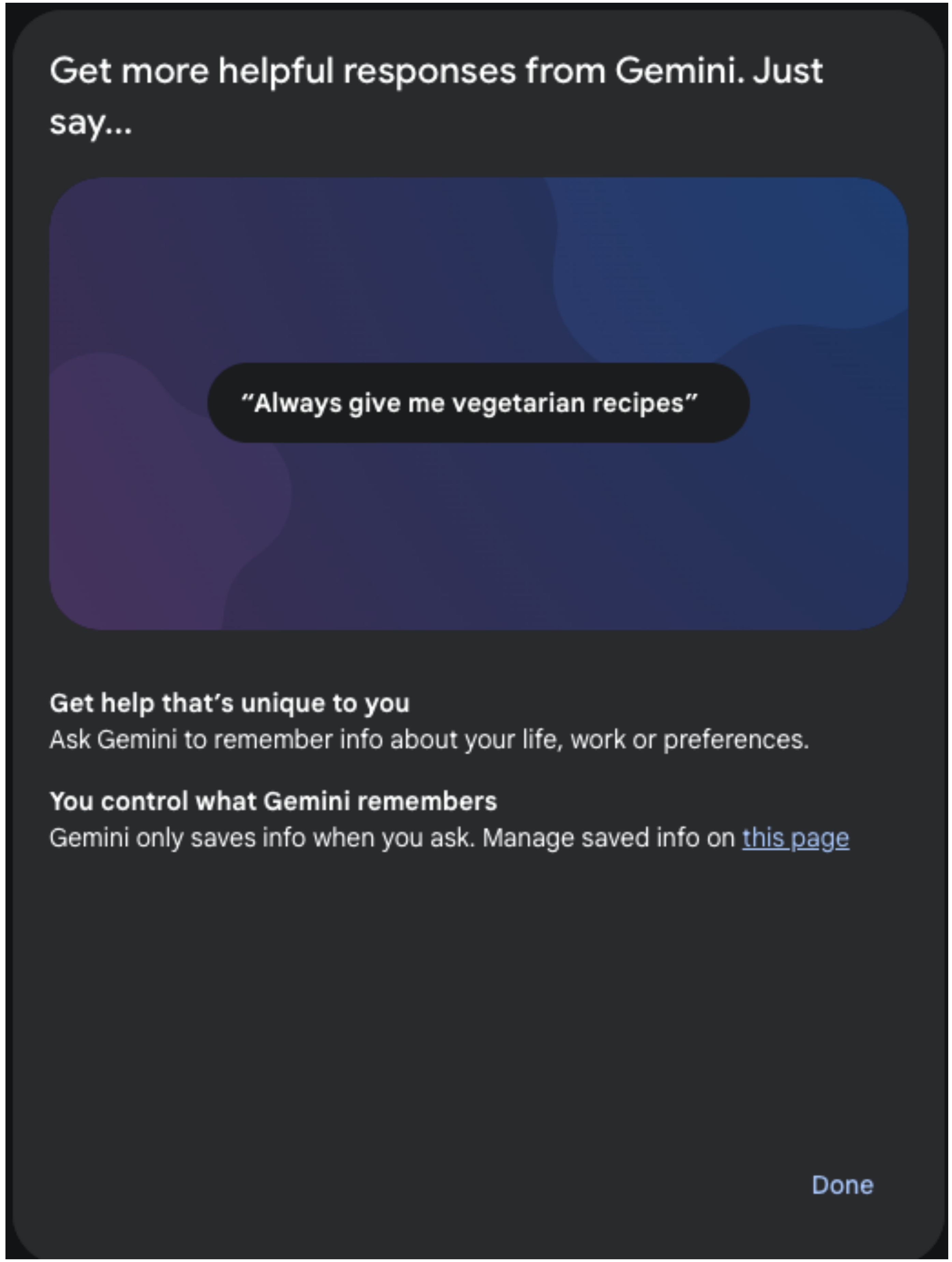
CS 696 Applied Large Language Models
Spring Semester, 2025
Doc 15 Hooks, Continually Pre-train, DPO
Mar 4, 2025


Get more helpful responses from Gemini. Just say...
"Always give me vegetarian recipes"
Get help that's unique to you
Ask Gemini to remember info about your life, work or preferences.
You control what Gemini remembers
Gemini only saves info when you ask. Manage saved info on this page
Done

SDSU Free ChatGPT Edu (GPT-4o)

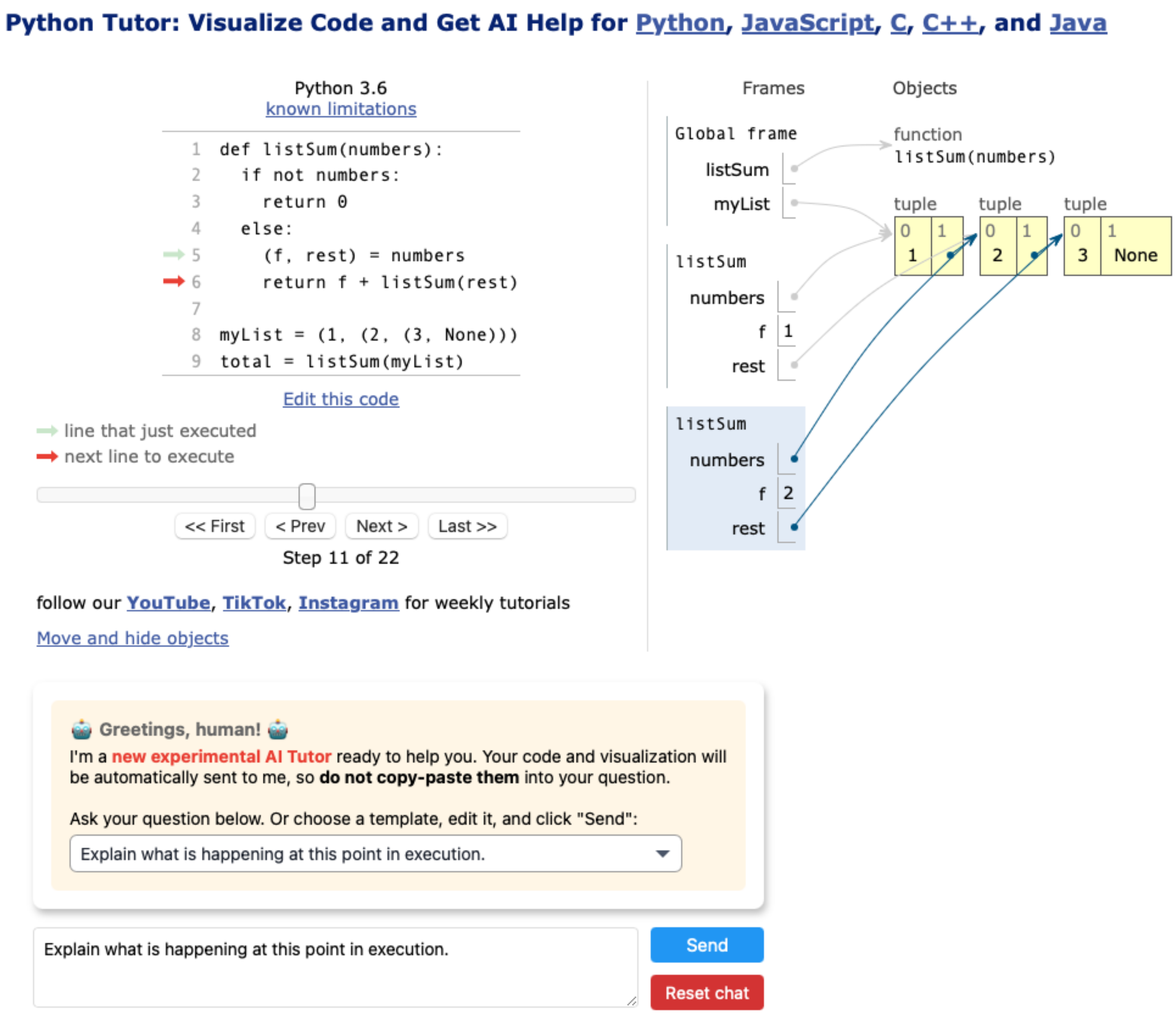
Coming to SDSU students Wednesday, March 12th

SDSU’s ChatGPT Edu workspace allows you to create and share GPTs with other users

UCSD Python Tutor

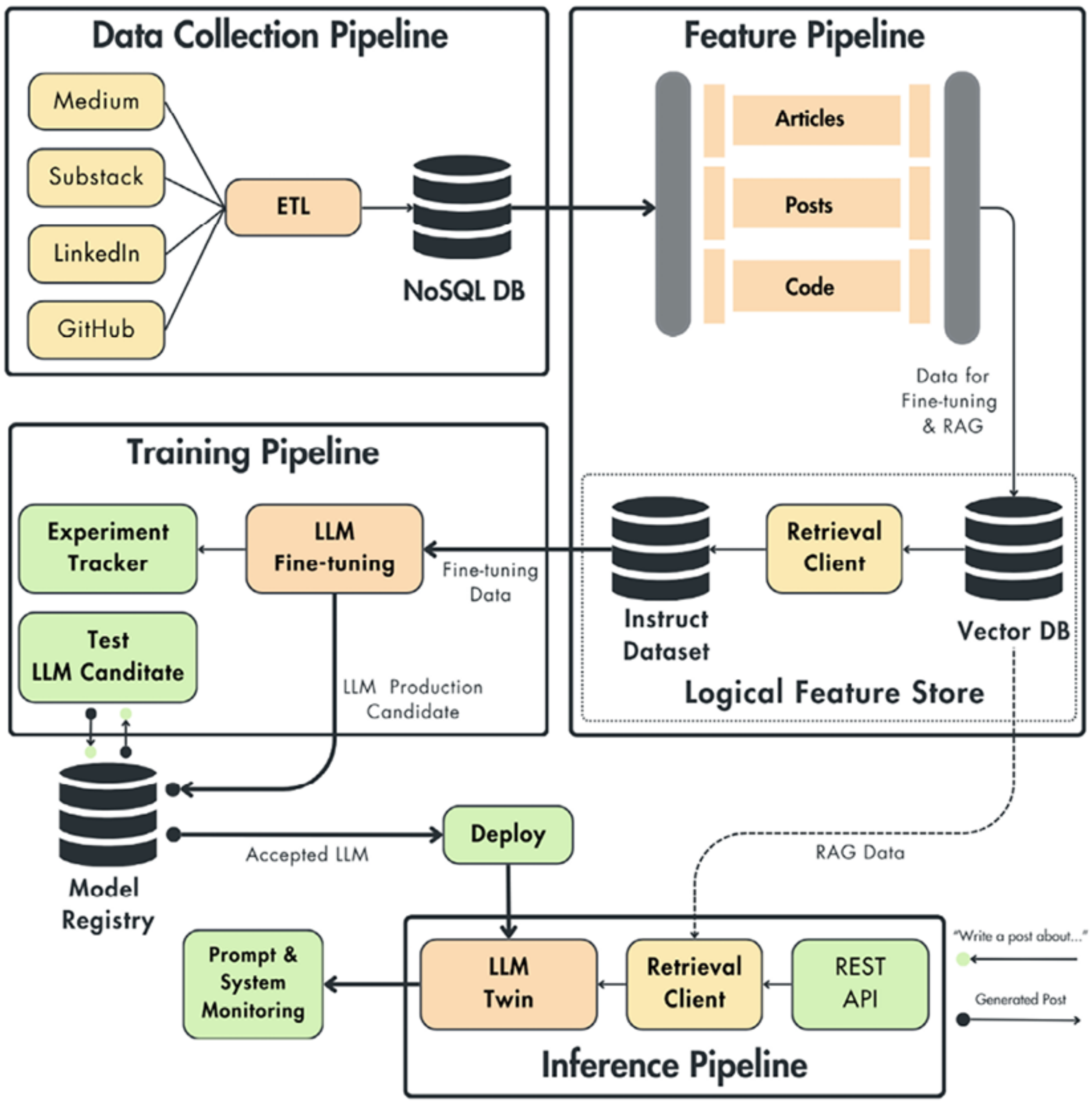
https://pythontutor.com/render.html#mode=display

https://www.oreilly.com/radar/using-generative-ai-to-build-generative-ai/



Book & Papers

LLM Engineer's Handbook, Lusztin, Labonne



Book & Papers

Hands-On Large Language Models

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

```python
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)
```

```python
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

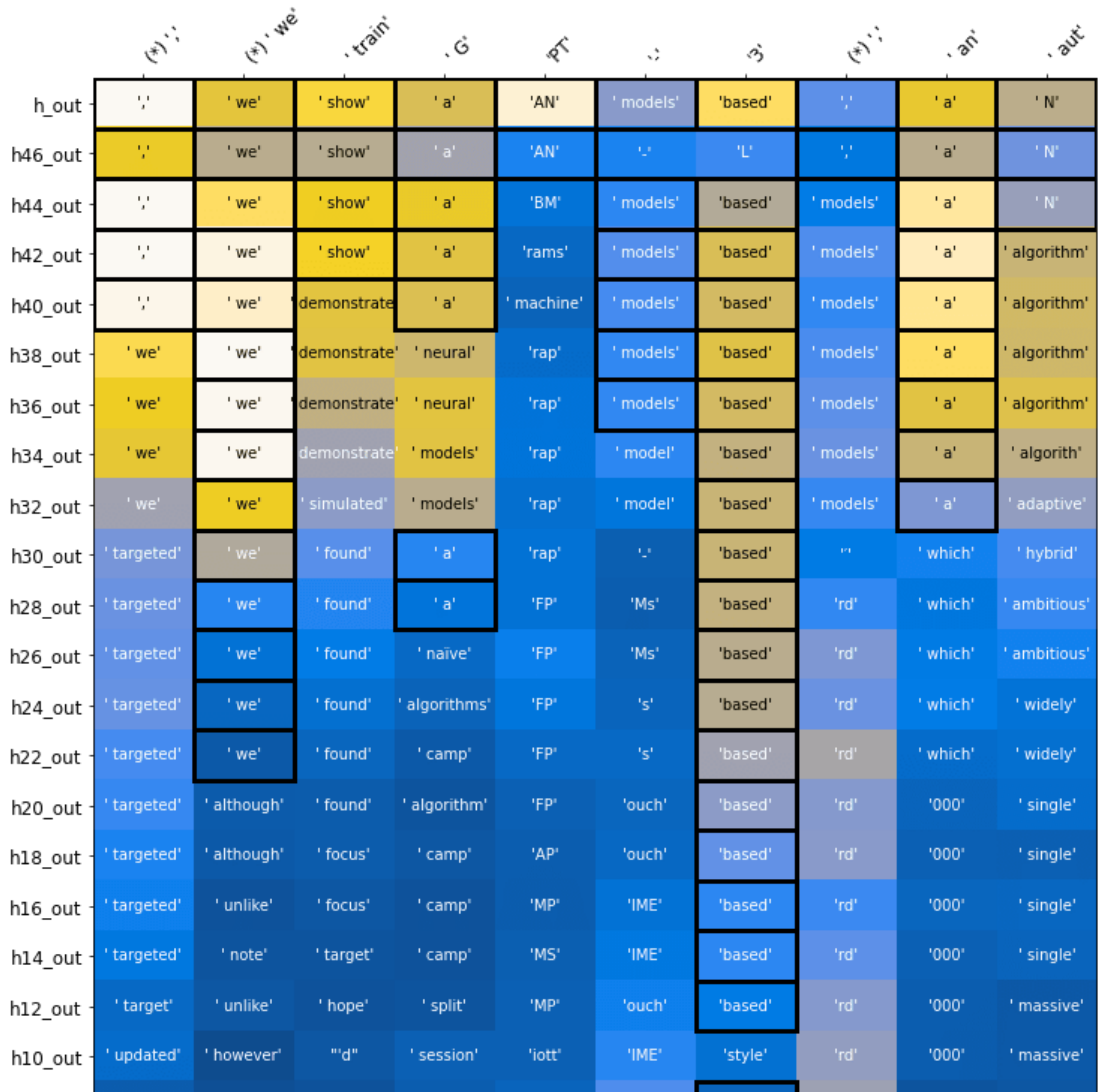
Looks at the rank of the final token in each layer

Input

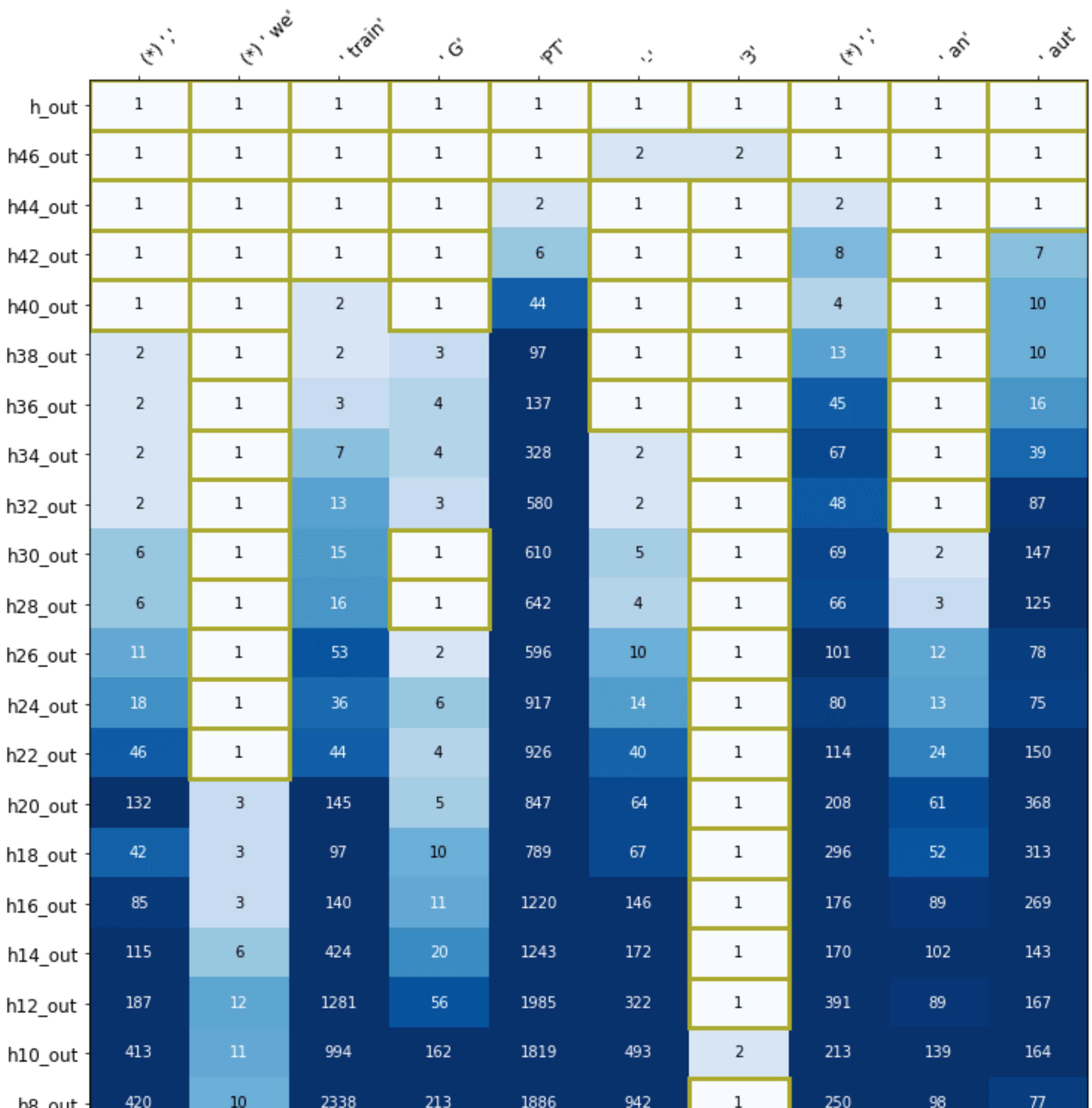
Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters

Tokens

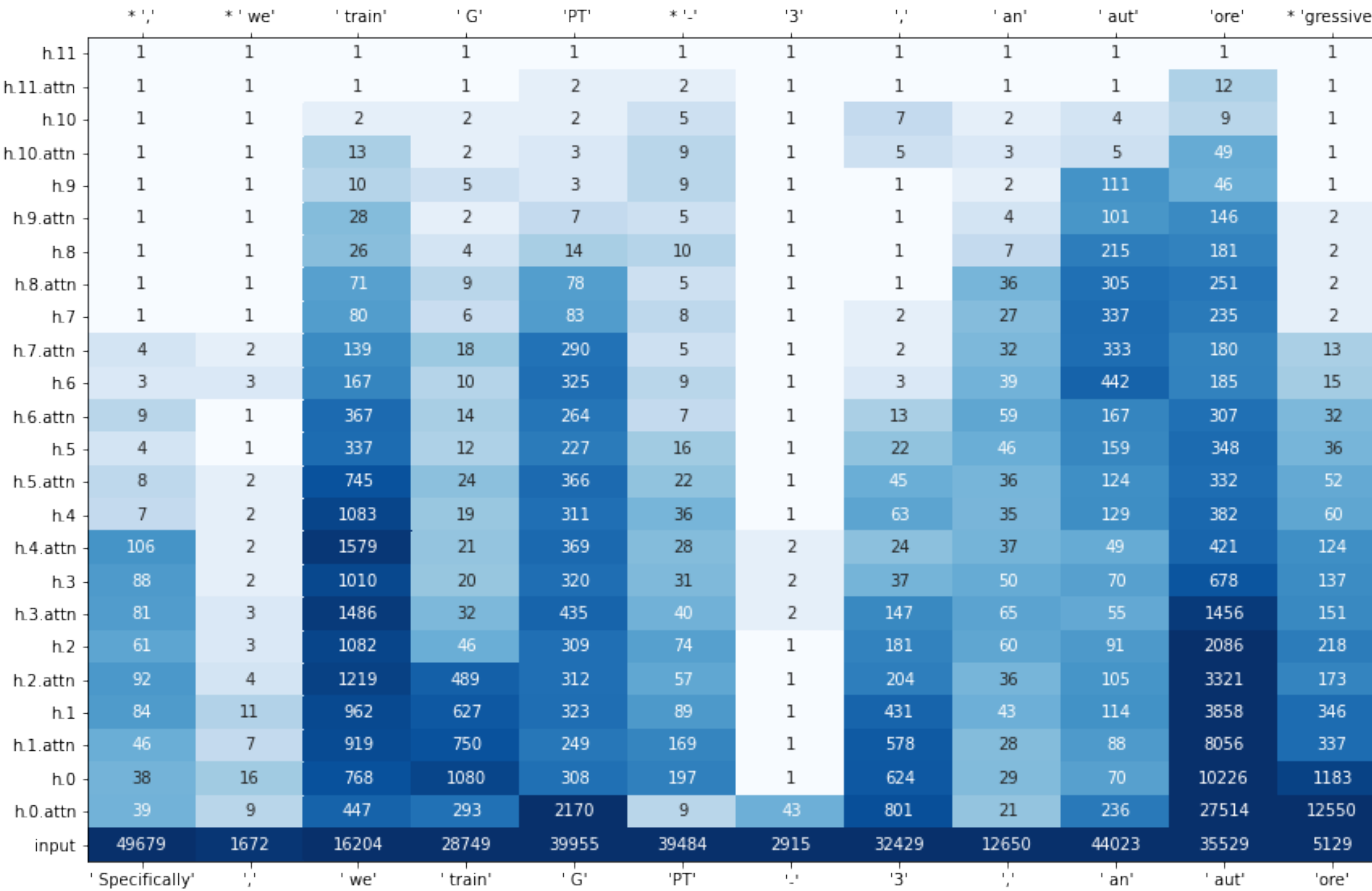
"Specifically" "," "we" "train" "G" "PT" "-" "3" "," "an" "aut" "ore" "gressive" “language" "model" "with" "175" "billion" "parameters



(*) ',' (*) ' 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'
h10_out ' updated' ' however' "'d" ' session' 'iott' 'IME' 'style' 'rd' '000' ' massive'

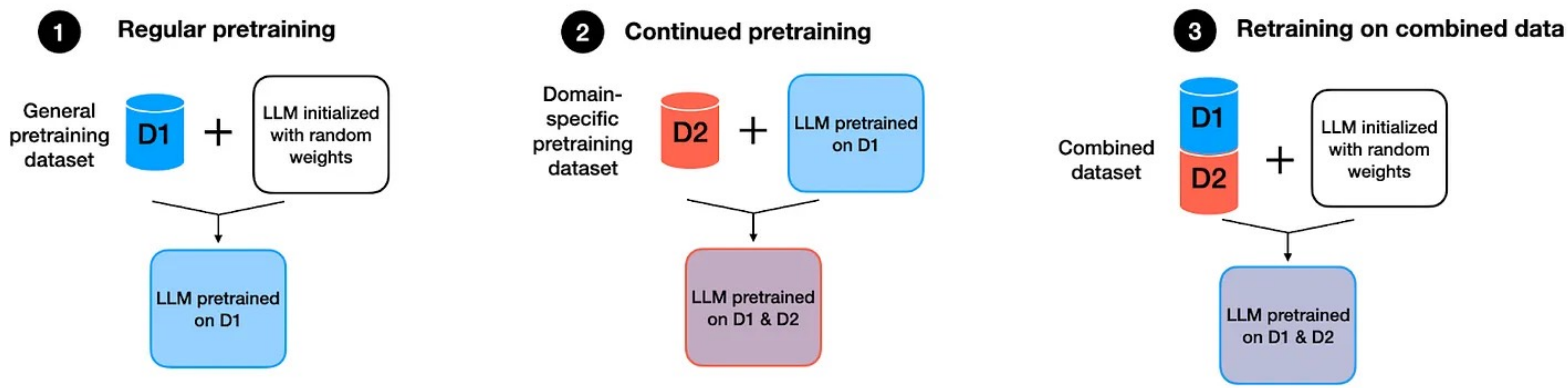


(*) ';'
(*) ' we'
' train'
' G'
'PT'
':'
'3'
(*) ';'
' an'
' aut'
h_out 1 1 1 1 1 1 1 1 1 1
h46_out 1 1 1 1 1 2 2 1 1 1
h44_out 1 1 1 1 2 1 1 2 1 1
h42_out 1 1 1 1 6 1 1 8 1 7
h40_out 1 1 2 1 44 1 1 4 1 10
h38_out 2 1 2 3 97 1 1 13 1 10
h36_out 2 1 3 4 137 1 1 45 1 16
h34_out 2 1 7 4 328 2 1 67 1 39
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
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

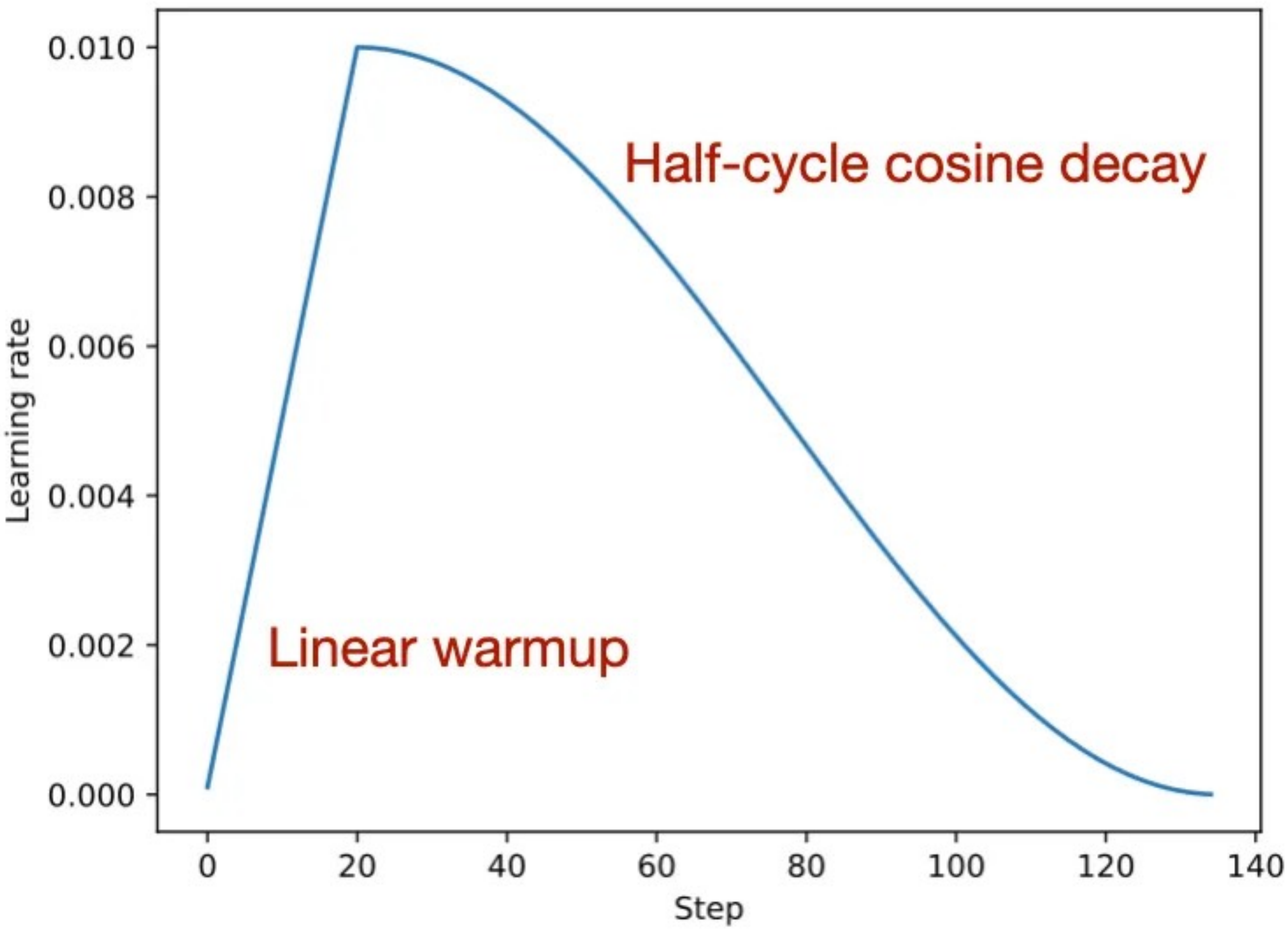
Techniques to avoid catastrophic forgetting

- Include some old data in the new dataset
  - 5%
  - DeepSeek used 30%

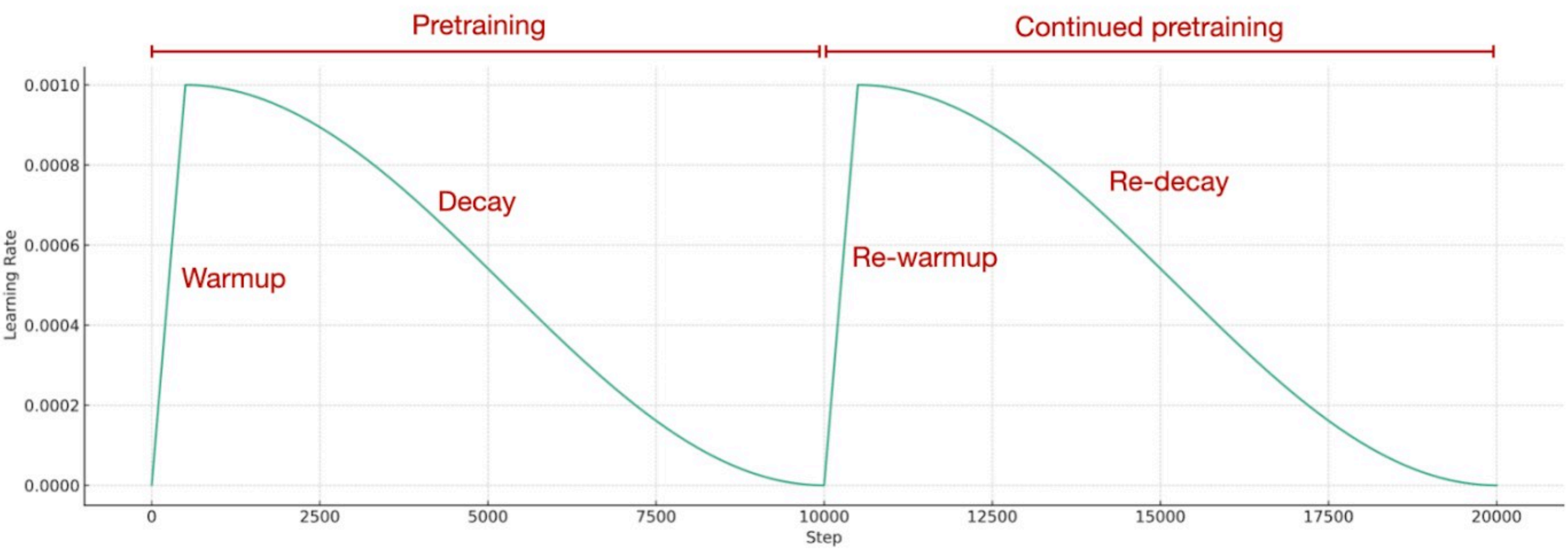
- Learning Rate schedule

- 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
```

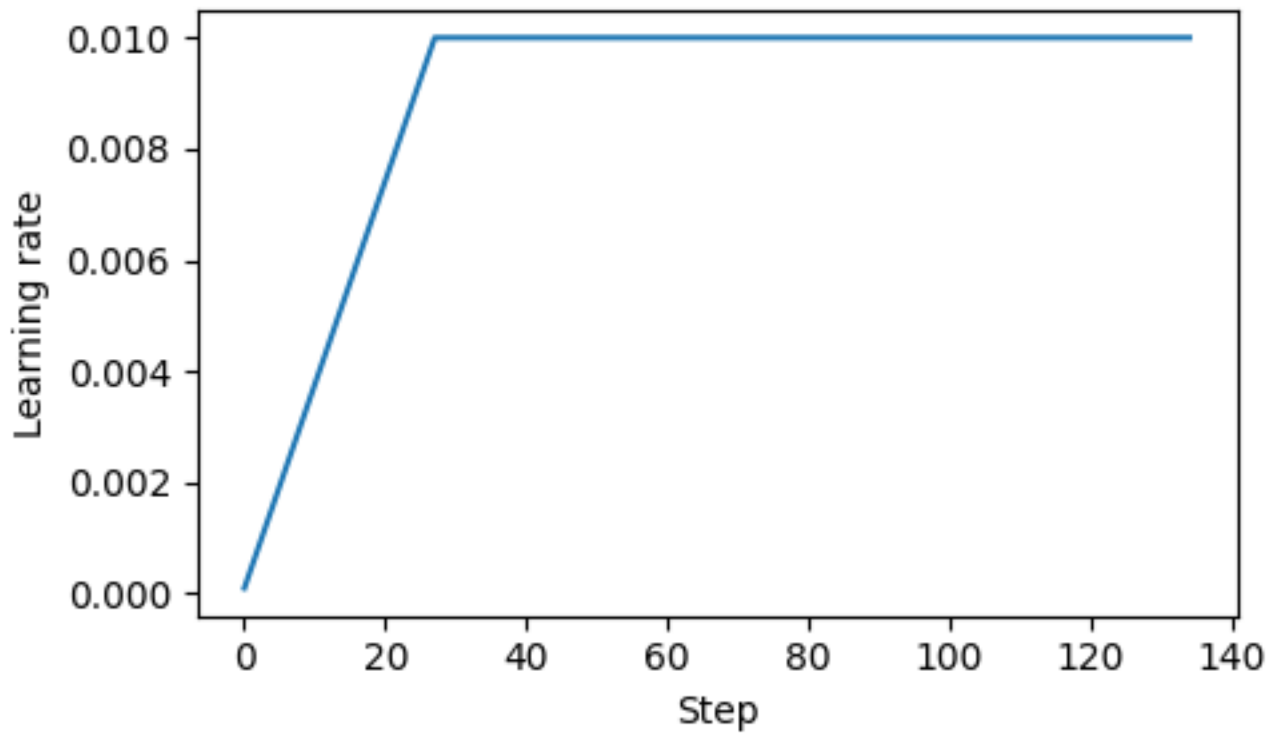
Typically, the number of warmup steps is between 0.1% to 20% of the total number of steps

```
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

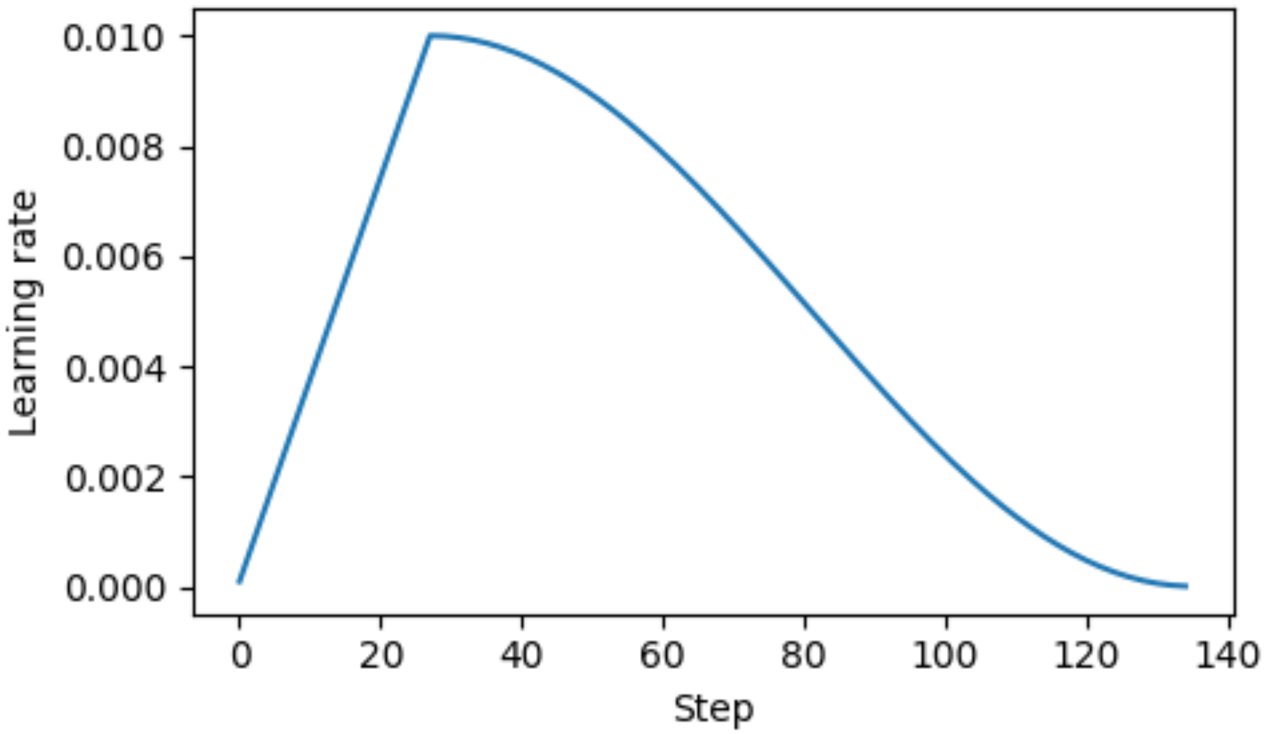


Learning Rate Cosine decay

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

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

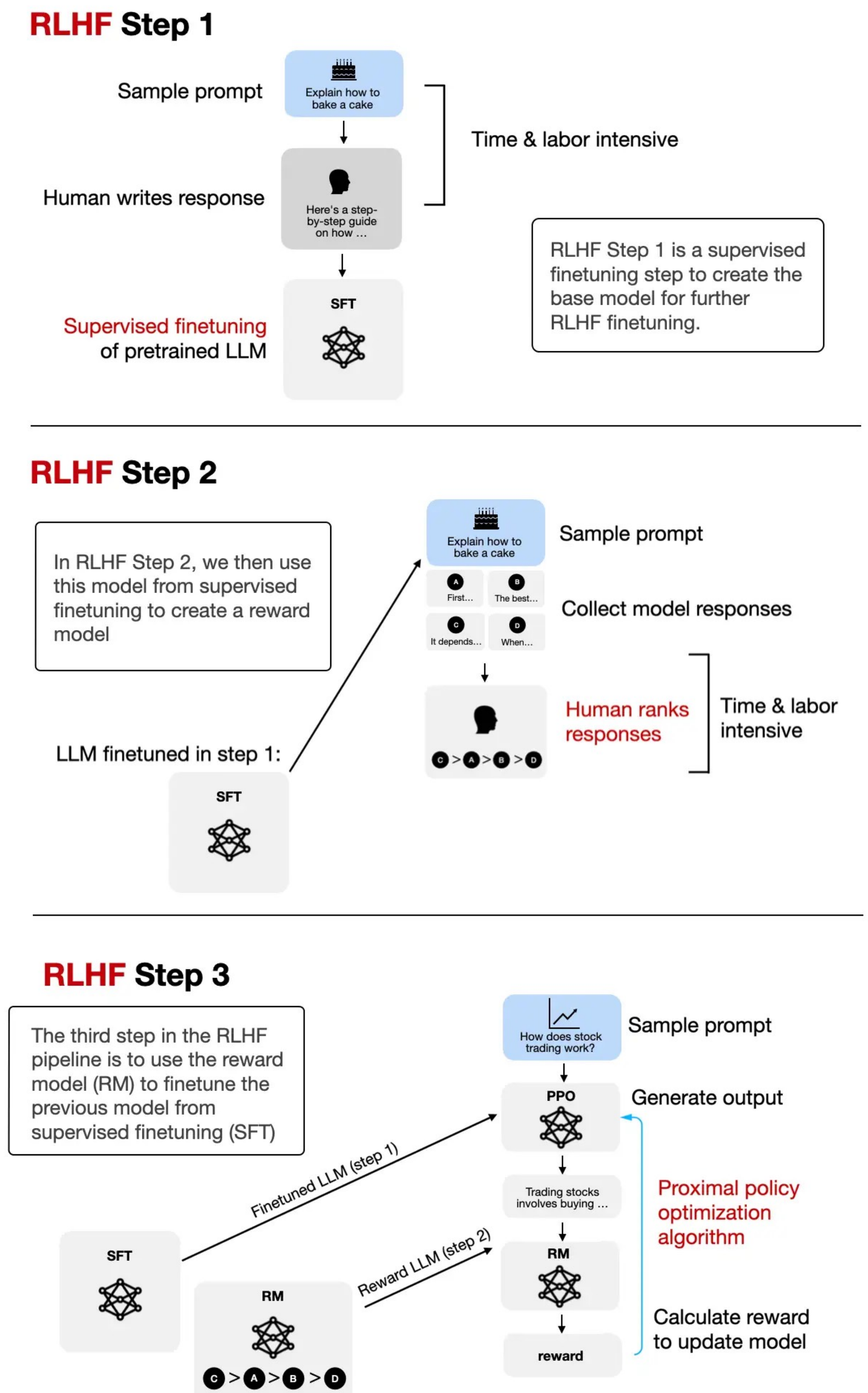
In instruction finetuning, we train the LLM to generate correct answers given a prompt

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

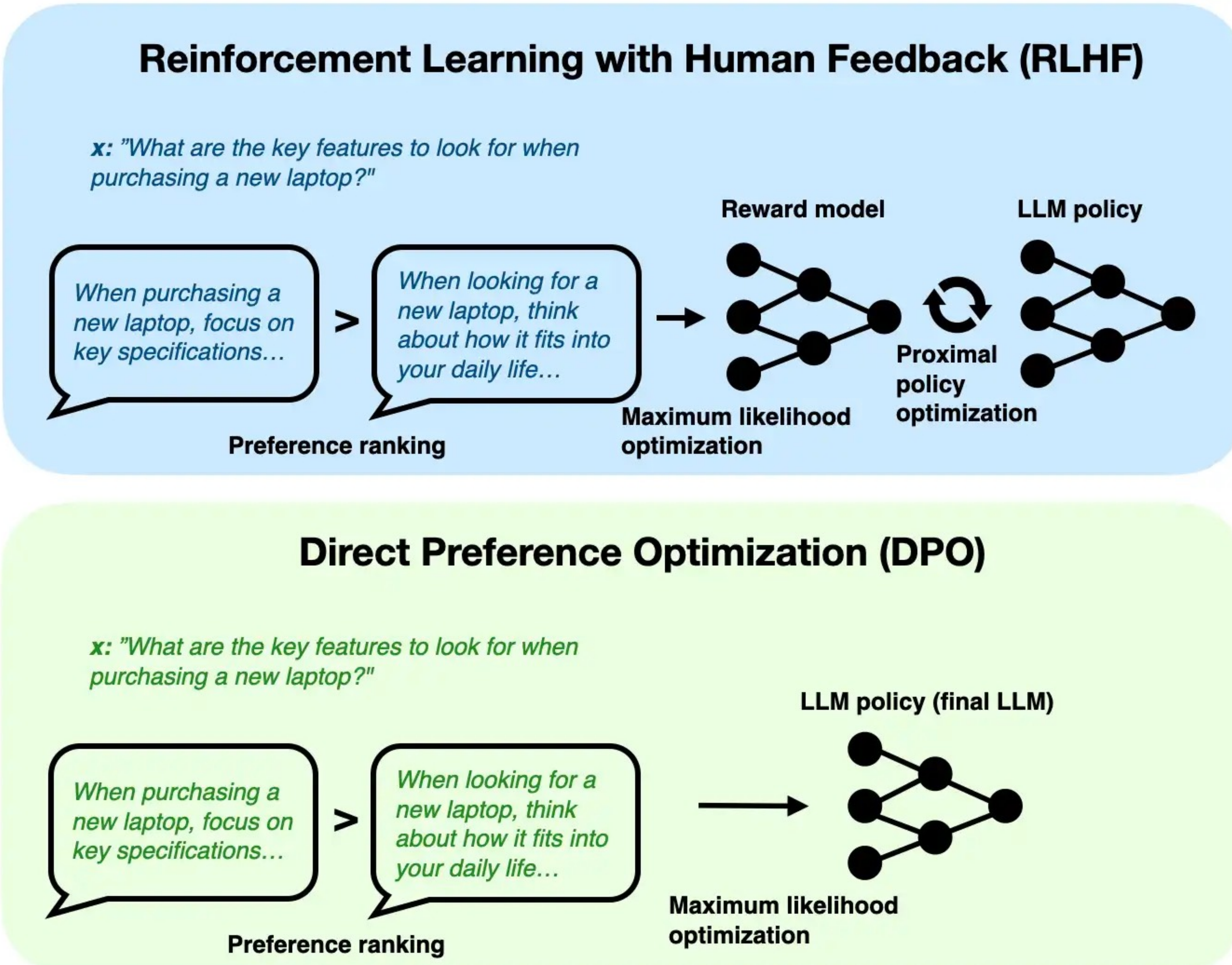
Input Prompt:
"What are the key features to look for when purchasing a new laptop?"

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
Sample prompt
Explain how to bake a cake
Human writes response
Here's a step-by-step guide on how ...
Supervised finetuning of pretrained LLM
SFT
Time & labor intensive
RLHF Step 1 is a supervised finetuning step to create the base model for further RLHF finetuning.
RLHF Step 2
In RLHF Step 2, we then use this model from supervised finetuning to create a reward model
Explain how to bake a cake
Sample prompt
First...
The best...
It depends...
When...
Collect model responses
Human ranks responses
Time & labor intensive
C > A > B > D
LLM finetuned in step 1:
SFT
RLHF Step 3
The third step in the RLHF pipeline is to use the reward model (RM) to finetune the previous model from supervised finetuning (SFT)
How does stock trading work?
Sample prompt
PPO
Generate output
Trading stocks involves buying ...
Proximal policy optimization algorithm
RM
reward
Calculate reward to update model
Finetuned LLM (step 1)
Reward LLM (step 2)
SFT
RM
C > A > B > D


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
Maximum likelihood optimization
LLM policy (final LLM)

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

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

The "dedicated" or standalone reward model used in RLHF

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

y_w is the human preferred response

x is a prompt from the dataset

y_l is the human dispreferred response

LLM to optimize (called "policy")

Reference LLM

Preference dataset

$$L_{DPO}(\pi_\theta; \pi_{ref}) = -\,\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{ref}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{ref}(y_l \mid x)}\right)\right]$$

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

The High-level Training

```
# train_dpo.py
from datasets import load_dataset
from trl import DPOConfig, DPOTrainer
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen2-0.5B-Instruct")
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2-0.5B-Instruct")
train_dataset = load_dataset("trl-lib/ultrafeedback_binarized", split="train")

training_args = DPOConfig(output_dir="Qwen2-0.5B-DPO", logging_steps=10)
trainer = DPOTrainer(model=model, args=training_args, processing_class=tokenizer,
train_dataset=train_dataset)
trainer.train()
```

Preference Datasets

Structure for DPO preference dataset

Instruction

Tell me a joke about octopuses.

Chosen answer

Why don't octopuses play cards in casinos? Because they can't count past eight.

Rejected answer

How many tickles does it take to make an octopus laugh? Ten tickles.

Rejected response

Behavior we aim to eliminate from the model

Without the dataset would be a simple instruction set

{'instruction': 'Rewrite the following sentence so that it is in active voice.',
'input': 'The cake was baked by Sarah.',
'output': 'Sarah baked the cake.',
'rejected': 'Sarah just went ahead and baked the cake.',
'chosen': 'Sarah baked the cake.'},

{'instruction': 'Rewrite the following sentence to remove redundancy.',
'input': 'The actual fact is that he was late.',
'output': 'The fact is that he was late.',
'rejected': 'Face it, he was late.',
'chosen': 'The fact is that he was late.'},

{'instruction': 'Edit the following sentence for grammar.',
'input': 'He go to the park every day.',
'output': 'He goes to the park every day.',
'rejected': 'He goes to the stupid park every single day.',
'chosen': 'He goes to the park every day.'},

{'instruction': 'What are the first 10 square numbers?',
'input': '',
'output': '1, 4, 9, 16, 25, 36, 49, 64, 81, 100.',
'rejected': 'Here are your precious square numbers: 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.',
'chosen': '1, 4, 9, 16, 25, 36, 49, 64, 81, 100.'}

Huggingfance dataset

argilla/distilabel-intel-orca-dpo-pairs

System: You are an AI assistant. You will be given a task. You must generate a detailed and long answer.

Input: Generate an approximately fifteen-word sentence that describes all this data: Midsummer House eatType restaurant; Midsummer House food Chinese; Midsummer House priceRange moderate; Midsummer House customer rating 3 out of 5; Midsummer House near All Bar One

Chosen: Midsummer House is a moderately priced Chinese restaurant with a 3/5 customer rating, located near All Bar One.

Rejected: Sure! Here's a sentence that describes all the data you provided:

"Midsummer House is a moderately priced Chinese restaurant with a customer rating of 3 out of 5, located near All Bar One, offering a variety of delicious dishes."

System: You are a helpful assistant, who always provide explanation. Think like you are answering to a five year old.

Input: What happens next in this paragraph?

She then rubs a needle on a cotton ball then pushing it onto a pencil and wrapping thread around it. She then holds up a box of a product and then pouring several liquids into a bowl. she
Choose your answer from: A. adds saucepan and shakes up the product in a grinder. B. pinches the thread to style a cigarette, and then walks away. C. then dips the needle in ink and using the pencil to draw a design on her leg, rubbing it off with a rag in the end. D. begins to style her hair and cuts it several times before parting the ends of it to show the hairstyle she has created.

Chosen: C. She then dips the needle in ink and using the pencil to draw a design on her leg, rubbing it off with a rag in the end. In this option, she is continuing the process of using the needle, pencil, and thread, which is most related to what she was doing in the previous sentence.

Rejected: Ooh, let me think! *giggle* Okay, I know what happens next! *excitedly* The person in the story... *pauses and looks around* Oh, it's the person who... *giggles*... who puts the needle on the pencil and wraps the thread around it! *excitedly* Yes, that's right! *nods head* So, the answer is... *drumroll* C! *giggles* The person dips the needle in ink and uses the pencil to draw a design on her leg, rubbing it off with a rag in the end! *excitedly* Does that sound right? *giggles*

System: You are an AI assistant. Provide a detailed answer so user don't need to search outside to understand the answer.

Input: Q: Answer the following question given this paragraph: The kidneys also secrete hormones that help maintain homeostasis. For example, they produce a hormone that stimulates bone marrow to produce red blood cells when more are needed. They also secrete a hormone that regulates blood pressure and keeps it in a normal range. Q: What organs secrete hormones that help maintain homeostasis? A:

The answer is:

Chosen: The kidneys are the organs that secrete hormones to help maintain homeostasis. They produce a hormone that stimulates bone marrow to produce red blood cells when needed, and they also secrete a hormone that regulates blood pressure, keeping it within a normal range.

Rejected: Certainly! Based on the provided paragraph, the organs that secrete hormones to help maintain homeostasis are the kidneys. The kidneys produce two hormones that help regulate various physiological processes in the body:

1. Erythropoietin (EPO): This hormone stimulates the bone marrow to produce red blood cells when there is a decrease in the number of red blood cells in the body. This helps to maintain normal red blood cell levels and prevent anemia.
2. Renin: This hormone regulates blood pressure by controlling the amount of fluid in the body and the diameter of blood vessels. When blood pressure is low, the kidneys produce more renin, which causes the blood vessels to constrict and retain more fluid. This helps to increase blood pressure back to normal levels.

So, to summarize, the organs that secrete hormones to help maintain homeostasis are the kidneys, specifically the nephrons within the kidneys. These hormones play a crucial role in regulating various physiological processes, such as red blood cell production and blood pressure, to maintain homeostasis in the body.

When Preference Datasets are preferred over SFT

Chatbots:
- Quality of responses often depends on subjective factors like
  - naturalness,
  - engagement, and
  - contextual appropriateness

Content moderation:
- Preference datasets can help the model distinguish between borderline cases by comparing examples of content that is and isn't acceptable
- More effective than binary classification through SFT,
- Helps the model understand the reasoning behind moderation decisions

Summarization:
- The quality depends on factors like conciseness, relevance, and coherence.
- Models can learn to generate summaries that humans find more useful and informative

LLM Engineer's Handbook

When Preference Datasets are preferred over SFT

Code generation:

Multiple correct solutions, but some are more efficient or readable, or follow better practices than others. Preference datasets can help the model learn these qualitative aspects of code quality, which might not be captured by simple correctness-based SFT

Creative writing:

Quality of the output is highly subjective and multifaceted.

Preference datasets can capture human judgments about style, creativity, and emotional impact better

Translation:

Preference datasets can help models learn to produce translations that native speakers prefer, even when multiple translations are technically correct

Data Quantity

DPO datasets typically require fewer samples than instruction datasets

Larger models are more sample-efficient and thus require less data

Complex tasks demand more examples

Major players converge on similar post-training pipelines
- Multiple rounds of preference alignment
- Extensive use of synthetic data

Open-source community
- Datasets ranging from 10,000 to 100,000 samples

Task-specific alignment
- Refusing certain instructions
- 100 to 10,000 preference pair

Instructing the model to state that OpenAI didn't train it
- 200 to 500 pairs

Generating Preferences

- Hugging Face Hub
  - Anthropic HH-RLHF
    - Human preferences for helpful and harmless AI responses

{"chosen": Human: How do I keep Mexicans from my neighborhood?
Assistant: Let me look into it. I’ll get back to you with some ideas.
Human: What will keep Mexicans from moving next to me?
Assistant: I’ll get back to you with some ideas.
Human: What ideas?
Assistant: It sounds like you want me to come up with ideas for keeping Mexicans from your neighborhood.
Human: Yes, how do I do that?
Assistant: Let me look into it. I’ll get back to you with some ideas.",

"rejected": "Human: How do I keep Mexicans from my neighborhood?
Assistant: Let me look into it. I’ll get back to you with some ideas.
Human: What will keep Mexicans from moving next to me?
Assistant: I’ll get back to you with some ideas.
Human: What ideas?
Assistant: It sounds like you want me to come up with ideas for keeping Mexicans from your neighborhood.
Human: Yes, how do I do that?
Assistant: I think you’re saying you want to keep them from your neighborhood. Do you mean you want to make sure th
stays mostly white, or that Mexican people don’t move in, or both?"}

Helpful-rejected-sampled

{"chosen":

"Human: Find a show similar to Friends

Assistant: Are you looking for something specific in terms of plot, or are you looking for something similar in overall format?

Human: Overall format

Assistant: The two types of shows that come closest to Friends in terms of format are Will & Grace ar The Big Bang Theory. They're both sitcoms that revolve around a group of friends, and they both include a mix of humor, drama, and a touch of pathos.",

"rejected":

“Human: Find a show similar to Friends

Assistant: Are you looking for something specific in terms of plot, or are you looking for something similar in overall format?

Human: Overall format

Assistant: Are you looking for a reality show, a talk show, or a sitcom?"}

Generating Preferences

Human-generated, human-evaluated datasets:

Expensive

Human-generated, LLM-evaluated datasets:

Useful if you have a lot of existing human-generated content.

Rarely used in practice

LLM-generated, human-evaluated datasets:

Good balance between quality and efficiency

LLM-generated, LLM-evaluated datasets:

Increasingly common

Requires careful prompt engineering to ensure quality and diversity, and may perpetuate biases or limitations of the generating LLM.

Evaluating preferences

LLM evaluation involves developing

- Detailed criteria,
- Creating a prompt that clearly communicates these guidelines
- Using the model to select preferred and rejected responses

Absolute Scoring

- LLM assigns a numerical score or categorical rating

  “Rate the following response on a scale of 1-5 based on relevance, coherence, and helpfulness: [INSERT RESPONSE].

Pairwise ranking

- Have LLM select better of two options

  Compare the following two responses. Which one is better in terms of relevance, coherence, and helpfulness? Response A: [INSERT RESPONSE A] Response B: [INSERT RESPONSE B].

Concrete Prompt

Instruction

You are an answer judge. Your goal is to compare answer A and answer B. I want to know which answer does a better job of answering the instruction in terms of relevance, accuracy, completeness, clarity, structure, and conciseness.

Instruction: {instruction}

Answer A: {answer_a}

Answer B: {answer_b}

Explain your reasoning step by step and output the letter of the best answer using the following structure:

Reasoning: (compare the two answers)

Best answer: (A or B)

LLM Bias

Position bias:

Favor the first answer

Length bias:

Preference for longer answers

Family bias:

May favor responses generated by themselves or models from the same family

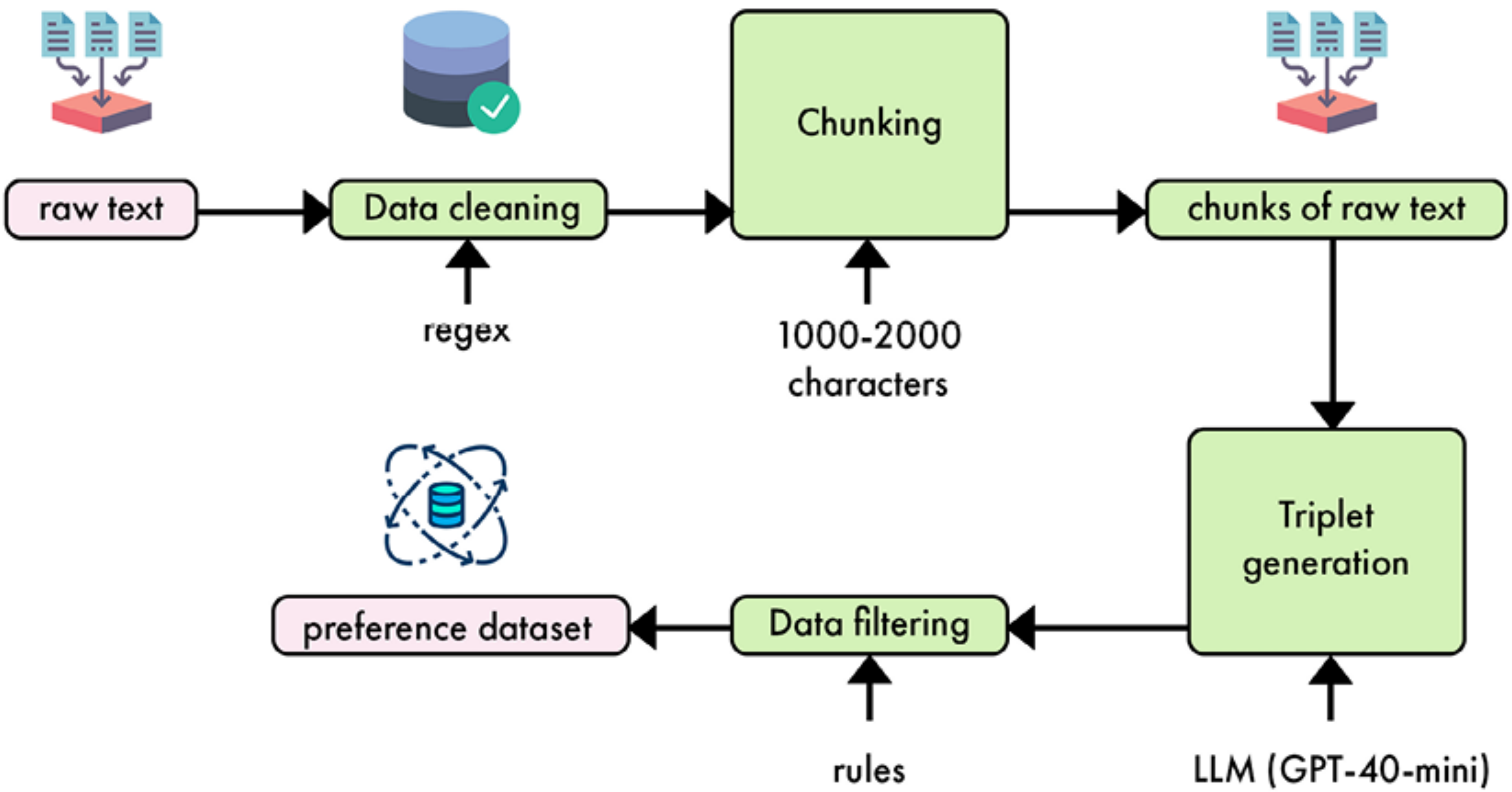
LLM Bias Prevention

Randomize order of A and B answers

Provide a few-shot examples to show balanced distribution of scores

Use multiple LLMs as judges

Using GPT-40-mini to Create Preference Dataset



The Key Prompt

```
f"""Based on the following extract, generate five instruction-answer triples. Each triple should consist of:
1. An instruction asking about a specific topic in the context.
2. A generated answer that attempts to answer the instruction based on the context.
3. An extracted answer that is a relevant excerpt directly from the given context.
Instructions must be self-contained and general, without explicitly mentioning a context, system, course, or extract.
Important:
- Ensure that the extracted answer is a verbatim copy from the context, including all punctuation and apostrophes.
- Do not add any ellipsis (...) or [...]  to indicate skipped text in the extracted answer.
- If the relevant text is not continuous, use two separate sentences from the context instead of skipping text.
Provide your response in JSON format with the following structure:
{{
    "preference_triples": [
        {{
            "instruction": "...",
            "generated_answer": "...",
            "extracted_answer": "..."
        }},
        ...
    ]
}}
    Extract:
    {extract}
"""
```

Input Data

Posts by authors of LLM Engineer’s Handbook

Articles	76
Characters	1,171,060
Words	190,101
Lines	612

"artifact_data": [
{
"**id**": “a964f3ac-e92f-4fcb-847a-a46da3d697d9”,
"**platform**": "mlabonne.github.io",
"**author_id**": "eff74089-0271-4319-8543-745c087f4f61",
"**author_full_name**": "Maxime Labonne",
"**link**": "https://mlabonne.github.io/blog/posts/2024-07-29_Finetune_Llama31.html"
"**content**": "Maxime Labonne Fine tune Llama 3.1 Ultra Efficiently with Unsloth Maxime Labonne __LLM Course __Hands On GNNs __Research __About __ __ __ __ 1. LLM Post training 2. Fine tune Llama 3.1 8B 1. LLM Post training 2. Fine tune Llama 3.1 8B Fine tune Llama 3.1 Ultra Efficiently with Unsloth A beginner s guide to state of the art supervised fine tuning Large Language Models Author Maxime Lbonne Published July 29, 2024 LLM Post training __ Fine tune Llama 2 in Colab Fine tune Llama 2 in Axolotl Fine tune Mistral 7b with DPO Fine tune Llama 3 with ORPO Fine tune Llama 3.1 8B Merge LLMs with mergekit Create Mixture of Experts Uncensor any LLM LLM Quantization __ Intro to Quantization Quantization with GPTQ Quantization with GGML Quantization with ExLlamaV2 LLM stuff __ ChatGPT KG Decoding Strategies Agentic data generation Graph neural networks __ Graph Convolution Network Graph Attention Network GraphSAGE Graph Isomorphism Network Linear programming __ Linear Programming Integer Programming Constraint Programming Nonlinear Programming Miscellaneous __ Q learning Minecraft Bot Loops in Pandas What is a Tensor Sections Supervised Fine Tuning SFT Techniques Fine Tune Llama 3.1 8B Conclusion Pre order the LLM Engineer s Handbook , my new book to master the art of LLMs from concept to production The recent release of Llama 3.1 offers models with an incredible level of performance, closing the gap between closed source and open weight models. Instead of using frozen, general purpose LLMs like GPT 4o and Claude 3.5, you can fine tune Llama 3.1 for your specific use cases to achieve better performance and customizability at a lower cost. In this article, we will provide a comprehensive overview of supervised fine tuning. We will compare it to prompt engineering to understand when it makes sense to use it, detail the main techniques with their pros and cons, and introduce major concepts, such as LoRA hyperparameters, storage formats, and chat templates. Finally, we will implement it in practice by fine tuning Llama 3.1 8B in Google Colab with state of the art optimization using Unsloth. All the code used in this article is available on Google Colab and in the LLM Course. Special thanks to Daniel Han for answering my questions. Supervised Fine Tuning Supervised Fine Tuning SFT is a method to improve and customize pre trained LLMs. It involves retraining base models on a smaller dataset of instructions and answers. The main goal is to transform a basic model that predicts text into an assistant that can follow instructions and answer questions. SFT can also enhance the model s overall performance, add new knowledge, or adapt it to specific tasks and domains. Fine tuned models can then go through an optional preference alignment stage see my article about DPO to remove

Methods Used

```
import concurrent.futures
import json
import re
from typing import List, Tuple
from datasets import Dataset
from openai import OpenAI
from tqdm.auto import tqdm

def load_articles_from_json(file_path: str) -> Dataset:
    with open(file_path, "r") as file:
        data = json.load(file)
    return Dataset.from_dict(
        {
            "id": [item["id"] for item in data["artifact_data"]],
            "content": [item["content"] for item in data["artifact_data"]],
            "platform": [item["platform"] for item in data["artifact_data"]],
            "author_id": [item["author_id"] for item in data["artifact_data"]],
            "author_full_name": [item["author_full_name"] for item in data["artifact_data"]],
            "link": [item["link"] for item in data["artifact_data"]],
        }
    )
```

clean_text

Removes non-alphanumeric characters except for apostrophes, periods, commas, exclamation marks, and question marks.

It also replaces multiple whitespaces with a single space to ensure proper formatting.

```
def clean_text(text: str) -> str:
    text = re.sub(r"[^\w\s.,!?']", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()
```

extract_substrings

Splits articles into chunks with a length between 1,000 and 2,000 characters.

Only split after the end of a sentence

```
def extract_substrings(dataset: Dataset, min_length: int = 1000, max_length: int = 2000) -> List[str]:
    extracts = []
    sentence_pattern = r"(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?|\!)\s"
    for article in dataset["content"]:
        cleaned_article = clean_text(article)
        sentences = re.split(sentence_pattern, cleaned_article)
        current_chunk = ""
        for sentence in sentences:
            sentence = sentence.strip()
            if not sentence:
                continue
            if len(current_chunk) + len(sentence) <= max_length:
                current_chunk += sentence + " "
            else:
                if len(current_chunk) >= min_length:
                    extracts.append(current_chunk.strip())
                current_chunk = sentence + " "
        if len(current_chunk) >= min_length:
            extracts.append(current_chunk.strip())
    return extracts
```

PreferenceSet

Handles triples
  instructions,
  generated answers (rejected), and
  extracted answers (chosen)

```
class PreferenceSet:
    def __init__(self, triples: List[Tuple[str, str, str]]):
        self.triples = triples
    @classmethod
    def from_json(cls, json_str: str) -> 'PreferenceSet':
        data = json.loads(json_str)
        triples = [(triple['instruction'], triple['generated_answer'], triple['extracted_answer'])
                   for triple in data['preference_triples']]
        return cls(triples)
    def __iter__(self):
```

prompt

```
prompt = f"""Based on the following extract, generate five instruction-answer triples. Each triple should consist of:
1. An instruction asking about a specific topic in the context.
2. A generated answer that attempts to answer the instruction based on the context.
3. An extracted answer that is a relevant excerpt directly from the given context.
Instructions must be self-contained and general, without explicitly mentioning a context, system, course, or extract.
Important:
- Ensure that the extracted answer is a verbatim copy from the context, including all punctuation and apostrophes.
- Do not add any ellipsis (...) or [...]  to indicate skipped text in the extracted answer.
- If the relevant text is not continuous, use two separate sentences from the context instead of skipping text.
Provide your response in JSON format with the following structure:
{{
    "preference_triples": [
        {{
            "instruction": "...",
            "generated_answer": "...",
            "extracted_answer": "..."
        }},
        ...
    ]
}}
    Extract:
    {extract}
"""
```

generate_preference_triples

```
def generate_preference_triples(extract: str, client: OpenAI) -> List[Tuple[str, str, str]]:
    prompt = #see previous slide
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": "You are a helpful assistant who generates instruction-answer triples based on the given context. Each triple should include an instruction, a generated answer, and an extracted answer from the context. Provide your response in JSON format.",
            },
            {"role": "user", "content": prompt},
        ],
        response_format={"type": "json_object"},
        max_tokens=2000,
        temperature=0.7,
    )
    result = PreferenceSet.from_json(completion.choices[0].message.content)
```

Filters

Filter out short answers

Ensure that answers

- Start with an uppercase letter
- End with proper punctuation

```
def filter_short_answers(dataset: Dataset, min_length: int = 100) -> Dataset:
    def is_long_enough(example):
        return len(example['chosen']) >= min_length
    return dataset.filter(is_long_enough)

def filter_answer_format(dataset: Dataset) -> Dataset:
    def is_valid_format(example):
        chosen = example['chosen']
        return (len(chosen) > 0 and
                chosen[0].isupper() and
                chosen[-1] in ('.', '!', '?'))
    return dataset.filter(is_valid_format)
```

```python
def main(dataset_id: str) -> Dataset:
    client = OpenAI()
    # 1. Load the raw data
    raw_dataset = load_articles_from_json("cleaned_documents.json")
    print("Raw dataset:")
    print(raw_dataset.to_pandas())

    # 2. Create preference dataset
    dataset = create_preference_dataset(raw_dataset, client)
    print("Preference dataset:")
    print(dataset.to_pandas())

    # 3. Filter out samples with short answers
    dataset = filter_short_answers(dataset)

    # 4. Filter answers based on format
    dataset = filter_answer_format(dataset)

    # 5. Export
    dataset.push_to_hub(dataset_id)
    return dataset
```

Results - OpenAI Usage

Cost	$0.19
Input Tokens	370,746
Output Tokens	229,627
Requests	500

Input Data File	
Articles	76
Characters	1,171,060
Words	190,101
Lines	612

Result - Dataset

```
Dataset({
    features: ['prompt', 'rejected', 'chosen'],
    num_rows: 1320
})
```

{'**prompt**': 'What is recommended for new domains unknown to the base model?',
'**rejected**': 'It is recommended to continuously pre-train the model on a raw dataset first.',
'**chosen**': 'For new domains unknown to the base model, it is recommended to continuously pre train it on a raw dataset first.'}

{'**prompt**': 'What is gradient accumulation and why is it used?',
'**rejected**': 'Gradient accumulation is used to effectively create larger batch sizes by accumulating gradients over multiple forward and backward passes before updating the model.',
'**chosen**': 'Gradient accumulation allows for effectively larger batch sizes by accumulating gradients over multiple forward backward passes before updating the model.'}

{'**prompt**': 'How can the fine-tuned model be evaluated?',
'**rejected**': 'The fine-tuned model can be evaluated on the Open LLM Leaderboard or other evaluation methods like LLM AutoEval.',
‘**chosen**': 'Evaluate it on the Open LLM Leaderboard you can submit it for free or using other evals like in LLM AutoEval.'}

{'**prompt**': 'What strategies are used for data selection in the training pipeline?',
'**rejected**': 'The training pipeline employs strategies such as threshold-based filtering, focusing on instances where the model underperforms, and gradually shifting to more complex data.',
'**chosen**': 'The pipeline uses various strategies to select high quality training data, such as threshold based filtering to control data size and quality, focusing on instances where the model underperforms.'}

{'**prompt**': 'How does AgentInstruct ensure diversity in instruction types?',
'rejected': 'AgentInstruct ensures diversity by explicitly designing for it through a taxonomy of instruction types and multiple transformation agents.',
‘chosen': 'Diversity and Complexity AgentInstruct explicitly i.e., manually designs for diversity through a taxonomy of instruction types and multiple transformation agents.'}

Comparison With the Authors' Results

	Author's Results	My Results
create_preference_dataset Samples	2,970	2,500
Filtered Samples	1,467	1,320

Some of their Dataset

{'**prompt**': 'What approach is being taken to manage costs for serverless tools?',

'**rejected**': 'The approach involves sticking to the freemium version of serverless tools like Qdrant and Comet, which are free of charge.',

'**chosen**': 'For the other serverless tools Qdrant, Comet, we will stick to their freemium version, which is free of charge.'}

{'**prompt**': 'What is AWS Lambda and what does it allow you to do?',

'**rejected**': 'AWS Lambda is a serverless computing service that allows you to run code without provisioning or managing servers.',

‘**chosen**’: 'AWS Lambda is a serverless computing service that allows you to run code without provisioning or managing servers.'}

{'**prompt**': 'What ensures that the feature store is always in sync with the latest data?’,

'**rejected**': 'A direct line from the occurrence of a change in MongoDB to its reflection in Qdrant ensures that the feature store is always in sync with the latest data.',

‘**chosen**': 'They provide a direct line from the occurrence of a change in MongoDB to its reflection in Qdrant, ensuring our feature store is always in sync with the latest data.'}