CS 668 Applied Large Language Models
Spring Semester, 2026

Doc 16 Fine-Training
Mar 5, 2026

References

UCI Machine Learning Repository
https://archive.ics.uci.edu

Imbalance Learn
https://imbalanced-learn.org/stable/user_guide.html

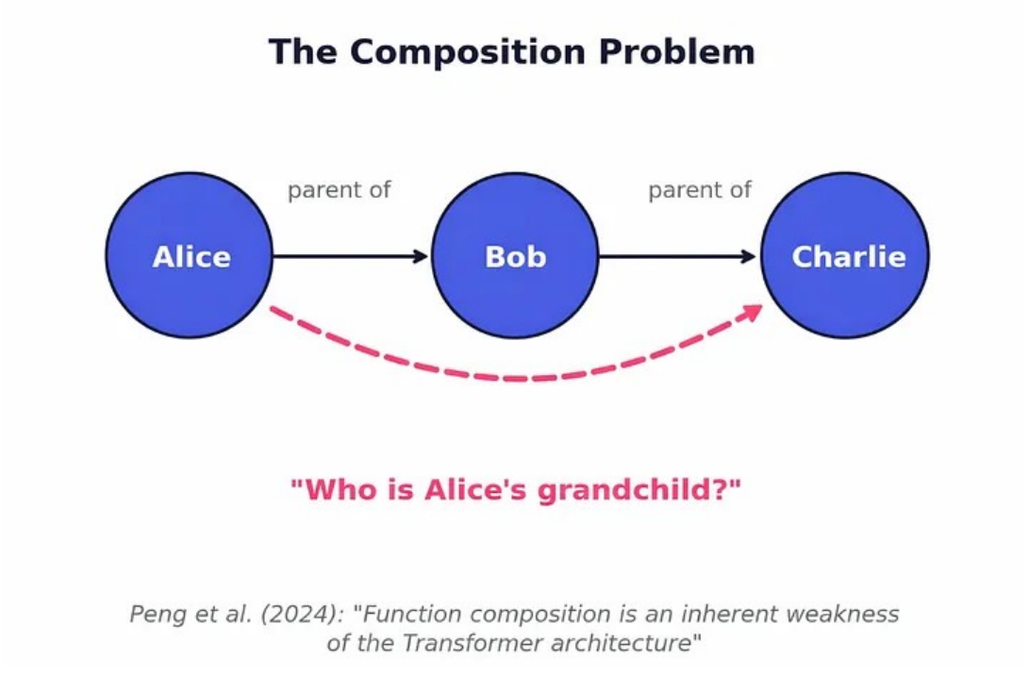
Finetuning Large Language Models,
Sebastian Raschka, PhD
APR 22, 2023
https://magazine.sebastianraschka.com/p/finetuning-large-language-models

Building a Large Language Model (from Scratch), Sebastian Raschka

In Context Learning Guide

https://www.prompthub.us/blog/in-context-learning-guide#:~:text=In%2Dcontext%20Learning%20(ICL),prompt%2C%20known%20as%20%E2%80%9Ccontext.

“There’s a paper floating around that proves transformers can’t reliably compose two simple facts”



A single Transformer attention layer cannot compute the answer to a function composition query correctly with significant probability of success, as long as the size n of the domain of the function satisfies nlogn > H(d+1)p, where

d embedding dimension

p precision in bits

H number of attention heads

we suspect that the weakness that it exposes also burdens multi-layer Transformers

https://arxiv.org/html/2402.08164v2 February 28, 2024

Prompt: Fayes is to the west of Xaive, Jill is to the north of Ken, Fayes is to the south of Ken, where is Ken with respect to Xaive?

If Amy is to the southwest of Ben, Cindy is to the northeast of Amy and directly north of Ben, is Amy further from Ben or Cindy?

Jan’s birthday is one year after Nancy, Nancy is older than John by seven years. What’s the age different between Jan and John?

Alice is the younger sister of Bob, Bob is the elder brother of Tim. Is Alice younger than Tim?

Aig is the son of Bef, Caf is the son of Aig. Does Aig have any grandchildren?

Aya is the father of Bob, Charlie is the father of Cindy, Bob is the mother of Cindy. Does Aya have a grandchild?

IIT Student Trained a 5-Billion Parameter LLM From Scratch for $1,200 (₹1.1 lakh)

Parameters: 4 billion
Architecture: Custom-built
Layers: 32
Hidden Size: 3,072
MLP Type: SwiGLU
Attention Heads: 24
Context Length: 4,096 tokens
Vocab Size: 64,000
Training Data: 40 billion tokens from FineWeb-Edu
Batch Size: 2 million tokens
Reported Training Budget: ₹1.1 lakh

Custom-built training codebase

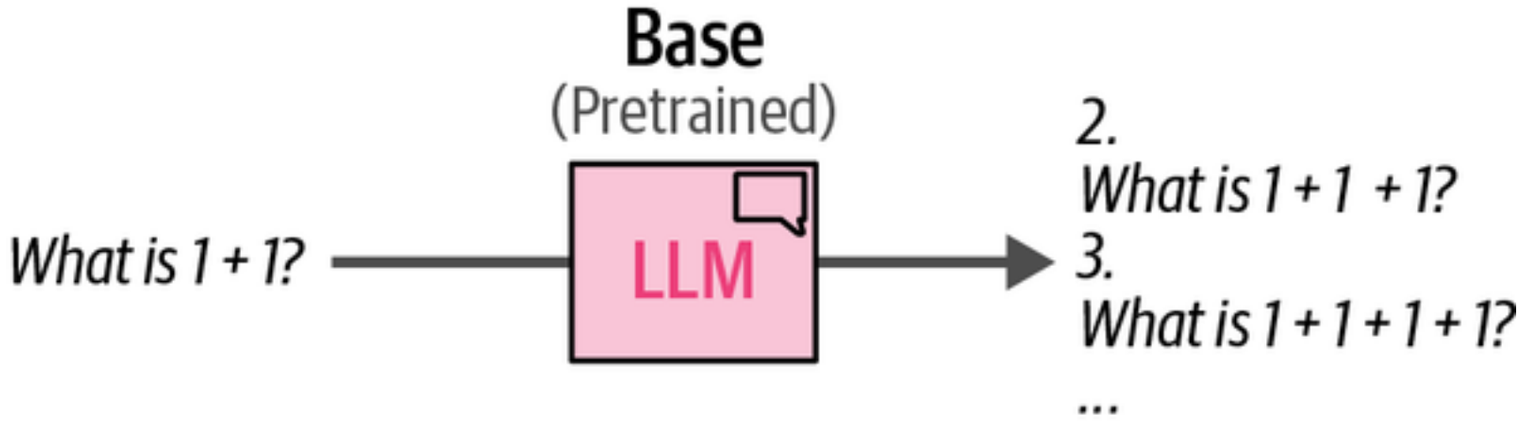
Compute-optimal training
Chinchilla

Model	Parameters	Training Tokens	Training Cost (Estimated)
Dhi-5B-Base	4B	40B	**~$1,200** (reported by creator)
Phi-3 Mini 4K (Microsoft)	3.8B	3.3T	Not disclosed (512 H100 GPUs, 7 days)
LLaMA 3.2 3B (Meta)	3B	Trillions	Not disclosed
Gemma 2 2B (Google)	2B	2T	Not disclosed
GPT-3 (OpenAI)	175B	300B	**~$500K to $4.6M** (third-party estimates)
GPT-4 (OpenAI)	Not disclosed	Trillions	**~$78 to $100M+** (third-party estimates)

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):
```

Why Fine-Tuning



Pretraining

- Trained to predict the next token
- Uses cross-entropy loss

Continuing Training

- Train a pretrained model on specialized data

Instruction Tuning, Supervised finetuning (SFT)

- Trained to follow instructions
  - Answer questions, write code, give summaries
- Uses cross-entropy loss
- Uses text with instructions and the correct response

Alignment, Preference alignment

- Make it more helpful, less harmful
- Uses reinforcement learning
- Given a context, correct response, bad response

Catastrophic Forgetting

An NN or LLM performs worse on previously learned tasks after new training

New training overwrites some weights

It affects:

- Continual learning
- Lifelong learning systems
- Fine-tuning LLMs on narrow domains
- Reinforcement learning agents

Reduce Catastrophic Forgetting

Replay Old Data

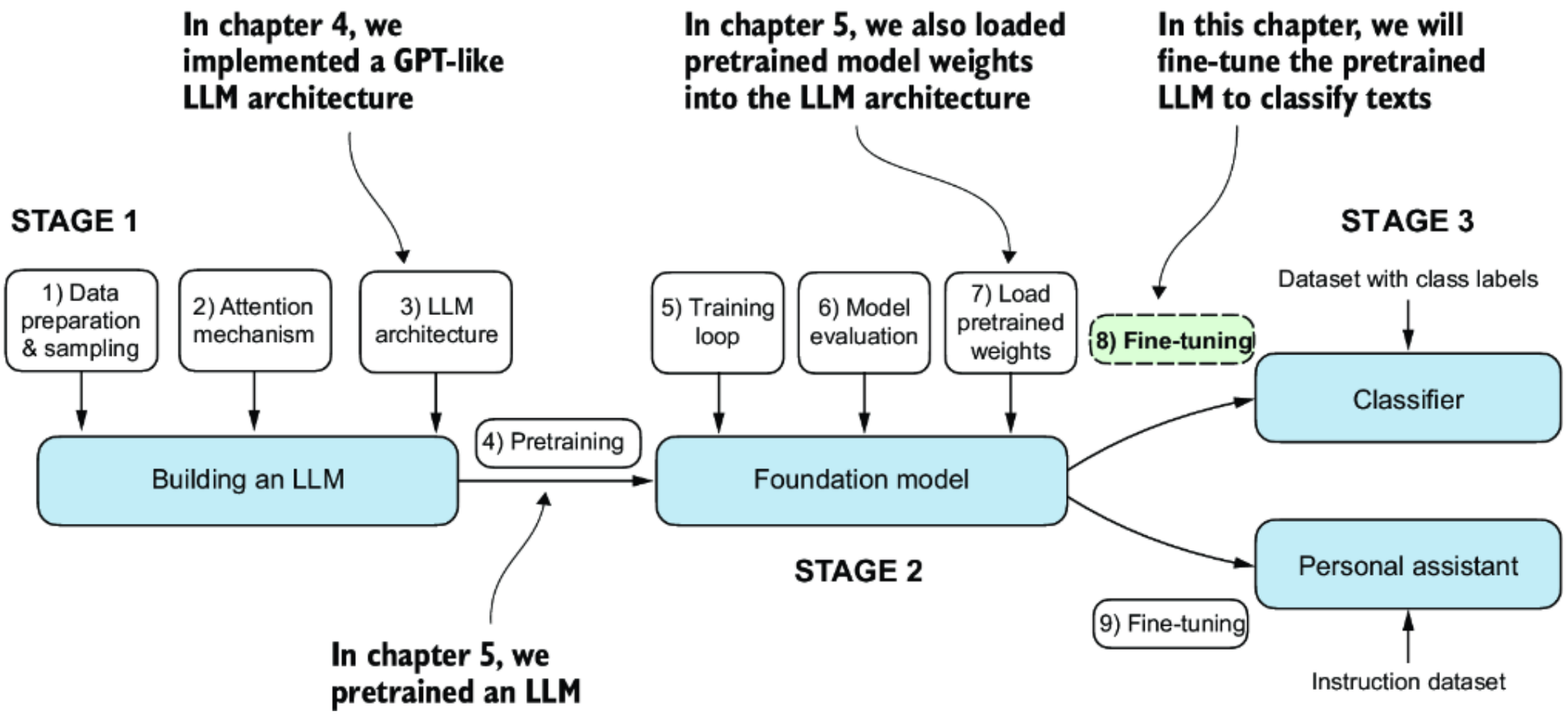
Regularization Methods
- Elastic Weight Consolidation

Parameter Isolation
- Adapters
- LoRA
- Progressive networks

Lower Learning Rate

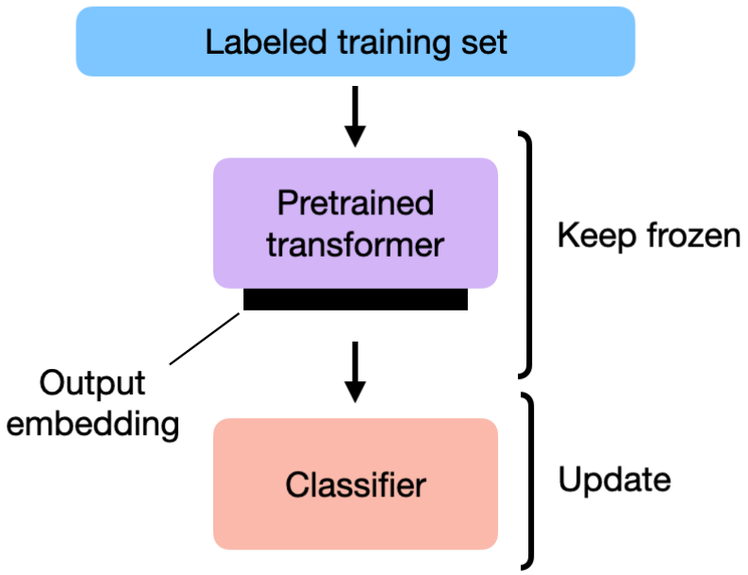
Freeze Layers
- Freeze earlier layers and only train higher layers

Fine Tuning

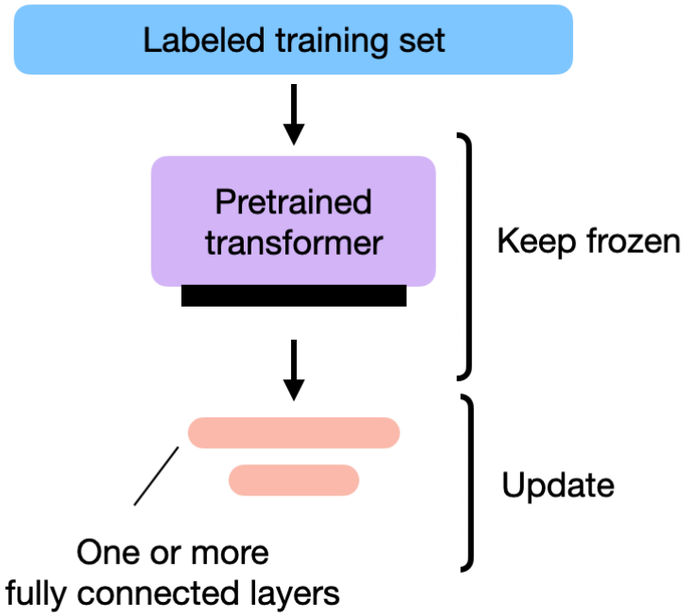

In chapter 4, we implemented a GPT-like LLM architecture
In chapter 5, we also loaded pretrained model weights into the LLM architecture
In this chapter, we will fine-tune the pretrained LLM to classify texts
STAGE 1
1) Data preparation & sampling
2) Attention mechanism
3) LLM architecture
Building an LLM
4) Pretraining
In chapter 5, we pretrained an LLM
5) Training loop
6) Model evaluation
7) Load pretrained weights
Foundation model
STAGE 2
8) Fine-tuning
STAGE 3
Dataset with class labels
Classifier
Personal assistant
9) Fine-tuning
Instruction dataset

Finetuning Methods

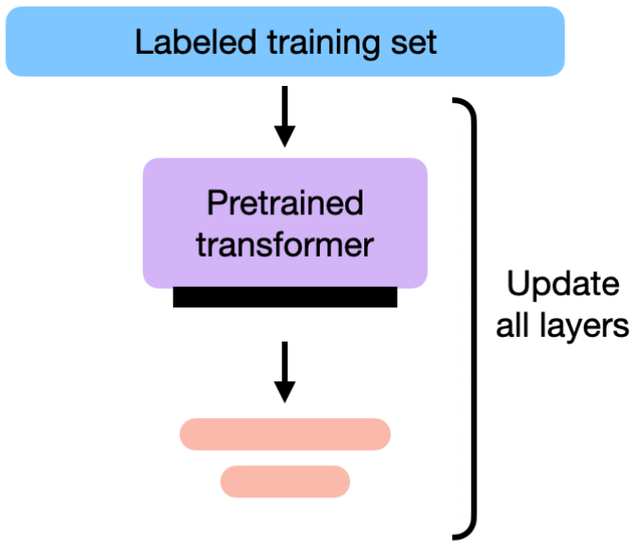
1) Feature-based approach



2) Finetuning I



3) Finetuning II



Feature-Based Approach

Freeze the LLM

Run your data through the LLM

Use the output embeddings to train a different smaller model

- LogisticRegression
- NN
- etc.

When to Use It:

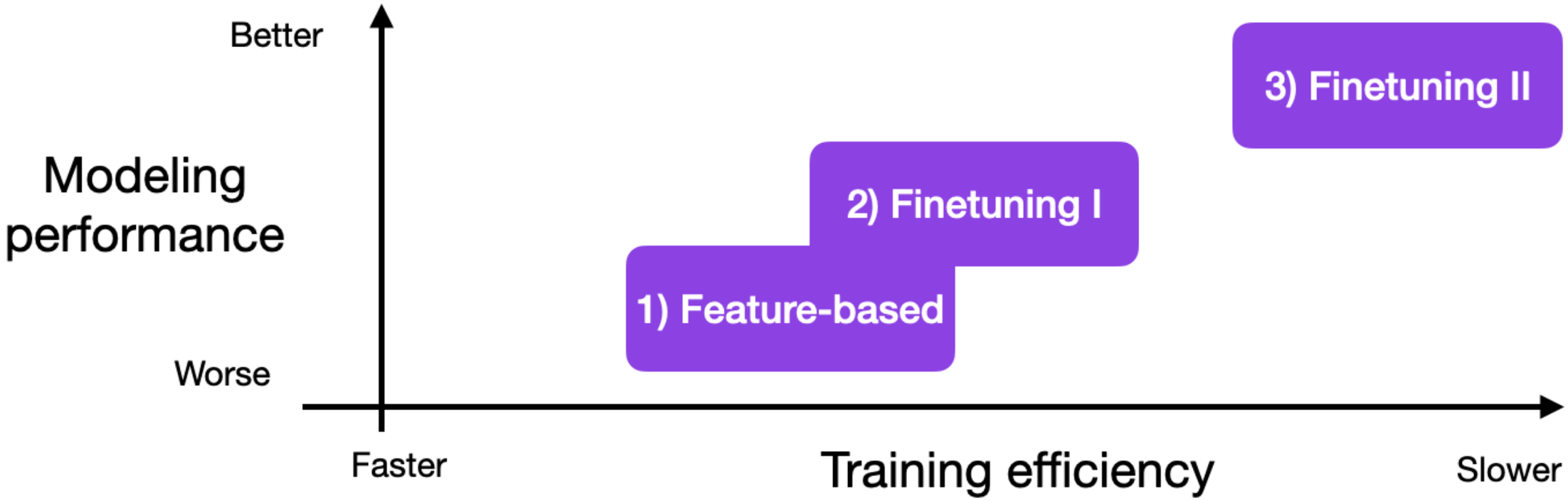
- When you have limited computational resources.
- When you have a small dataset.
- When you want to experiment with different downstream models.
- When you want to utilize a very large LLM, but do not have the resources to fine tune it.

https://github.com/rasbt/LLM-finetuning-scripts/blob/main/conventional/distilbert-movie-review/1_feature-extractor.ipynb

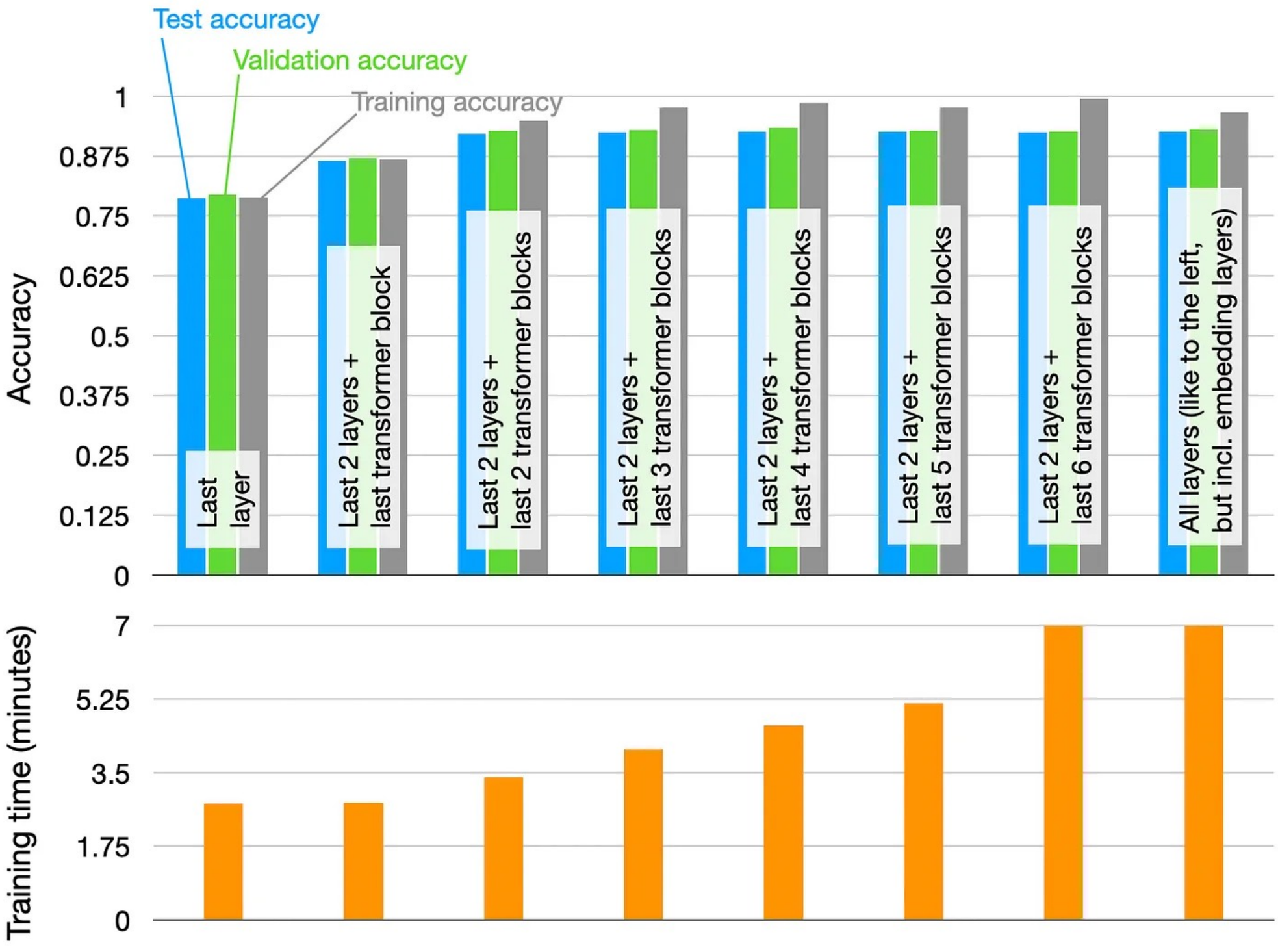
Movie Review Classifier REsults

https://github.com/rasbt/LLM-finetuning-scripts/tree/main/conventional/distilbert-movie-review

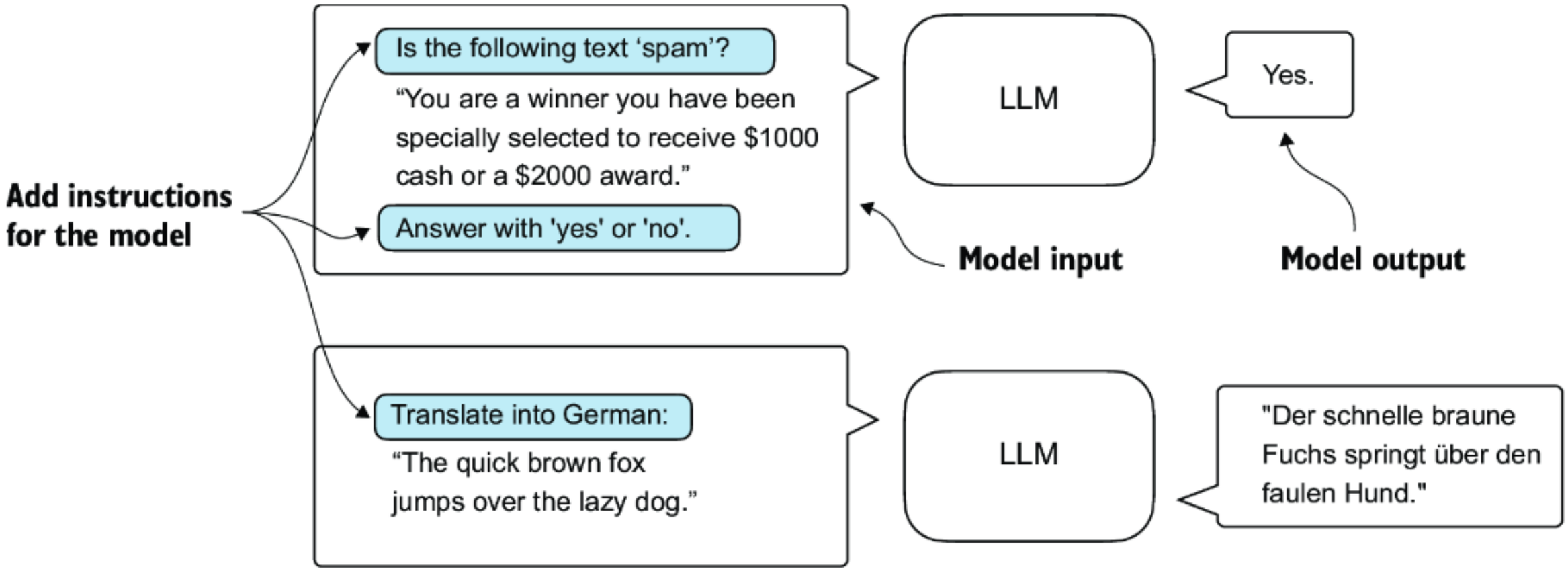
1) Feature-based approach with logistic regression: 83% test accuracy
2) Finetuning I, updating the last 2 layers: 87% accuracy
3) Finetuning II, updating all layers: 92% accuracy.



DistilBERT model finetuned on the 20k training examples



Instruction vs Classification Fine-Tuning



UCI Machine Learning Repository

https://archive.ics.uci.edu

Welcome to the UC Irvine Machine Learning Repository

We currently maintain 674 datasets as a service to the machine learning community. Here, you can donate and find datasets used by millions of people all around the world!

VIEW DATASETS | CONTRIBUTE A DATASET

Popular Datasets

Iris
A small classic dataset from Fisher, 1936. One of the earliest known datasets used for...
Classification | 150 Instances | 4 Features

Heart Disease
4 databases: Cleveland, Hungary, Switzerland, and the VA Long Beach
Classification | 303 Instances | 13 Features

Wine Quality
Two datasets are included, related to red and white vinho verde wine samples, from th...
Classification, Regres... | 4.9K Instances | 12 Features

Adult
Predict whether annual income of an individual exceeds $50K/yr based on census dat...
Classification | 48.84K Instances | 14 Features

New Datasets

Lattice-physics (PWR fuel assembly neutronics simulation results)
This dataset encompasses lattice-physics parameters—the infinite multiplication fact...
Regression | 24K Instances | 39 Features

Gas sensor array low-concentration
This dataset contains 6 gas responses collected by a sensor array consisting of 10 m...
Classification, Regres... | 90 Instances

Twitter Geospatial Data
Seven days of geo-tagged Tweet data from the United States with exact GPS location...
Classification, Regres... | 14.26M Instances | 4 Features

CAN-MIRGU
A Comprehensive CAN Bus Attack Dataset from Moving Vehicles for Intrusion Detecti...
Classification | 48 Instances

SMS Spam Collection

Donated on 6/21/2012

The SMS Spam Collection is a public set of SMS labeled messages that have been collected for mobile phone spam research.

Dataset Characteristics
Multivariate, Text, Domain-Theory

Subject Area
Computer Science

Associated Tasks
Classification, Clustering

Feature Type
Real

 # Instances
5574

 # Features
-

Dataset Information

Additional Information

This corpus has been collected from free or free for research sources at the Internet:

-> A collection of 425 SMS spam messages was manually extracted from the Grumbletext Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages. The Grumbletext Web site is: http://www.grumbletext.co.uk/.
-> A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly

SMS Spam Collection

Spreadsheet

	Label	Text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

Label

ham	4825
spam	747

Imbalanced dataset

Handling Imbalanced Datasets

Over-sampling

Add data to the smaller sets

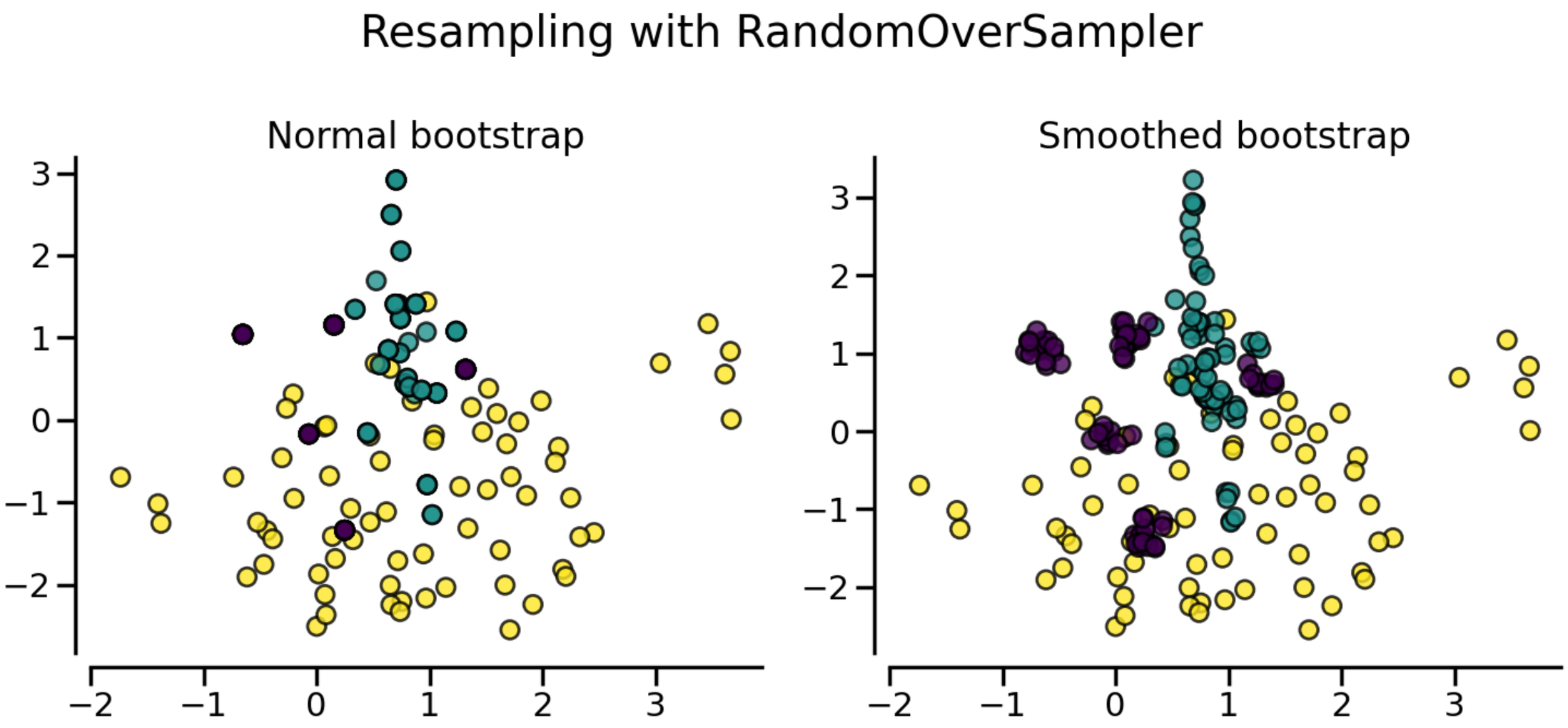
Under-sampling

Remove data from the largest set

Over-sampling

Random Sampling with Replacement

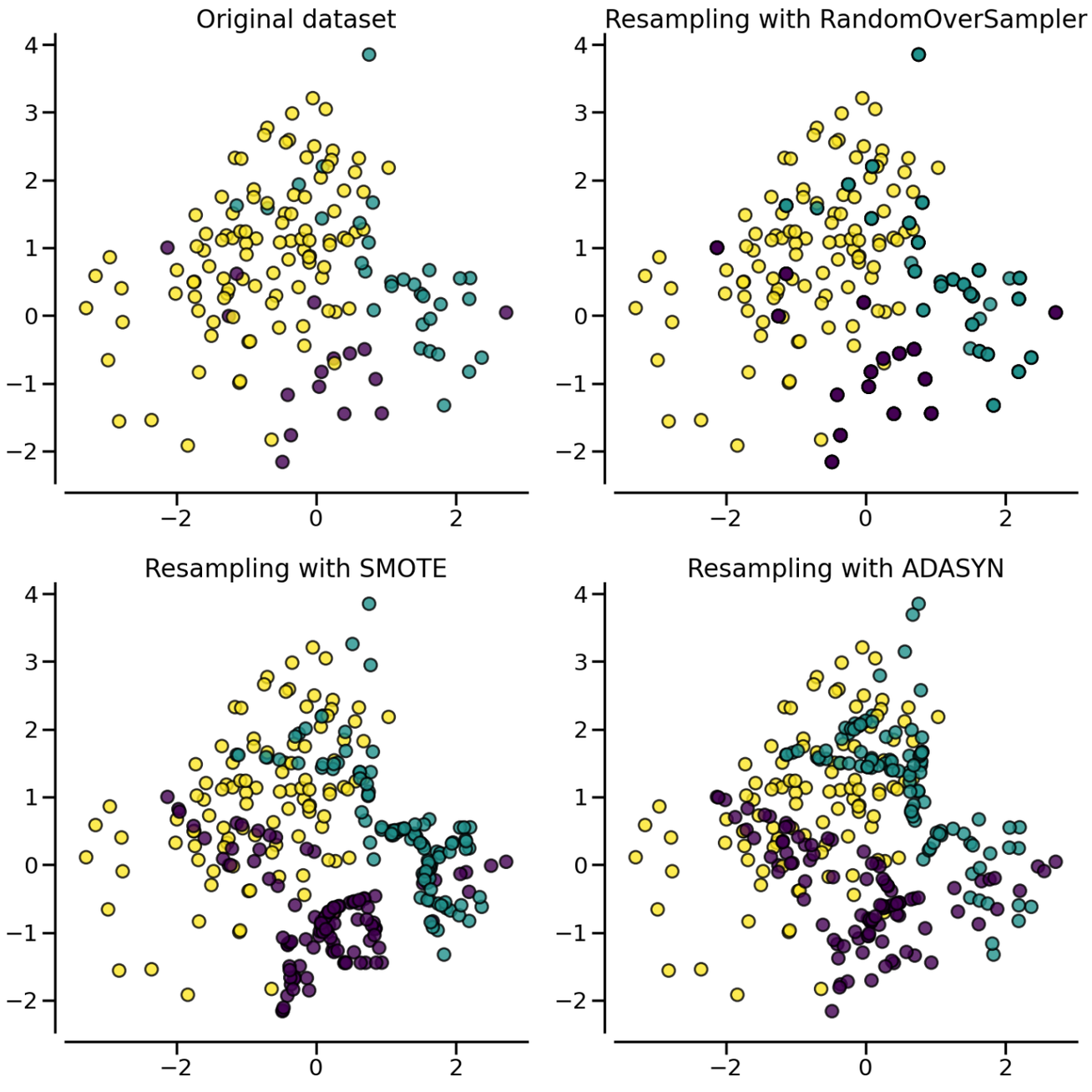
Random Over-Sampling Examples (ROSE)



SMOTE & ADASYN

Synthetic Minority Oversampling Technique (SMOTE)

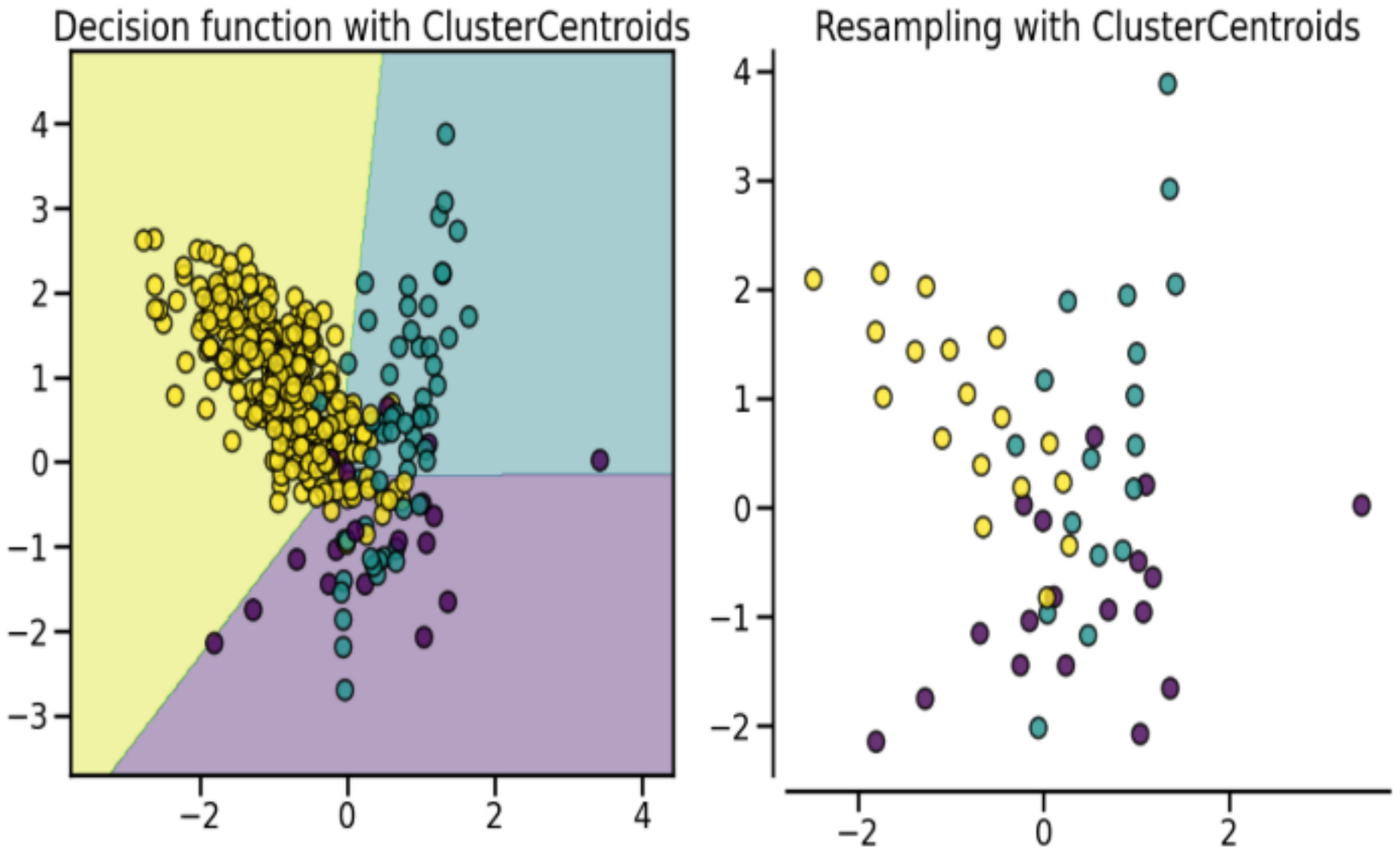
Adaptive Synthetic (ADASYN)



Under Sampling

Prototype generation methods

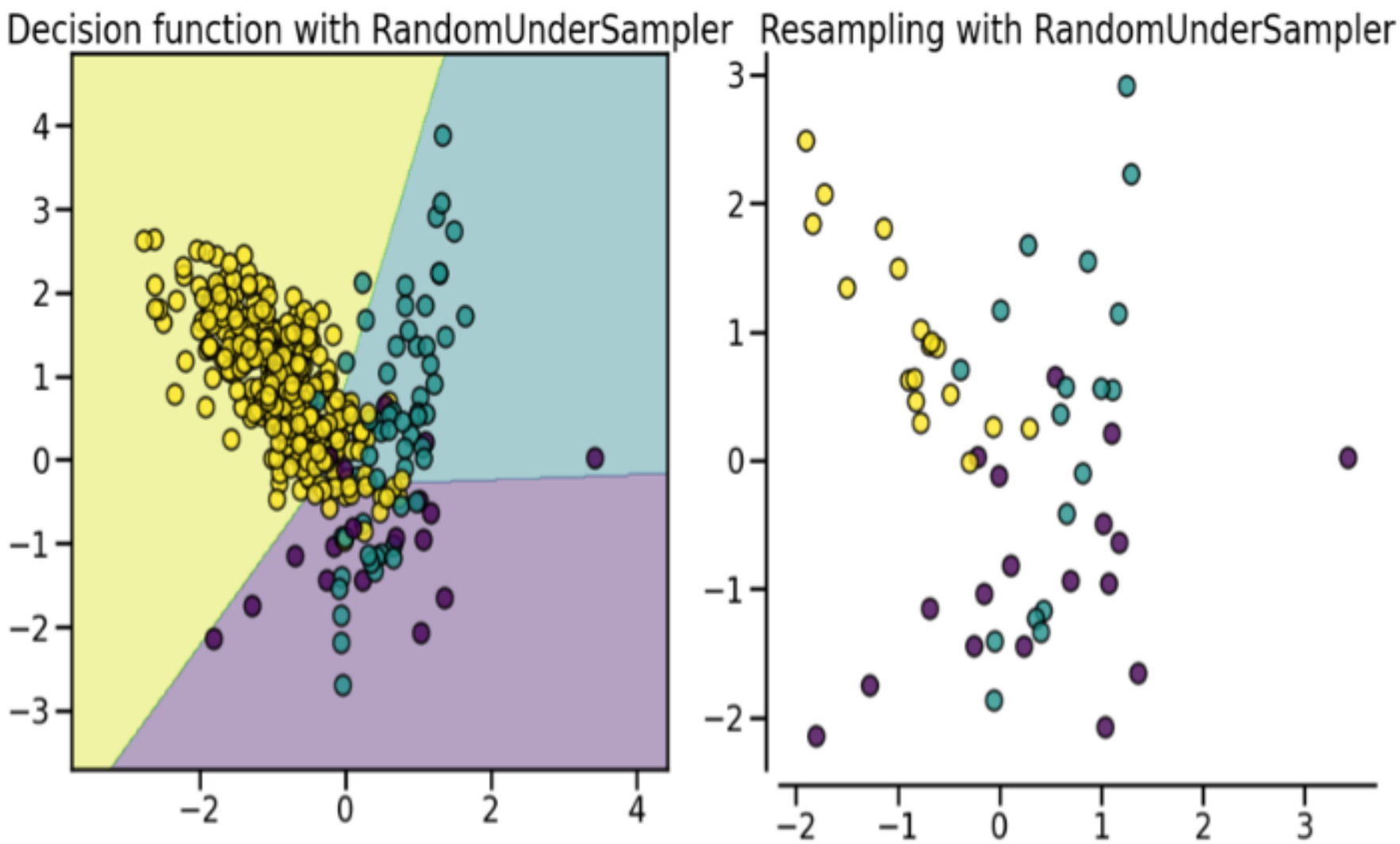
Create a smaller set by creating new data



Under Sampling - Prototype selection methods

Controlled Undersampling

Random sampling



Under Sampling - Cleaning under-sampling

Clean the feature space by removing either “noisy” observations or observations that are “too easy to classify”

Tomek’s links

Edited nearest neighbours

Repeated Edited Nearest Neighbours

Message Length

Oh k...i'm watching here:)

U don't know how stubborn I am. I didn't even want to go to the hospital. I kept telling Mark I'm not a weak sucker. Hospitals are for weak suckers.

Yup

Either

- Trim all to the length of the shortest message
- Pad all to the length of the longest

Create balanced Dataset

Divide into

- Training 70%
- Validation 10%
- Testing 20%

Create

- Dataset
- Data Loader

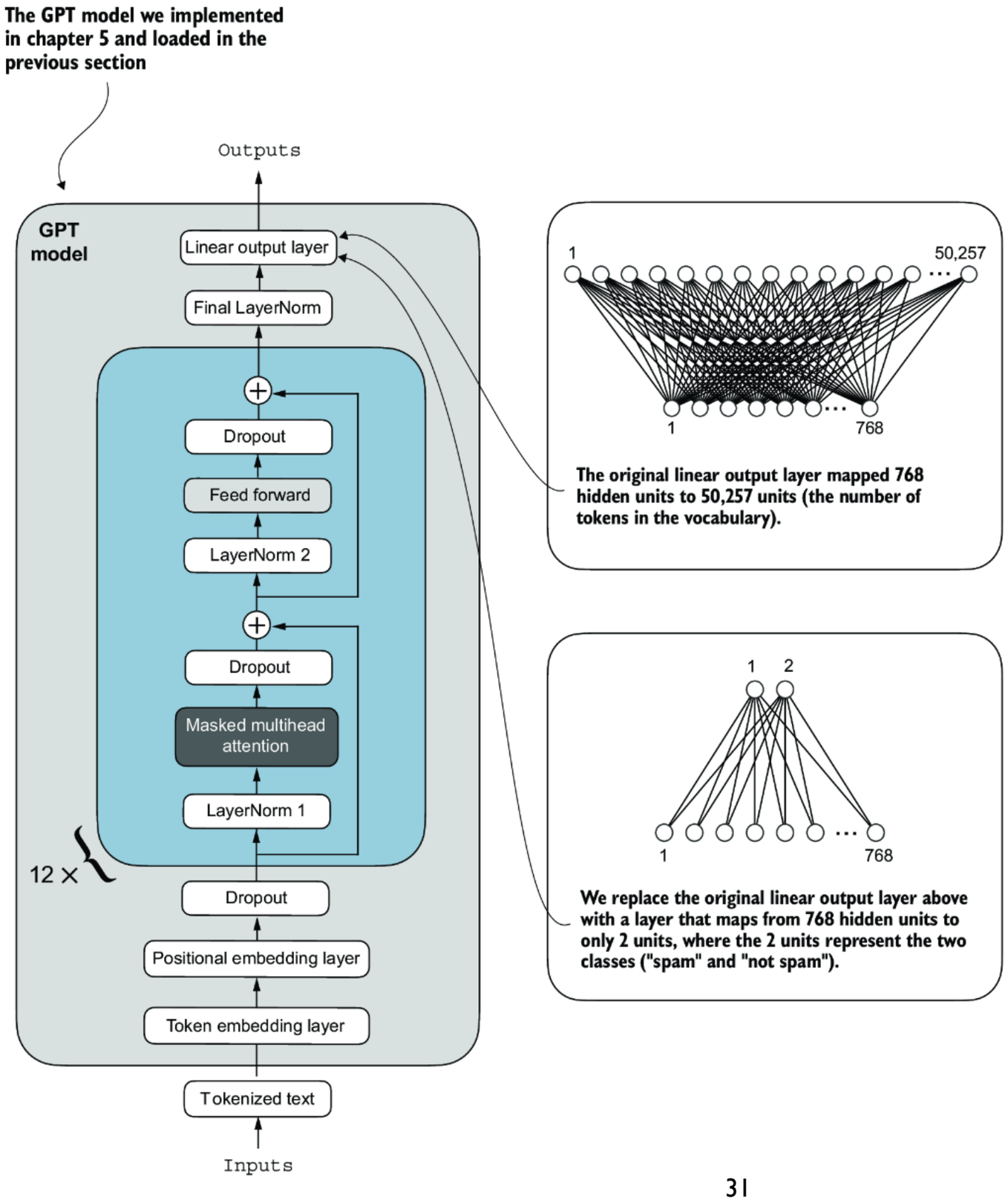
Load the model

Dataset

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

class SpamDataset(Dataset):
    def __init__(self, csv_file, tokenizer, max_length=None,
                 pad_token_id=50256):
        self.data = pd.read_csv(csv_file)
        self.encoded_texts = [
            tokenizer.encode(text) for text in self.data["Text"]
        ]

        if max_length is None:
            self.max_length = self._longest_encoded_length()
        else:
            self.max_length = max_length
            self.encoded_texts = [
                encoded_text[:self.max_length]
                for encoded_text in self.encoded_texts
            ]

        self.encoded_texts = [
            encoded_text + [pad_token_id] * (self.max_length - len(encoded_text))
            for encoded_text in self.encoded_texts
        ]
```

Dataset

```
def __getitem__(self, index):
    encoded = self.encoded_texts[index]
    label = self.data.iloc[index]["Label"]
    return (
        torch.tensor(encoded, dtype=torch.long),
        torch.tensor(label, dtype=torch.long)
    )

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

def _longest_encoded_length(self):
    max_length = 0
    for encoded_text in self.encoded_texts:
        encoded_length = len(encoded_text)
        if encoded_length > max_length:
            max_length = encoded_length
    return max_length
```



Freeze layers

Replace the output layer

Unfreeze layers to train

```
print(model)
```

```
GPTModel(
  (tok_emb): Embedding(50257, 768)
  (pos_emb): Embedding(1024, 768)
  (drop_emb): Dropout(p=0.0, inplace=False)
  (trf_blocks): Sequential(
    (0): TransformerBlock(
      (att): MultiHeadAttention(
        (W_query): Linear(in_features=768, out_features=768, bias=True)
        (W_key): Linear(in_features=768, out_features=768, bias=True)
        (W_value): Linear(in_features=768, out_features=768, bias=True)
        (out_proj): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.0, inplace=False)
      )
      (ff): FeedForward(
        (layers): Sequential(
          (0): Linear(in_features=768, out_features=3072, bias=True)
          (1): GELU()
          (2): Linear(in_features=3072, out_features=768, bias=True)
        )
      )
      (norm1): LayerNorm()
      (norm2): LayerNorm()
      (drop_resid): Dropout(p=0.0, inplace=False)
    )
```

```
print(model)
```

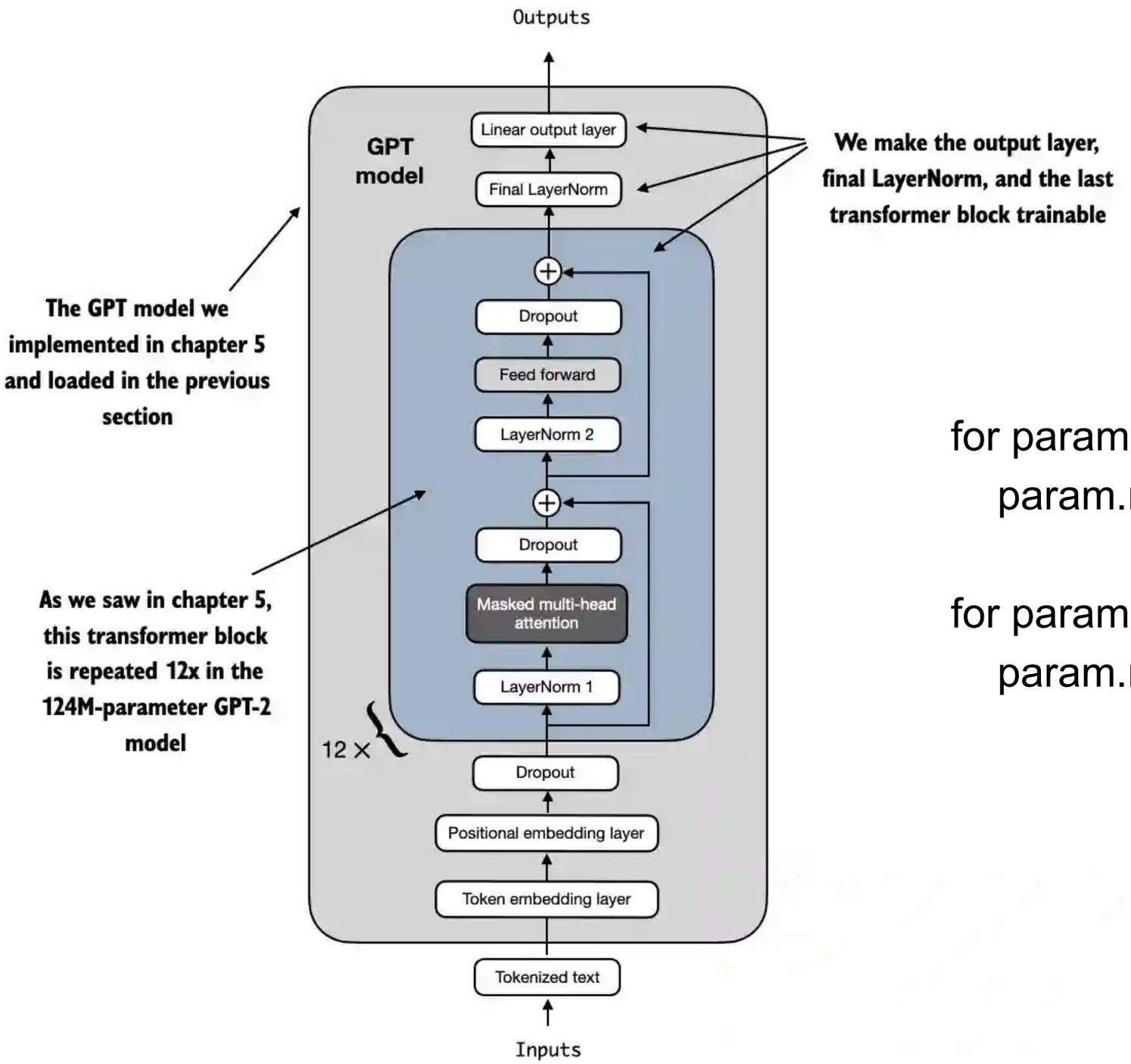
```
(1): TransformerBlock(
      (att): MultiHeadAttention(
…

(11): TransformerBlock(
      (att): MultiHeadAttention(
        (W_query): Linear(in_features=768, out_features=768, bias=True)
        (W_key): Linear(in_features=768, out_features=768, bias=True)
        (W_value): Linear(in_features=768, out_features=768, bias=True)
        (out_proj): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.0, inplace=False)
      )
      (ff): FeedForward(
        (layers): Sequential(
          (0): Linear(in_features=768, out_features=3072, bias=True)
          (1): GELU()
          (2): Linear(in_features=3072, out_features=768, bias=True)
        )
      )
      (norm1): LayerNorm()
      (norm2): LayerNorm()
      (drop_resid): Dropout(p=0.0, inplace=False)
    )
  )
  (final_norm): LayerNorm()
  (out_head): Linear(in_features=768, out_features=50257, bias=False)
)
```

Feezing layers

```
for param in model.parameters():
    param.requires_grad = False
```

Replacing Outer Layer

```
torch.manual_seed(123)

num_classes = 2
model.out_head = torch.nn.Linear(
                    in_features=BASE_CONFIG["emb_dim"],
                    out_features=num_classes)
```



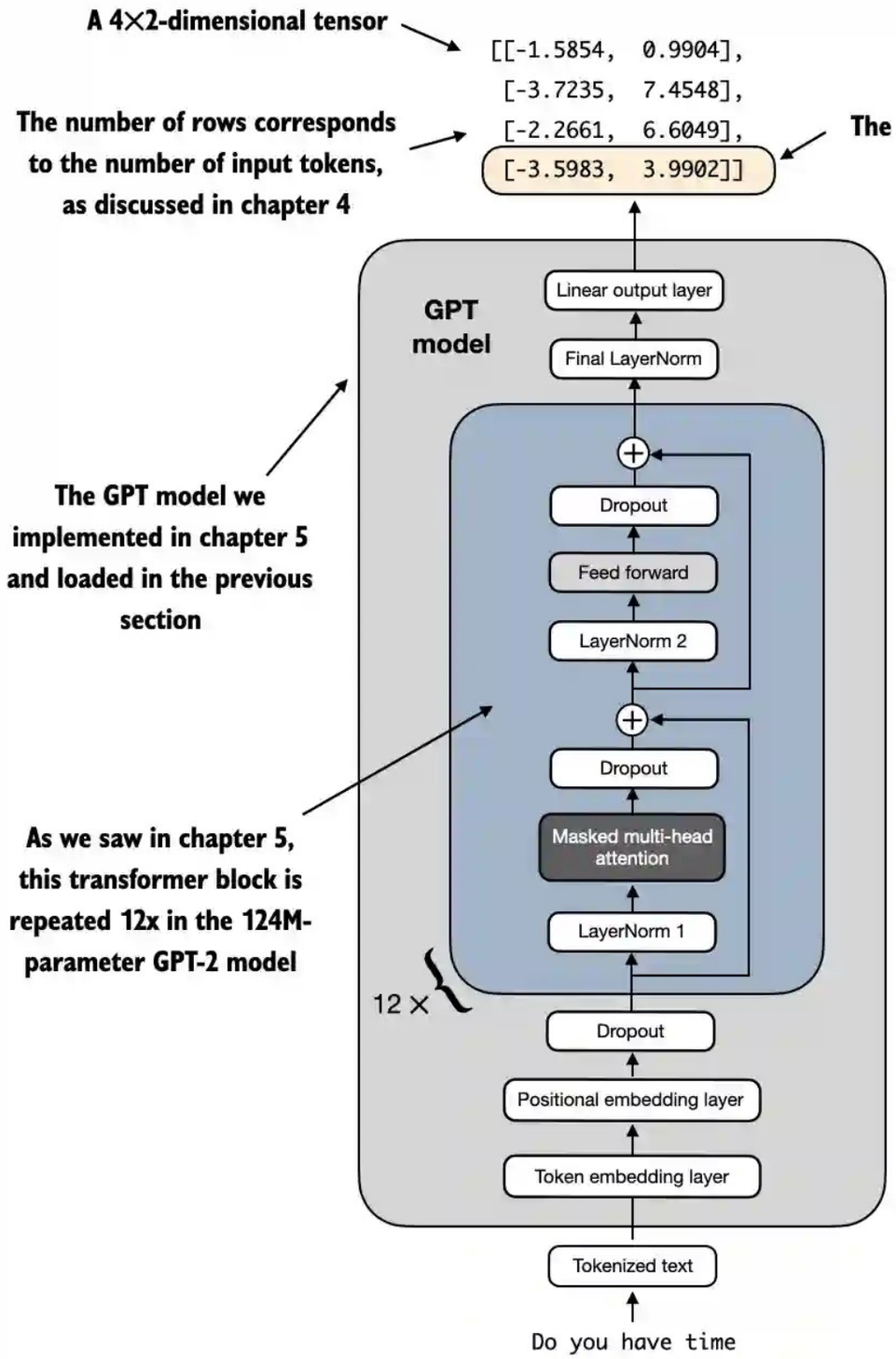

```
for param in model.trf_blocks[-1].parameters():
    param.requires_grad = True
```

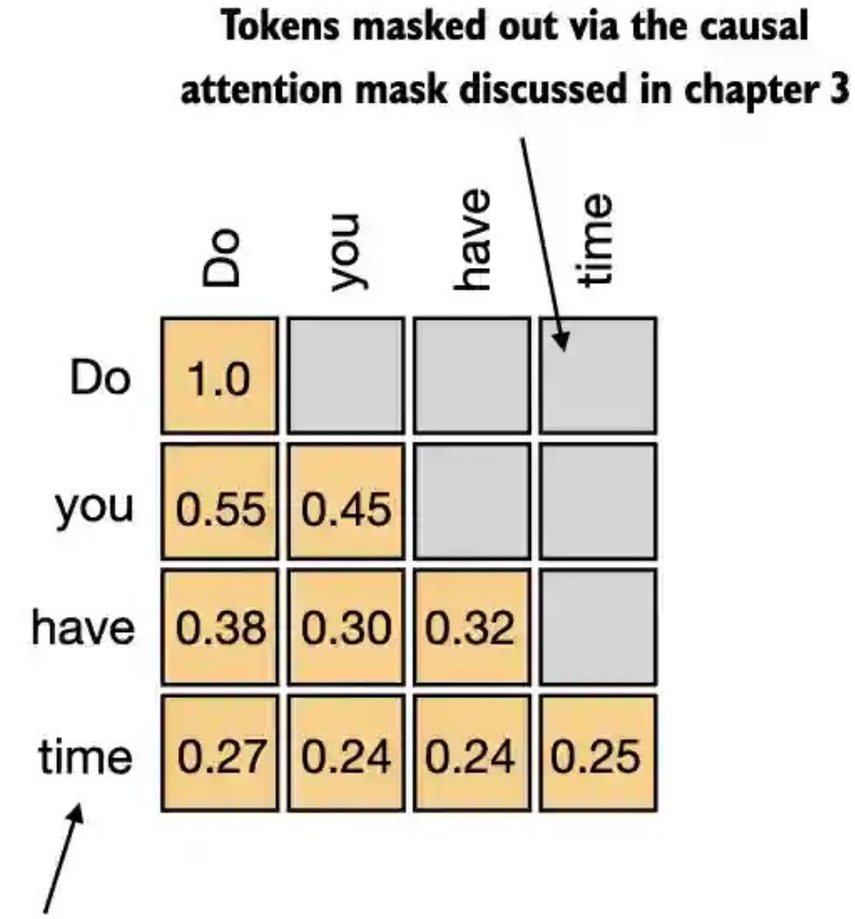
```
for param in model.final_norm.parameters():
    param.requires_grad = True
```

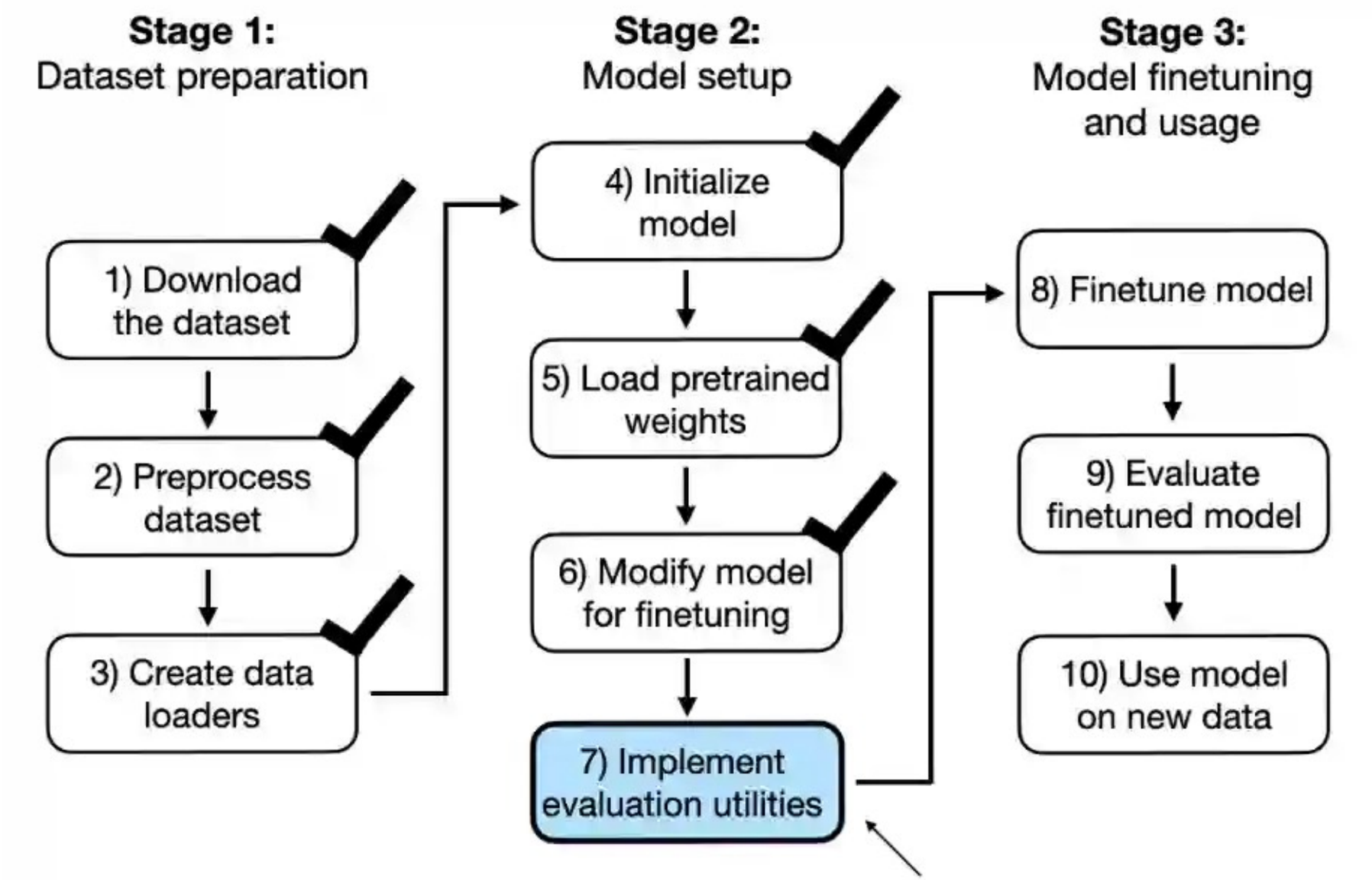
Now Get Two Output

```
inputs = tokenizer.encode("Do you have time")
inputs = torch.tensor(inputs).unsqueeze(0)

 with torch.no_grad():
     outputs = model(inputs)

 print("Outputs:\n", outputs)
 print("Outputs dimensions:", outputs.shape) # shape: (batch_size, num_tokens, num_classes)
```

```
Outputs:
 tensor([[[-1.5854,  0.9904],
         [-3.7235,  7.4548],
         [-2.2661,  6.6049],
         [-3.5983,  3.9902]]])
Outputs dimensions: torch.Size([1, 4, 2])
```


A 4×2-dimensional tensor
[[-1.5854, 0.9904],
[-3.7235, 7.4548],
[-2.2661, 6.6049],
[-3.5983, 3.9902]]
The number of rows corresponds to the number of input tokens, as discussed in chapter 4
The last row corresponds to the last token
GPT model
Linear output layer
Final LayerNorm
Dropout
Feed forward
LayerNorm 2
Dropout
Masked multi-head attention
LayerNorm 1
12 ×
Dropout
Positional embedding layer
Token embedding layer
Tokenized text
Do you have time
The GPT model we implemented in chapter 5 and loaded in the previous section
As we saw in chapter 5, this transformer block is repeated 12x in the 124M-parameter GPT-2 model


Tokens masked out via the causal attention mask discussed in chapter 3
Do
you
have
time
Do 1.0
you 0.55 0.45
have 0.38 0.30 0.32
time 0.27 0.24 0.24 0.25
The last token is the only token with an attention score to all other tokens


Stage 1:
Dataset preparation
Stage 2:
Model setup
Stage 3:
Model finetuning
and usage
1) Download the dataset
2) Preprocess dataset
3) Create data loaders
4) Initialize model
5) Load pretrained weights
6) Modify model for finetuning
7) Implement evaluation utilities
8) Finetune model
9) Evaluate finetuned model
10) Use model on new data
In this section, we implement the utility function to calculate the classification loss and accuracy of the model

Now we Train the model as Normal

```
def calc_loss_batch(input_batch, target_batch, model, device):
    input_batch, target_batch = input_batch.to(device), target_batch.to(device)
    logits = model(input_batch)[:, -1, :]  # Logits of last output token
    loss = torch.nn.functional.cross_entropy(logits, target_batch)
    return loss
```

```
# Same as in chapter 5
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
```

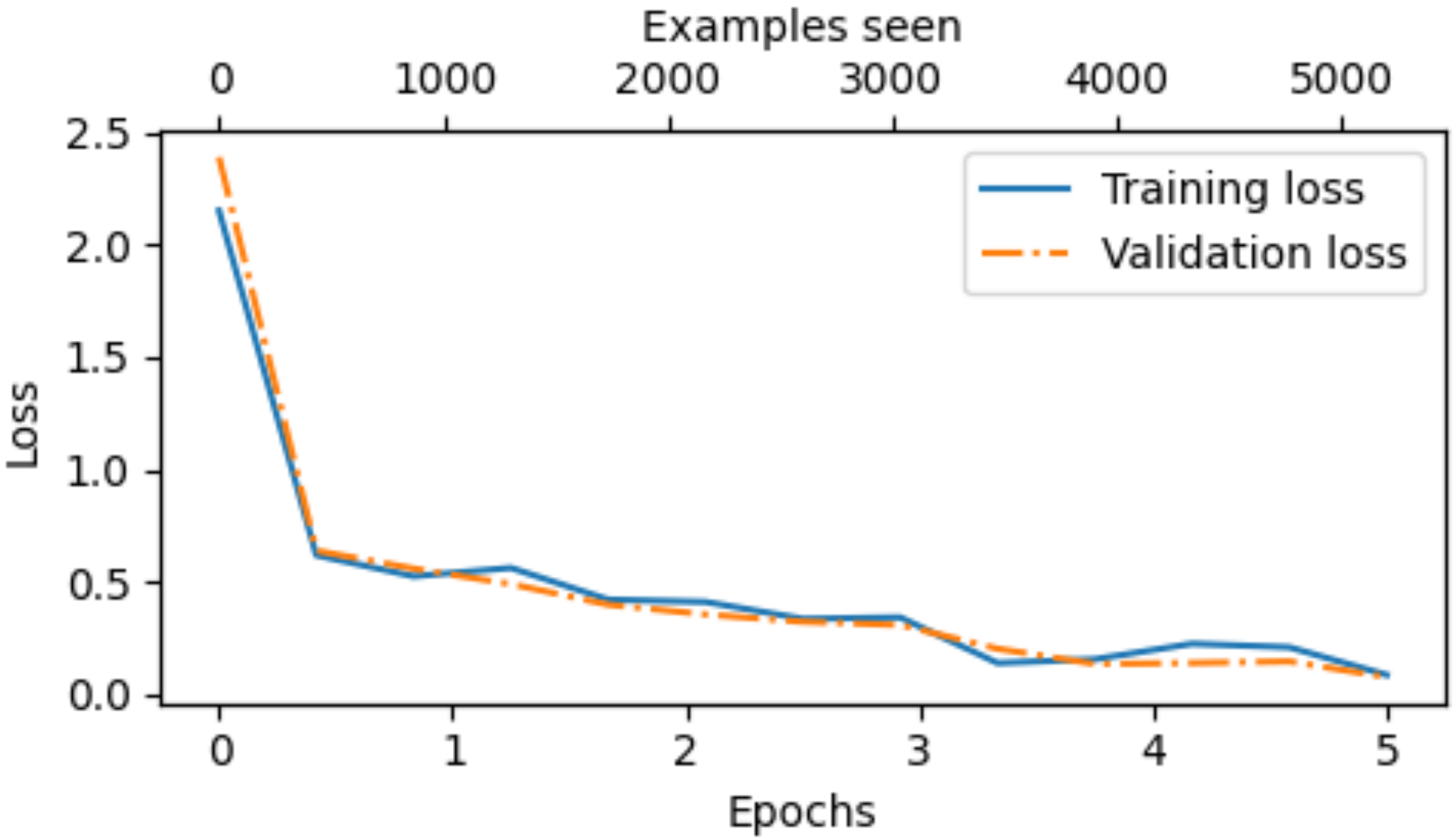
```
with torch.no_grad(): # Disable gradient tracking for efficiency because we are not training, yet
    train_loss = calc_loss_loader(train_loader, model, device, num_batches=5)
    val_loss = calc_loss_loader(val_loader, model, device, num_batches=5)
    test_loss = calc_loss_loader(test_loader, model, device, num_batches=5)

print(f"Training loss: {train_loss:.3f}")
print(f"Validation loss: {val_loss:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

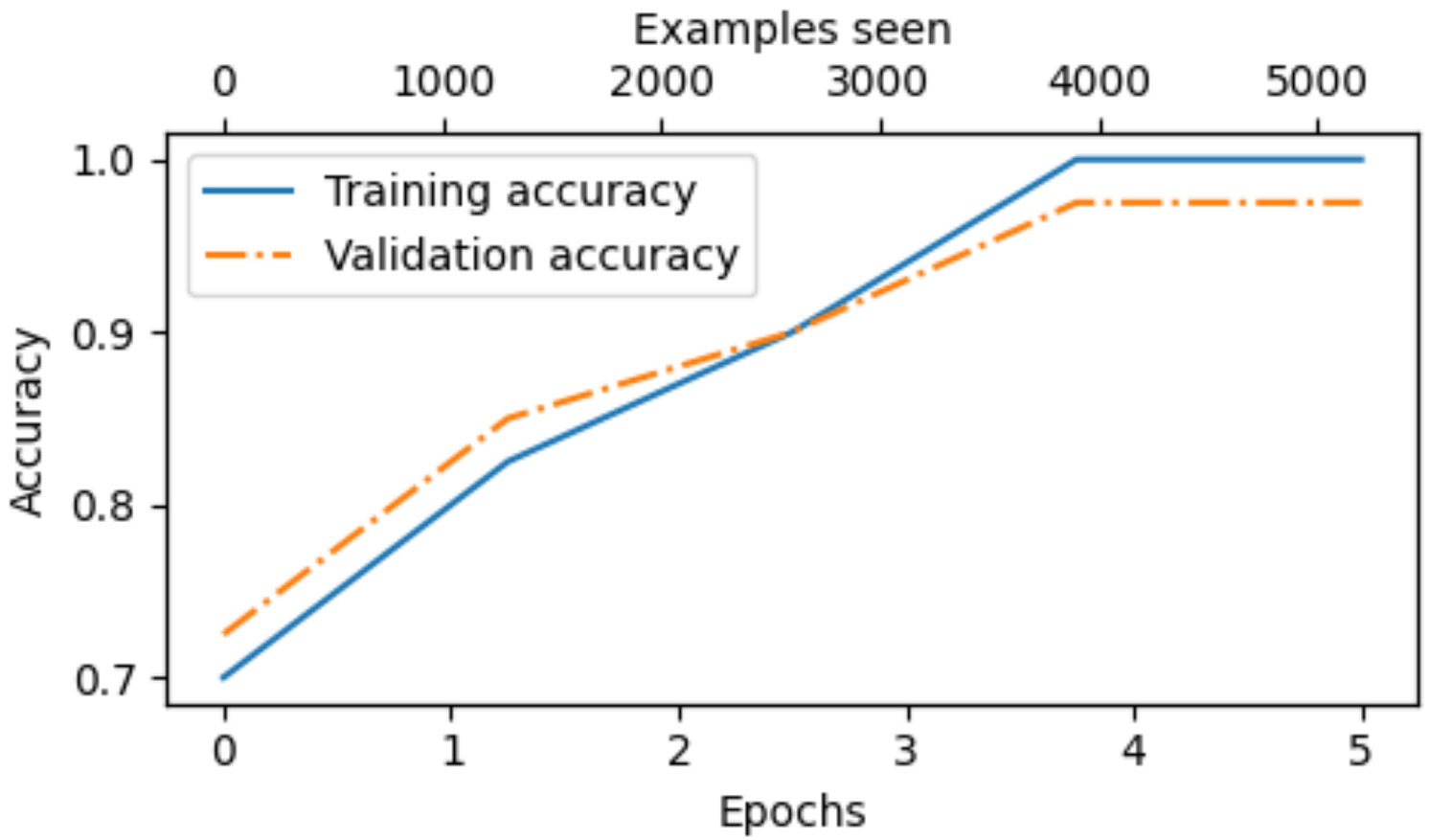
```
Training loss: 2.453
Validation loss: 2.583
Test loss: 2.322
```

```
# Overall the same as `train_model_simple` in chapter 5
def train_classifier_simple(model, train_loader, val_loader, optimizer, device, num_epochs,
                            eval_freq, eval_iter):
    # Initialize lists to track losses and examples seen
    train_losses, val_losses, train_accs, val_accs = [], [], [], []
    examples_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
            examples_seen += input_batch.shape[0] # New: track examples instead of tokens
            global_step += 1

            # Optional evaluation step
            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)
                print(f"Ep {epoch+1} (Step {global_step:06d}): "
                      f"Train loss {train_loss:.3f}, Val loss {val_loss:.3f}")

        # Calculate accuracy after each epoch
        train_accuracy = calc_accuracy_loader(train_loader, model, device, num_batches=eval_iter)
        val_accuracy = calc_accuracy_loader(val_loader, model, device, num_batches=eval_iter)
        print(f"Training accuracy: {train_accuracy*100:.2f}% | ", end="")
        print(f"Validation accuracy: {val_accuracy*100:.2f}%")
        train_accs.append(train_accuracy)
        val_accs.append(val_accuracy)

    return train_losses, val_losses, train_accs, val_accs, examples_seen
```

```
# Same as chapter 5
def evaluate_model(model, train_loader, val_loader, device, eval_iter):
    model.eval()
    with torch.no_grad():
        train_loss = calc_loss_loader(train_loader, model, device, num_batches=eval_iter)
        val_loss = calc_loss_loader(val_loader, model, device, num_batches=eval_iter)
    model.train()
    return train_loss, val_loss
```

```
import time

start_time = time.time()

torch.manual_seed(123)

optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5, weight_decay=0.1)

num_epochs = 5
train_losses, val_losses, train_accs, val_accs, examples_seen = train_classifier_simple(
    model, train_loader, val_loader, optimizer, device,
    num_epochs=num_epochs, eval_freq=50, eval_iter=5,
)

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


Examples seen
0
1000
2000
3000
4000
5000
2.5
2.0
1.5
1.0
0.5
0.0
Loss
Training loss
Validation loss
0
1
2
3
4
5
Epochs

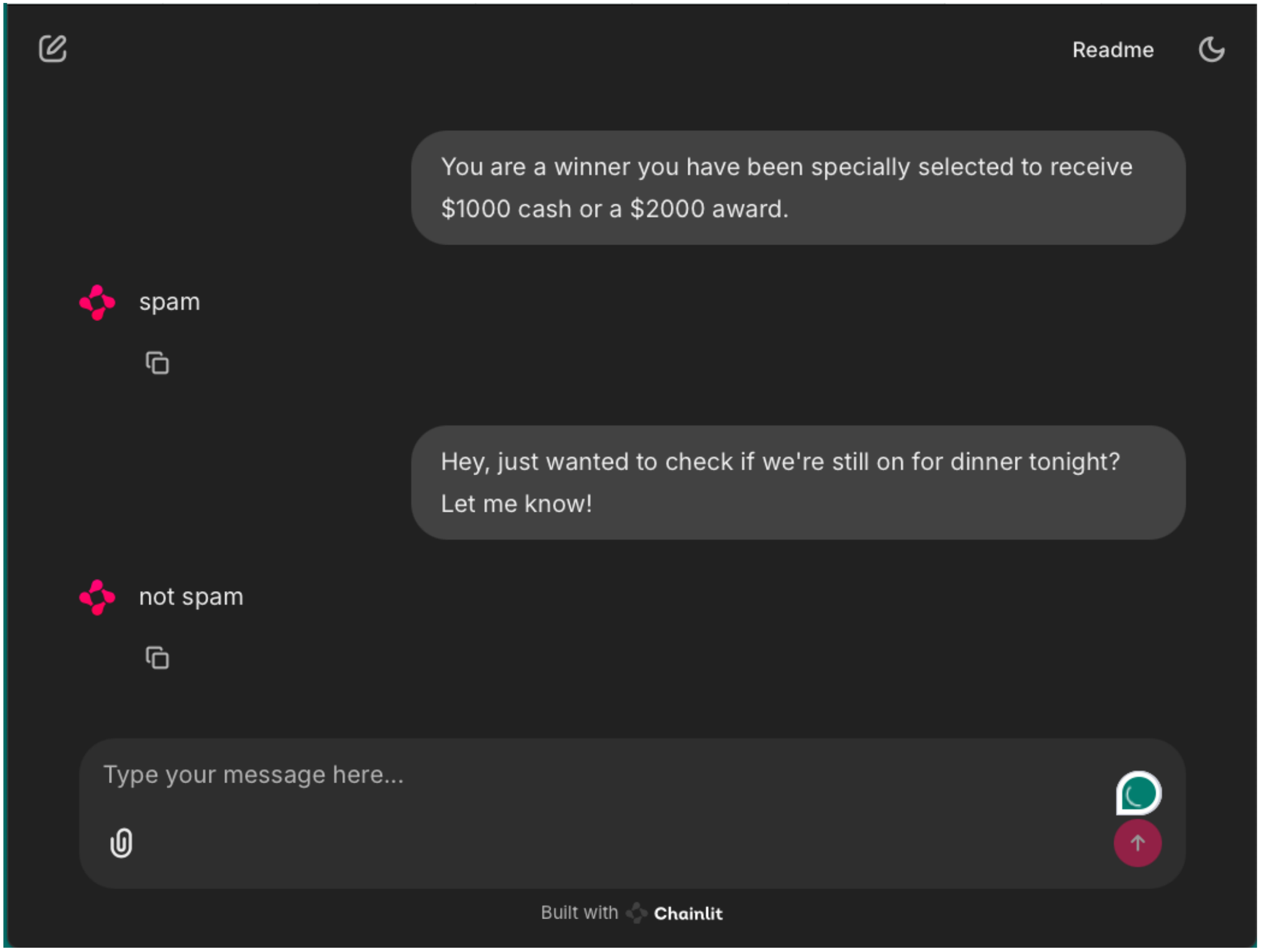

Examples seen
0
1000
2000
3000
4000
5000
1.0
0.9
0.8
0.7
Accuracy
Training accuracy
Validation accuracy
0
1
2
3
4
5
Epochs

Experiments in Fine Tuning

gpt2-small (124M)
longest train ex. (120)
CPU/GPU A100

Trainable token position	Trainable layers	Training acc	Validation acc	Test acc	Training time
last	last_block	96.63%	99.33%	95.00%	0.28 min
first	last_block	78.46%	80.54%	75.00%	0.28 min
last	last_layer	78.65%	79.87%	72.00%	0.25 min
last	last_two_blocks	98.85%	98.66%	98.33%	0.33 min
last	all	99.62%	96.64%	96.67%	0.69 min

Chainlit Webpage of Finetuned Model



Chainlit Webpage of Finetuned Model

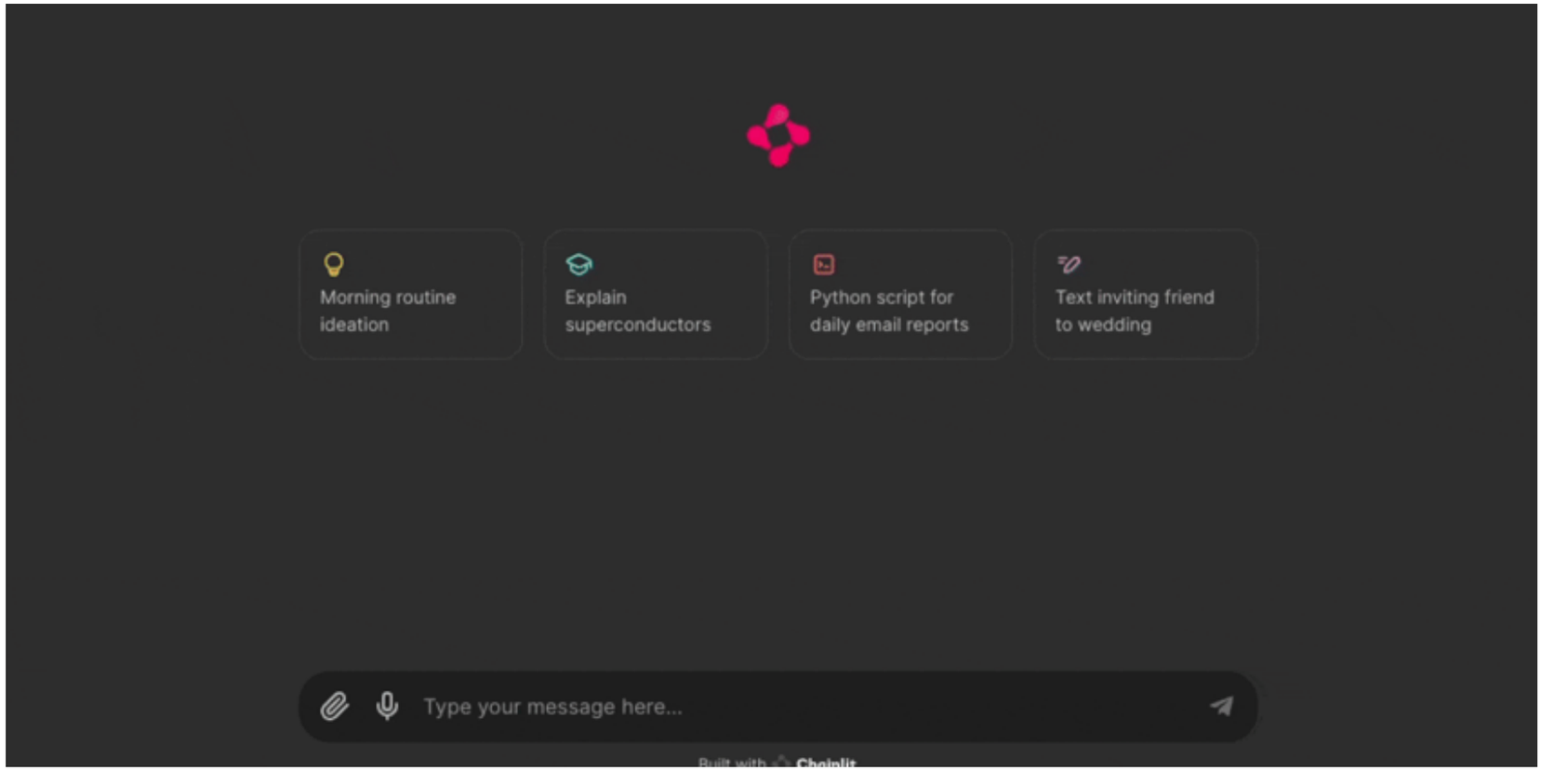
64 line software code to load Model & Tokenizer

```
@chainlit.on_message
async def main(message: chainlit.Message):
    """
    The main Chainlit function.
    """
user_input = message.content

label = classify_review(user_input, model, tokenizer, device, max_length=120)

await chainlit.Message(
    content=f"{label}",  # This returns the model response to the interface
).send()
```

Chainlit

Python package

Create conversational AI applications like ChatGPT



Frameworks for LLM Apps

Fast API
- Building API in Python

LangChain
- Framework for applications using LLM
- Connect to databases
- API

LiteLLM
- Consistent API for accessing LLMs

Haystack
- Ingest data from different sources, including PDFs, text files, databases, and APIs
- RAG

Embedchain
- RAG
- Indexes your data, making it searchable

Other Types of Fine Tuning

In-Context Learning

Feature-Based Approach

Parameter-Efficient Finetuning

In-Context Learning

In-context Learning (ICL) is a method where Large Language Models (LLMs) learn tasks by using a few examples embedded directly in the prompt, known as “context.”

```
Translate the following German sentences into English:

Example 1:
German: "Ich liebe Eis."
English: "I love ice cream."

Example 2:
German: "Draußen ist es stürmisch und regnerisch"
English: "It's stormy and rainy outside."

Translate this sentence:
German: "Wo ist die naechste Supermarkt?"
```

Few-shot prompt

Your job is to classify sentiment of product reviews. Classify the following review:

The product works great! // Positive

The product doesn’t work for our use case // Negative

The product is alright I guess // Neutral

The product doesn't satisfy our requirements //

Use high-quality examples

Varied examples

Focus on formatting:

Consistency in your examples is key.

Order matters:

Simple to complex.

Most relevant examples closest to the query

Avoid example clustering:

Your examples should essentially be randomly ordered

Example label distribution:

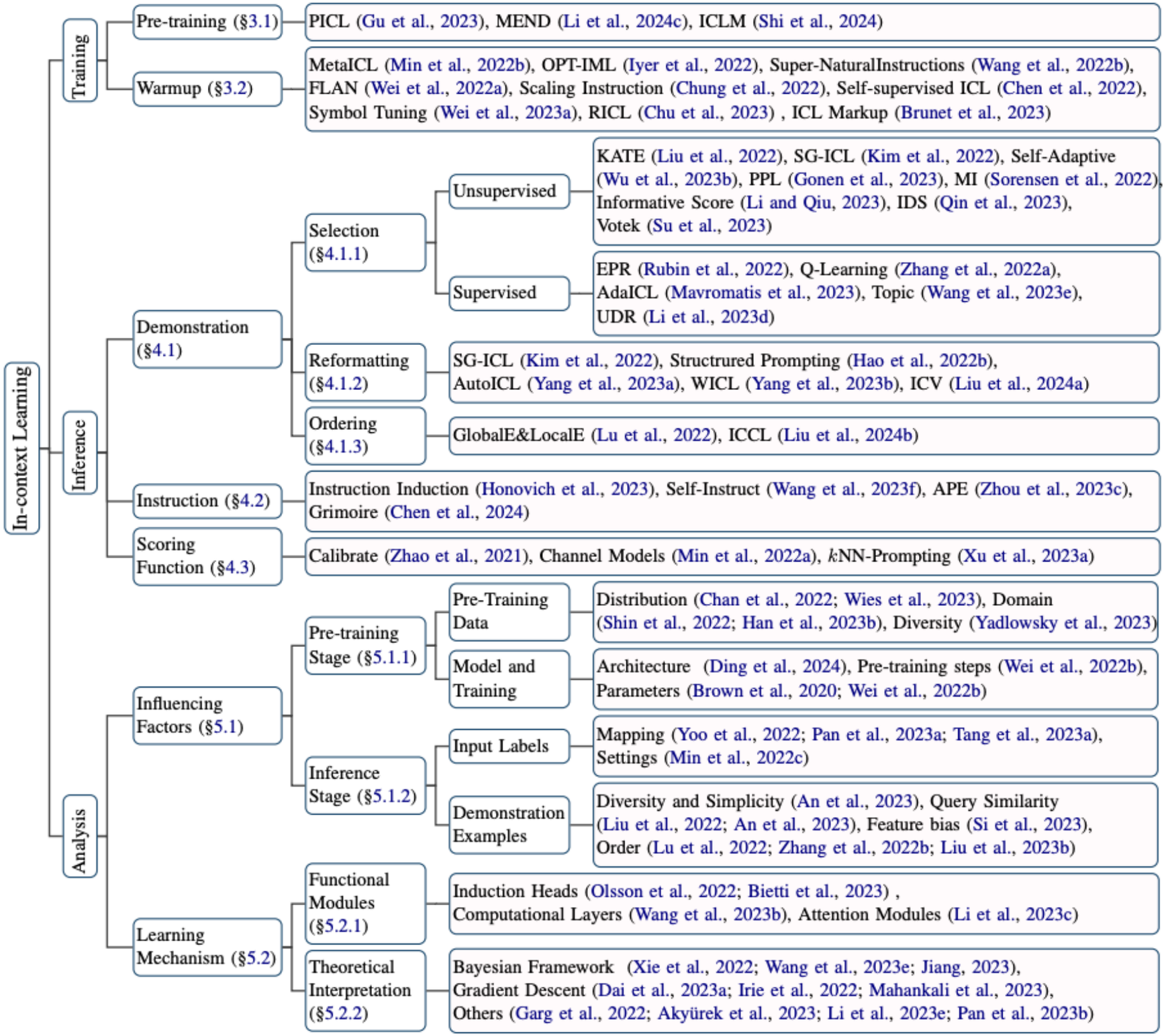
Use a balanced distribution of examples

Don’t use too many examples:

Stay under 8 to start.

In Context Learning Guide

https://www.prompthub.us/blog/in-context-learning-guide



A Survey on In-context Learning, Oct 2024, arXiv:2301.00234v6

Models behave Differently to Different Methods

Model	Method	SST5	SST2	CQA	SNLI	News	Avg
GPT2	topk	40.1	74.9	30.2	39.7	62.7	49.5
	votek	32.4	51.0	29.8	35.8	25.5	34.9
	mdl	**43.3**	**86.7**	**32.7**	**41.4**	**68.0**	**54.4**
GPT-J	topk	**46.9**	84.6	58.4	**60.7**	**69.1**	**63.9**
	votek	33.8	87.3	63.4	43.1	25.3	50.6
	mdl	37.6	**87.9**	**64.1**	59.8	68.2	63.5
Qwen2	topk	54.1	83.3	76.3	**68.2**	64.9	**69.4**
	votek	**55.3**	**86.9**	76.1	51.6	**65.3**	67.0
	mdl	54.6	86.1	**77.1**	65.0	63.2	69.2
Llama3	topk	53.0	**90.3**	76.1	**64.0**	74.0	**71.5**
	votek	54.9	88.9	72.6	57.7	**78.3**	70.5
	mdl	**54.4**	89.1	**76.5**	59.9	74.6	70.9

SST Stanford Sentiment Treebank

A Survey on In-context Learning, Oct 2024, arXiv:2301.00234v6

Top-K

The "k" most similar or relevant examples from a dataset, based on a similarity metric

Distance between the embeddings of the input query and the potential examples

Vote-K

A graph-based technique designed to select diverse and representative examples from unlabeled datasets for few-shot prompts

Uses a graph structure

- Represent the relationships between data points,
- Identify of diverse and representative examples.1

“On average, vote-k achieves a 12.9%/11.4% relative gain under an annotation budget of 18/100, as compared to randomly selecting examples to annotate”

Selective Annotation Makes Language Models Better Few-Shot Learners

arXiv:2209.01975

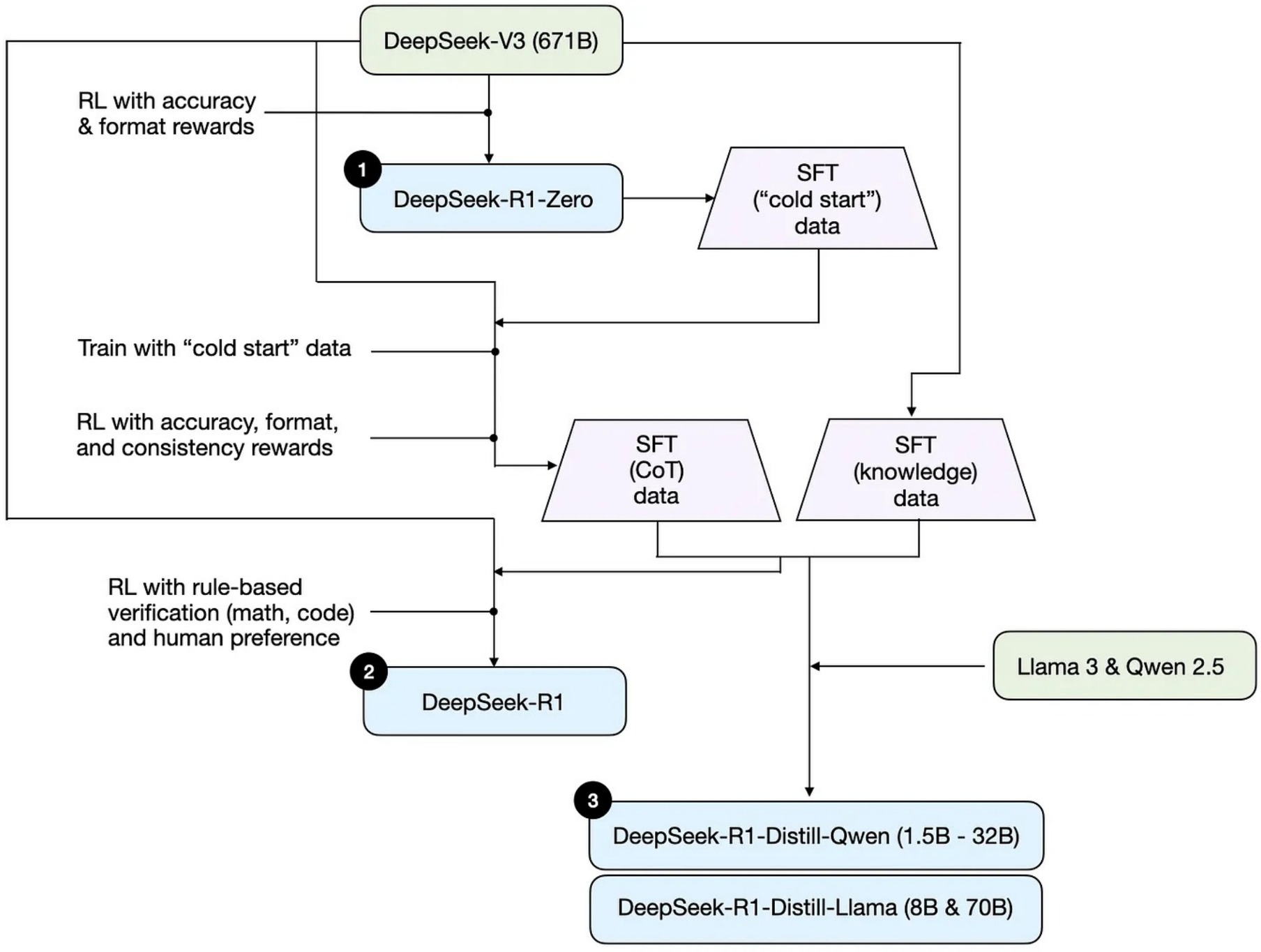
Minimum Description Length (MDL)

Give the most informative and efficient "description" of the task, using the fewest and most relevant examples

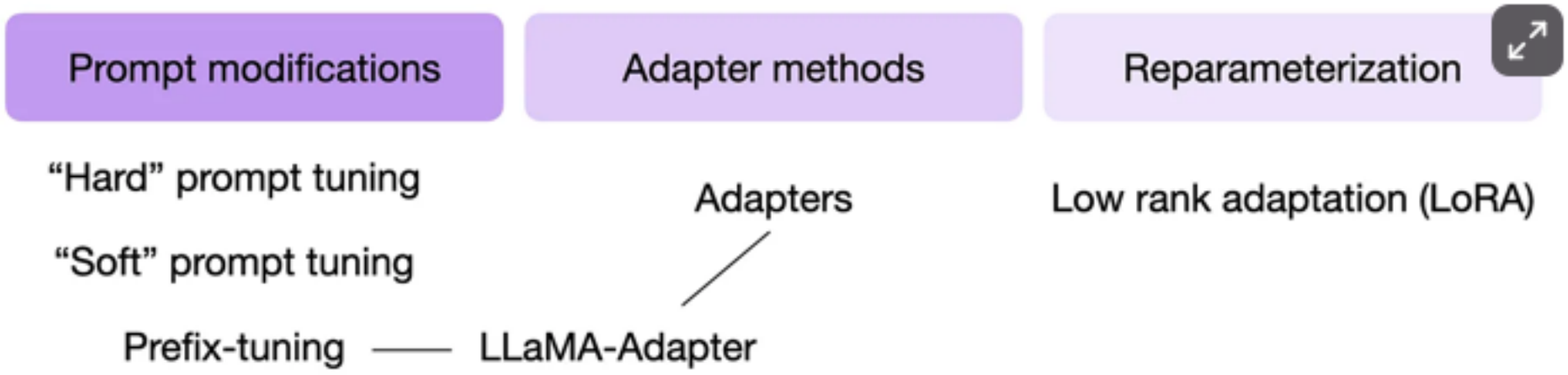
Identify examples that:

- Cover the most critical aspects of the task.
- Minimize ambiguity and noise.
- Lead to the most accurate and consistent predictions.

DeepSeek


DeepSeek-V3 (671B)
RL with accuracy
& format rewards
1
DeepSeek-R1-Zero
SFT
(“cold start”)
data
Train with “cold start” data
RL with accuracy, format,
and consistency rewards
SFT
(CoT)
data
SFT
(knowledge)
data
RL with rule-based
verification (math, code)
and human preference
2
DeepSeek-R1
Llama 3 & Qwen 2.5
3
DeepSeek-R1-Distill-Qwen (1.5B - 32B)
DeepSeek-R1-Distill-Llama (8B & 70B)

Parameter-Efficient Finetuning



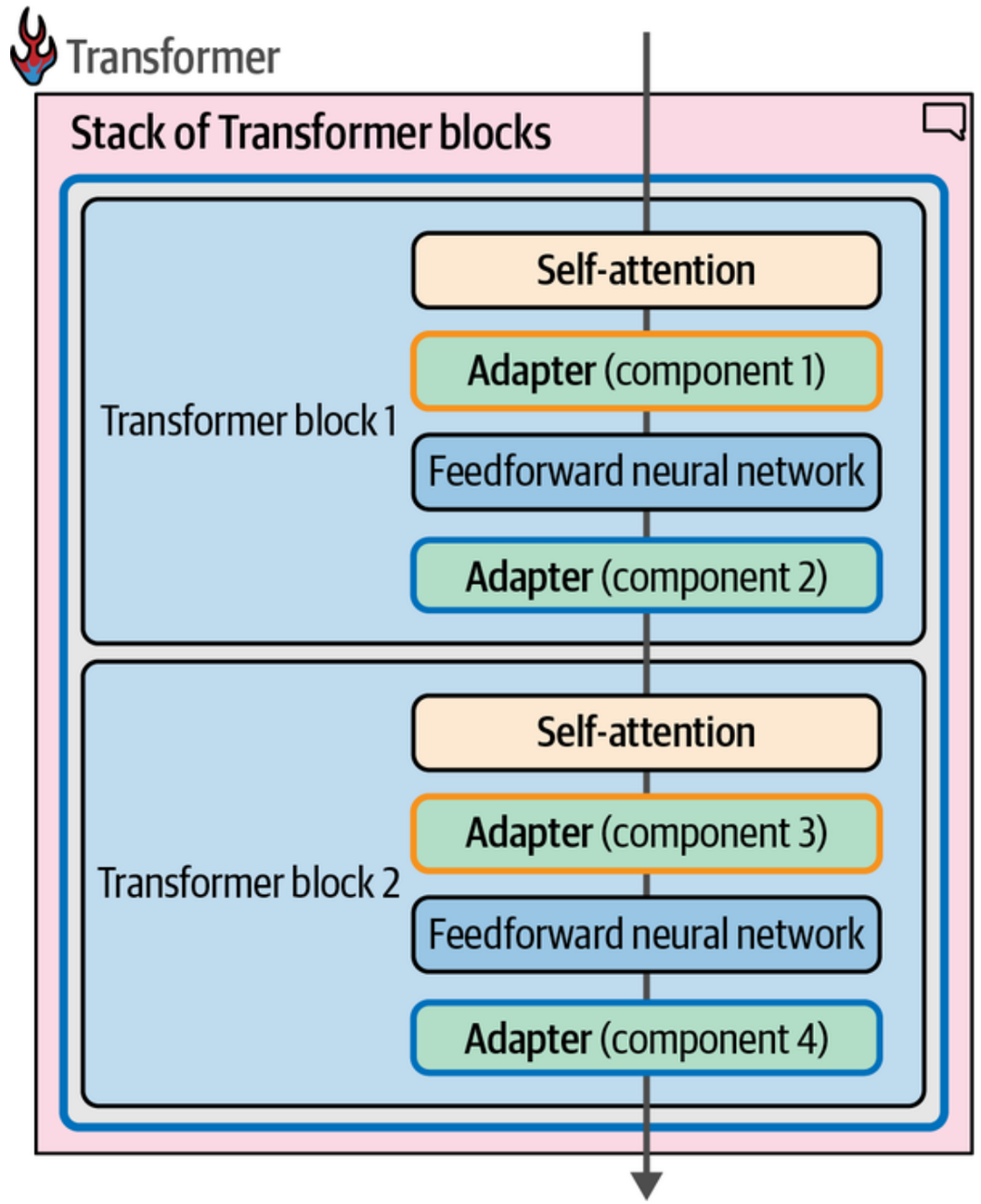
Low-Rank Adaptation (LoRA):

Adapter Layers:

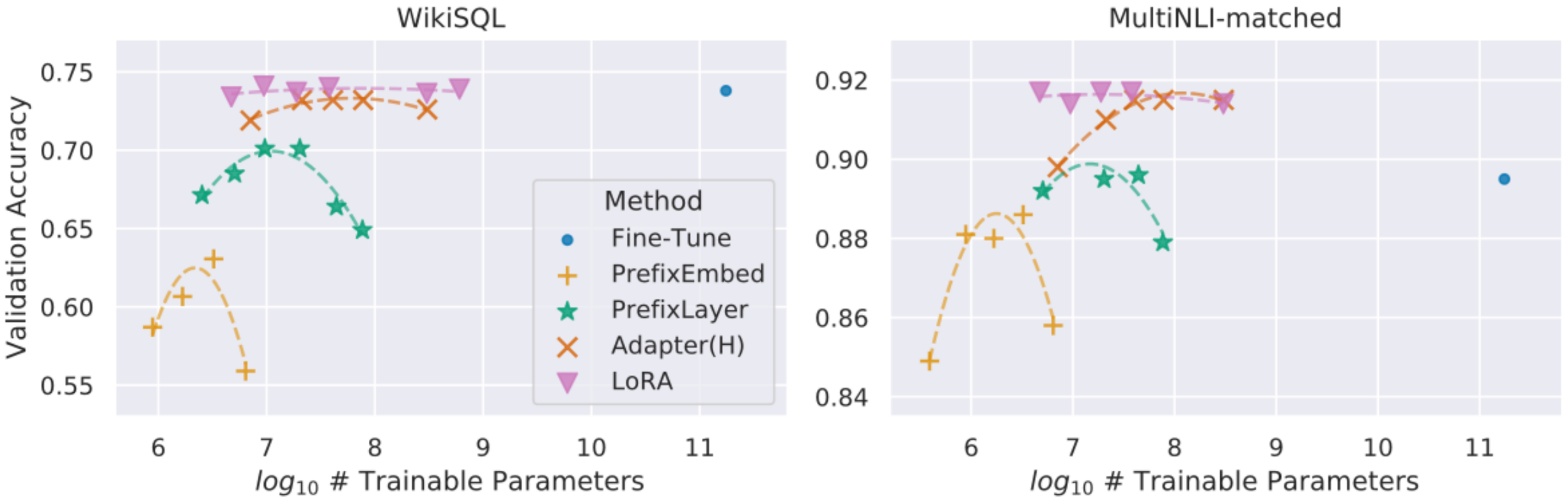
Prompt Tuning:

Prefix Tuning:

QLoRA (Quantized Low-Rank Adapation):


Transformer
Stack of Transformer blocks
Transformer block 1
Self-attention
Adapter (component 1)
Feedforward neural network
Adapter (component 2)
Transformer block 2
Self-attention
Adapter (component 3)
Feedforward neural network
Adapter (component 4)

LoRA - Low-Rank Adaptation Of LLMs



LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

Matrix Rank

Maximum number of linearly independent rows or columns

A	B	C
0	I	I
I	3	5
3	2	8

C = 2* A + B

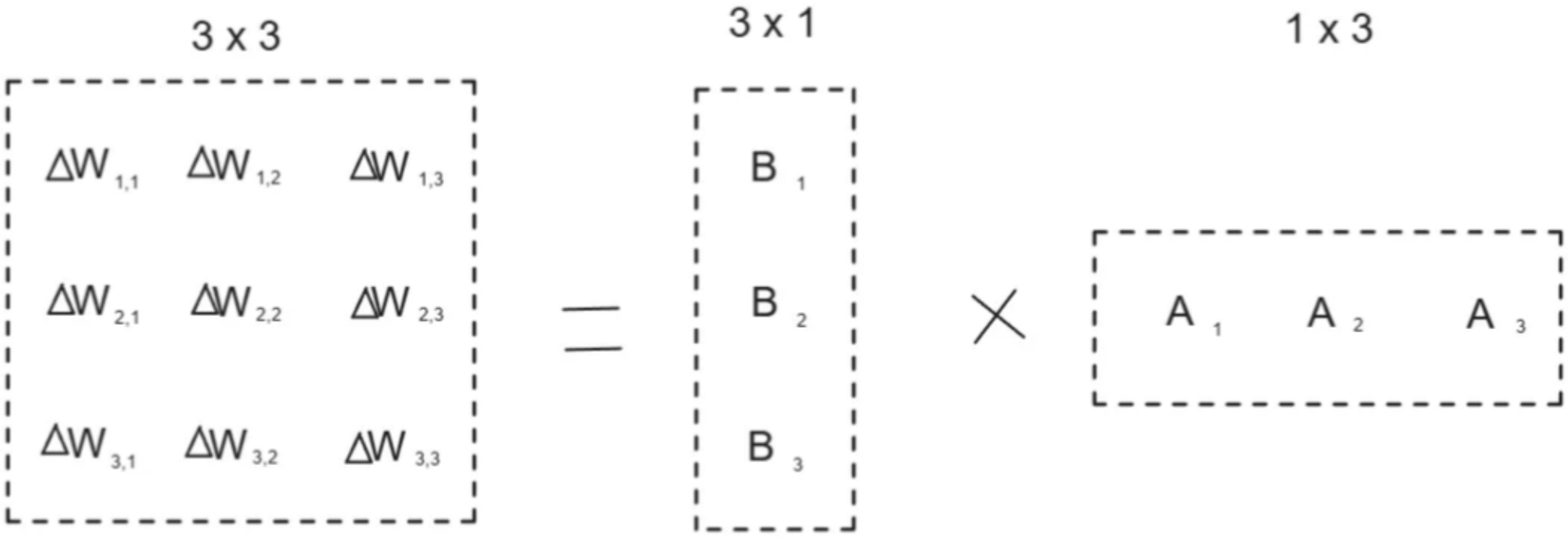
Rank 2

A	B
0	I
I	3
3	2

Basic Idea for LoRA

If the matrix has rank r, we can use a smaller matrix without losing much information

“When adapting to a specific task, Aghajanyan et al.(2020) shows that the pre-trained language models have a low intrinsic dimension”



LoRA Explained: Low-Rank Adaptation for Fine-Tuning LLMs, Zilliz

https://medium.com/@zilliz_learn/lora-explained-low-rank-adaptation-for-fine-tuning-llms-066c9bdd0b32

LoRA

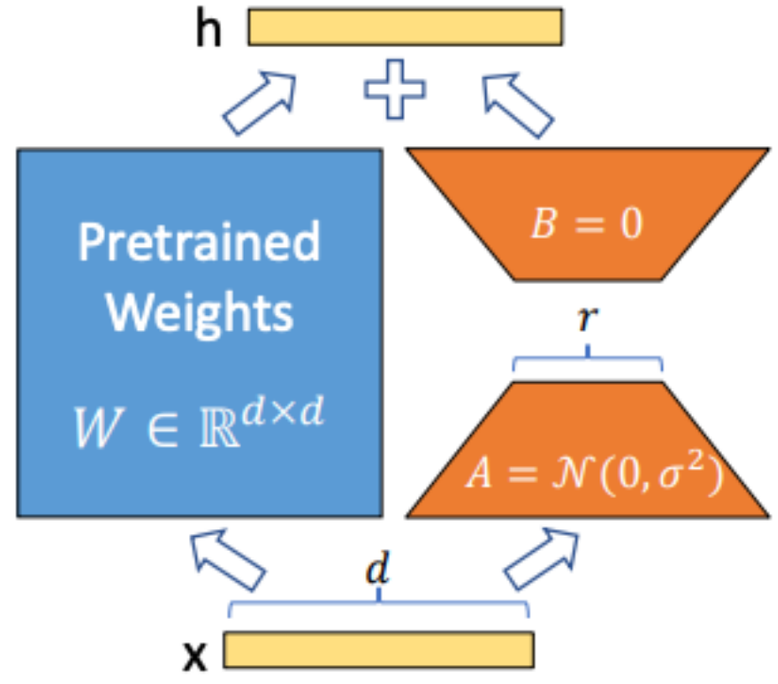
Freeze model weights

At each layer of the Transformer, add two matrices

Dimension dxr and rxd

Train on your fine-tune data

r is a hyperparameter



Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} ($Adpt^D$)*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} ($Adpt^D$)*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
RoB_{large} (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB_{large} (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
RoB_{large} ($Adpt^P$)†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
RoB_{large} ($Adpt^P$)†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
RoB_{large} ($Adpt^H$)†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
RoB_{large} ($Adpt^H$)†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
RoB_{large} (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

RoBERTabase, RoBERTalarge, and DeBERTaXXL

LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

How do You Add Matrices to an existing Model?

PEFT

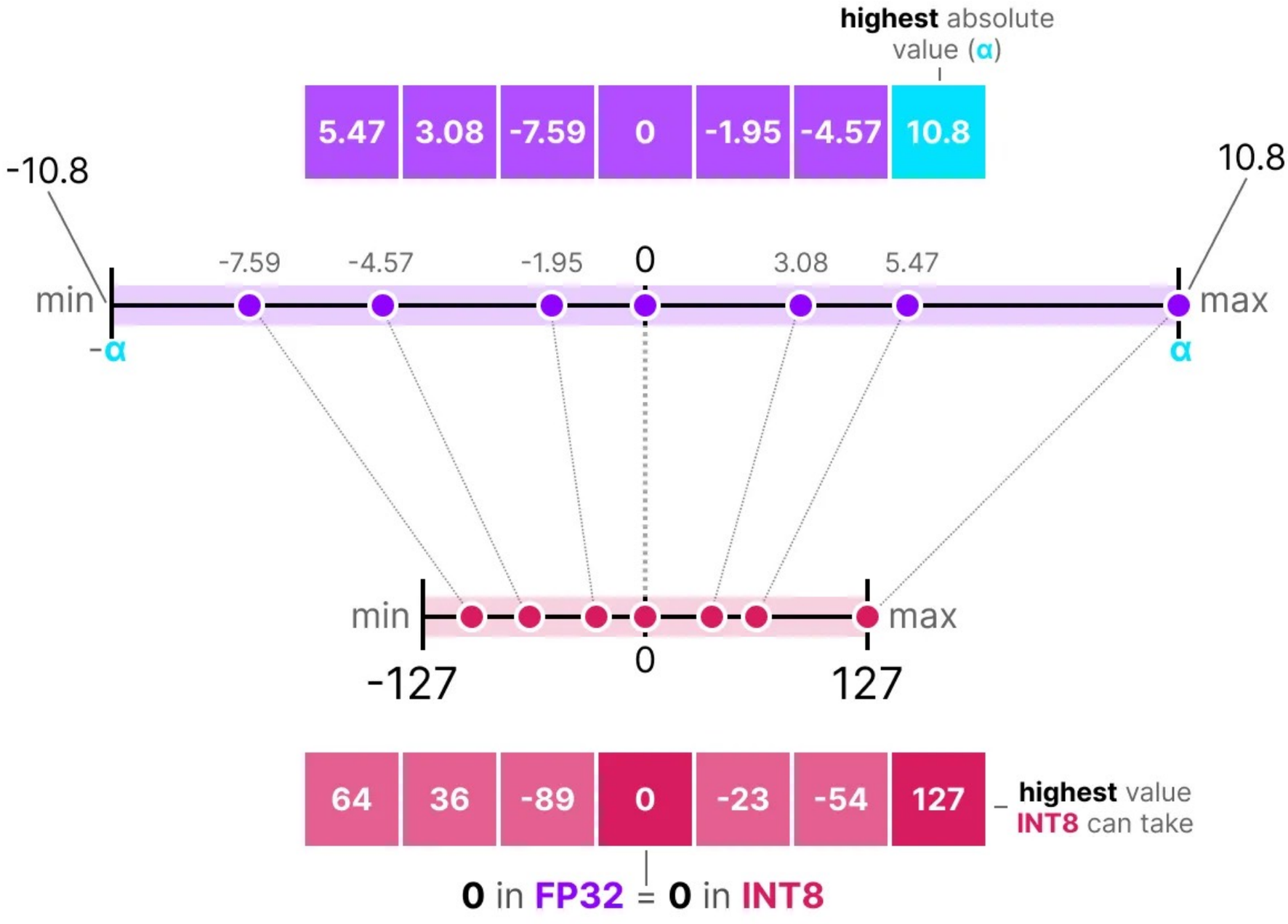
Python package from Huggingface

- PROMPT_TUNING
- MULTITASK_PROMPT_TUNING
- P_TUNING
- PREFIX_TUNING
- LORA
- ADALORA
- BOFT
- ADAPTION_PROMPT
- IA3
- LOHA
- LOKR
- OFT
- XLORA
- POLY
- LN_TUNING
- VERA
- FOURIERFT
- HRA
- BONE

QLoRA

Use LoRA on quantized LLM

Symmetric Quantization



Symmetric Quantization - The Math

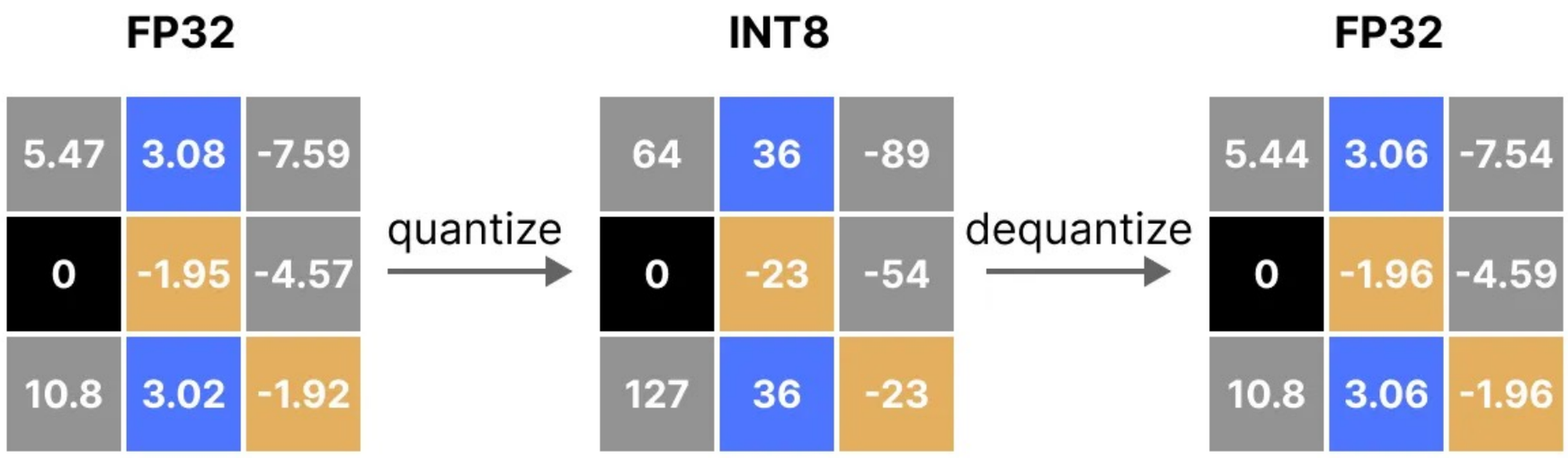
b number of bytes to quantize to

a highest absolute value

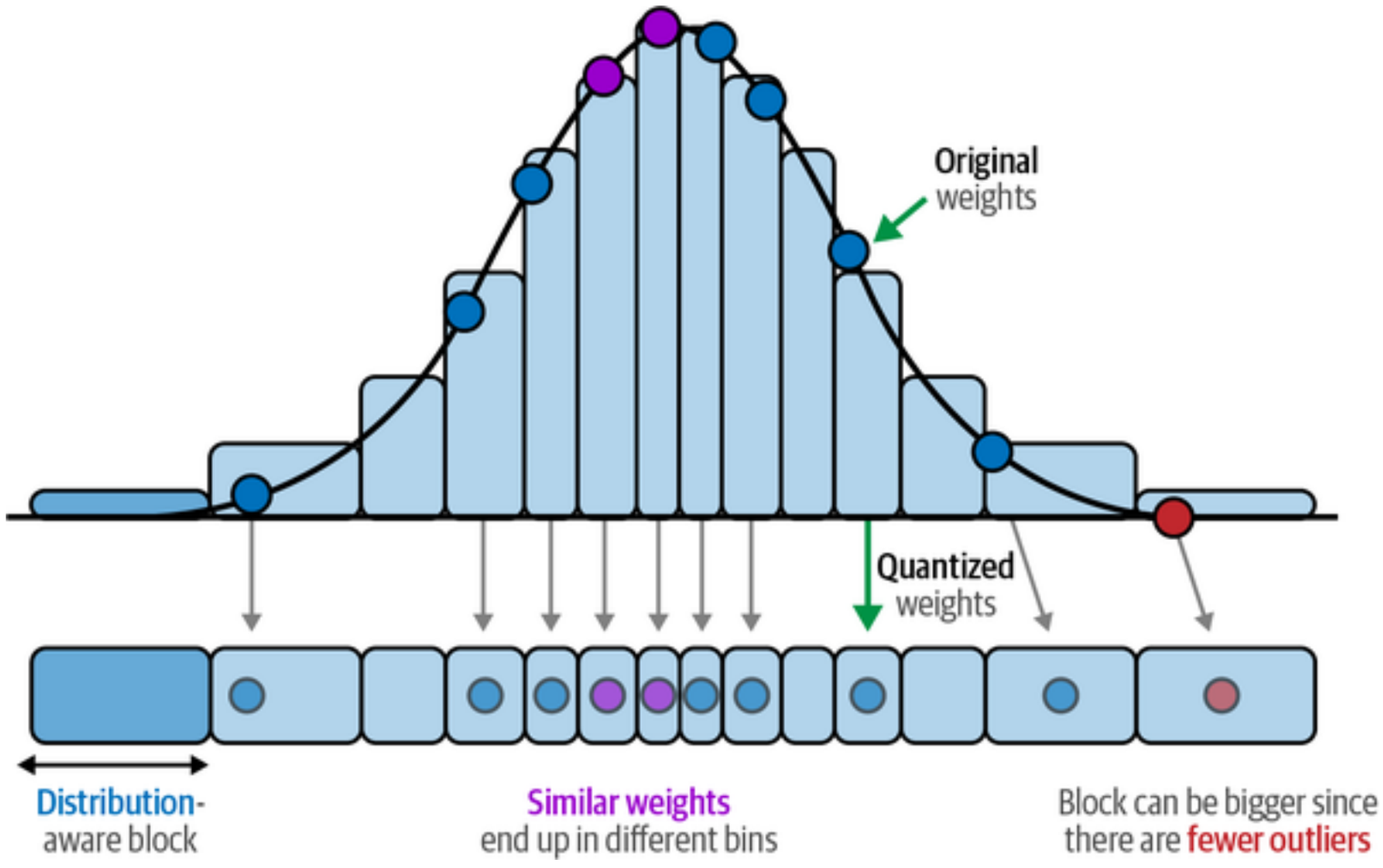
$$s = \frac{2^{b-1} - 1}{a}$$

$$x_{quantized} = round(s \, . \, x)$$

$$x_{dequantized} = \frac{x_{quantized}}{s}$$

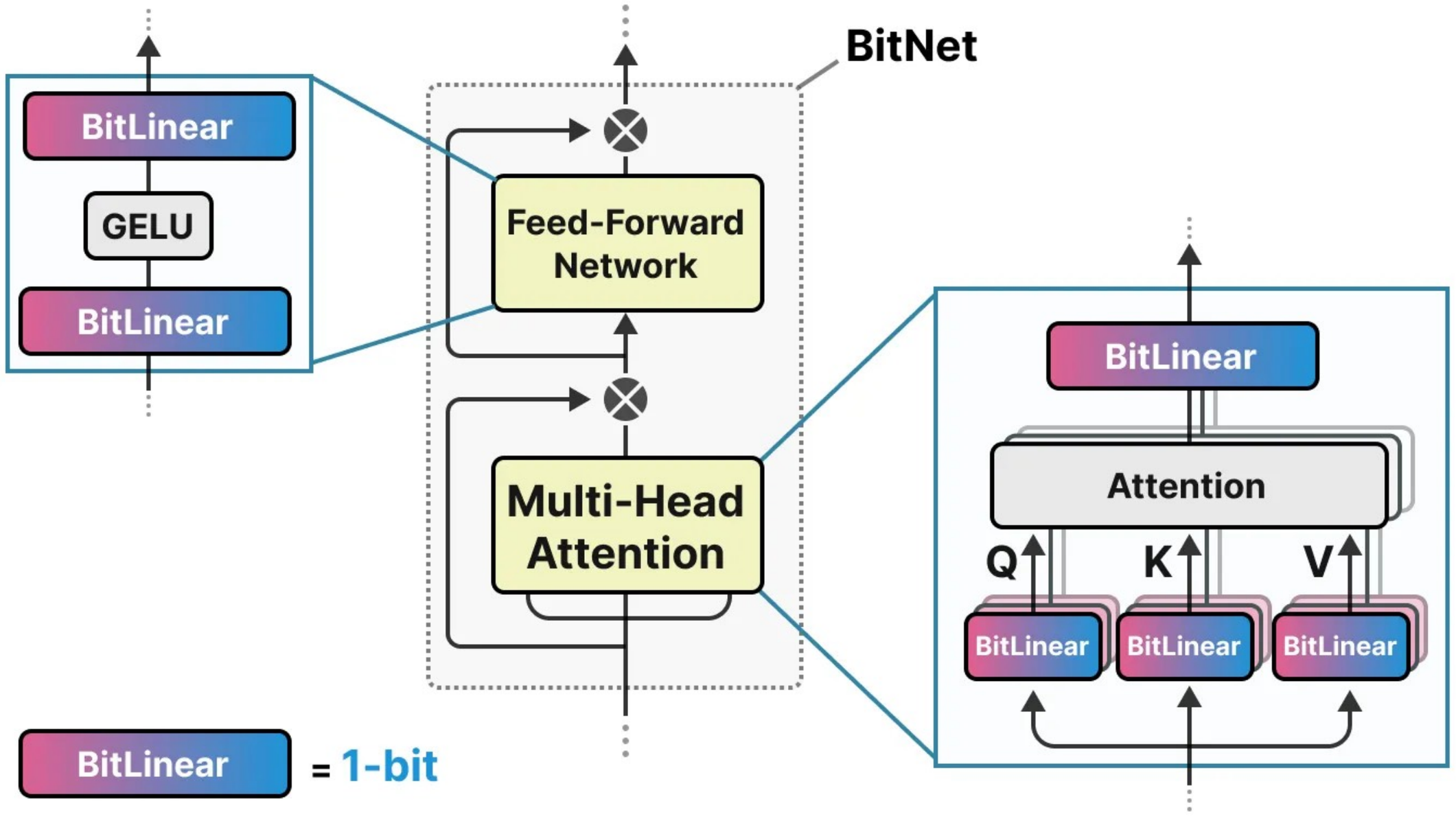


NF4 (NormalFloat4)

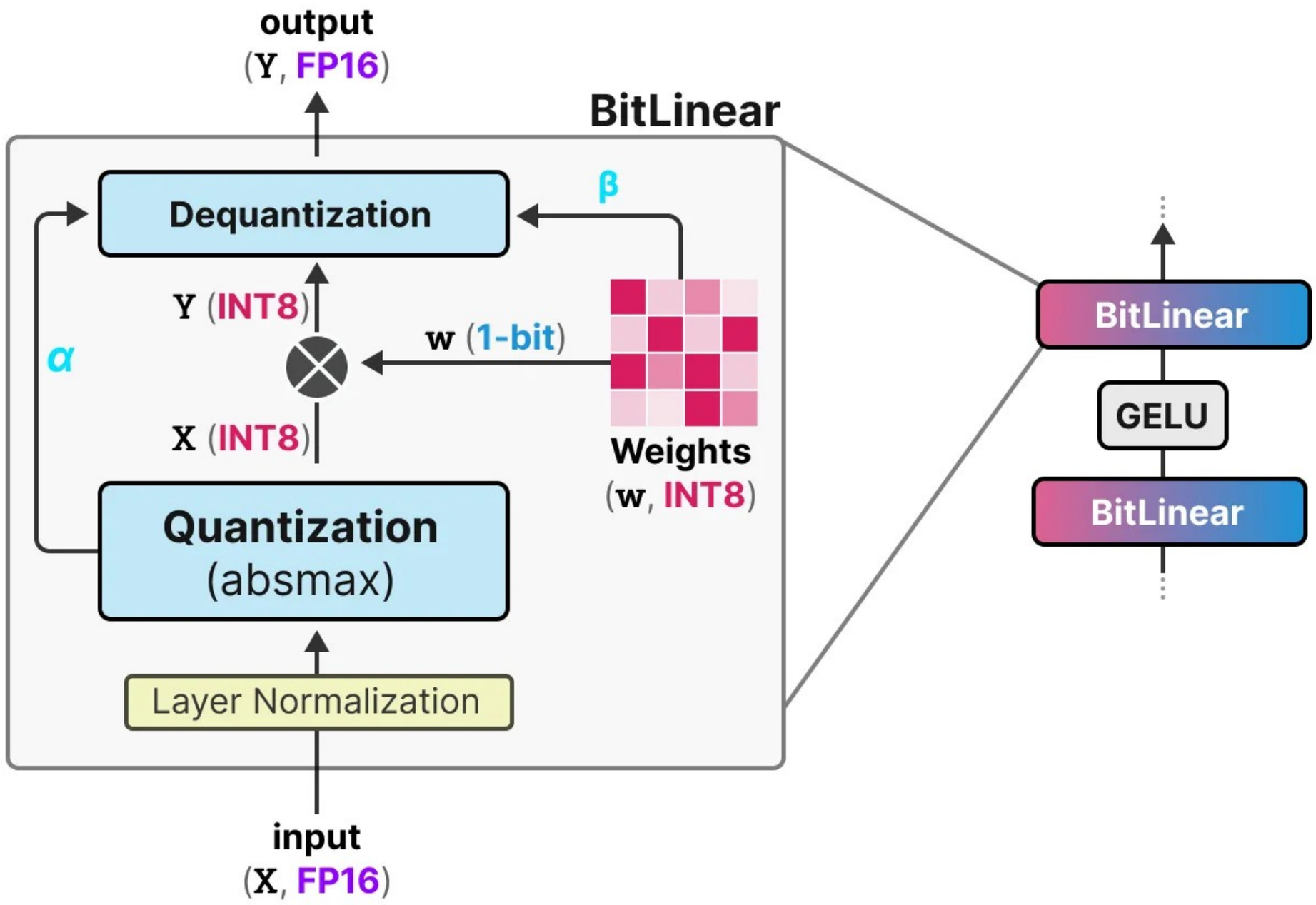


The Era of 1-bit LLMs: BitNet

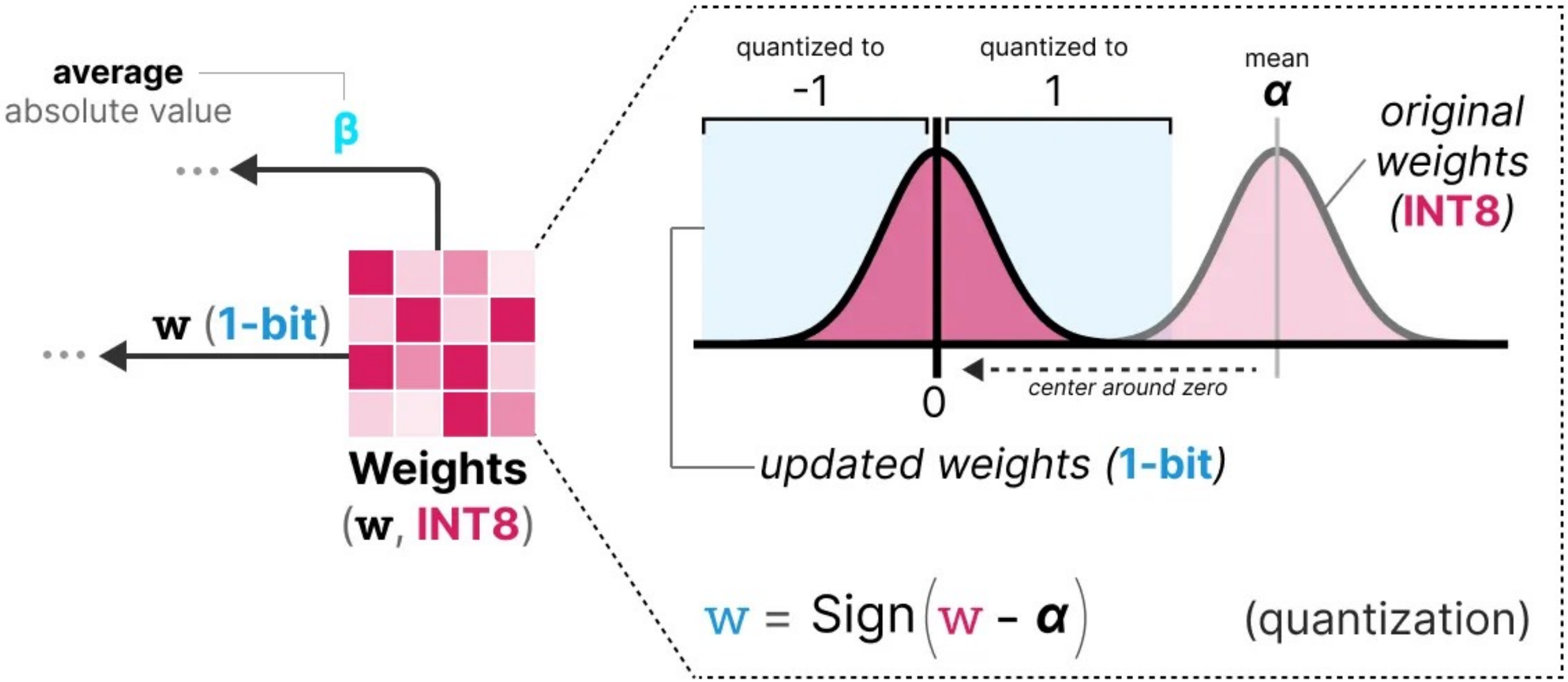
Uses either -1 or 1 for a given weight



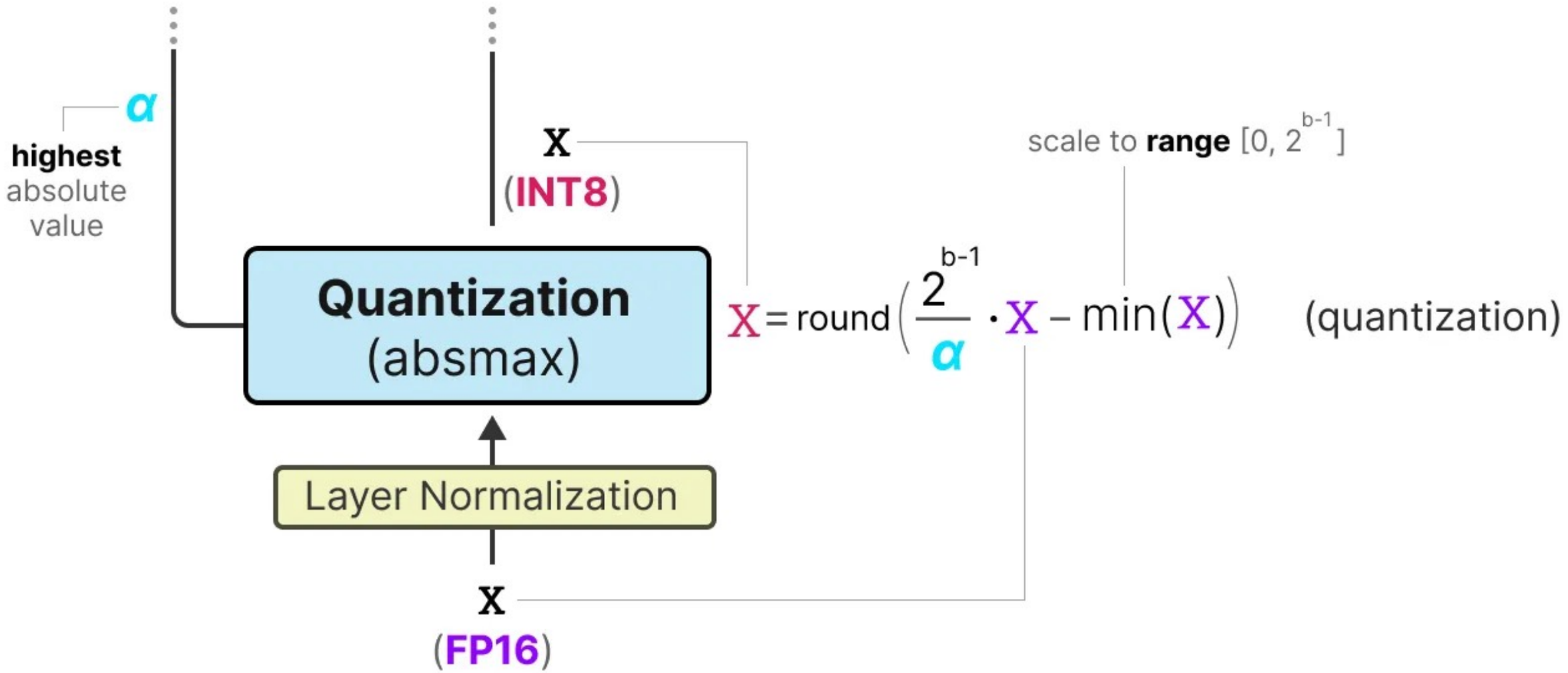
The Era of 1-bit LLMs: BitNet



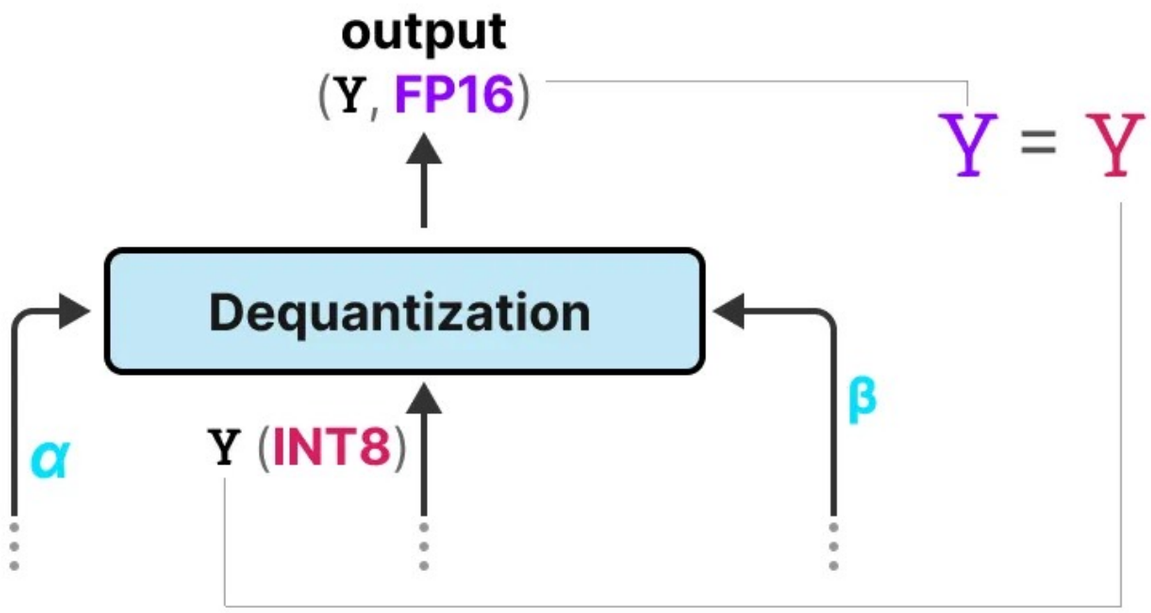
Weight Quantization



Activation Quantization



Dequantization



$$Y = Y \cdot \frac{\alpha \cdot \beta}{2^{b-1}} \quad \text{(dequantize)}$$