What’s inside (ready to run):

* vgg6\_cifar/
  + models/vgg6.py — VGG6 with --activation and --use\_bn/--no\_bn
  + models/activations.py — ReLU/Sigmoid/Tanh/SiLU/GELU registry
  + data/cifar10.py — normalization + augmentations + dataloaders
  + engine/trainer.py — training/eval loops + cosine LR warmup
  + engine/optim\_factory.py — SGD, Nesterov-SGD, Adam/AdamW, Adagrad, RMSprop, Nadam
  + utils/logger.py, utils/metrics.py
  + scripts/
    - train\_baseline.py — Q1 baseline entry
    - train\_experiment.py — Q2–Q4 runner (activation/optimizer/BN/BS/epochs/LR/momentum)
    - sweep\_grid.py — **epochs list default = 20,40,60,80,100**
    - plot\_curves.py, plot\_scatter\_valacc\_vs\_step.py
* requirements.txt
* README.md with exact commands (Q1–Q5)

**Full run steps for every question (Jupyter/Colab or terminal)**

**0) Setup (once)**

**Colab**

!pip install -r requirements.txt

**Local**

pip install -r requirements.txt

**Q1. Training Baseline (10 pts)**

**(a) CIFAR-10 with normalization + augmentation**

python -m vgg6\_cifar.scripts.train\_baseline \

--data\_dir ./data --out\_dir ./runs/baseline \

--epochs 60 --batch\_size 128 --lr 0.1 --optimizer sgd --momentum 0.9 \

--weight\_decay 5e-4 --label\_smoothing 0.0 \

--aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42

The script prints transforms and normalization (mean/std) and saves them in logs.

**(b) Train one strong configuration**

* The command above is your baseline (VGG6 + ReLU + **BatchNorm ON** + cosine LR warmup + AMP).
* Best checkpoint (best.pt) is saved by **validation accuracy**.

**(c) Report final test top-1 + curves**

python -m vgg6\_cifar.scripts.plot\_curves \

--metrics\_csv ./runs/baseline/metrics.csv \

--out\_dir ./runs/baseline

Artifacts: final\_test\_metrics.json (contains test\_top1\_acc), loss\_curves.png, accuracy\_curves.png.

**Q2. Model Performance on Different Configurations (60 pts)**

We’ll use train\_experiment.py and (optionally) sweep\_grid.py.

**Q2(a) Vary activation (ReLU, Sigmoid, Tanh, SiLU, GELU)**

for act in relu silu gelu tanh sigmoid; do

python -m vgg6\_cifar.scripts.train\_experiment \

--data\_dir ./data --out\_dir ./runs/act\_${act} \

--activation $act --optimizer sgd --lr 0.1 --batch\_size 128 --epochs 40 \

--use\_bn \

--aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

done

* Keep optimizer/BS/LR the same so you isolate the activation’s effect.
* Compare best\_val\_acc + test\_top1\_acc across runs.

**Q2(b) Vary optimizer (SGD, Nesterov-SGD, Adam, AdamW, Adagrad, RMSprop, Nadam)**

python -m vgg6\_cifar.scripts.train\_experiment --data\_dir ./data --out\_dir ./runs/opt\_sgd --activation relu --optimizer sgd --lr 0.1 --batch\_size 128 --epochs 40 --use\_bn --aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

python -m vgg6\_cifar.scripts.train\_experiment --data\_dir ./data --out\_dir ./runs/opt\_nesterov --activation relu --optimizer nesterov-sgd --lr 0.1 --batch\_size 128 --epochs 40 --use\_bn --aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

python -m vgg6\_cifar.scripts.train\_experiment --data\_dir ./data --out\_dir ./runs/opt\_adam --activation relu --optimizer adam --lr 0.001 --batch\_size 128 --epochs 40 --use\_bn --aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

python -m vgg6\_cifar.scripts.train\_experiment --data\_dir ./data --out\_dir ./runs/opt\_adamw --activation relu --optimizer adamw --lr 0.001 --batch\_size 128 --epochs 40 --use\_bn --aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

python -m vgg6\_cifar.scripts.train\_experiment --data\_dir ./data --out\_dir ./runs/opt\_rmsprop --activation relu --optimizer rmsprop --lr 0.01 --batch\_size 128 --epochs 40 --use\_bn --aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

python -m vgg6\_cifar.scripts.train\_experiment --data\_dir ./data --out\_dir ./runs/opt\_nadam --activation relu --optimizer nadam --lr 0.001 --batch\_size 128 --epochs 40 --use\_bn --aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

python -m vgg6\_cifar.scripts.train\_experiment --data\_dir ./data --out\_dir ./runs/opt\_adagrad --activation relu --optimizer adagrad --lr 0.05 --batch\_size 128 --epochs 40 --use\_bn --aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

Why the different LRs? Adaptive optimizers generally want a smaller LR than SGD.

**Q2(c) Vary batch size, epochs, learning rate, momentum, BN**

Two illustrative runs:

# Small BS, longer train, smaller LR, momentum 0.9, BN on

python -m vgg6\_cifar.scripts.train\_experiment \

--data\_dir ./data --out\_dir ./runs/hp\_bs64\_e80\_lr005\_m09\_bnTrue \

--activation relu --optimizer sgd --lr 0.05 --batch\_size 64 --epochs 80 \

--momentum 0.9 --use\_bn \

--aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

# Large BS, shorter train, big LR, momentum 0.0, BN off

python -m vgg6\_cifar.scripts.train\_experiment \

--data\_dir ./data --out\_dir ./runs/hp\_bs256\_e40\_lr02\_m00\_bnFalse \

--activation relu --optimizer sgd --lr 0.2 --batch\_size 256 --epochs 40 \

--momentum 0.0 --no\_bn \

--aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

**Full grid sweep** across your requested axes, including **epochs = 20, 40, 60, 80, 100**:

python -m vgg6\_cifar.scripts.sweep\_grid \

--data\_dir ./data --base\_out ./runs/sweeps\_ext \

--epochs\_list 20,40,60,80,100 \

--batch\_sizes 32,64,128,256,512 \

--lrs 0.2,0.1,0.05,0.01,0.001 \

--momentums 0.0,0.9 \

--optimizers sgd,nesterov-sgd,adam,adamw,rmsprop,nadam,adagrad \

--activations relu,silu,gelu,tanh,sigmoid \

--batch\_norms true,false \

--amp --wandb --seed 42

This produces runs/sweeps\_ext/sweep\_summary.csv with columns:  
activation,use\_bn,batch\_size,epochs,lr,momentum,optimizer,best\_val\_acc,test\_top1\_acc.

**Q3. Plots (10 pts)**

**(a) W&B parallel-coordinates plot**

* Ensure your runs used --wandb.
* In W&B: add axes activation, use\_bn, batch\_size, epochs, lr, momentum, optimizer, best\_val\_acc.
* Screenshot it for the PDF.

**(b) Validation accuracy vs step (scatter)**

python -m vgg6\_cifar.scripts.plot\_scatter\_valacc\_vs\_step \

--metrics\_csv ./runs/act\_gelu/metrics.csv \

--out\_png ./runs/act\_gelu/scatter\_valacc\_vs\_step.png

**(c) Training/validation loss/accuracy curves**

python -m vgg6\_cifar.scripts.plot\_curves \

--metrics\_csv ./runs/act\_gelu/metrics.csv \

--out\_dir ./runs/act\_gelu

**Q4. Final Model Performance (10 pts)**

1. From the W&B parallel-coordinates, pick **exactly one** best-validation-accuracy configuration.
2. Re-run it to verify reproducibility:

python -m vgg6\_cifar.scripts.train\_experiment \

--data\_dir ./data --out\_dir ./runs/final\_best \

--activation <best\_act> \

--optimizer <best\_opt> \

--lr <best\_lr> \

--batch\_size <best\_bs> \

--epochs <best\_epochs> \

--momentum <best\_m> \

--use\_bn # or --no\_bn

--aug\_hflip --aug\_crop --aug\_cutout --aug\_jitter --amp --seed 42 --wandb

1. Put final\_test\_metrics.json and best.pt into your repo. Include the exact command above in your PDF.

**Q5. Reproducibility & Repository (10 pts)**

* Clean, modular code already present (models/data/engine/utils/scripts).
* README.md contains exact commands, environment details, and requirements.txt.
* Use --seed 42 everywhere.
* Upload trained model (runs/final\_best/best.pt) to GitHub; paste repo link inside the PDF.