🔬 Lab: Design an End-to-End Data Pipeline for AI

Goal:
Create a full AI pipeline that connects ETL → Model Training → Inference using GPU-accelerated components and efficient data flow techniques.

Tools Used:

- **Python** (PyTorch or TensorFlow)
- **NVIDIA DALI or cuDF**
- **Docker**
- **Triton Inference Server**
- **Kafka** (or simulated data stream)
- **Jupyter Notebook**
- **NGC containers (optional)**

✅ Lab Setup

Step 1: Prepare Environment

1. Spin up a GPU-enabled VM or use a local workstation with at least 1 NVIDIA GPU (A100, V100, or RTX).
2. Install the following:

```
pip install torch torchvision torchaudio
pip install nvidia-pyindex nvidia-dali-cuda110
```

```
pip install jupyterlab
```

3. (Optional) Pull NGC containers for consistency:

```
docker pull nvcr.io/nvidia/pytorch:24.03-py3
docker pull nvcr.io/nvidia/tritonserver:24.03-py3
```

🔧 PART 1: ETL Pipeline (Extract → Transform → Load)

Step 2: Simulate or Pull Dataset

- Use CIFAR-10 or MNIST for fast experimentation:

```
from torchvision.datasets import CIFAR10
from torchvision import transforms
dataset = CIFAR10(root='./data', download=True)
```

Step 3: Preprocess with NVIDIA DALI (optional but recommended)

- Use DALI to accelerate preprocessing:

```
from nvidia.dali.pipeline import Pipeline
import nvidia.dali.ops as ops
import nvidia.dali.types as types

class SimplePipeline(Pipeline):
    def __init__(self, batch_size, num_threads, device_id):
        super(SimplePipeline, self).__init__(batch_size, num_threads, dev
        self.input = ops.FileReader(file_root="data/cifar10/train")
        self.decode = ops.ImageDecoder(device="mixed", output_type=types.
        self.resize = ops.Resize(resize_x=32, resize_y=32)
        self.cmnp = ops.CropMirrorNormalize(device="gpu",
                        output_dtype=types.FLOAT,
```

```
                                output_layout=types.NCHW,
                                mean=[0.5*255]*3,
                                std=[0.5*255]*3)

        def define_graph(self):
            jpegs, labels = self.input()
            images = self.decode(jpegs)
            images = self.resize(images)
            output = self.cmnp(images)
            return output, labels
```

🧠 PART 2: Model Training

Step 4: Create or Load Model

```
import torch.nn as nn
import torch.optim as optim

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(32*32*3, 128),
    nn.ReLU(),
    nn.Linear(128, 10)
)
```

Step 5: Train Model

```
from torch.utils.data import DataLoader
from torchvision import transforms

transform = transforms.Compose([transforms.ToTensor()])
train_loader = DataLoader(dataset, batch_size=64, shuffle=True)

optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
criterion = nn.CrossEntropyLoss()

for epoch in range(5):
    for images, labels in train_loader:
        preds = model(images)
        loss = criterion(preds, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Step 6: Save Model for Deployment

```
torch.save(model.state_dict(), "model.pth")
```

⚙️ PART 3: Model Inference Deployment (Triton)

Step 7: Convert Model to Triton Format

Create model repository structure:

```
model_repository/
└── my_model/
    ├── 1/
    |   └── model.pt
    └── config.pbtxt
```

Sample `config.pbtxt`:

```
name: "my_model"
platform: "pytorch_libtorch"
max_batch_size: 8
input [
```

```
  {
    name: "INPUT__0"
    data_type: TYPE_FP32
    dims: [3, 32, 32]
  }
]
output [
  {
    name: "OUTPUT__0"
    data_type: TYPE_FP32
    dims: [10]
  }
]
```

Step 8: Run Triton Server

```
docker run --gpus all --rm -p8000:8000 -p8001:8001 -p8002:8002 \
  -v$(pwd)/model_repository:/models \
  nvcr.io/nvidia/tritonserver:24.03-py3 \
  tritonserver --model-repository=/models
```

🌐 PART 4: Inference Client

Step 9: Send Data to Model

```
import numpy as np
import tritonclient.http as httpclient

client = httpclient.InferenceServerClient(url="localhost:8000")

inputs = httpclient.InferInput("INPUT__0", [1, 3, 32, 32], "FP32")
inputs.set_data_from_numpy(np.random.rand(1, 3, 32, 32).astype(np.float32))

outputs = httpclient.InferRequestedOutput("OUTPUT__0")
```

```python
result = client.infer("my_model", inputs=[inputs], outputs=[outputs])
print(result.as_numpy("OUTPUT__0"))
```

📈 PART 5: Monitoring and Optimization

Step 10: Measure Performance

- Use `nvidia-smi` to monitor GPU utilization during training and inference
- Use `perf_analyzer` (comes with Triton) to benchmark:

  ```
  perf_analyzer -m my_model -b 8 -u localhost:8000
  ```

✅ Deliverables for Lab Completion

- Screenshot of successful training loop
- Screenshot of Triton running and responding to inference
- Screenshot or CSV of performance analyzer output
- Diagram of pipeline architecture (ETL → Training → Inference)
