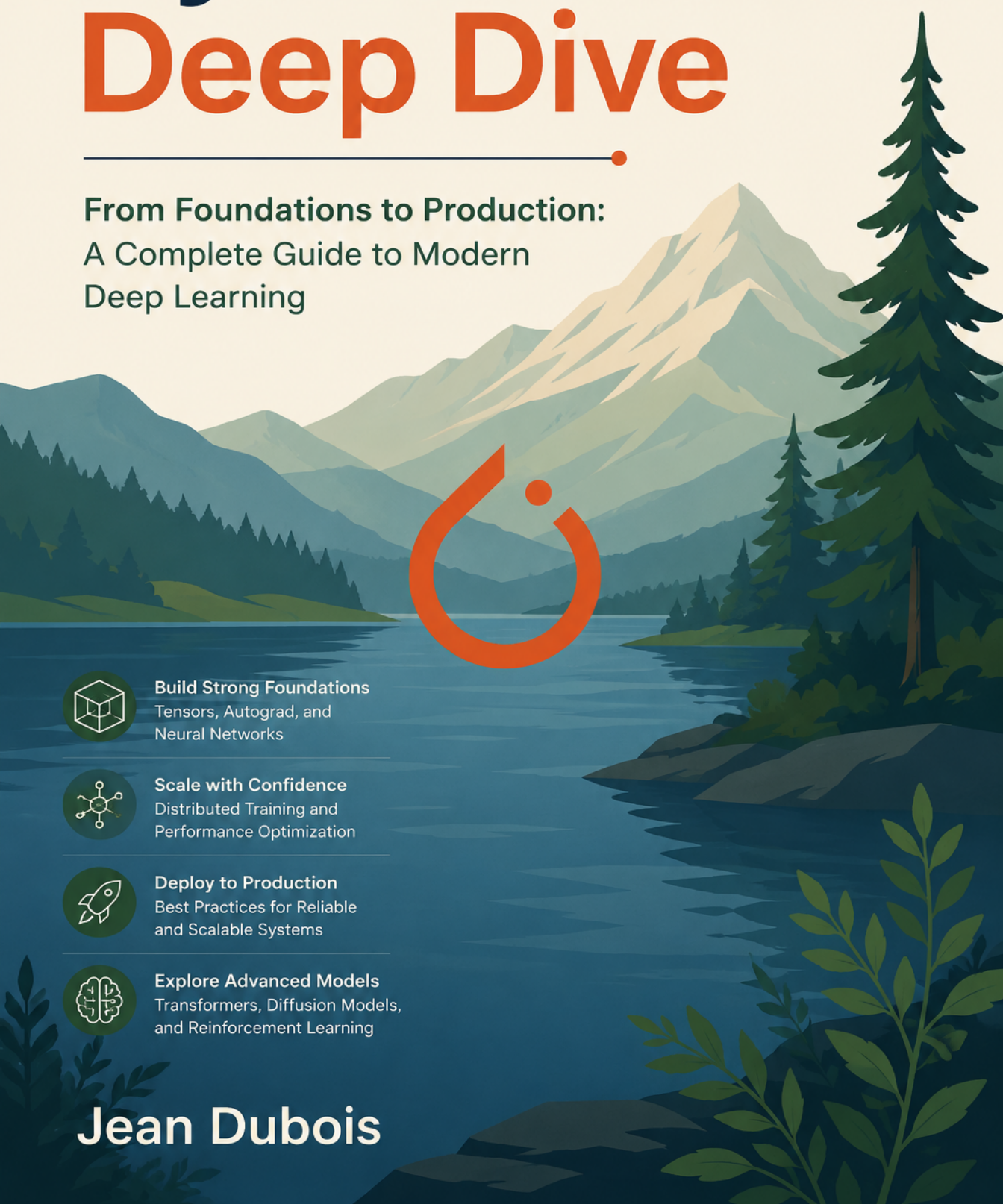
PyTorch Deep Dive

From Foundations to Production: A Complete Guide to Modern Deep Learning



Build Strong Foundations
Tensors, Autograd, and Neural Networks

Scale with Confidence
Distributed Training and Performance Optimization

Deploy to Production
Best Practices for Reliable and Scalable Systems

Explore Advanced Models
Transformers, Diffusion Models, and Reinforcement Learning

Jean Dubois

PyTorch Deep Dive

From Foundations to Production: A Complete Guide to Modern Deep Learning

Steve T. Publications

This book is available at https://leanpub.com/pytorchdeepdive

This version was published on 2026-07-13

Contents

From Foundations to Production: A Complete Guide to Modern Deep Learning

About this book: PyTorch has become the dominant deep learning framework in both research and industry, prized for its Pythonic design, dynamic computation graphs, and rapidly maturing ecosystem. This book takes you from absolute beginner to expert-level practitioner, covering every facet of the PyTorch ecosystem with detailed explanations, fully working code examples, and production-ready best practices. You will learn tensor operations, automatic differentiation, neural network construction, distributed training, model optimization, deployment strategies, and advanced architectures including transformers, diffusion models, and reinforcement learning. Whether you are building your first neural network or deploying large-scale production systems, this book serves as both a learning path and a lasting reference.

Introduction: Why PyTorch, Why Now

In 2017, at the Neural Information Processing Systems (NePS) conference, a surprising shift was underway. For years, TensorFlow had dominated the deep learning landscape with its static computation graph and industrial-grade deployment tools. But researchers were quietly migrating to a newer framework called PyTorch, and the momentum was building fast. By 2023, PyTorch had become the framework of choice for over 60 percent of published machine learning papers, and it has only consolidated that lead since then [1].

This book is not a tutorial collection. It is a comprehensive treatment of PyTorch from first principles to production deployment, designed for readers who want to understand not just how to use the framework, but why things work the way they do. Every concept builds on previous chapters, and every code example is production-ready rather than toy-level.

The Define-by-Run Revolution

The fundamental design decision that set PyTorch apart from its predecessors is the "define-by-run" computation model. In TensorFlow 1.x, you first defined a static computation graph, then executed it within a session. This meant that the entire computational structure had to be known before any computation took place. Debugging was painful because errors only manifested during execution, not during graph construction. Variable-length sequences required special placeholders, and conditional logic in the computation graph was awkward at best.

PyTorch takes a fundamentally different approach. The computation graph is built dynamically, on-the-fly, as Python code executes. Every tensor operation you write immediately becomes part of a directed acyclic graph (DAG) that tracks how outputs depend on inputs. This means you can use ordinary Python control flow, print intermediate values for debugging, and even modify the network architecture mid-computation based on input data. The trade-off is that the graph must be rebuilt each iteration, but PyTorch mitigates this cost through clever caching and, in PyTorch 2.x, through its torch.compile system.

The define-by-run philosophy has profound implications for research velocity. When you can write a novel architecture in code that reads almost like pseudocode, the gap between idea and experiment shrinks dramatically. This is why PyTorch won over the research community so decisively, and why it has become the lingua franca of machine learning research.

What You Will Learn

This book is organized into fifteen chapters that progress logically from foundations to mastery:

- **Chapters 1 through 3** establish the core abstractions: tensors as data structures, autograd as the differentiation engine, and nn.Module as the neural network container. These chapters form the foundation for everything that follows.
- **Chapters 4 through 6** cover the training pipeline: data loading with Dataset and DataLoader, building training loops from scratch, optimizers, loss functions, learning rate schedulers, and mixed precision training. By the end of Chapter 6, you will be able to train any standard neural network architecture.
- **Chapters 7 through 8** address scaling and engineering: GPU acceleration, distributed training with DDP and FSDP, experiment tracking, profiling, debugging, and reproducibility. These are the skills that separate hobbyists from professionals.
- **Chapter 9** covers model serialization, checkpointing, TorchScript, quantization, and pruning – everything you need to take a trained model into production.
- **Chapter 10** surveys the PyTorch ecosystem: torchvision for computer vision, torchaudio for audio processing, torchtext for natural language, and integration with Hugging Face's Transformers library.
- **Chapters 11 through 13** are devoted to advanced architectures and end-to-end projects. You will build transformers from scratch, implement image classification and object detection systems, train generative models including GANs and diffusion models, and explore reinforcement learning with PPO.
- **Chapters 14 through 15** cover deployment and performance optimization: ONNX export, TorchServe, containerization, production monitoring, torch.compile deep dives, memory optimization, and the latest best practices.

How to Read This Book

Read this book sequentially. Every chapter assumes knowledge from the previous ones, and skipping ahead will leave gaps in your understanding. The code examples are designed to be complete and runnable; copy them into your own environment and experiment with modifications. When you encounter a concept that feels abstract, look for the concrete example that follows – every theoretical explanation is grounded in working code.

Some chapters are denser than others. Chapters 3 (autograd), 7 (distributed training), and 11 (transformers) contain concepts that benefit from slow, careful reading. Do not rush through them. The payoff in understanding will compound throughout the rest of the book.

Prerequisites

This book assumes you are comfortable with Python, including object-oriented programming, list comprehensions, and working with libraries like NumPy. Basic linear algebra (matrix multiplication, vector operations) and calculus (derivatives, chain rule) will help you follow the mathematical explanations, though we introduce each concept from first principles. No prior experience with deep learning or PyTorch is required.

Versions and Compatibility

This book targets PyTorch 2.x and its associated ecosystem libraries. The core APIs discussed here have been stable across PyTorch 2.0 through current releases, but specific features like torch.compile continue to evolve rapidly. Where version-specific behavior matters, we note it explicitly. All code examples are tested against recent stable releases and should work with any PyTorch 2.x installation.

Let us begin.

Chapter 1: Foundations of Deep Learning and PyTorch

What Is Deep Learning: A Brief History

Deep learning is a subfield of machine learning that uses multi-layered neural networks to learn hierarchical representations of data. The term "deep" refers to the depth of these networks – the number of stacked layers that transform raw input into increasingly abstract features. Before a network recognizes a cat in an image, its early layers detect edges, then textures, then shapes, and finally object parts. This hierarchy of features is what makes deep learning so powerful for perception tasks.

The story of deep learning begins with the perceptron, a single-neuron model proposed by Frank Rosenblatt in 1958. The perceptron could learn linear decision boundaries but failed on problems like XOR, which require non-linear separation. In 1969, Marvin Minsky and Seymour Papert published "Perceptrons," a book that catalogued these limitations and effectively stalled neural network research for over a decade.

The revival came in the 1980s with backpropagation, independently popularized by Rumelhart, Hinton, and Williams in 1986. Backpropagation is an application of the chain rule from calculus that efficiently computes gradients through multiple layers of a network. With backpropagation, networks could learn non-linear representations by stacking layers and adjusting weights based on gradient signals.

The next breakthrough was ReLU (Rectified Linear Unit), introduced by Glorot, Bordes, and Bengio in 2010 as an alternative to the sigmoid activation function. ReLU solved the vanishing gradient problem that plagued deep networks with sigmoid activations, enabling training of much deeper architectures. Alex Krizhevsky's ImageNet-winning entry in 2012 (AlexNet) combined deep convolutional layers, ReLU activations, GPU acceleration, and dropout regularization to achieve unprecedented accuracy on image classification.

Since then, the trajectory has been clear: deeper networks, more data, better optimization algorithms (Adam, introduced by Kingma and Ba in 2014), and increasingly sophisticated architectures. The transformer architecture, introduced by Vaswani et al. in 2017 ("Attention Is All You Need"), replaced recurrent networks for sequence modeling and became the foundation of large language models. Deep learning has evolved from a research curiosity into the backbone of modern artificial intelligence.

Why PyTorch: Philosophy and Design Principles

PyTorch was created by Facebook's Artificial Intelligence Research (FAIR) team, led by Soumith Chintala, as a Python successor to Torch, a Lua-based scientific computing framework. It was first released in 2016 and has since become the dominant deep learning framework in research and an increasingly popular choice in production.

Several design principles explain PyTorch's success:

Pythonic First-Class Design. PyTorch is not a C++ library with a Python wrapper bolted on. It is designed from the ground up to feel like natural Python. Tensors are first-class objects that you can manipulate with intuitive operations. Models are regular Python classes. Training loops are regular Python for-loops. This means you can use every tool in your existing Python toolkit: debuggers, profilers, IDE features, and third-party libraries.

Dynamic Computation Graphs. Unlike static-graph frameworks that require you to define the entire computation before execution, PyTorch builds the computation graph dynamically as operations execute. This "define-by-run" approach means you can use normal Python control flow (if statements, loops, recursion) within your model's forward pass. The architecture can change from one forward pass to the next based on input data, which is essential for variable-length sequences, dynamic pruning, and research exploration.

Eager Execution. By default, PyTorch executes operations immediately when you call them. There is no separate "build" phase followed by a "run" phase. This makes debugging straightforward: you can print intermediate values, use pdb.set_trace(), and inspect tensors at any point during execution. When you need graph-level optimization, PyTorch 2.x provides torch.compile as an opt-in feature rather than the default mode.

Composable Design. PyTorch's architecture is built on composable abstractions. Tensors are the data layer, autograd is the differentiation layer, nn.Module is the model container, and optimizers are the update rules. Each layer is independently usable and well-documented. You can use tensors without building models, use autograd for scientific computing without neural networks, and combine components in novel ways that the framework does not explicitly support.

Strong GPU Support. PyTorch provides seamless CPU-to-GPU transfer with a single `.to(device)` call. The same code runs on CPU or GPU without modification. Under the hood, PyTorch leverages CUDA for NVIDIA GPUs, ROCm for AMD GPUs, and Intel's oneDNN for CPUs, providing a unified interface across hardware backends.

Thriving Ecosystem. The PyTorch ecosystem has matured significantly. torchvision provides vision-specific datasets, transforms, and pre-trained models. torchaudio handles audio processing. torchtext covers text pipelines. Hugging Face's Transformers library is built on PyTorch. ONNX export enables cross-framework deployment. TorchServe provides production model serving. This ecosystem means you rarely need to reinvent infrastructure.

The Dynamic Computation Graph Explained

Understanding the dynamic computation graph is essential to understanding PyTorch itself. Let us build intuition with a concrete example.

Consider a simple computation: `z = x * y + w`, where `x`, `y`, and `w` are tensors that require gradient tracking. In PyTorch, you would write:

```
import torch

x = torch.tensor(2.0, requires_grad=True)
y = torch.tensor(3.0, requires_grad=True)
w = torch.tensor(1.0, requires_grad=True)

z = x * y + w
z.backward()

print(x.grad)  # tensor(3.) -- derivative of z with respect to x
print(y.grad)  # tensor(2.) -- derivative of z with respect to y
print(w.grad)  # tensor(1.) -- derivative of z with respect to w
```

Here is what happens behind the scenes:

1. When you create `x`, `y`, and `w` with `requires_grad=True`, PyTorch marks these tensors as leaf nodes in a computation graph. Each tensor internally stores a reference to a gradient function (grad_fn) that knows how to compute gradients for the operation that created it. Leaf tensors have no grad_fn because they are user-created, not the result of an operation.
2. When you execute `z = x * y + w`, PyTorch performs two operations: first a multiplication (`x * y`), then an addition (`+ w`). Each operation creates a new tensor and records a gradient function in the computation graph. The multiplication creates a MulBackward node, and the addition creates an AddBackward node. These nodes form a directed acyclic graph where edges point from inputs to outputs.
3. When you call `z.backward()`, PyTorch traverses this graph in reverse (hence "backward" propagation), applying the chain rule at each node. The gradient flows from `z` back through the addition node, then through the multiplication node, finally reaching the leaf tensors `x`, `y`, and `w`. Each leaf tensor accumulates its gradient in the `.grad` attribute.

The graph is rebuilt from scratch on every forward pass. This means that if your forward pass contains conditional logic, the graph structure can change between iterations:

```
import torch

x = torch.tensor(2.0, requires_grad=True)
y = torch.tensor(3.0, requires_grad=True)

if x > 0:
    z = x * y
else:
    z = x + y

z.backward()
# The graph only contains the multiplication path
# because the condition was true
```

This flexibility is what makes PyTorch so powerful for research. You can implement architectures that dynamically change their structure based on input data – something that is extremely difficult or impossible with static-graph frameworks.

The trade-off is performance. Rebuilding the graph each iteration adds overhead. In PyTorch 1.x, this was accepted as the cost of flexibility. In PyTorch 2.x, torch.compile captures the computation graph across iterations and optimizes it, giving you both flexibility and performance. We will explore torch.compile in detail in Chapter 15.

Setting Up Your Environment: Installation, CUDA, Conda

A proper development environment is the foundation of a productive PyTorch workflow. Let us walk through the recommended setup.

Using Conda. Conda is the recommended package manager for PyTorch development because it handles both Python packages and system-level dependencies (like CUDA libraries) in a unified way. Create a dedicated environment for your PyTorch work:

```
# Create a new conda environment with Python 3.10+
conda create -n pytorch-env python=3.10 -y
conda activate pytorch-env

# Install PyTorch with CUDA support (adjust version as needed)
conda install pytorch torchvision torchaudio pytorch-cuda=12.4 -c pytorch -c
↪  nvidia
```

PyTorch provides pre-built binaries for multiple CUDA versions. Choose the CUDA version that matches your GPU driver. NVIDIA drivers are forward-compatible, meaning a newer driver supports older CUDA versions. Check your driver's supported CUDA version with `nvidia-smi`.

Using pip. If you prefer pip, PyTorch provides an installer at pytorch.org that generates the correct command for your configuration:

```
# For CUDA 12.4
pip install torch torchvision torchaudio --index-url
↪  https://download.pytorch.org/whl/cu124

# For CPU-only (no GPU)
pip install torch torchvision torchaudio --index-url
↪  https://download.pytorch.org/whl/cpu
```

Verifying Your Installation. After installation, verify that PyTorch is working correctly:

```python
import torch

# Check version
print(f"PyTorch version: {torch.__version__}")

# Check CUDA availability
print(f"CUDA available: {torch.cuda.is_available()}")

# List available GPUs
if torch.cuda.is_available():
    print(f"GPU count: {torch.cuda.device_count()}")
    print(f"Current GPU: {torch.cuda.get_device_name(0)}")
    print(f"GPU memory (MB): {torch.cuda.mem_get_info(0)}")

# Run a simple computation on GPU
if torch.cuda.is_available():
    x = torch.randn(3, 3, device='cuda')
    y = torch.randn(3, 3, device='cuda')
    z = x @ y  # Matrix multiplication on GPU
    print(f"GPU computation result shape: {z.shape}")
```

Development Tools. For a productive development experience, install these additional packages:

```bash
# Experiment tracking
pip install wandb tensorboard

# Visualization
pip install matplotlib seaborn

# Data handling
pip install pandas numpy scikit-learn

# Hugging Face ecosystem (for NLP projects)
pip install transformers datasets accelerate peft bitsandbytes

# Profiling and debugging
pip install torch-tb-profiler
```

Version Pinning for Reproducibility. In production environments, pin your PyTorch version to ensure reproducibility:

```
# Save exact versions
conda env export > environment.yml

# Or with pip
pip freeze > requirements.txt
```

Your First PyTorch Program: Hello World to Tensors

Let us write a complete, minimal program that demonstrates the core PyTorch workflow: creating tensors, defining a simple model, computing a loss, and performing a gradient update.

```
import torch
import torch.nn as nn

# Step 1: Create some data
# Simulating a simple linear relationship: y = 2x + 3 with noise
torch.manual_seed(42)  # For reproducibility
x = torch.randn(100, 1)       # 100 samples, 1 feature
y = 2.0 * x + 3.0 + torch.randn(100, 1) * 0.1  # True relationship + noise

# Step 2: Define a simple linear model
model = nn.Linear(1, 1)  # One input, one output

# Step 3: Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Step 4: Training loop
num_epochs = 500
for epoch in range(num_epochs):
    # Forward pass: compute predictions
    y_pred = model(x)

    # Compute loss
    loss = criterion(y_pred, y)

    # Backward pass: compute gradients
    optimizer.zero_grad()  # Clear previous gradients
    loss.backward()        # Compute new gradients

    # Update weights
    optimizer.step()

    # Log progress every 100 epochs
    if (epoch + 1) % 100 == 0:
```

```
        weight = model.weight.item()
        bias = model.bias.item()
        print(f"Epoch {epoch + 1}/{num_epochs} -- "
              f"Loss: {loss.item():.4f}, Weight: {weight:.4f}, Bias: {bias:.4f}")

# Step 5: Evaluate the trained model
print(f"\nTrue relationship: y = 2x + 3")
print(f"Learned relationship: y = {model.weight.item():.4f}x +
↪  {model.bias.item():.4f}")

# Make predictions on new data
x_test = torch.tensor([[1.0], [2.0], [3.0]])
y_test = model(x_test)
print(f"\nPredictions for x = [1, 2, 3]: {y_test.flatten().tolist()}")
```

This program demonstrates the canonical PyTorch training loop:

1. **Forward pass**: Data flows through the model to produce predictions.
2. **Loss computation**: A loss function measures the difference between predictions and targets.
3. **Backward pass**: Gradients are computed via autograd's chain rule.
4. **Parameter update**: The optimizer adjusts weights in the direction that reduces loss.

Each of these steps will be explored in depth in subsequent chapters. For now, notice how the code reads almost like pseudocode – that is the PyTorch philosophy in action.

Verifying Your Setup and Understanding Device Placement

Device placement is one of the most common sources of confusion for beginners. In PyTorch, every tensor lives on a specific device (CPU or GPU), and all operands in a computation must reside on the same device. Attempting to operate on tensors from different devices raises a RuntimeError.

Here is how to manage device placement correctly:

```python
import torch

# Check what devices are available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Create a tensor on the default device (CPU)
x_cpu = torch.randn(3, 3)
print(f"x_cpu device: {x_cpu.device}")

# Move tensor to GPU
if torch.cuda.is_available():
    x_gpu = x_cpu.to(device)
    print(f"x_gpu device: {x_gpu.device}")

    # Or create directly on GPU
    y_gpu = torch.randn(3, 3, device=device)
    print(f"y_gpu device: {y_gpu.device}")

    # Operations work seamlessly when all operands are on the same device
    z_gpu = x_gpu @ y_gpu
    print(f"z_gpu device: {z_gpu.device}")

    # Moving results back to CPU for inspection
    z_cpu = z_gpu.cpu()
    print(f"z_cpu device: {z_cpu.device}")

# Best practice: define a device variable and use it throughout
def get_device():
    """Return the best available device."""
    if torch.cuda.is_available():
        return torch.device("cuda")
    elif torch.backends.mps.is_available():
        # Apple Silicon support via Metal Performance Shaders
        return torch.device("mps")
    else:
        return torch.device("cpu")

device = get_device()
```

Common Device-Related Errors:

Error	Cause	Solution
`Expected all tensors to be on the same device`	Mixed CPU/GPU operands	Move all tensors to the same device before computation

Error	Cause	Solution
`CUDA out of memory`	GPU memory exhausted	Reduce batch size, use gradient checkpointing, or clear cached memory with `torch.cuda.empty_cache()`
`Found no NVIDIA driver on your system`	CUDA not properly installed	Verify NVIDIA driver and CUDA toolkit installation

Multi-GPU Considerations. When multiple GPUs are available, you can target a specific GPU:

```
# Use GPU 1 specifically
device = torch.device("cuda:1")
x = torch.randn(3, 3, device=device)

# Check memory across all GPUs
for i in range(torch.cuda.device_count()):
    mem_allocated = torch.cuda.memory_allocated(i)
    mem_reserved = torch.cuda.memory_reserved(i)
    print(f"GPU {i}: allocated={mem_allocated/1e6:.1f}MB,
    ↪  reserved={mem_reserved/1e6:.1f}MB")
```

Apple Silicon (MPS). PyTorch 1.12+ supports Apple's Metal Performance Shaders (MPS) for GPU acceleration on Mac computers with M-series chips. The API is similar to CUDA:

```
if torch.backends.mps.is_available():
    mps_device = torch.device("mps")
    x = torch.ones(3, 3, device=mps_device)
    print(f"MPS tensor: {x}")
```

Note that MPS support is still maturing and not all operations are accelerated. Fall back to CPU for unsupported operations.

Summary

This chapter has established the foundation for everything that follows. You now understand why PyTorch became the dominant deep learning framework, how its dynamic computation graph works, how to set up a development environment, and the basic structure of a training loop. The next chapter dives deep into tensors, the fundamental data structure that underlies every PyTorch operation.

Chapter 2: Tensors – The Building Blocks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

What Are Tensors: Mathematical Foundations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Creating Tensors: From NumPy to Random to Zeros

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Tensor Properties: Shape, Dtype, Device, and Strides

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Element-wise Operations and Broadcasting Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Linear Algebra: Matrix Multiplication, SVD, and Eigenvalues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Indexing, Slicing, Reshaping, and View Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

In-place Operations and Memory Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Performance Tips: Contiguous Memory and Coalesced Access

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 3: Automatic Differentiation with Autograd

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

The Mathematics of Gradients and Backpropagation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

How Autograd Builds the Computation Graph Dynamically

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Gradient Computation: backward(), retain_graph, and detach()

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Gradient Accumulation for Large Batches

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Hook Functions: Inspecting Activations and Gradients

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Custom Autograd Functions: Extending the Engine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Common Pitfalls: Detaching, In-place Modifications, and NaN Gradients

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Debugging Your Computation Graph

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 4: Building Neural Networks with nn.Module

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

The nn.Module Base Class: Parameters, Buffers, and Submodules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Building Blocks: Linear Layers, Convolutional Layers, Pooling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Activation Functions: ReLU, GELU, Swish, and When to Use Each

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Normalization Layers: BatchNorm, LayerNorm, GroupNorm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Container Classes: Sequential, ModuleList, ModuleDict

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Designing the Forward Method: Control Flow and Dynamic Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Custom Layers: Writing Your Own nn.Module Subclasses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Parameter Inspection and Weight Initialization Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 5: Data Loading with Datasets and DataLoaders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

The Dataset Abstract Base Class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Building Custom Datasets: From CSV to Image Folders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Transforms and Compose: Data Augmentation Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

DataLoader: Batching, Shuffling, Workers, and Pin Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Multi-process Data Loading: Performance Tuning and Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Handling Imbalanced Data: Weighted Sampling and Stratification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Streaming and On-the-fly Data Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Custom Collate Functions for Variable-length Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 6: Training Loops, Optimizers, and Loss Functions

Anatomy of a Training Loop: The Canonical Pattern

Validation and Early Stopping Patterns

Optimizers: SGD, Adam, AdamW, RMSprop – When to Use Which

Loss Functions: CrossEntropy, MSELoss, Focal Loss, and Custom Losses

Learning Rate Schedulers: Step, CosineAnnealing, OneCycleLR

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Gradient Clipping and Stability Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Mixed Precision Training with GradScaler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 7: GPU Acceleration and Distributed Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

CUDA Fundamentals: Device Placement and Memory Transfers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

torch.cuda API: Streams, Events, and Synchronization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

DataParallel vs DistributedDataParallel: Architecture Comparison

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Setting Up Multi-GPU Training with DDP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Multi-node Distributed Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Horovod and FSDP for Large Model Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Profiling GPU Performance with torch.profiler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Memory Optimization: Offloading, Gradient Checkpointing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 8: Experiment Tracking, Reproducibility, and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Reproducibility: Seeds, Deterministic Operations, and Version Pinning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Logging Strategies: TensorBoard, WandB, and Custom Loggers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Experiment Tracking: Managing Hyperparameters and Artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Debugging Neural Networks: NaN Detection and Gradient Flow Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Profiling with torch.profiler: Identifying Bottlenecks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Memory Debugging: Finding Leaks and Overhead

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Unit Testing Your Models and Training Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 9: Model Serialization, Checkpointing, and Inference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Saving and Loading: state_dict vs Full Model Serialization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Checkpointing Strategies: Periodic, Best-model, and Resume Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

TorchScript: Tracing vs Scripting for Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Inference Optimization: torch.compile and Inductor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Batch Inference and Throughput Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Model Size Reduction: Quantization (PTQ, QAT)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Pruning: Unstructured and Structured Approaches

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 10: The PyTorch Ecosystem – Vision, Audio, and Text

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

torchvision: Transforms, Datasets, and Model Zoo

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Pre-trained Models: ResNet, EfficientNet, Vision Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

torchaudio: Audio Processing, Feature Extraction, and Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

torchtext: Text Tokenization, Vocabularies, and Data Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Hugging Face Integration: Transformers Library with PyTorch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Cross-domain Transfer Learning Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 11: Advanced Architectures – Transformers and Beyond

Attention Mechanisms: Self-Attention, Multi-Head, and Cross-Attention

Building a Transformer from Scratch in PyTorch

nn.Transformer: The Built-in Implementation

Positional Encodings: Sinusoidal, Learned, and RoPE

Sequence Modeling: RNNs, LSTMs, GRUs vs Transformers

Fine-tuning Large Language Models with LoRA and QLoRA

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Causal Masking, KV-Cache, and Inference Optimization for LLMs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 12: End-to-End Projects – Computer Vision

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Image Classification: From Scratch to Transfer Learning on CIFAR/ImageNet

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Object Detection: Implementing a Simple R-CNN and Using Detectron2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Semantic Segmentation: U-Net Architecture and Training Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Data Augmentation Strategies for Vision Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Evaluation Metrics: mAP, IoU, Confusion Matrices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 13: End-to-End Projects – NLP, Generative Models, and RL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Text Classification and Named Entity Recognition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Language Modeling: Training a GPT-style Model from Scratch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Generative Adversarial Networks: DCGAN and StyleGAN Concepts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Variational Autoencoders: Latent Space Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Reinforcement Learning: Q-Learning, Policy Gradients, and PPO

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Diffusion Models: The Architecture Behind Modern Image Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 14: Deployment and Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Exporting to ONNX: Graph Conversion and Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

TorchServe: Model Serving Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Containerizing PyTorch Models with Docker

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Building REST APIs for Model Inference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Monitoring and Drift Detection in Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

CI/CD for ML Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Chapter 15: Performance Optimization and Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

torch.compile: The Inductor Compiler Explained

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Memory Optimization: Gradient Checkpointing, Activation Recomputation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Operator Fusion and Kernel-Level Optimizations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Benchmarking Methodology: Measuring What Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Common Anti-patterns and How to Avoid Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Production Best Practices Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

The Future of PyTorch: Trends and Roadmap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

Conclusion: From First Tensor to Production System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/pytorchdeepdive.