Preparing for fine-tuning

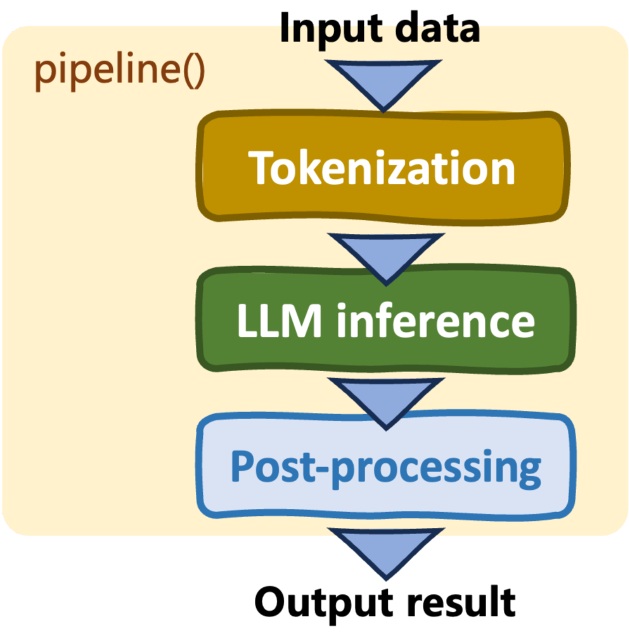
INTRODUCTION TO LLMS IN PYTHON

Jasmin Ludolf
Senior Data Science Content Developer, DataCamp

Pipelines and auto classes

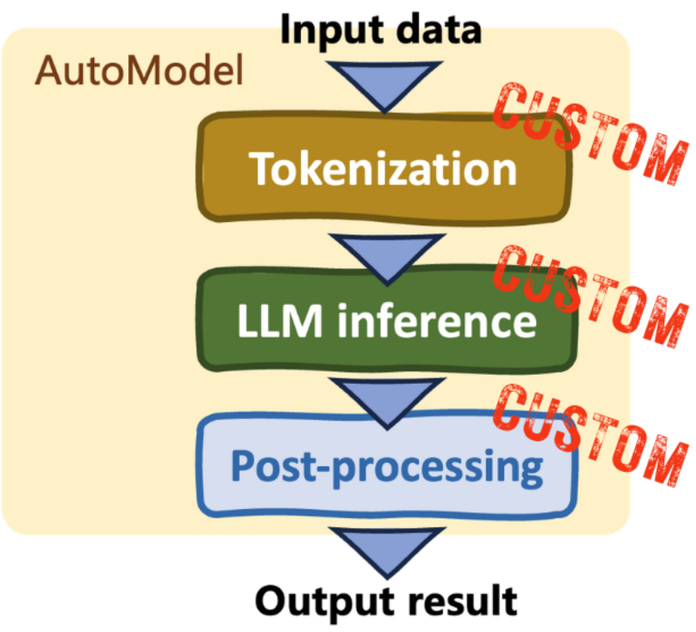
Pipelines: `pipeline()`

- Streamlines tasks
- Automatic model and tokenizer selection
- Limited control

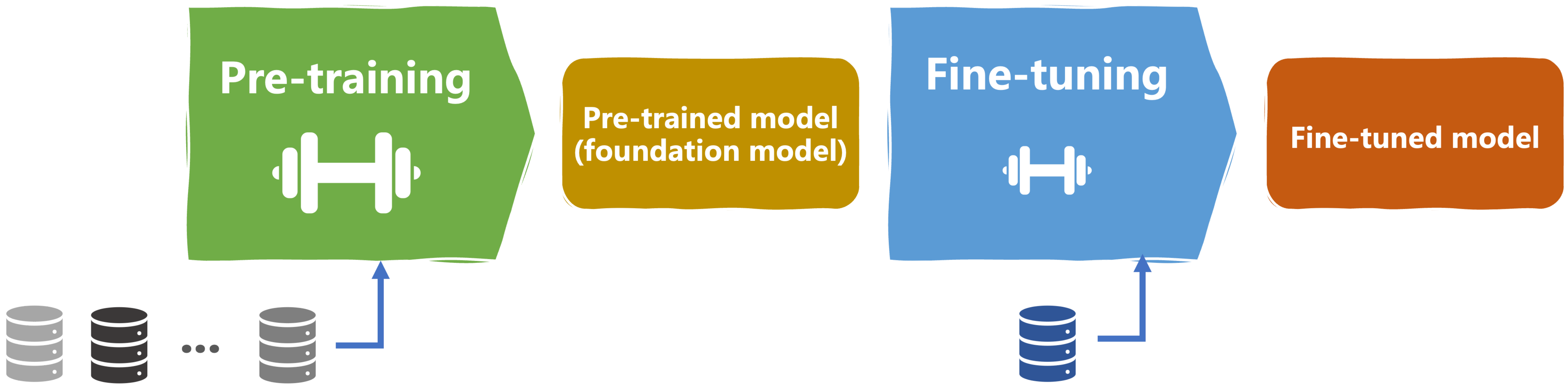


Auto classes (`AutoModel` **class)**

- Customization
- Manual adjustments
- Supports fine-tuning



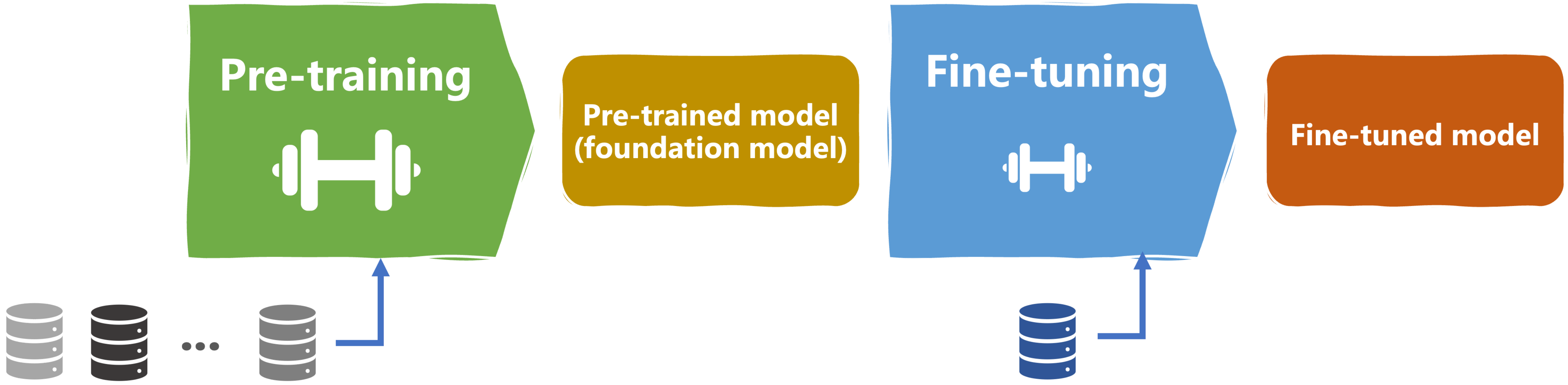
LLM lifecycle



Pre-training

- Broad data
- Learn general patterns

LLM lifecycle



Pre-training

- Broad data
- Learn general patterns

Fine-tuning

Domain specific

Specialized tasks

Loading a dataset for fine-tuning

```
from datasets import load_dataset

train_data = load_dataset("imdb", split="train")
train_data = data.shard(num_shards=4, index=0)

test_data = load_dataset("imdb", split="test")
test_data = data.shard(num_shards=4, index=0)
```

- `load_dataset()` : loads a dataset from Hugging Face hub
  - **imdb**: review classification

Auto classes

```
from transformers import AutoModel, AutoTokenizer
from transformers import AutoModelForSequenceClassification


model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased")
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

Tokenization

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from datasets import load_dataset

train_data = load_dataset("imdb", split="train")
train_data = data.shard(num_shards=4, index=0)
test_data = load_dataset("imdb", split="test")
test_data = data.shard(num_shards=4, index=0)

model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased")
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Tokenize the data
tokenized_training_data = tokenizer(train_data["text"], return_tensors="pt", padding=True, truncation=True,
                                    max_length=64)

tokenized_test_data = tokenizer(test_data["text"], return_tensors="pt", padding=True, truncation=True,
                                max_length=64)
```

Tokenization output

```
print(tokenized_training_data)
```

```
{'input_ids': tensor([[  101,  1045, 12524,  1045,  2572,  8025,  1011,  3756,
2013,  2026, 2678,  3573,  2138,  1997,  2035,  1996,  6704,  2008,  5129,  2009,
2043,  2009, 2001,  2034,  2207,  1999,  3476,  1012,  1045,  2036, ...
```

Tokenizing row by row

```
def tokenize_function(text_data):
    return tokenizer(text_data["text"], return_tensors="pt", padding=True, truncation=True, max_length=64)

# Tokenize in batches
tokenized_in_batches = train_data.map(tokenize_function, batched=True)

# Tokenize row by row
tokenized_by_row = train_data.map(tokenize_function, batched=False)
```

```
Dataset({
    features: ['text', 'label', 'input_ids', 'token_type_ids', 'attention_mask'],
    num_rows: 1563
})
```

Subword tokenization

- Common in modern tokenizers
- Words split into meaningful sub-parts

Unbelievably

Subword tokenization

- Common in modern tokenizers
- Words split into meaningful sub-parts

$\boxed{\text{Un}}\boxed{\text{believ}}\boxed{\text{ably}}$

Let's practice!

INTRODUCTION TO LLMS IN PYTHON

Fine-tuning through training

INTRODUCTION TO LLMS IN PYTHON

Jasmin Ludolf
Senior Data Science Content Developer, DataCamp

Training Arguments

```
from transformers import Trainer,
TrainingArguments

training_args = TrainingArguments(
  output_dir="./finetuned",
  evaluation_strategy="epoch",
  num_train_epochs=3,
  learning_rate=2e-5,



)
```

- `TrainingArguments()` : customize training settings
- See documentation for all parameters
- Values depend on use, dataset, speed
- `output_dir` : output directory
- `eval_strategy` : when to evaluate "epoch", "steps", or "none"
- `num_train_epochs` : number of training epochs
- `learning_rate` : for optimizer

Training Arguments

```
from transformers import Trainer,
TrainingArguments

training_args = TrainingArguments(
  output_dir="./finetuned",
  evaluation_strategy="epoch",
  num_train_epochs=3,
  learning_rate=2e-5,
  per_device_train_batch_size=8,
  per_device_eval_batch_size=8,
  weight_decay=0.01,
)
```

- `per_device_train_batch_size` and `per_device_eval_batch_size` define the batch size
- `weight_decay` : applied to the optimizer to avoid overfitting

Trainer class

```
from transformers import Trainer,
TrainingArguments

training_args = TrainingArguments(...)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_training_data,
    eval_dataset=tokenized_test_data,
    tokenizer=tokenizer
)

trainer.train()
```

- `model` : the model to fine-tune
- `args` : the training arguments
- `train_dataset` : the data used for training
- `eval_dataset` : the data used for evaluation
- `tokenizer` : the tokenizer

Number of training loops: Dataset size, `num_train_epochs` , `per_device_train_batch_size` and `per_device_eval_batch_size`

Trainer output

```
{'eval_loss': 0.398524671792984, 'eval_runtime': 33.3145, 'eval_samples_per_second': 46.916,
'eval_steps_per_second': 5.883, 'epoch': 1.0}
{'eval_loss': 0.1745782047510147, 'eval_runtime': 33.5202, 'eval_samples_per_second': 46.629,
'eval_steps_per_second': 5.847, 'epoch': 2.0}
{'loss': 0.4272, 'grad_norm': 15.558795928955078, 'learning_rate': 2.993197278911565e-06,
'epoch': 2.5510204081632653}
{'eval_loss': 0.12216147780418396, 'eval_runtime': 33.2238, 'eval_samples_per_second': 47.045,
'eval_steps_per_second': 5.899, 'epoch': 3.0}
{'train_runtime': 673.0528, 'train_samples_per_second': 6.967, 'train_steps_per_second': 0.874,
'train_loss': 0.4002853834710153, 'epoch': 3.0}
TrainOutput(global_step=588, training_loss=0.4002853834710153, metrics={'train_runtime': 673.0528,
'train_samples_per_second': 6.967, 'train_steps_per_second': 0.874,
'train_loss': 0.4002853834710153, 'epoch': 3.0})
```

Using the fine-tuned model

```
new_data = ["This is movie was disappointing!", "This is the best movie ever!"]

new_input = tokenizer(new_data, return_tensors="pt", padding=True, truncation=True, max_length=64)

with torch.no_grad():
    outputs = model(**new_input)

predicted_labels = torch.argmax(outputs.logits, dim=1).tolist()

label_map = {0: "NEGATIVE", 1: "POSITIVE"}
for i, predicted_label in enumerate(predicted_labels):
    sentiment = label_map[predicted_label]
    print(f"\nInput Text {i + 1}: {new_data[i]}")
    print(f"Predicted Label: {sentiment}")
```

Fine-tuning results

```
Input Text 1: This is movie was disappointing!
Predicted Sentiment: NEGATIVE


Input Text 2: This is the best movie ever!
Predicted Sentiment: POSITIVE
```

Saving models and tokenizers

```
model.save_pretrained("my_finetuned_files")
```

```
tokenizer.save_pretrained("my_finetuned_files")
```

```
# Loading a saved model
model = AutoModelForSequenceClassification.from_pretrained("my_finetuned_files")
tokenizer = AutoTokenizer.from_pretrained("my_finetuned_files")
```

Let's practice!

INTRODUCTION TO LLMS IN PYTHON

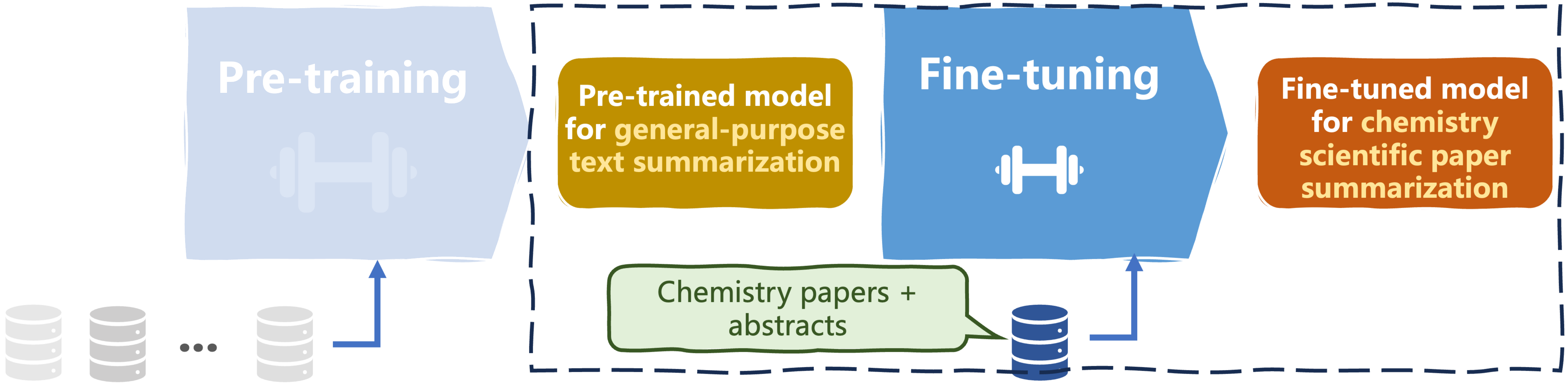
Fine-tuning approaches

INTRODUCTION TO LLMS IN PYTHON

Jasmin Ludolf

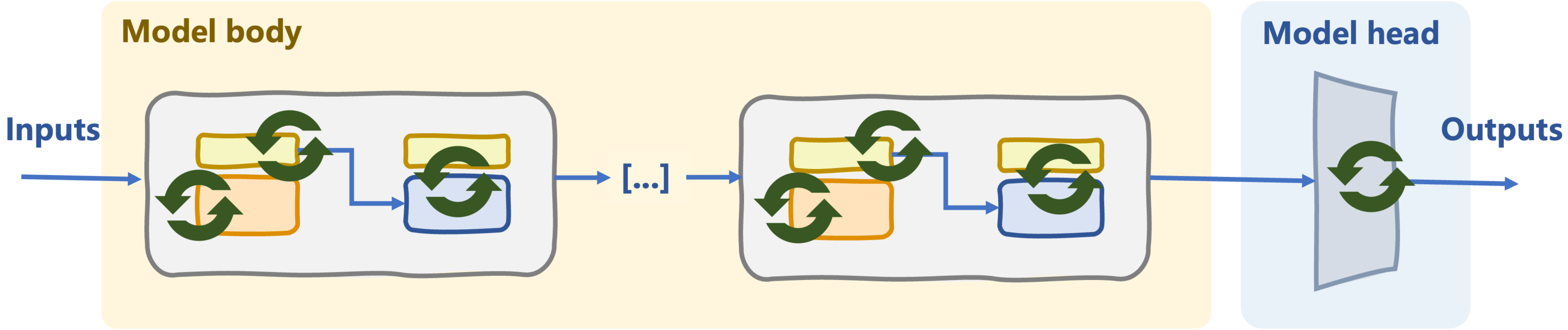
Senior Data Science Content Developer, DataCamp

Fine-tuning


Pre-training
Pre-trained model for general-purpose text summarization
Fine-tuning
Fine-tuned model for chemistry scientific paper summarization
Chemistry papers + abstracts

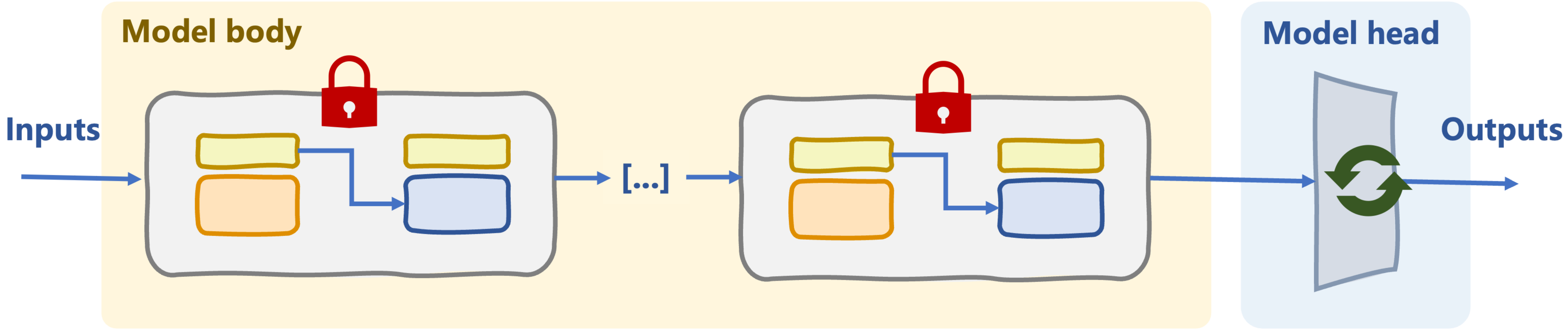
Full fine-tuning

- The entire model weights are updated
- Computationally expensive



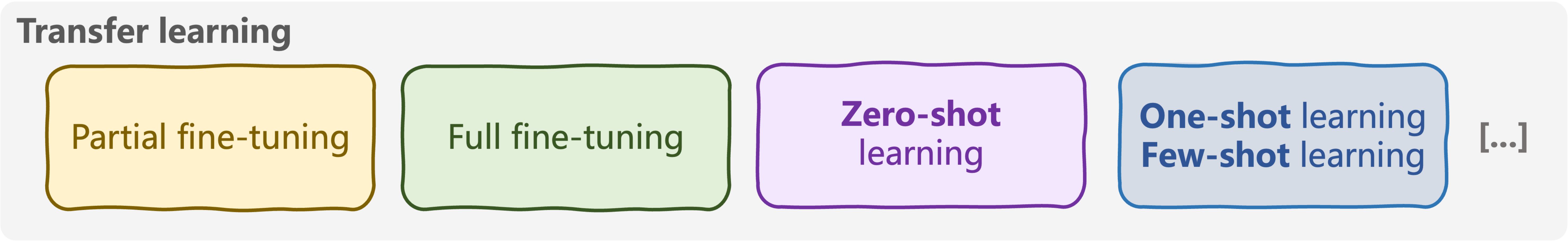
Partial fine-tuning

- Some layers are fixed
- Only task-specific layers are updated



Transfer learning

- A pre-trained model is adapted to a different but related task
- Leverages knowledge from one domain to a related one



N-shot learning

- Zero-shot learning: no examples
- One-shot learning: one example
- Few-shot learning: several examples

One-shot learning

```
from transformers import pipeline

generator = pipeline(task="sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-englisl

input_text = """
Classify the sentiment of this sentence as either Positive or Negative.
Example:
Text: "I'm feeling great today!" Sentiment: Positive
Text: "The weather today is lovely." Sentiment:
"""

result = generator(input_text, max_length=100)
print(result[0]["label"])
```

```
POSITIVE
```

Let's practice!

INTRODUCTION TO LLMS IN PYTHON