Embedded Artificial Intelligence

SETI Master

Lab 1 — Vitis AI Workflow

Apptainer, conda, tagged runs and first frugal comparisons

Author:	Alaa Eddine Mazouz
Academic year:	2026–2027
Version:	1.11
Date:	26 October 2026
Dataset:	FashionMNIST (final task: CIFAR-10)
Tools:	Vitis AI PyTorch CPU container (Apptainer)

Télécom Paris

Work in the lab folder `TP01_vitis_ai_workflow/`, which contains `scripts/` and `src/`.

Raw datasets. Raw data are shared in the course-root `datasets/` folder; a standalone lab uses `$HOME/.cache/c5/datasets` (or `$XDG_CACHE_HOME/c5/datasets`). Set `C5_DATA=/absolute/path` before sourcing `scripts/00_source_env.sh` to override it. Existing datasets stay in place; experiment outputs stay in this lab.

Purpose of this lab

Lab 1 sets up the workflow that every later lab reuses: the Vitis AI container, its conda environment, and *tagged runs*, where each experiment is one named folder with its settings, logs, models and results. You train small MLPs and CNNs on FashionMNIST, export each one as a quantized Vitis AI model (XMODEL), and ask what a larger model or a longer training actually buys. The ideas behind it (data splits, parameters and MACs, latency and throughput, the DPU flow) are in the L01 slides. The board is not used in this lab; Lab 2 compiles the XMODEL for the KV260.

Plan

Part	Content	Section
Setup	the container and its launcher	1
Tagged runs	first run, run folder, data splits	2
Block A	four models, same training budget	3
Block B	one model, three epoch budgets	4
Report	what the summary and the figure contain	5
Final task	the same study on CIFAR-10	6
Extension	CPU latency in the report (code to write; optional)	7

Start a run, then read the next section or answer the questions while it trains.

Before starting

Each listing starts with a comment that says where it runs:

- `# host`: a terminal on the lab machine, in the `TP01_vitis_ai_workflow` folder. The lab machines open `zsh`: type `bash` once in every new terminal, before the `source` commands of the course (scripts started with `bash script.sh` need nothing);
- `# container`: a shell inside the Vitis AI container, in the same folder; its prompt starts with `(vitis-ai-pytorch)`.

Datasets

FashionMNIST (30 MB) and, for the final task, CIFAR-10 (170 MB) are downloaded on first use, which is slow on the lab network (CIFAR-10 can take an hour). Keep every lab folder of the course side by side in one folder, for example `~/c5/`. As soon as two lab folders are there, the lab scripts keep the datasets of all the labs in `datasets/` in that folder and `data/` holds links to it. A standalone lab uses `${XDG_CACHE_HOME:-$HOME/.cache}/c5/datasets`; set an absolute `C5_DATA` to choose another store. Existing raw data are reused in place: the next lab neither moves nor overwrites them. When the lab machine has the course's shared copy of the datasets, the links point to it and nothing is downloaded. `00_source_env.sh`, below, makes the links.

Three contexts

- The **host** is your normal Linux session: it gives you the machine and your files.
- The **Apptainer container** runs the Vitis AI software stack, stored in one image (a `.sif`

file), on top of the host. The image is read-only; the lab folder and your home folder stay visible inside, so the files you create there remain after you leave the container.

- The **conda environment** `vitis-ai-pytorch`, inside the container, provides the Python packages: PyTorch and the Vitis AI quantizer `pytorch_nndct`.

1 Set up the environment

Type `bash` and press Enter separately in each new host terminal. Wait for the Bash prompt before pasting a command block. The first `cd` below assumes you are at the course checkout root.

Normal student entry

The supported entry point activates and checks the pinned Conda environment:

```
# host (after entering Bash separately)
cd TP01_vitis_ai_workflow
source scripts/00_source_env.sh
apptainer --version
bash scripts/01_host_sanity.sh
bash scripts/02_enter_container.sh
```

Wait for `Conda environment ready.`, then run a lightweight first command inside the container:

```
# container
bash scripts/04_train_ptq.sh --help
```

Type `exit` to return to the host. Files in the lab folder persist.

Optional FashionMNIST download fallback

Keep the normal lab scripts as the default. Only if the official FashionMNIST download is slow or fails, run this from the TP01 folder *inside the activated container*. The host initialization above must already have created the `data/` links. This process tries the verified GitLab mirror first, retains the official mirrors as fallbacks, and checks the original MD5 values; it changes no course file. Existing complete data are reused. Ask the instructor about incomplete or corrupt existing data instead of overwriting them.

```
# container, TP01 folder; fallback only
python3 - <<'PY'
from torchvision import datasets as d
mirror = ('https://gitlab.telecom-paris.fr/api/v4/projects/10956/'
          'packages/generic/c5-datasets/2026.11.16/')
d.FashionMNIST.mirrors = [mirror] + d.FashionMNIST.mirrors
d.FashionMNIST('data', download=True)
PY
```

FashionMNIST is MIT-licensed, copyright 2017 Zalando SE; the package includes its license and attribution.

Optional CIFAR-10 download fallback

Keep the normal lab scripts and official source as the default. If that download is slow, run this inside the activated container after normal setup. It prepares the same persistent `C5_DATA` store, using torchvision's original MD5 check, extraction and cache logic:

```
# container; fallback only
python3 - <<'PY'
import os
from torchvision.datasets import CIFAR10
CIFAR10.url = ('https://gitlab.telecom-paris.fr/api/v4/projects/10956/'
               'packages/generic/c5-datasets/2026.11.16-cifar10/'
               'cifar-10-python.tar.gz')
CIFAR10(os.environ['C5_DATA'], download=True)
PY
```

The URL override lasts only for this process. Later lab commands retain the official source and reuse these files across TP01–TP03. If GitLab is unavailable, use the normal download instructions. Preserve incomplete/corrupt existing data for diagnosis. The mirror was authorized by the instructor; the original redistribution terms remain unresolved, and no new license is asserted.

The container, by hand (optional exploration)

The Vitis AI image of the course is one file, shared where installed or kept as an existing private copy (possibly in `/tmp/$USER`, see below); `VAI_SIF` holds its actual path. This manual route explains the launcher; it is not an additional setup step. From the lab folder in a host Bash shell:

```
# host
source scripts/00_source_env.sh     # sets VAI_SIF, the image of the course (and the PATH)
apptainer --version
ls -lh "$VAI_SIF"                    # missing? bash scripts/01_host_sanity.sh pulls it
mkdir -p "$C5_DATA"
apptainer exec --bind "$PWD:$PWD" --bind "$C5_DATA:$C5_DATA" --pwd "$PWD" "$VAI_SIF" bash
```

`apptainer exec <image> <command>` runs a command inside the image, here an interactive `bash`. `--bind "$PWD"` makes the lab folder visible at the same path inside the container and `--pwd "$PWD"` starts the shell there. The prompt changes: you are in the container. Its Python packages come from a conda environment, which you activate, check, and leave:

```
# container
source /opt/vitis_ai/conda/etc/profile.d/conda.sh
conda activate vitis-ai-pytorch
python3 -c "import torch, pytorch_nndct; print('torch', torch.__version__)"
exit
```

What the scripts do

- `00_source_env.sh` defines the folders of the lab and `VAI_SIF`, the path of the image, and prints them. It must be *sourced*, so that the variables stay in your shell. It reads `scripts/course_env.sh`, the same file in every lab of the course.
- `01_host_sanity.sh` checks the lab files, Apptainer, the image and the free disk space. Do not continue until it passes.
- `02_enter_container.sh` runs the `apptainer exec` command above, with `"$VAI_SIF"`. The shell activates `vitis-ai-pytorch` by itself (it runs `scripts/03_activate_vai_conda.sh`) and checks the packages. It is ready when it prints `torch: <version>`, `pytorch_nndct: OK` and then `Conda environment ready.` Type `exit` to leave it.

The image is chosen in this order, in every lab of the course:

1. the file named by `VAI_SIF`, if you exported it yourself;
2. IT's installed shared copy: `/comelec/softs/opt/vitis-ai-pytorch-cpu_ubuntu2004-3.5.0.306.sif`; the legacy shared location remains supported;
3. a private copy, `/tmp/$USER/vitis-ai-pytorch-cpu.sif`.

For an existing image in a temporary or other location, set `VAI_SIF` to its absolute path before sourcing `00_source_env.sh`; the same image can be opened manually with the command above. If the shared copy is missing on your machine, the sanity script pulls the private copy once, with the command below (several GB, 10 to 20 minutes). `/tmp` is local to one machine: on another machine, the pull happens again.

```
# host, after source scripts/00_source_env.sh (the sanity script does it for you)
mkdir -p /tmp/$USER
apptainer pull "$VAI_SIF" "$C5_VAI_PULL_URI"
```

In another shell inside the container, or if the prompt does not start with `(vitis-ai-pytorch)`, activate the environment yourself (with `source`, not `bash`):

```
# container
source scripts/03_activate_vai_conda.sh
```

Script	Where	Role
`00_source_env.sh`	host (`source`)	folders and image selection
`01_host_sanity.sh`	host	checks; pulls the image if it is missing
`02_enter_container.sh`	host	opens the container shell, environment active
`03_activate_vai_conda.sh`	container (`source`)	activates and checks `vitis-ai-pytorch`
`04_train_ptq.sh`	container	trains one model, exports its PTQ XMODEL, in `out/runs/<TAG>/`
`05_make_report.sh`	container	compares selected runs in `out/report/<name>/`
`clean.sh`	either	deletes `out/runs/` and/or `out/report/`

2 Tagged runs

One experiment is one *tag*, and one tag is one folder, `out/runs/<TAG>/`. Everything the run produces goes there, and later scripts (the report in this lab, compilation and deployment in the next labs) find a model only through its tag. A tag is a folder name: letters, digits, `.`, `_` or `-`. If you omit `--tag`, one is built from the settings, for example `fashionmnist_mlp_w1_e2_cal256`.

A tag that already exists is refused, so a finished run is never overwritten by mistake. To train it again on purpose, add `--overwrite`: the old folder is deleted first. Two runs that you want to compare must have two tags.

```
# container
bash scripts/04_train_ptq.sh --help
bash scripts/04_train_ptq.sh --dataset fashionmnist --arch mlp --width 1 --epochs 2 --tag
    first_mlp
ls -R out/runs/first_mlp
cat out/runs/first_mlp/meta.json
```

Option (default)	Meaning
`--dataset` (`fashionmnist`)	`mnist`, `fashionmnist` or `cifar10`
`--arch` (`mlp`)	`mlp`: one hidden layer of $64\,w$ units; `cnn`: two 3×3 convolutions ($8w$, $16w$ channels) with ReLU and max-pooling, then two fully connected layers ($64w$, 10)
`--width` (1)	the width multiplier w
`--epochs` (2)	passes over the training split
`--tag`, `--overwrite`	run folder name; replace an existing run
`--lr` (0.001), `--seed` (0)	Adam learning rate; seed of the initial weights, the shuffling and the train/validation split
`--calib_steps` (256)	training batches (64 images each) used to calibrate the quantizer
`--train_limit` (0)	train on the first N images of the training split only (0: all); for quick tests

What one run does

1. **Float training.** The official training set (60,000 images) is split once, with `--seed`: 90 % trains the model, 10 % (6,000 images) is the *validation* split. Each epoch appends a line to `float/train_log.csv`: the training loss and accuracy averaged over the batches of the epoch (the model changes during the epoch), and the validation loss and accuracy of the model at the end of the epoch.
2. **Float test.** The final model is saved (`float/model_float.pth`) and measured once on the official *test* set (10,000 images): `float/eval_float.json`.
3. **Post-training quantization (PTQ).** The Vitis AI quantizer observes the activation ranges on `--calib_steps` training batches (*calibration*, never on test images), converts the weights and activations to 8-bit integers, and exports `ptq/model_ptq.xmodel` after one forward pass with batch size 1, the batch size the DPU runs. The quantized model is then measured on the same test set: `ptq/ptq_metrics.json` gives its accuracy and the drop in points against the float model.

Lab 2 studies quantization; here, only check that the drop is small.

Train, validation, test

Decide with **validation** results: which width, how many epochs. The **test** accuracy is the final measurement of a model you have already chosen; if you used it to choose, it would no longer measure how the model behaves on new images. `meta.json` records the split sizes and the seed, so the same split can be rebuilt.

Questions 1–3

1. In `meta.json`, find the number of training, validation and test images, the number of parameters and the number of MACs (multiply-accumulate operations for one image) of `first_mlp`. Recompute the parameter count by hand (784 inputs, 64 hidden units, 10 outputs).
2. Run the same command again without `--overwrite`. What happens, and why is that the right behaviour?
3. Why does the script calibrate the quantizer on training images and not on test images?

3 Block A: model size at a fixed training budget

If every model gets the same number of epochs, what does a larger model buy? Unless CIFAR-10 is already there (shared copy or another lab), first start the download of the final task in a second

terminal, so that it runs during Blocks A and B. In that terminal, type `bash` and press Enter separately; wait for the Bash prompt before entering this host block. Adjust the checkout path if needed:

```
# host, second terminal after entering Bash separately
cd ~/c5_seti_students/TP01_vitis_ai_workflow
source scripts/00_source_env.sh
bash scripts/02_enter_container.sh
```

Wait for `Conda environment ready.`, then run this separate block inside the container (or use the optional GitLab fallback above if the official download is slow):

```
# container
python3 - <<'PY'
import os
from torchvision.datasets import CIFAR10
CIFAR10(os.environ['C5_DATA'], download=True)
PY
```

```
# container
bash scripts/04_train_ptq.sh --arch mlp --width 1 --epochs 2 --tag mlp_w1_e2
bash scripts/04_train_ptq.sh --arch mlp --width 2 --epochs 2 --tag mlp_w2_e2
bash scripts/04_train_ptq.sh --arch cnn --width 1 --epochs 2 --tag cnn_w1_e2
bash scripts/04_train_ptq.sh --arch cnn --width 2 --epochs 2 --tag cnn_w2_e2
bash scripts/05_make_report.sh --name block_a mlp_w1_e2 mlp_w2_e2 cnn_w1_e2 cnn_w2_e2
```

(`fashionmnist` is the default dataset.) The report prints a table and writes `out/report/block_a/`; Section 5 explains its content.

Questions 4–7

4. `mlp_w1_e2` and `cnn_w1_e2` have almost the same number of parameters. Compare their MACs and their validation accuracy. Why does the CNN need so many more operations per parameter?
5. Going from width 1 to width 2 doubles the MLP, but multiplies the CNN parameters by about 4. Which layer causes this?
6. Which model gives the best validation accuracy per parameter? Per MAC?
7. From which model on does the accuracy gain look too small for its cost? Answer with numbers.

4 Block B: training budget for one model

Keep the model fixed (`cnn`, width 2) and change only the number of epochs.

```
# container
bash scripts/04_train_ptq.sh --arch cnn --width 2 --epochs 1 --tag mid_cnn_e1
bash scripts/04_train_ptq.sh --arch cnn --width 2 --epochs 3 --tag mid_cnn_e3
bash scripts/04_train_ptq.sh --arch cnn --width 2 --epochs 5 --tag mid_cnn_e5
bash scripts/05_make_report.sh --name block_b 'mid_cnn_e*'
```

A quoted pattern selects every matching tag. The three runs use the same seed, hence the same initial weights and the same order of the training images: the first k epochs of `mid_cnn_e5` should repeat those of `mid_cnn_e`k. Check it in their `train_log.csv` files. In `compare.png` the three curves therefore overlap: the shorter runs are drawn on top of the longer ones, a diamond marks the last epoch of each run, and the bottom panel gives the final validation accuracy against the number of epochs.

Questions 8–10

8. After how many epochs does the validation accuracy stop improving noticeably?
9. Compare the train and validation curves of `mid_cnn_e5`. Is there a sign of overfitting?
10. Training cost is roughly proportional to the number of epochs. Which budget would you choose for an embedded product, where the model is trained once but must be retrained for each new dataset? Justify.

5 Reading the report

```
# container
bash scripts/05_make_report.sh --help
bash scripts/05_make_report.sh                    # every run, in out/report/all/
```

The report reads the run folders and never changes them. It needs, for each run, `meta.json` and `float/train_log.csv`; a run stopped during its first epoch is skipped with a message. A figure holds at most 12 runs. It writes:

- `summary.csv`: one row per run with the tag, dataset, architecture, width, epochs, parameters, MACs, the train and validation accuracy and loss after the last epoch, the float and PTQ test accuracy and the PTQ drop in points;
- `compare.png`: the accuracy and loss curves (train solid, validation dashed) and the final validation accuracy against the number of parameters (log scale).

If a number looks wrong, go back to the run folder: the report is only as reliable as what it reads.

```
# container
cat out/runs/<TAG>/meta.json
cat out/runs/<TAG>/float/train_log.csv
cat out/runs/<TAG>/ptq/ptq_metrics.json
```

To delete runs: `bash scripts/clean.sh --runs` (all runs), `--report` (all reports) or `--all`. Deleted runs cannot be recovered; to redo one run, use `--overwrite` instead.

6 Final task: CIFAR-10

Final task

Repeat a reduced study on CIFAR-10 (`--dataset cifar10`: 32×32 colour images, clearly harder than FashionMNIST): one smaller and one larger model, explicit tags, and a report of these runs only (`--name cifar ...`). CPU training is slower on CIFAR-10: keep 2–3 epochs, the same budget for both models.

- Do the conclusions of Blocks A and B still hold? Is the larger model justified from a frugal point of view?
- Compare the parameters and MACs of the CIFAR-10 models with the FashionMNIST ones. Which layer changes, and why?

Another `torchvision` dataset needs a new case in `get_dataset()` in `src/tp01_vai_ptq.py`; the CNN accepts 28×28 and 32×32 inputs (add a `transforms.Resize` otherwise).

7 Extension: CPU latency (code to write)

Accuracy is not enough for embedded AI: latency, memory and energy count too, and the next labs measure them on the KV260. As a first step, add the CPU inference time of each float model to the report: for example a small script that loads `float/model_float.pth` (rebuild the model with `build_model` from `src/model.py` and the settings in `meta.json`), times it, and writes the result into the run folder, and a new column and figure (accuracy against latency) in the report.

Write down your protocol: `torch.no_grad()` and `model.eval()`, a warm-up before timing, a fixed batch size (1, as on the DPU, and a larger one), many repetitions, and the median rather than the mean. Does the latency follow the number of parameters or the number of MACs?