What is Fine-Tuning?

Fine-tuning is the process of taking a **pre-trained model** and training it a bit more on your **specific dataset** so it performs well on your task (e.g., sentiment analysis, spam detection, topic classification, etc.).

Key Idea

Pre-trained models (like BERT, GPT, ResNet, etc.) are trained on **very large, general-purpose datasets** like:

- BERT: Wikipedia + BookCorpus
- GPT: Web pages, books, forums, etc.

But your task (e.g., classifying medical reports, customer reviews, legal text) might be **very specific**.

So instead of training from scratch (which needs lots of data & compute), you start with a pre-trained model and train it **a little more** on your data — that's **fine-tuning**.

What is BERT?

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model developed by **Google AI** in 2018. It's designed for **natural language understanding (NLU)** tasks like:

- Text classification
- Named entity recognition (NER)
- Question answering
- Sentiment analysis

Example

Let’s say you want to build a model to classify emails as **spam** or **not spam**:

1. **Pre-trained model**: BERT already understands English sentence structure and word meanings.
2. **Your dataset**: 10,000 labeled emails (spam/not spam).
3. **Fine-tuning**: You train BERT on your dataset so it learns what spam looks like.

What Changes During Fine-Tuning?

- You **replace the final layer** (e.g., classification head) to fit your task (e.g., 2 labels for spam vs not-spam).
- Then you **train** on your dataset — adjusting the model's weights slightly to learn your task.
- You often use **a smaller learning rate** to avoid "forgetting" what the model already knows.

Benefits:-

From-Scratch Training	Fine-Tuning
Needs tons of data	Needs less data
Expensive (compute)	Cheaper & faster
Starts from zero	Starts with knowledge

Here's a **complete working Python script** to fine-tune a **BERT model** on a **CSV file for text classification** using **Hugging Face Transformers + PyTorch**.

Supports: Custom CSV file, binary or multi-class labels

Example CSV Format

text	label
"I love this product"	positive
"Absolutely terrible!"	negative

Assumes your CSV has two columns: `text` and `label`

🧠 Full Fine-Tuning Script:-

```
# Fine-tune BERT on your own CSV dataset
# Requirements: pip install transformers datasets evaluate pandas scikit-learn
torch

import pandas as pd
from datasets import Dataset, train_test_split
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer
)
from sklearn.preprocessing import LabelEncoder
import numpy as np
import torch

# 1. Load CSV dataset
df = pd.read_csv("your_dataset.csv")  # 🔁 Replace with your path
df = df.dropna(subset=["text", "label"])  # Clean missing data

# 2. Encode labels as integers
label_encoder = LabelEncoder()
df["label"] = label_encoder.fit_transform(df["label"])
num_labels = len(label_encoder.classes_)

# 3. Convert to Hugging Face dataset
dataset = Dataset.from_pandas(df)

# 4. Tokenize text
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

def tokenize(example):
    return tokenizer(example["text"], padding="max_length", truncation=True,
max_length=256)

dataset = dataset.map(tokenize)

# 5. Train/test split
split_dataset = dataset.train_test_split(test_size=0.2, seed=42)
train_dataset = split_dataset["train"]
eval_dataset = split_dataset["test"]

# 6. Load model
```

```
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased",
num_labels=num_labels)

# 7. Define training args
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_strategy="epoch",
)

# 8. Define accuracy metric
import evaluate
accuracy = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return accuracy.compute(predictions=predictions, references=labels)

# 9. Define Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    compute_metrics=compute_metrics,
)

# 10. Train
trainer.train()

# 11. Evaluate
results = trainer.evaluate()
print("Final evaluation results:", results)

# 12. Save model and tokenizer
trainer.save_model("my-finetuned-bert")
tokenizer.save_pretrained("my-finetuned-bert")
```

Expected CSV Format

```
text,label
"I love this movie",positive
"This film is terrible",negative
```

Here's a list of powerful, **open-source LLMs comparable to GPT-4** that **you can fine-tune** on your own tasks — including chatbots, code generation, summarization, and more.

 These models are capable, open-source, and support fine-tuning with tools like **Hugging Face**, **LoRA**, **QLoRA**, or **Direct Training**.

Top Open-Source LLMs You Can Fine-Tune

Model	Base Size	Creator	Key Features	License
LLaMA 3	8B, 70B	Meta	GPT-4-class quality, strong reasoning, chat-ready	Custom (non-commercial use)
Mistral	7B	Mistral AI	Fast, small, GPT-3.5-level performance	Apache 2.0
Mixtral (MoE)	12.9B (active)	Mistral AI	Sparse Mixture of Experts, high performance	Apache 2.0
Phi-3	3.8B, 7B	Microsoft	Compact, very strong on reasoning, code, math	MIT
OpenChat	Based on LLaMA 2	OpenChat Team	Instruction-tuned LLaMA, chat-style LLM	Apache 2.0
Gemma	2B, 7B	Google DeepMind	Clean open license, fast to fine-tune	Apache 2.0
Falcon	7B, 40B	TII (UAE)	Good general-purpose LLM	Apache 2.0
Qwen	7B, 14B	Alibaba	Top benchmark scores, multilingual	Apache 2.0
WizardLM	LLaMA-based	NLP engineers	Instruction-tuned LLMs	Apache 2.0

How to Fine-Tune These Models

You can fine-tune them using:

⚙️ Lightweight Fine-Tuning

- **LoRA / QLoRA** (Low-Rank Adaptation): Very efficient (low GPU memory use)
- Tools:
  - 🤗 PEFT (Parameter-Efficient Fine-Tuning)
  - QLoRA
  - Axolotl

⚙️ Full Fine-Tuning (GPU-intensive)

- Fine-tune all model weights (requires 24–80GB GPU+)
- Use libraries like Hugging Face `transformers` or DeepSpeed

🧰 Popular Fine-Tuning Frameworks

Tool	Best For	Description
PEFT (Hugging Face)	LoRA, QLoRA	Simple LoRA-based fine-tuning
Axolotl	All-in-one fine-tuning tool	YAML config + dataset support
LLaMA Factory	Chat-style tuning	Fine-tune LLaMA/Mistral-style chatbots
DeepSpeed	Full fine-tuning	Optimized for large-scale models
FastChat	Chatbot hosting + tuning	Backend for OpenChat and Vicuna