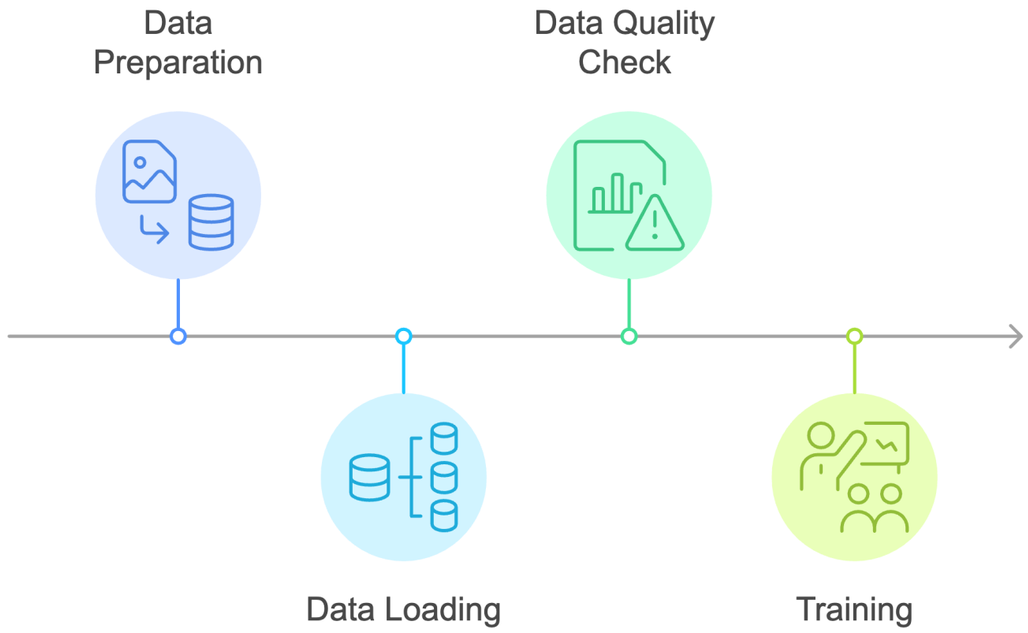
Managing data with LightningDataModule

SCALABLE AI MODELS WITH PYTORCH LIGHTNING

Sergiy Tkachuk
Director, GenAI Productivity

Data preparation for model training

- Poorly prepared data results in training issues
  - Slow training speeds
  - Frequent interruptions
  - Convergence failure



Why use LightningDataModule?

- □ Centralizes dataset handling
- □ Standardizes data preparation workflows
- □ Simplifies training and evaluation phases

Managing data with LightningDataModule

Key methods:

- `prepare_data` : Download and set up data
- `setup` : Split data into train, validation, and test sets

```
class ImageDataModule(pl.LightningDataModule):
    def __init__(self, data_dir="./data", batch_size=32):
        super().__init__()
        ...
    def prepare_data(self):
        datasets.MNIST(self.data_dir, train=True, download=True)
    def setup(self, stage=None):
        dataset = datasets.MNIST(self.data_dir, train=True, transform=self.transform)
        self.train_data, self.val_data = random_split(dataset, [55000, 5000])
        self.test_data = datasets.MNIST(self.data_dir, train=False, transform=self.transform)
```

Creating the train DataLoader

- Supplies batches of training data
- Helps optimize GPU utilization
- Enables efficient iteration over large datasets

```
def train_dataloader(self):
    return DataLoader(self.train_data, batch_size=self.batch_size, shuffle=True)
```

Creating the validation DataLoader

- Supplies data for model validation
- Helps monitor generalization performance
- Ensures consistency across evaluation runs through shuffling

```
def val_dataloader(self):
    return DataLoader(self.val_data, batch_size=self.batch_size)
```

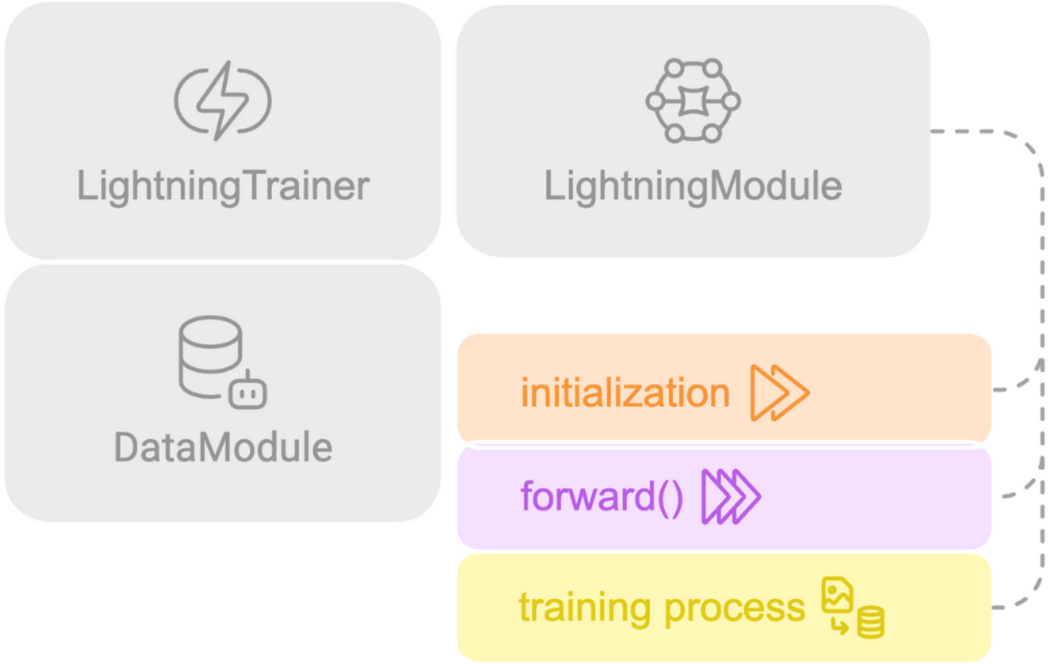
Creating the test DataLoader

- Supplies data for final model evaluation after training is completed
- Simulates real-world performance assessment
- Ensures unbiased performance measurement

```
def test_dataloader(self):
    return DataLoader(self.test_data, batch_size=self.batch_size)
```

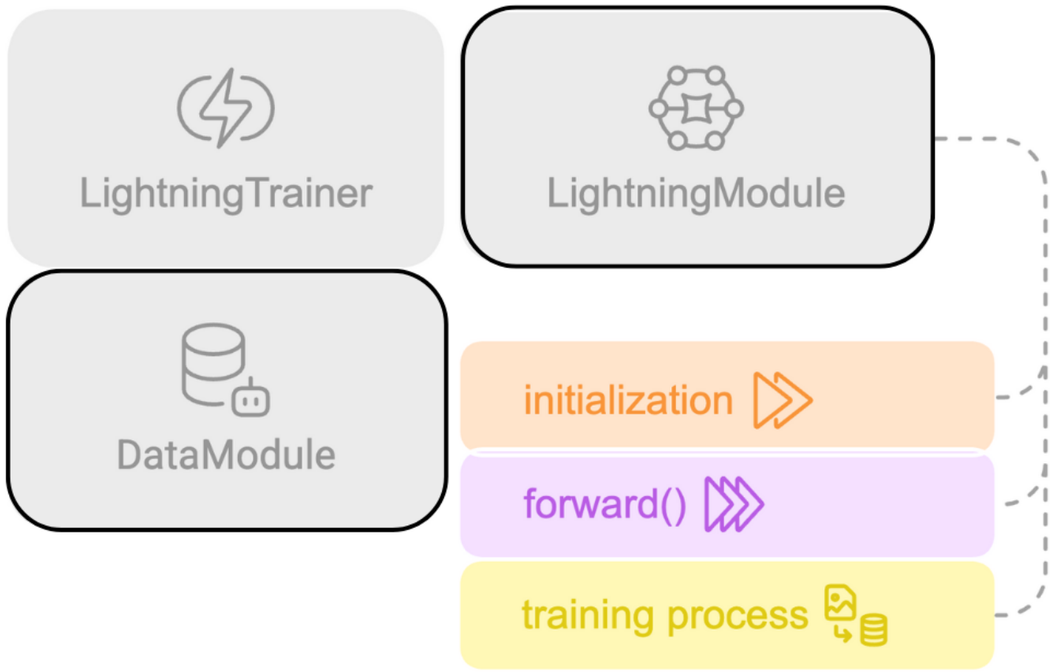
Connecting DataModule to LightningModule

- Modular design separates data and model logic



Connecting DataModule to LightningModule

- Modular design separates data and model logic
- `LightningDataModule` pairs with `LightningModule`
- Standardized workflow enhances reproducibility



Let's practice!

SCALABLE AI MODELS WITH PYTORCH LIGHTNING

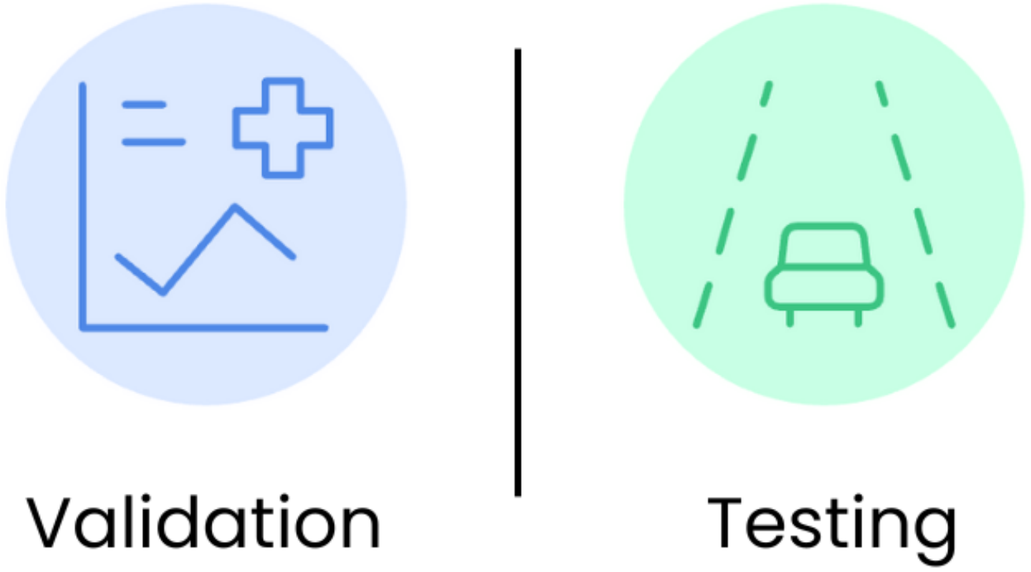
Incorporating validation and testing

SCALABLE AI MODELS WITH PYTORCH LIGHTNING

Sergiy Tkachuk
Director, GenAI Productivity

Why incorporate validation and testing?

- Validation
  - Identify model performance issues early
  - Prevent overfitting and underfitting
- Testing
  - Performance on unseen data



Implementing validation

- Evaluate model performance at each epoch
- Aggregate metrics for a more stable view

```
def validation_step(self, batch, batch_idx):
  x, y = batch
  preds = self(x)
  loss = F.cross_entropy(preds, y)
  self.log('val_loss', loss)

def validation_epoch_end(self, outputs):
  avg_loss = torch.stack([x['loss'] for x in outputs]).mean()
  self.log('avg_val_loss', avg_loss)
```

Implementing testing

- Assess final model performance on unseen data
- Benchmark real-world effectiveness
- Provide metrics for model deployment

```
def test_step(self, batch, batch_idx):
  x, y = batch
  y_hat = self(x)
  loss = F.cross_entropy(y_hat, y)
  self.log('test_loss', loss)

def test_epoch_end(self, outputs):
  avg_loss = torch.stack([x['loss'] for x in outputs]).mean()
  self.log('avg_test_loss', avg_loss)
```

Evaluation with Torchmetrics

- Monitor metrics such as accuracy
- Easily integrate into Lightning workflow
- Initialize accuracy
- Calculate accuracy at each validation step

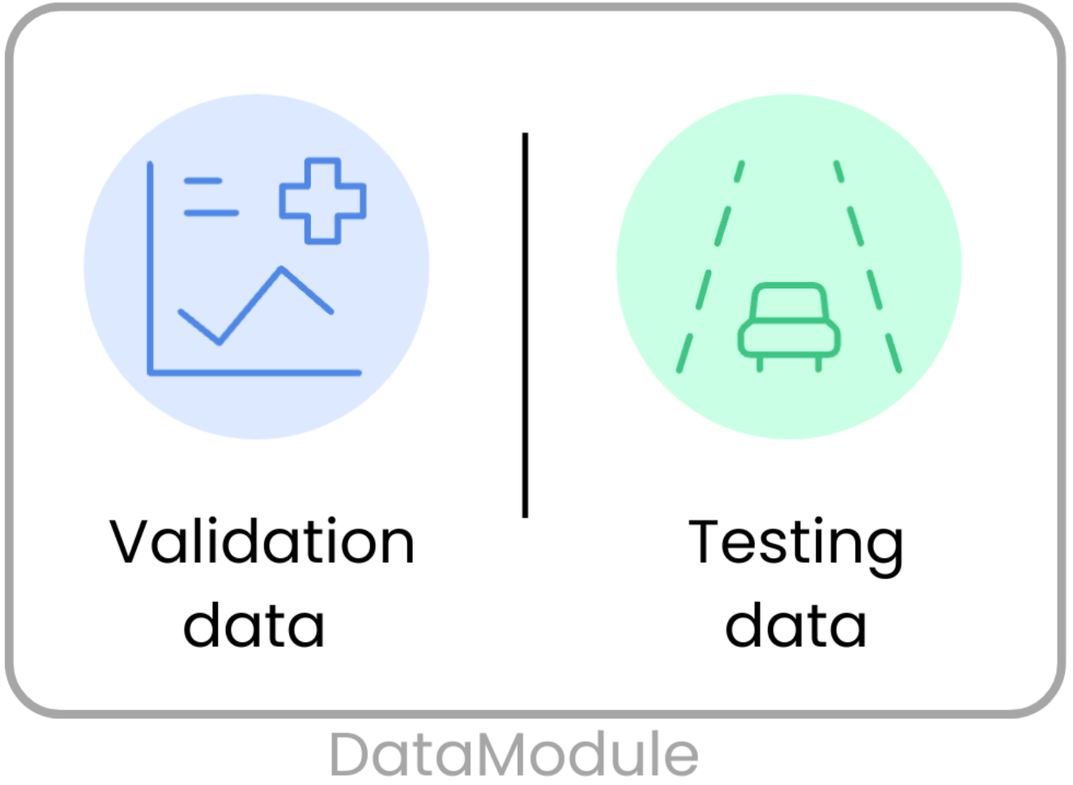
```
from torchmetrics import Accuracy

class BaseModel(pl.LightningModule):
    def __init__(self):
        super().__init__()
        self.accuracy = Accuracy()

    def validation_step(self, batch, batch_idx):
        x, y = batch
        preds = self(x)
        acc = self.accuracy(preds, y)
        self.log('val_acc', acc)
```

Connecting DataModule, validation, and testing

- Data logic centralized in DataModule
- Consistent train/val/test data splits
- Automatic validation metric logging
- Reproducible pipeline from prep to reporting



Let's practice!

SCALABLE AI MODELS WITH PYTORCH LIGHTNING

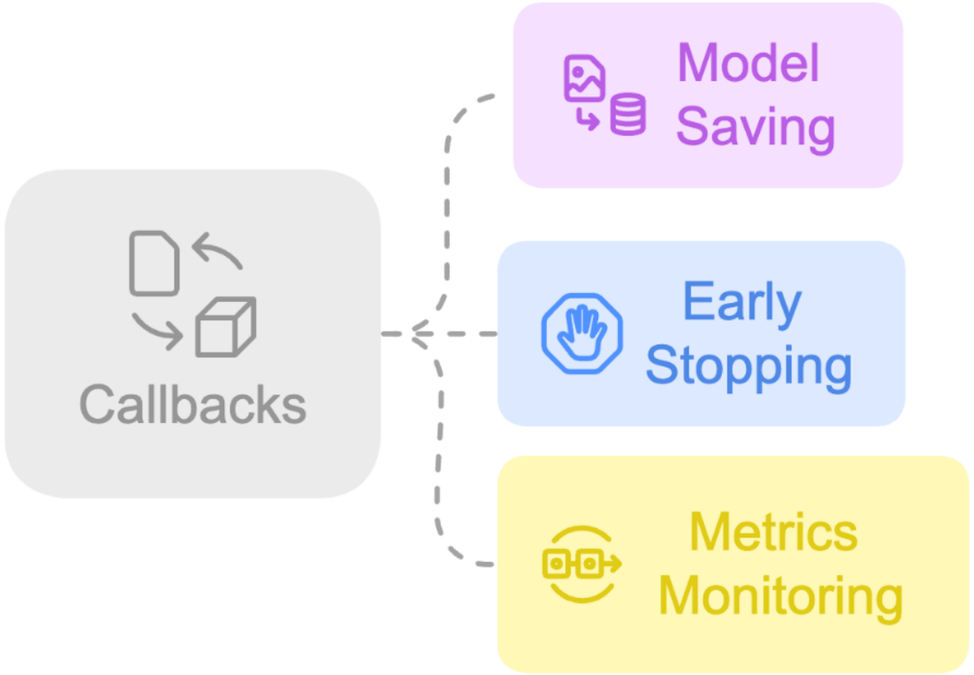
Enhancing training with Lightning callbacks

SCALABLE AI MODELS WITH PYTORCH LIGHTNING

Sergiy Tkachuk
Director, GenAI Productivity

What are callbacks?

- Functions executed at key stages of training
- Add custom actions without cluttering code
- Enhance flexibility and control



What are callbacks?

```
from lightning.pytorch.callbacks import Callback

class MyPrintingCallback(Callback):
    def on_train_start(self, trainer, pl_module):
        print("Training is starting")
    def on_train_end(self, trainer, pl_module):
        print("Training is ending")
```

- Adding custom actions at various stages of training

Lightning ModelCheckpoint callback

- Automatically saves model at specified intervals
- Choose metric to track
- Keep only the best model

```
from lightning.pytorch.callbacks
import ModelCheckpoint
checkpoint_callback = ModelCheckpoint(
    monitor='val_loss',
    dirpath='my/path/',
    filename='{epoch}-{val_loss:.2f}',
    save_top_k=1,
    mode='min'
)
```

[1] https://lightning.ai/docs/pytorch/stable/api/lightning.pytorch.callbacks.ModelCheckpoint.html

Lightning EarlyStopping callback

- Monitor a metric
- Stop training when the metric stops improving

```
from lightning.pytorch.callbacks
import EarlyStopping

early_stopping_callback = EarlyStopping(
    monitor='val_loss',
    patience=3,
    mode='min'
)
```

[1] https://lightning.ai/docs/pytorch/stable/api/lightning.pytorch.callbacks.EarlyStopping.html

Customizing and using lightning callbacks

```
from lightning.pytorch import Trainer
from lightning.pytorch.callbacks import EarlyStopping, ModelCheckpoint
checkpoint = ModelCheckpoint(
    monitor='val_accuracy',
    save_top_k=2,
    mode='max')
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=5,
    mode='max')

trainer = Trainer(max_epochs=50, callbacks=[checkpoint, early_stopping])
```

[1] https://lightning.ai/docs/pytorch/stable/common/trainer.html

Let's practice!

SCALABLE AI MODELS WITH PYTORCH LIGHTNING