U0.L2 — Training Hygiene on CIFAR-10

Flow-Based Generative Models · UFRJ · 2026.2

01 From template to stack

Today your training loop grows up

By the end of these 90 minutes, the U0.L1 `training-loop-template` becomes the **hygiene stack** — template **v2**:

- **config-driven runs** — no magic numbers anywhere,
- **experiment tracking** (MLflow) — every run logged, comparable, permanent,
- **schedules + checkpoint discipline** — the U0.T2 recipe card, wired in,
- **reproducibility** — seed, config, commit, environment: all recorded.

The stack is frozen today and never re-taught. PS0 will require, verbatim: *"trained with the hygiene stack"* — after today that is a **checkable claim**.

Centerpiece (30 min): the *mystery run* — one of the runs you will be handed has been sabotaged. You will diagnose it **from its curves alone**.

Why this lab is scripts, not a notebook

Hygiene is about **runs** — things you launch, tag, compare, and resume. A run is a terminal artifact, not a cell you re-execute:

```
python train_cifar.py --config configs/baseline.yaml
```

Two new top-level directories in the course repo layout:

```
your-course-repo/
  labs/       # notebooks return in U0.L3
  common/     # train.py: v1 now, v2 today
  configs/    # NEW: one yaml per run
  runs/       # checkpoints (gitignored)
  mlflow.db   # NEW: MLflow store (sqlite, gitignored)
```

Notebooks are for studying *code*; scripts are for studying *runs*. Today's object of study is the run.

One run = one config = one MLflow run

CONFIGS/BASELINE.YAML — THE WHOLE EXPERIMENT, AS DATA

```yaml
run_name: U0L2-baseline-s0
seed: 0
data:  {dataset: cifar10, normalize: true, batch: 256}
model: {arch: resnet9, norm: batch}  # classifier: BN ok
opt:   {name: adamw, lr: 3.0e-4, weight_decay: 0.01}
sched: {name: warmup_cosine, warmup_frac: 0.03}
epochs: 8
```

No magic numbers in code — anything that can vary between runs lives here; two runs differ by a config **diff**, never an edit history.

TODO A.1 (config part): wire the config into the U0.L1 `Trainer` — interface preserved (v2 adds only keyword-only optional args).

02 Experiment tracking with MLflow

The standing decision: MLflow, running locally

The course logging standard, fixed today and used through U3:

- **Open source, zero vendor accounts** — nothing to sign up for,
- **your data stays on your machine** — a local SQLite file, fully offline,
- **production alignment** — the tracker you are most likely to meet at work,
- **run comparison built in** — the overlay view is exactly what the mystery run (today) and the U3.L1 payoff plot (later) need.

```
mlflow ui --backend-store-uri sqlite:///mlflow.db
```

You may meet **wandb** or **TensorBoard** elsewhere — same concepts, different plumbing. We name them once and move on.

Wiring the Trainer

Reused in: every training run for the rest of the course.

```
import mlflow
mlflow.set_tracking_uri("sqlite:///mlflow.db")
mlflow.set_experiment("fbgm-2026")  # ONE per course phase

with mlflow.start_run(run_name=cfg.run_name):
    mlflow.log_params(flatten(cfg))  # resolved config
    mlflow.set_tags({"session": "U0.L2",
                     "tag": "baseline", "seed": cfg.seed})
    trainer.fit(train_loader, val_loader, cfg.epochs)
```

TODO B.1 (tracking part): add the `mlflow.log_metric` calls inside `fit` — **per step:** train loss, lr; **per epoch:** val loss/acc, grad norm (global L_2), weight norm, throughput (img/s).

Naming and tags — conventions that scale to U3

One experiment per course phase; tags do the filtering.

Convention	Value
Experiment	`fbgm-2026`
Tags	`session` = `U0.L2` · `tag` = `baseline` · `seed` = `0`
Run name	`{session}-{tag}-s{seed}` → `U0L2-baseline-s0`

Why tags and not one experiment per session: **cross-session overlays**. In U3.L1 you will overlay your U2.L2 (simulation-based) runs against your U3.L1 (simulation-free) runs — same experiment, filtered by `session` tag. That plot is the punchline of the whole course arc; the convention that makes it possible costs nothing today.

What to log, and why

This checklist is one half of the `hygiene-stack` artifact.

CHECKLIST — WHAT EVERY RUN LOGS (HYGIENE-STACK, PART 1)

1. **Train AND val loss** — the gap between them is the first diagnostic.
2. **Learning rate** — schedule bugs are the #1 silent killer; log what the optimizer actually *used*, not what the config *asked for*.
3. **Grad norm** (global L_2) — explosions and vanishing, visible live; U0.T2's init/scale pictures made these curves readable.
4. **Weight norm** — decay and init sanity.
5. **Throughput (img/s)** — a sudden drop is the silent-CPU-fallback detector.
6. **Sample/prediction figures** via `mlflow.log_figure` — in U3 this same call logs generated-image panels.

Live: two seeds in the compare view

Launch two short baseline runs, seeds 0 and 1, then:

```
mlflow ui          # select both runs -> Compare
```

What you should see, and say out loud:

- the two loss curves **wiggle differently but land together** — that spread is the seed variance you eyeballed in the U0.L1 homework-lite, now measured;
- the LR curves are **identical** — the schedule is deterministic;
- params differ in exactly **one field**: `seed`. Config diff = run diff.

03 Schedules and checkpoints

The recipe card, back on screen

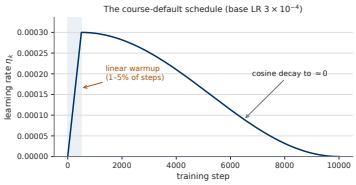
Cited from the previous theory session, not re-derived — this box is **the U0.T2 defaults**:

RECIPE CARD — OPTIMIZER-SCHEDULE-DEFAULTS (CITE AS: THE U0.T2 DEFAULTS)

1. **Optimizer:** AdamW, base LR 3×10^{-4}, β = (0.9, 0.999), weight decay 0.01; norm parameters and embeddings excluded from decay.
2. **Schedule:** linear warmup over 1–5% of steps, then cosine decay to ≈ 0.
3. **Initialization:** He for ReLU-family; zero-init last layers where sensible.
4. **Normalization:** GN/LN for generative nets; **BatchNorm only in bootcamp classifiers** — that is us, today, deliberately.
5. **Batch size:** largest that fits; always reported.

Today the card stops being advice and becomes **running code**.

Wiring warmup–cosine




```
def warmup_cosine(step):          # multiplier on base LR
    if step < warmup:
        return step / warmup
    p = (step - warmup) / (total - warmup)
    return 0.5 * (1 + math.cos(math.pi * p))
```

TODO C.1 (schedule part): wrap in `LambdaLR`, step per *step*, and check the logged LR traces this curve. **The plot is the test.**

Checkpoint discipline: `last.pt` and `best.pt`

Two files per run, updated on different triggers:

- **`last.pt`** — every epoch, unconditionally: the resume point.
- **`best.pt`** — only when the **val** metric improves: the deliverable. Selection is on val, by a named metric — never on test.

CHECKPOINT CONTRACT — WHAT SAVE_CKPT(TAG) STORES (V2)

```
{"model": model.state_dict(), "opt": opt.state_dict(),
 "sched": sched.state_dict(), "epoch": epoch,
 "config": cfg, "mlflow_run_id": run_id}
```

A checkpoint that cannot **resume** is just a weights file — hence the optimizer/scheduler state and the config.

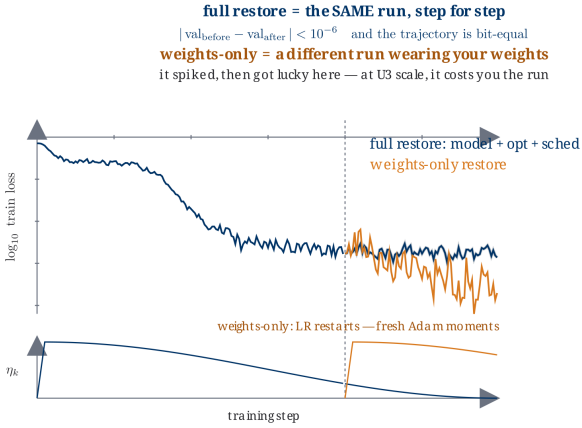
Resume must actually work

TODO C.2 (checkpoint part): implement resume-from-checkpoint in the v2 `Trainer`, then prove it with the **round-trip assert**:

```
# train 1 epoch -> save -> load into a fresh Trainer
# -> evaluate: the val loss must come back identical
assert abs(val_before - val_after) < 1e-6
```

Why we are strict *now*: in U3 your models cross session boundaries — U3.L2's trained flow is reused in U3.L3 for classifier-free guidance. A resume bug discovered *there* costs a lab; discovered *here* it costs five minutes.

The bad resume, animated



Animated in the web deck — final frame shown.

04 Reproducibility

Seeds, again — now with dataloaders

`set_seed(cfg.seed)` from U0.L1 covers python / NumPy / torch / CUDA. Two gaps it does **not** cover:

```
loader = DataLoader(
    ds, batch_size=cfg.data.batch_size, shuffle=True,
    generator=torch.Generator().manual_seed(cfg.seed),
    worker_init_fn=seed_worker)  # workers: derived seeds
```

- Without these, **shuffling order changes with worker count**.
- Full determinism exists: `torch.use_deterministic_algorithms(True)` — honest caveat: slower, and some GPU ops are simply unsupported.

Course policy, verbatim: *deterministic for debugging, fast for training, always seeded.*

Capture the environment

A run you cannot reconstruct is a run you cannot defend. Three captures, all cheap:

- **Environment:** `pip freeze > requirements.lock` — committed per lab.
- **Code version:** the git commit, as an MLflow tag:

```
sha = subprocess.check_output(
    ["git", "rev-parse", "HEAD"]).decode().strip()
mlflow.set_tag("git_commit", sha)
```

- MLflow **auto-captures** the commit for script runs (verified) — but records **no dirty flag**: on a dirty tree the auto-SHA is a lie.

TODO D.1 (repro part): find `mlflow.source.git.commit` in your tags; now look for a dirty flag — none. Hence our own `git_commit` + `git_dirty`.

The reproducibility checklist

The other half of the `hygiene-stack` artifact.

CHECKLIST — REPRODUCIBILITY (HYGIENE-STACK, PART 2)

1. **Seed** — set once, logged as a param, workers included.
2. **Config** — the resolved config logged at run start; no magic numbers.
3. **Commit** — git hash as a tag; dirty tree means the hash is a lie.
4. **Environment** — lockfile committed alongside the lab.
5. **Data version** — dataset name + version pinned in the config.

Five lines. PSO is graded on **correctness + reproducibility** — this checklist, not sample quality, is the rubric.

05 The mystery run

The mystery run: setup

One of these runs is broken. Nobody will tell you how.

```
unzip mystery-runs.zip
mlflow ui --backend-store-uri sqlite:///mystery.db
```

Inside: sabotaged runs and clean baselines (3 seeds each), plus the instructor's 30-epoch reference run. Fully offline; identical for everyone.

Rules of engagement:

- The logged configs have been **scrubbed** — you may not read the bug off a param. **Curves and metrics only.**
- Everything you need was covered in the last 60 minutes.

The task

From curves alone, in pairs:

1. **Describe** the pathology — what is wrong, in observable terms?
2. Name **at least two candidate causes**.
3. **Rank** them — which is more likely, given *these* curves?
4. Propose the **single cheapest discriminating experiment** — the one measurement that best separates your candidates.
5. **Run it.** Confirm or kill your hypothesis.

Steps 3–4 are the discipline being trained: not “what could be wrong” but *“what would you look at first, and why.”*

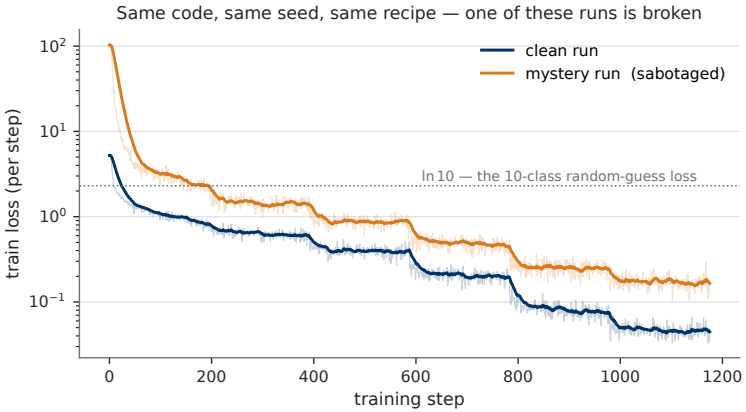
How the 30 minutes run

Phase	Min	What happens
Silent reading	10	Pairs; curves on screen
Board round	10	Hypotheses; forced ranking
Discriminate	10	Run the test; confirm; reveal

What a discriminating experiment looks like here:

- log the **input batch statistics** (mean/std) of the training data,
- or overlay **per-layer grad or weight norms**, mystery vs clean,
- or a 20-step run at a **lower LR** — does the pathology move?

The presenting symptom



Same architecture, same optimizer, same schedule, same number of steps.

The lesson

> ***You debugged a model without reading its config. In U2 and U3 the models get stranger, but the curves speak the same language. Log first, hypothesize second, discriminate cheaply third.***

This is why Block B's checklist exists: at minute 50 you diagnosed *only* what someone had logged at minute 20.

- In U2/U3 you cannot eyeball a velocity field and see the bug.
- You *can* read its loss, its grad norms, its NFE, its throughput.
- The instrument panel is the same; only the aircraft changes.

06 Baseline and freeze

Launch the clean baseline

Everything, together, once:

```
python train_cifar.py --config configs/baseline.yaml
```

- Short run, live (5–8 epochs — curves, not accuracy, are the object).
- In the compare view: overlay your live run against the **30-epoch reference** that shipped inside `mystery-runs.zip`.
- Your curve should trace the reference's early epochs. If it does not, you now own the diagnostic toolkit to say *why*.

Freeze: the hygiene stack

ARTIFACT — HYGIENE-STACK (FROZEN TODAY)

hygiene-stack = `training-loop-template` **v2** (config-driven + MLflow + schedules + checkpoint discipline) **+** the what-to-log checklist **+** the reproducibility checklist.

- Commit it now. This exact code trains every model you build in this course.
- **PSo requires, verbatim: "trained with the hygiene stack."** Config + seed + commit + MLflow store in the submission make that claim checkable.
- Deviations from the stack in later labs: permitted **in writing, with a reason** — same rule as the recipe card.

What's next

- **Next session (U0.T3): we go latent.** Latent-variable models, the ELBO, and the VAE — the first models of the course that *generate*.
- The hygiene stack is **assumed from here on** — never re-taught, always required.
- Your CIFAR-10 baseline keeps training tonight; look at its curves tomorrow. You know how now.