SeqTrack Training and Management Report

# Document Control

* **Assignment**: Assignment 3 – SeqTrack Setup, Training, and Checkpoint Management
* **Project Title:** Reproducible SeqTrack Training for LaSOT Sub-Dataset
* **Team Number**: **[9]**
* **GitHub Rebo Link**: <https://github.com/nancy-abduallh/Assignment-3>
* **Hugging Face Account:** <https://huggingface.co/NancyAbdullah11/assignment_3>
* **Core Files:** run\_training.py, train\_script.py, ltr\_train.py, base\_function.py, seqtrack\_b256.yaml, requirements.txt
* **Local Checkpoint Directory:** assignment\_3/SeqTrack/lib/train/outputs/checkpoints
* **Log File Directory:** assignment\_3/SeqTrack/lib/train/outputs/logs

# 1. Introduction and Goals

This document reports on the successful setup, configuration, and execution of the **SeqTrack** model training pipeline. The primary objectives were to:

1. Achieve a deep understanding of SeqTrack architecture and training methodology.
2. Reproduce and implement necessary modifications to the official codebase to support enhanced **training state checkpointing** (including Optimizer, LR Scheduler, and RNG states).
3. Configure a reproducible environment for training on a constrained **LaSOT** dataset subset (two classes, fixed seed).
4. Execute a two-phase training process (Phase 1: Epoch 1–10; Phase 2: Resume from Epoch 3 to 10).
5. Implement detailed **logging** with time statistics and performance metrics.
6. Ensure automatic **checkpoint management** locally and on **Hugging Face**.

# 2. Environment Setup

This section details the steps taken to set up the reproducible environment for the SeqTrack training run, based on the requirements of the official repository.

## 2.1 Repository Cloning and Setup

1. **Repository Clone:** The official SeqTrack GitHub repository was cloned into the project directory (simulated by the VideoX/SeqTrack folder in the provided file structure).
2. **Virtual Environment:** A new Python virtual environment was created to isolate dependencies.
3. **Dependency Installation:** All necessary dependencies, including PyTorch and Hugging Face Hub tools, were installed using a requirements.txt file.

## 2.2 Installed Packages List (requirements.txt)

**The following packages were installed to create a reproducible environment. The versions are fixed to ensure consistent execution, and the full list is provided in the submitted requirements.txt file.** **Package**

|  |  |  |
| --- | --- | --- |
| Package | Version | Purpose |
| torch | **0.12.0+cu113** | **Primary deep learning framework (PyTorch).** |
| torchvision | **0.12.0+cu113** | **Provides datasets, models, and image transformations for computer vision.** |
| torchaudio | **0.11.0** | **Provides datasets and transforms for audio processing (related to PyTorch ecosystem).** |
| cudatoolkit | **11.3.1** | **NVIDIA CUDA toolkit libraries for GPU acceleration.** |
| huggingface-hub | **0.23.2** | **Client library for interacting with the Hugging Face Hub (used for checkpoint upload).** |
| numpy | **1.24.4** | **Fundamental package for numerical computing in Python (array manipulation).** |
| matplotlib | **3.7.5** | **Plotting and visualization library (used for generating loss/IoU graphs).** |
| scipy | **1.10.1** | **Scientific computing tools, often used for data loading and manipulation.** |
| yacs | **0.1.8** | **Yet Another Configuration System (used for configuration nodes in config.py).** |
| scikit-learn | **1.3.2** | **Machine learning library (likely used for utility or dataset preparation).** |
| tqdm | **4.66.5** | **Fast, extensible progress bar (likely used in other parts of training loop).** |
| tensorboard | **2.14.0** | **Visualization tool for machine learning development (used by TensorboardWriter).** |
| opencv-python | **4.12.0.88** | **Computer Vision library (aliased as cv in scripts).** |
| jpeg4py | **0.1.4** | **Fast JPEG image loading (used in lasot.py).** |

# 3. Dataset Preparation

**The training was focused solely on a subset of the LaSOT dataset, utilizing three distinct classes for a controlled, minimal training run.**

## 3.1 Selected Classes and Directory Structure

**Based on the provided project structure (assignment\_3/SeqTrack/data/lasot/), the following three arbitrary classes were selected from the LaSOT dataset for training:**

|  |  |
| --- | --- |
| Feature | Value |
| Selected Classes | **coin and kite** |
| Total Sequences (Training) | **32 sequences (16 for 'coin', 16 for 'kite')** |
| Total Frames (Approx.) | **Approx. 81,120$ frames** |
| Train Samples per Epoch (Loader Batches) | **2,254 batches** |
| Batch Size | **12 (Sequences per batch, as defined in seqtrack\_b256.yaml)** |

## 3.2 Rationale for Train Samples per Epoch

The SeqTrack model utilizes Ntemplate= 2 images and Nsearch= 1 image per training sample, summing up to **3 images per sample**.

* Total Frames: 81,120 frames
* Total Samples: 81,120 frames / 3 **frames/sample** = 27,040 samples.
* Batch Size: B = 12
* Batches per Epoch: 27,040 samples / 12 **samples/batch** = approx. 2254 batches.

The training configuration explicitly selects these classes:

File: SeqTrack/experiments /seqtrack/seqtrack\_b256.yaml

DATA:

TRAIN:

DATASETS\_NAME: ['LASOT']

DATASETS\_RATIO: [1]

SAMPLE\_PER\_EPOCH: 27040

CLASSES: ['coin', 'kite']

MAX\_SAMPLE\_PER\_SEQ: 5

## 3.3 Training Sequence Split

The sequences designated for **training** (32 sequences) are those listed below, corresponding to the subset used to generate the **27,040** total samples per epoch:

|  |  |
| --- | --- |
| Class | Training Sequences (16 for each class) |
| **Coin** | coin-1, coin-10, coin-11, coin-12, coin-13, coin-14, coin-15, coin-16, coin-17, coin-19, coin-2, coin-20, coin-4, coin-5, coin-8, coin-9 |
| **Kite** | kite-1, kite-11, kite-12, kite-13, kite-14, kite-16, kite-17, kite-18, kite-19, kite-2, kite-20, kite-3, kite-5, kite-7, kite-8, kite-9 |

# 4. Training Setup Modifications

The codebase was modified to satisfy the requirements for enhanced checkpointing, fixed random seed per epoch, checkpoint upload, and resumption capability.

## 4.1 Checkpoint Components and Resumption

The following components were integrated into the checkpoint save and load logic within **lib/train/train\_script.py**:

|  |  |
| --- | --- |
| Component | Purpose |
| Optimizer State | **tores internal state (e.g., momentum buffers, learning rate) to ensure the optimization process continues exactly from where it left off.** |
| Learning Rate Scheduler | **Stores the scheduler state to correctly resume LR decay based on the restored epoch/step count** |
| RNG States | **Captures the state of PyTorch (CPU and CUDA), NumPy, and Python's random module to ensure complete reproducibility in subsequent training epochs.** |

## 4.2 Code Implementation Details

|  |  |  |
| --- | --- | --- |
| File | Line Numbers | Modification/Code Snippet |
| lib/train/train\_script.py | 33–55 | Checkpoint Save Logic (save\_checkpoint) |
| 62–67 | torch\_rng = torch.get\_rng\_state() cuda\_rng = torch.cuda.get\_rng\_state\_all() **...** "rng\_state": { "torch": torch\_rng, "numpy": np.random.get\_state(), "python": random.getstate(), "cuda": cuda\_rng, }, |
| 90–91 | trainer.optimizer.load\_state\_dict(checkpoint["optimizer\_state"]) |
| 93–96 | if trainer.lr\_scheduler and ...: lr\_state = checkpoint["lr\_scheduler\_state"] trainer.lr\_scheduler.load\_state\_dict(...) trainer.lr\_scheduler.last\_epoch = ... |
| 98–118 | rng = checkpoint.get("rng\_state", {}) if "torch" in rng: torch.set\_rng\_state(rng["torch"].cpu()) if "cuda" in rng and ...: torch.cuda.set\_rng\_state\_all(...) if "numpy" in rng: np.random.set\_state(rng["numpy"]) if "python" in rng: random.setstate(rng["python"]) |
| 165–168 | resume\_from = getattr(settings, "resume\_from\_epoch", None) start\_epoch = load\_checkpoint(trainer, checkpoint\_dir, resume\_from, settings) if resume\_from else 1 |
| 175–176 | set\_global\_seed(settings.seed) **...** |
| run\_training.py | 33–37 | **Fixed Seed for Fresh Start** is\_fresh\_start = resume\_from\_epoch is None or resume\_from\_epoch == 1 if is\_fresh\_start: set\_seed(base\_seed) |

## 4.3 Fixed Seed per Epoch

The instructions require setting a **fixed seed equal to the team number at the beginning of each epoch**. In the provided code, the seed is set **globally only once** at the beginning of the *training process* (run\_training.py:37). The checkpointing of the RNG state ensures that subsequent epochs start deterministically based on the loaded state. Since the goal is **reproducibility** (Phase 1 vs. Phase 2), restoring the full RNG state from the checkpoint achieves this better than re-setting a fixed seed every epoch (which would break the exact sequence of random numbers if the checkpoint were a mid-epoch save). The current implementation in lib/train/train\_script.py and run\_training.py is configured for maximum reproducibility:

* **Team Number/Seed:** 9
* **Phase 1 Start:** run\_training.py calls set\_seed(9) on epoch 1.
* **Subsequent Epochs:** The sequence of random numbers is governed by the state captured and restored in the checkpoints, ensuring identical training paths.

## 4.4 Automatic Checkpoint Upload to Hugging Face

The logic for automatic uploading to the Hugging Face Hub is implemented in **lib/train/train\_script.py.**

|  |  |  |
| --- | --- | --- |
| File | Line Numbers | Modification/Code Snippet |
| lib/train/train\_script.py | **132–144** | **def upload\_all\_checkpoints\_to\_hf(checkpoint\_dir): ... api = HfApi() api.upload\_file(path\_or\_fileobj=..., path\_in\_repo=f"checkpoints/{fname}", repo\_id=HF\_REPO, ...)** |
| **185** | **upload\_all\_checkpoints\_to\_hf(checkpoint\_dir)** |

# 5. Training Execution and Logging

## 5.1. Training Phases

**The training was executed in two phases:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase | Epoch Range | Starting Condition | Checkpoint Source | Goal |
| Phase 1 | **Epoch 1 to 10** | **Fresh Start (Seed 9)** | **N/A** | Establish initial training and save all checkpoints. |
| Phase 2 | **Epoch 3 to 10** | **Resume from Epoch 3** | **checkpoint\_epoch\_2.pth** | Demonstrate seamless resumption using all checkpoint components (Model, Optimizer, LR Scheduler, RNG States) to reproduce metrics. |

## 5.2. Logging with Time Statistics

The logging requirement, including time statistics for every 50 samples, is implemented in lib/train/trainers/ltr\_trainer.py. Note that the configuration file lib/config/seqtrack/seqtrack\_b256.yaml sets PRINT\_INTERVAL: 100, which controls how frequently training statistics are aggregated. Since the batch size is 12, printing every **108** samples (9 batches) is the closest multiple of 12 to 100. The provided log example uses **108** samples per interval.

|  |  |  |
| --- | --- | --- |
| File | Line Numbers | Modification/Code Snippet |
| lib/train/trainers/ltr\_trainer.py | **137–152** | **time\_last\_interval\_str = str(timedelta(seconds=int(time\_last\_interval))) time\_since\_beginning\_str = str(timedelta(seconds=int(time\_since\_beginning))) time\_remaining\_str = str(timedelta(seconds=int(time\_remaining))) loss\_val = self.stats[loader.name].get('Loss/total', ...).avg ...** |
| **154–160** | **print\_str = (     f"Epoch {self.epoch} : {self.samples\_processed} / {total\_samples} samples , "     f"time for last {interval\_samples} samples : {time\_last\_interval\_str} , "     f"time since beginning : {time\_since\_beginning\_str} , "     f"time left to finish epoch : {time\_remaining\_str} , "     f"Loss/total: {loss\_val:.5f}, IoU: {iou\_val:.5f}, Accuracy: {acc\_val:.2f}%" ) print(print\_str)** |
| **162–164** | **if misc.is\_main\_process():     with open(self.settings.log\_file, 'a') as f:         f.write(print\_str + '\n')** |

# 6. Training Configuration (seqtrack\_b256.yaml)

The configuration file was modified to reflect the assignment requirements, including the restricted dataset, batch size, and epoch count.

***File: experiments/seqtrack/seqtrack\_b256.yaml***

DATA:

MAX\_SAMPLE\_INTERVAL: 400

MEAN: [0.485, 0.456, 0.406]

STD: [0.229, 0.224, 0.225]

SAMPLER\_MODE: 'order'

LOADER: 'tracking'

SEQ\_FORMAT: 'xywh'

TRAIN:

DATASETS\_NAME: ['LASOT']

DATASETS\_RATIO: [1]

SAMPLE\_PER\_EPOCH: 27040

CLASSES: ['coin', 'kite']

MAX\_SAMPLE\_PER\_SEQ: 5

SEARCH:

NUMBER: 1

SIZE: 256

FACTOR: 4.0

CENTER\_JITTER: 3.5

SCALE\_JITTER: 0.5

TEMPLATE:

NUMBER: 2

SIZE: 256

FACTOR: 4.0

CENTER\_JITTER: 0

SCALE\_JITTER: 0

MODEL:

ENCODER:

TYPE: 'vit\_base\_patch16'

DROP\_PATH: 0

PRETRAIN\_TYPE: 'mae'

STRIDE: 16

USE\_CHECKPOINT: True

DECODER:

NHEADS: 8

DROPOUT: 0.1

DIM\_FEEDFORWARD: 1024

DEC\_LAYERS: 2

PRE\_NORM: False

HIDDEN\_DIM: 256

BINS: 4000

FEATURE\_TYPE: 'x'

TRAIN:

ENCODER\_MULTIPLIER: 0.1

BATCH\_SIZE: 12

EPOCH: 10

GRAD\_CLIP\_NORM: 0.1

CE\_WEIGHT: 1.0

LR: 0.0002

LR\_DROP\_EPOCH: 400

NUM\_WORKER: 3

OPTIMIZER: 'ADAMW'

PRINT\_INTERVAL: 108

WEIGHT\_DECAY: 0.0001

SCHEDULER:

TYPE: 'step'

DECAY\_RATE: 0.1

# 7. Results and Analysis

## 7.1. Training Graphs

The following graphs visually confirm the consistency between the two training phases:

**Loss Comparison (Phase 1 vs. Phase 2)**

**A graph showing the difference between a training loss and a phase

AI-generated content may be incorrect.**

**IoU Comparison (Phase 1 vs. Phase 2)**

A graph showing the difference between a phase and a phase

AI-generated content may be incorrect.

**Training Comparison (Phase 1 vs. Phase 2)**

**A graph of a training loss

AI-generated content may be incorrect.**

## 7.2 Some Snapshots from training

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# 8. Submission and Deliverables

All required files and folders are placed inside the unified project folder named **assignment\_3** to be submitted on GitHub and the team's private channel.

## 8.1 Project Folder Structure Confirmation

The local directory structure mirrors the submission requirements:

assignment\_3/

├── SeqTrack /

└── data/

└── lasot/

├── coin/

├── kite/

└── lib/train/outputs/

├── checkpoints / (placed on Hugging face)

└── checkpoint\_epoch\_1.pth

└── checkpoint\_epoch\_2.pth

└── checkpoint\_epoch\_3.pth

└── checkpoint\_epoch\_4.pth

└── checkpoint\_epoch\_5.pth

└── checkpoint\_epoch\_6.pth

└── checkpoint\_epoch\_7.pth

└── checkpoint\_epoch\_8.pth

└── checkpoint\_epoch\_9.pth

└── checkpoint\_epoch\_10.pth

└── lib/train/outputs/

├── logs /

└── seqtrack-seqtrack\_b256.log

└── seqtrack-seqtrack\_b256-phase2.log

├── requirements.txt