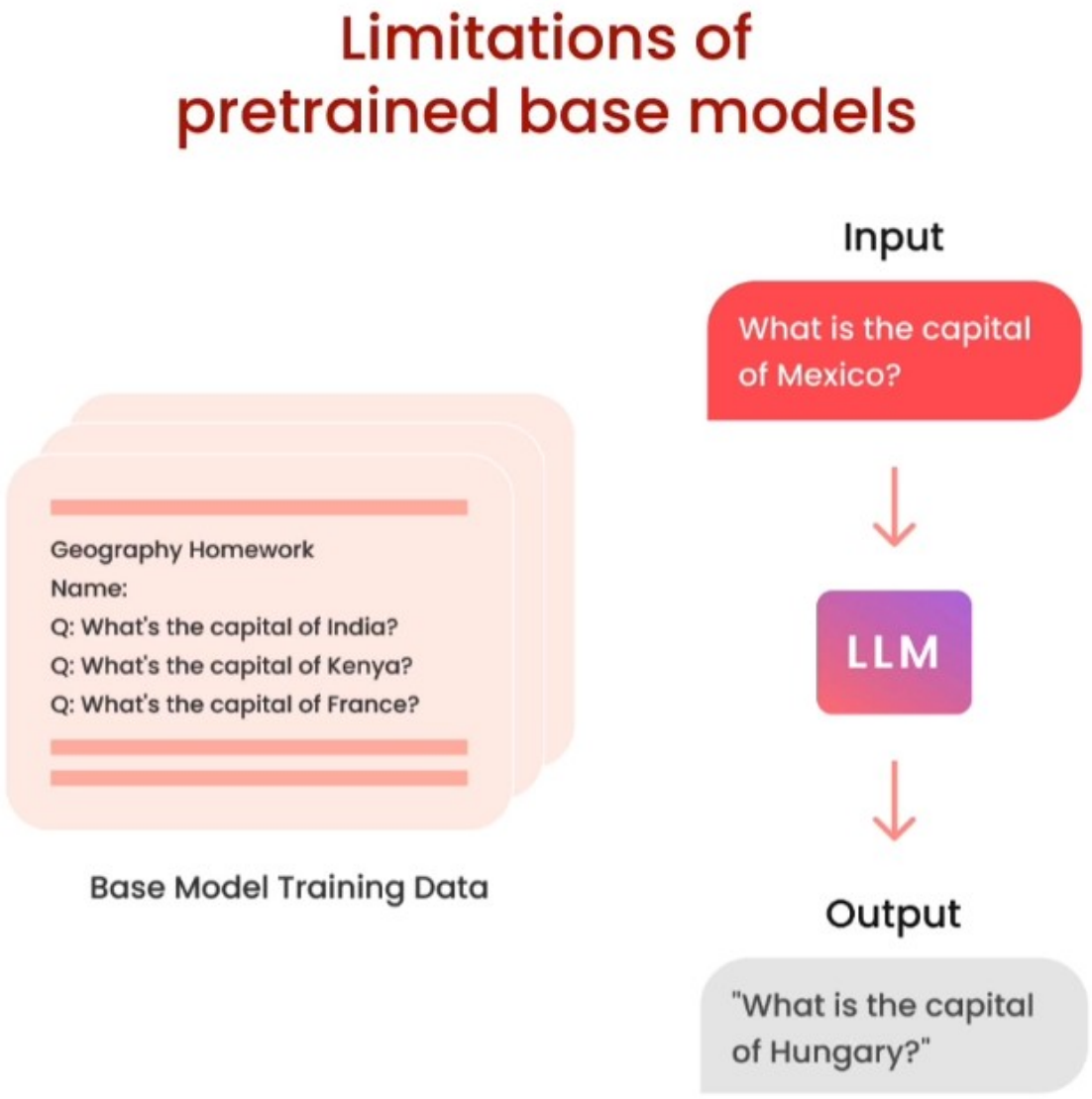
Fine-Tuning LLMs

by Daniel Voigt Godoy

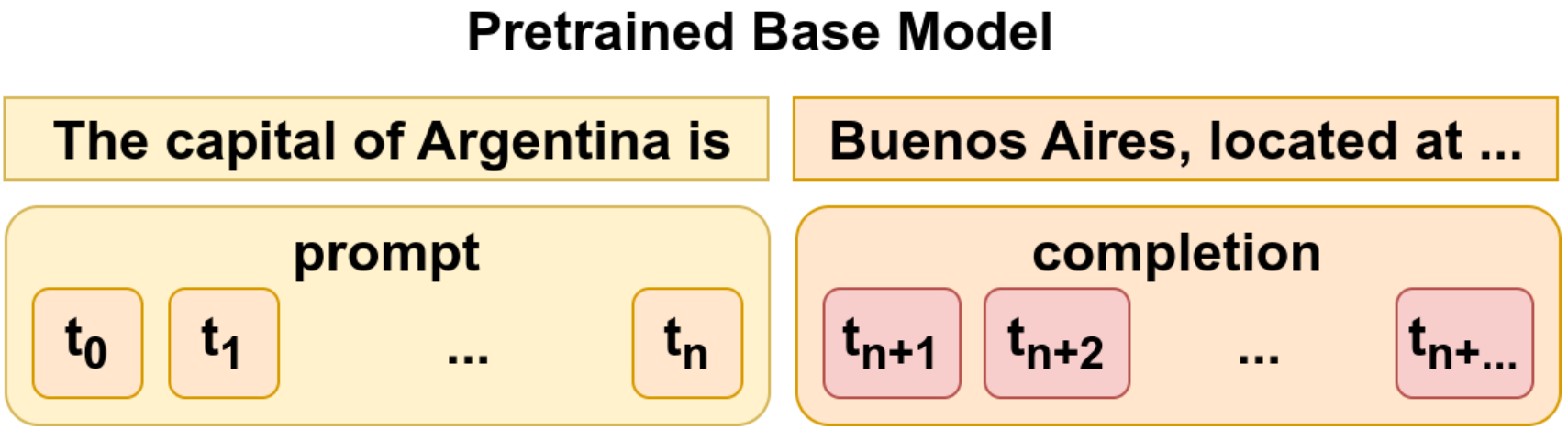
Data Science Retreat – November 2024

Why Fine-Tuning?



Source: https://www.deeplearning.ai/short-courses/finetuning-large-language-models/

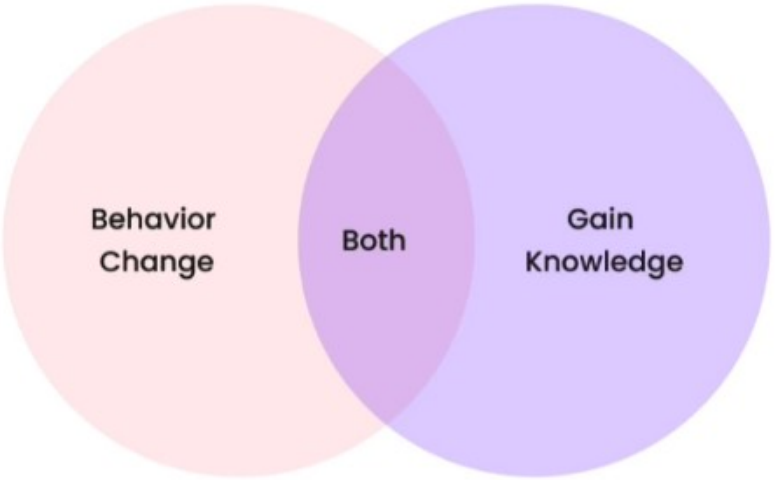
Next Token Prediction



Why Fine-Tuning?

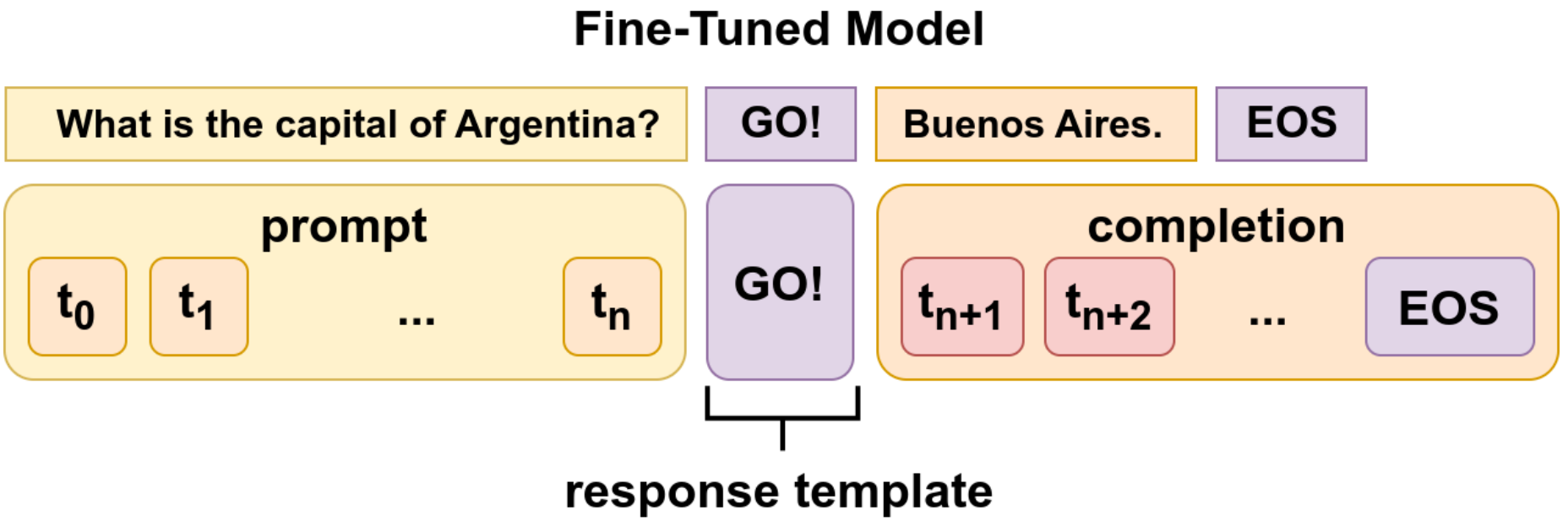
What is finetuning doing for you?

- Behavior change
  - Learning to respond more consistently
  - Learning to focus, e.g. moderation
  - Teasing out capability, e.g. better at conversation
- Gain knowledge
  - Increasing knowledge of new specific concepts
  - Correcting old incorrect information
- Both



Source: https://www.deeplearning.ai/short-courses/finetuning-large-language-models/

Triggering Responses



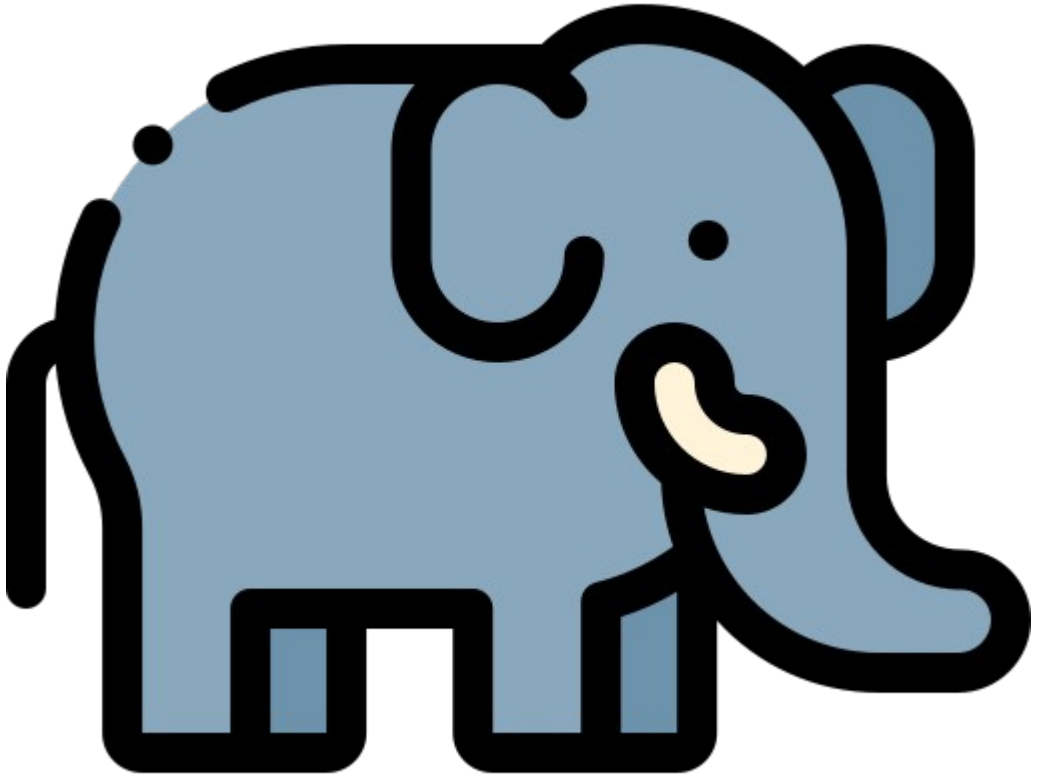
Fine-Tuning

Ways to Fine-Tune the LLM's

1. **Self-Supervised Fine Tuning:** This involves predicting the next word or token in a given sequence. In scenarios where you require the language model to "*emulate a specific writing style*" of user such approach is very useful for fine-tuning.
2. **Supervised Fine Tuning:** This will involve creating input/output pairs or question/answer or prompt/response pairs related to the domain in which you want the model to be fine-tuned. This approach can be useful in scenario where the model has to *"respond to user prompts"*.
3. **Reinforcement based Fine Tuning:** This involves taking the supervised/self-supervised model output via human labellers for labelling and passing this via a reward model (like Proximal Policy Optimization) to "*improve the results*".

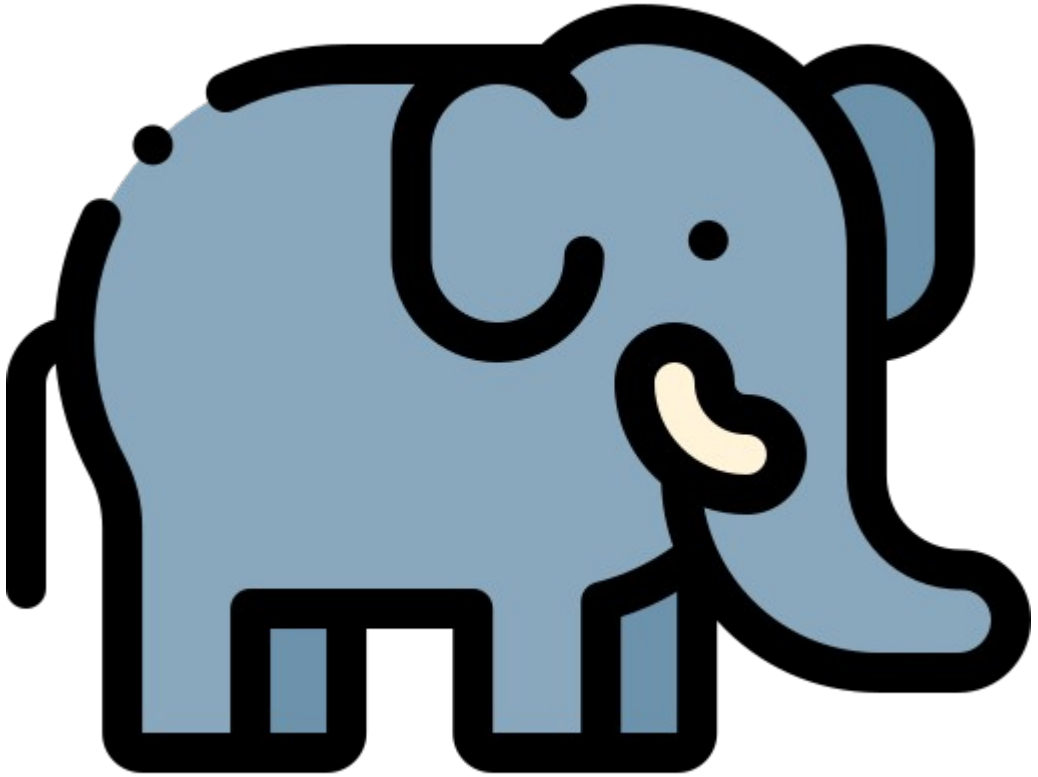
Source: https://medium.com/@rahulpant.me/fine-tuning-llms-basics-an-example-b9ea039751e8

How to Fine-Tune an LLM?

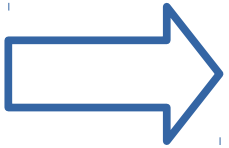


LLM

How to Fine-Tune an LLM?

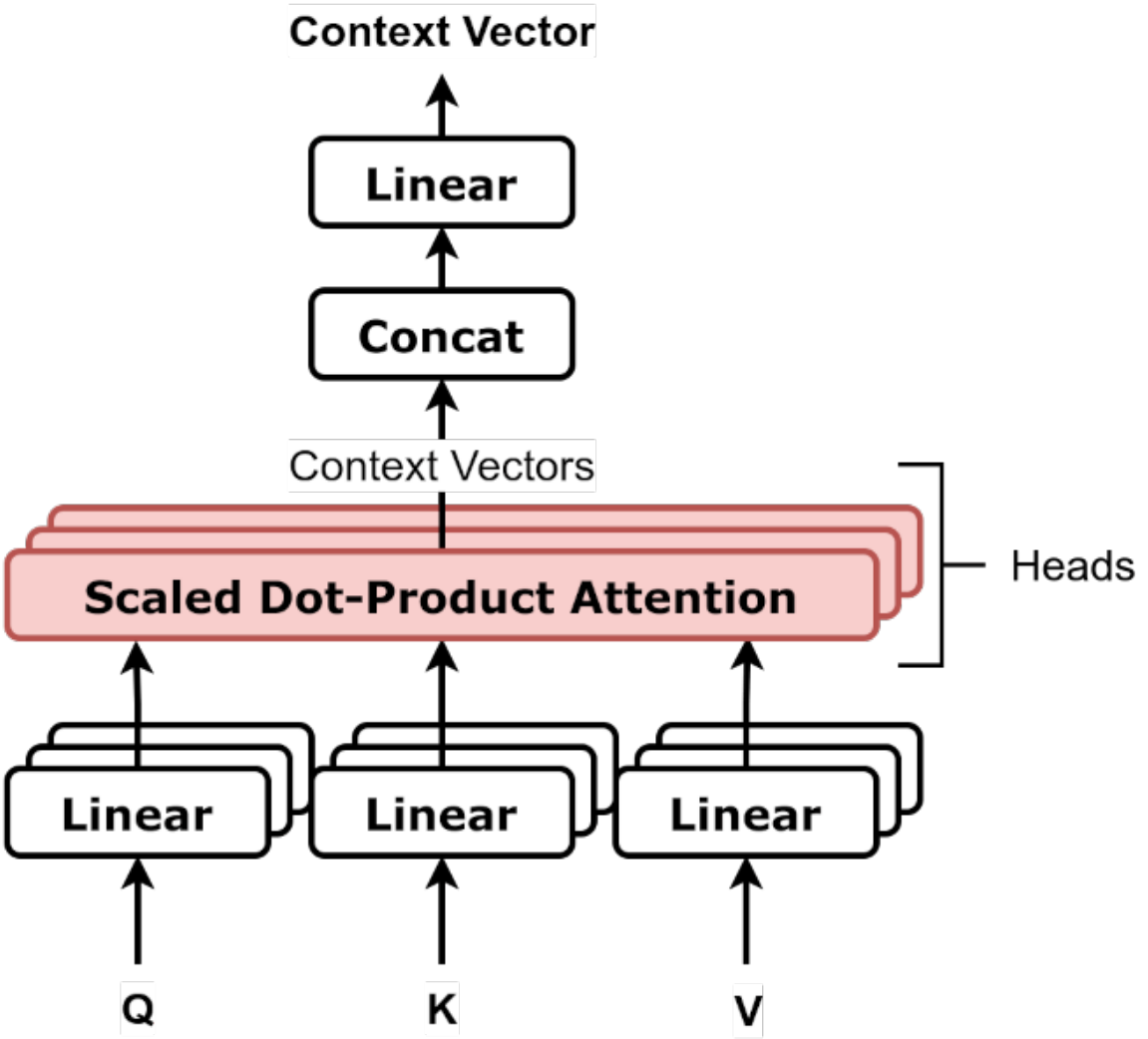


LLM



GPU

Multi-Head Attention (MHA)



Many Problems!

- Base model takes a lot of space (p)
- Optimizer's states (Adam) takes even more space (2p)
- Gradients take a lot of space (p)
- Activations take a lot of space, especially if the sequences are long
- Attention is quadratic on the sequence length (10x longer => 100x more expensive to handle)

p = #parameters

Many Problems!

- ~~Base model takes a lot of space (p)~~ **Quantization**
- Optimizer's states (Adam) takes even more space (2p)
- Gradients take a lot of space (p)
- Activations take a lot of space, especially if the sequences are long
- Attention is quadratic on the sequence length (10x longer => 100x more expensive to handle)

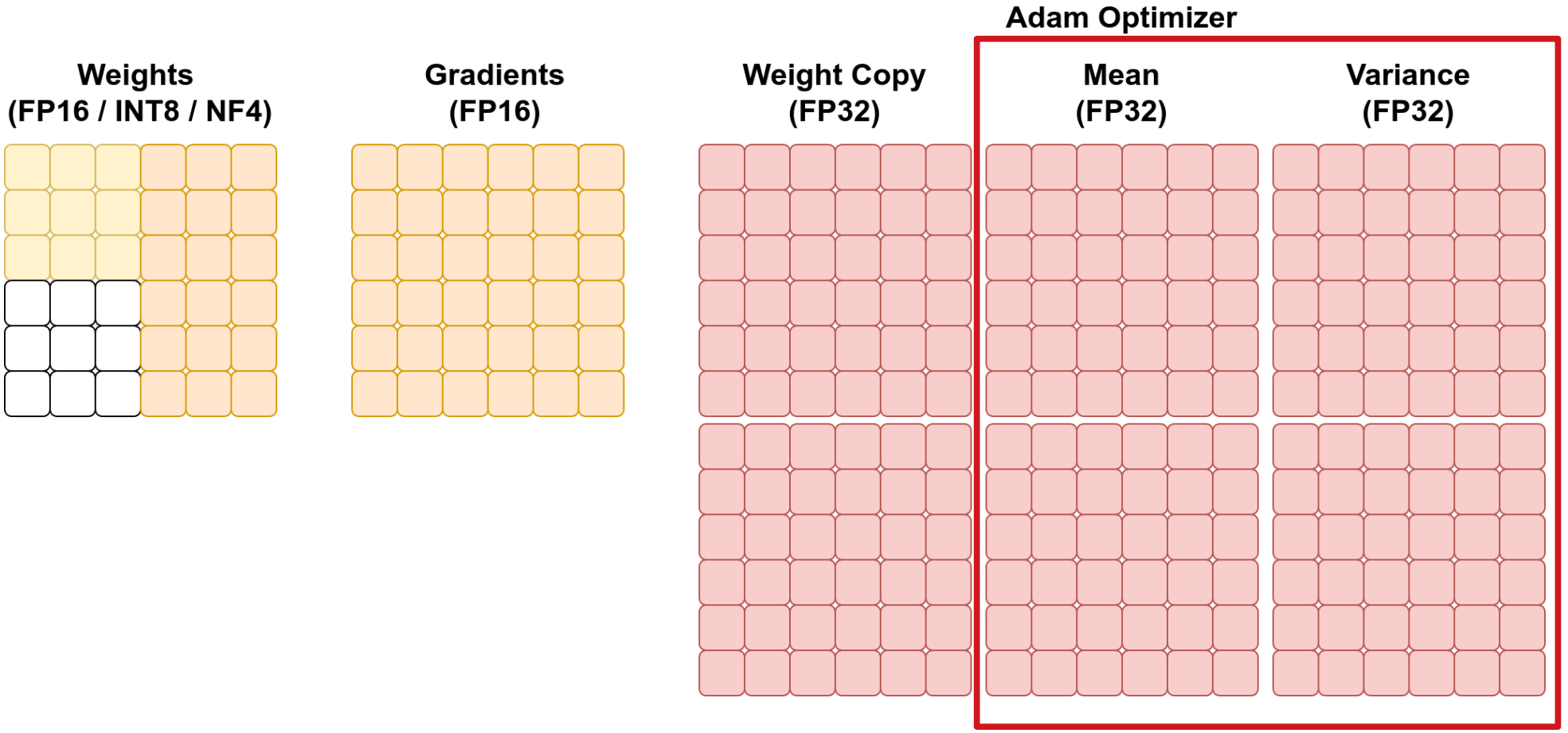
p = #parameters

Many Problems!

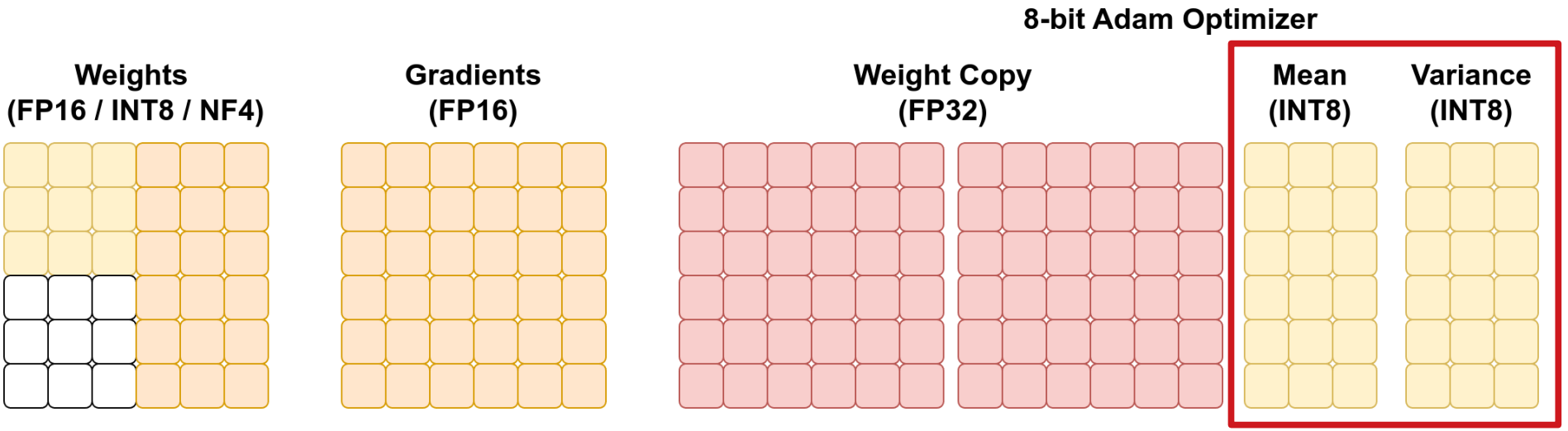
- ~~Base model takes a lot of space (p)~~ **Quantization**
- **Optimizer's states (Adam) takes even more space (2p)**
- **Gradients take a lot of space (p)**
- Activations take a lot of space, especially if the sequences are long
- Attention is quadratic on the sequence length (10x longer => 100x more expensive to handle)

p = #parameters

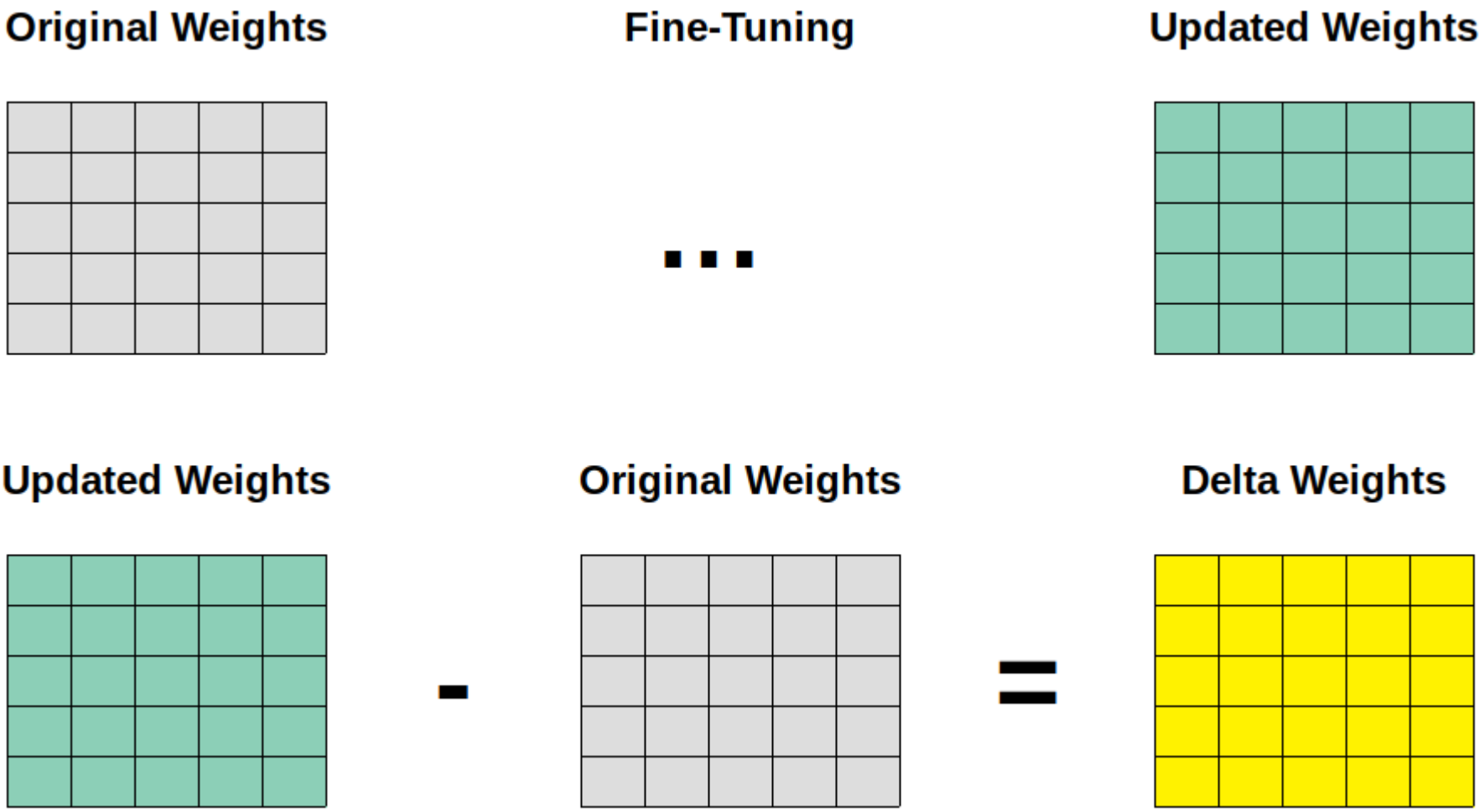
The Optimizer Problem



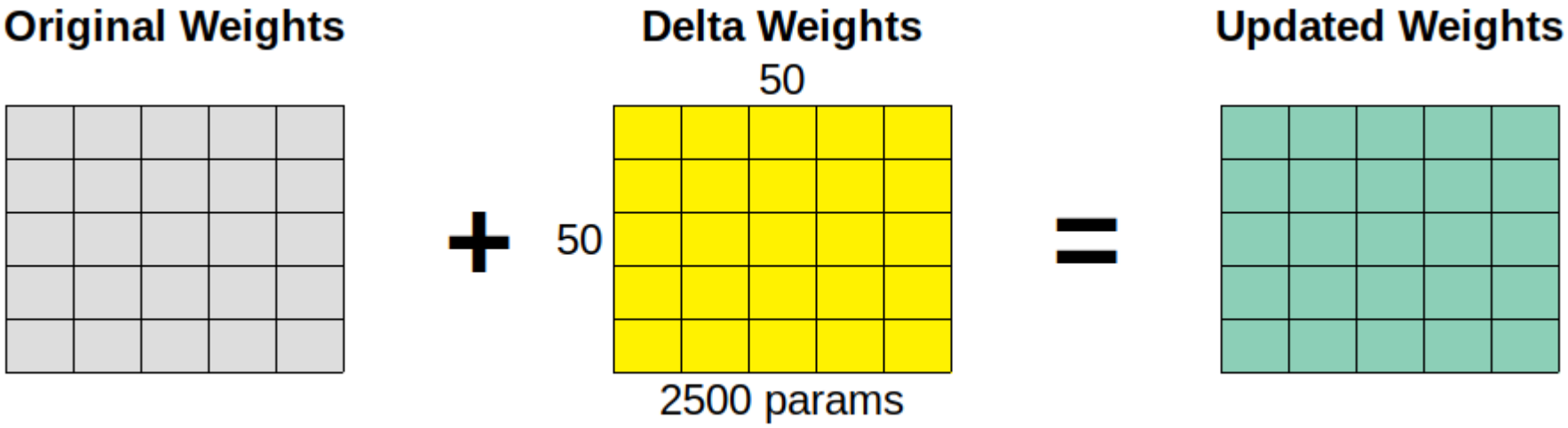
The Optimizer Problem



Low-Rank Adaptation(LoRA)



Low-Rank Adaptation(LoRA)



Low-Rank Adaptation(LoRA)

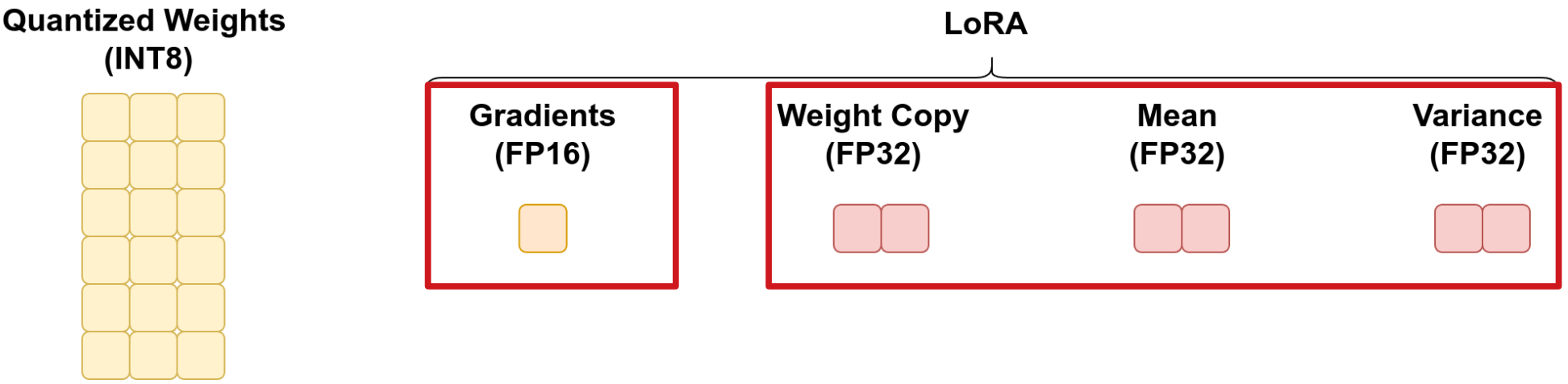
$$W_{LoRA} = W_{orig} + \Delta W$$

$$W_{LoRA} = W_{orig} + \frac{\alpha}{r}\Delta W \qquad \Delta W = BA$$

```
def regular_forward_matmul(x, W):
    h = x @ W
return h

def lora_forward_matmul(x, W, W_A, W_B):
    h = x @ W  # regular matrix multiplication
    h += x @ (W_A @ W_B)*alpha # use scaled LoRA weights
return h
```

The Optimizer Problem



Many Problems!

- ~~Base model takes a lot of space (p)~~ **Quantization**
- ~~Optimizer's states (Adam) takes even more space (2p)~~ **LoRA / 8-bit Adam**
- ~~Gradients take a lot of space (p)~~ **LoRA**
- Activations take a lot of space, especially if the sequences are long
- Attention is quadratic on the sequence length (10x longer => 100x more expensive to handle)

p = #parameters

Low-Rank Adaptation(LoRA)

The Five Commandments of Low-Rank Adaptation

1) Utilize LoRA for efficient model fine-tuning, focusing on keeping parameter sizes minimal.
2) Employ the PEFT library for LoRA implementation, avoiding the need for complex coding.
3) Extend LoRA adaptations to all linear layers, enhancing overall model capabilities.
4) Keep biases* and layer norms trainable, as they are critical for model adaptability and don’t require low-rank adaptations.
5) Apply Quantized-LoRA — QLoRA** — to preserve GPU VRAM and train your model, enabling the training of larger models.

*beware that training biases will modify the base model’s behavior!

**more about quantization later

Source: https://towardsdatascience.com/implementing-lora-from-scratch-20f838b046f1

HuggingFace’s PEFT

PEFT

🤗 PEFT (Parameter-Efficient Fine-Tuning) is a library for efficiently adapting large pretrained models to various downstream applications without fine-tuning all of a model’s parameters because it is prohibitively costly. PEFT methods only fine-tune a small number of (extra) model parameters - significantly decreasing computational and storage costs - while yielding performance comparable to a fully fine-tuned model. This makes it more accessible to train and store large language models (LLMs) on consumer hardware.

PEFT is integrated with the Transformers, Diffusers, and Accelerate libraries to provide a faster and easier way to load, train, and use large models for inference.

Source: https://huggingface.co/docs/peft/en/index

HuggingFace’s PEFT

Common LoRA parameters in PEFT

As with other methods supported by PEFT, to fine-tune a model using LoRA, you need to:

1. Instantiate a base model.
2. Create a configuration (`LoraConfig`) where you define LoRA-specific parameters.
3. Wrap the base model with `get_peft_model()` to get a trainable `PeftModel`.
4. Train the `PeftModel` as you normally would train the base model.

Source: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora

HuggingFace’s PEFT

`LoraConfig` allows you to control how LoRA is applied to the base model through the following parameters:

- `r`: the rank of the update matrices, expressed in `int`. Lower rank results in smaller update matrices with fewer trainable parameters.
- `target_modules`: The modules (for example, attention blocks) to apply the LoRA update matrices.
- `lora_alpha`: LoRA scaling factor.
- `bias`: Specifies if the `bias` parameters should be trained. Can be `'none'`, `'all'` or `'lora_only'`.
- `use_rslora`: When set to True, uses Rank-Stabilized LoRA which sets the adapter scaling factor to `lora_alpha/math.sqrt(r)`, since it was proven to work better. Otherwise, it will use the original default value of `lora_alpha/r`.
- `modules_to_save`: List of modules apart from LoRA layers to be set as trainable and saved in the final checkpoint. These typically include model’s custom head that is randomly initialized for the fine-tuning task.

Source: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora

HuggingFace’s PEFT

Initialization options

The initialization of LoRA weights is controlled by the parameter `init_lora_weights` of the `LoraConfig`. By default, PEFT initializes LoRA weights the same way as the reference implementation, i.e. using Kaiming-uniform for weight A and initializing weight B as zeros, resulting in an identity transform.

It is also possible to pass `init_lora_weights="gaussian"`. As the name suggests, this results in initializing weight A with a Gaussian distribution (weight B is still zeros). This corresponds to the way that diffusers initializes LoRA weights.

Source: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora

HuggingFace’s PEFT

When quantizing the base model, e.g. for QLoRA training, consider using the LoftQ initialization, which has been shown to improve the performance with quantization. The idea is that the LoRA weights are initialized such that the quantization error is minimized. To use this option, *do not* quantize the base model. Instead, proceed as follows:

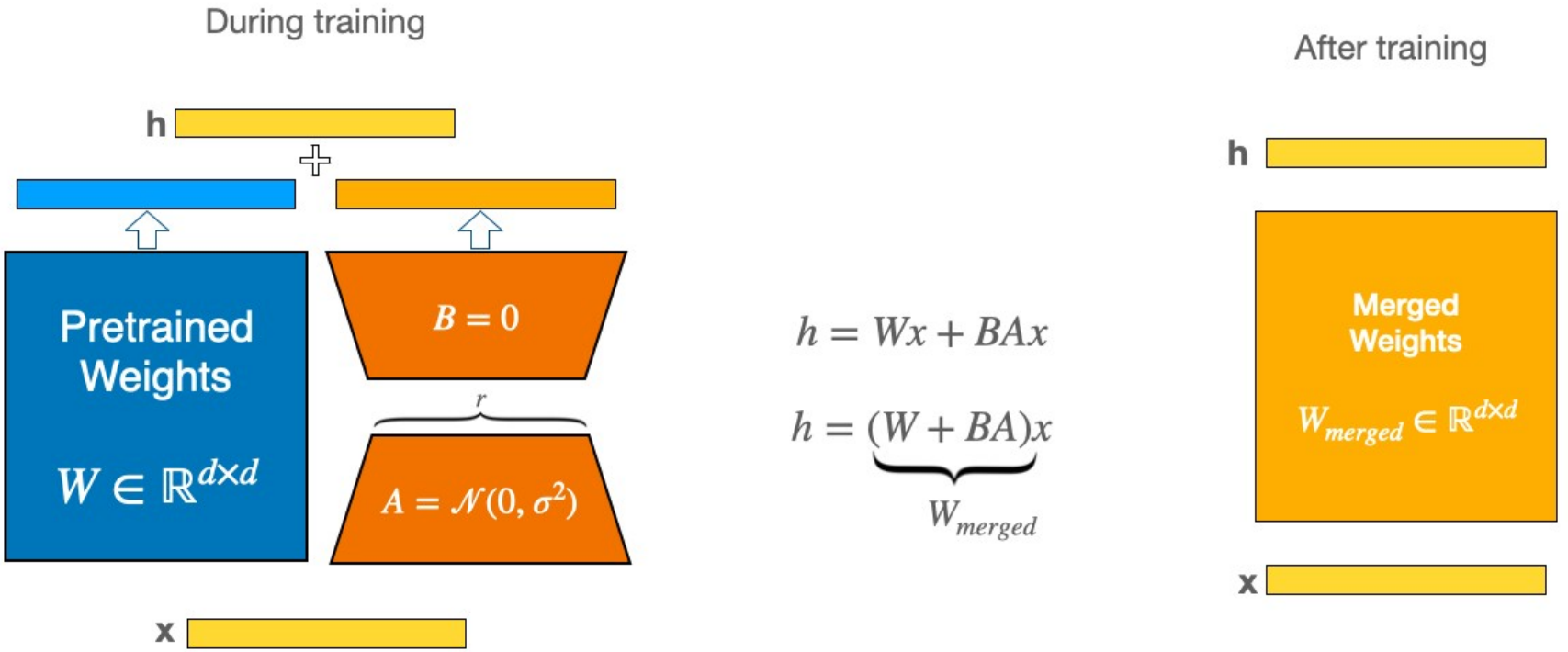
```
from peft import LoftQConfig, LoraConfig, get_peft_model

base_model = AutoModelForCausalLM.from_pretrained(...)  # don't quantize here
loftq_config = LoftQConfig(loftq_bits=4, ...)           # set 4bit quantization
lora_config = LoraConfig(..., init_lora_weights="loftq", loftq_config=loftq_config)
peft_model = get_peft_model(base_model, lora_config)
```

There is also an option to set `initialize_lora_weights=False`. When choosing this option, the LoRA weights are initialized such that they do *not* result in an identity transform. This is useful for debugging and testing purposes and should not be used otherwise.

Source: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora

HuggingFace's PEFT



Source: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora

HuggingFace’s PEFT

Utils for LoRA

Use merge_adapter() to merge the LoRa layers into the base model while retaining the PeftModel. This will help in later unmerging, deleting, loading different adapters and so on.

Use unmerge_adapter() to unmerge the LoRa layers from the base model while retaining the PeftModel. This will help in later merging, deleting, loading different adapters and so on.

Use unload() to get back the base model without the merging of the active lora modules. This will help when you want to get back the pretrained base model in some applications when you want to reset the model to its original state. For example, in Stable Diffusion WebUi, when the user wants to infer with base model post trying out LoRAs.

Use delete_adapter() to delete an existing adapter.

Use add_weighted_adapter() to combine multiple LoRAs into a new adapter based on the user provided weighing scheme.

Source: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora

Hands-On

Notebook 2 - PeFT LoRA

Harvard Sentences

Harvard Sentences

From "IEEE Recommended Practice for Speech Quality Measurements." *IEEE Transactions on Audio and Electroacoustics*, Vol. 17, Issue 3, 225-246, 1969. APPENDIX C 1965 Revised List of Phonetically Balanced Sentences (Harvard Sentences). DOI 10.1109/IEEESTD.1969.7405210 (IEEE standard 297-1969) and 10.1109/TAU.1969.1162058

Also found at CMU site and the *TSP Speech Database* at McGill.

List 1

1. The birch canoe slid on the smooth planks.
2. Glue the sheet to the dark blue background.
3. It's easy to tell the depth of a well.
4. These days a chicken leg is a rare dish.
5. Rice is often served in round bowls.
6. The juice of lemons makes fine punch.
7. The box was thrown beside the parked truck.
8. The hogs were fed chopped corn and garbage.
9. Four hours of steady work faced us.
10. A large size in stockings is hard to sell.

Source: https://www.cs.columbia.edu/~hgs/audio/harvard.html

Pig Latin

Rules [edit]

For words that begin with consonant sounds, the initial consonant is moved to the end of the word, then "ay" is added, as in the following examples:[12]

- "pig" = "igpay"
- "latin" = "atinlay"
- "banana" = "ananabay"

When words begin with consonant clusters (multiple consonants that form one sound), the whole sound is moved to the end (before adding "ay") when speaking or writing.[13]

- "friends" = "iendsfray"
- "smile" = "ilesmay"
- "string" = "ingstray"

For words that begin with vowel sounds, one just adds "hay", "way", "nay" or "yay" to the end. Examples are:

- "eat" = "eatway"
- "omelet" = "omeletway"
- "are" = "areway"

Source: https://en.wikipedia.org/wiki/Pig_Latin

Pig Latin

```
import re
from string import punctuation

def pig_latin(sentence):
    toks = [t.lower() for t in re.findall(r'\w+|[^\s\w]+', sentence) if len(t) > 0]

    def convert(string):
        # if starts with a vowel, just add "ay"
        # else move the consonants to the end and add "ay"
        if string in punctuation:
            return string
        elif string[0].lower() in {'a', 'e', 'i', 'o', 'u'}:
            return ' ' + string + 'way'
        else:
            beginning_consonants = []
            for i in range(len(string)):
                if string[i].lower() in {'a', 'e', 'i', 'o', 'u'}:
                    break
                beginning_consonants.append(string[i])
            return ' ' + string[i:] + ''.join(beginning_consonants) + 'ay'

    return ''.join([convert(t) for t in toks]).strip()
```

Data Collators

Data Collator

Data collators are objects that will form a batch by using a list of dataset elements as input. These elements are of the same type as the elements of `train_dataset` or `eval_dataset`.

To be able to build batches, data collators may apply some processing (like padding). Some of them (like DataCollatorForLanguageModeling) also apply some random data augmentation (like random masking) on the formed batch.

Source: https://huggingface.co/docs/transformers/en/main_classes/data_collator

Collator for LM

DataCollatorForLanguageModeling

class transformers.**DataCollatorForLanguageModeling** <source>

```
( tokenizer: PreTrainedTokenizerBase, mlm: bool = True, mlm_probability: float = 0.15,
pad_to_multiple_of: Optional = None, tf_experimental_compile: bool = False, return_tensors: str = 'pt' )
```

Parameters

- **tokenizer** (PreTrainedTokenizer or PreTrainedTokenizerFast) — The tokenizer used for encoding the data.
- **mlm** (bool, *optional*, defaults to True) — Whether or not to use masked language modeling. If set to False, the labels are the same as the inputs with the padding tokens ignored (by setting them to -100). Otherwise, the labels are -100 for non-masked tokens and the value to predict for the masked token.
- **mlm_probability** (float, *optional*, defaults to 0.15) — The probability with which to (randomly) mask tokens in the input, when mlm is set to True.
- **pad_to_multiple_of** (int, *optional*) — If set will pad the sequence to a multiple of the provided value.
- **return_tensors** (str) — The type of Tensor to return. Allowable values are “np”, “pt” and “tf”.

Set to FALSE

Source: https://huggingface.co/docs/transformers/en/main_classes/data_collator

Collator for Completions

Train on completions only

You can use the `DataCollatorForCompletionOnlyLM` to train your model on the generated prompts only.

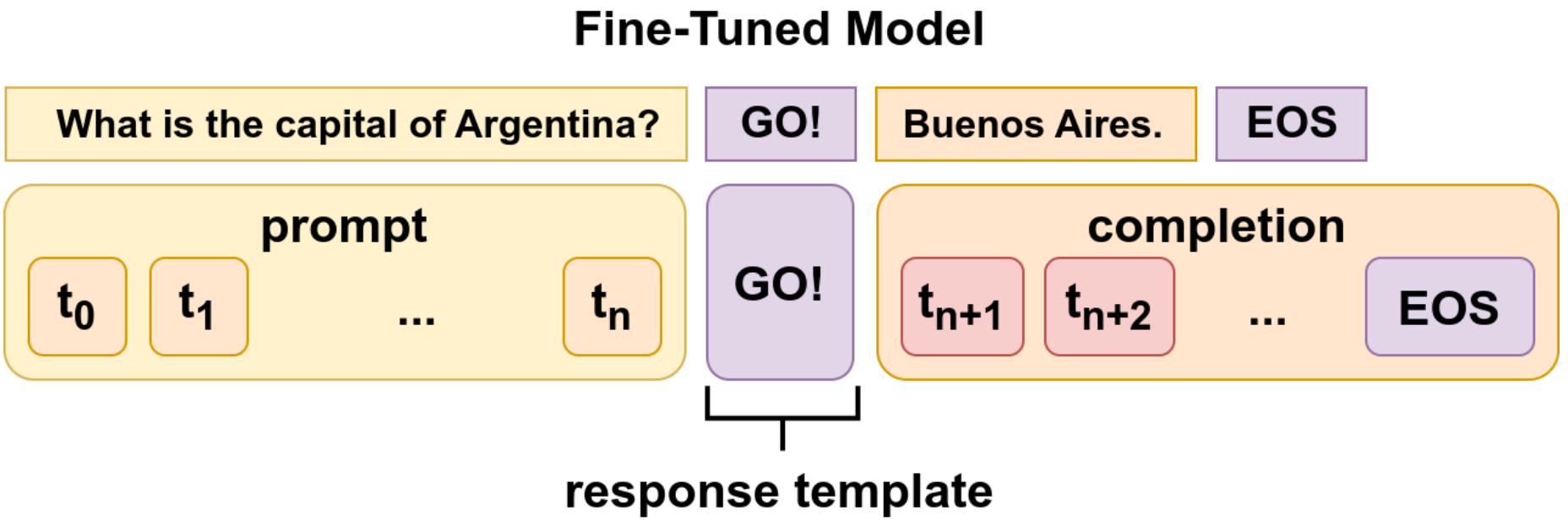
```python
def formatting_prompts_func(example):
    output_texts = []
    for i in range(len(example['instruction'])):
        text = f"### Question: {example['instruction'][i]}\n ### Answer: {example['output'][i]}"
        output_texts.append(text)
    return output_texts

response_template = " ### Answer:"
collator = DataCollatorForCompletionOnlyLM(response_template, tokenizer=tokenizer)
```

Make sure to have a `pad_token_id` which is different from `eos_token_id` which can result in the model not properly predicting EOS (End of Sentence) tokens during generation.

Source: https://huggingface.co/docs/trl/en/sft_trainer

Triggering Responses



Hands-On

Notebook 3 - Pig Latin and Data Collators

Many Problems!

- ~~Base model takes a lot of space (p)~~ **Quantization**
- ~~Optimizer's states (Adam) takes even more space (2p)~~ **LoRA / 8-bit Adam**
- ~~Gradients take a lot of space (p)~~ **LoRA**
- **Activations take a lot of space, especially if the sequences are long**
- **Attention is quadratic on the sequence length (10x longer => 100x more expensive to handle)**

p = #parameters

The Activation Problem

$$\text{memory}_{\text{model}} = L 12 h^2$$

$$\text{memory}_{\text{act}} = L \left(\alpha h b s + \beta n_{\text{heads}} b s^2 \right)$$

$$\text{memory}_{\text{act}}^{\text{eager attn}} \underset{\substack{\alpha=34 \\ \beta=5}}{=} L \left(34 h b s + 5 n_{\text{heads}} b s^2 \right)$$

$$\text{memory}_{\text{act}}^{\text{flash attn}} \underset{\substack{\alpha=34 \\ \beta=0}}{=} L \left(34 h b s \right)$$

L = # Transformer blocks / “layers”
h = model’s dimensionality
n_heads = # attention heads

b = batch size
s = sequence length

The Activation Problem

$$\frac{\text{memory}_{\text{act}}^{\text{eager attn}}}{\text{memory}_{\text{model}}} = \left[\frac{34bs}{12h} + \frac{5n_{\text{heads}}bs^2}{12h^2}\right]\frac{p_{\text{comp}}}{p_{\text{model}}}$$

$$\underset{b=1}{\approx} \left[3\frac{s}{h} + \frac{1}{2}n_{\text{heads}}\left(\frac{s}{h}\right)^2\right]\frac{p_{\text{comp}}}{p_{\text{model}}}$$

$$\frac{\text{memory}_{\text{act}}^{\text{flash attn}}}{\text{memory}_{\text{model}}} = \frac{34bs}{12h}\frac{p_{\text{comp}}}{p_{\text{model}}} \underset{b=1}{\approx} 3\frac{s}{h}\frac{p_{\text{comp}}}{p_{\text{model}}}$$

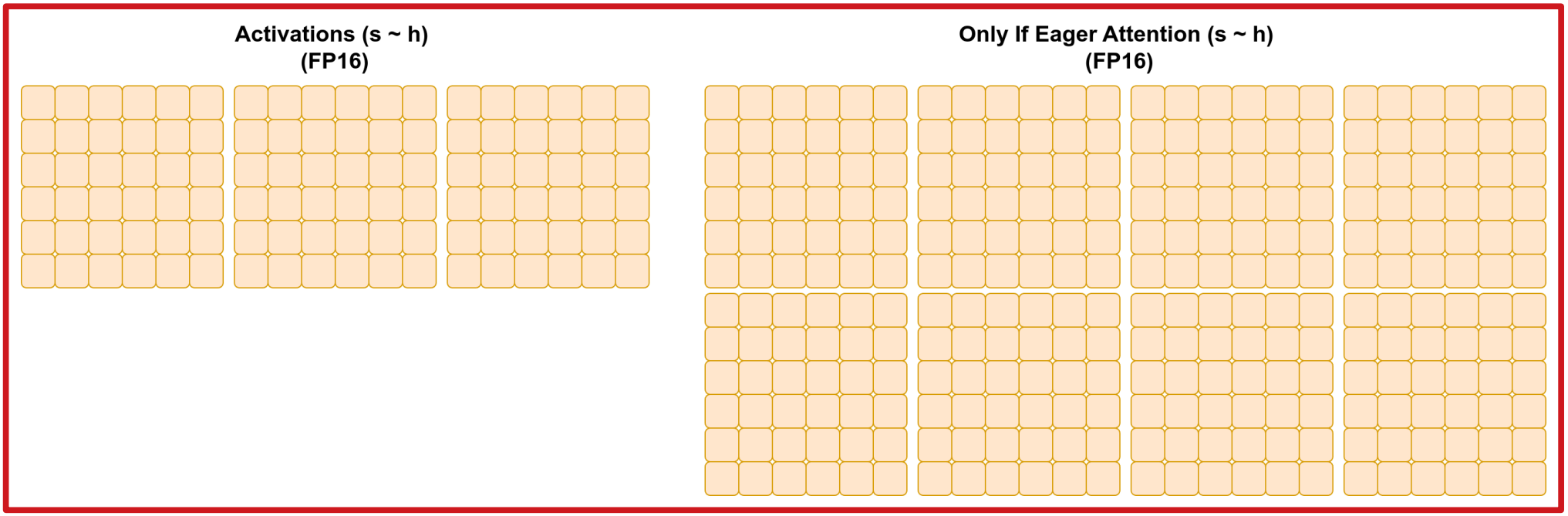
L = # Transformer blocks / “layers”
h = model’s dimensionality
n_heads = # attention heads

b = batch size
s = sequence length

The Activation Problem

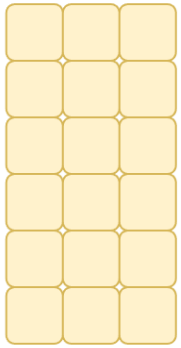
Model	hidden_dim (h)	max seq (s)	n_heads (n_h)	n_layers (L)
OPT-350M	1024	2048	16	24
Phi-3.5 Mini	3072	4096	32	32
Llama-2 7B	4096	4096	32	32
Llama-3.2 3B	3072	8192	24	28
Mistral 8B	4096	32768	32	32
Qwen-2.5 7B	3584	32768	28	28
OLMo 7B	4096	2048	32	32

The Activation Problem



The Activation Problem

Quantized Weights
(INT8)



LoRA

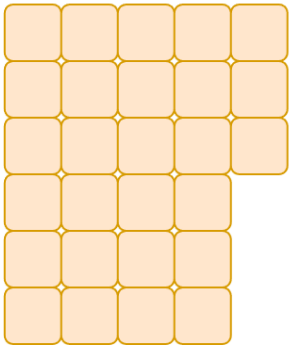
Gradients
(FP16)

Weight Copy
(FP32)

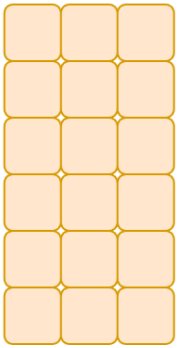
Mean
(FP32)

Variance
(FP32)

Activations (s ~ h/4)
(FP16)



Only If Eager Attention (s ~ h/4)
(FP16)



HuggingFace Trainer

Trainer

The Trainer class provides an API for feature-complete training in PyTorch, and it supports distributed training on multiple GPUs/TPUs, mixed precision for NVIDIA GPUs, AMD GPUs, and `torch.amp` for PyTorch. Trainer goes hand-in-hand with the TrainingArguments class, which offers a wide range of options to customize how a model is trained. Together, these two classes provide a complete training API.

Seq2SeqTrainer and Seq2SeqTrainingArguments inherit from the Trainer and `TrainingArgument` classes and they're adapted for training models for sequence-to-sequence tasks such as summarization or translation.

> The Trainer class is optimized for 🤗 Transformers models and can have surprising behaviors when used with other models. When using it with your own model, make sure:
>
> - your model always return tuples or subclasses of ModelOutput
> - your model can compute the loss if a `labels` argument is provided and that loss is returned as the first element of the tuple (if your model returns tuples)
> - your model can accept multiple label arguments (use `label_names` in TrainingArguments to indicate their name to the Trainer) but none of them should be named `"label"`

Source: https://huggingface.co/docs/transformers/en/main_classes/trainer

TrainingArguments

class transformers.TrainingArguments <source>

```
( output_dir: str, overwrite_output_dir: bool = False, do_train: bool = False, do_eval: bool = False,
do_predict: bool = False, eval_strategy: Union = 'no', prediction_loss_only: bool = False,
per_device_train_batch_size: int = 8, per_device_eval_batch_size: int = 8, per_gpu_train_batch_size:
Optional = None, per_gpu_eval_batch_size: Optional = None, gradient_accumulation_steps: int = 1,
eval_accumulation_steps: Optional = None, eval_delay: Optional = 0, learning_rate: float = 5e-05,
weight_decay: float = 0.0, adam_beta1: float = 0.9, adam_beta2: float = 0.999, adam_epsilon: float =
1e-08, max_grad_norm: float = 1.0, num_train_epochs: float = 3.0, max_steps: int = -1,
lr_scheduler_type: Union = 'linear', lr_scheduler_kwargs: Union = <factory>, warmup_ratio: float = 0.0,
warmup_steps: int = 0, log_level: Optional = 'passive', log_level_replica: Optional = 'warning',
log_on_each_node: bool = True, logging_dir: Optional = None, logging_strategy: Union = 'steps',
logging_first_step: bool = False, logging_steps: float = 500, logging_nan_inf_filter: bool = True,
save_strategy: Union = 'steps', save_steps: float = 500, save_total_limit: Optional = None,
save_safetensors: Optional = True, save_on_each_node: bool = False, save_only_model: bool = False,
restore_callback_states_from_checkpoint: bool = False, no_cuda: bool = False, use_cpu: bool = False,
use_mps_device: bool = False, seed: int = 42, data_seed: Optional = None, jit_mode_eval: bool = False,
use_ipex: bool = False, bf16: bool = False, fp16: bool = False, fp16_opt_level: str = '01',
half_precision_backend: str = 'auto', bf16_full_eval: bool = False, fp16_full_eval: bool = False, tf32:
Optional = None, local_rank: int = -1, ddp_backend: Optional = None, tpu_num_cores: Optional = None,
tpu_metrics_debug: bool = False, debug: Union = '', dataloader_drop_last: bool = False, eval_steps:
Optional = None, dataloader_num_workers: int = 0, dataloader_prefetch_factor: Optional = None,
past_index: int = -1, run_name: Optional = None, disable_tqdm: Optional = None, remove_unused_columns:
Optional = True, label_names: Optional = None, load_best_model_at_end: Optional = False,
metric_for_best_model: Optional = None, greater_is_better: Optional = None, ignore_data_skip: bool =
False, fsdp: Union = '', fsdp_min_num_params: int = 0, fsdp_config: Union = None,
```


Source: https://huggingface.co/docs/transformers/en/main_classes/trainer

TrainingArguments

```
fsdp_transformer_layer_cls_to_wrap: Optional = None, accelerator_config: Union = None, deepspeed: Union
= None, label_smoothing_factor: float = 0.0, optim: Union = 'adamw_torch', optim_args: Optional = None,
adafactor: bool = False, group_by_length: bool = False, length_column_name: Optional = 'length',
report_to: Union = None, ddp_find_unused_parameters: Optional = None, ddp_bucket_cap_mb: Optional =
None, ddp_broadcast_buffers: Optional = None, dataloader_pin_memory: bool = True,
dataloader_persistent_workers: bool = False, skip_memory_metrics: bool = True,
use_legacy_prediction_loop: bool = False, push_to_hub: bool = False, resume_from_checkpoint: Optional =
None, hub_model_id: Optional = None, hub_strategy: Union = 'every_save', hub_token: Optional = None,
hub_private_repo: bool = False, hub_always_push: bool = False, gradient_checkpointing: bool = False,
gradient_checkpointing_kwargs: Union = None, include_inputs_for_metrics: bool = False,
eval_do_concat_batches: bool = True, fp16_backend: str = 'auto', evaluation_strategy: Union = None,
push_to_hub_model_id: Optional = None, push_to_hub_organization: Optional = None, push_to_hub_token:
Optional = None, mp_parameters: str = '', auto_find_batch_size: bool = False, full_determinism: bool =
False, torchdynamo: Optional = None, ray_scope: Optional = 'last', ddp_timeout: Optional = 1800,
torch_compile: bool = False, torch_compile_backend: Optional = None, torch_compile_mode: Optional =
None, dispatch_batches: Optional = None, split_batches: Optional = None, include_tokens_per_second:
Optional = False, include_num_input_tokens_seen: Optional = False, neftune_noise_alpha: Optional = None,
optim_target_modules: Union = None, batch_eval_metrics: bool = False )
```


Source: https://huggingface.co/docs/transformers/en/main_classes/trainer

HuggingFace SFTTrainer

Supervised Fine-tuning Trainer

Supervised fine-tuning (or SFT for short) is a crucial step in RLHF. In TRL we provide an easy-to-use API to create your SFT models and train them with few lines of code on your dataset.

Quickstart

If you have a dataset hosted on the 🤗 Hub, you can easily fine-tune your SFT model using SFTTrainer from TRL. Let us assume your dataset is `imdb`, the text you want to predict is inside the `text` field of the dataset, and you want to fine-tune the `facebook/opt-350m` model. The following code-snippet takes care of all the data pre-processing and training for you:

```
from datasets import load_dataset
from trl import SFTTrainer

dataset = load_dataset("imdb", split="train")

trainer = SFTTrainer(
    "facebook/opt-350m",
    train_dataset=dataset,
    dataset_text_field="text",
    max_seq_length=512,
)
trainer.train()
```

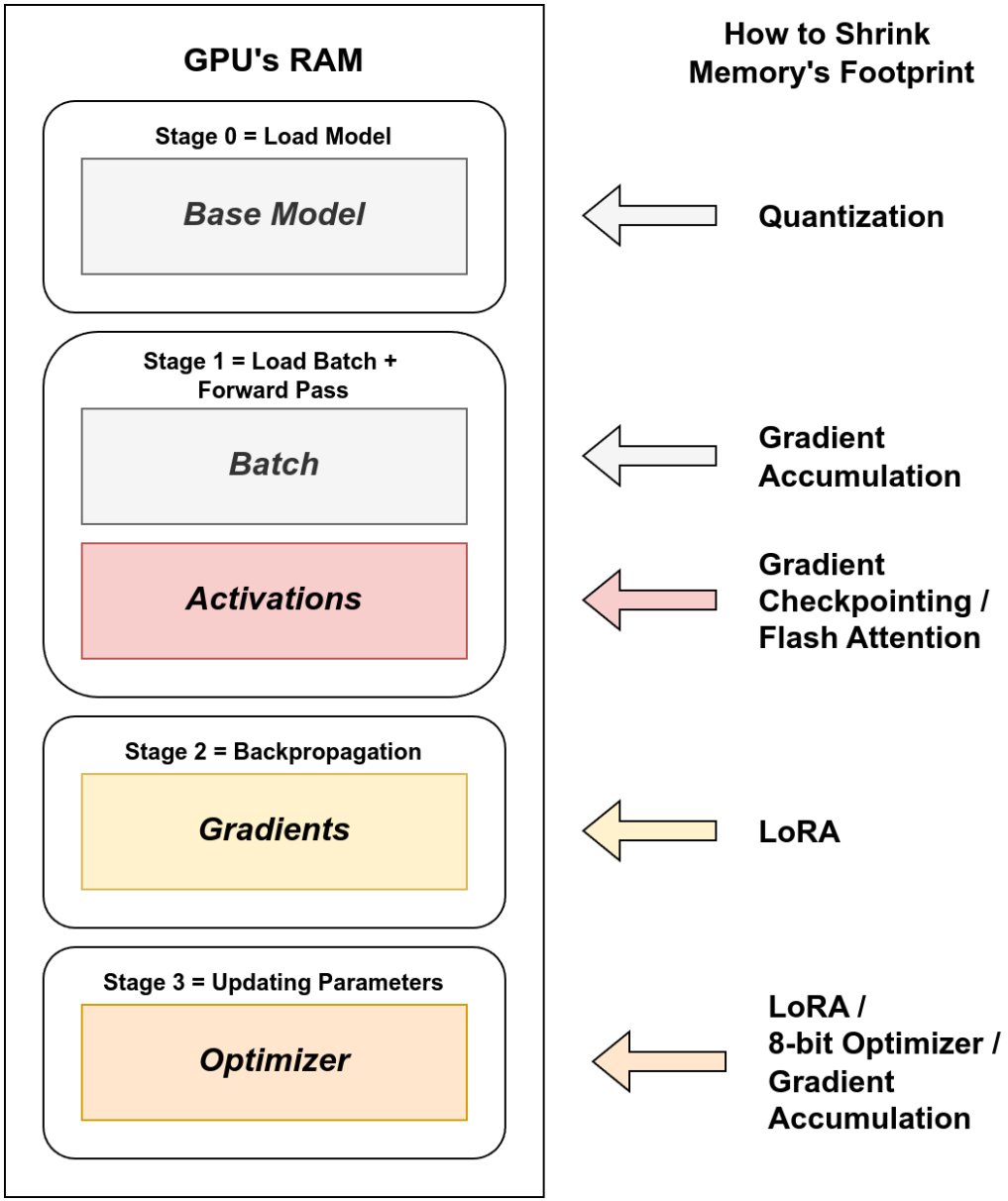

Source: https://huggingface.co/docs/trl/en/sft_trainer

Many Problems!

- ~~Base model takes a lot of space (p)~~ **Quantization**
- ~~Optimizer's states (Adam) takes even more space (2p)~~ **LoRA / 8-bit Adam**
- ~~Gradients take a lot of space (p)~~ **LoRA**
- ~~Activations take a lot of space, especially if the sequences are long~~ **Checkpointing**
- ~~Attention is quadratic on the sequence length (10x longer => 100x more expensive to handle)~~ **Flash Attention**

p = #parameters

Mo'Problems, Mo'Solutions



Mixin’Matching

Batch size = 1, Sequence length = 2000						Attention Implementation		
#	Model	Grad Checkp	LoRA	Optim	Grad Acc	Eager	SDPA	FA2
1	16-bit	No	No	Adam	No	15,071	3,737	3,742
2					Yes	15,071	2,908	2,807
3				8-bit Adam	No	15,071	2,908	2,807
4					Yes	15,071	2,908	2,807
5			Yes	Adam	No	15,266	3,109	3,108
6					Yes	15,266	3,109	3,108
7				8-bit Adam	No	15,266	3,109	3,108
8					Yes	15,266	3,109	3,108
9		Yes	No	Adam	No	3,532	3,537	3,538
10					Yes	2,589	1,667	1,657
11				8-bit Adam	No	2,589	1,667	1,657
12					Yes	2,589	1,667	1,657
13			Yes	Adam	No	2,059	1,603	1,595
14					Yes	2,059	1,603	1,595
15				8-bit Adam	No	2,059	1,603	1,595
16					Yes	2,059	1,603	1,595

Mixin’Matching

OPT-350M – Batch size = 1

#	Model	Grad Checkp	LoRA	Optim	Grad Acc	Seq Len	Mixed Prec	Attention Implementation		
								Eager	SDPA	FA2
1	8-bit	No	Yes	Either	Either	2000	No	17,744	5,544	5,544
2							Yes	20,491	5,294	5,193
3						500	No	2,518	1,717	1,715
4							Yes	2,670	1,692	1,668
5		Yes				2000	No	2,501	2,294	2,275
6							Yes	2,481	2,144	2,126
7						500	No	903	904	903
8							Yes	906	906	905

Falcon 7B – Batch size = 1

#	Model	Grad Checkp	LoRA	Optim	Grad Acc	Seq Len	Mixed Prec	Attention Implementation		
								Eager	SDPA	FA2
1	8-bit	No	Yes	Either	Either	2000	No	n.a.	OOM	OOM
2							Yes	n.a.	OOM	OOM
3						500	No	n.a.	15,050	12,483
4							Yes	n.a.	13,957	12,824
5		Yes				2000	No	n.a.	12,280	11,511
6							Yes	n.a.	12,280	11,511
7						500	No	n.a.	10,369	10,369
8							Yes	n.a.	10,369	10,369

Hands-On

Notebook 3.1 - Fine-tuning Pig Latin

Yoda

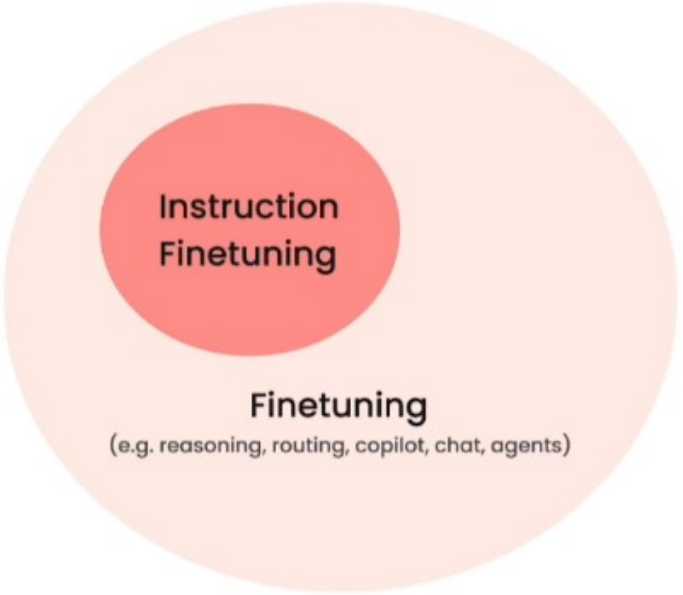
Source: https://lingojam.com/EnglishtoYoda

Exercise

Exercise - Notebook 3.2 - Fine-tuning Yoda

Instruction-Tuning

What is instruction finetuning?



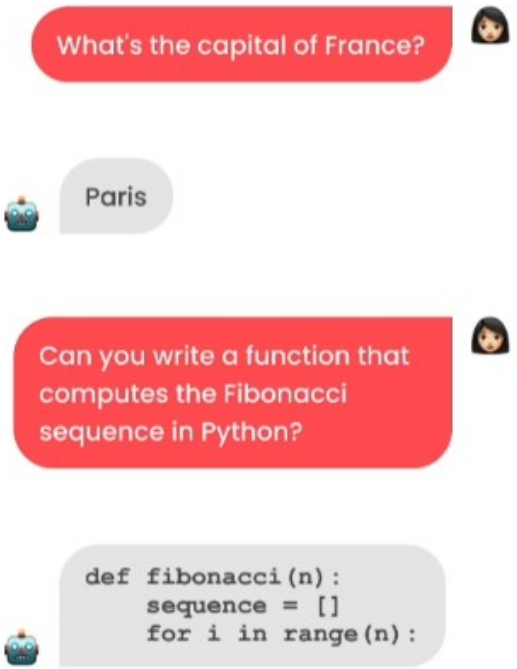
- AKA "instruction-tuned" or "instruction-following" LLMs
- Teaches model to behave more like a chatbot
- Better user interface for model interaction
  - Turned GPT-3 into ChatGPT
  - Increase AI adoption, from thousands of researchers to millions of people

Source: https://www.deeplearning.ai/short-courses/finetuning-large-language-models/

Instruction-Tuning

Instruction Finetuning Generalization

- Can access model's pre-existing knowledge
- Generalize following instructions to other data, not in finetuning dataset



Source: https://www.deeplearning.ai/short-courses/finetuning-large-language-models/

Instruction Datasets

Instruction-following datasets

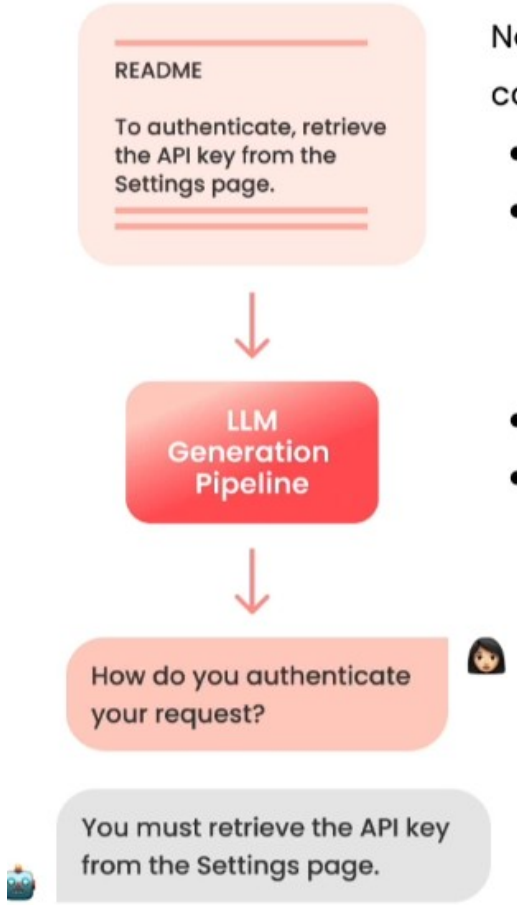
Some existing data is ready as-is, online:

- FAQs
- Customer support conversations
- Slack messages

Steps to prepare your data

1. Collect instruction-response pairs
2. Concatenate pairs (add prompt template, if applicable)
3. Tokenize: Pad, Truncate
4. Split into train/test

LLM Data Generation



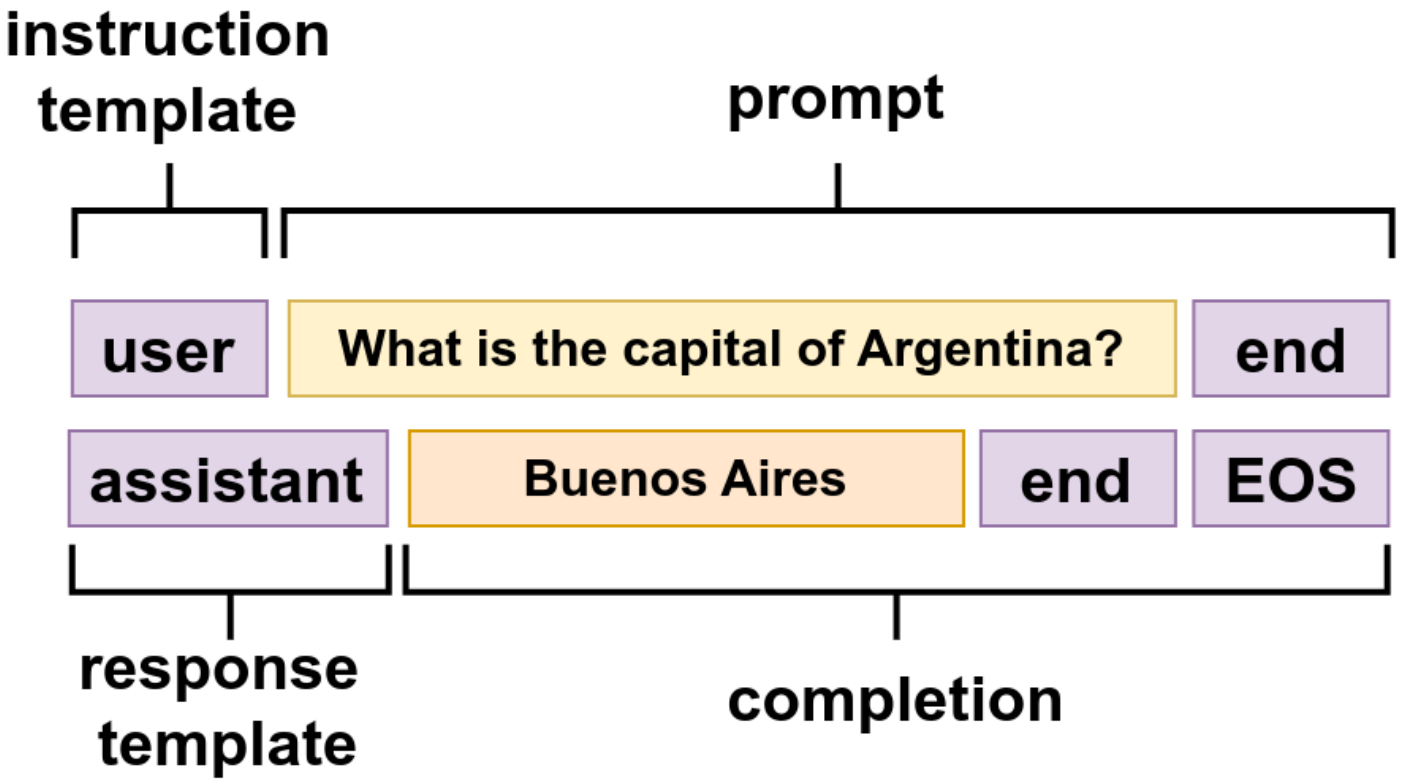
Non-Q&A data can also be converted to Q&A

- Using a prompt template
- Using another LLM

- ChatGPT ("Alpaca")
- Open-source models

Source: https://www.deeplearning.ai/short-courses/finetuning-large-language-models/

Chat Template



Chat Templates

Templates for Chat Models

Introduction

An increasingly common use case for LLMs is **chat**. In a chat context, rather than continuing a single string of text (as is the case with a standard language model), the model instead continues a conversation that consists of one or more **messages**, each of which includes a **role**, like “user” or “assistant”, as well as message text.

Much like tokenization, different models expect very different input formats for chat. This is the reason we added **chat templates** as a feature. Chat templates are part of the tokenizer. They specify how to convert conversations, represented as lists of messages, into a single tokenizable string in the format that the model expects.

```
chat = [
  {"role": "user", "content": "Hello, how are you?"},
  {"role": "assistant", "content": "I'm doing great. How can I help you today?"},
  {"role": "user", "content": "I'd like to show off how chat templating works!"},
]
tokenizer.apply_chat_template(chat, tokenize=False)
```

"<s>[INST] Hello, how are you? [/INST]I'm doing great. How can I help you today?</s> [INST] I'd like to show off how chat templating works! [/INST]"

Source: https://huggingface.co/docs/transformers/main/en/chat_templating

Chat Templates

How do I use chat templates?

As you can see in the example above, chat templates are easy to use. Simply build a list of messages, with `role` and `content` keys, and then pass it to the apply_chat_template() method. Once you do that, you'll get output that's ready to go! When using chat templates as input for model generation, it's also a good idea to use `add_generation_prompt=True` to add a generation prompt.

```
messages = [
    {"role": "system", "content": "You are a friendly chatbot who always
responds in the style of a pirate"},
    {"role": "user", "content": "How many helicopters can a human eat in one
sitting?"},
]
tokenized_chat = tokenizer.apply_chat_template(messages, tokenize=False,
add_generation_prompt=True, return_tensors="pt")
```

```
<|system|>
You are a friendly chatbot who always responds in the style of a
pirate</s>
<|user|>
How many helicopters can a human eat in one sitting?</s>
<|assistant|>
```

Generation prompt

Source: https://huggingface.co/docs/transformers/main/en/chat_templating

Chat Templates

What template should I use?

When setting the template for a model that's already been trained for chat, you should ensure that the template exactly matches the message formatting that the model saw during training, or else you will probably experience performance degradation. This is true even if you're training the model further - you will probably get the best performance if you keep the chat tokens constant. This is very analogous to tokenization - you generally get the best performance for inference or fine-tuning when you precisely match the tokenization used during training.

If you're training a model from scratch, or fine-tuning a base language model for chat, on the other hand, you have a lot of freedom to choose an appropriate template! LLMs are smart enough to learn to handle lots of different input formats. One popular choice is the ChatML format, and this is a good, flexible choice for many use-cases. It looks like this:

```
{% for message in messages %}
    {{'<|im_start|>' + message['role'] + '\n' + message['content'] + '<|im_end|>' + '\n'}}
{% endfor %}
```

<|im_start|>system
You are a helpful chatbot that will do its best not to say anything so stupid that people tweet about it.**<|im_end|>**
<|im_start|>user
How are you?**<|im_end|>**
<|im_start|>assistant
I'm doing great!**<|im_end|>**

Source: https://huggingface.co/docs/transformers/main/en/chat_templating

Hands-On

Notebook 6 - Instruction-tuning Pythia

The End

THANK YOU!