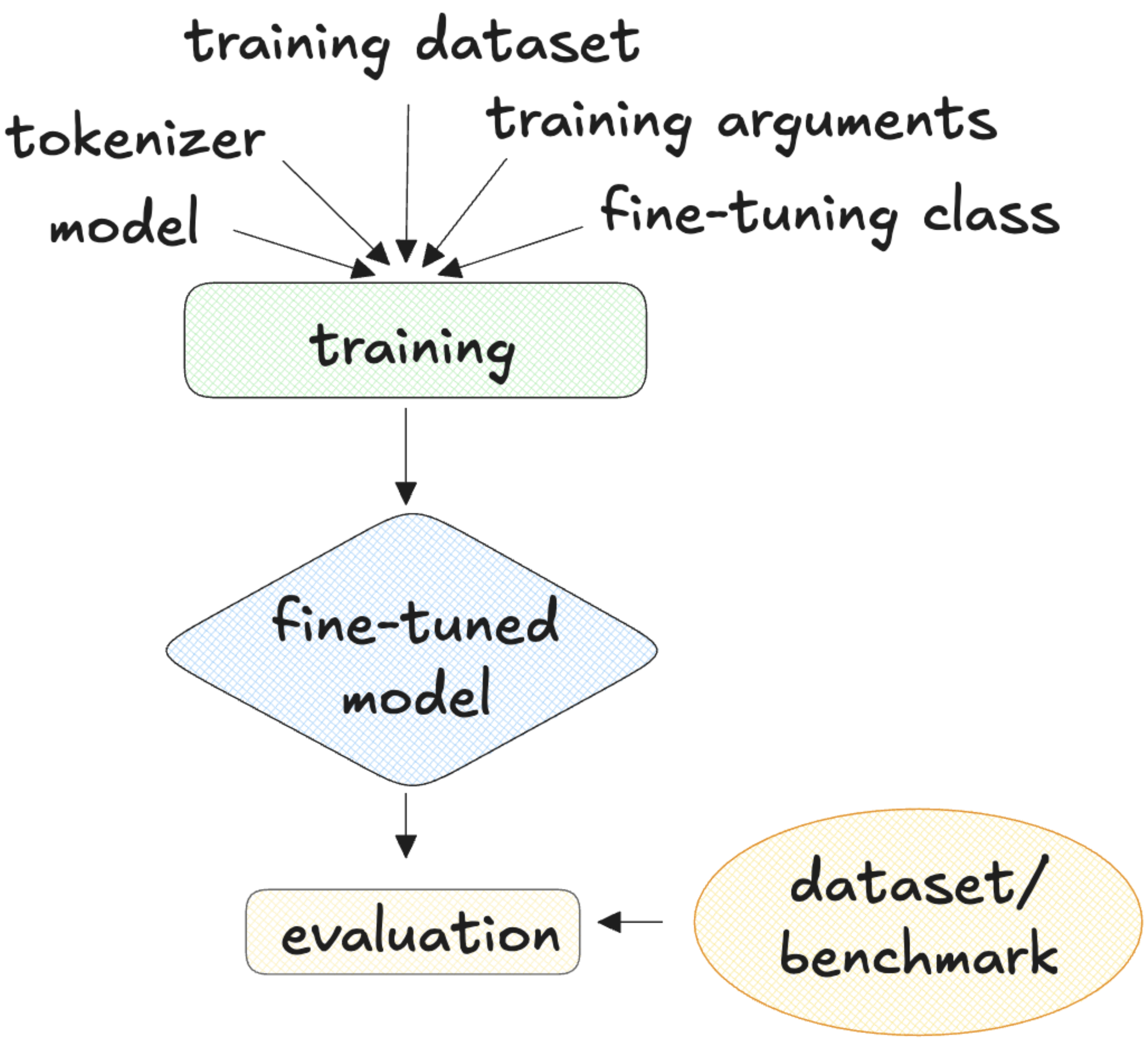
Model fine-tuning with Hugging Face

FINE-TUNING WITH LLAMA 3

Francesca Donadoni
Curriculum Manager, DataCamp

What do we need to conduct fine-tuning?

1. Language model + tokenizer (a **LLama** model, such as TinyLLama-v0)
2. Training dataset (**the Bitext customer service dataset**)
3. Training arguments
4. Conduct fine-tuning (**SFTTrainer** from **TRL**)
5. Evaluation benchmark or dataset



How to load models and tokenizers with Auto classes

```
model_name="Maykeye/TinyLLama-v0"

model = AutoModelForCausalLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
```

[1] https://huggingface.co/docs/transformers/main/en/model_doc/auto

Defining training parameters with TrainingArguments

```
training_arguments = TrainingArguments(
    per_device_train_batch_size=1,
    learning_rate=2e-3,
    max_grad_norm=0.3,
    max_steps=200,
    ...
    gradient_accumulation_steps=2,
    save_steps=10,
)
```

[1] https://huggingface.co/docs/transformers/v4.40.1/en/main_classes/trainer#transformers.TrainingArguments

How to set up training with SFTTrainer

```
trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset,
    dataset_text_field='conversation',
    max_seq_length=250,
    args=training_arguments
)
```

Understanding fine-tuning results with SFTTrainer

```
trainer.train()
```

```
TrainOutput(global_step=200, training_loss=1.9401231002807617,
            metrics={'train_runtime': 142.5501,
                     'train_samples_per_second': 2.806,
                     'train_steps_per_second': 1.403,
                     'total_flos': 1461265827840.0,
                     'train_loss': 1.9401231002807617,
                     'epoch': 2.0})
```

How to evaluate a trained model Using ROUGE-1

- ROUGE-1: Ratio of word overlap between a reference and generated text

```python
import evaluate
rouge = evaluate.load('rouge')
predictions = ["hello there", "general kenobi"]
references = ["hello there", "master yoda"]
results = rouge.compute(predictions=predictions, references=references)
print(results)
```

```
{'rouge1': 0.5, 'rouge2': 0.5, 'rougeL': 0.5, 'rougeLsum': 0.5}
```

[1] https://huggingface.co/spaces/evaluate-metric/rouge

How to use the ROUGE-1 score

1. Use the evaluation set in `evaluation_dataset`

```
def generate_predictions_and_reference(dataset):
    predictions = []
    references = []
    for row in dataset:
        inputs = tokenizer.encode(row["instruction"], return_tensors="pt")
        outputs = model.generate(inputs)
        decoded_outputs = tokenizer.decode(outputs[0, inputs.shape[1]:], skip_special_tokens = True)
        references += [row["response"]]
        predictions += [decoded_outputs]
    return references, predictions
```

How to run ROUGE-1 on an evaluation set

```
references, predictions = generate_predictions_and_reference(evaluation_dataset)
```

```
rouge = evaluate.load('rouge')
results = rouge.compute(predictions=predictions, references=references)
```

```
print(results)
```

Finetuning vs no finetuning

Fine-tuned

```
{'rouge1': 0.22425812699023645,
 'rouge2': 0.039502543246449,
 'rougeL': 0.1501513006868983,
 'rougeLsum': 0.18685597710721613}
```

No fine-tuning

```
{'rouge1': 0.1310928764315105,
 'rouge2': 0.04581654122835097,
 'rougeL': 0.08415351421221628,
 'rougeLsum': 0.1224749866097021}
```

Let's practice!

FINE-TUNING WITH LLAMA 3

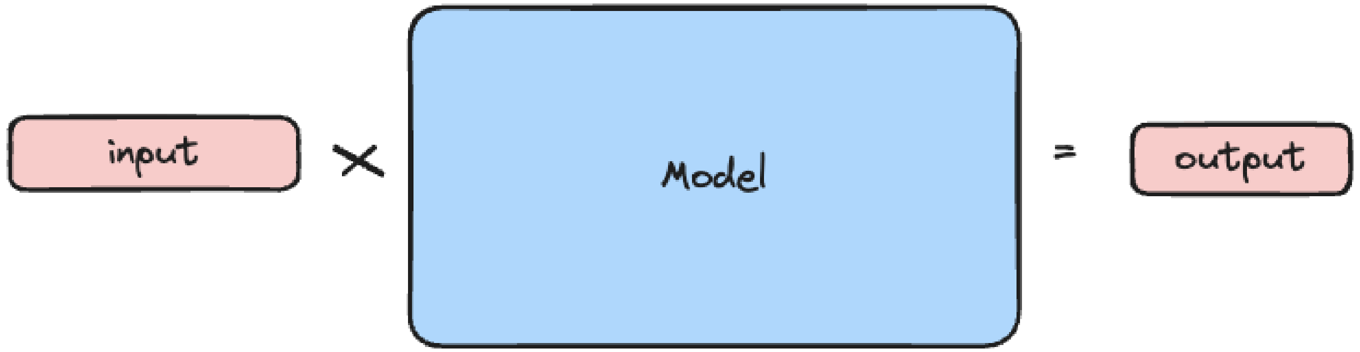
Efficient fine-tuning with LoRA

FINE-TUNING WITH LLAMA 3

Francesca Donadoni
Curriculum Manager, DataCamp

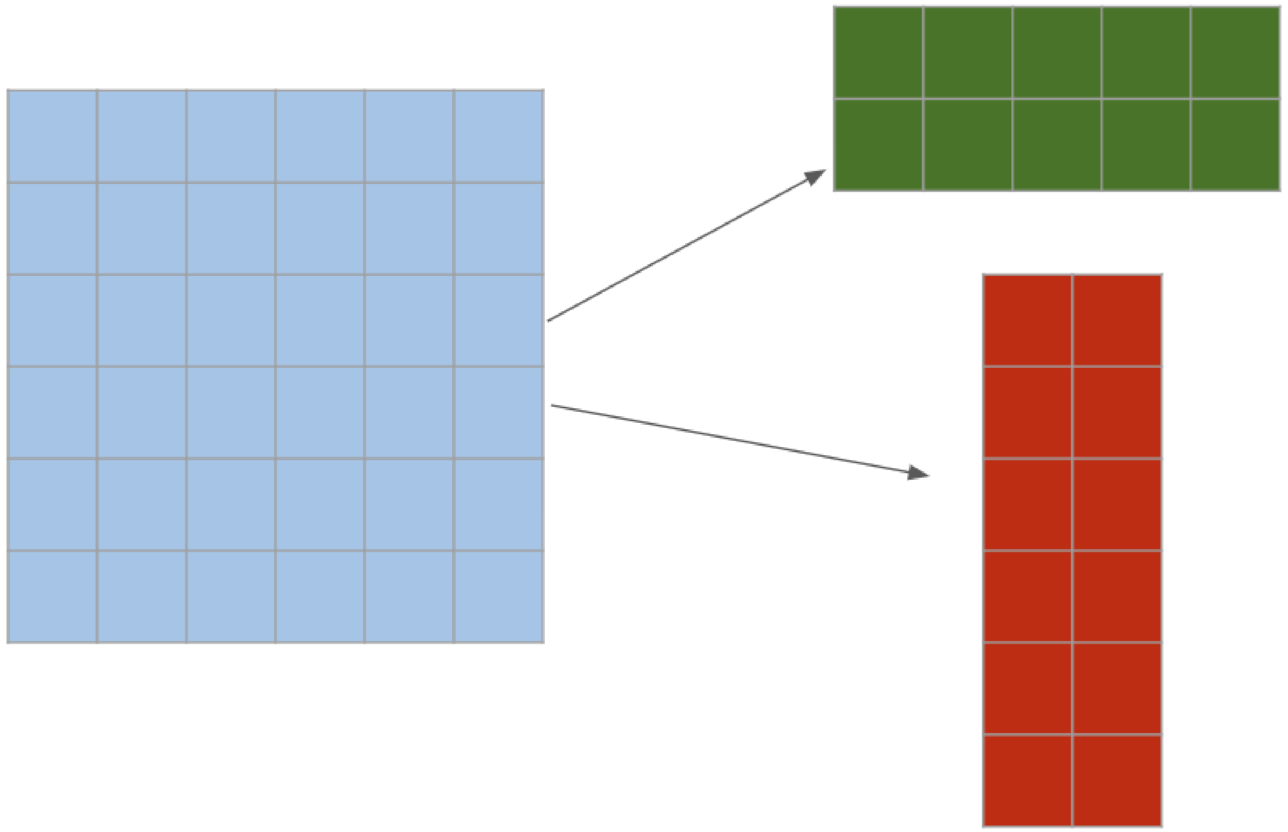
What happens when we train a model?

- Tokens are input data forming a vector
- Matrix (model) multiplication
- Results in output vectors
- Errors are used to update model weights
- Model size determines training difficulty



What is LoRA

- Low-rank Decomposition
- Reduces training parameters
- Maintains performance
- Regularization effect



How to implement LoRA using PEFT

```
from peft import LoraConfig
lora_config = LoraConfig(
    r=12,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=['q_proj', 'v_proj']
)
```

Integrating LoRA configuration in training

```
trainer = SFTTrainer(
    model=model,
    train_dataset=ds,
    max_seq_length=250,
    dataset_text_field='conversation',
    tokenizer=tokenizer,
    args=training_arguments
    peft_config=lora_config,
)
trainer.train()
```

LoRA vs regular finetuning

- `TinyLlama/TinyLlama-1.1B-Chat-v1.0`
- 1.1 billion parameters
- 11k samples
- ~30 minutes

- `nvidia/Llama3-ChatQA-1.5-8B`
- 8 billion parameters
- 11k samples
- ~30 minutes

Let's practice!

FINE-TUNING WITH LLAMA 3

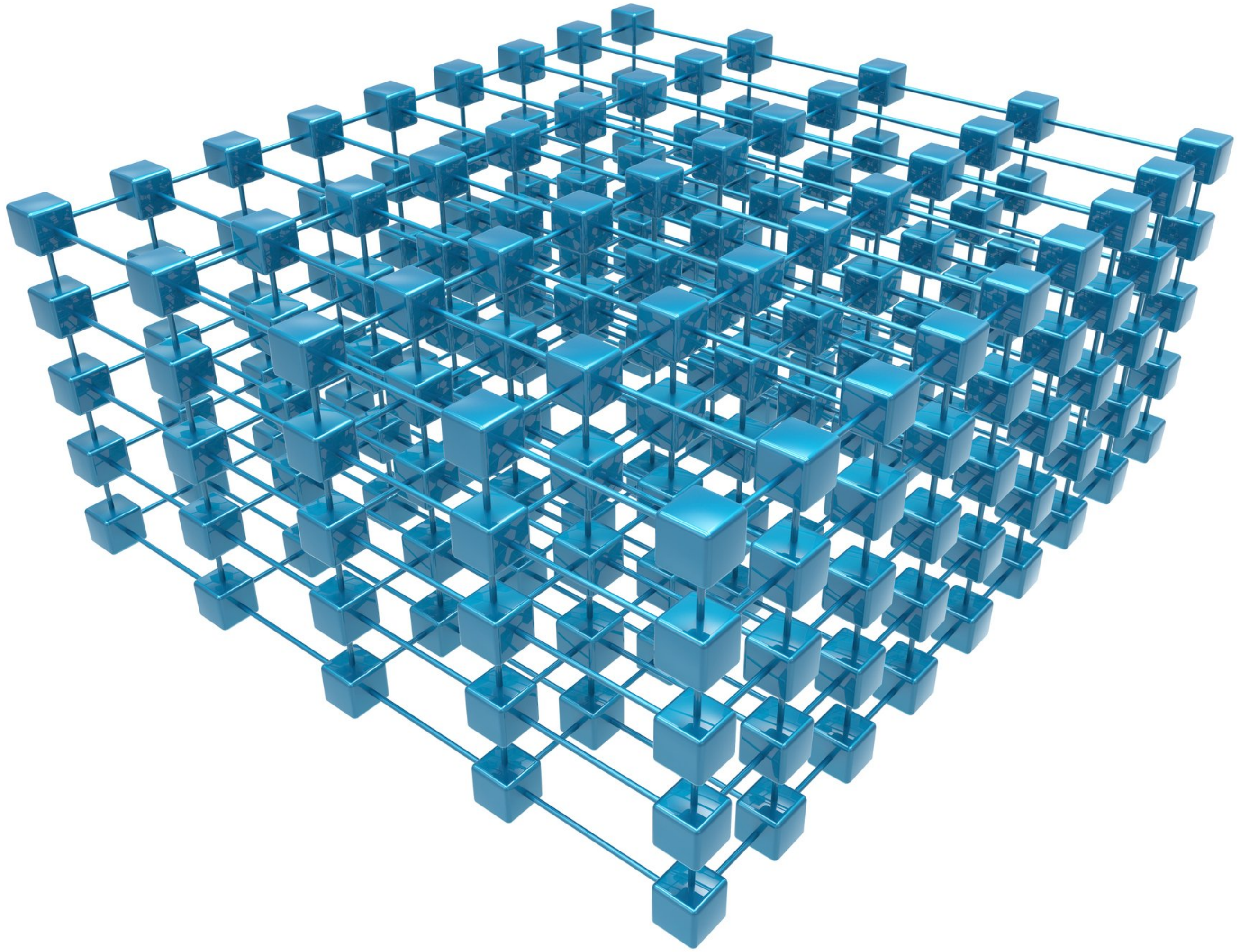
Making models smaller with quantization

FINE-TUNING WITH LLAMA 3

Francesca Donadoni
Curriculum Manager, DataCamp

What is quantization?

- Reducing model precision
- 32-bit float to:
  - 8-bit integer
  - 4-bit integer
- Quantization-aware training



Types of quantization

- **Weight quantization:** reduce weight precision
- **Activation quantization:** reduces precision of activation values
- **Post-Training Quantization:** reduce model precision after training

Configuring quantization with bitsandbytes

```
from transformers import BitsAndBytesConfig
bnb_config = BitsAndBytesConfig(
```

- set **precision** (load_in_4_bit, load_in_8_bit)

```
    load_in_4bit=True,
```

- set **quantization type** ('fp4' or 4-bit float, 'nf4' or normalized 4-bit float)

```
    bnb_4bit_quant_type="nf4",
```

- set **compute precision** (32-bit float or 16-bit bfloat)

```
    bnb_4bit_compute_dtype=torch.bfloat16)
```

Loading model with quantization

```python
from transformers import BitsAndBytesConfig, AutoModelForCausalLM

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
model = AutoModelForCausalLM.from_pretrained(
    "nvidia/Llama3-ChatQA-1.5-8B",
    quantization_config=bnb_config
)
```

Using a quantized model

```
promptstr = """System: You are a helpful chatbot who answers questions about planets.
User: Explain the history of Mars
Assistant: """
inputs = tokenizer.encode(promptstr, return_tensors="pt")
outputs = model.generate(inputs, max_length=200)
decoded_outputs = tokenizer.decode(outputs[0, inputs.shape[1]:], skip_special_tokens = True)
print(decoded_outputs)
```

```
Here is a brief history of Mars:
- 4.6 billion years ago: Mars formed as part of the solar system.
- 3.8 billion years ago: Mars had a thick atmosphere and liquid water on its surface.
- 3.8 billion years ago to 3.5 billion years ago: Mars lost its magnetic field and atmosphere,
and became a cold, dry planet.
- 3.5 billion years ago to present: Mars has been cold and dry, with a thin atmosphere.
```

Finetuning a quantized model

- Full quantization does not support fine-tuning
- LoRA adaptation

```
trainer = SFTTrainer(
    model=model,
    peft_config=peft_config,
    train_dataset=ds,
    max_seq_length=250,
    dataset_text_field='conversation',
    tokenizer=tokenizer,
    args=training_arguments
)
trainer.train()
```

Let's practice!

FINE-TUNING WITH LLAMA 3

Congratulations!

FINE-TUNING WITH LLAMA 3

Francesca Donadoni
Curriculum Manager, DataCamp

Your achievements

- Fine-tuning to improve and customize model performance

Chapter 1:

- Data preparation
- Fine-tuning TorchTune recipes
- Custom configuration

Chapter 2:

- Optimal hardware usage
- LoRA
- Quantization

Your achievements

- Fine-tuning to improve and customize model performance

Chapter 1:

- Data preparation
- Fine-tuning TorchTune recipes
- Custom configuration

Chapter 2:

- Optimal hardware usage
- LoRA
- Quantization

Keep learning!

FINE-TUNING WITH LLAMA 3