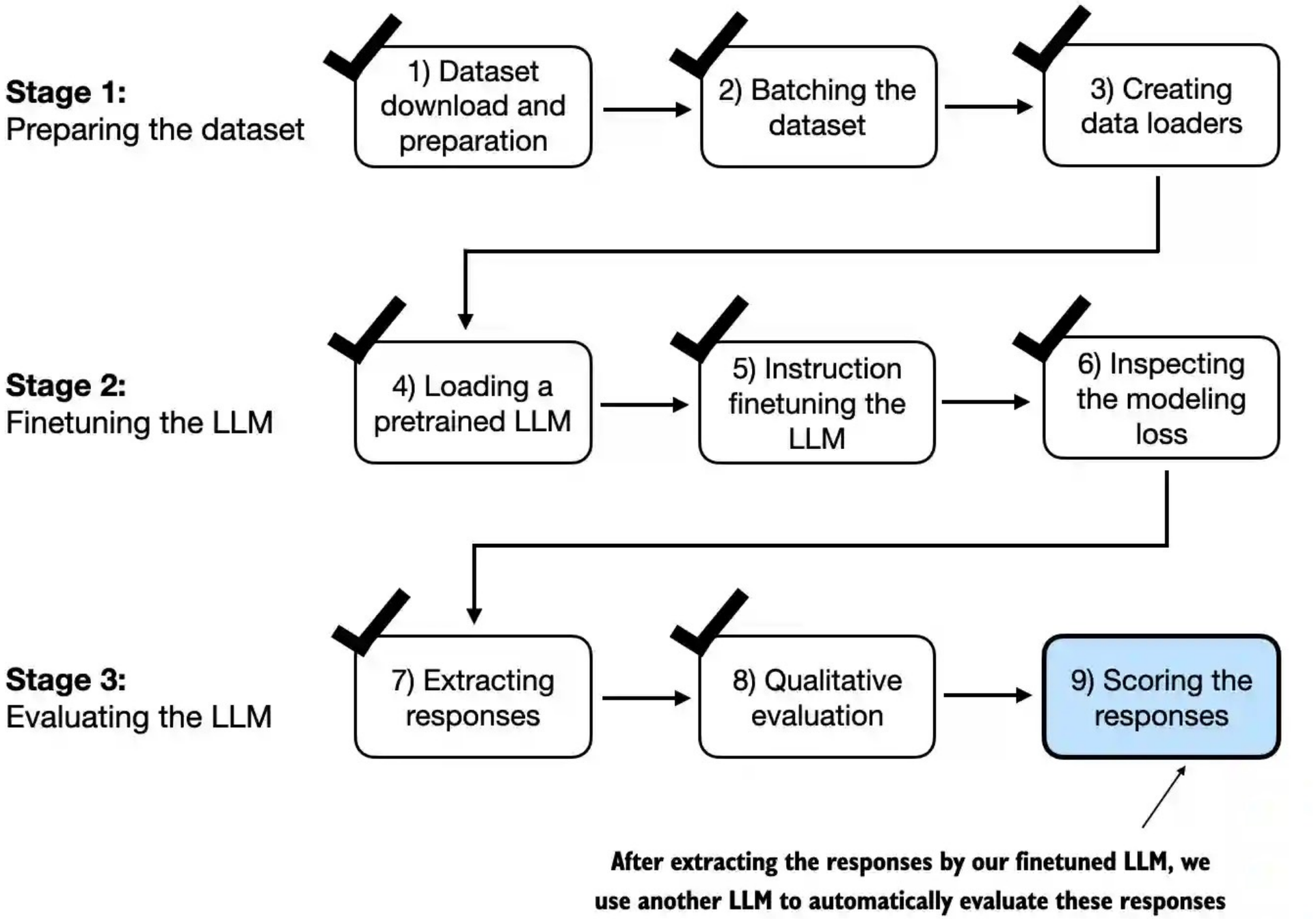
Ep 2 (Step 000230): Train loss 0.294, Val loss 0.657

Below is an instruction that describes a task. Write a response that appropriately completes the request. ### Instruction: Convert the active sentence to passive: 'The chef cooks the meal every day.' ### Response: The meal is cooked every day by the chef.<|endoftext|>The following is an instruction that describes a task. Write a response that appropriately completes the request. ### Instruction: What is the capital of the United Kingdom

Training completed in 22.82 minutes.

Plot saved as loss-plot-standalone.pdf


Tokens seen
0
20000
40000
60000
80000
100000
120000
2.5
2.0
1.5
1.0
0.5
Loss
Training loss
Validation loss
0
1
2
Epochs


Stage 1:
Preparing the dataset
1) Dataset download and preparation
2) Batching the dataset
3) Creating data loaders
Stage 2:
Finetuning the LLM
4) Loading a pretrained LLM
5) Instruction finetuning the LLM
6) Inspecting the modeling loss
Stage 3:
Evaluating the LLM
7) Extracting responses
8) Qualitative evaluation
9) Scoring the responses
After extracting the responses by our finetuned LLM, we use another LLM to automatically evaluate these responses

Save Model Responses

```
from tqdm import tqdm

for i, entry in tqdm(enumerate(test_data), total=len(test_data)):

    input_text = format_input(entry)

    token_ids = generate(
        model=model,
        idx=text_to_token_ids(input_text, tokenizer).to(device),
        max_new_tokens=256,
        context_size=BASE_CONFIG["context_length"],
        eos_id=50256
    )
    generated_text = token_ids_to_text(token_ids, tokenizer)
    response_text = generated_text[len(input_text):].replace("### Response:", "").strip()

    test_data[i]["model_response"] = response_text


with open("instruction-data-with-response.json", "w") as file:
    json.dump(test_data, file, indent=4)  # "indent" for pretty-printing
```

Sample Results

instruction"Correct the punctuation in the sentence."
input"Its time to go home."
output"The corrected sentence should be: 'It's time to go home.'"
model_response"The punctuation in the sentence is incorrect."

instruction"Classify the following items as either solid, liquid, or gas."
input"Mercury, oxygen, wood"
output"Mercury - Liquid Oxygen - Gas Wood - Solid"
model_response"Solid: Mercury Liquid: Oxygen Gas: Wood"

instruction"Edit the given text to ensure all plural nouns are spelled correctly."
input"The birds sings beautiful songs."
output"The birds sing beautiful songs."
model_response"The birds sings beautiful songs."

instruction"Generate a sentence using the word 'generous'."
input""
output"He is very generous and always helps those in need."
model_response"She was very generous and gave the poor man a meal."

Saving The Model

```
import re



file_name = f"{re.sub(r'[ ()]', '', CHOOSE_MODEL) }-sft.pth"
torch.save(model.state_dict(), file_name)
print(f"Model saved as {file_name}”)

# Load model via
# model.load_state_dict(torch.load(“gpt2-medium355M-sft.pth"))
```

Model	Device	Runtime for 2 Epochs
gpt2-medium (355M)	CPU (M3 MacBook Air)	15.78 minutes
gpt2-medium (355M)	GPU (M3 MacBook Air)	10.77 minutes
gpt2-medium (355M)	GPU (M4Max MacBookPro)	1.86 minutes
gpt2-medium (355M)	GPU (L4)	1.83 minutes
gpt2-medium (355M)	GPU (A100)	0.86 minutes
gpt2-small (124M)	CPU (M3 MacBook Air)	5.74 minutes
gpt2-small (124M)	GPU (M3 MacBook Air)	3.73 minutes
gpt2-small (124M)	GPU (L4)	0.69 minutes
gpt2-small (124M)	GPU (A100)	0.39 minutes

Evaluation Techniques

Short-answer and multiple choice benchmarks

MMLU ("Measuring Massive Multitask Language Understanding", https://arxiv.org/abs/2009.03300)

Human preference comparison to other LLMs,

LMSYS chatbot arena (https://arena.lmsys.org)

Automated conversational benchmarks,

Another LLM evaluates the responses

AlpacaEval (https://tatsu-lab.github.io/alpaca_eval/)

Using Ollama to Evauate Model

```
import urllib.request

def query_model(
    prompt,
    model="llama3",
    url="http://127.0.0.1:11434/api/chat/"
):
    # Create the data payload as a dictionary
    data = {
    "model": model,
    "messages": [
        {"role": "user", "content": prompt}
    ],
    "options": {     # Settings below are required for deterministic responses
        "seed": 123,
        "temperature": 0,
        "num_ctx": 2048
    }
}
```

Using Ollama to Evauate Model

```
    # Convert the dictionary to a JSON formatted string and encode it to bytes
    payload = json.dumps(data).encode("utf-8")

    request = urllib.request.Request(
        url,
        data=payload,
        method="POST"
    )
    request.add_header("Content-Type", "application/json")

    # Send the request and capture the response
    response_data = ""
    with urllib.request.urlopen(request) as response:
        # Read and decode the response
        while True:
            line = response.readline().decode("utf-8")
            if not line:
                break
            response_json = json.loads(line)
            response_data += response_json["message"]["content"]

    return response_data

model = "llama3"
result = query_model("What do Llamas eat?", model)
print(result)
```

```
for entry in test_data[:3]:
    prompt = (
        f"Given the input `{format_input(entry)}` "
        f"and correct output `{entry['output']}`, "
        f"score the model response `{entry['model_response']}`"
        f" on a scale from 0 to 100, where 100 is the best score. "
    )
    print("\nDataset response:")
    print(">>", entry['output'])
    print("\nModel response:")
    print(">>", entry["model_response"])
    print("\nScore:")
    print(">>", query_model(prompt))
    print("\n-------------------------")
```

instruction"Rewrite the sentence using a simile."
input"The car is very fast."
output"The car is as fast as lightning."
model_response"The car is as fast as a bullet."

Dataset response:
>> The car is as fast as lightning.

Model response:
>> The car is as fast as a bullet.

Score:
>> I'd rate the model response "The car is as fast as a bullet." an 85 out of 100.

Here's why:

* The response uses a simile correctly, comparing the speed of the car to something else (in this case, a bullet).
* The comparison is relevant and makes sense, as bullets are known for their high velocity.
* The phrase "as fast as" is used correctly to introduce the simile.

The only reason I wouldn't give it a perfect score is that some people might find the comparison slightly less vivid or evocative than others. For example, comparing something to lightning (as in the original response) can be more dramatic and attention-grabbing. However, "as fast as a bullet" is still a strong and effective simile that effectively conveys the idea of the car's speed.

Overall, I think the model did a great job!

The Test

```
def generate_model_scores(json_data, json_key, model="llama3"):
    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 integer number only."
        )
        score = query_model(prompt, model)
        try:
            scores.append(int(score))
        except ValueError:
            print(f"Could not convert score: {score}")
            continue

    return scores

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

```
Number of scores: 110 of 110
Average score: 50.32
```

```
Number of scores: 110 of 110
Average score: 47.63
```

```
Llama 3 8B base model score: 58.51
Llama 3 8B instruct model score: 82.65
```

Better Evaluation Prompt

prompt = """

You are a fair judge assistant tasked with providing clear, objective feedback based on specific criteria, ensuring each assessment reflects the absolute standards set for performance.

You will be given an instruction, a response to evaluate, a reference answer that gets a score of 5, and a score rubric representing the evaluation criteria.

Write a detailed feedback that assess the quality of the response strictly based on the given score rubric, not evaluating in general.

Please do not generate any other opening, closing, and explanations.

Here is the rubric you should use to build your answer:

1: The response fails to address the instructions, providing irrelevant, incorrect, or excessively verbose information that detracts from the user's request.

2: The response partially addresses the instructions but includes significant inaccuracies, irrelevant details, or excessive elaboration that detracts from the main task.

3: The response follows the instructions with some minor inaccuracies or omissions. It is generally relevant and clear, but may include some unnecessary details or could be more concise.

4: The response adheres to the instructions, offering clear, accurate, and relevant information in a concise manner, with only occasional, minor instances of excessive detail or slight lack of clarity.

5: The response fully adheres to the instructions, providing a clear, accurate, and relevant answer in a concise and efficient manner. It addresses all aspects of the request without unnecessary details or elaboration

Provide your feedback as follows:

Feedback:::

Evaluation: (your rationale for the rating, as a text)

Total rating: (your rating, as a number between 1 and 5)

You MUST provide values for 'Evaluation:' and 'Total rating:' in your answer.

Now here is the instruction, the reference answer, and the response.

Instruction: {instruction}

Reference Answer: {reference}

Answer: {answer}

Provide your feedback. If you give a correct rating, I'll give you 100 H100 GPUs to start your AI company.

Feedback:::

Evaluation: """

https://github.com/rasbt/LLMs-from-scratch/discussions/449

Hooks for Models

Debugging

Inspect intermediate activations and gradients

Logging

Profiling

Custom Training

Implement custom gradient modifications or training strategies.

Feature Extraction

Extract intermediate representations of the input.

Model Surgery

Modify the model's behavior by changing activations or gradients.

Hooks for Models

Methods on torch.nn.Module

register_full_backward_hook(hook, prepend=False)

Called every time the gradients with respect to a module are computed

register_forward_pre_hook(hook, *, prepend=False, with_kwargs=False)

Called every time before forward() is invoked

register_forward_hook(hook, *, prepend=False, with_kwargs=False, always_call=False)

Called every time after forward()

register_load_state_dict_post_hook(hook)

Run after module’s load_state_dict() is called

```
import torch
from transformers import BertModel

def my_hook(module, input, output):
    print(f"Hook called on layer: {module}")
    print("Input shape:", input)
    print("Output shape:", output[0].shape)
    return None # or modify the output if needed

model = BertModel.from_pretrained('bert-base-uncased')

target_layer = model.encoder.layer[4]  # Access a specific layer

handle = target_layer.register_forward_hook(my_hook)

input_text = "The quick brown fox jumps over the lazy dog."
input_ids = tokenizer(input_text, return_tensors='pt').input_ids

with torch.no_grad():
    output = model(input_ids)

handle.remove()
```

```
Hook called on layer: BertLayer(
  (attention): BertAttention(
    (self): BertSdpaSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
    (intermediate_act_fn): GELUActivation()
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
Input shape: (tensor([[[ 0.1237, -0.6409, -0.5723,  ...,  0.4785,  0.3859,  0.4612],
         [-0.2302, -0.1894,  0.3409,  ...,  0.5503,  0.3314, -1.3512],
         [ 0.1004, -1.6581,  0.8088,  ...,  0.0929,  0.2219, -1.2052],
         ...,
         [ 1.5722, -0.5156,  0.1883,  ...,  -0.1616, -0.5337,  0.3734]
```

Now to get Top Tokens - Functions

```
import torch.nn.functional as F
import torch

activations = {}

def get_hook(layer_num):
    def hook(model,input,output):
        activations[layer_num] = output[0].detach() # not just last token, entire set of activations
    return hook

def register_hooks(model):
    list_of_hooks = []
    for i in range(32):
        list_of_hooks.append(model.model.layers[i-1].register_forward_hook(get_hook(i)))
    return list_of_hooks
```

Decoding an LLM’s Thoughts: Logit Lens in Just 25 Lines of Code, Nikhil Anand
https://ai.plainenglish.io/decoding-an-llms-thoughts-logit-lens-in-just-25-lines-of-code-100c1dbf2ac0

Now to get Top Tokens - Function

```
def get_top_tokens(model, activations):
    top_tokens = [ ]
    token_pos = -1

    for layer in range(32):
        probabilities = F.softmax(model.lm_head(model.model.norm(activations[layer][0,token_pos,:])),dim=0)
        max_index = torch.argmax(probabilities)
        top_tokens.append(tokenizer.batch_decode([max_index]))
    return top_tokens
```

Model & Prompt

```
from transformers import AutoModelForCausalLM, AutoTokenizer

device = torch.device("cpu")

MODEL_ID = "mistralai/Mistral-7B-Instruct-v0.2"
model = AutoModelForCausalLM.from_pretrained(MODEL_ID,torch_dtype=torch.float16).eval()
tokenizer = AutoTokenizer.from_pretrained(MODEL_ID)
model.to(device)
```

```
prompt = "Trump works at McDonald's. Trump works at"

all_hooks = register_hooks(model)
tokenizer.pad_token = "<s>"
eos_token = tokenizer.eos_token_id
input_ids = tokenizer(prompt,return_tensors="pt",padding=True).input_ids.to(device)
fwd_pass = model(input_ids)
```

```
top_tokens = get_top_tokens(model, activations, tokenizer)
for i in range(32):
    print(i, " ", top_tokens[i][0])

  for hook in all_hooks:
      hook.remove()
```

```
0   McDonald
1   least
2   op
3   op
4   op
5   op
6   McDonald
7   prompt
8   aval
9   fucking
10   WH
11   WH
12   EO
13   amber
14   typen
15   mechanics
16   jobs
17   jobs
18   jobs
19   McDonald
20   McDonald
21   McDonald
22   McDonald
23   McDonald
24   McDonald
25   McDonald
26   McDonald
27   McDonald
28   McDonald
29   McDonald
30   McDonald
31   McDonald
```

Logit Lens

Article

https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens

Notebook

https://colab.research.google.com/drive/1MjdfK2srcerLrAJDRaJQKO0sUiZ-hQtA?usp=sharing#scrollTo=Dm8E7OcBbqi1

Looks at top-1 token after each layer

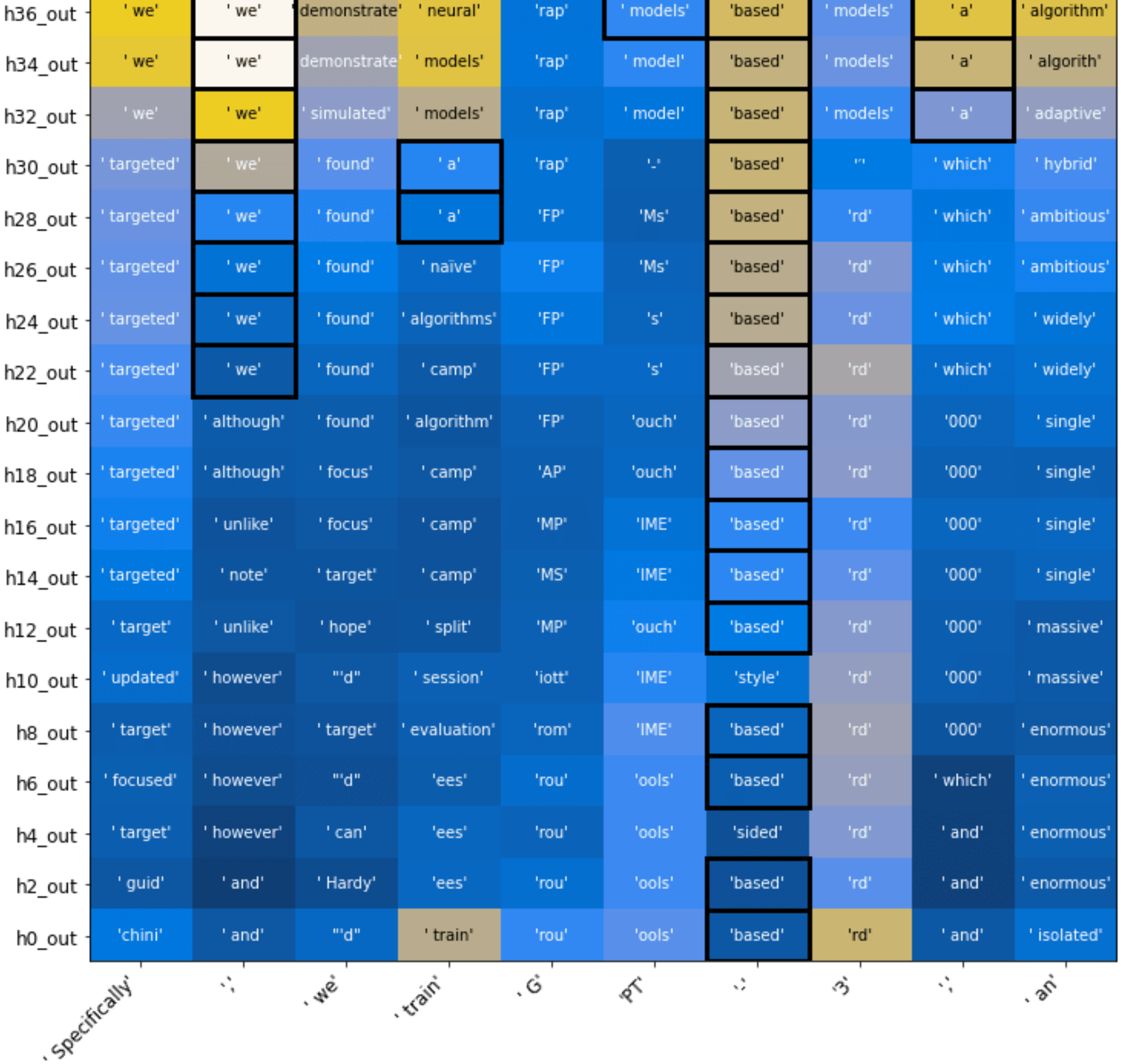
Looks at the rank of the final token in each layer

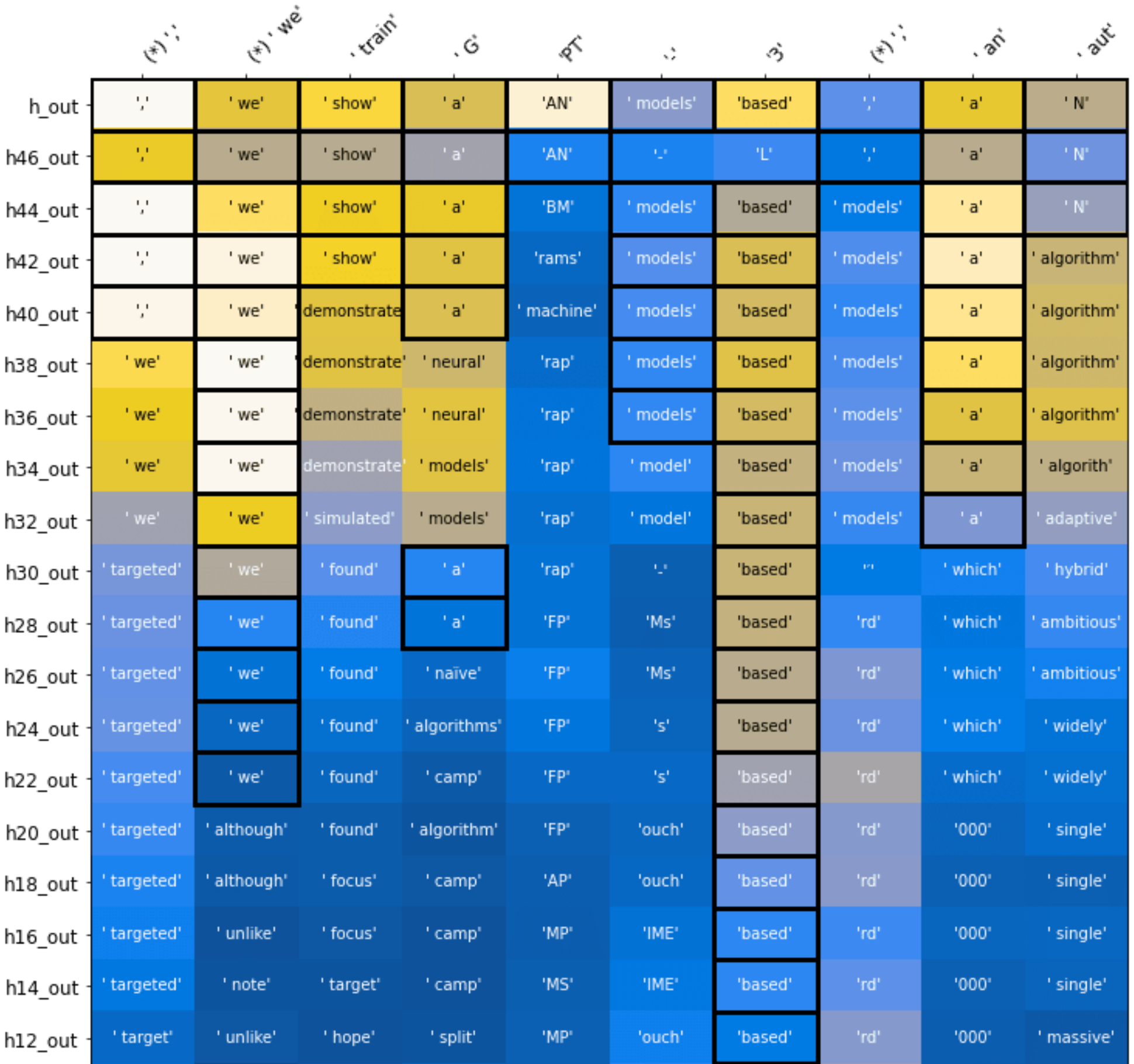
Input

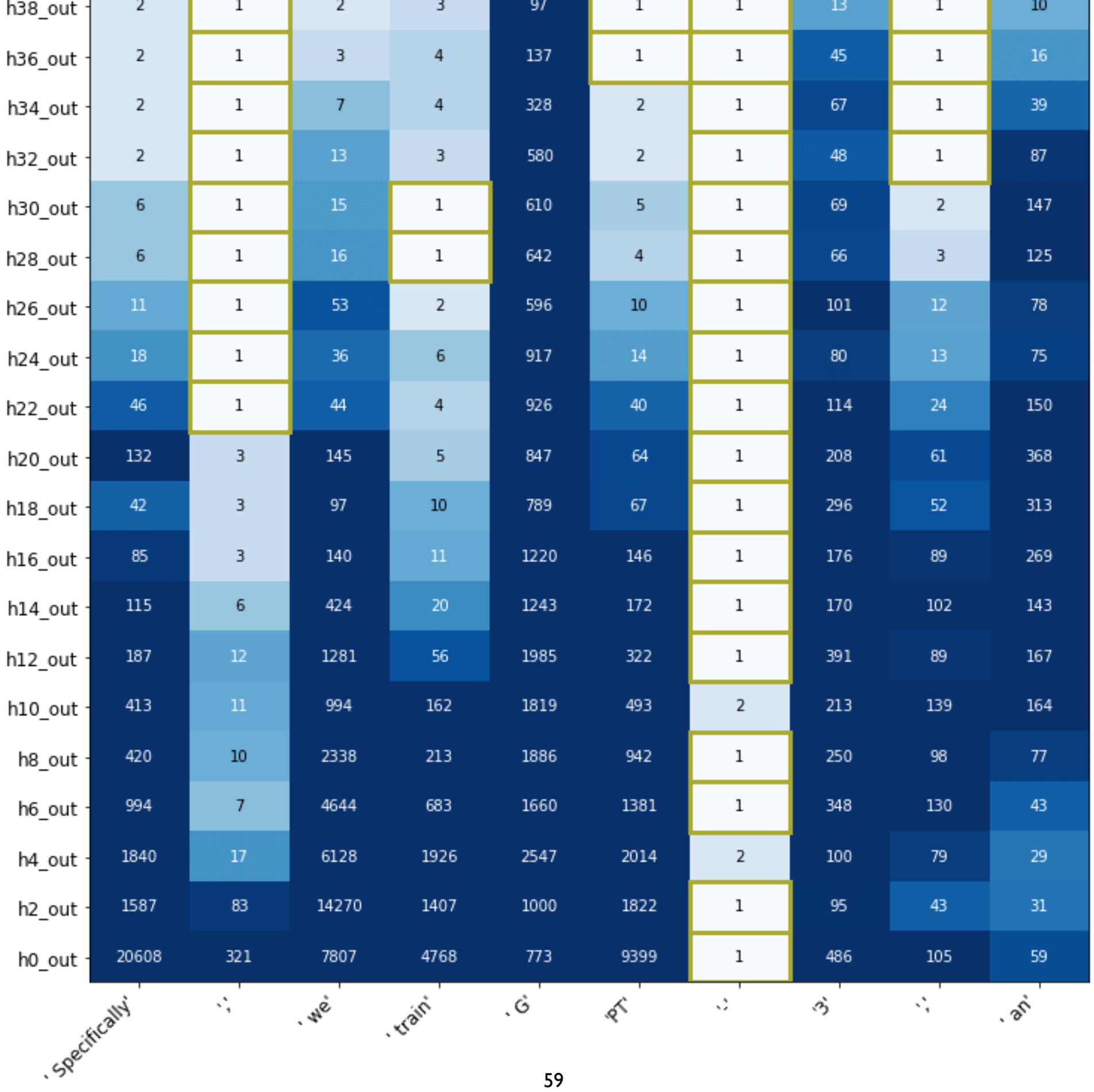
Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters

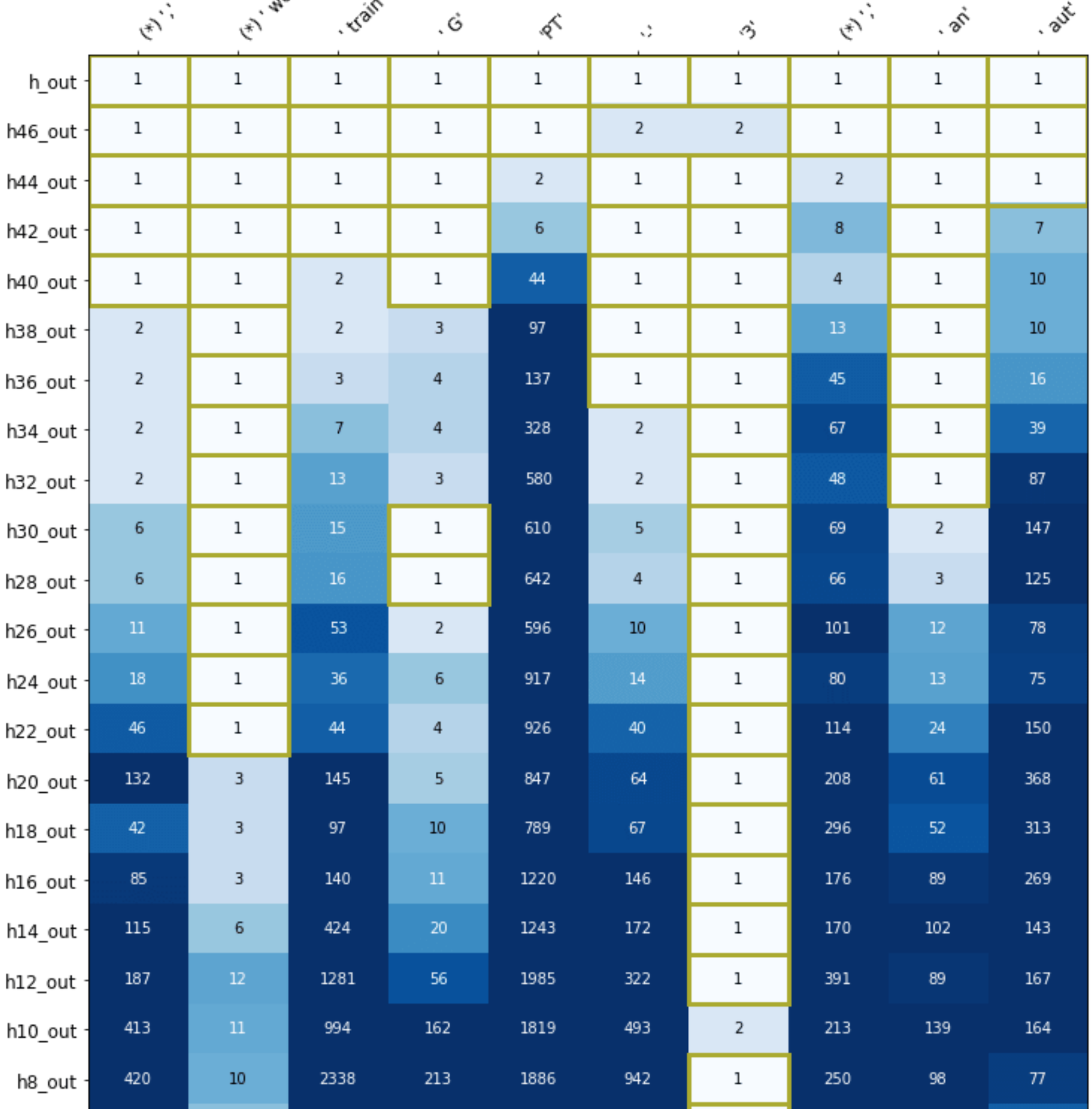
Tokens

"Specifically" "," "we" "train" "G" "PT" "-" "3" "," "an" "aut" "ore" "gressive"
“language" "model" "with" "175" "billion" "parameters


h36_out ' we' ' we' demonstrate' ' neural' 'rap' ' models' 'based' ' models' ' a' ' algorithm'
h34_out ' we' ' we' demonstrate' ' models' 'rap' ' model' 'based' ' models' ' a' ' algorith'
h32_out ' we' ' we' ' simulated' ' models' 'rap' ' model' 'based' ' models' ' a' ' adaptive'
h30_out ' targeted' ' we' ' found' ' a' 'rap' '-' 'based' '' ' which' ' hybrid'
h28_out ' targeted' ' we' ' found' ' a' 'FP' 'Ms' 'based' 'rd' ' which' ' ambitious'
h26_out ' targeted' ' we' ' found' ' naïve' 'FP' 'Ms' 'based' 'rd' ' which' ' ambitious'
h24_out ' targeted' ' we' ' found' ' algorithms' 'FP' 's' 'based' 'rd' ' which' ' widely'
h22_out ' targeted' ' we' ' found' ' camp' 'FP' 's' 'based' 'rd' ' which' ' widely'
h20_out ' targeted' ' although' ' found' ' algorithm' 'FP' 'ouch' 'based' 'rd' '000' ' single'
h18_out ' targeted' ' although' ' focus' ' camp' 'AP' 'ouch' 'based' 'rd' '000' ' single'
h16_out ' targeted' ' unlike' ' focus' ' camp' 'MP' 'IME' 'based' 'rd' '000' ' single'
h14_out ' targeted' ' note' ' target' ' camp' 'MS' 'IME' 'based' 'rd' '000' ' single'
h12_out ' target' ' unlike' ' hope' ' split' 'MP' 'ouch' 'based' 'rd' '000' ' massive'
h10_out ' updated' ' however' "'d" ' session' 'iott' 'IME' 'style' 'rd' '000' ' massive'
h8_out ' target' ' however' ' target' ' evaluation' 'rom' 'IME' 'based' 'rd' '000' ' enormous'
h6_out ' focused' ' however' "'d" 'ees' 'rou' 'ools' 'based' 'rd' ' which' ' enormous'
h4_out ' target' ' however' ' can' 'ees' 'rou' 'ools' 'sided' 'rd' ' and' ' enormous'
h2_out ' guid' ' and' ' Hardy' 'ees' 'rou' 'ools' 'based' 'rd' ' and' ' enormous'
h0_out 'chini' ' and' "'d" ' train' 'rou' 'ools' 'based' 'rd' ' and' ' isolated'
' Specifically' ',' ' we' ' train' ' G' 'PT' ':' '3' ',' ' an'

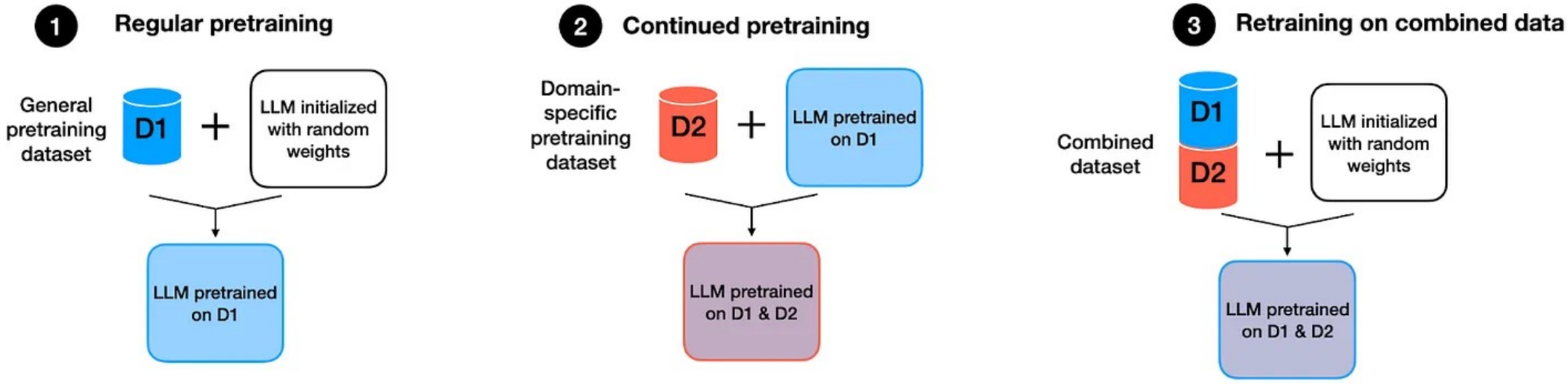

(*) ' ,'
(*) ' we'
' train'
' G'
'PT'
'.'
'3'
(*) ' ,'
' an'
' aut'
h_out ',' ' we' ' show' ' a' 'AN' ' models' 'based' ',' ' a' ' N'
h46_out ',' ' we' ' show' ' a' 'AN' '-' 'L' ',' ' a' ' N'
h44_out ',' ' we' ' show' ' a' 'BM' ' models' 'based' ' models' ' a' ' N'
h42_out ',' ' we' ' show' ' a' 'rams' ' models' 'based' ' models' ' a' ' algorithm'
h40_out ',' ' we' ' demonstrate' ' a' ' machine' ' models' 'based' ' models' ' a' ' algorithm'
h38_out ' we' ' we' ' demonstrate' ' neural' 'rap' ' models' 'based' ' models' ' a' ' algorithm'
h36_out ' we' ' we' ' demonstrate' ' neural' 'rap' ' models' 'based' ' models' ' a' ' algorithm'
h34_out ' we' ' we' ' demonstrate' ' models' 'rap' ' model' 'based' ' models' ' a' ' algorith'
h32_out ' we' ' we' ' simulated' ' models' 'rap' ' model' 'based' ' models' ' a' ' adaptive'
h30_out ' targeted' ' we' ' found' ' a' 'rap' '-' 'based' '' ' which' ' hybrid'
h28_out ' targeted' ' we' ' found' ' a' 'FP' 'Ms' 'based' 'rd' ' which' ' ambitious'
h26_out ' targeted' ' we' ' found' ' naïve' 'FP' 'Ms' 'based' 'rd' ' which' ' ambitious'
h24_out ' targeted' ' we' ' found' ' algorithms' 'FP' 's' 'based' 'rd' ' which' ' widely'
h22_out ' targeted' ' we' ' found' ' camp' 'FP' 's' 'based' 'rd' ' which' ' widely'
h20_out ' targeted' ' although' ' found' ' algorithm' 'FP' 'ouch' 'based' 'rd' '000' ' single'
h18_out ' targeted' ' although' ' focus' ' camp' 'AP' 'ouch' 'based' 'rd' '000' ' single'
h16_out ' targeted' ' unlike' ' focus' ' camp' 'MP' 'IME' 'based' 'rd' '000' ' single'
h14_out ' targeted' ' note' ' target' ' camp' 'MS' 'IME' 'based' 'rd' '000' ' single'
h12_out ' target' ' unlike' ' hope' ' split' 'MP' 'ouch' 'based' 'rd' '000' ' massive'


h38_out 2 1 2 3 97 1 1 13 1 10
h36_out 2 1 3 4 137 1 1 45 1 16
h34_out 2 1 7 4 328 2 1 67 1 39
h32_out 2 1 13 3 580 2 1 48 1 87
h30_out 6 1 15 1 610 5 1 69 2 147
h28_out 6 1 16 1 642 4 1 66 3 125
h26_out 11 1 53 2 596 10 1 101 12 78
h24_out 18 1 36 6 917 14 1 80 13 75
h22_out 46 1 44 4 926 40 1 114 24 150
h20_out 132 3 145 5 847 64 1 208 61 368
h18_out 42 3 97 10 789 67 1 296 52 313
h16_out 85 3 140 11 1220 146 1 176 89 269
h14_out 115 6 424 20 1243 172 1 170 102 143
h12_out 187 12 1281 56 1985 322 1 391 89 167
h10_out 413 11 994 162 1819 493 2 213 139 164
h8_out 420 10 2338 213 1886 942 1 250 98 77
h6_out 994 7 4644 683 1660 1381 1 348 130 43
h4_out 1840 17 6128 1926 2547 2014 2 100 79 29
h2_out 1587 83 14270 1407 1000 1822 1 95 43 31
h0_out 20608 321 7807 4768 773 9399 1 486 105 59
' Specifically' ',' ' we' ' train' ' G' 'PT' '.' '3' ',' ' an'


(*) ';'
(*) ' we
' train
' G
'PT'
':'
'3'
(*) ';'
' an'
' aut'
h_out 1 1 1 1 1 1 1 1 1 1
h46_out 1 1 1 1 1 2 2 1 1 1
h44_out 1 1 1 1 2 1 1 2 1 1
h42_out 1 1 1 1 6 1 1 8 1 7
h40_out 1 1 2 1 44 1 1 4 1 10
h38_out 2 1 2 3 97 1 1 13 1 10
h36_out 2 1 3 4 137 1 1 45 1 16
h34_out 2 1 7 4 328 2 1 67 1 39
h32_out 2 1 13 3 580 2 1 48 1 87
h30_out 6 1 15 1 610 5 1 69 2 147
h28_out 6 1 16 1 642 4 1 66 3 125
h26_out 11 1 53 2 596 10 1 101 12 78
h24_out 18 1 36 6 917 14 1 80 13 75
h22_out 46 1 44 4 926 40 1 114 24 150
h20_out 132 3 145 5 847 64 1 208 61 368
h18_out 42 3 97 10 789 67 1 296 52 313
h16_out 85 3 140 11 1220 146 1 176 89 269
h14_out 115 6 424 20 1243 172 1 170 102 143
h12_out 187 12 1281 56 1985 322 1 391 89 167
h10_out 413 11 994 162 1819 493 2 213 139 164
h8_out 420 10 2338 213 1886 942 1 250 98 77

	* ','	* ' we'	' train'	' G'	'PT'	* '-'	'3'	','	' an'	' aut'	'ore'	* 'gressive
h.11	1	1	1	1	1	1	1	1	1	1	1	1
h.11.attn	1	1	1	1	2	2	1	1	1	1	12	1
h.10	1	1	2	2	2	5	1	7	2	4	9	1
h.10.attn	1	1	13	2	3	9	1	5	3	5	49	1
h.9	1	1	10	5	3	9	1	1	2	111	46	1
h.9.attn	1	1	28	2	7	5	1	1	4	101	146	2
h.8	1	1	26	4	14	10	1	1	7	215	181	2
h.8.attn	1	1	71	9	78	5	1	1	36	305	251	2
h.7	1	1	80	6	83	8	1	2	27	337	235	2
h.7.attn	4	2	139	18	290	5	1	2	32	333	180	13
h.6	3	3	167	10	325	9	1	3	39	442	185	15
h.6.attn	9	1	367	14	264	7	1	13	59	167	307	32
h.5	4	1	337	12	227	16	1	22	46	159	348	36
h.5.attn	8	2	745	24	366	22	1	45	36	124	332	52
h.4	7	2	1083	19	311	36	1	63	35	129	382	60
h.4.attn	106	2	1579	21	369	28	2	24	37	49	421	124
h.3	88	2	1010	20	320	31	2	37	50	70	678	137
h.3.attn	81	3	1486	32	435	40	2	147	65	55	1456	151
h.2	61	3	1082	46	309	74	1	181	60	91	2086	218
h.2.attn	92	4	1219	489	312	57	1	204	36	105	3321	173
h.1	84	11	962	627	323	89	1	431	43	114	3858	346
h.1.attn	46	7	919	750	249	169	1	578	28	88	8056	337
h.0	38	16	768	1080	308	197	1	624	29	70	10226	1183
h.0.attn	39	9	447	293	2170	9	43	801	21	236	27514	12550
input	49679	1672	16204	28749	39955	39484	2915	32429	12650	44023	35529	5129
	' Specifically'	','	' we'	' train'	' G'	'PT'	'-'	'3'	','	' an'	' aut'	'ore'

Continually Pre-train Large Language Models



Regular pretraining:

Initializing a model with random weights and pretraining it on dataset D1.

Continued pretraining: T

Taking the pretrained model from the scenario above and further pretraining it on dataset D2.

Retraining on the combined dataset:

Initializing a model with random weights, as in the first scenario, but training it on the combination (union) of datasets D1 and D2.

Tips for LLM Pretraining and Evaluating Reward Models, SEBASTIAN RASCHKA, PHD

https://magazine.sebastianraschka.com/p/tips-for-llm-pretraining-and-evaluating-rms

Catastrophic forgetting

When training with new data

Forgets previously learned information

Techniques to avoid catastrophic forgetting

Include some old data in the new dataset

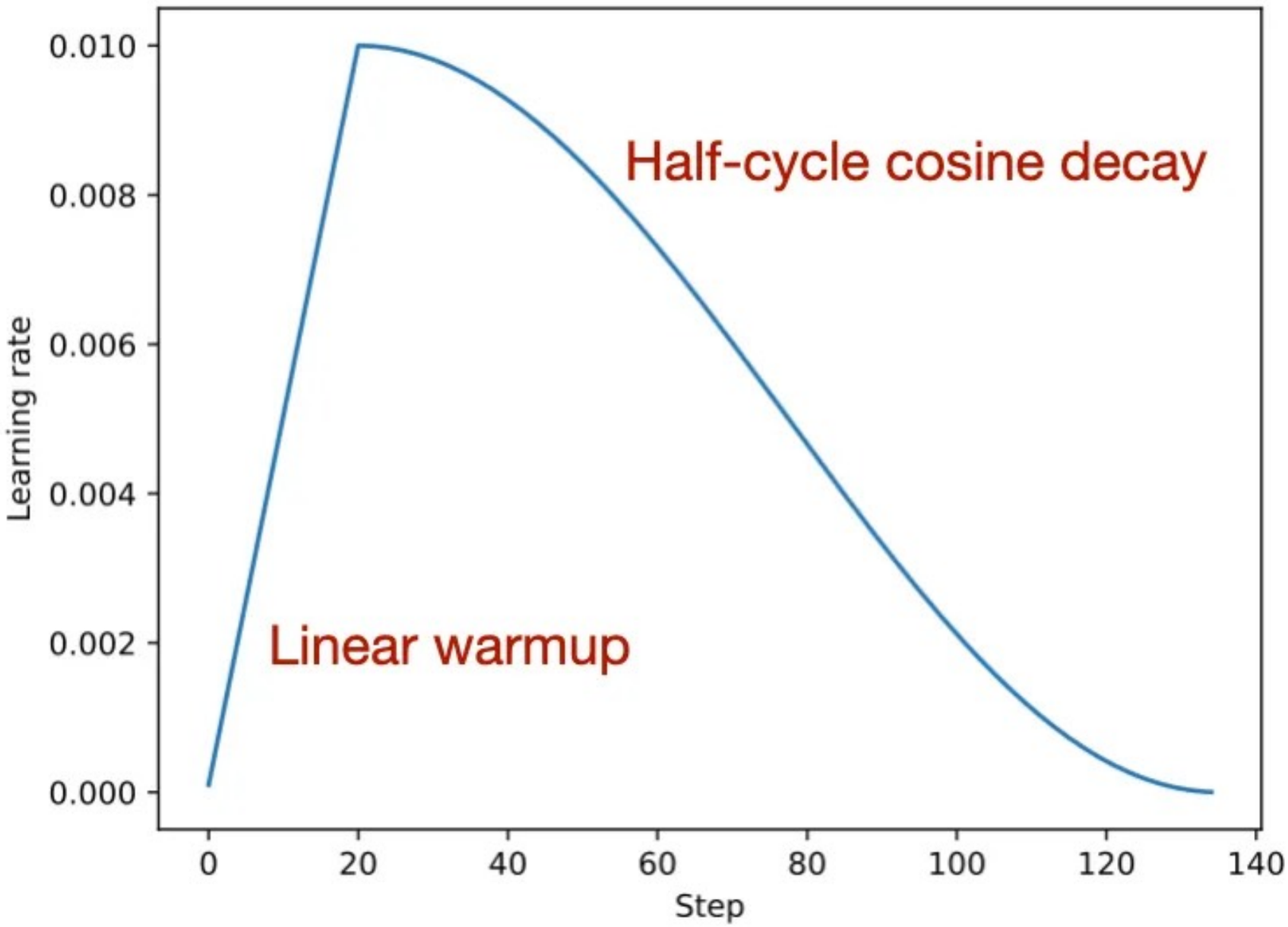
5%

DeepSeek used 30%

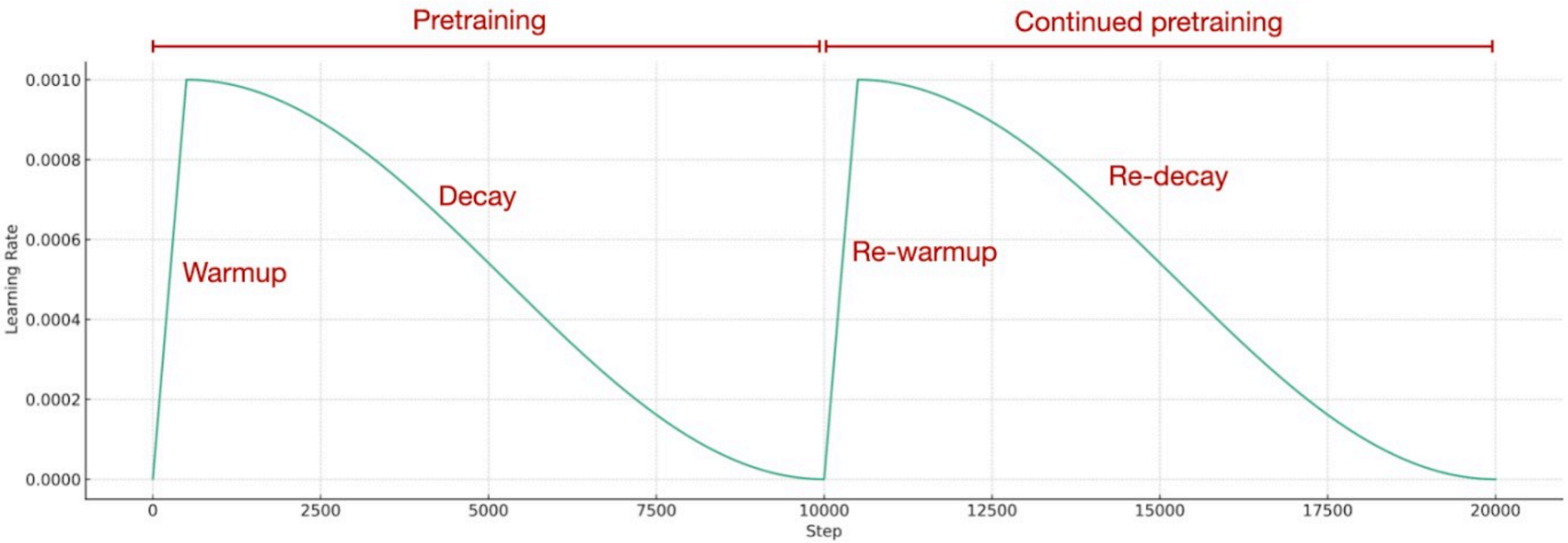
Learning Rate schedule

Add more tokens

Learning Rate Schedules



Repeat the Warmup and Decay



Learning Rate warmup - Model

From Appendix D of the text

```
from previous_chapters import create_dataloader_v1

train_ratio = 0.90
split_idx = int(train_ratio * len(text_data))

torch.manual_seed(123)

train_loader = create_dataloader_v1(
    text_data[:split_idx],
    batch_size=2,
    max_length=GPT_CONFIG_124M["context_length"],
    stride=GPT_CONFIG_124M["context_length"],
    drop_last=True,
    shuffle=True,
    num_workers=0
)

val_loader = create_dataloader_v1(
    text_data[split_idx:],
    batch_size=2,
    max_length=GPT_CONFIG_124M["context_length"],
    stride=GPT_CONFIG_124M["context_length"],
    drop_last=False,
    shuffle=False,
    num_workers=0
)
```

Learning Rate warmup

n_epochs = 15
initial_lr = 0.0001
peak_lr = 0.01

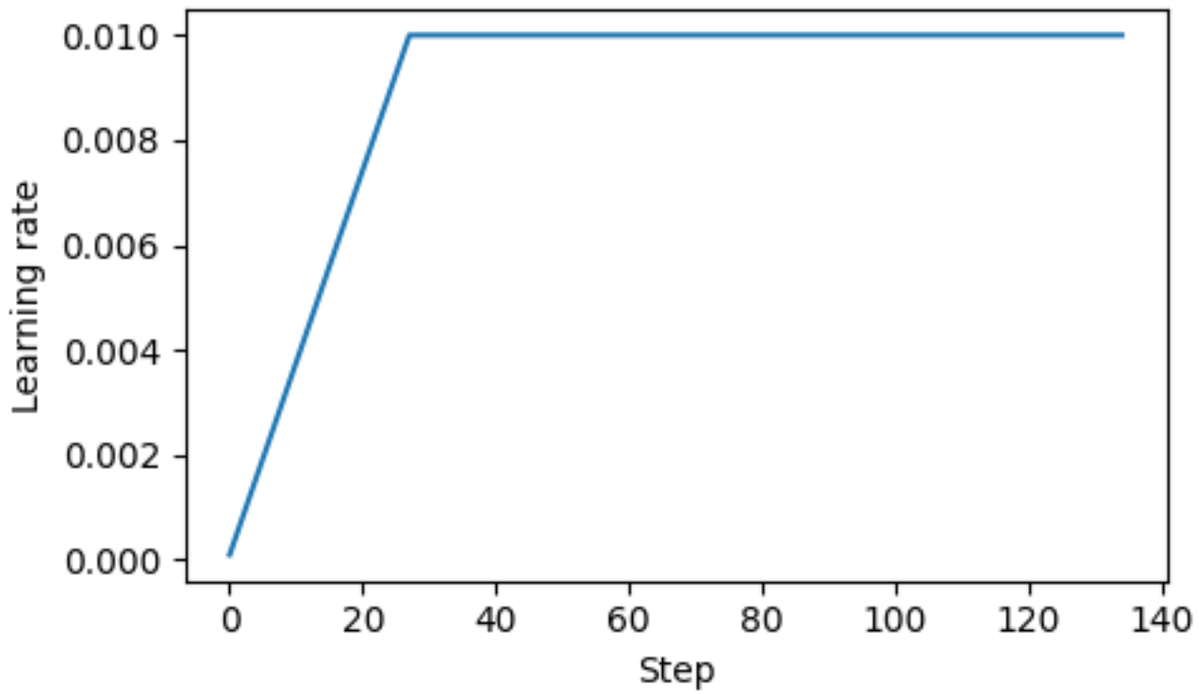
Typically, the number of warmup steps is between 0.1% to 20% of the total number of steps

total_steps = len(train_loader) * n_epochs
warmup_steps = int(0.2 * total_steps)

Warmup Code

```
lr_increment = (peak_lr - initial_lr) / warmup_steps

global_step = -1
track_lrs = []

optimizer = torch.optim.AdamW(model.parameters(), weigh

for epoch in range(n_epochs):
    for input_batch, target_batch in train_loader:
        optimizer.zero_grad()
        global_step += 1

        if global_step < warmup_steps:
            lr = initial_lr + global_step * lr_increment
        else:
            lr = peak_lr

        # Apply the calculated learning rate to the optimizer
        for param_group in optimizer.param_groups:
            param_group["lr"] = lr
        track_lrs.append(optimizer.param_groups[0]["lr"])

        # Calculate loss and update weights
```

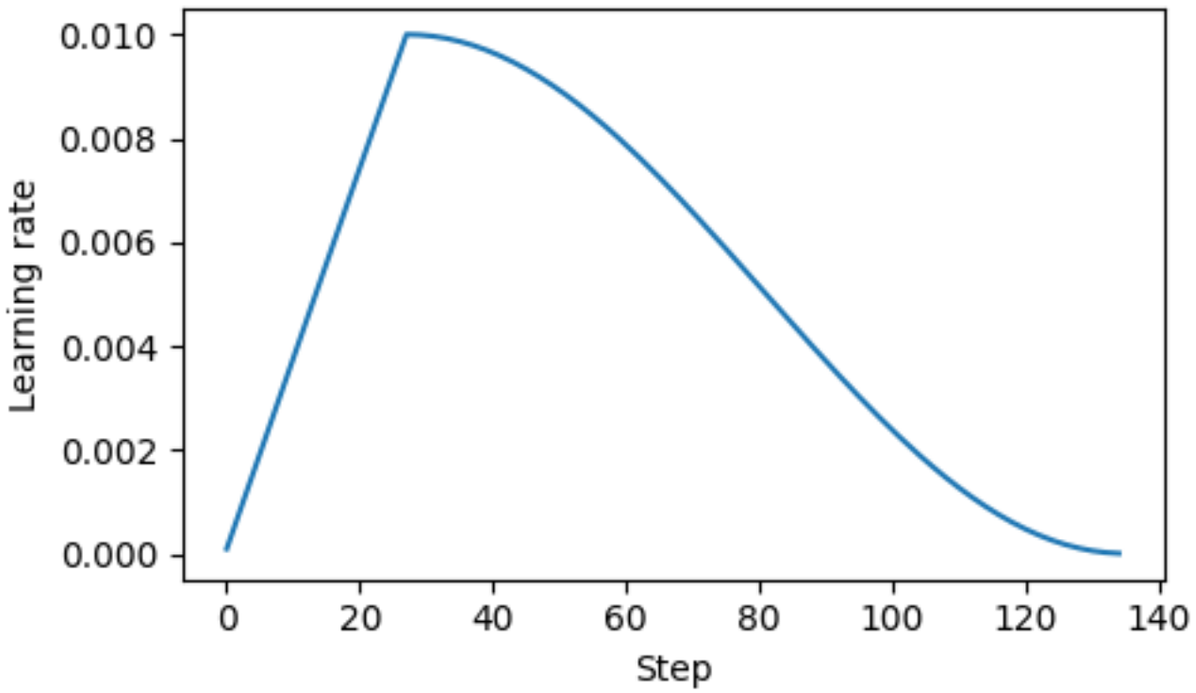
Learning Rate



Learning Rate Cosine decay

Learning rate follows a cosine curve,
initial value -> near zero following a half-cosine cycle

Reduces the risk of overshooting minima as the training progresses



```
import math

min_lr = 0.1 * initial_lr
track_lrs = []

lr_increment = (peak_lr - initial_lr) / warmup_steps
global_step = -1

for epoch in range(n_epochs):
    for input_batch, target_batch in train_loader:
        optimizer.zero_grad()
        global_step += 1

        # Adjust the learning rate based on the current phase (warmup or cosine annealing)
        if global_step < warmup_steps:
    # Linear warmup
    lr = initial_lr + global_step * lr_increment
else:
    # Cosine annealing after warmup
    progress = ((global_step - warmup_steps) /
                (total_training_steps - warmup_steps))
    lr = min_lr + (peak_lr - min_lr) * 0.5 * ( 1 + math.cos(math.pi * progress))

for param_group in optimizer.param_groups:
    param_group["lr"] = lr
track_lrs.append(optimizer.param_groups[0]["lr"])

# Calculate loss and update weights
```

Gradient clipping

Setting maximum value for Gradients

Ensures updates to the model's parameters are in manageable range

max_norm=1.0 in PyTorch's clip_grad_norm_ method

The norm of the gradients is clipped so the maximum norm does not exceed 1.0

See Appendix D for code

Reinforcement Learning - RL

Training an agent to interact with an environment to maximize a reward

Agent:

The LLM

Environment:

Can be the users interacting with the LLM

A simulated environment, or even

Another model evaluating the LLM's output

Reward:

A signal indicating how "good" the LLM's response is

The Challenge

The challenge of generating "good" text with LLMs

Defining "good" text in terms of:

- Helpfulness:
  - Providing relevant and informative answers
- Harmlessness
  - Avoiding toxic, biased, or unsafe content
- Alignment:
  - Reflecting human values and preferences

Limitations of traditional supervised learning in addressing these challenges

- Needs a lot of data

RLHF: Reinforcement Learning from Human Feedback

Prominent technique for aligning LLMs

Key steps

Pre-training a base LLM on a massive text corpus

Training a reward model based on human feedback on LLM outputs

Fine-tuning the LLM using RL, using the reward model

DPO: Direct Preference Optimization

Direct Preference Optimization: Your Language Model is Secretly a Reward Model

July 2024

Simplifying RLHF by directly optimizing the LLM based on human preferences

Eliminating the need for a separate reward model,
leading to more stable and efficient training.

A typical LLM development flow

In instruction finetuning, we train the LLM to generate correct answers given a prompt

Multiple ways to give a correct answer, and correct answers can differ in style

Input Prompt:

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

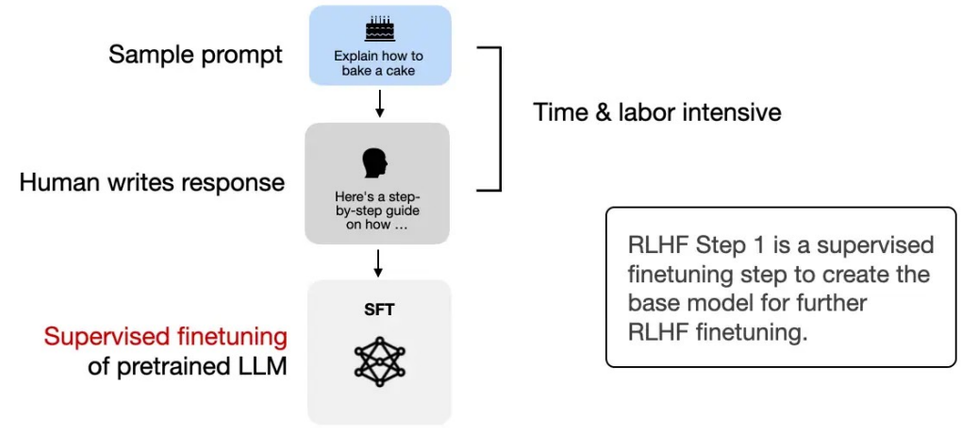
Answer 1: Technical Response

"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."

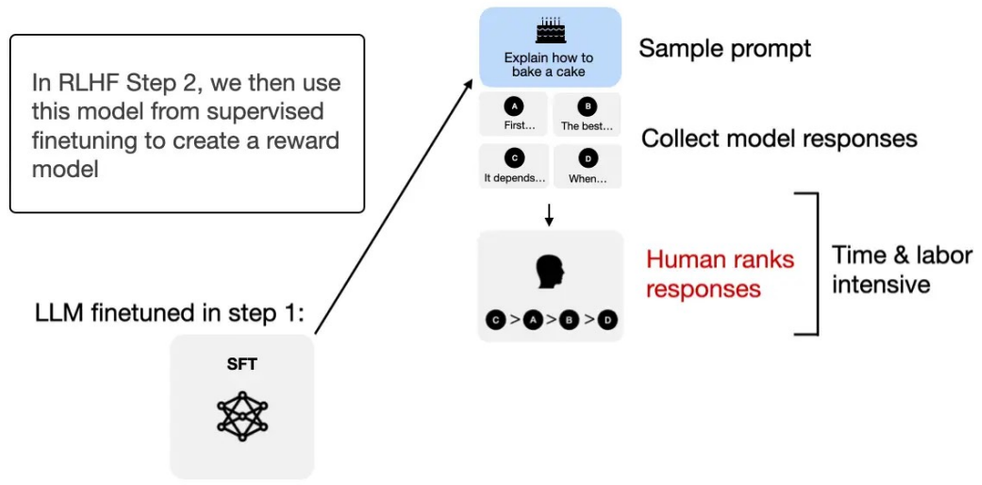
Answer 2: User-Friendly Response

"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."

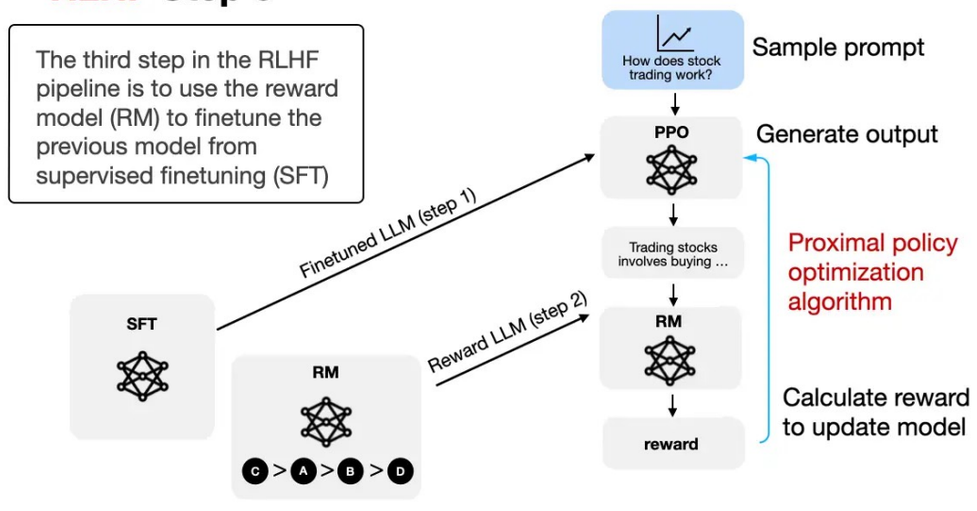
RLHF Step 1

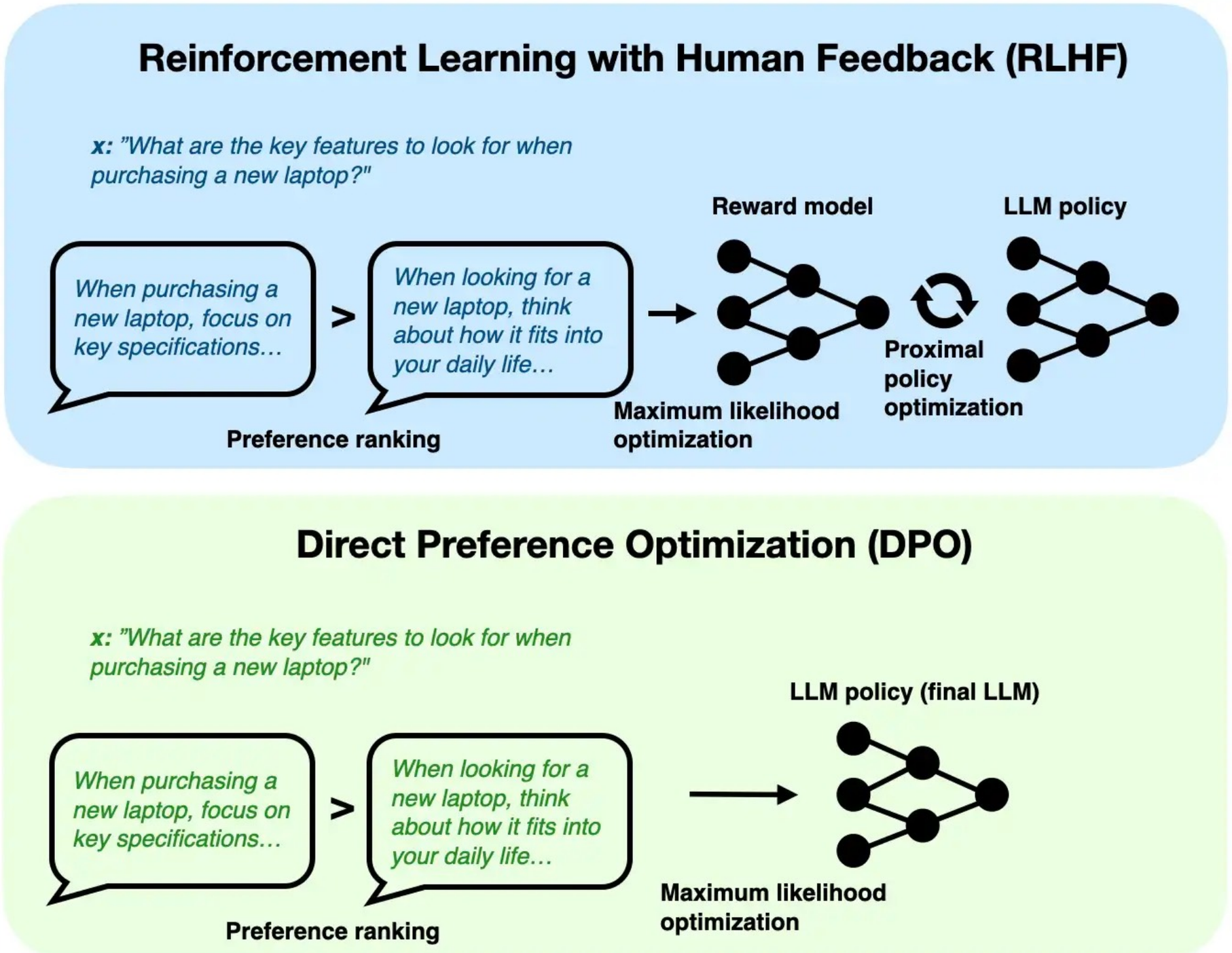


RLHF Step 2



RLHF Step 3




Reinforcement Learning with Human Feedback (RLHF)
x: "What are the key features to look for when purchasing a new laptop?"
When purchasing a new laptop, focus on key specifications…
>
When looking for a new laptop, think about how it fits into your daily life…
Preference ranking
Reward model
Maximum likelihood optimization
Proximal policy optimization
LLM policy
Direct Preference Optimization (DPO)
x: "What are the key features to look for when purchasing a new laptop?"
When purchasing a new laptop, focus on key specifications…
>
When looking for a new laptop, think about how it fits into your daily life…
Preference ranking
LLM policy (final LLM)
Maximum likelihood optimization

Performance

The original DPO found that GPT-4 and humans prefers the answers generated by DPO most of the time over regular supervised finetuned models (SFT) or models finetuned with RLHF using PPO

	DPO	SFT	PPO-1
N respondents	272	122	199
GPT-4 (S) win %	47	27	13
GPT-4 (C) win %	54	32	12
Human win %	58	43	17

Direct Preference Optimization:
Your Language Model is Secretly a Reward Model, Jul 2024, arXiv:2305.18290v3

Tips for LLM Pretraining and Evaluating Reward Models
https://magazine.sebastianraschka.com/p/tips-for-llm-pretraining-and-evaluating-rms

RewardBench Results

The score can be interpreted as an accuracy: how many times did the model select the correct response out of the total number of tasks

The top ranked model is a dedicated reward model

Most entries are DPO models

The "dedicated" or standalone reward model used in RLHF

Reward Model	Avg	Chat	Chat Hard	Safety	Reason	Prior Sets
berkeley-nest/Starling-RM-34B	**81.5**	96.9	59.0	**89.9**	90.3	71.4
allenai/tulu-2-dpo-70b	77.0	97.5	60.8	85.1	88.9	52.8
mistralai/Mixtral-8x7B-Instruct-v0.1	75.8	95.0	65.2	76.5	**92.1**	50.3
berkeley-nest/Starling-RM-7B-alpha	74.7	**98.0**	43.5	88.6	74.6	68.6
NousResearch/Nous-Hermes-2-Mixtral-8x7B-DPO	73.9	91.6	62.3	81.7	81.2	52.7
HuggingFaceH4/zephyr-7b-alpha	73.6	91.6	63.2	70.0	89.6	53.5
NousResearch/Nous-Hermes-2-Mistral-7B-DPO	73.5	92.2	59.5	83.8	76.7	55.5
allenai/tulu-2-dpo-13b	72.9	95.8	56.6	78.4	84.2	49.5
openbmb/UltraRM-13b	71.3	96.1	55.2	45.8	81.9	**77.2**
HuggingFaceH4/zephyr-7b-beta	70.7	95.3	62.6	54.1	89.6	52.2
allenai/tulu-2-dpo-7b	70.4	97.5	54.6	74.3	78.1	47.7
stabilityai/stablelm-zephyr-3b	70.1	86.3	58.2	74.0	81.3	50.7
HuggingFaceH4/zephyr-7b-gemma-v0.1	66.6	95.8	51.5	55.1	79.0	51.7
Qwen/Qwen1.5-72B-Chat	66.2	62.3	67.3	71.8	87.4	42.3
allenai/OLMo-7B-Instruct	66.1	89.7	48.9	64.1	76.3	51.7
IDEA-CCNL/Ziya-LLaMA-7B-Reward	66.0	88.0	41.3	62.5	73.7	64.6
stabilityai/stablelm-2-zephyr-1_6b	65.9	96.6	46.6	60.0	77.4	48.7
Qwen/Qwen1.5-14B-Chat	65.8	57.3	67.4	77.2	85.9	41.2
Qwen/Qwen1.5-7B-Chat	65.6	53.6	**69.8**	75.3	86.4	42.9
OpenAssistant/oasst-rm-2.1-pythia-1.4b-epoch-2.5	65.1	88.5	47.8	62.1	61.4	65.8
Random	50.0	50.0	50.0	50.0	50.0	50.0

Sequence Classifier ( )

Direct Preference Optimization ( ), and a random model ( ).

RewardBench: Evaluating Reward Models for Language Modeling
arXiv:2403.13787v2

Two Views on DPO

Direct Preference Optimization:

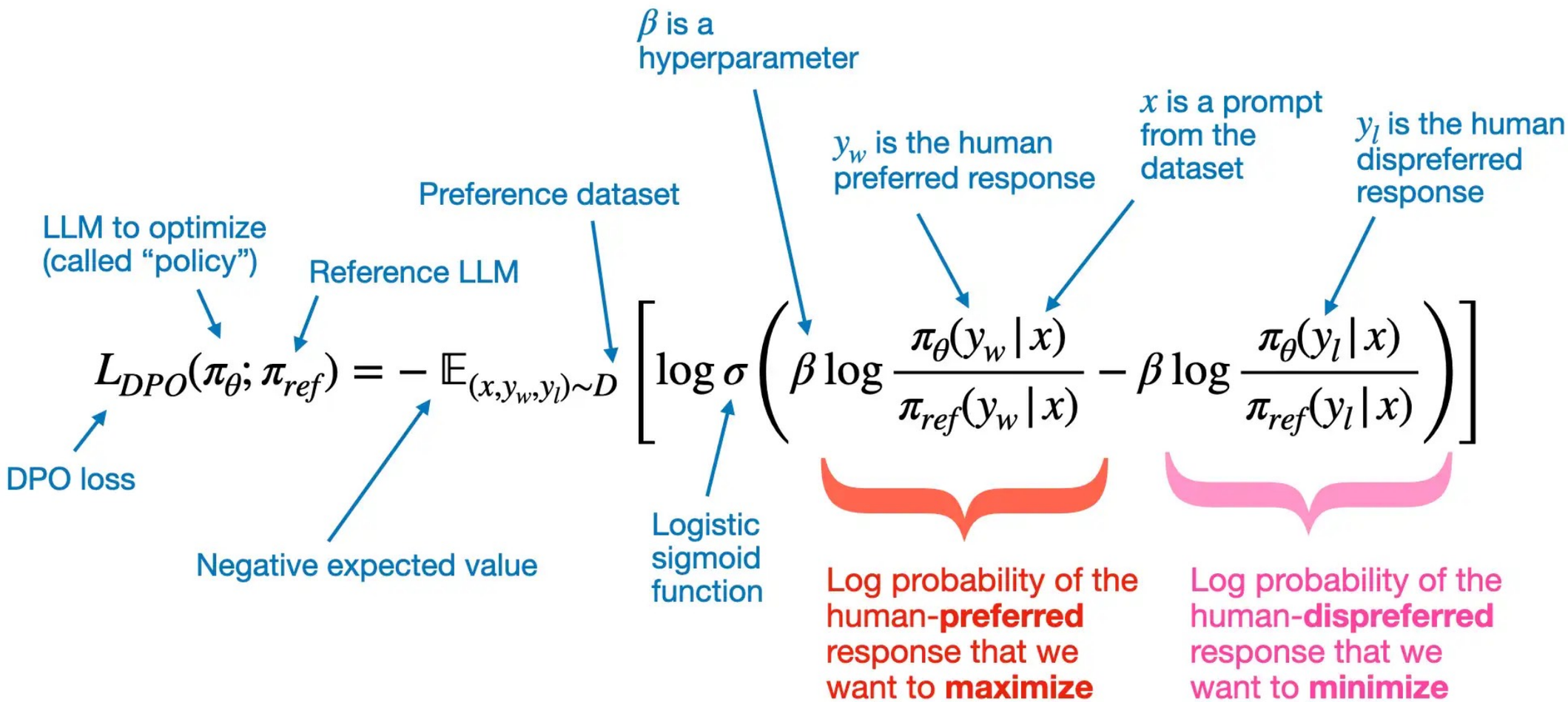
Your Language Model is Secretly a Reward Model,

> With virtually no tuning of hyperparameters, DPO performs similarly or better than existing RLHF algorithms, including those based on PPO; DPO thus meaningfully reduces the barrier to training more language models from human preferences

Tips for LLM Pretraining and Evaluating Reward Models

https://magazine.sebastianraschka.com/p/tips-for-llm-pretraining-and-evaluating-rms

> Also, many DPO models can be found at the top of most LLM leaderboards. However, because DPO is much simpler to use than RLHF with a dedicated reward model, there are many more DPO models out there. So, it is hard to say whether DPO is actually better in a head-to-head comparison as there are no equivalent models of these models (that is, models with exactly the same architecture trained on exactly the same dataset but using DPO instead of RLHF with a dedicated reward model).


β is a hyperparameter
x is a prompt from the dataset
y_w is the human preferred response
y_l is the human dispreferred response
Preference dataset
LLM to optimize (called "policy")
Reference LLM
$L_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right]$
DPO loss
Negative expected value
Logistic sigmoid function
Log probability of the human-**preferred** response that we want to **maximize**
Log probability of the human-**dispreferred** response that we want to **minimize**

TRL - Transformer Reinforcement Learning

HuggingFace library to train transformer language models with Reinforcement Learning

SFTTrainer: Supervise Fine-tune

RewardTrainer

PPOTrainer

DPOTrainer