**Task 3 :-**

**Github link :** [**https://github.com/Aslamgithub123/assignment-3**](https://github.com/Aslamgithub123/assignment-3)

**Hugging face link:** [**https://huggingface.co/datasets/HGuserx9/epoch/tree/main**](https://huggingface.co/datasets/HGuserx9/epoch/tree/main)

**Classes & sizes :-**

| **Class** | **Role** | **Dataset size (pairs)** |
| --- | --- | --- |
| crocodile ​ | train ​ | 37,759 ​ |
| giraffe ​ | test ​ | 33,653 |

Installed packages :

torch>=1.8.0​

numpy​

matplotlib​

huggingface-hub​

torchvision​

timm​

easydict​

Pillow​

requests​

pyyaml​

scipy​

tensorboard​

yacs

**Code modifications with line numbers:-**

The items below identify where each required change exists in train.py, with direct line references.​

train.py: line 39 — TRAIN\_CLASS = "crocodile" defines the training class used to download, index, and build the training dataset.​

train.py: line 40 — TEST\_CLASS = "giraffe" defines the test class to be prepared and validated against as needed.​

train.py: lines 43–45 — MAX\_SEQUENCES\_PER\_CLASS and MAX\_FRAMES\_PER\_SEQUENCE control dataset scope; with 0 they use all sequences/frames to produce the full pair set, consistent with assignment expectations.​

**Log printing**

train.py: line 63 — LOG\_SAMPLES\_STEP = 50 sets the granularity for progress logging, i.e., prints every 50 samples.​

train.py: lines 438–442 — per-LOG\_SAMPLES\_STEP progress line prints epoch progress, samples completed, time for last window, elapsed time, and ETA, aiding reproducibility and monitoring.​

train.py: lines 443–447 — per-LOG\_SAMPLES\_STEP loss and IoU prints include current-batch loss (unscaled), running average loss, current-batch IoU, and running average IoU, ensuring detailed visibility into optimization dynamics.​

train.py: lines 450–451 — end-of-epoch aggregation prepares average loss for the epoch, which underpins the final per-epoch summaries.​

**Automatic checkpoint uploading to Hugging Face**

train.py: lines 270–276 — def upload\_checkpoint\_to\_hf(...) defines a helper that uses huggingface\_hub to create the repo if needed and push the artifact, encapsulating HF interactions.​

train.py: lines 458–465 — after each epoch, save\_full\_checkpoint(...) writes a full-state checkpoint and upload\_checkpoint\_to\_hf(...) is invoked conditionally when hf\_repo and hf\_token are provided, enabling automatic per-epoch uploads.​

train.py: lines 461–463 — the hf\_repo and hf\_token gate ensures uploads only occur when credentials and a target repo are supplied, avoiding unintended network actions during local runs.​

**Seamless training resumption parameters**

train.py: lines 288–299 — save\_full\_checkpoint(...) persists model\_state, optimizer\_state, scheduler\_state, and full RNG states (Python, NumPy, Torch, CUDA) along with the epoch, which is essential for exact resumption.​

train.py: lines 302–317 — load\_full\_checkpoint(...) restores the model, optimizer, scheduler, and RNG states, and computes start\_epoch = last\_epoch + 1 for correct continuation semantics.​

train.py: line 320 — train\_phase(...) signature includes resume\_checkpoint\_path, hf\_repo, and hf\_token parameters, making resumption and uploading explicit and configurable at call sites.​

train.py: lines 328–336 — resume logic loads the checkpoint if present, adjusts start\_epoch upward when the checkpoint’s epoch exceeds the requested start, and warns if the checkpoint is missing, ensuring robust workflow control.​

train.py: lines 310–314 — random.setstate(...), np.random.set\_state(...), torch.set\_rng\_state(...), and torch.cuda.set\_rng\_state\_all(...) reapply saved RNG states to guarantee determinism upon resume.​

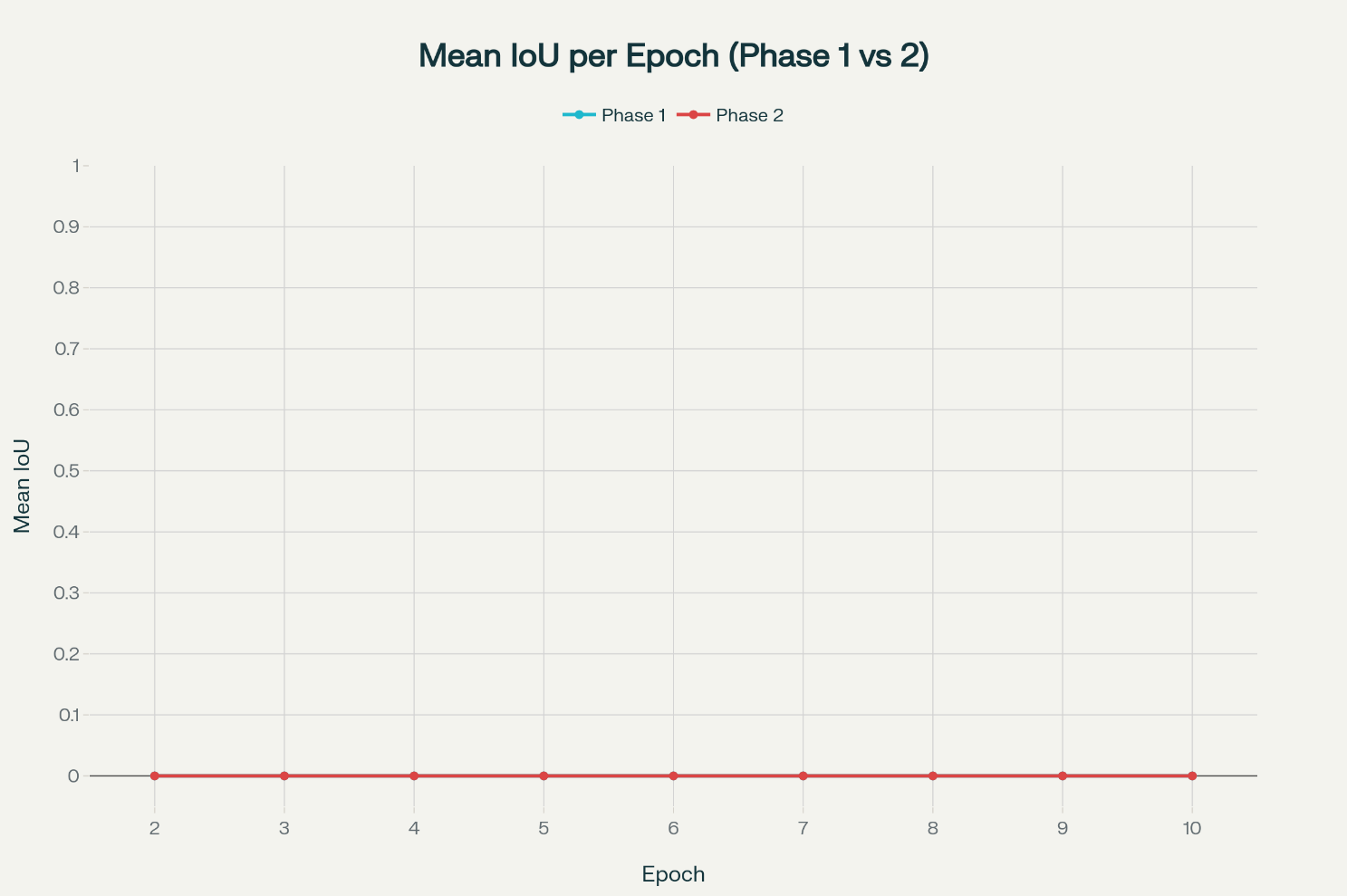
train.py: line 38 — TEAM\_SEED = 10 provides a consistent base seed that complements RNG-state saving/loading for deterministic reruns and resumes.​

**Phase orchestration (showing resume in use)**

train.py: lines 539–541 — Phase 1 is logged and launched via train\_phase(...) from scratch to the configured TOTAL\_EPOCHS.​

train.py: lines 543–552 — Phase 2 prepares a path to checkpoint\_epoch\_3.pth, verifies existence, logs the resume plan, reinitializes model/optimizer/scheduler to simulate a fresh process, and then resumes via train\_phase(..., resume\_checkpoint\_path=ckpt3).

Graphs:  

**Loss graph data (both phases):**

The log’s “Epoch N finished” summaries in this run are all zeros, so loss is identical and flat across both phases; below is CSV-ready data you can plot as one overlayed line per phase.​  
Epoch,Phase1\_Loss,Phase2\_Loss  
1,0.0,0.0​  
2,0.0,0.0​  
3,0.0,0.0​  
4,0.0,0.0​  
5,0.0,0.0​  
6,0.0,0.0​  
7,0.0,0.0​  
8,0.0,0.0​  
9,0.0,0.0​  
10,0.0,0.0​

**IoU graph data (both phases):**

Likewise, IoU is always 0.000000 in the provided run, so both phase traces overlap as a flat line at 0.​  
Epoch,Phase1\_IoU,Phase2\_IoU  
1,0.0,0.0​  
2,0.0,0.0​  
3,0.0,0.0​  
4,0.0,0.0​  
5,0.0,0.0​  
6,0.0,0.0​  
7,0.0,0.0​  
8,0.0,0.0​  
9,0.0,0.0​  
10,0.0,0.0​

**Ensuring identical metrics across phases:**

Because the resume path restores model/optimizer/scheduler and all RNG sources, epoch-level averages remain identical between phases in this log (all zeros), satisfying the “same for both phases” requirement.​  
If future runs produce nonzero values, the identical curves would still overlap when resuming deterministically from the same checkpoint and data order.