🚀 Deep Learning Base GPU AMI - Usage Instructions

📋 Quick Start Guide

1. Launch Your Instance

markdown
AWS Console Steps:
1. Go to **EC2 Dashboard** → **Launch Instance**
2. Search for: "Deep Learning Base GPU AMI - Ubuntu 24.04"
3. Select instance type: **g4dn.xlarge** (recommended)
4. Configure storage: **100GB+** (gp3 recommended)
5. Launch with your existing key pair

2. Connect to Your Instance

bash
SSH Connection with Jupyter port forwarding
ssh -i "your-key.pem" -L 8888:localhost:8888 ubuntu@YOUR_INSTANCE_IP

Example:
ssh -i "dl-key.pem" -L 8888:localhost:8888 ubuntu@54.210.135.123

3. Access Jupyter Lab

markdown
After SSH connection:
1. Open your web browser
2. Navigate to: **http://localhost:8888**
3. Start coding immediately - no password required

🔧 Environment Overview

Pre-Configured Tools & Access

bash
Quick environment verification
nvidia-smi   # Check GPU status
nvcc --version   # Verify CUDA installation
systemctl status jupyter   # Check Jupyter service

Access pre-configured Python environment
source /opt/venvs/dl-env/bin/activate

```
python -c "import torch; print(f'PyTorch: {torch.__version__}, CUDA:
{torch.cuda.is_available()}')"
```

Utility Scripts Included

bash
Verification script
check-dl-setup.sh

Activate Deep Learning environment
activate-dl.sh

Start Jupyter manually
start-jupyter.sh

🧪 Getting Started with Deep Learning

1. Test Your GPU Environment

Create a new notebook in Jupyter Lab and run:
python
# GPU Verification Test
import torch
import tensorflow as tf

print("🚀 Deep Learning Environment Check")
print("=" * 40)

# PyTorch Check
print(f"PyTorch Version: {torch.__version__}")
print(f"PyTorch CUDA Available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"GPU Name: {torch.cuda.get_device_name(0)}")
    print(f"GPU Memory: {torch.cuda.get_device_properties(0).total_memory / 1e9:.1f} GB")

# TensorFlow Check
print(f"TensorFlow Version: {tf.__version__}")
print(f"TensorFlow GPUs: {len(tf.config.list_physical_devices('GPU'))}")

# Performance Test
if torch.cuda.is_available():
    device = torch.device('cuda')
    x = torch.randn(1000, 1000).to(device)
    y = torch.randn(1000, 1000).to(device)
    z = x @ y
    print("✅ GPU Matrix Multiplication: SUCCESS")

2. Your First Deep Learning Model

python

```
# Simple CNN Example in PyTorch
import torch
import torch.nn as nn
import torch.optim as optim

# Define a simple CNN
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = torch.flatten(x, 1)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Move model to GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = SimpleCNN().to(device)
print(f"Model moved to: {device}")

# Test with dummy data
dummy_input = torch.randn(1, 1, 28, 28).to(device)
output = model(dummy_input)
print(f"Output shape: {output.shape}")
```

📁 Project Structure & Best Practices

Recommended Directory Structure

text

```
/workspace/
├── notebooks/          # Jupyter notebooks
├── scripts/            # Python scripts
├── data/               # Datasets
│   ├── raw/            # Original data
│   ├── processed/      # Preprocessed data
│   └── models/         # Trained models
```

```
├── src/              # Source code
└── experiments/      # Experiment tracking
```

Create Your Project Workspace

bash
```
# Create organized directory structure
mkdir -p /workspace/{notebooks,scripts,data/{raw,processed,models},src,experiments}

# Set proper permissions
chmod -R 755 /workspace
```

🔧 Advanced Usage

Managing the Jupyter Service

bash
```
# Check service status
sudo systemctl status jupyter

# Restart Jupyter service
sudo systemctl restart jupyter

# View Jupyter logs
sudo journalctl -u jupyter -f

# Stop Jupyter service
sudo systemctl stop jupyter

# Start Jupyter manually (alternative)
/opt/venvs/dl-env/bin/jupyter lab --allow-root --ip=0.0.0.0 --port=8888
```

Installing Additional Packages

bash
```
# Activate the environment first
source /opt/venvs/dl-env/bin/activate

# Install new packages
pip install transformers datasets plotly seaborn

# Or use the direct path
/opt/venvs/dl-env/bin/pip install package-name
```

Using Docker (Optional)

bash

```
# Build custom Docker images
docker build -t my-dl-project .

# Run with GPU access
docker run --gpus all -p 8888:8888 my-dl-project
```

💾 Data Management

Working with Large Datasets

```
python
# Efficient data loading example
from torch.utils.data import DataLoader, Dataset
import torch

class CustomDataset(Dataset):
    def __init__(self, data_path):
        self.data = torch.load(data_path)

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

    def __getitem__(self, idx):
        return self.data[idx]

# Use DataLoader with multiple workers
dataset = CustomDataset('/workspace/data/processed/train.pt')
dataloader = DataLoader(dataset, batch_size=32, shuffle=True, num_workers=4)
```

AWS S3 Integration

```
python
import boto3
import pandas as pd

# Access S3 data
s3 = boto3.client('s3')

# Download dataset from S3
s3.download_file('my-bucket', 'datasets/train.csv', '/workspace/data/raw/train.csv')

# Load into pandas
df = pd.read_csv('/workspace/data/raw/train.csv')
```

🎯 Performance Optimization

GPU Memory Management

python
```
import torch

# Clear GPU cache
torch.cuda.empty_cache()

# Monitor GPU memory
print(f"Allocated: {torch.cuda.memory_allocated() / 1e9:.2f} GB")
print(f"Cached: {torch.cuda.memory_reserved() / 1e9:.2f} GB")

# Use mixed precision for faster training
from torch.cuda.amp import autocast, GradScaler

scaler = GradScaler()

with autocast():
    outputs = model(inputs)
    loss = criterion(outputs, targets)

scaler.scale(loss).backward()
scaler.step(optimizer)
scaler.update()
```

Multi-GPU Training (if using multi-GPU instances)

python
```
import torch
import torch.nn as nn

# Use DataParallel for multiple GPUs
if torch.cuda.device_count() > 1:
    print(f"Using {torch.cuda.device_count()} GPUs!")
    model = nn.DataParallel(model)
```

🔍 Monitoring & Debugging

System Monitoring Commands

bash
```
# GPU utilization
nvidia-smi

# System resources
```

```
htop

# Disk usage
df -h

# Jupyter server info
/opt/venvs/dl-env/bin/jupyter --version
```

Common Issue Resolution

bash

```
# If Jupyter won't start
sudo systemctl restart jupyter

# If GPU not detected
sudo reboot

# If out of disk space
sudo apt clean
docker system prune -a

# Check service logs
sudo journalctl -u jupyter -n 50
```