

Deep Learning



PyTorch



Learning Objectives

By the end of this lesson, you will be able to:

- 👁️ Analyze the basics of PyTorch
- 👁️ Implement a neural network using PyTorch
- 👁️ Create an image classification model using PyTorch



Business Scenario

XYZ Inc. aims to build a real-time fraud detection system using PyTorch, a flexible and user-friendly deep learning framework. They will leverage PyTorch's Tensor class for data handling and the Autograd, Optim, and NN modules for model development and training.

To efficiently train on large datasets, they plan to use data parallelism across multiple GPUs. However, they acknowledge the need for quality data, a well-suited model, and strategies to address the class imbalance inherent in fraud detection tasks.

Through this approach, XYZ Inc. strives to enhance its fraud detection process, ensuring secure and reliable customer service.





Introduction to PyTorch



Discussion

Discussion: PyTorch



- What are the benefits of using PyTorch?
- What are the key features of PyTorch?

PyTorch

It is an open-source deep learning framework that accelerates the path from research prototyping to production deployment.



PyTorch is a flexible and widely used deep learning framework, renowned for its capabilities in areas like computer vision, natural language processing, and reinforcement learning.

Characteristics of PyTorch

Provides a flexible and fast
deep learning platform

Offers GPU acceleration and
automatic differentiation

The PyTorch logo, featuring a red flame icon and the word "PyTorch" in black text, is centered within a light gray circle. Three lines (blue, orange, and gray) extend from the circle to the surrounding text blocks.

PyTorch

Has evolved into an open-source project with
a large community

Features of PyTorch



Simple interface: It provides a user-friendly and intuitive interface for easy model prototyping.



Eager execution: It offers flexible and instant model evaluation with dynamic computational graphs.

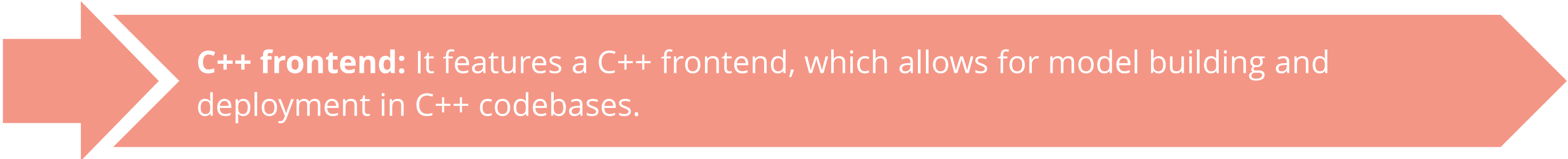


Distributed training: It supports efficient and scalable training across multiple GPUs or machines.

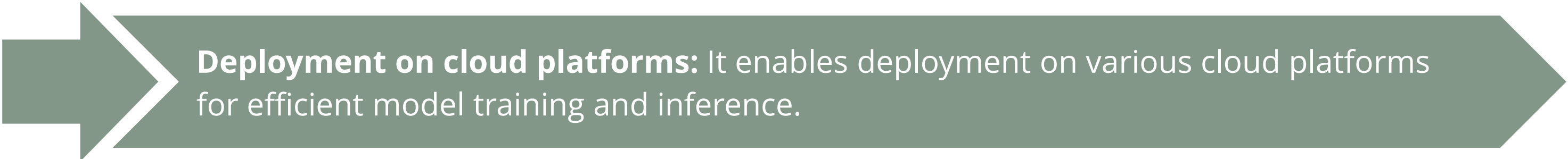
Features of PyTorch



Native ONNX support: It offers support for ONNX format, facilitating interoperability between various deep learning frameworks.

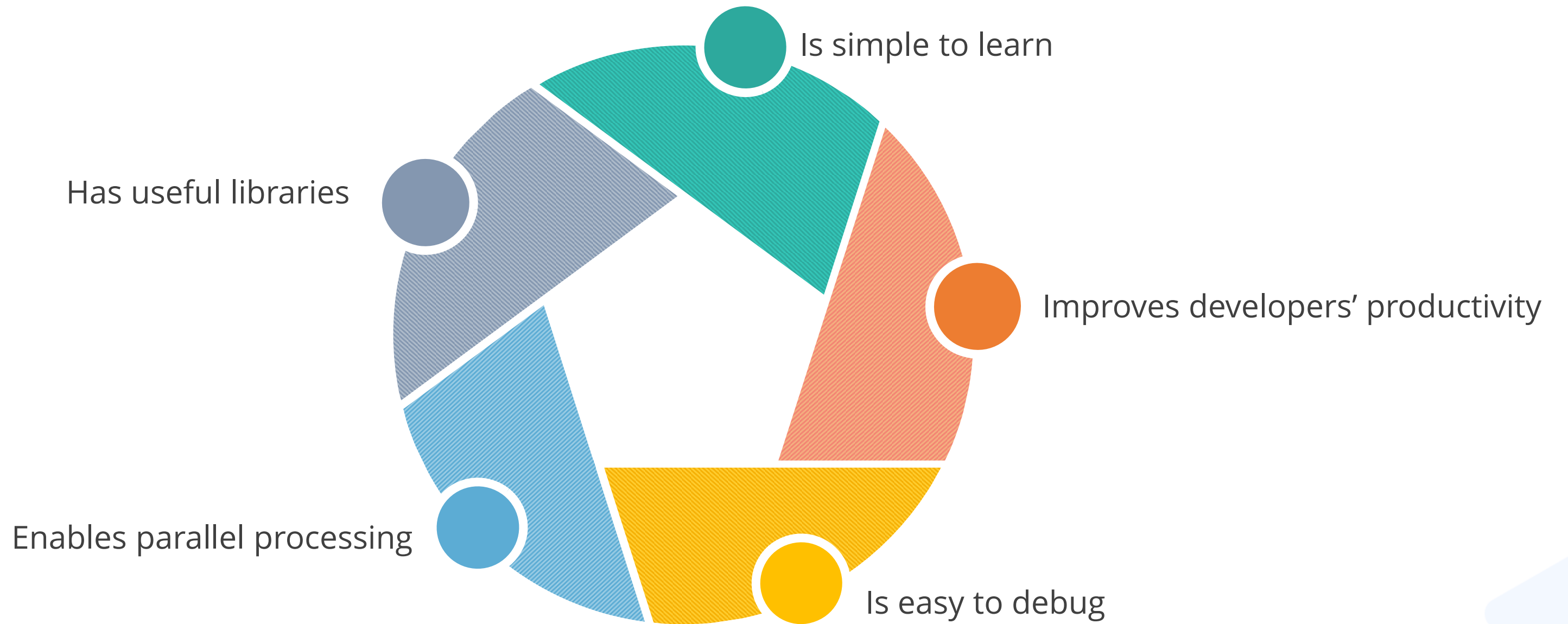


C++ frontend: It features a C++ frontend, which allows for model building and deployment in C++ codebases.



Deployment on cloud platforms: It enables deployment on various cloud platforms for efficient model training and inference.

Advantages of PyTorch



Advantages of PyTorch: Is Simple to Learn

The dynamic computational graph used by PyTorch is similar to traditional programming paradigms.



It benefits from extensive documentation and robust community support.

Advantages of PyTorch: Improves Developers' Productivity

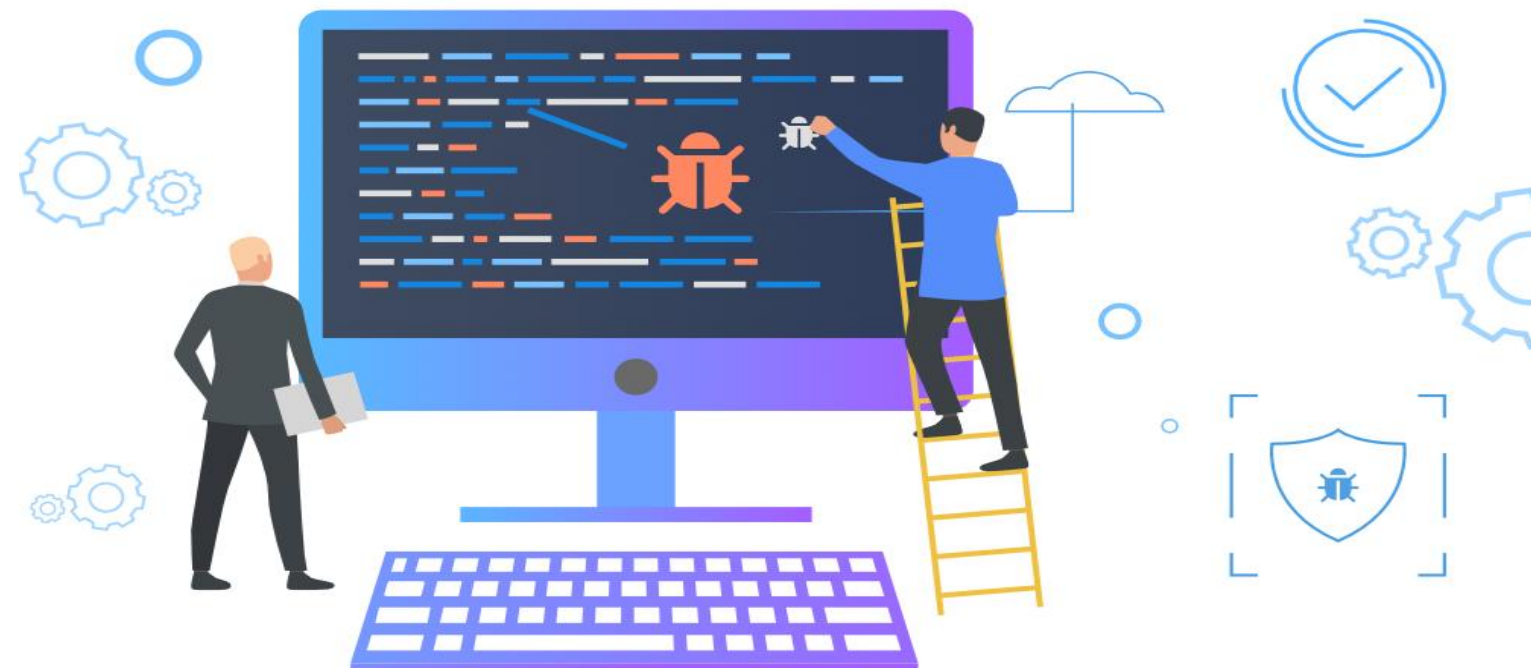
PyTorch has an interface with Python and several powerful APIs.



PyTorch provides features and resources that make automation and streamlining of the deep learning workflow possible.

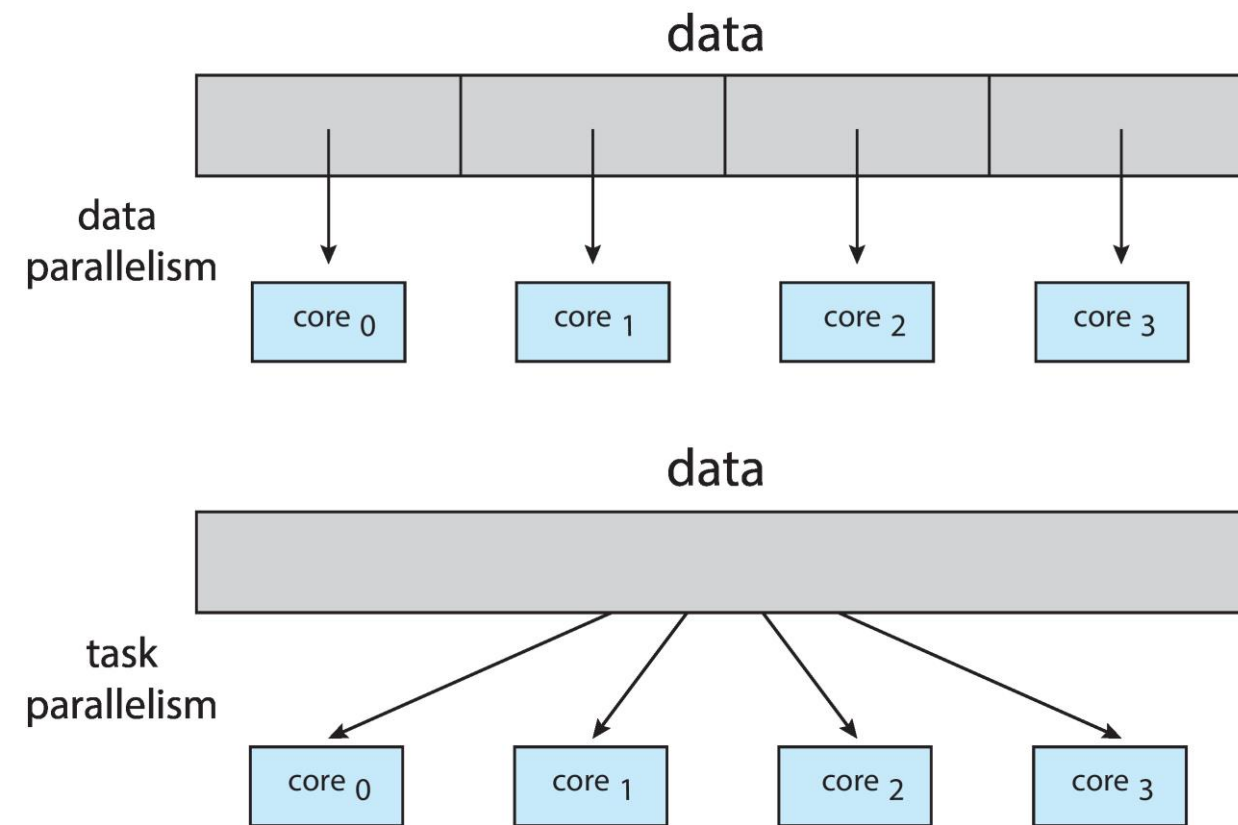
Advantages of PyTorch: Is Easy to Debug

PyTorch can use debugging tools like pdb and ipdb (Ipython-enabled Python Debugger).



Advantages of PyTorch: Enables Parallel Processing

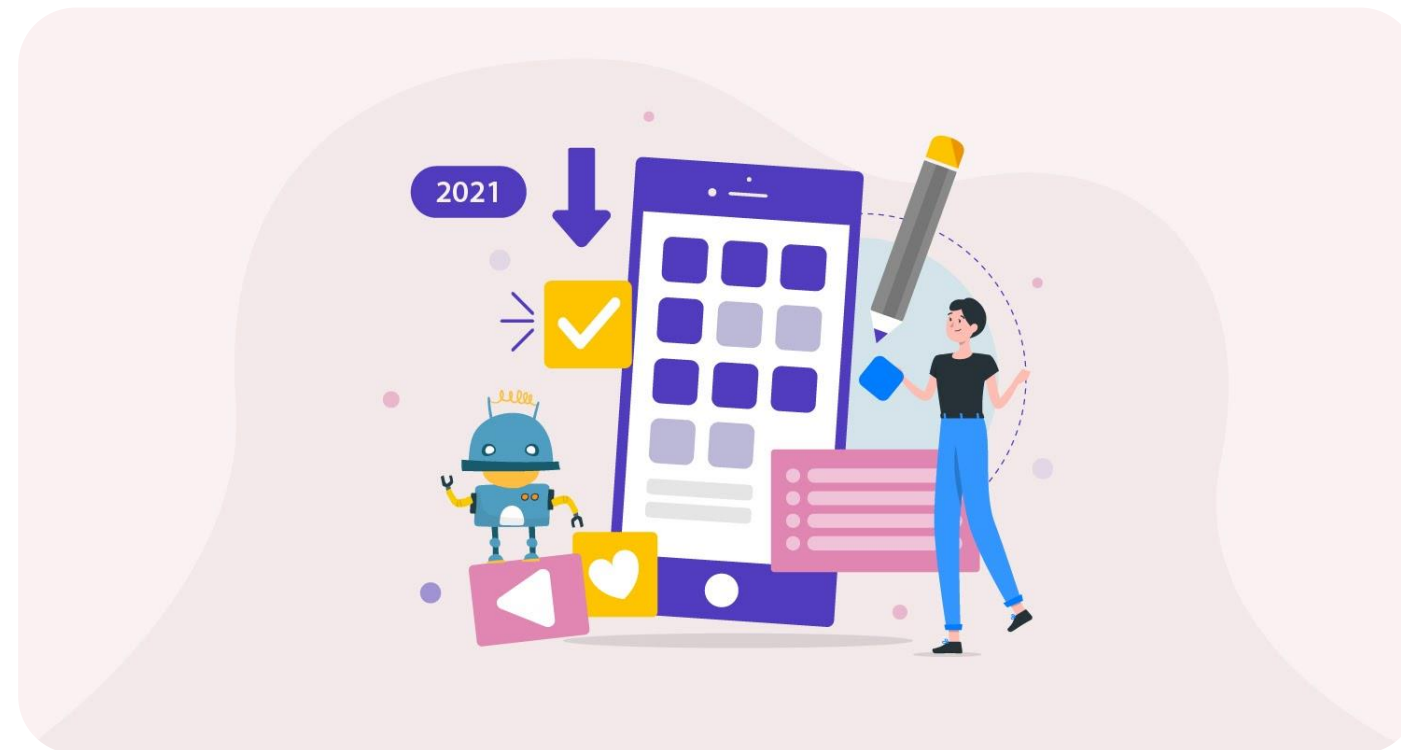
PyTorch can distribute the computational tasks among multiple CPUs.



This is possible using the data parallelism feature, which wraps any module and allows parallel processing.

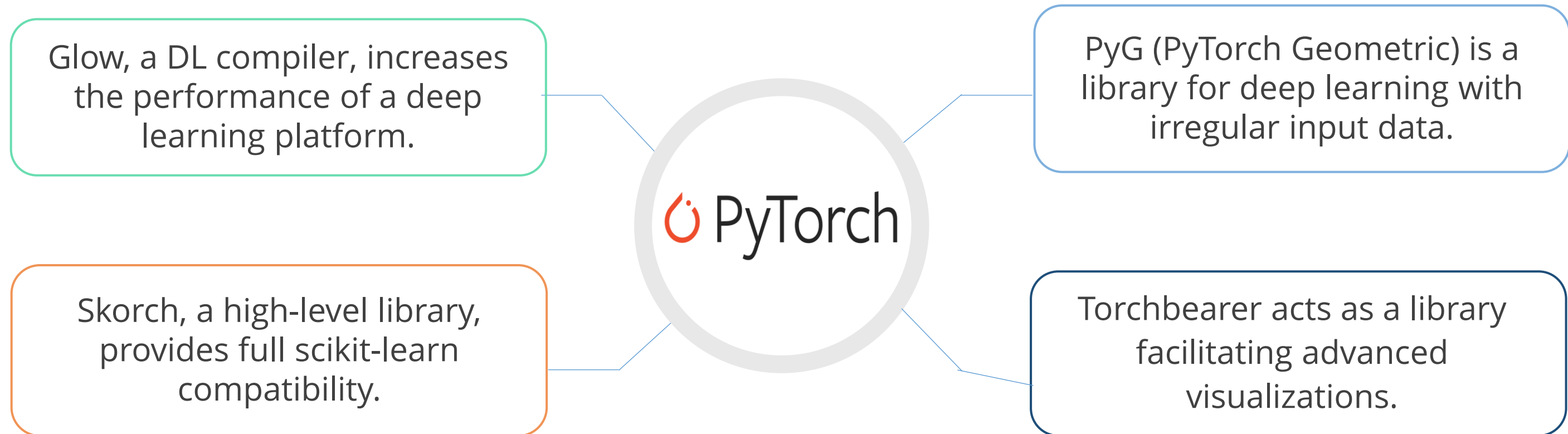
Advantages of PyTorch: Has Useful Libraries

PyTorch has a large community of developers and researchers who have built tools and libraries to improve PyTorch.



PyTorch Ecosystems

The following are the PyTorch ecosystems:



Discussion: PyTorch



- What are the benefits of using PyTorch?

Answer: PyTorch offers ease of use, a dynamic computational graph, and strong community support.

- What are the key features of PyTorch?

Answer: PyTorch excels in efficient tensor operations, offers automated differentiation, and provides comprehensive support for neural networks.



Installation of PyTorch

Installation of PyTorch

Use the following command to install PyTorch:

Syntax:-

```
pip install torch torchvision torchaudio
```

Multiple installation commands for different configurations:

- Install PyTorch without CUDA support using **pip**:

Syntax:-

```
pip install torch torchvision torchaudio
```

Installation of PyTorch

- Install PyTorch with CUDA support using **pip**:

Syntax:-

```
pip install torch torchvision torchaudio --pre -f  
https://download.pytorch.org/whl/nightly/cu111/torch_nightly.h  
tml
```

- Install PyTorch with CUDA support using **conda**:

Syntax:-

```
conda install pytorch torchvision torchaudio  
cudatoolkit=<version>
```

PyTorch Tensors

In PyTorch, the fundamental unit is a class known as Tensor, which is a vector or a matrix.

The following code snippet shows how it is used:

Syntax:-

```
import torch  
torch.Tensor
```

Modules in PyTorch: AutoGrad

AutoGrad provides automatic differentiation for gradient computation during deep learning training.

Syntax:-

```
import torch
torch.autograd
```

The **torch.autograd** package provides classes and functions to implement automatic differentiation.

Modules in PyTorch: Optim

It is responsible for implementing optimization algorithms used for building neural networks.

Syntax:-

```
import torch
torch.optim
```

The **torch.optim** package provides various optimization algorithms.

Modules in PyTorch: NN

It is responsible for creating computational graphs.

Syntax:-

```
import torch  
torch.nn
```

It provides many classes and modules to implement and train neural networks.

The AutoGrad can also be used, but it is too low-level for defining complex neural networks.

Data Parallelism

It is the distribution of tasks on multiple CPUs or GPUs at the same time.

Syntax:-

```
import torch
torch.nn.DataParallel
```

It is implemented using the **DataParallel** module of the torch.



Example: Building a DL Model with the Fashion-MNIST dataset

Building a DL Model with the Fashion-MNIST Dataset

The Fashion-MNIST dataset, featuring 10 classes of fashion articles and 28x28 pixel images, is utilized in deep learning applications.

The steps to be followed are:

01

Import libraries and dataset

Gather the required libraries and then import the dataset

02

Load and explore the data

Initiate the dataset, preprocess it, and understand its structure and characteristics

Building a DL Model with the Fashion-MNIST Dataset

The steps to be followed are:

03

Define the model architecture

Establish the structure of the neural network, including layers, neurons, and activation functions

04

Compile and train the model

Set up the model's learning process and train it on the dataset

06

Evaluate the model

Assess the performance of the trained model on a separate test dataset

Import the Library

Import essential libraries

Example:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms
```

Load and Explore the Dataset

Load the Fashion-MNIST dataset, preprocess it, and explore its structure

Example:

```
# Data preprocessing: normalization
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Loading the dataset
train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True,
transform=transform)
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True,
transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32,
shuffle=False)

# Printing the shape of the datasets
print(f'Training data: {len(train_dataset)} samples')
print(f'Testing data: {len(test_dataset)} samples')
```

Output:

```
Training data: 60000 samples
Testing data: 10000 samples
```


Defining the Model

Define simple CNN with one convolutional layer, one max pooling layer, and two fully connected layers

Example:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)  # 1 input channel, 32
        output channels, 3x3 kernel
        self.pool = nn.MaxPool2d(2, 2)  # 2x2 pooling
        self.fc1 = nn.Linear(13*13*32, 100)  # Flattened dimensions
        after pooling
        self.fc2 = nn.Linear(100, 10)  # 10 classes for FashionMNIST

    def forward(self, x):
        x = self.conv1(x)
        x = nn.ReLU()(x)
        x = self.pool(x)
        x = x.view(-1, 13*13*32)  # Flatten
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.fc2(x)
        return nn.Softmax(dim=1)(x)

model = CNN()
```

Compile and Train the Model

The model is compiled and trained for 10 epochs using 'Adam' optimizer and cross entropy loss function.

Example:

```
# Setting up the loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model for 10 epochs
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward + backward + optimize
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    running_loss += loss.item()

print(f"Epoch {epoch+1}/{num_epochs}, Loss: {running_loss/len(train_loader)}")
```

Output:

```
Epoch 1/10, Loss: 1.8133027095158896
Epoch 2/10, Loss: 1.6204359053929647
Epoch 3/10, Loss: 1.5715269567489625
Epoch 4/10, Loss: 1.5599591334025065
Epoch 5/10, Loss: 1.5515947381337483
Epoch 6/10, Loss: 1.5452620505015056
Epoch 7/10, Loss: 1.5412239967981973
Epoch 8/10, Loss: 1.5354691167195638
Epoch 9/10, Loss: 1.5328466287612914
Epoch 10/10, Loss: 1.5303263650894166
```

Evaluate the Model

The model's performance is evaluated on the test set.

Example:

```
# Evaluating the model
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Model accuracy on test set: {accuracy}%")
```

Output:

Model accuracy on test set: 89.91%



Example: MNIST Digit Classifier

Example: MNIST Digit Classifier

Create a Deep Learning Model with the MNIST dataset to predict the handwritten digits using TensorFlow

The steps to be followed are:

01

Import and Load the Dataset

→ Load the dataset, import the required libraries and normalize pixel values, reshape images

02

Visualize the Data

→ Visualize the seventh image from a batch of the MNIST training dataset

03

Define the Model

→ Define a Fully Connected Neural Network (also known as a Dense Neural Network or Multi-Layer Perceptron)

Example: MNIST Digit Classifier

The steps to be followed are:

- 04** Compile the Model → Specify the optimizer and loss function
- 05** Fit the Model → Train the model with specified epochs and batch sizes on the training data
- 06** Evaluate and Predict the model → Evaluate model performance and use the trained model to make predictions on new or unseen data

Import and Load the Dataset

Import PyTorch and load the MNIST dataset of handwritten digits, 0 to 9, applies transformations (convert to tensor and normalize), and creates data loaders

Example:

```
import torch
import torchvision.transforms as transforms
from torchvision.datasets import MNIST
from torch.utils.data import DataLoader

# Define transformation: Convert image to tensor and normalize
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])

# Download and load the dataset
train_dataset = MNIST(root='./data', train=True, transform=transform,
                       download=True)
test_dataset = MNIST(root='./data', train=False, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Visualize the Data

Let's visualize the seventh image from a batch of the MNIST training dataset and prints its corresponding label

Example:

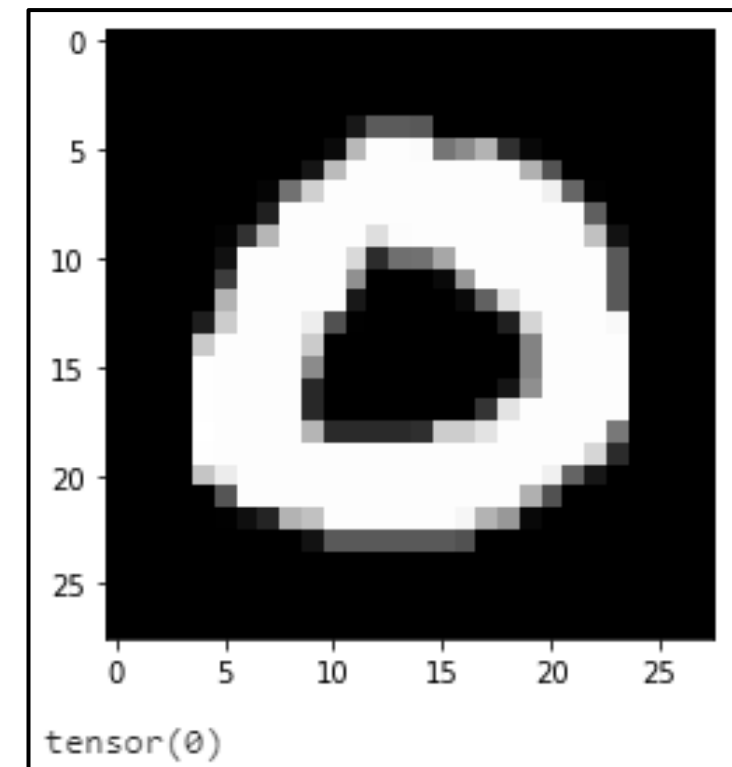
```
import matplotlib.pyplot as plt

dataiter = iter(train_loader)
images, labels = next(dataiter)

plt.imshow(images[6].numpy().squeeze(),
            cmap='gray')
plt.show()

print(labels[6])
```

Output:



Define the Model

Define a neural network model with three fully connected layers

Example:

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.softmax(self.fc3(x), dim=1)
        return x

model = Net()
```

Compile the Model

Define the loss function (cross-entropy loss) and optimizer (Adam)

Example:

```
import torch.optim as optim

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

Fit the Model

Train the model for 10 epochs using the training data.

Example:

```
epochs = 10
for epoch in range(epochs):
    running_loss = 0.0
    for i, (images, labels) in enumerate(train_loader, 1): # Added
        enumeration to get batch number
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    # Print average loss for the epoch
    print(f"Epoch [{epoch+1}/{epochs}], Loss:
    {running_loss/i:.4f}")
```

Output:

Epoch [1/10],	Loss: 1.6146
Epoch [2/10],	Loss: 1.5399
Epoch [3/10],	Loss: 1.5274
Epoch [4/10],	Loss: 1.5198
Epoch [5/10],	Loss: 1.5147
Epoch [6/10],	Loss: 1.5099
Epoch [7/10],	Loss: 1.5082
Epoch [8/10],	Loss: 1.5073
Epoch [9/10],	Loss: 1.5039
Epoch [10/10],	Loss: 1.5025

Evaluate the Model

Evaluates the model's performance on the test set and prints the accuracy

Example:

```
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Test Accuracy: {100 * correct / total}%')
```

Output:

Test Accuracy: 95.55%

Predict the Model

The predict() function predicts the digit for the input.

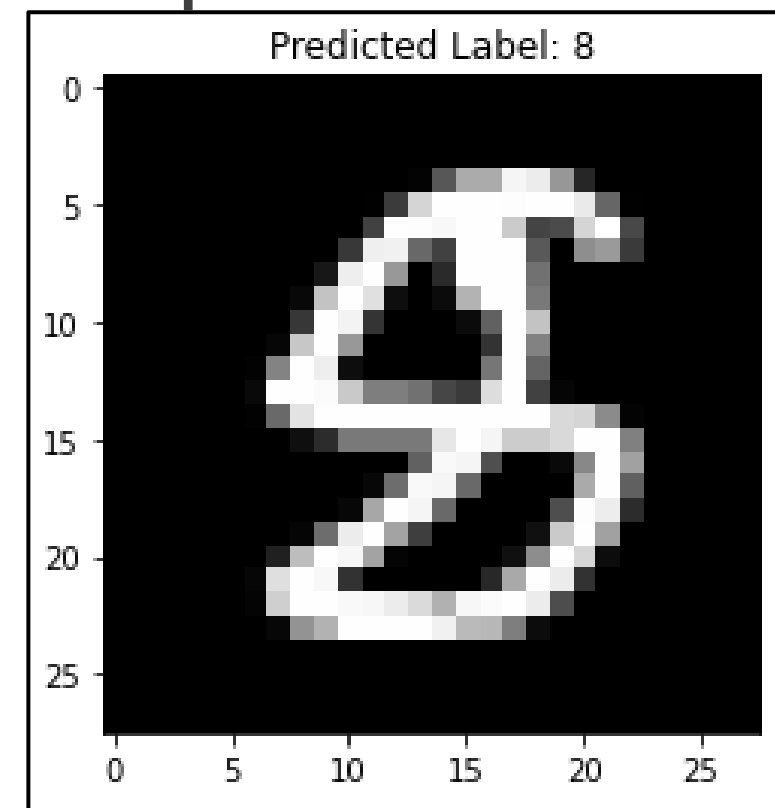
Example:

```
dataiter = iter(train_loader)
images, labels = next(dataiter)

outputs = model(images[1:2])
_, predicted = torch.max(outputs.data, 1)

plt.imshow(images[1].numpy().squeeze(), cmap='gray')
plt.title(f"Predicted Label: {predicted.item()}")
plt.show()
```

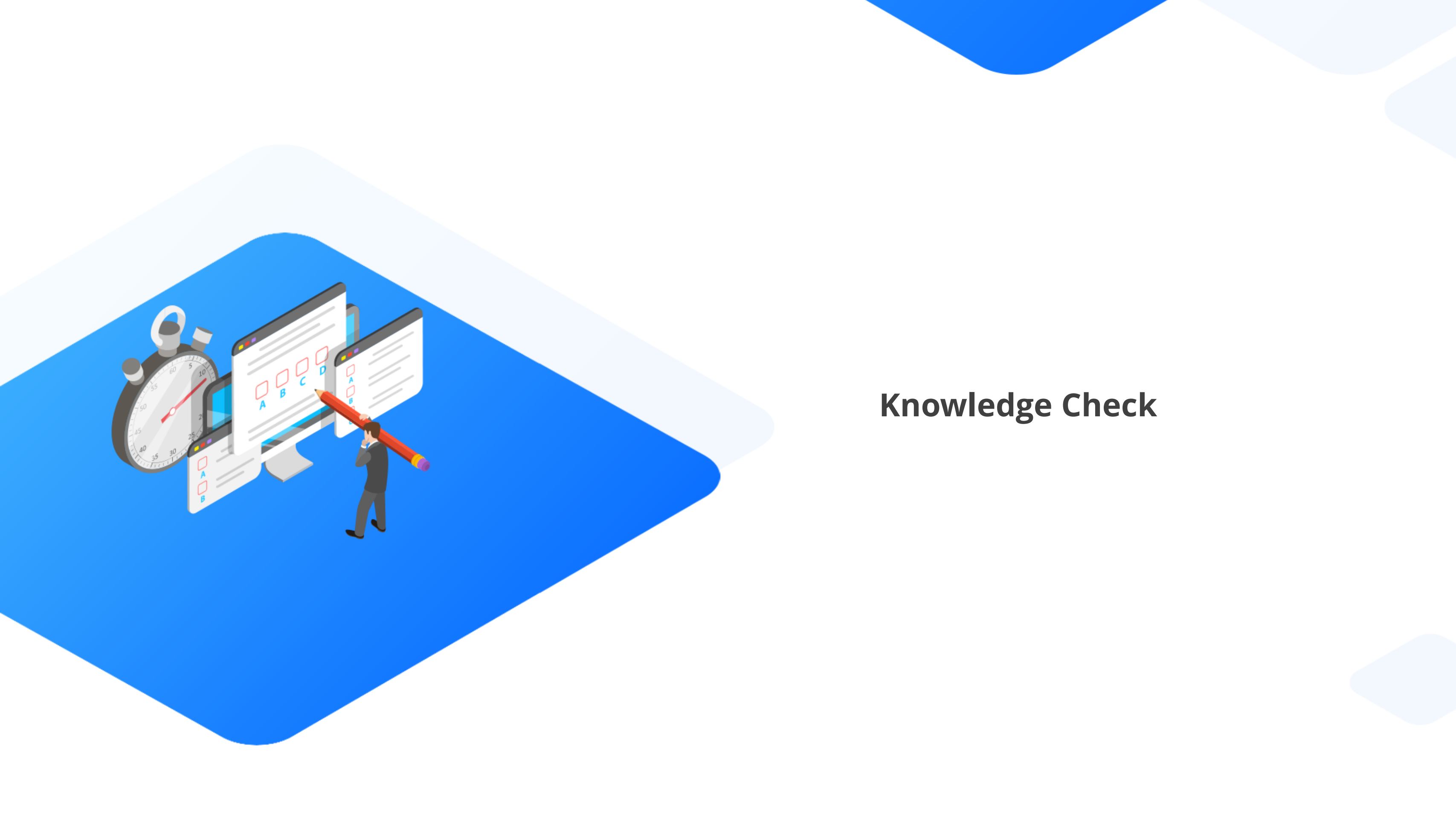
Output:



Key Takeaways

- PyTorch is an open-source machine learning framework that accelerates the path from research prototyping to production deployment.
- In PyTorch, the fundamental unit is a class known as Tensor, which is a vector or a matrix.
- Some of the modules in PyTorch are AutoGrad, Optim, and NN.
- Data parallelism is the distribution of tasks on multiple CPUs or GPUs at the same time.





Knowledge Check

Knowledge Check

1

What is a Tensor in PyTorch?

- A. A type of variable in PyTorch
- B. A vector or a matrix in PyTorch
- C. A module in PyTorch
- D. A type of activation function in PyTorch



Knowledge Check

1

What is a Tensor in PyTorch?

- A. A type of variable in PyTorch
- B. A vector or a matrix in PyTorch
- C. A module in PyTorch
- D. A type of activation function in PyTorch

The correct answer is **B**

The basic entity in PyTorch is a class called Tensor, which is a vector or a matrix.



Knowledge Check

2

What is data parallelism in PyTorch?

- A. A method of distributing tasks on multiple CPUs or GPUs at the same time
- B. A method of performing automatic differentiation in PyTorch
- C. A method of creating computational graphs
- D. A method of implementing optimization algorithms in PyTorch



Knowledge Check

2

What is data parallelism in PyTorch?

- A. A method of distributing tasks on multiple CPUs or GPUs at the same time
- B. A method of performing automatic differentiation in PyTorch
- C. A method of creating computational graphs
- D. A method of implementing optimization algorithms in PyTorch

The correct answer is **A**

It is a method of distributing tasks on multiple CPUs or GPUs at the same time.



Knowledge Check

3

What is the purpose of the Optim module in PyTorch?

- A. To perform automatic differentiation
- B. To create computational graphs
- C. To distribute tasks on multiple CPUs or GPUs
- D. To implement optimization algorithms for building neural networks



Knowledge Check

3

What is the purpose of the Optim module in PyTorch?

- A. To perform automatic differentiation
- B. To create computational graphs
- C. To distribute tasks on multiple CPUs or GPUs
- D. To implement optimization algorithms for building neural networks

The correct answer is **D**

The Optim module in PyTorch is responsible for implementing optimization algorithms used for building neural networks





Thank You!