AI CHEAT SHEET Nº 01

PYTORCH LIGHTNING

1 Prepare data 2 Define LightningModule 3 Configure Trainer 4 Fit, test & predict

01 GENERAL

PyTorch Lightning is a lightweight wrapper around PyTorch that **separates research code from engineering code**. You write the model logic in a **LightningModule**. The **Trainer** handles the loop, devices, precision, checkpointing and logging. No more **.to(device)**, **loss.backward()** or **optimizer.step()** by hand.

import lightning as L	Root package	**L.Trainer(...)**	Runs all loops
L.LightningModule	Model + logic	**L.seed_everything(42)**	Reproducibility
L.LightningDataModule	Data pipeline	**self.save_hyperparameters()**	Store init args

*Older code uses **import pytorch_lightning as pl**. Same API, legacy package name.*

02 LIGHTNINGMODULE HOOKS

forward(x): inference / prediction logic, used by **model(x)**

training_step(batch, idx): return the loss; Lightning runs backward & step

validation_step / test_step: compute & **self.log** metrics, no grads

predict_step: what **trainer.predict()** returns per batch

configure_optimizers: return optimizer(s) & LR scheduler(s)

on_train_epoch_end and 20+ more hooks to inject logic at any point

on_fit_start → training_step ×N → validation_step ×M → on_*_epoch_end ↺

03 PREPARE DATA

A **LightningDataModule** bundles download, split, transforms and the **DataLoader**s so the same data pipeline is reusable across experiments. Plain DataLoaders passed to **trainer.fit()** work too.

```python
class MNISTData(L.LightningDataModule):
    def __init__(self, data_dir="./data", batch_size=64):
        super().__init__()
        self.data_dir, self.bs = data_dir, batch_size

    def prepare_data(self):     # download once (rank 0 only)
        MNIST(self.data_dir, download=True)

    def setup(self, stage):     # split/transform on every GPU
        full = MNIST(self.data_dir, transform=ToTensor())
        self.train, self.val = random_split(full, [55000, 5000])

    def train_dataloader(self):
        return DataLoader(self.train, batch_size=self.bs,
                          shuffle=True, num_workers=4)

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

CALLBACKS (LIGHTNING.PYTORCH.CALLBACKS)

LearningRateMonitor()	Log the LR each step / epoch
StochasticWeightAveraging()	SWA for better generalization
Tuner(trainer).lr_find(model)	Suggest a learning rate (lightning.pytorch.tuner)

04 DEFINE LIGHTNINGMODULE

A LightningModule **is** an **nn.Module** with training hooks added. Any PyTorch model, e.g. an **nn.Sequential**, goes inside it.

```python
import torch, torch.nn as nn, torch.nn.functional as F
from torch.optim.lr_scheduler import CosineAnnealingLR
import lightning as L

class LitClassifier(L.LightningModule):
    def __init__(self, lr=1e-3):
        super().__init__()
        self.save_hyperparameters()   # -> self.hparams.lr
        self.net = nn.Sequential(
            nn.Flatten(), nn.Linear(28*28, 128),
            nn.ReLU(), nn.Dropout(0.2), nn.Linear(128, 10))

    def forward(self, x):
        return self.net(x)

    def training_step(self, batch, batch_idx):
        x, y = batch                  # already on device
        loss = F.cross_entropy(self(x), y)
        self.log("train_loss", loss, prog_bar=True)
        return loss          # Lightning does backward + step

    def validation_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = F.cross_entropy(logits, y)
        acc = (logits.argmax(1) == y).float().mean()
        self.log_dict({"val_loss": loss, "val_acc": acc},
                      prog_bar=True)

    def configure_optimizers(self):
        opt = torch.optim.Adam(self.parameters(), lr=self.hparams.lr)
        sched = CosineAnnealingLR(opt, T_max=self.trainer.max_epochs)
        return {"optimizer": opt, "lr_scheduler": sched}
```

05 LOGGING

self.log(name, value)	Logs a scalar. Use **on_step** / **on_epoch** to choose when it is reduced
prog_bar=True	Show the value in the progress bar
sync_dist=True	Average across GPUs in distributed runs
self.log_dict({...})	Log many metrics at once

LOGGERS (LIGHTNING.PYTORCH.LOGGERS)

TensorBoardLogger("logs/")	Default when tensorboard is installed
CSVLogger("logs/")	Plain CSV, zero dependencies
WandbLogger(project="x")	Weights & Biases (also MLflow, Comet, Neptune)

*Pass with **L.Trainer(logger=WandbLogger(...))**. For robust metrics use **torchmetrics**, e.g. **Accuracy(task="multiclass", num_classes=10)**.*

06 CONFIGURE TRAINER & FIT

```python
from lightning.pytorch.callbacks import (
    ModelCheckpoint, EarlyStopping)

ckpt = ModelCheckpoint(monitor="val_loss", mode="min")
stop = EarlyStopping(monitor="val_loss", patience=3)
trainer = L.Trainer(
    max_epochs=10,
    accelerator="auto",    # "gpu", "cpu", "mps"
    devices="auto",        # 1, 4, [0, 2], -1 = all
    precision="16-mixed",  # "bf16-mixed", "32-true"
    callbacks=[ckpt, stop])
model, dm = LitClassifier(lr=1e-3), MNISTData()
trainer.fit(model, datamodule=dm)   # train + val
trainer.test(model, datamodule=dm)  # test_step
preds = trainer.predict(model, loader)
```

USEFUL TRAINER FLAGS

strategy="ddp"	Multi-GPU (or "fsdp", "deepspeed")
num_nodes=2	Multi-node training
gradient_clip_val=1.0	Clip gradient norm
accumulate_grad_batches=4	Bigger effective batch size
val_check_interval=0.25	Validate 4× per epoch
log_every_n_steps=50	Logging frequency

DEBUGGING FLAGS

fast_dev_run=True	1 batch of train/val/test, catches bugs
overfit_batches=10	Can the model overfit 10 batches?
limit_train_batches=0.1	Use 10% of the data per epoch
detect_anomaly=True	Find the op that produced NaN / Inf
profiler="simple"	Time every hook ("advanced", "pytorch")

07 SAVE/LOAD CHECKPOINT

A **.ckpt** holds weights, optimizer, scheduler, epoch & hparams. **ModelCheckpoint** writes them to the log folder.

```python
model = LitClassifier.load_from_checkpoint("best.ckpt")
model.eval()                          # for inference
trainer.fit(model, datamodule=dm, ckpt_path="last")
trainer.save_checkpoint("manual.ckpt")
```

BEYOND THE TRAINER

torch.compile(model)	Compile, then fit as usual
LightningCLI(Model, Data)	YAML / CLI config for free
L.Fabric()	Own loop, Lightning scaling

github.com/arnabdeypolimi/ai-cheatsheet / CC BY 4.0 / v1.0 · 10/2026