**Q5: How do you track, monitor, and audit ML training?**

Tracking, monitoring, and auditing ML training are essential to ensure the reliability, reproducibility, and transparency of ML models. Here are strategies and best practices for tracking, monitoring, and auditing ML training for both BERT and SpaCy models.

**Key Concepts**

* **Tracking:** Keeping a record of experiments, configurations, and results.
* **Monitoring:** Observing the system's performance, resource usage, and behavior during and after training.
* **Auditing:** Maintaining a log of changes, access, and modifications to the models and data.

**Tools for Tracking and Monitoring**

1. **MLflow:** An open-source platform for managing the ML lifecycle, including experimentation, reproducibility, and deployment.
2. **TensorBoard:** A suite of visualization tools for TensorFlow that can be used to monitor training metrics.
3. **Weights & Biases:** A platform for experiment tracking, model monitoring, and collaboration.
4. **Prometheus & Grafana:** Tools for monitoring and alerting system metrics.

**Tracking and Monitoring with MLflow**

MLflow provides a comprehensive solution for tracking and monitoring ML experiments.

1. **Install MLflow:**

pip install mlflow

1. **Set Up MLflow Tracking:**

import mlflow

import mlflow.sklearn

# Start a new run

mlflow.start\_run()

# Log parameters

mlflow.log\_param("learning\_rate", 2e-5)

mlflow.log\_param("batch\_size", 8)

mlflow.log\_param("epochs", 3)

# Log metrics

mlflow.log\_metric("precision", precision)

mlflow.log\_metric("recall", recall)

mlflow.log\_metric("f1", f1)

# Log the model

mlflow.pytorch.log\_model(model, "model")

# End the run

mlflow.end\_run()

mlflow ui

This command starts the MLflow tracking server, which can be accessed at http://localhost:5000.

**Tracking and Monitoring with TensorBoard**

For TensorFlow-based models, TensorBoard provides robust tracking and visualization.

1. **Install TensorBoard:**

pip install tensorboard

1. **Integrate TensorBoard with Training:**

from torch.utils.tensorboard import SummaryWriter

# Create a SummaryWriter to log metrics

writer = SummaryWriter()

# During training, log metrics

for epoch in range(num\_epochs):

# ... training loop ...

writer.add\_scalar('Loss/train', loss, epoch)

writer.add\_scalar('Accuracy/train', accuracy, epoch)

writer.close()

1. **Launch TensorBoard:**

tensorboard --logdir=./logs

**Monitoring System Metrics with Prometheus and Grafana**

1. **Install Prometheus and Grafana:** Follow the installation guides from the Prometheus and Grafana websites.
2. **Set Up Prometheus to Monitor Metrics:**
   * Configure Prometheus to scrape metrics from your training servers.
   * Use Prometheus client libraries to expose metrics.
3. **Set Up Grafana for Visualization:**
   * Add Prometheus as a data source in Grafana.
   * Create dashboards to visualize metrics such as CPU usage, memory usage, and GPU utilization.

**Auditing ML Training**

1. **Maintain a Log of Changes:**
   * Use version control systems like Git to track changes to code, configurations, and datasets.
   * Keep a detailed changelog and commit messages.
2. **Access Controls and Permissions:**
   * Use role-based access control (RBAC) to manage permissions for accessing and modifying models and data.
   * Audit logs to track who accessed what and when.
3. **Data Provenance:**
   * Maintain records of data sources, preprocessing steps, and transformations applied to the data.

**Example Code for Tracking with MLflow**

**BERT Model Training with MLflow Tracking:**

import mlflow

import mlflow.pytorch

from transformers import Trainer, TrainingArguments, AutoModelForTokenClassification, AutoTokenizer

# Load model and tokenizer

model = AutoModelForTokenClassification.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

tokenizer = AutoTokenizer.from\_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

# Define training arguments

training\_args = TrainingArguments(

output\_dir="./results",

evaluation\_strategy="epoch",

learning\_rate=2e-5,

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\_total\_limit=2,

save\_steps=500,

)

# Initialize the Trainer

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=val\_dataset,

tokenizer=tokenizer

)

# Start an MLflow run

mlflow.start\_run()

# Log parameters

mlflow.log\_param("learning\_rate", training\_args.learning\_rate)

mlflow.log\_param("batch\_size", training\_args.per\_device\_train\_batch\_size)

mlflow.log\_param("num\_epochs", training\_args.num\_train\_epochs)

# Train the model

trainer.train()

# Evaluate the model

results = trainer.evaluate()

# Log metrics

mlflow.log\_metric("eval\_loss", results["eval\_loss"])

mlflow.log\_metric("eval\_accuracy", results["eval\_accuracy"])

# Log the model

mlflow.pytorch.log\_model(model, "model")

# End the run

mlflow.end\_run()

**SpaCy Model Training with MLflow Tracking:**

import mlflow

import spacy

from spacy.training.example import Example

from spacy.scorer import Scorer

# Load SpaCy model

nlp = spacy.load("en\_core\_web\_sm")

# Start an MLflow run

mlflow.start\_run()

# Log parameters

mlflow.log\_param("model\_name", "en\_core\_web\_sm")

# Define a function to train the model

def train\_spacy\_model(train\_data, val\_data):

optimizer = nlp.begin\_training()

for epoch in range(10):

losses = {}

for text, annotations in train\_data:

example = Example.from\_dict(nlp.make\_doc(text), annotations)

nlp.update([example], drop=0.5, sgd=optimizer, losses=losses)

print(f"Epoch {epoch} Losses: {losses}")

mlflow.log\_metric("epoch\_loss", losses['ner'], step=epoch)

return nlp

# Train the model

train\_data = [("John Doe went to New York last week.", {"entities": [(0, 8, "PERSON"), (17, 25, "GPE")]}), ...]

val\_data = [("Jane Smith traveled to Paris.", {"entities": [(0, 10, "PERSON"), (22, 27, "GPE")]}), ...]

nlp\_trained = train\_spacy\_model(train\_data, val\_data)

# Evaluate the model

scorer = Scorer()

examples = [Example.from\_dict(nlp\_trained.make\_doc(text), ann) for text, ann in val\_data]

scores = scorer.score(examples)

# Log metrics

mlflow.log\_metric("precision", scores["ents\_p"])

mlflow.log\_metric("recall", scores["ents\_r"])

mlflow.log\_metric("f1\_score", scores["ents\_f"])

# Save the model

mlflow.spacy.log\_model(nlp\_trained, "spacy\_model")

# End the run

mlflow.end\_run()