🚀 PROJECT-BASED AI SKILL-UP ROADMAP

Learn by Building: "IndustrialMind" - End-to-End ML Platform

Diego Carriel Lopez | 12-Month Project Journey

🎯 THE PHILOSOPHY

> ***"The best way to learn is to build something real."***

Instead of: `Learn PyTorch → Learn MLOps → Learn K8s → Build Project`

We do: `Build Project → Learn PyTorch WHILE building → Learn MLOps WHILE deploying → etc.`

Every week you ship something. Every feature teaches a skill.

🏗 THE PROJECT: IndustrialMind

What We're Building

An **open-source Industrial AI Platform** that: 1. Ingests real-time sensor data from manufacturing equipment 2. Detects anomalies and predicts failures using ML 3. Provides a Knowledge Graph of equipment relationships 4. Offers an LLM-powered assistant for technicians 5. Deploys with full MLOps pipeline

Why This Project?

Your Experience	Project Component	Skills Gained
Nestlé InfluxDB work	Data ingestion layer	Real-world relevance
Neo4J certification	Knowledge Graph module	Showcase expertise
LangChain/RAG experience	LLM Assistant	Deepen LLM skills
Streamlit apps	Dashboard	Production UI
NEW	PyTorch models	Fill critical gap
NEW	MLOps pipeline	Fill critical gap
NEW	Kubernetes deployment	Fill critical gap

The Stack (Matches $200K+ Job Requirements)

```
┌─────────────────────────────────────────────────────────────────┐
│                         INDUSTRIALMIND                          │
├─────────────────────────────────────────────────────────────────┤
│  📊 DATA LAYER                                                  │
│  ├── Apache Kafka (streaming)                                   │
│  ├── InfluxDB (time series storage)                             │
│  └── PostgreSQL (metadata)                                      │
├─────────────────────────────────────────────────────────────────┤
│  🧠 ML LAYER                                                    │
│  ├── PyTorch (models)                                           │
│  ├── Hugging Face (transformers)                                │
│  └── ONNX (model optimization)                                  │
├─────────────────────────────────────────────────────────────────┤
│  🔗 KNOWLEDGE LAYER                                             │
│  ├── Neo4J (graph database)                                     │
│  ├── ChromaDB (vector store)                                    │
│  └── LangChain (orchestration)                                  │
├─────────────────────────────────────────────────────────────────┤
│  🚀 MLOPS LAYER                                                 │
│  ├── MLflow (experiment tracking)                               │
│  ├── DVC (data versioning)                                      │
│  ├── Docker (containerization)                                  │
│  └── Kubernetes (orchestration)                                 │
├─────────────────────────────────────────────────────────────────┤
│  ☁ CLOUD LAYER                                                  │
│  ├── AWS SageMaker OR Azure ML                                  │
│  ├── AWS Lambda / Azure Functions                               │
│  └── Terraform (IaC)                                            │
├─────────────────────────────────────────────────────────────────┤
│  🖥 FRONTEND                                                    │
│  ├── Next.js (main dashboard)                                   │
│  ├── FastAPI (backend API)                                      │
│  └── Streamlit (ML demos)                                       │
└─────────────────────────────────────────────────────────────────┘
```

📅 12-MONTH PROJECT ROADMAP

Overview

Month	Project Phase	Key Deliverable	Skills Acquired
1	Project Setup & Data Simulator	Working data pipeline	Kafka, Docker basics
2	First PyTorch Model	Anomaly detector	PyTorch fundamentals
3	MLflow Integration	Tracked experiments	MLOps basics
4	Time Series Forecasting	Predictive model	Advanced PyTorch
5	Knowledge Graph	Equipment graph	Neo4J advanced
6	RAG System	Document Q&A	Vector DBs, LangChain
7	LLM Fine-tuning	Domain-adapted LLM	LoRA, PEFT
8	Kubernetes Deployment	K8s cluster	K8s, Helm
9	Cloud Migration	AWS/Azure deployment	Cloud ML platforms
10	CI/CD Pipeline	Automated deployment	GitHub Actions, MLOps
11	Monitoring & Observability	Production monitoring	Prometheus, Grafana
12	Polish & Documentation	Portfolio-ready	Technical writing

🗓 MONTH 1: Foundation & Data Pipeline

Goal: Build a working data ingestion system that simulates industrial sensors

Week 1: Project Bootstrap

Tasks:

```
# Day 1-2: Repository setup
mkdir industrialmind && cd industrialmind
git init
# Create structure:
# industrialmind/
# ├── docker-compose.yml
# ├── README.md
# ├── data-simulator/
# ├── data-ingestion/
# ├── ml-models/
# ├── knowledge-graph/
# ├── llm-assistant/
# ├── api/
# ├── frontend/
# ├── mlops/
# └── docs/
```

Deliverable: GitHub repo with proper structure, README, and .gitignore

Learn While Building: - Resource: [Docker in 1 Hour]() (FREE) - Resource: [Git Best Practices]()

Week 2: Industrial Data Simulator

Tasks: - Build Python simulator that generates realistic sensor data - Simulate: temperature, vibration, pressure, power consumption - Add realistic patterns: normal operation, degradation, failure modes

Code to Write:

```
# data-simulator/simulator.py
import numpy as np
from dataclasses import dataclass
from enum import Enum

class MachineState(Enum):
    NORMAL = "normal"
    DEGRADING = "degrading"
    FAILING = "failing"

@dataclass
class SensorReading:
    timestamp: datetime
    machine_id: str
    temperature: float
    vibration: float
    pressure: float
    power: float
    state: MachineState  # for training labels

class IndustrialSimulator:
```

```
    def generate_reading(self, machine_id: str) -> SensorReading:
        # Implement realistic sensor patterns
        pass
```

Deliverable: Working simulator producing 1000+ readings/minute

Learn While Building: - Resource: [Time Series Simulation]()

Week 3: Kafka + InfluxDB Pipeline

Tasks: - Set up Kafka for streaming (Docker) - Set up InfluxDB for storage (Docker) - Connect simulator → Kafka → InfluxDB

docker-compose.yml:

```yaml
version: '3.8'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092

  influxdb:
    image: influxdb:2.7
    ports:
      - "8086:8086"
    volumes:
      - influxdb-data:/var/lib/influxdb2

  simulator:
    build: ./data-simulator
    depends_on:
      - kafka

  ingestion:
    build: ./data-ingestion
    depends_on:
      - kafka
      - influxdb

volumes:
  influxdb-data:
```

Deliverable: `docker-compose up` starts entire pipeline

Learn While Building: - Resource: [Kafka Basics in 30 min]() (FREE) - Resource: [InfluxDB Python Client]()

Week 4: Basic Visualization

Tasks: - Create simple Streamlit dashboard - Show real-time sensor readings - Plot time series with Plotly

Deliverable: Live dashboard at http://localhost:8501

Learn While Building: - You already know Streamlit! Just apply it.

Month 1 Checkpoint ✅

```
□ GitHub repo with proper structure
□ Docker Compose running all services
□ Simulator generating realistic data
□ Kafka streaming working
□ InfluxDB storing data
□ Basic Streamlit dashboard
□ README with setup instructions
```

Skills Acquired: - ✅ Docker & Docker Compose - ✅ Kafka fundamentals - ✅ Streaming data architecture - ✅ Project organization

---

🗓 MONTH 2: First PyTorch Model

Goal: Build an anomaly detection model in PyTorch (NOT TensorFlow!)

Week 5: PyTorch Fundamentals

Tasks: - Convert your existing TensorFlow knowledge to PyTorch - Build a simple autoencoder for anomaly detection

Learn While Building:

```python
# ml-models/anomaly_detector/model.py
import torch
import torch.nn as nn

class SensorAutoencoder(nn.Module):
    def __init__(self, input_dim: int = 4, latent_dim: int = 2):
        super().__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 16),
            nn.ReLU(),
            nn.Linear(16, 8),
            nn.ReLU(),
            nn.Linear(8, latent_dim)
        )

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 8),
            nn.ReLU(),
            nn.Linear(8, 16),
            nn.ReLU(),
            nn.Linear(16, input_dim)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

    def get_reconstruction_error(self, x):
        reconstructed = self.forward(x)
        return torch.mean((x - reconstructed) ** 2, dim=1)
```

Resource: PyTorch in 60 Minutes (FREE, Official)

Week 6: Training Pipeline

Tasks: - Create PyTorch Dataset and DataLoader - Implement training loop with validation - Add early stopping

Code Structure:

```python
# ml-models/anomaly_detector/train.py
from torch.utils.data import Dataset, DataLoader

class SensorDataset(Dataset):
    def __init__(self, influxdb_client, time_range):
        self.data = self._load_from_influx(influxdb_client, time_range)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return torch.tensor(self.data[idx], dtype=torch.float32)

def train_model(model, train_loader, val_loader, epochs=100):
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.MSELoss()

    for epoch in range(epochs):
        model.train()
        for batch in train_loader:
            optimizer.zero_grad()
            output = model(batch)
            loss = criterion(output, batch)
            loss.backward()
            optimizer.step()

        # Validation
        model.eval()
        val_loss = evaluate(model, val_loader, criterion)
        print(f"Epoch {epoch}: Val Loss = {val_loss:.4f}")
```

Deliverable: Trained model with validation metrics

Week 7: Model Evaluation & Thresholding

Tasks: - Implement anomaly scoring - Find optimal threshold using validation data - Create evaluation metrics (precision, recall, F1)

Code:

```python
# ml-models/anomaly_detector/evaluate.py
import numpy as np
from sklearn.metrics import precision_recall_curve, f1_score

def find_optimal_threshold(model, val_data, val_labels):
    """Find threshold that maximizes F1 score"""
```

```
    model.eval()
    with torch.no_grad():
        errors = model.get_reconstruction_error(val_data).numpy()

    precisions, recalls, thresholds = precision_recall_curve(val_labels, errors)
    f1_scores = 2 * (precisions * recalls) / (precisions + recalls + 1e-8)

    optimal_idx = np.argmax(f1_scores)
    return thresholds[optimal_idx], f1_scores[optimal_idx]
```

Week 8: Real-time Inference Service

Tasks: - Create FastAPI endpoint for predictions - Dockerize the model service - Connect to Kafka for real-time scoring

Code:

```
# api/anomaly_service.py
from fastapi import FastAPI
import torch

app = FastAPI()

# Load model at startup
model = SensorAutoencoder()
model.load_state_dict(torch.load("models/autoencoder.pt"))
model.eval()
threshold = 0.05  # From evaluation

@app.post("/predict")
async def predict_anomaly(reading: SensorReading):
    tensor = torch.tensor([[
        reading.temperature,
        reading.vibration,
        reading.pressure,
        reading.power
    ]])

    with torch.no_grad():
        error = model.get_reconstruction_error(tensor).item()

    return {
        "is_anomaly": error > threshold,
        "anomaly_score": error,
        "threshold": threshold
    }
```

Deliverable: Working API at http://localhost:8000/docs

Month 2 Checkpoint ✅

```
□ PyTorch autoencoder model
□ Training pipeline with validation
□ Anomaly threshold optimization
□ FastAPI inference service
□ Docker container for model
□ Integration with data pipeline
```

Skills Acquired: - ✅ PyTorch fundamentals (THE critical gap!) - ✅ Custom Dataset/DataLoader - ✅ Training loops - ✅ Model evaluation - ✅ FastAPI ML serving

---

🗓 MONTH 3: MLOps with MLflow

Goal: Add experiment tracking and model registry

Week 9: MLflow Setup

Tasks: - Add MLflow to Docker Compose - Integrate tracking into training script - Log parameters, metrics, and artifacts

docker-compose addition:

```
  mlflow:
    image: ghcr.io/mlflow/mlflow:v2.9.2
    ports:
      - "5000:5000"
    command: mlflow server --host 0.0.0.0 --port 5000
    volumes:
      - mlflow-data:/mlflow
```

Training with MLflow:

```
import mlflow
import mlflow.pytorch

mlflow.set_tracking_uri("http://localhost:5000")
```

```
mlflow.set_experiment("anomaly-detection")

with mlflow.start_run():
    # Log parameters
    mlflow.log_params({
        "input_dim": 4,
        "latent_dim": 2,
        "learning_rate": 1e-3,
        "epochs": 100
    })

    # Training loop
    for epoch in range(epochs):
        train_loss = train_epoch(model, train_loader)
        val_loss = evaluate(model, val_loader)

        mlflow.log_metrics({
            "train_loss": train_loss,
            "val_loss": val_loss
        }, step=epoch)

    # Log model
    mlflow.pytorch.log_model(model, "model")

    # Log threshold
    mlflow.log_metric("optimal_threshold", threshold)
    mlflow.log_metric("f1_score", f1)
```

Deliverable: MLflow UI showing experiments at http://localhost:5000

Week 10: Model Registry

Tasks: - Register best models in MLflow Registry - Implement model versioning - Create model promotion workflow (staging → production)

Code:

```
# mlops/model_registry.py
from mlflow.tracking import MlflowClient

client = MlflowClient()

# Register model
model_uri = f"runs:/{run_id}/model"
mv = mlflow.register_model(model_uri, "anomaly-detector")

# Transition to staging
client.transition_model_version_stage(
    name="anomaly-detector",
    version=mv.version,
    stage="Staging"
)

# After validation, promote to production
client.transition_model_version_stage(
    name="anomaly-detector",
    version=mv.version,
    stage="Production"
)
```

Week 11: DVC for Data Versioning

Tasks: - Set up DVC for data versioning - Version training datasets - Create reproducible training pipelines

Commands:

```
# Initialize DVC
dvc init

# Track data
dvc add data/training_data.parquet

# Create pipeline
dvc.yaml:
stages:
  prepare:
    cmd: python scripts/prepare_data.py
    deps:
      - data/raw/
    outs:
      - data/processed/

  train:
    cmd: python ml-models/anomaly_detector/train.py
    deps:
      - data/processed/
      - ml-models/anomaly_detector/model.py
    outs:
      - models/autoencoder.pt
```

```
    metrics:
      - metrics.json:
          cache: false
```

Week 12: Automated Retraining

Tasks: - Create script that monitors model performance - Trigger retraining when performance degrades - Automate with simple cron job (CI/CD comes later)

Deliverable: Working MLOps pipeline with versioning

Month 3 Checkpoint ✅

```
□ MLflow tracking integrated
□ Model registry with staging/production
□ DVC for data versioning
□ Reproducible training pipeline
□ Automated retraining trigger
□ Documentation of MLOps workflow
```

Skills Acquired: - ✅ MLflow (experiment tracking, registry) - ✅ DVC (data versioning) - ✅ MLOps pipelines - ✅ Model lifecycle management

---

🗓 MONTH 4: Advanced Time Series with PyTorch

Goal: Build a predictive maintenance model using Transformer architecture

Week 13-14: Temporal Fusion Transformer

Tasks: - Implement attention-based time series model - Handle multi-horizon forecasting - Predict "time to failure"

Architecture:

```
# ml-models/predictive/temporal_transformer.py
import torch
import torch.nn as nn

class TemporalAttention(nn.Module):
    def __init__(self, d_model, n_heads):
        super().__init__()
        self.attention = nn.MultiheadAttention(d_model, n_heads)
        self.norm = nn.LayerNorm(d_model)

    def forward(self, x):
        attn_out, _ = self.attention(x, x, x)
        return self.norm(x + attn_out)

class PredictiveMaintenanceModel(nn.Module):
    def __init__(self, input_dim, d_model=64, n_heads=4, n_layers=3):
        super().__init__()

        self.input_projection = nn.Linear(input_dim, d_model)
        self.positional_encoding = PositionalEncoding(d_model)

        self.transformer_layers = nn.ModuleList([
            TemporalAttention(d_model, n_heads)
            for _ in range(n_layers)
        ])

        self.output_layer = nn.Linear(d_model, 1)  # Time to failure

    def forward(self, x):
        # x shape: (batch, seq_len, input_dim)
        x = self.input_projection(x)
        x = self.positional_encoding(x)

        for layer in self.transformer_layers:
            x = layer(x)

        # Take last timestep
        return self.output_layer(x[:, -1, :])
```

Resource: [Temporal Fusion Transformers Paper]

Week 15-16: Multi-Task Learning

Tasks: - Combine anomaly detection + failure prediction - Share encoder, separate heads - Joint training

Code:

```
class MultiTaskModel(nn.Module):
    def __init__(self, input_dim):
        super().__init__()

        # Shared encoder
```

```
        self.encoder = SharedEncoder(input_dim)

        # Task-specific heads
        self.anomaly_head = AnomalyHead()
        self.prediction_head = PredictionHead()

    def forward(self, x):
        features = self.encoder(x)

        anomaly_score = self.anomaly_head(features)
        time_to_failure = self.prediction_head(features)

        return {
            "anomaly_score": anomaly_score,
            "time_to_failure": time_to_failure
        }
```

Month 4 Checkpoint ✅

```
☐ Transformer-based time series model
☐ Positional encoding implementation
☐ Multi-horizon forecasting
☐ Multi-task learning architecture
☐ Model comparison in MLflow
```

Skills Acquired: - ✅ Advanced PyTorch (Transformers!) - ✅ Attention mechanisms - ✅ Time series deep learning - ✅ Multi-task learning

---

📅 MONTH 5: Knowledge Graph Integration

Goal: Build equipment relationship graph with Neo4J

Week 17-18: Graph Schema Design

Tasks: - Design ontology for industrial equipment - Model: Equipment → Components → Sensors → Readings - Add relationships: DEPENDS*ON, CONNECTED*TO, UPSTREAM_OF

Cypher Schema:

```
// Node types
CREATE CONSTRAINT equipment_id IF NOT EXISTS FOR (e:Equipment) REQUIRE e.id IS UNIQUE;
CREATE CONSTRAINT sensor_id IF NOT EXISTS FOR (s:Sensor) REQUIRE s.id IS UNIQUE;

// Equipment node
CREATE (e:Equipment {
    id: "MACHINE_001",
    type: "CNC_Mill",
    manufacturer: "Siemens",
    install_date: date("2020-01-15"),
    location: "Building_A_Floor_2"
})

// Sensor nodes
CREATE (s:Sensor {
    id: "TEMP_001",
    type: "temperature",
    unit: "celsius",
    min_threshold: 20,
    max_threshold: 80
})

// Relationships
MATCH (e:Equipment {id: "MACHINE_001"})
MATCH (s:Sensor {id: "TEMP_001"})
CREATE (e)-[:HAS_SENSOR {position: "spindle"}]->(s)

// Equipment dependencies
MATCH (e1:Equipment {id: "MACHINE_001"})
MATCH (e2:Equipment {id: "MACHINE_002"})
CREATE (e1)-[:FEEDS_INTO {product: "part_A"}]->(e2)
```

Week 19-20: Graph-Enhanced ML

Tasks: - Use graph features in ML models - Propagate anomalies through connected equipment - Implement graph neural network (optional advanced)

Code:

```
# knowledge-graph/graph_features.py
from neo4j import GraphDatabase

class GraphFeatureExtractor:
    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, password))
```

```
    def get_equipment_context(self, equipment_id: str) -> dict:
        """Get graph-based features for equipment"""
        query = """
        MATCH (e:Equipment {id: $equipment_id})
        OPTIONAL MATCH (e)-[:DEPENDS_ON]->(upstream:Equipment)
        OPTIONAL MATCH (downstream:Equipment)-[:DEPENDS_ON]->(e)
        OPTIONAL MATCH (e)-[:HAS_SENSOR]->(s:Sensor)
        RETURN
            e.type as equipment_type,
            count(DISTINCT upstream) as upstream_count,
            count(DISTINCT downstream) as downstream_count,
            count(DISTINCT s) as sensor_count,
            collect(DISTINCT upstream.id) as upstream_ids
        """

        with self.driver.session() as session:
            result = session.run(query, equipment_id=equipment_id)
            return result.single().data()

    def propagate_anomaly(self, equipment_id: str, anomaly_score: float):
        """Alert downstream equipment of potential issues"""
        query = """
        MATCH (e:Equipment {id: $equipment_id})
        MATCH (downstream:Equipment)-[:DEPENDS_ON*1..3]->(e)
        RETURN downstream.id as affected_equipment,
               length(path) as distance
        """
        # Implementation...
```

Month 5 Checkpoint ✅

```
□ Neo4J schema for industrial equipment
□ Graph population from metadata
□ Graph-based feature extraction
□ Anomaly propagation algorithm
□ Graph visualization dashboard
```

Skills Acquired: - ✅ Advanced Neo4J (beyond certification) - ✅ Graph data modeling - ✅ Graph algorithms for ML - ✅ Cypher optimization

---

📅 MONTH 6: RAG System for Technicians

Goal: Build Q&A system over equipment manuals and logs

Week 21-22: Document Processing Pipeline

Tasks: - Ingest PDF manuals, maintenance logs - Chunk documents intelligently - Create embeddings with sentence-transformers

Code:

```
# llm-assistant/document_processor.py
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma

class DocumentProcessor:
    def __init__(self):
        self.embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2"
        )
        self.splitter = RecursiveCharacterTextSplitter(
            chunk_size=500,
            chunk_overlap=50,
            separators=["\n\n", "\n", ".", " "]
        )
        self.vectorstore = Chroma(
            persist_directory="./chroma_db",
            embedding_function=self.embeddings
        )

    def ingest_manual(self, pdf_path: str, equipment_id: str):
        loader = PyPDFLoader(pdf_path)
        documents = loader.load()

        # Add metadata
        for doc in documents:
            doc.metadata["equipment_id"] = equipment_id
            doc.metadata["source_type"] = "manual"

        chunks = self.splitter.split_documents(documents)
        self.vectorstore.add_documents(chunks)
```

Week 23-24: RAG Chain with Graph Context

Tasks: - Combine vector search with graph context - Create hybrid retrieval - Build conversational chain

Code:

```
# llm-assistant/rag_chain.py
from langchain.chains import ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI
from langchain.prompts import PromptTemplate

class IndustrialRAG:
    def __init__(self, vectorstore, graph_extractor, llm=None):
        self.vectorstore = vectorstore
        self.graph = graph_extractor
        self.llm = llm or ChatOpenAI(model="gpt-3.5-turbo")

        self.prompt = PromptTemplate(
            template="""You are an expert industrial maintenance assistant.

Equipment Context from Knowledge Graph:
{graph_context}

Relevant Documentation:
{documents}

Current Sensor Readings:
{sensor_data}

User Question: {question}

Provide a helpful, technically accurate response:""",
            input_variables=["graph_context", "documents", "sensor_data", "question"]
        )

    def query(self, question: str, equipment_id: str) -> str:
        # Get graph context
        graph_context = self.graph.get_equipment_context(equipment_id)

        # Get relevant documents
        docs = self.vectorstore.similarity_search(
            question,
            k=3,
            filter={"equipment_id": equipment_id}
        )

        # Get current sensor data
        sensor_data = self.get_current_readings(equipment_id)

        # Generate response
        response = self.llm.invoke(
            self.prompt.format(
                graph_context=graph_context,
                documents=docs,
                sensor_data=sensor_data,
                question=question
            )
        )

        return response.content
```

Month 6 Checkpoint ✅

```
□ Document ingestion pipeline
□ ChromaDB vector store
□ Hybrid retrieval (vector + graph)
□ Conversational RAG chain
□ Chat UI in Streamlit
□ Evaluation on test questions
```

Skills Acquired: - ✅ Advanced RAG architectures - ✅ Hybrid retrieval systems - ✅ Vector databases (ChromaDB) - ✅ LangChain advanced patterns

---

📅 MONTH 7: LLM Fine-tuning

Goal: Fine-tune open-source LLM for industrial domain

Week 25-26: Dataset Preparation

Tasks: - Create instruction-tuning dataset from maintenance logs - Format: (instruction, input, output) triplets - Quality filtering and deduplication

Dataset Format:

```
{
  "instruction": "Diagnose the issue based on the sensor readings",
  "input": "Equipment: CNC_Mill_001\nTemperature: 85°C (normal: 40-70°C)\nVibration: 2.5mm/s (normal: 0.5-1.5mm/s)\nPressure: Normal",
  "output": "The high temperature and elevated vibration suggest bearing wear. Recommended actions:\n1. Inspect spindle bearings\n2. Ch
}
```

Week 27-28: LoRA Fine-tuning

Tasks: - Fine-tune Mistral-7B or LLaMA-2-7B with LoRA - Use QLoRA for memory efficiency - Implement evaluation metrics

Code:

```
# llm-assistant/finetune.py
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
from datasets import load_dataset
from trl import SFTTrainer

# Quantization config for QLoRA
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True
)

# Load model
model = AutoModelForCausalLM.from_pretrained(
    "mistralai/Mistral-7B-v0.1",
    quantization_config=bnb_config,
    device_map="auto"
)

# LoRA config
lora_config = LoraConfig(
    r=16,  # rank
    lora_alpha=32,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

# Prepare model
model = prepare_model_for_kbit_training(model)
model = get_peft_model(model, lora_config)

# Train
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    dataset_text_field="text",
    max_seq_length=512,
    args=TrainingArguments(
        output_dir="./industrial-mistral-lora",
        per_device_train_batch_size=4,
        gradient_accumulation_steps=4,
        num_train_epochs=3,
        learning_rate=2e-4,
        fp16=True,
        logging_steps=10,
        save_strategy="epoch"
    )
)

trainer.train()
```

Resource: [Hugging Face PEFT Guide]() (FREE)

Month 7 Checkpoint ✅

```
□ Instruction-tuning dataset created
□ LoRA fine-tuning pipeline
□ Model evaluation (perplexity, task accuracy)
□ Merged LoRA weights for inference
□ Comparison: base vs fine-tuned
□ Model uploaded to Hugging Face Hub
```

Skills Acquired: - ✅ LLM Fine-tuning (LoRA, QLoRA, PEFT) - ✅ Instruction tuning - ✅ Quantization techniques - ✅ Hugging Face ecosystem

---

📅 MONTH 8: Kubernetes Deployment

Goal: Deploy entire system on Kubernetes

Week 29-30: Kubernetes Fundamentals

Tasks: - Set up local K8s (minikube or kind) - Create deployments for each service - Configure services and ingress

Learn While Building:

```yaml
# kubernetes/deployments/ml-api.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-api
  labels:
    app: ml-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ml-api
  template:
    metadata:
      labels:
        app: ml-api
    spec:
      containers:
      - name: ml-api
        image: industrialmind/ml-api:latest
        ports:
        - containerPort: 8000
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        env:
        - name: MLFLOW_TRACKING_URI
          value: "http://mlflow:5000"
        - name: MODEL_NAME
          value: "anomaly-detector"
        - name: MODEL_STAGE
          value: "Production"
---
apiVersion: v1
kind: Service
metadata:
  name: ml-api
spec:
  selector:
    app: ml-api
  ports:
  - port: 80
    targetPort: 8000
  type: ClusterIP
```

Week 31-32: Helm Charts & Scaling

Tasks: - Create Helm chart for easy deployment - Implement horizontal pod autoscaling - Set up GPU node pool for inference

Helm Chart Structure:

```
industrialmind-chart/
├── Chart.yaml
├── values.yaml
├── templates/
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── ingress.yaml
│   ├── configmap.yaml
│   └── hpa.yaml
```

Resource: [Kubernetes in 1 Hour](#) (FREE)

Month 8 Checkpoint ✅

```
□ All services running on K8s
□ Helm chart for deployment
□ Horizontal Pod Autoscaler
□ Ingress configuration
□ Secrets management
□ Local cluster fully functional
```

Skills Acquired: - ✅ Kubernetes fundamentals - ✅ Helm charts - ✅ Container orchestration - ✅ Scaling strategies

---

🗓 MONTH 9: Cloud Deployment (AWS)

Goal: Deploy to AWS with SageMaker integration

Week 33-34: AWS Infrastructure

Tasks: - Set up EKS cluster - Deploy to AWS with Terraform - Integrate with AWS services

Terraform:

```
# terraform/main.tf
provider "aws" {
  region = "eu-west-1"
}

module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  cluster_name    = "industrialmind-cluster"
  cluster_version = "1.28"

  vpc_id     = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnets

  eks_managed_node_groups = {
    general = {
      desired_size = 2
      min_size     = 1
      max_size     = 4
      instance_types = ["t3.medium"]
    }

    ml = {
      desired_size = 1
      min_size     = 0
      max_size     = 2
      instance_types = ["g4dn.xlarge"]  # GPU
      labels = {
        workload = "ml-inference"
      }
    }
  }
}
```

Week 35-36: SageMaker Integration

Tasks: - Deploy models to SageMaker endpoints - Set up SageMaker Pipelines - Implement A/B testing

Code:

```
# mlops/sagemaker_deploy.py
import sagemaker
from sagemaker.pytorch import PyTorchModel

def deploy_to_sagemaker(model_artifact_path: str):
    role = sagemaker.get_execution_role()

    pytorch_model = PyTorchModel(
        model_data=model_artifact_path,
        role=role,
        framework_version="2.0",
        py_version="py310",
        entry_point="inference.py"
    )

    predictor = pytorch_model.deploy(
        instance_type="ml.g4dn.xlarge",
        initial_instance_count=1,
        endpoint_name="anomaly-detector-prod"
    )

    return predictor
```

Month 9 Checkpoint ✅

```
□ EKS cluster running
□ Terraform infrastructure as code
□ SageMaker endpoints deployed
□ Cost monitoring set up
□ AWS Well-Architected review
□ Pass AWS ML Specialty exam!
```

Skills Acquired: - ✅ AWS EKS - ✅ AWS SageMaker - ✅ Terraform - ✅ Cloud architecture

---

📅 MONTH 10: CI/CD Pipeline

Goal: Fully automated deployment pipeline

Week 37-38: GitHub Actions for ML

Tasks: - Automated testing on PR - Model training on merge to main - Automated deployment to staging

.github/workflows/ml-pipeline.yaml:

```yaml
name: ML Pipeline

on:
  push:
    branches: [main]
    paths:
      - 'ml-models/**'
      - 'data/**'
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run tests
        run: pytest tests/ -v

      - name: Run model tests
        run: python -m pytest ml-models/tests/ -v

  train:
    needs: test
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: eu-west-1

      - name: Train model
        run: |
          python ml-models/anomaly_detector/train.py \
            --mlflow-tracking-uri ${{ secrets.MLFLOW_URI }}

      - name: Register model
        run: python mlops/register_model.py

  deploy-staging:
    needs: train
    runs-on: ubuntu-latest
    environment: staging
    steps:
      - name: Deploy to staging
        run: |
          aws sagemaker update-endpoint \
            --endpoint-name anomaly-detector-staging \
            --endpoint-config-name ${{ steps.train.outputs.config_name }}
```

Week 39-40: Model Validation Gates

Tasks: - Automated model validation before production - Performance regression checks - Data drift detection

Code:

```python
# mlops/validation_gate.py
class ModelValidationGate:
    def __init__(self, production_model, candidate_model):
        self.prod = production_model
        self.candidate = candidate_model

    def validate(self, test_data) -> bool:
        """Check if candidate model passes all gates"""

        # Performance gate
        prod_metrics = self.evaluate(self.prod, test_data)
        candidate_metrics = self.evaluate(self.candidate, test_data)

        if candidate_metrics['f1'] < prod_metrics['f1'] * 0.95:
            return False, "F1 score regression > 5%"
```

```
        # Latency gate
        if candidate_metrics['p99_latency'] > 100:  # ms
            return False, "P99 latency exceeds 100ms"

        # Bias gate
        if self.detect_bias(self.candidate, test_data):
            return False, "Model shows significant bias"

        return True, "All gates passed"
```

Month 10 Checkpoint ✅

```
□ GitHub Actions CI/CD pipeline
□ Automated testing
□ Automated training
□ Model validation gates
□ Staging deployment automation
□ Production deployment with approval
```

Skills Acquired: - ✅ CI/CD for ML - ✅ GitHub Actions - ✅ Automated testing - ✅ Deployment automation

---

📅 MONTH 11: Monitoring & Observability

Goal: Production-grade monitoring

Week 41-42: Prometheus & Grafana

Tasks: - Model performance monitoring - Data drift detection - Alert configuration

Metrics to Track:

```
# api/metrics.py
from prometheus_client import Counter, Histogram, Gauge

# Request metrics
PREDICTION_COUNTER = Counter(
    'predictions_total',
    'Total predictions',
    ['model_version', 'result']
)

PREDICTION_LATENCY = Histogram(
    'prediction_latency_seconds',
    'Prediction latency',
    buckets=[0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1.0]
)

# Model metrics
ANOMALY_SCORE = Histogram(
    'anomaly_score',
    'Distribution of anomaly scores',
    buckets=[0.01, 0.05, 0.1, 0.2, 0.5, 1.0]
)

DATA_DRIFT_SCORE = Gauge(
    'data_drift_score',
    'Current data drift score',
    ['feature']
)
```

Week 43-44: Data Drift & Model Decay

Tasks: - Implement drift detection - Set up alerting - Create Grafana dashboards

Code:

```
# mlops/drift_detection.py
from evidently.metrics import DataDriftTable
from evidently.report import Report

class DriftDetector:
    def __init__(self, reference_data):
        self.reference = reference_data

    def check_drift(self, current_data) -> dict:
        report = Report(metrics=[DataDriftTable()])
        report.run(
            reference_data=self.reference,
            current_data=current_data
        )

        results = report.as_dict()

        # Update Prometheus metrics
```

```
        for feature, drift_score in results['drift_by_feature'].items():
            DATA_DRIFT_SCORE.labels(feature=feature).set(drift_score)

        return results
```

Month 11 Checkpoint ✅

```
☐ Prometheus metrics collection
☐ Grafana dashboards
☐ Data drift detection
☐ Model performance monitoring
☐ Alerting configuration
☐ Runbook documentation
```

Skills Acquired: - ✅ Prometheus & Grafana - ✅ ML monitoring - ✅ Data drift detection - ✅ Observability best practices

---

📅 MONTH 12: Polish & Job Search

Goal: Portfolio-ready project + Active job search

Week 45-46: Documentation & Demo

Tasks: - Complete README with architecture diagram - Create demo video (5-10 minutes) - Write blog post about the project - Prepare technical presentation

README Structure:

```
# IndustrialMind 🏭🧠

> End-to-end ML platform for industrial predictive maintenance

[![CI/CD](badge)](link)
[![License](badge)](link)
[![Demo](badge)](link)

## 🎯 What This Project Demonstrates

- **PyTorch** deep learning for anomaly detection
- **Transformer** architecture for time series
- **MLOps** pipeline with MLflow, DVC
- **Knowledge Graph** with Neo4J
- **LLM/RAG** system with fine-tuned model
- **Kubernetes** deployment at scale
- **AWS SageMaker** integration

## 🏗 Architecture

[Architecture diagram]

## 🚀 Quick Start

```bash
docker-compose up -d
```

📊 Results

Model	F1 Score	Latency (p99)
Autoencoder	0.92	15ms
Transformer	0.95	45ms

[More documentation...]

```

### Week 47-48: Active Job Search

**Tasks:**
- Apply to 50+ positions
- Tailor applications to each role
- Prepare for technical interviews

**Application Tracker:**

| Company | Role | Location | Status | Notes |
|---------|------|----------|--------|-------|
| Google | ML Engineer | Zurich | Applied | |
| Roche | AI Lead | Basel | Applied | |
| G42 | Senior ML | Dubai | Applied | |
| ... | ... | ... | ... | |

### Month 12 Checkpoint ✅
```
```

```
```

☐ README polished ☐ Demo video recorded ☐ Blog post published ☐ 50+ applications sent ☐ Interview prep complete ☐ First interviews scheduled!

```

---

## 📊 SKILLS MATRIX: Before vs After

| Skill | Before | After | Evidence |
|-------|--------|-------|----------|
| PyTorch | ⭐⭐ | ⭐⭐⭐⭐⭐ | 3 production models |
| MLOps | ⭐⭐ | ⭐⭐⭐⭐⭐ | Full pipeline |
| Kubernetes | ⭐⭐ | ⭐⭐⭐⭐ | EKS deployment |
| AWS | ⭐ | ⭐⭐⭐⭐ | SageMaker + cert |
| LLM Fine-tuning | ⭐⭐ | ⭐⭐⭐⭐⭐ | Published model |
| CI/CD | ⭐⭐⭐ | ⭐⭐⭐⭐⭐ | GitHub Actions |
| Monitoring | ⭐⭐ | ⭐⭐⭐⭐ | Prometheus/Grafana |

---

## 💰 EXPECTED OUTCOMES

### Portfolio Impact

After completing IndustrialMind, you'll have:

1. **GitHub repo** with 1000+ commits, production-quality code
2. **Published model** on Hugging Face Hub
3. **Blog posts** showing thought leadership
4. **Demo video** for quick showcasing
5. **Live deployment** (even if small scale)

### Interview Talking Points

Every component maps to interview questions:

| Question | Your Answer |
|----------|-------------|
| "Tell me about a PyTorch project" | IndustrialMind anomaly detector |
| "How do you handle MLOps?" | MLflow + DVC + GitHub Actions |
| "Experience with Kubernetes?" | Deployed entire platform on EKS |
| "LLM experience?" | Fine-tuned Mistral with LoRA |
| "How do you monitor models?" | Prometheus + Grafana + drift detection |

### Salary Expectation

With this portfolio, you can confidently target:

| Region | Role | Expected Salary |
|--------|------|-----------------|
| Switzerland | Senior ML Engineer | CHF 140-170K |
| UAE | ML Lead | AED 500-650K (0% tax) |
| UK | Staff ML Engineer | £100-130K |
| USA | Senior MLE | $180-250K |

---

## 🎬 START NOW

### This Week's Tasks

```

☐ Create GitHub repo: industrialmind ☐ Set up basic project structure ☐ Write initial README ☐ Install Docker and Docker Compose ☐ Start Week 1 tasks ```

Resources You Need

Resource	Purpose	Cost
GitHub	Code hosting	FREE
Docker Desktop	Containers	FREE
AWS Free Tier	Cloud (12 months)	FREE
Hugging Face	Models	FREE
MLflow	Tracking	FREE
Neo4J Community	Graph DB	FREE

Total Cost: €0 (using free tiers)

📞 ACCOUNTABILITY

Weekly Check-in Template

Every Sunday, answer:

1. ✅ What did I ship this week?
2. 🎯 What's the goal for next week?
3. 🚧 What's blocking me?
4. 📚 What did I learn?

Monthly Milestones

Month	Must Ship	Nice to Have
1	Data pipeline	Performance optimization
2	PyTorch model	Advanced architecture
3	MLflow integration	Automated retraining
4	Transformer model	Multi-task learning
5	Knowledge Graph	Graph neural network
6	RAG system	Hybrid retrieval
7	Fine-tuned LLM	Multi-modal
8	K8s deployment	Auto-scaling
9	AWS deployment	Multi-region
10	CI/CD pipeline	Canary deployments
11	Monitoring	Automated remediation
12	Job offers!	Multiple offers

"*The best time to plant a tree was 20 years ago. The second best time is now.*"

Let's build. 🚀