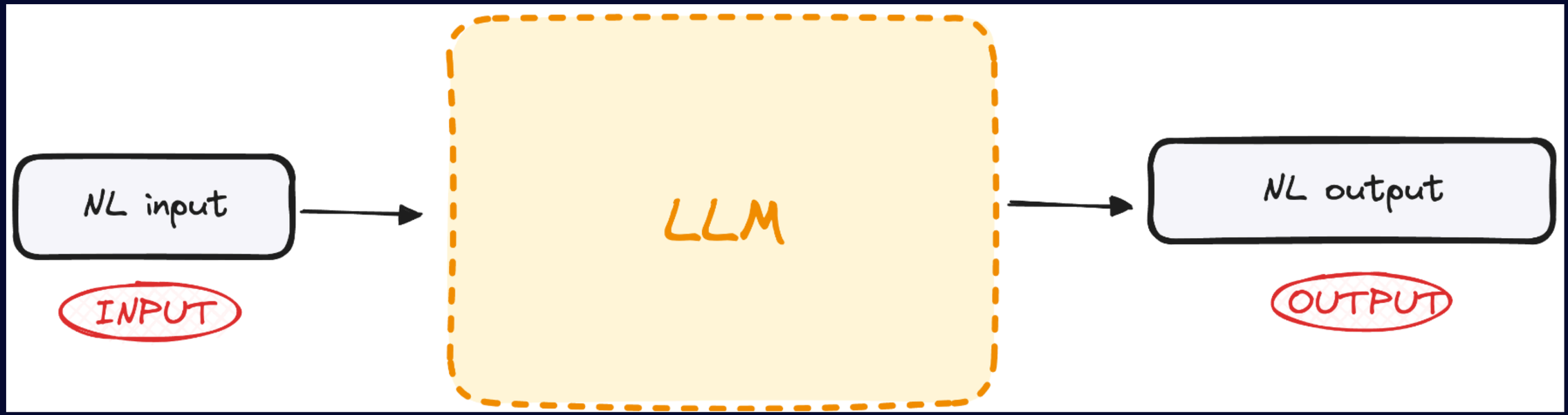
LARGE LANGUAGE MODELS AND PRE-TRAINED MODELS

LLMs are ML algorithms designed to predict subsequent words in sentences.

Examples: BERT, Mistral7B, trained on vast text data to understand language fundamentals.

Based on the Transformers architecture, utilizing self-attention mechanisms.



Model Performance and Fine-tuning

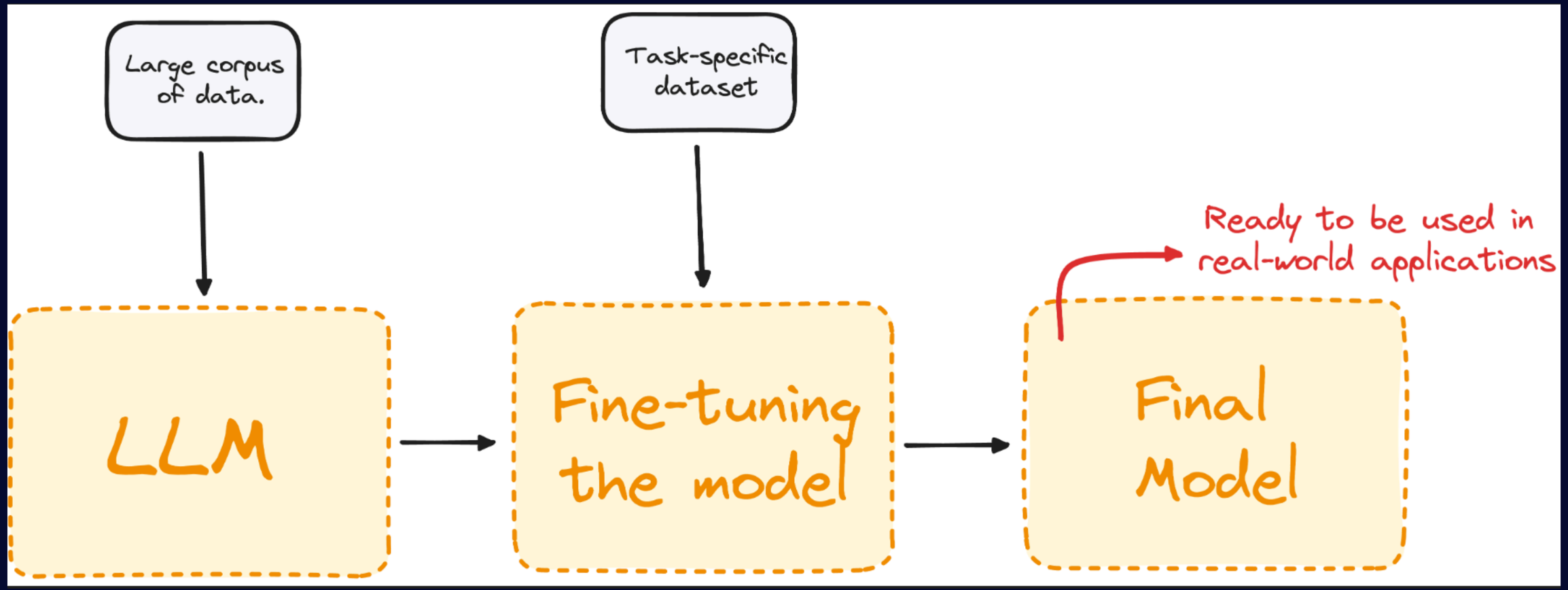
Models excel at understanding natural language and generating human-like text.

General performance is strong; however, domain-specific tasks can be challenging.

Fine-tuning involves additional training on a domain-specific dataset.

Benefits include reduced computational costs and enhanced model versatility.

https://www.linkedin.com/in/zartashiaafzal/



https://www.linkedin.com/in/zartashiaafzal/

Types of Fine-tuning

01 Few-shot learning: Provides initial examples for better contextual understanding.

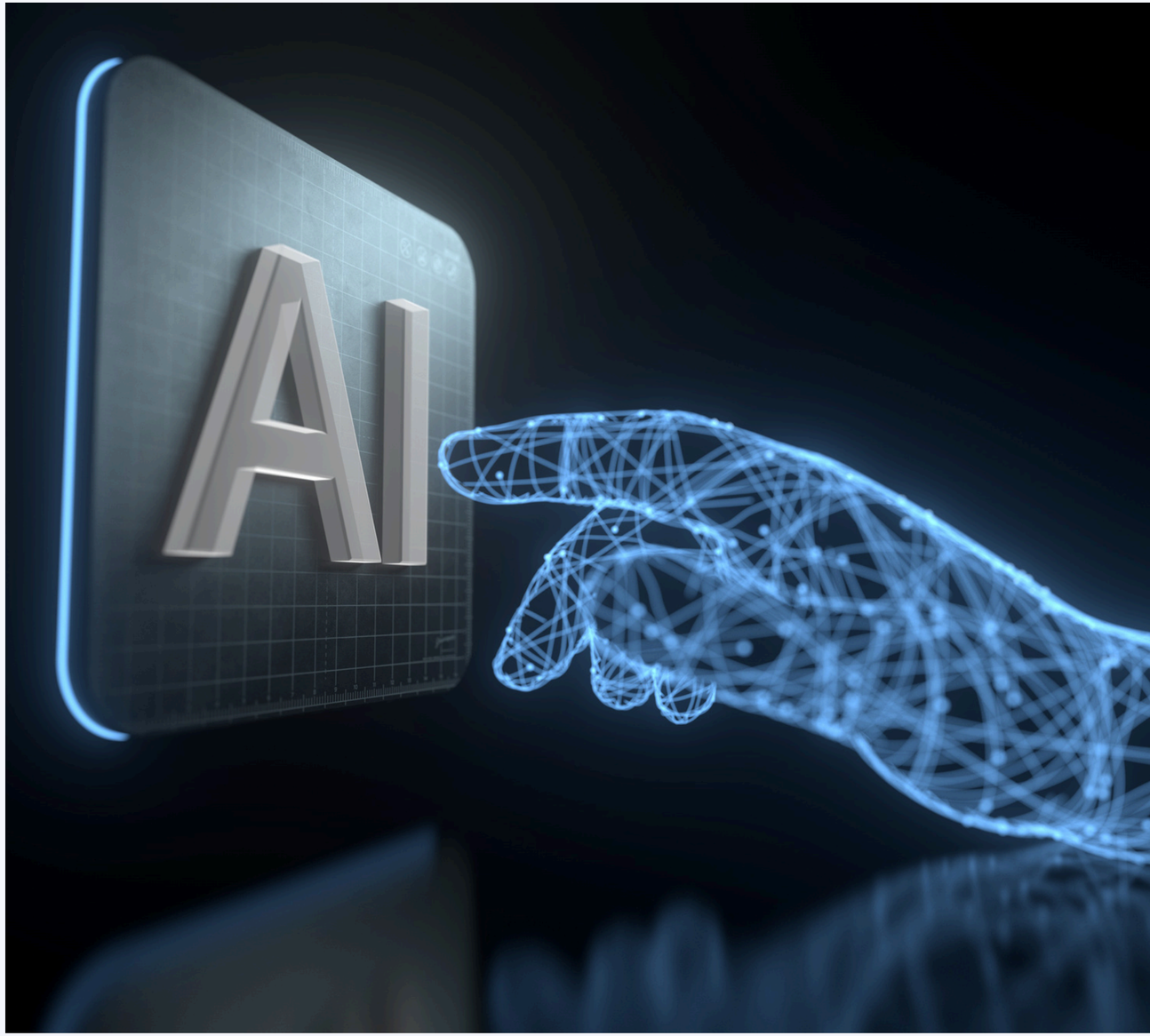
02 Transfer learning: Applies broad model knowledge to specific tasks.

03 Domain-specific fine-tuning: Adapts model for specific industry needs.

04 Supervised fine-tuning: Uses labeled data for targeted training tasks.

Introduction to Hugging Face

- An open-source AI company offering tools and models for NLP tasks.
- Provides access to state-of-the-art ML models like the Transformers library.
- Supports community-driven AI development through collaboration on models and datasets.

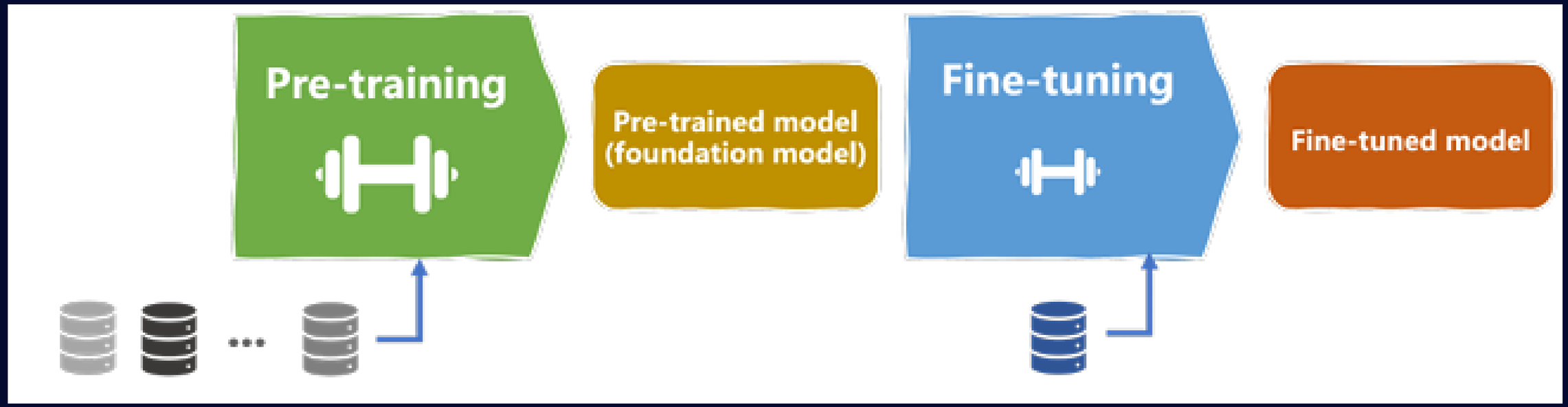


Generic Modules of the Transformers Library

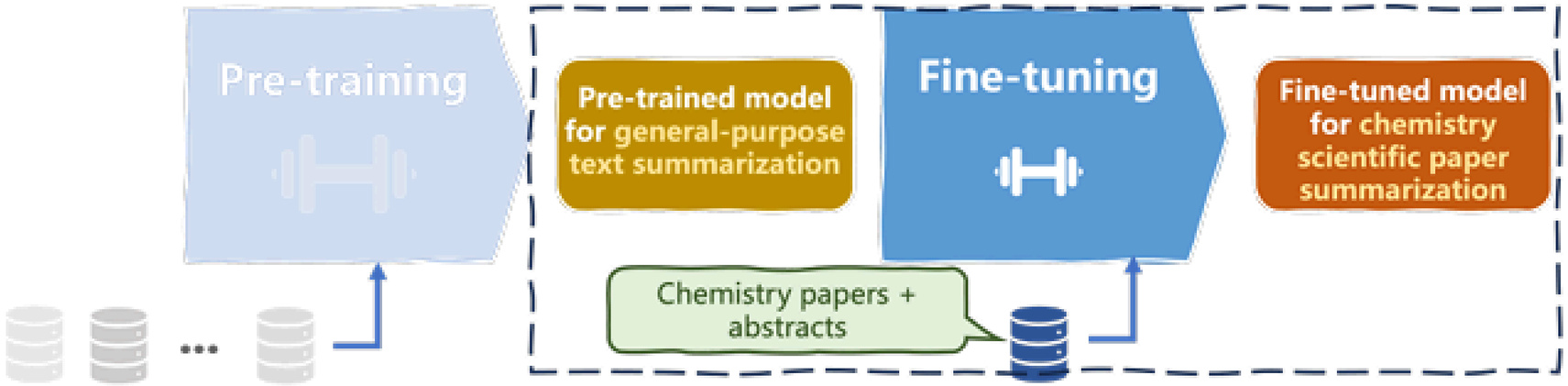
- Model: Supports a variety of NLP tasks with different architectures.
- Tokenizer: Handles conversion of text to tokens.
- Pipeline: Simplifies performing NLP tasks with pre-trained models.
- Trainer: Aids in training and fine-tuning models.

LLM fine-tuning and transfer learning

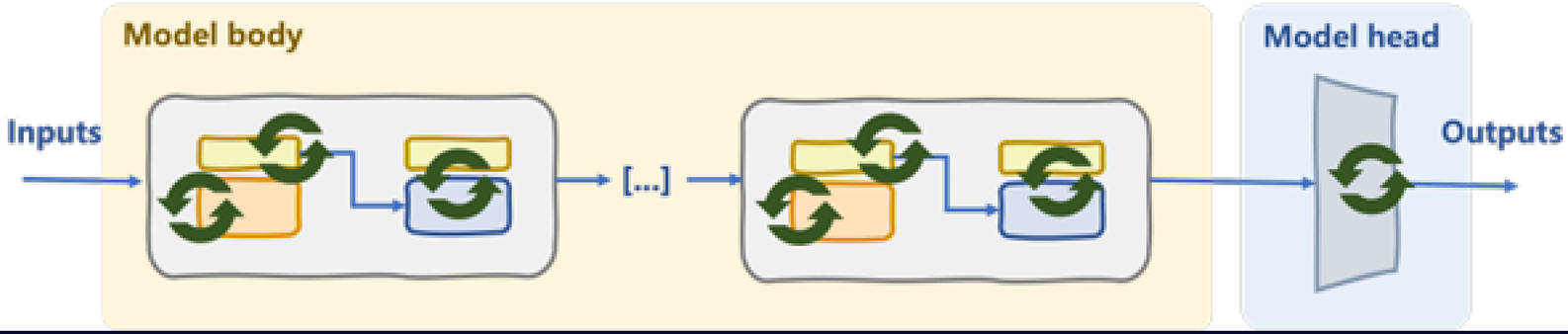
OVERVIEW OF LLM LIFECYCLE



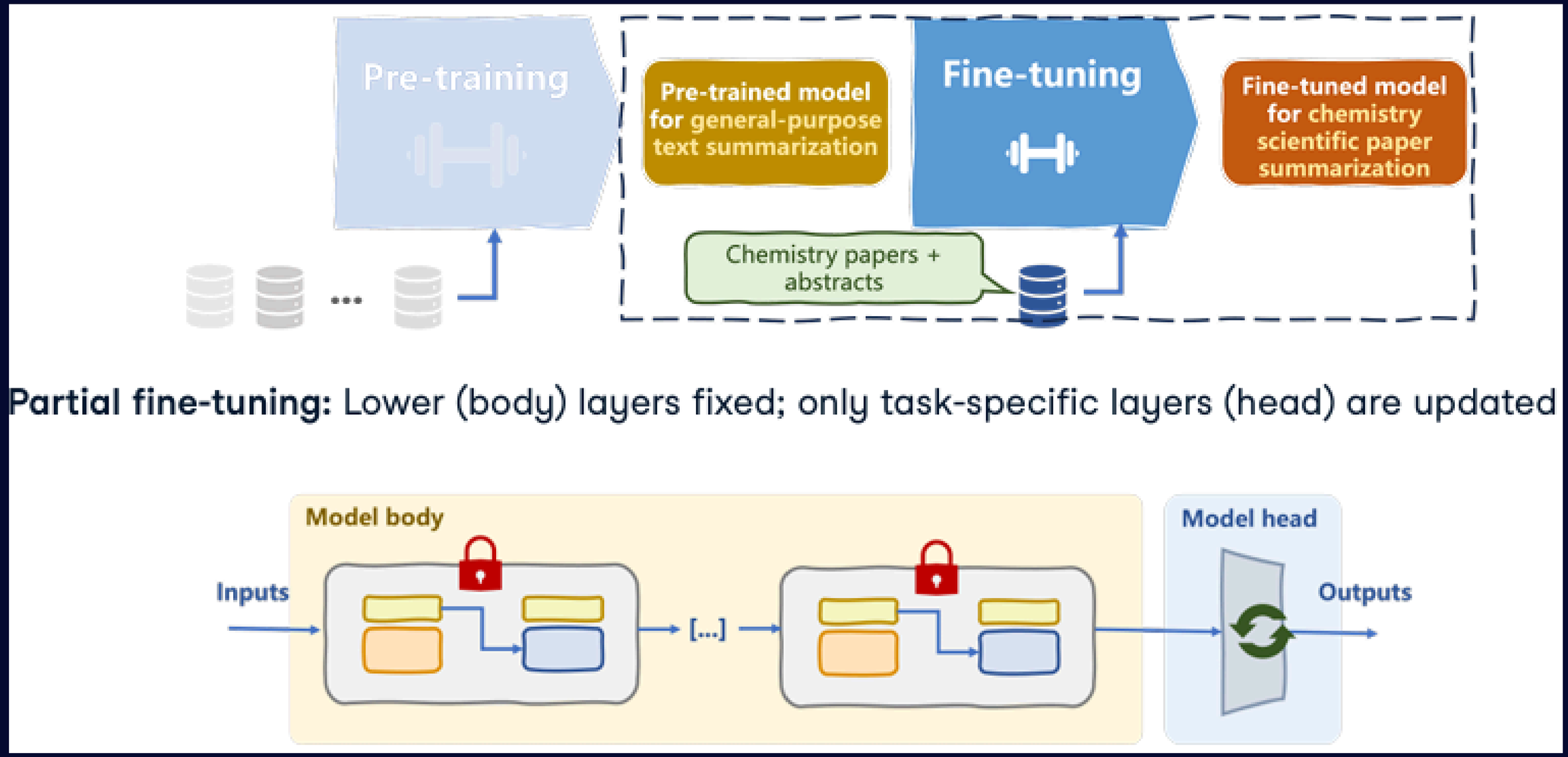
FULL FINE-TUNING



Full fine-tuning: The entire model weights are updated; more computationally expensive



PARTIAL FINE-TUNING



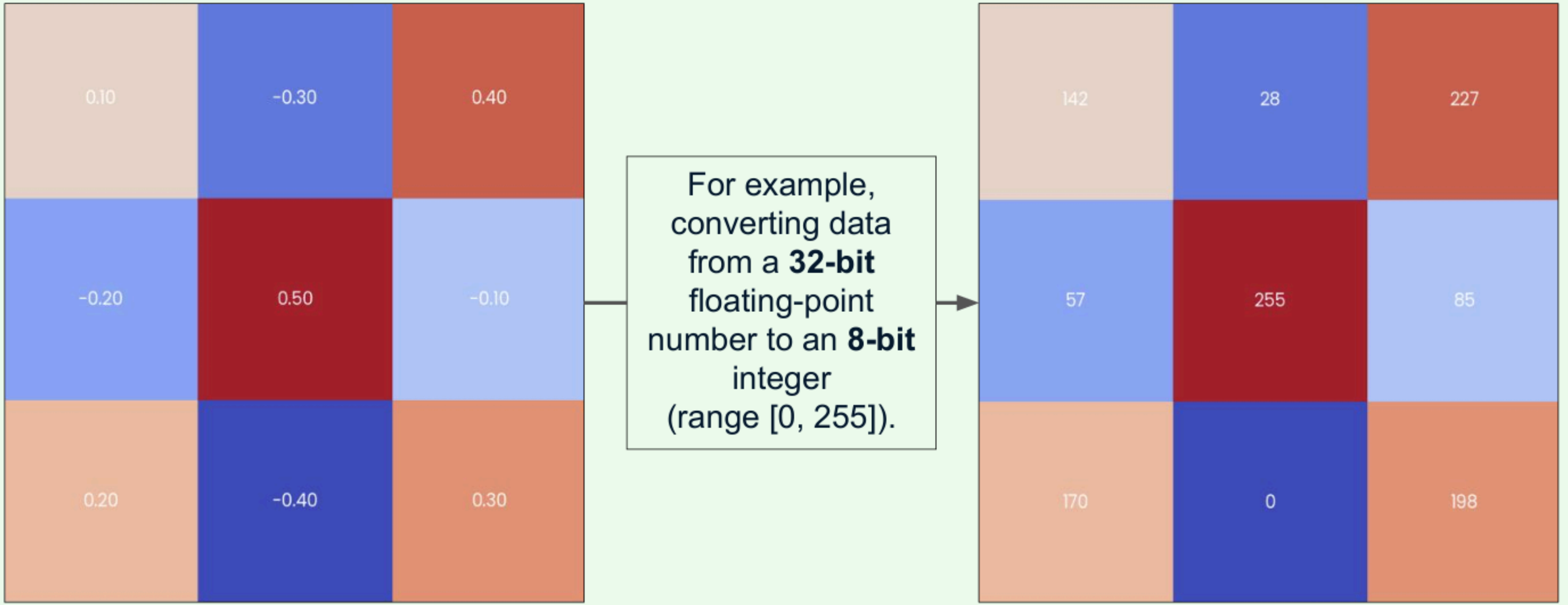
Transfer Learning

Model trained on one task is adapted for a different but related task
These models are typically fine-tuned on a smaller, task-specific dataset.
Types of Transfer Learning:

- Partial Fine-tuning: Adjusting some parts of a pre-trained model.
- Complete Fine-tuning: Entire model adjustments for new tasks.
- Zero-shot Learning: Enables the model to tackle tasks it has never explicitly learned during its training phase.
- One-shot and Few-shot Learning: These methods allow the model to adapt to new tasks using minimal data — as few as one or a handful of examples.

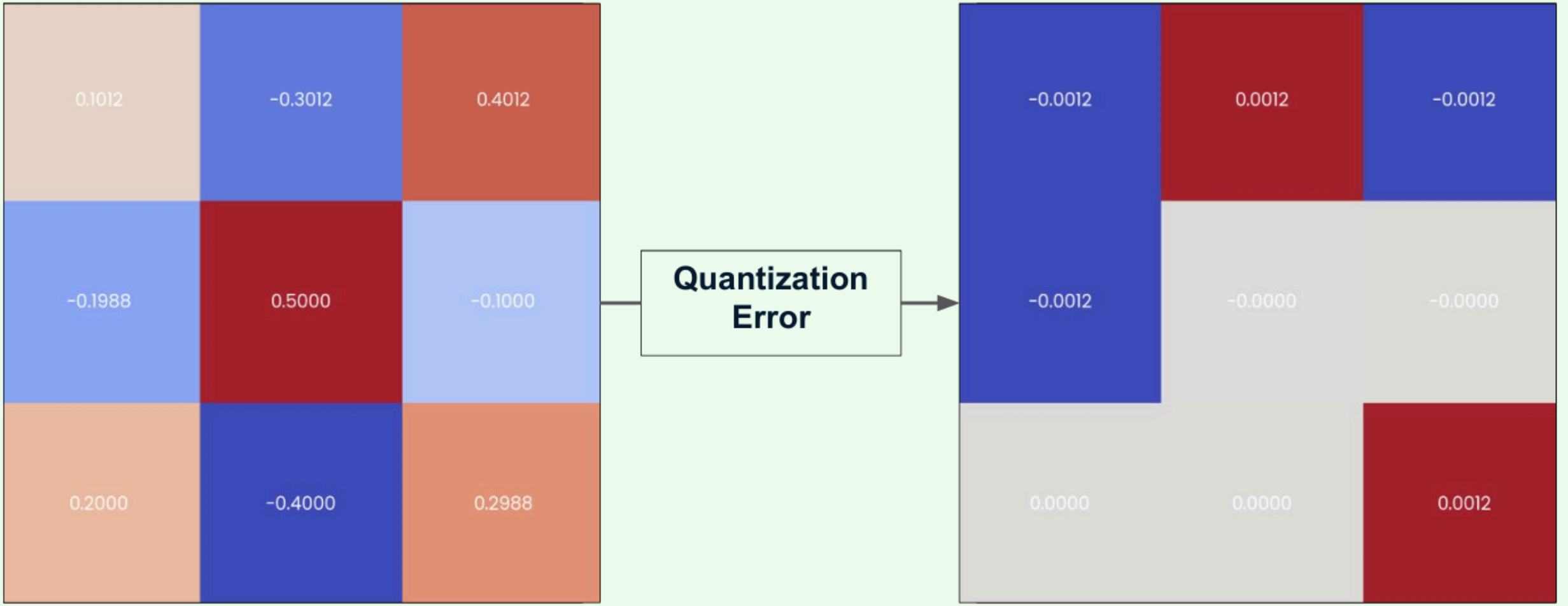
LLM Quantization

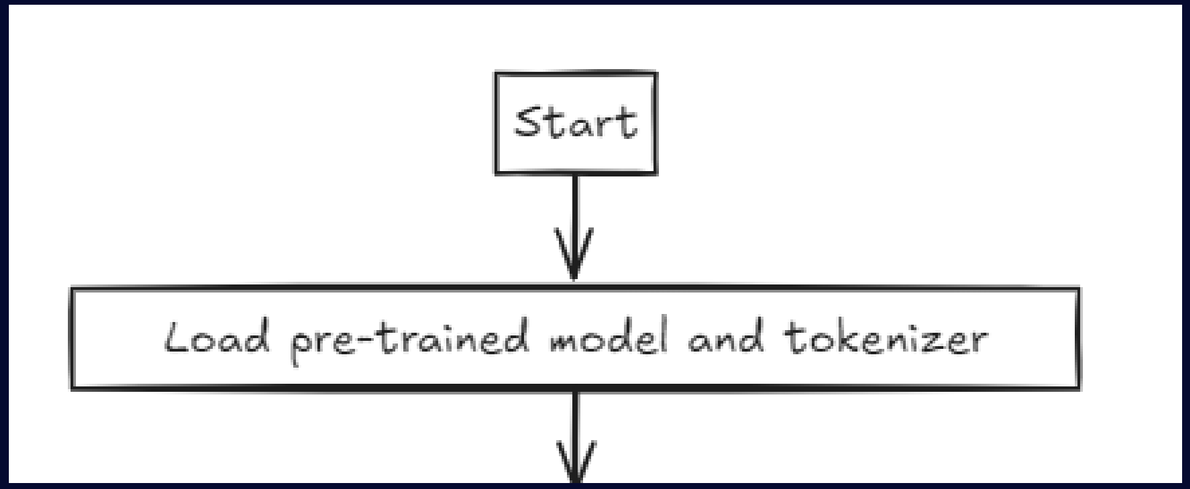
- Model compression technique that converts the **weights and/or activations** from high-precision values to lower-precision ones.



LLM Quantization

- Quantization shrinks LLMs to **consume less memory, and require less storage space**, while minimizing the performance degradation.




Start
Load pre-trained model and tokenizer

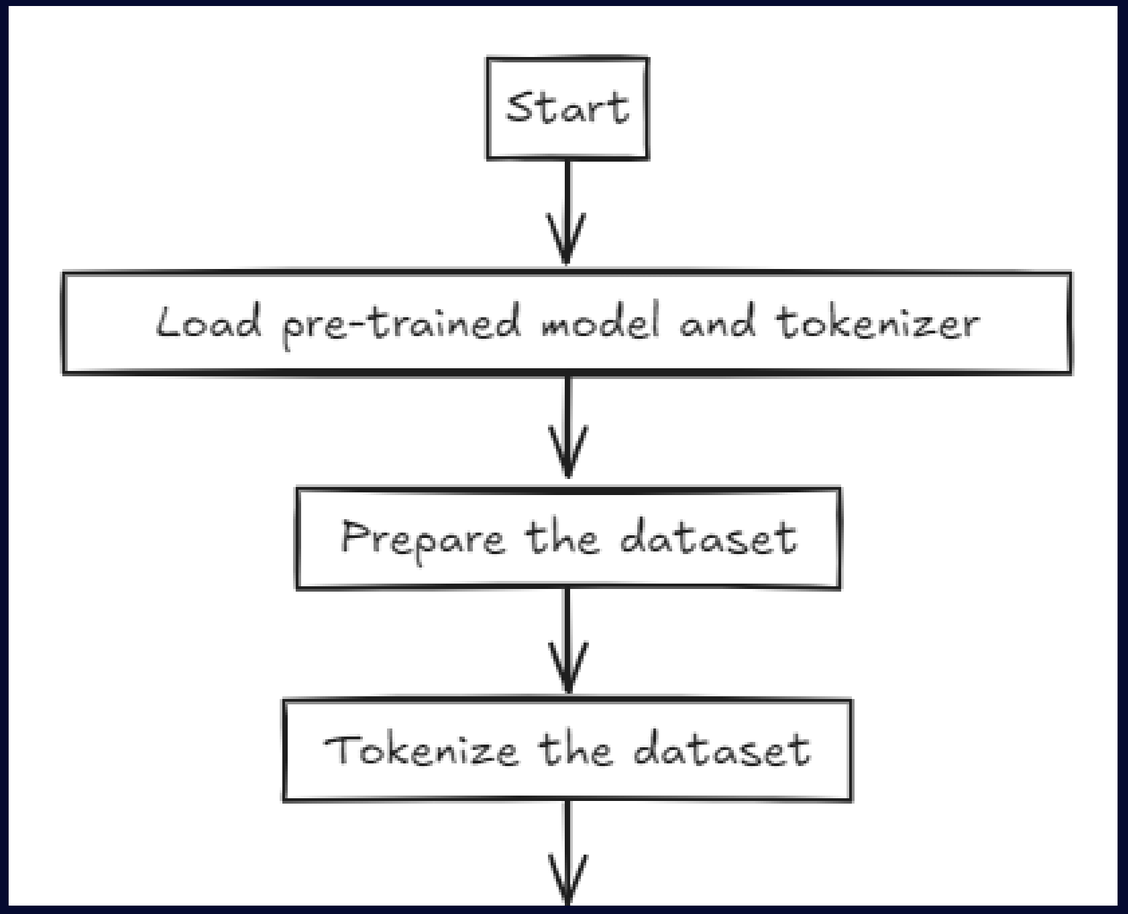
FINE-TUNING A PRE-TRAINED HUGGING FACE LLM

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

model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
model_name, num_labels=2)
```

Load BERT-based model for text classification and associated tokenizer

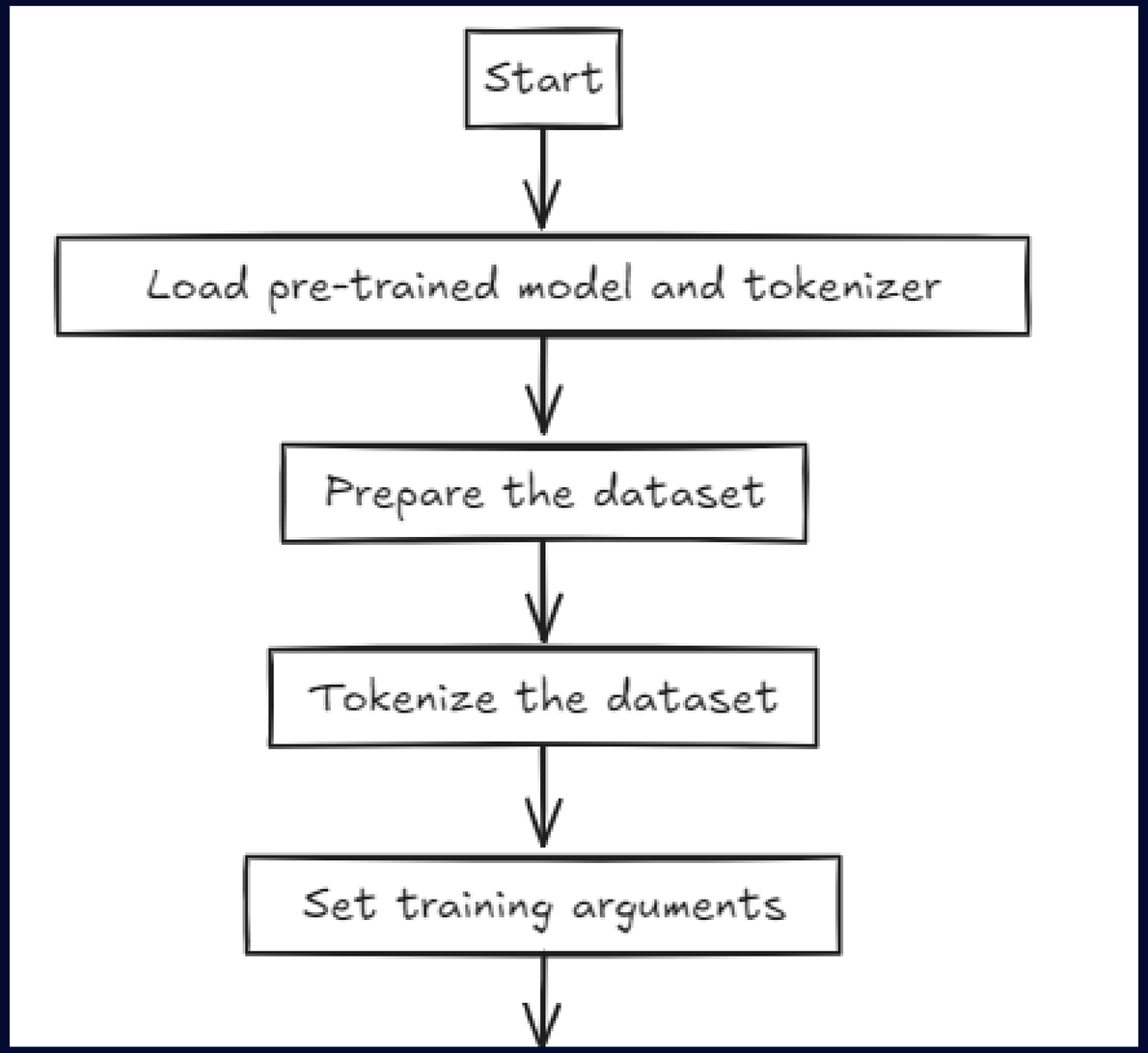
Tokenize dataset used for fine-tuning IMDB reviews dataset


Start
Load pre-trained model and tokenizer
Prepare the dataset
Tokenize the dataset

```python
def tokenize_function(examples):
    return tokenizer(
        examples["text"], padding="max_length", truncation=True)
data = load_dataset("imdb")
tokenized_data = data.map(tokenize_function, batched=True)
```

truncation= True truncates input sequences beyond model's

max_length batched= True to process examples in batches rather than individually

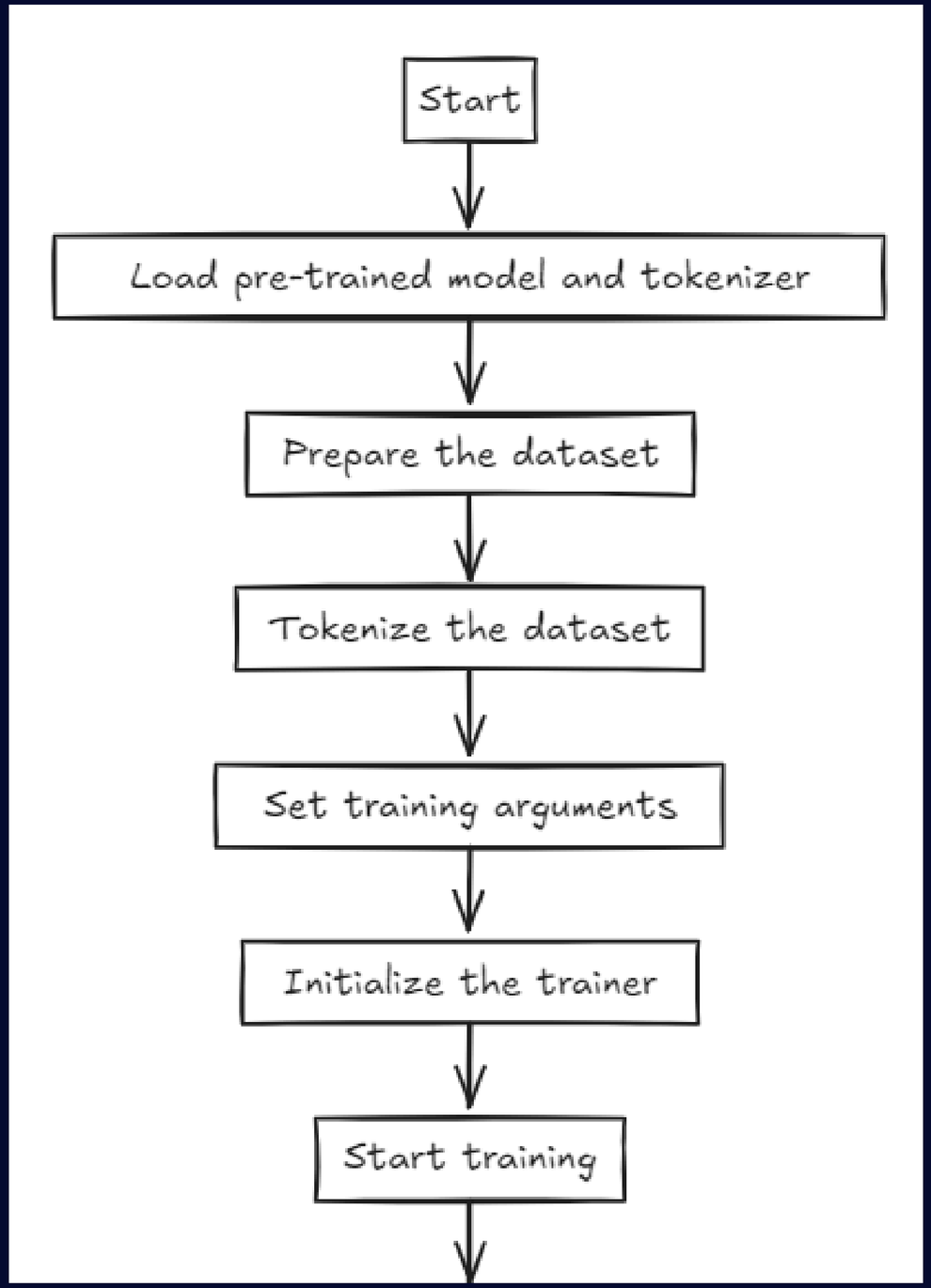

Start
Load pre-trained model and tokenizer
Prepare the dataset
Tokenize the dataset
Set training arguments

```python
from transformers import Trainer, TrainingArguments
training_args = TrainingArguments(
output_dir="./smaller_bert_finetuned",
per_device_train_batch_size=8,
num_train_epochs=3,
evaluation_strategy="steps",
eval_steps=500,
save_steps=500,
logging_dir="./logs",
)

```

Training Arguments

class: customize training settings
Output directory, batch size per GPU, epochs, etc.

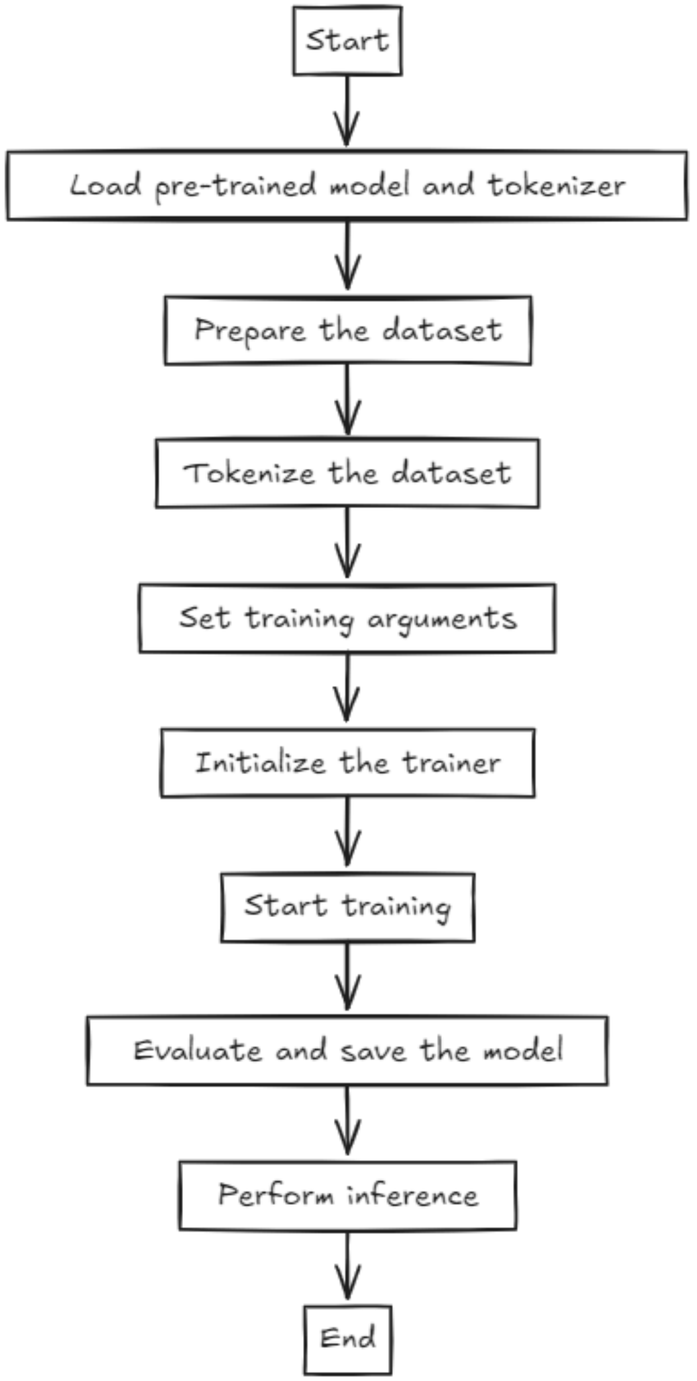

Start
Load pre-trained model and tokenizer
Prepare the dataset
Tokenize the dataset
Set training arguments
Initialize the trainer
Start training

```python
trainer = Trainer(
model=model,
args=training_args,
train_dataset=tokenized_datasets["train"],
eval_dataset=tokenized_datasets["test"],
)
trainer.train()
```

Trainer class: manage training and validation loop

Specify model, training arguments, training and validation sets

trainer.train() : execute training loop


Start
Load pre-trained model and tokenizer
Prepare the dataset
Tokenize the dataset
Set training arguments
Initialize the trainer
Start training
Evaluate and save the model
Perform inference
End

After fine-tuning, inference is performed as usual Tokenize inputs, pass them to the LLM, obtain and post-process outputs

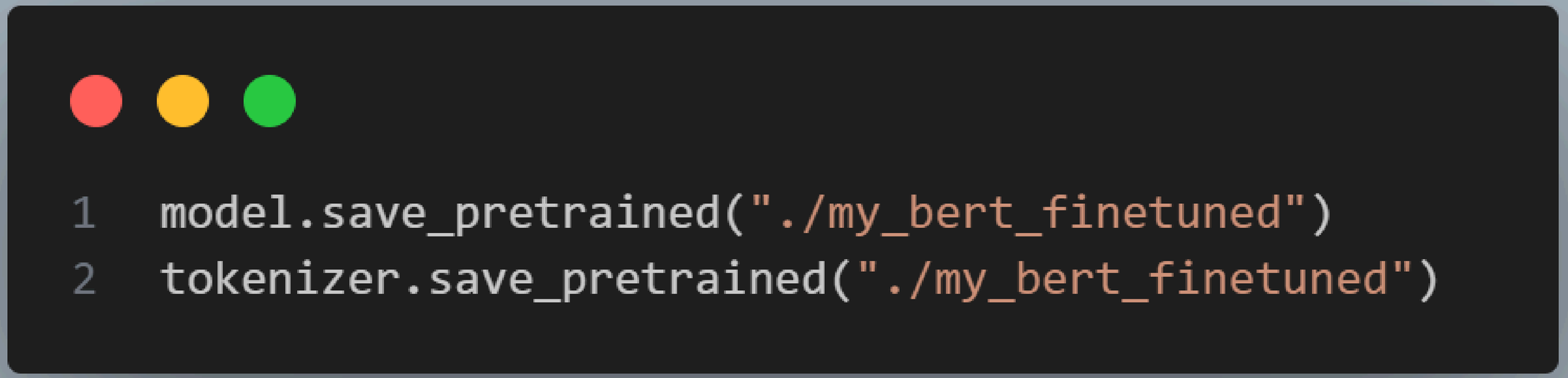
```
example_input = tokenizer("I am absolutely amazed with this new and revolutionary AI device",
                          return_tensors="pt")

output = model(**example_input)

predicted_label = torch.argmax(output.logits, dim=1).item()
print("Predicted Label:", predicted_label)
```

Predicted Label: 0

https://www.linkedin.com/in/zartashiaafzal/



Fine-tuned model and tokenizer can be saved using .save_pretrained()