# *PyTorch Lightning (PL)*

Rowel Atienza, PhD

University of the Philippines

github.com/roatienza

2022

*Decouple research code from engineering.*
– https://www.pytorchlightning.ai

# Supervised Learning using PL

Experience

Task

Performance

PyTorch Lightning removes boiler plate code so we can focus on ETP

# PyTorch vs PyTorch Lightning

## PyTorch

Define train/val/test functions and `for` loops

Call optimizer/scheduler routines

Define dataset/dataloader class

Move model and data to/from device

Compute, accumulate & display performance metrics & losses

## PyTorch Lightning

None

# Installation

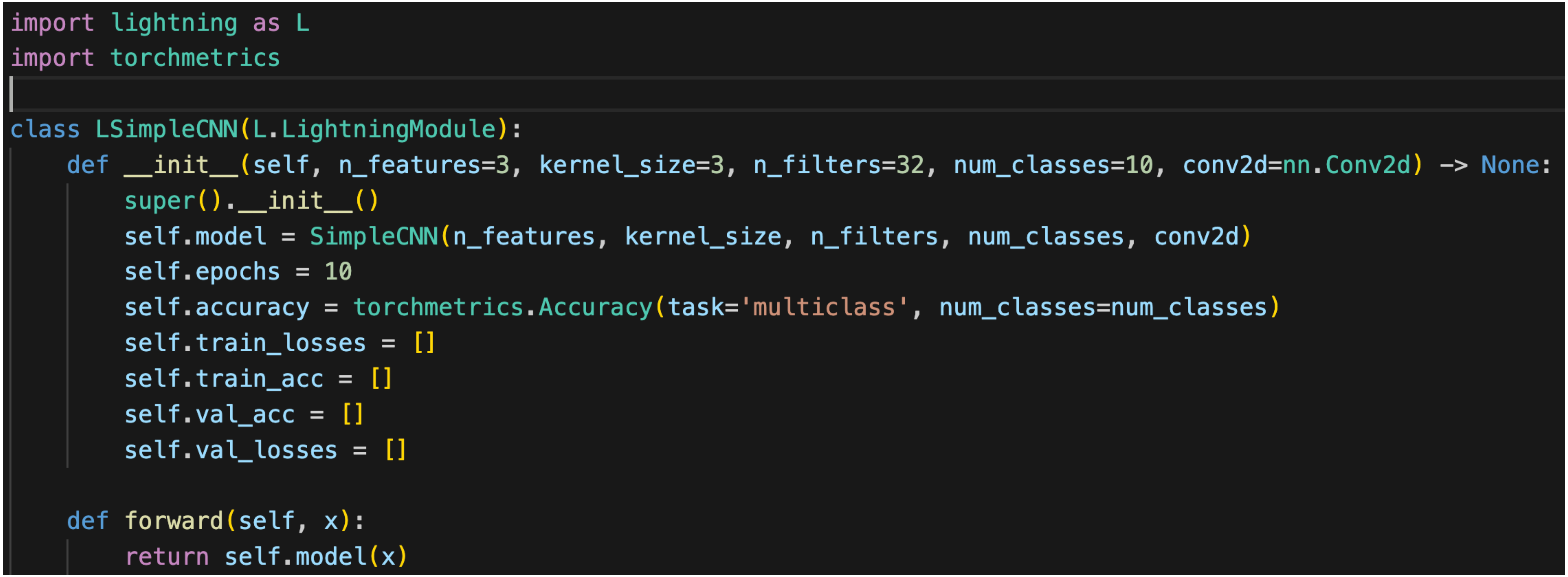
```
pip install lightning
pip install torchmetrics

import lightning as L
```

# PyTorch → PL

Model, Data and Performance Measure Inside a `LightningModule (LM)`

# An LM has a Model

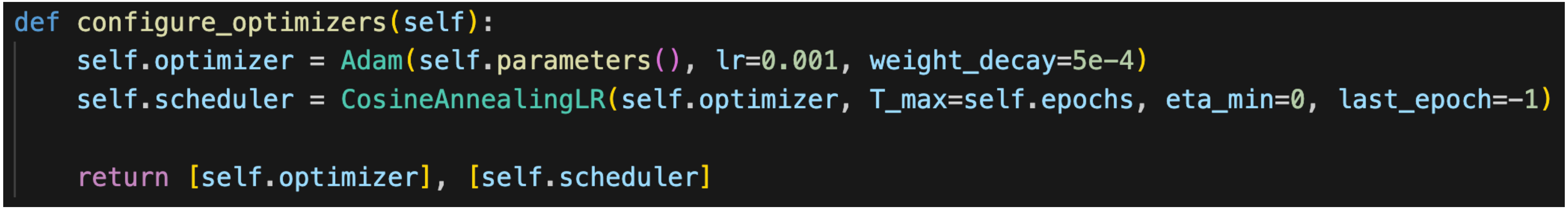


```python
import lightning as L
import torchmetrics

class LSimpleCNN(L.LightningModule):
    def __init__(self, n_features=3, kernel_size=3, n_filters=32, num_classes=10, conv2d=nn.Conv2d) -> None:
        super().__init__()
        self.model = SimpleCNN(n_features, kernel_size, n_filters, num_classes, conv2d)
        self.epochs = 10
        self.accuracy = torchmetrics.Accuracy(task='multiclass', num_classes=num_classes)
        self.train_losses = []
        self.train_acc = []
        self.val_acc = []
        self.val_losses = []

    def forward(self, x):
        return self.model(x)
```

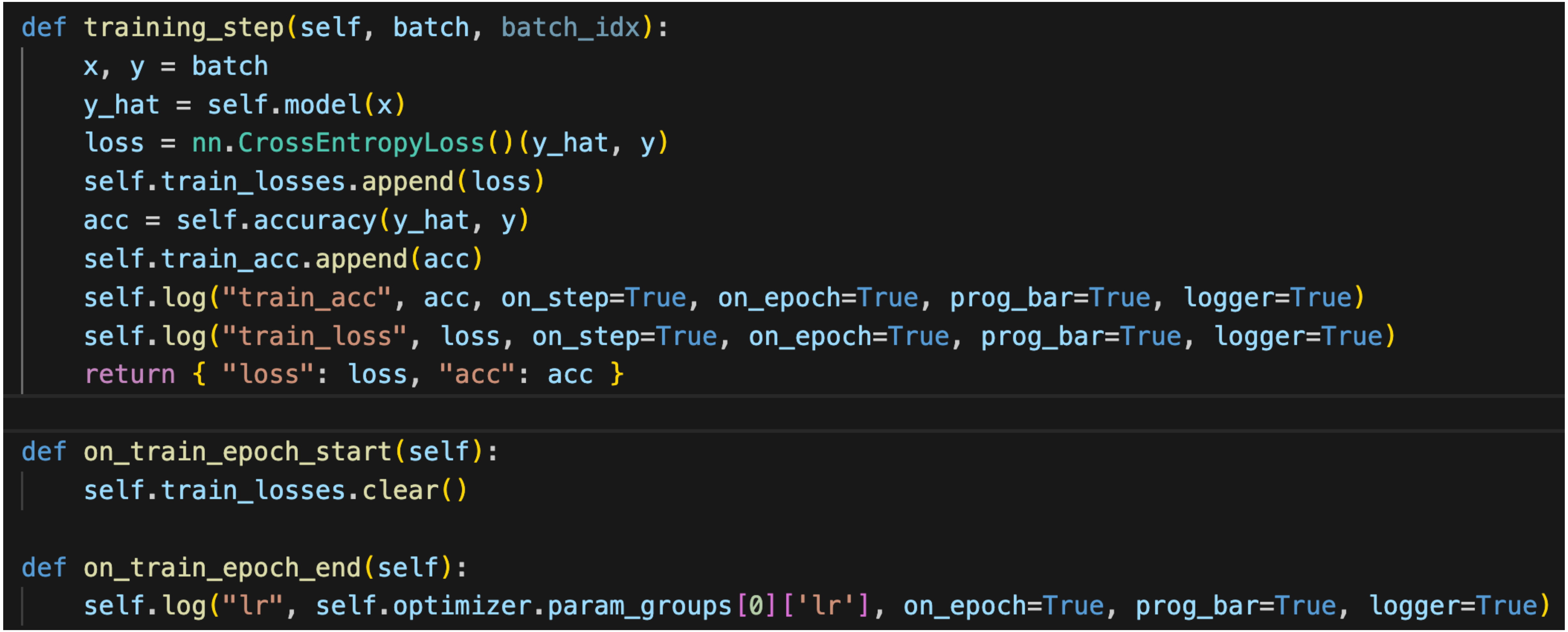
# Optimizer



```
def configure_optimizers(self):
    self.optimizer = Adam(self.parameters(), lr=0.001, weight_decay=5e-4)
    self.scheduler = CosineAnnealingLR(self.optimizer, T_max=self.epochs, eta_min=0, last_epoch=-1)

    return [self.optimizer], [self.scheduler]
```

# Train

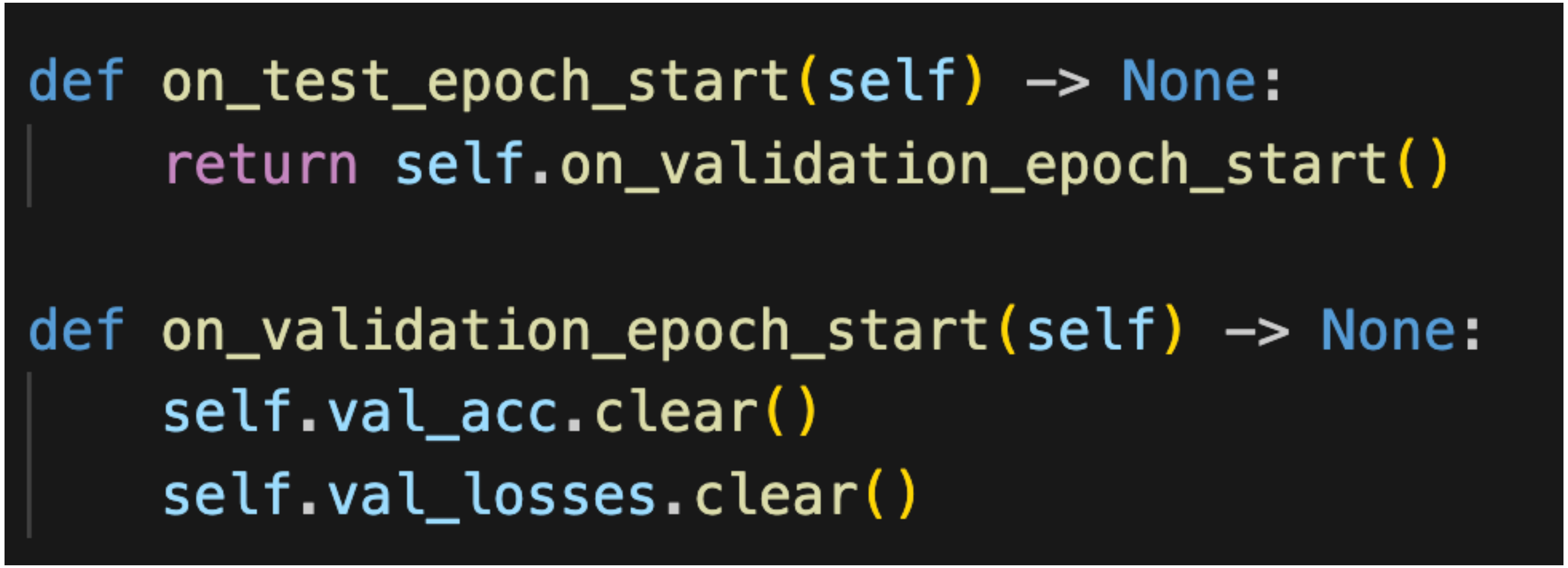


```python
def training_step(self, batch, batch_idx):
    x, y = batch
    y_hat = self.model(x)
    loss = nn.CrossEntropyLoss()(y_hat, y)
    self.train_losses.append(loss)
    acc = self.accuracy(y_hat, y)
    self.train_acc.append(acc)
    self.log("train_acc", acc, on_step=True, on_epoch=True, prog_bar=True, logger=True)
    self.log("train_loss", loss, on_step=True, on_epoch=True, prog_bar=True, logger=True)
    return { "loss": loss, "acc": acc }


def on_train_epoch_start(self):
    self.train_losses.clear()

def on_train_epoch_end(self):
    self.log("lr", self.optimizer.param_groups[0]['lr'], on_epoch=True, prog_bar=True, logger=True)
```

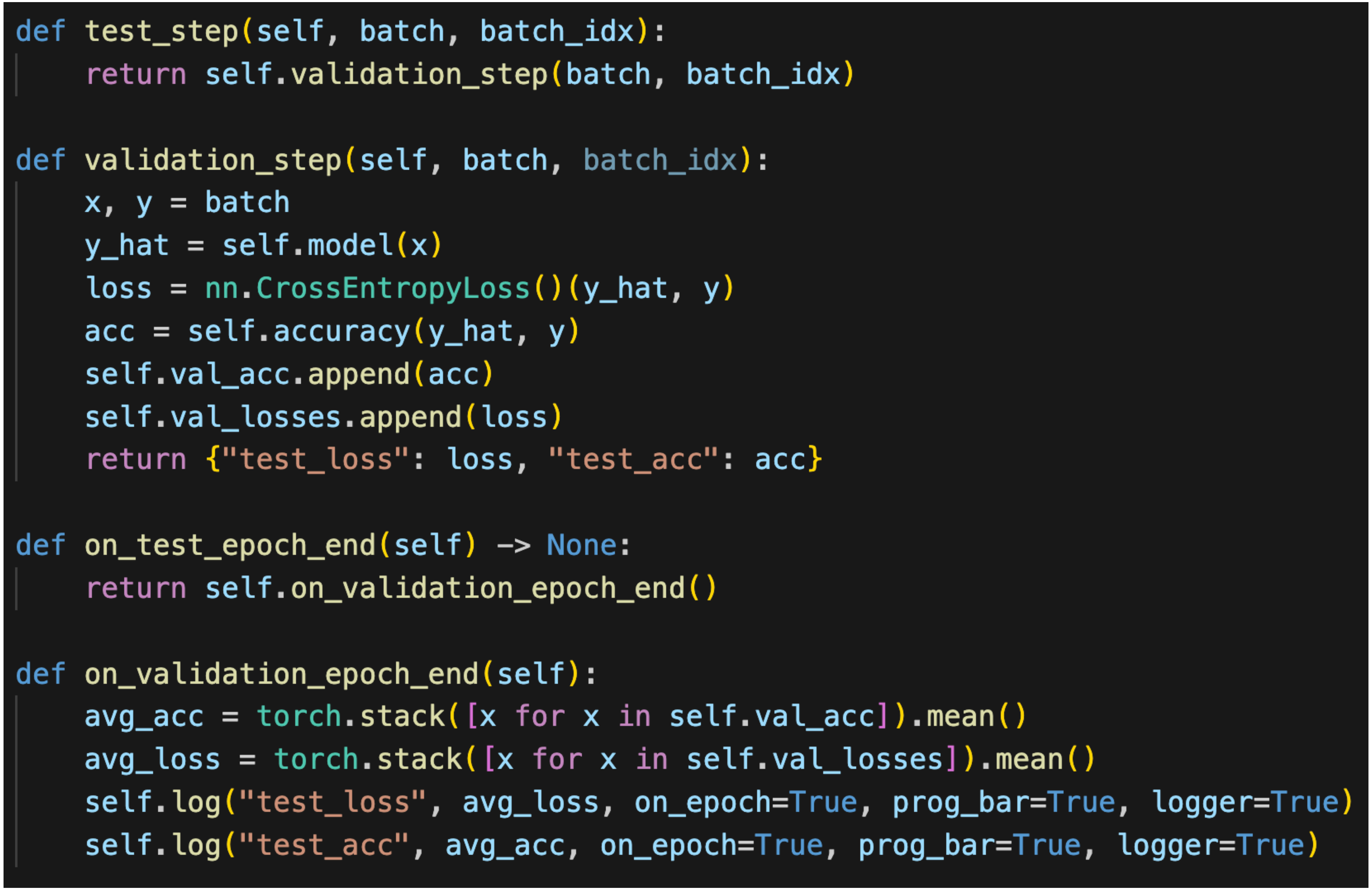
# Test



```python
def on_test_epoch_start(self) -> None:
    return self.on_validation_epoch_start()

def on_validation_epoch_start(self) -> None:
    self.val_acc.clear()
    self.val_losses.clear()
```

# Test



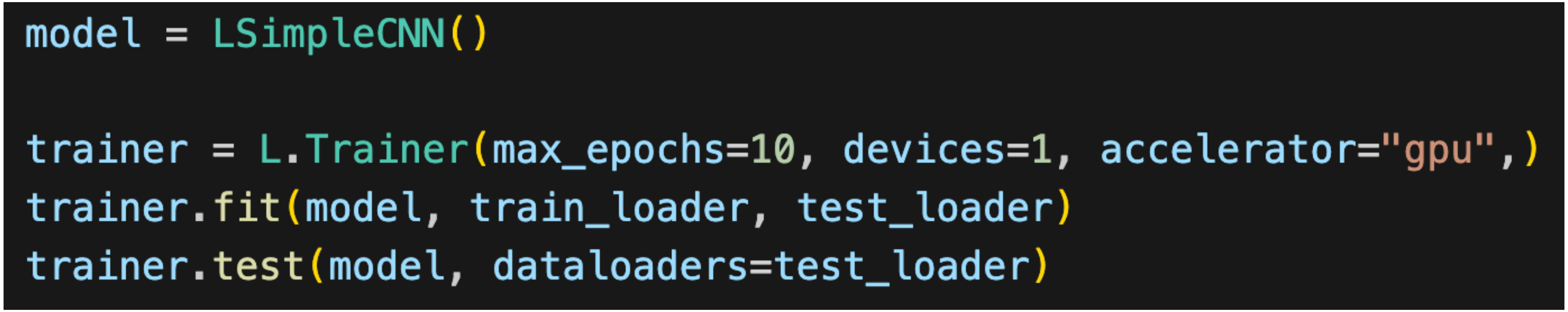
```python
def test_step(self, batch, batch_idx):
    return self.validation_step(batch, batch_idx)

def validation_step(self, batch, batch_idx):
    x, y = batch
    y_hat = self.model(x)
    loss = nn.CrossEntropyLoss()(y_hat, y)
    acc = self.accuracy(y_hat, y)
    self.val_acc.append(acc)
    self.val_losses.append(loss)
    return {"test_loss": loss, "test_acc": acc}

def on_test_epoch_end(self) -> None:
    return self.on_validation_epoch_end()

def on_validation_epoch_end(self):
    avg_acc = torch.stack([x for x in self.val_acc]).mean()
    avg_loss = torch.stack([x for x in self.val_losses]).mean()
    self.log("test_loss", avg_loss, on_epoch=True, prog_bar=True, logger=True)
    self.log("test_acc", avg_acc, on_epoch=True, prog_bar=True, logger=True)
```

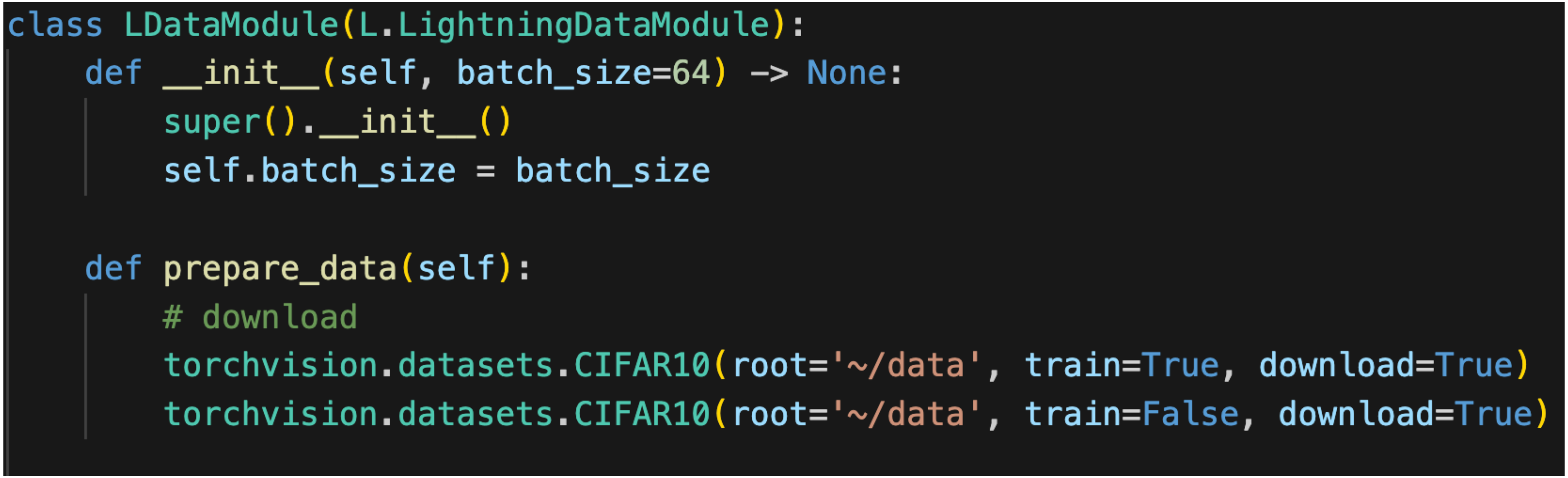
# Instantiate a Model and a Trainer



```
model = LSimpleCNN()

trainer = L.Trainer(max_epochs=10, devices=1, accelerator="gpu",)
trainer.fit(model, train_loader, test_loader)
trainer.test(model, dataloaders=test_loader)
```

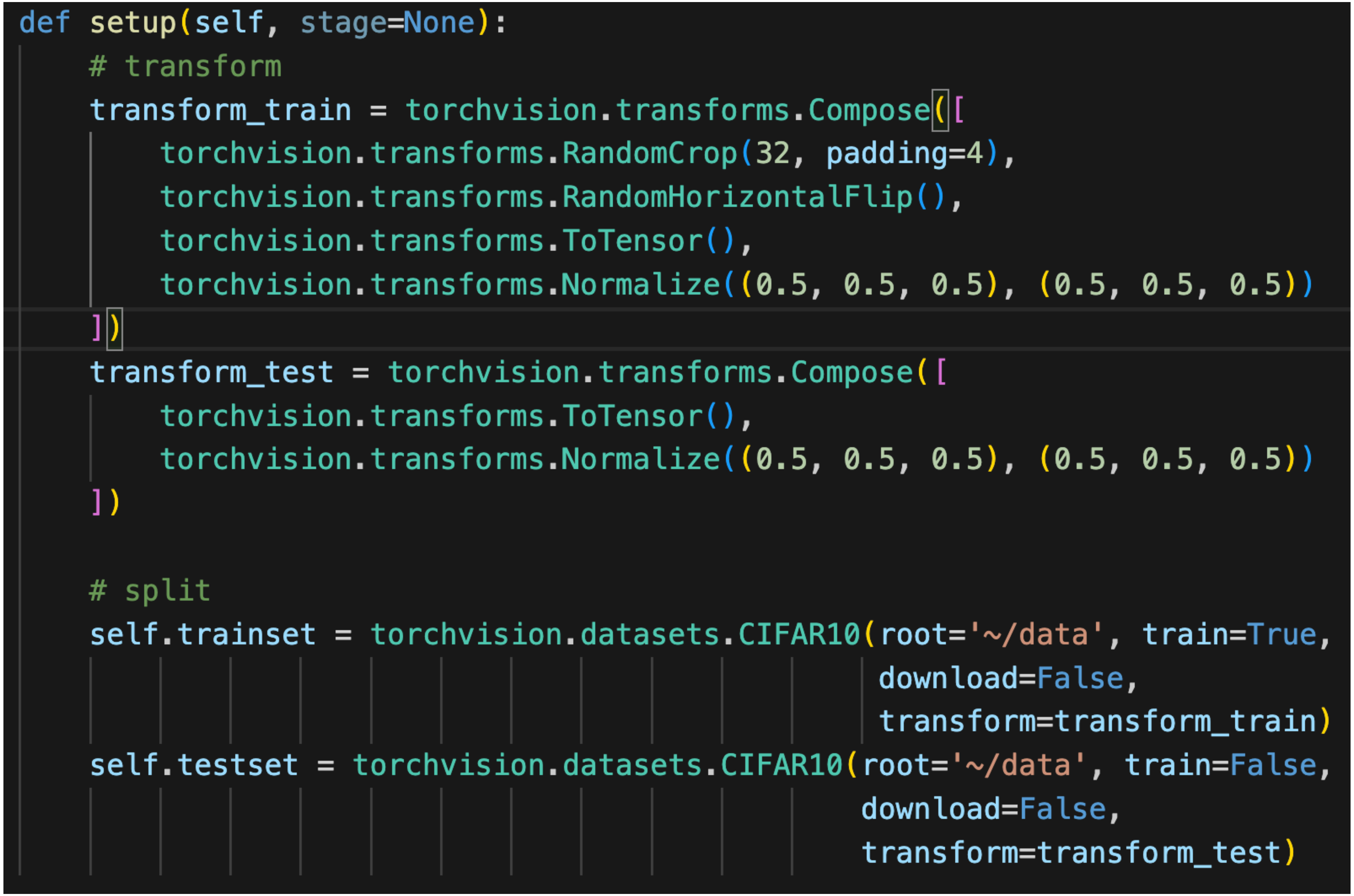
# LightningDataModule Advanced data handling



```
class LDataModule(L.LightningDataModule):
    def __init__(self, batch_size=64) -> None:
        super().__init__()
        self.batch_size = batch_size

    def prepare_data(self):
        # download
        torchvision.datasets.CIFAR10(root='~/data', train=True, download=True)
        torchvision.datasets.CIFAR10(root='~/data', train=False, download=True)
```

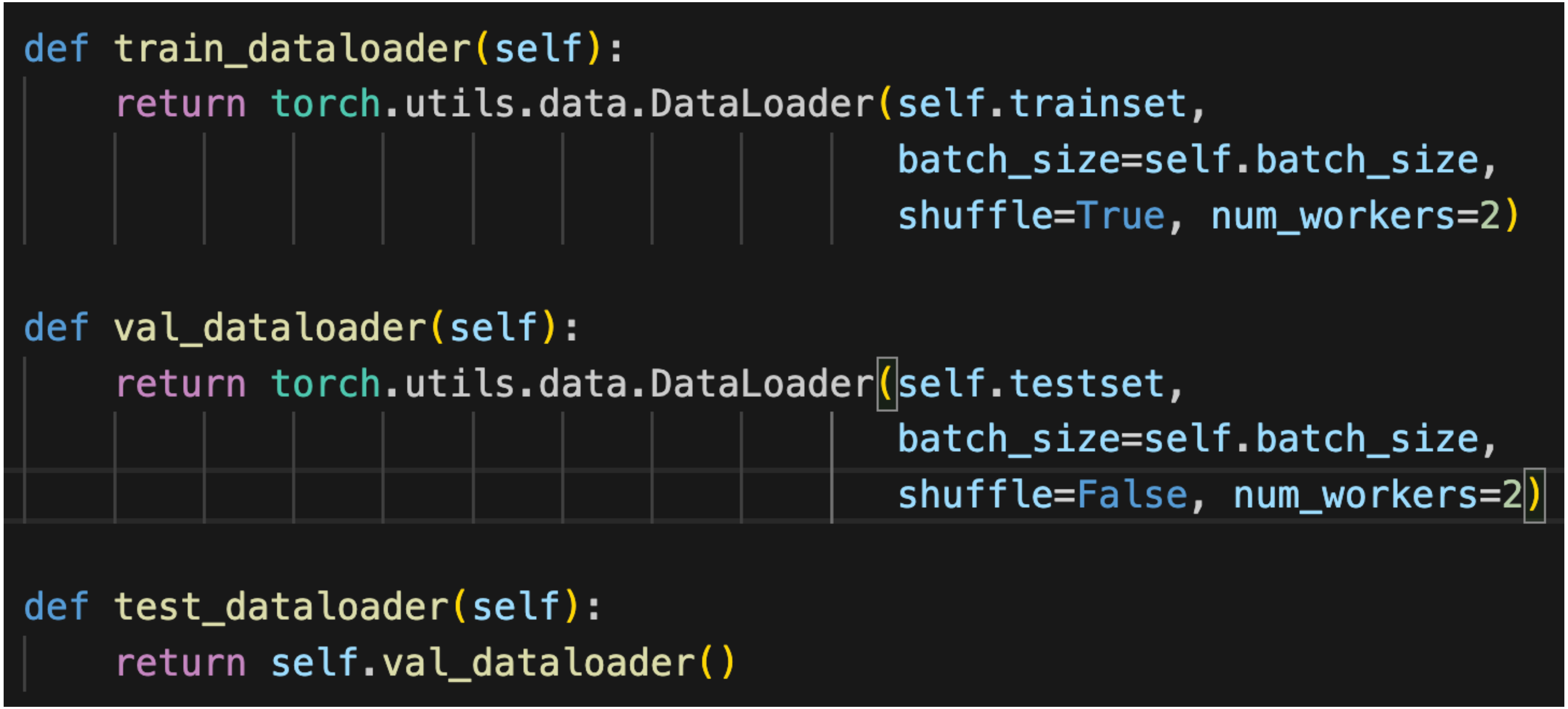
# Setup



```
def setup(self, stage=None):
    # transform
    transform_train = torchvision.transforms.Compose([
        torchvision.transforms.RandomCrop(32, padding=4),
        torchvision.transforms.RandomHorizontalFlip(),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    transform_test = torchvision.transforms.Compose([
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    # split
    self.trainset = torchvision.datasets.CIFAR10(root='~/data', train=True,
                                                 download=False,
                                                 transform=transform_train)
    self.testset = torchvision.datasets.CIFAR10(root='~/data', train=False,
                                                download=False,
                                                transform=transform_test)
```

# Serve

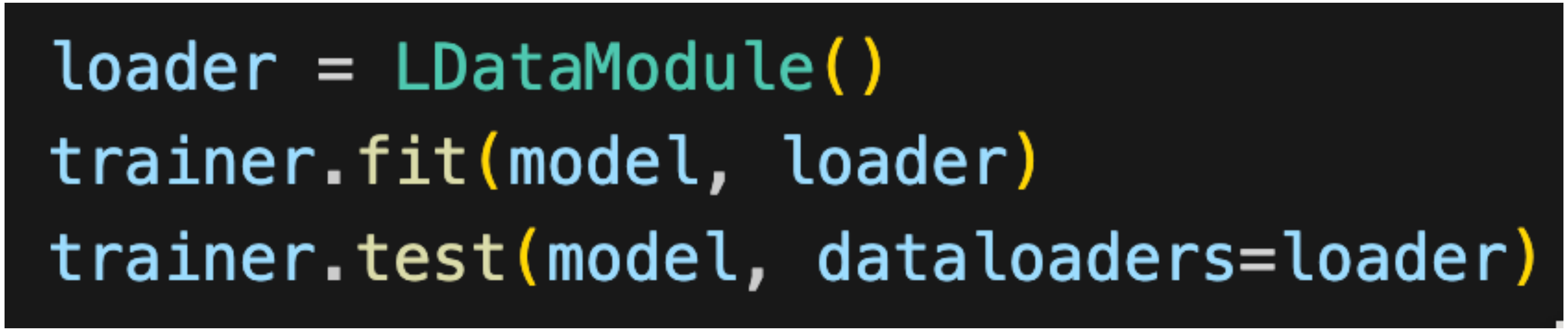


```python
def train_dataloader(self):
    return torch.utils.data.DataLoader(self.trainset,
                                       batch_size=self.batch_size,
                                       shuffle=True, num_workers=2)

def val_dataloader(self):
    return torch.utils.data.DataLoader(self.testset,
                                       batch_size=self.batch_size,
                                       shuffle=False, num_workers=2)

def test_dataloader(self):
    return self.val_dataloader()
```

# Use PL Data Module



```
loader = LDataModule()
trainer.fit(model, loader)
trainer.test(model, dataloaders=loader)
```

# End