Kempner INSTITUTE

HARVARD UNIVERSITY

Optimizing ML Workflows on an AI Cluster

May 6, 2025

Objectives

By the end of this workshop, you will be able to:

- How to use Weights & Biases to log experiments, perform hyperparameter sweeps, and perform model comparisons
- How to implement effective model checkpointing practices, including resuming from a checkpoint if a job fails

Agenda

1 Introduction

2 Getting set up

3 Data loading & processing

4 Experiment tracking

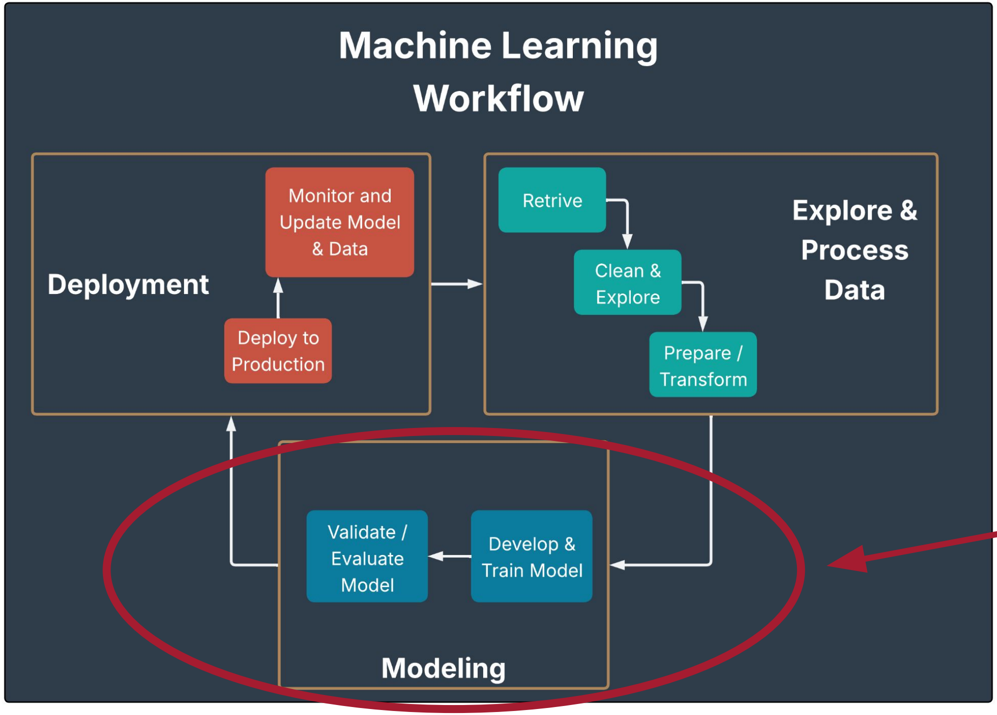
5 Hyperparameter sweeps & evaluation

6 Checkpointing & resuming

7 Distributed training

8 And beyond!

Machine Learning Workflow



We'll especially focus on this in the context of research needs and using an AI cluster

Modified from https://developer.nvidia.com/blog/machine-learning-in-practice-ml-workflows/

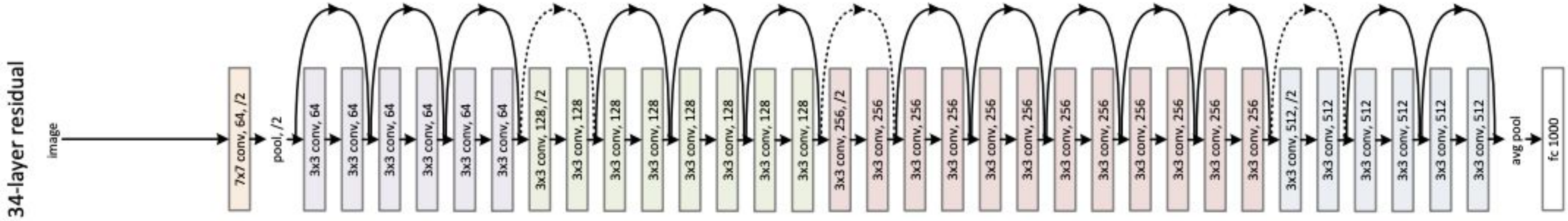
So much to think about…

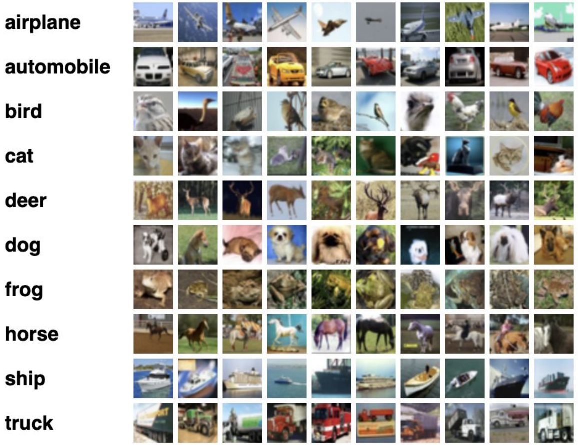
- **Data Management:** local or remote data, download needed, authentication, raw or preprocessed
- **Filesystem & Storage:** local disk, network-mounted, parallel filesystem, I/O throughput
- **Training Configuration:** batch size, optimizer, learning rate, epochs, loss function
- **Monitoring:** convergence metric, early stopping, training/val loss curves, logging tools
- **Failure Handling:** failure recovery, logs and metrics for debugging, checkpoint frequency - save all or best model
- **Memory Usage:** mixed precision, gradient accumulation, optimized dataloader
- **Scalability:** multi-GPU or multi-node, DDP/FSDP strategy
- **Hyperparameter optimization:** model size, learning rate, batch size, etc
- **ML Frameworks:** PyTorch, TensorFlow, JAX, Huggingface, Lightning, Keras
- **Environment Management:** conda, pip, Docker, Podman, Singularity, orchestration tools
- ….

Common Pitfalls for ML Researchers

- Difficult to keep track of all the different models you're training
- Hidden configurations (e.g. hyperparameters are scattered across scripts or hardcoded, so difficult to reproduce results)
- Unreproducible results (e.g. training is different on different machines/environments)
- Wasted or inefficient compute use (e.g. job ends before training is complete and lose progress, requesting two GPUs but only actually using one)
- …

Our running example: ResNet-50 on CIFAR-10

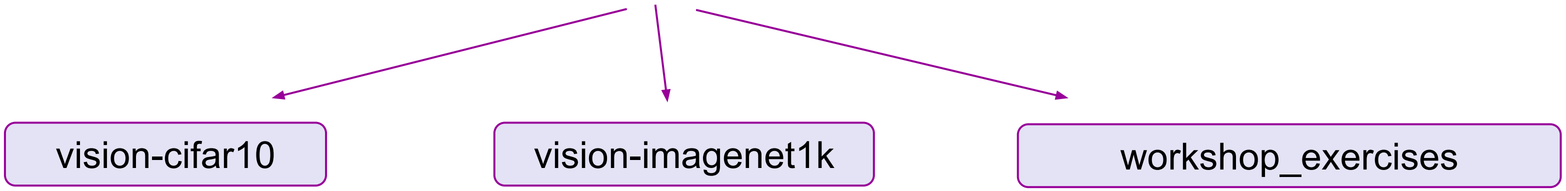




ResNet50 from He et al, 2015; CIFAR-10 image from https://www.cs.toronto.edu/~kriz/cifar.html

Examples & Exercises

Git repo for workshop: https://github.com/KempnerInstitute/optimizing-ml-workflow



vision-cifar10

vision-imagenet1k

workshop_exercises

Exercise: Look through base training code

We'll start with a base training code that we'll iteratively add functionality to.

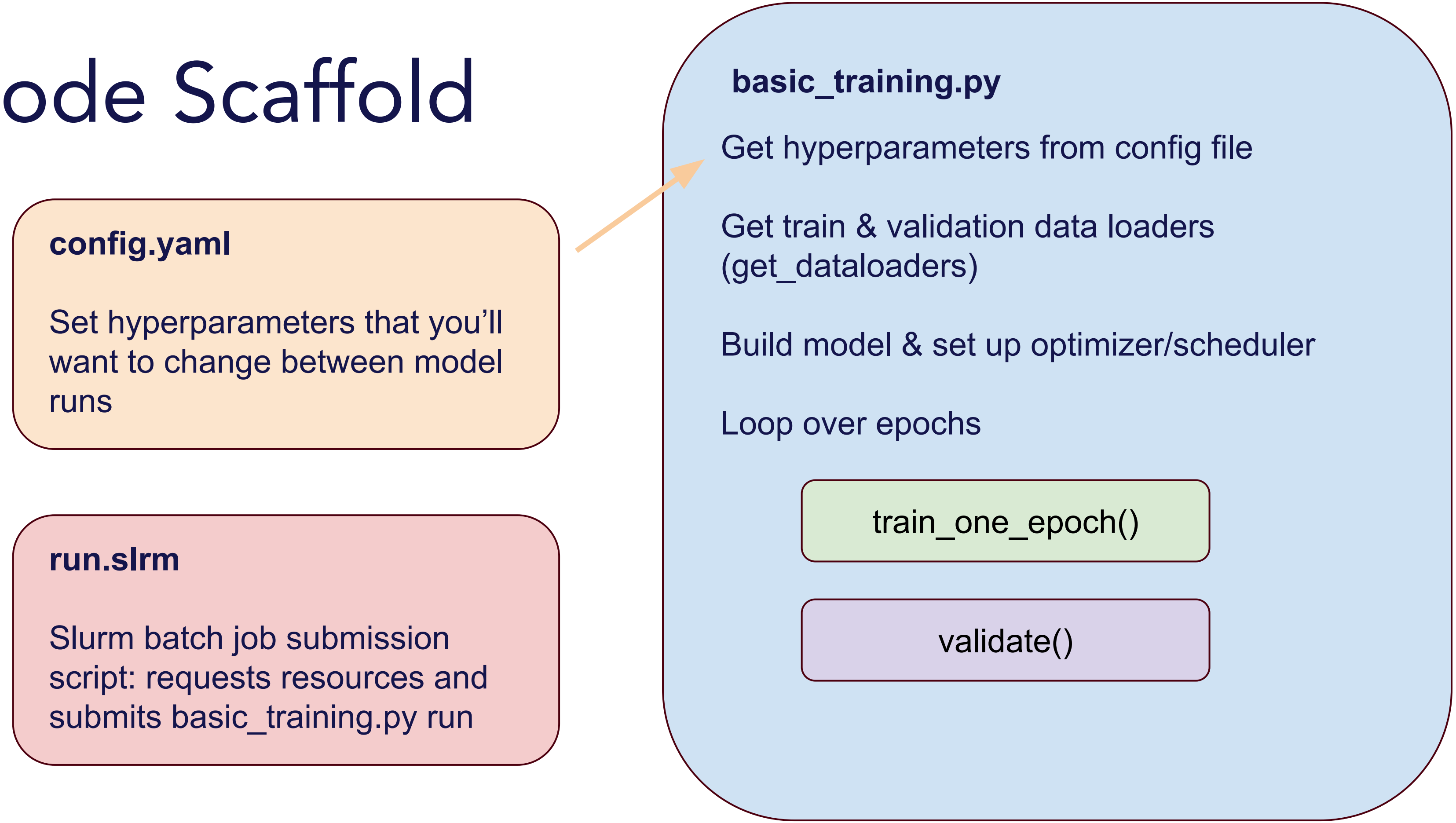
1) Familiarize yourself with the files in https://github.com/KempnerInstitute/optimizing-ml-workflow/tree/main/workshop_exercises/basic_training/

2) What are the key functions in basic_training.py and what do they do? Where is the training happening?

3) How can you change hyperparameters of the code such as learning rate?

4) Any questions on the code?

Code Scaffold



config.yaml

Set hyperparameters that you'll want to change between model runs

run.slrm

Slurm batch job submission script: requests resources and submits basic_training.py run

basic_training.py

Get hyperparameters from config file

Get train & validation data loaders (get_dataloaders)

Build model & set up optimizer/scheduler

Loop over epochs

train_one_epoch()

validate()

Benefits of YAML-based Configuration

What do you think some benefits of having hyperparameters in a YAML-based configuration are?

Why use YAML-based configuration?

- Separation of code and parameters: keeps training logic clean and reusable
- Easier experimentation: change aspects of model training without editing code
- Improved reproducibility: one file captures everything about the run, you can log that file without saving the entire code repo
- Sweep-friendly: Compatible with tools like W&B for hyperparameter tuning

What should be in the config?

- Model hyperparameters (i.e. number of layers)
- Dataset paths, splits, transforms
- Training options (i.e. batch_size)
- Checkpointing settings
- Logging and monitoring settings
- Anything!

The more you represent different experiment settings in the configuration file - rather than hardcoding them - the better!

Agenda

1 Introduction

2 **Getting set up**

3 Data loading & processing

4 Experiment tracking

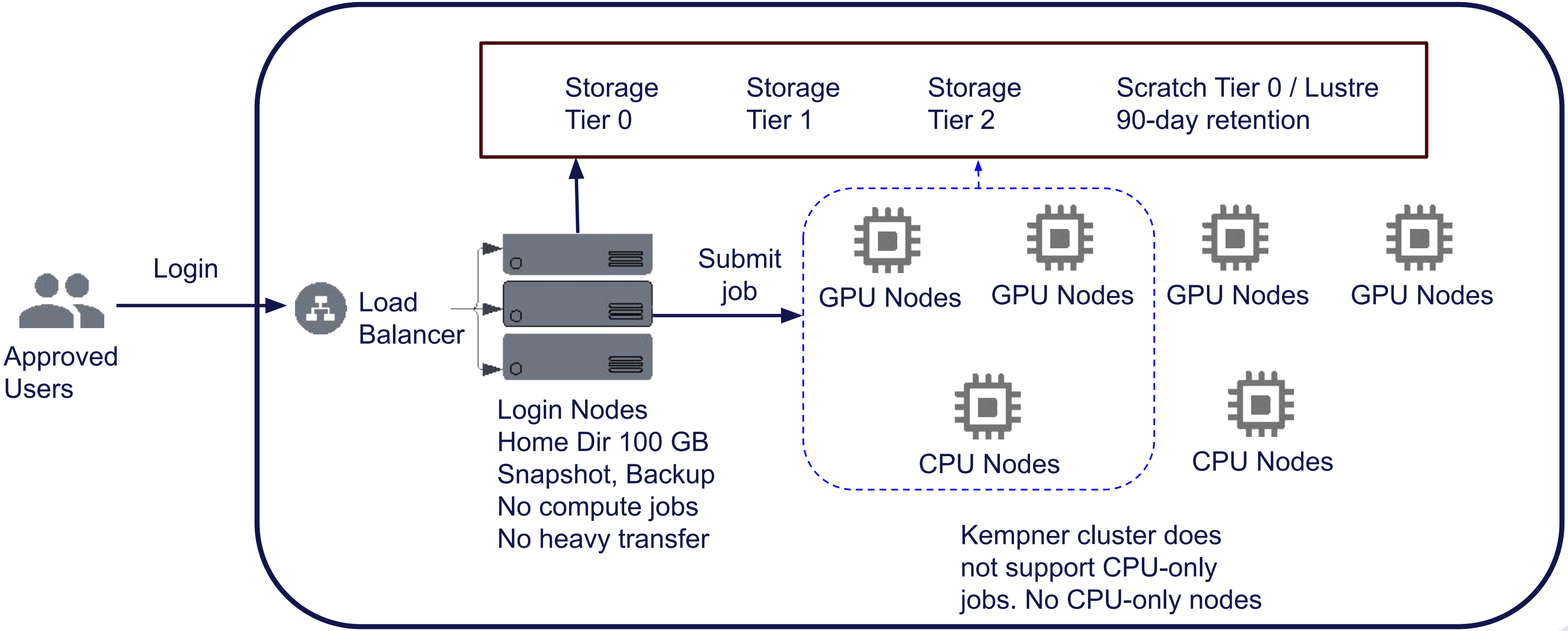
5 Hyperparameter sweeps & evaluation

6 Checkpointing & resuming

7 Distributed training

8 And beyond!

Know your cluster



Accessing the cluster

ssh <username>@login.rc.harvard.edu

Open OnDemand

https://handbook.eng.kempnerinstitute.harvard.edu/s1_high_performance_computing/kempner_cluster/accessing_and_navigating_the_cluster.html

Job submission

```
#sbatch -p gpu_requeue, (gpu_test or gpu)

#sbatch -A <your-account>

#sbatch –reservation mlopt_workshop
```

Where to Store Code & Data

Home Directory

- Quota: 100 GB per user
- FileSystem Type: NFS
- Use Cases: Save a copy of your code (not big dataset or runtime assets

Lab Directory

- Quota: 4 TB (default), PI can pay for more storage.
- FileSystem Type: Lustre (holylfsxxx, holylabs) or Isilon (/n/{PATH})
- Use Cases: Luster: High speed I/O, distributed data processing; Isilon: offer backup and snapshots, high performance (< Lustre)

Scratch Space

- Path: /n/netscratch/
- Quota: 50 TB per lab
- FileSystem Type: VAST
- Use Cases: Data processing, AI workflows, High performance and throughput

What is FASRC data retention policy on netscratch?

Software management - Package Managers

Tool	Category	Focus Ecosystem	Built For
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
Poetry	Python package/project manager	Pure Python	Python developers
UV	Fast Python package manager	Pure Python	Speed and modern workflows

How do you share your computational environment with others?

Read more about Spack, Conda on Kempner Computing Handbook

Environment/Software Management - Containers

Aspect	Conda / Mamba	Singularity / Docker
System Libraries	Limited; depends on host environment	Included in container; full control
Rep		
Roo		
HPC Compatibility	Good	Singularity: Excellent, Docker: Often blocked
MPI / Native Code Support	Limited	Strong; full stack packaging possible
Use Case Suitability	Lightweight, Python/R environments	Full-stack scientific/AI workflows

Can you think of a use case where Singularity could resolve the computational environment issues in your project?

Read more about Singularity on Kempner Computing Handbook

Singularity exercise: https://github.com/KempnerInstitute/optimizing-ml-workflow/tree/main/singularity_build

Agenda

1 Introduction

2 Getting set up

3 Data loading & processing

4 Experiment tracking

5 Hyperparameter sweeps & evaluation

6 Checkpointing & resuming

7 Distributed training

8 And beyond!

Data Loader Parameters

```
DataLoader('dataset': dataset,
           'batch_size': batch_size,
           'shuffle': True,
           'num_workers': num_workers,
           'pin_memory': pin_memory,
           'prefetch_factor': prefetch_factor
)
```

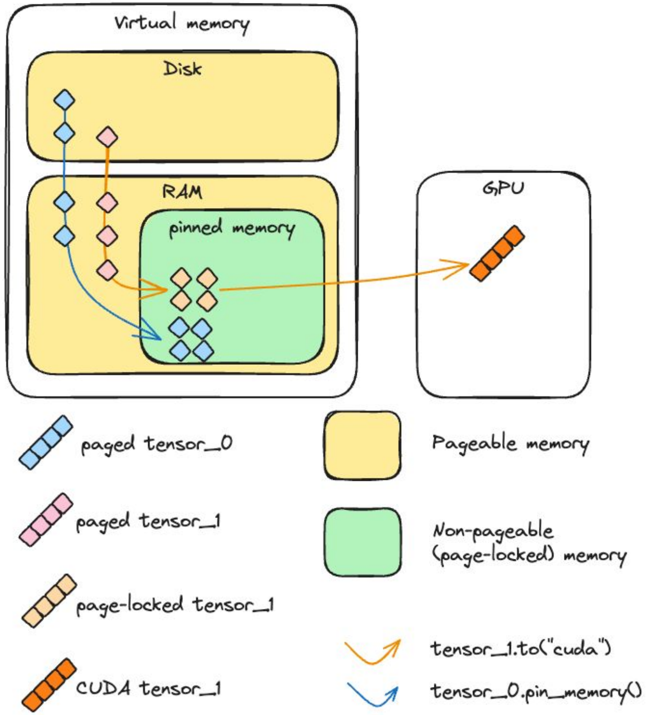


Image credit: https://pytorch.org/tutorials/intermediate/pinmem_nonblock.html

Exercise: Try out DataLoader parameters in Colab

https://colab.research.google.com/drive/1UCWz2ceLJcfZGWrMtHuF_2gpmoH-Qqwb?usp=sharing

Data Pre-Processing on GPUs

DALI

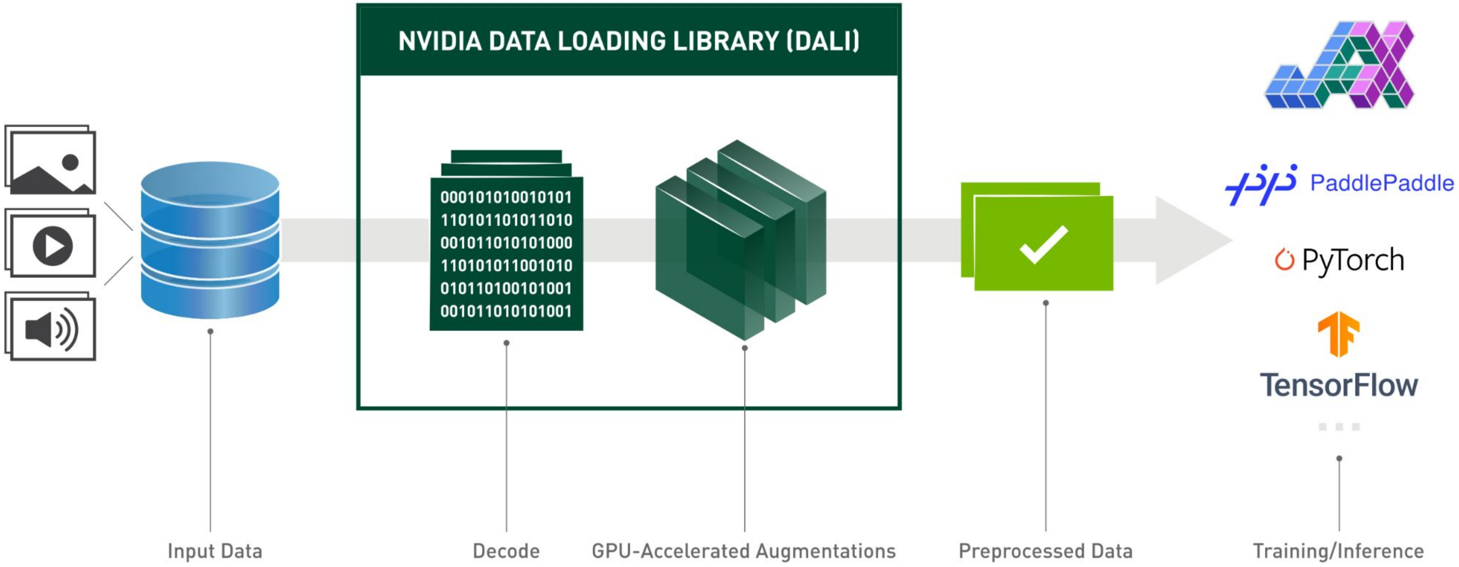


Image credit: https://github.com/NVIDIA/DALI

Agenda

1 Introduction

2 Getting set up

3 Data loading & processing

4 Experiment tracking

5 Hyperparameter sweeps & evaluation

6 Checkpointing & resuming

7 Distributed training

8 And beyond!

Why Track Experiments?

Ok I’ve found my best model of the million I tried. What hyperparameters did it use?

model_3layers_codev2_newdata_final.pth

I’ve made a huge mistake

Careful experiment tracking (tracking not only hyperparameters/configs but also code, data, metrics) saves your own time and makes your work reproducible

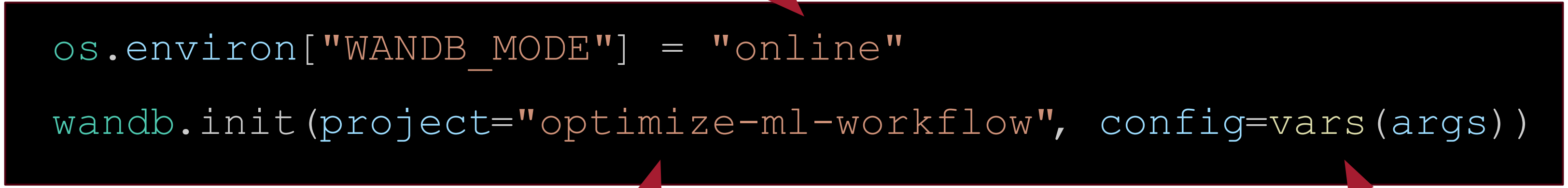
Weights and Biases (wandb)

- Widely used tool which is free for academic researchers
- Integrates easily with most machine learning frameworks, including PyTorch
- Relatively easy-to-use and has amazing web interface
- Helps with:
    - Experiment tracking
    - Visualizations
    - Hyperparameter sweeps
    - Dataset and media logging
    - Model monitoring
    - Model checkpointing
    - Collaboration and sharing

Add wandb integration to your code in a few lines

online = syncs to server/web interface in real time (requires internet)
offline = logs everything locally, can sync later

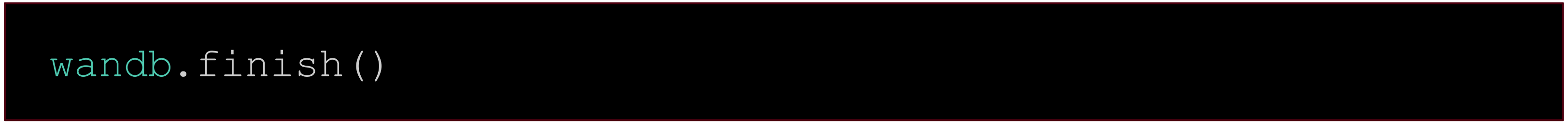
Could also define name of run, otherwise it is randomly generated



```
os.environ["WANDB_MODE"] = "online"

wandb.init(project="optimize-ml-workflow", config=vars(args))
```

Project name to organize experiments under

Dictionary of hyperparameters defining the configuration of the run (from argparse)



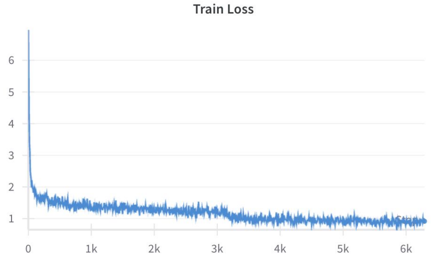
```
wandb.finish()
```

Logging Scalar Metrics

- Use wandb.log() to record scalar values like loss, accuracy, or learning rate from scheduler

```
wandb.log({"Scalar Name 1": value1, "Scalar Name 2": value2})
```

- W&B will automatically generate line plots for each logged key with step number on the x-axis, will update the plot each time the scalar is logged



Logging More Than Scalars

W&B Data Type	Example Use	Example Code
Images & custom plots	Log a few example images with labels	`wandb.log({'example_image': wandb.Image(img, caption=f'Label: {label}')})`
Confusion matrix	Show classification accuracy across classes on validation data, see how this changes over training	`wandb.log({'val_confusion_matrix': wandb.plot.confusion_matrix(y_true=val_labels, preds=val_preds, class_names=class_names)})`
Text	Log predicted vs true class	`wandb.log({'text_prediction': wandb.Html(f'<b>Pred:</b> {pred}<br><b>True:</b> {true}')})`

Table	Log all test images, predicted label, and actual label on every epoch	`table = wandb.Table(columns=['Image', 'Predicted', 'Actual'])`
		`...add_data...`
		`wandb.log({'text_predictions': table})`

Exercise: Try out Weights & Biases

Find the relevant files in workshop_exercises/wandb_tracking

1) Log in to weights and biases on the command line:
   a) Get API key from https://wandb.ai/authorize
   b) Run this on command line of HPC cluster:

      wandb login YOUR_API_KEY_HERE
2) Find where we've added initialization and finish to the code scaffold (add_wandb_exercise.py)
3) Add logging of training loss, validation loss, and validation accuracy one per epoch
4) Submit job
5) Explore weights and biases interface as the job is running

Advanced Challenges:

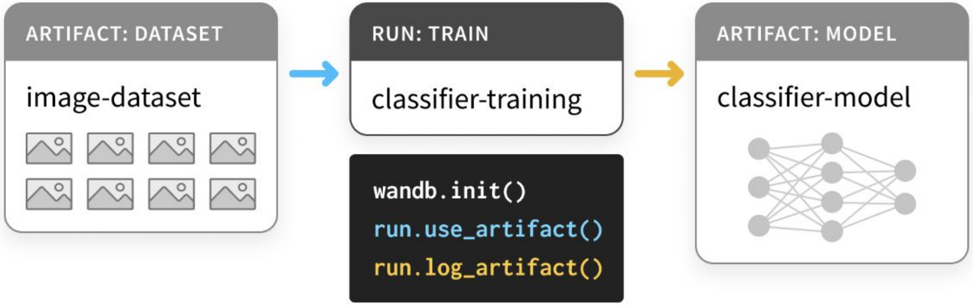
1) Try logging something other than a scalar metric
2) Submit advanced solutions script (has other logged data types) & explore interface

Live Demo of W&B Interface

Sample Project: https://wandb.ai/anmolmann/pytorch-cnn-fashion

Artifacts

- Artifacts are versioned files or directories stored in W&B
- Could be model checkpoints, config files, datasets, outputs, etc
- Can track lineage of artifact:
  - Run A created artifact model:v0
  - Run C used artifact model:v0 for fine-tuning
- Artifacts can be downloaded and re-used
- Helps with reproducibility!



```
wandb.init()
run.use_artifact()
run.log_artifact()
```

https://docs.wandb.ai/tutorials/artifacts/

Agenda

1 Introduction

2 Getting set up

3 Data loading & processing

4 Experiment tracking

5 **Hyperparameter sweeps & evaluation**

6 Checkpointing & resuming

7 Distributed training

8 And beyond!

Hyperparameter sweeps

Hyperparameter sweeps: systematically explores combinations of training parameters (e.g., learning rate, batch size, optimizer) to find configurations that improve model performance

Why sweep hyperparameters?

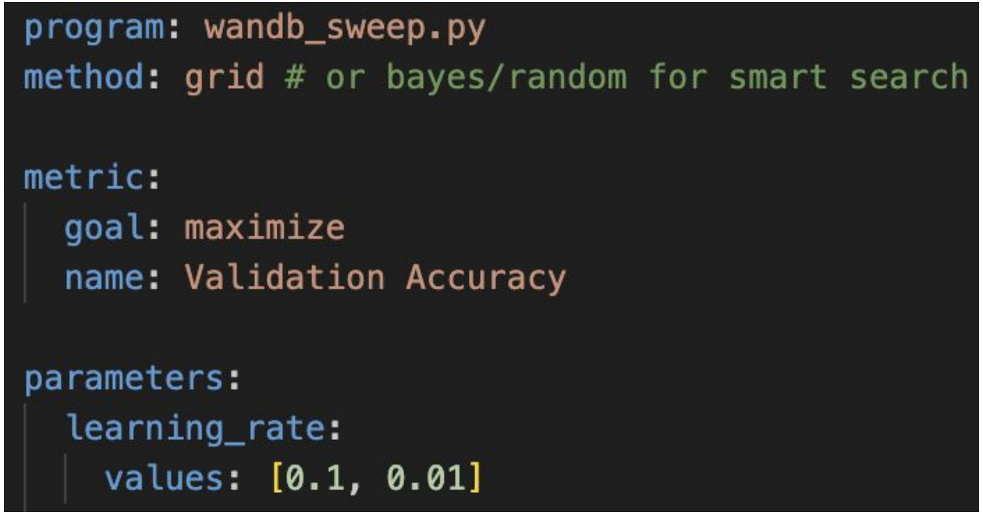
- Small changes can have big impacts on results
- In research, you often want to understand how hyperparameters affect training, or try models on multiple datasets, or multiple models on one dataset
- Automated sweeping saves a lot of time

Luckily, weights & Biases is an excellent tool for hyperparameter sweeps!

Weights & Biases Sweeps Step 1

Step 1) Define the sweep

- Create a sweep_config.yaml file
- Pick a search method: grid, random, or Bayesian
- Specify hyperparameters with multiple values to cycle through in sweep_config.yaml
- We will still use config.yaml for hyperparameter we are not searching over (sweep_config will overwrite config.yaml)



```yaml
program: wandb_sweep.py
method: grid # or bayes/random for smart search

metric:
  goal: maximize
  name: Validation Accuracy

parameters:
  learning_rate:
    values: [0.1, 0.01]
```

Weights & Biases Sweeps

Step 2) Initialize the sweep

W&B Sweep Controllers manage sweeps.

Initialize sweep using command line argument (from within mamba env):

```
wandb sweep --project project_name sweep_config.yaml
```

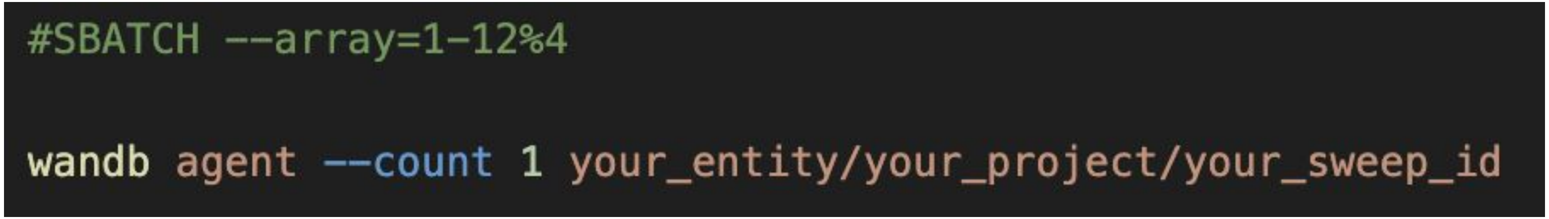
After initializing the sweep, we get a sweep id which uniquely identifies the sweep.

Weights & Biases Sweeps

Step 3) Run sweep agents

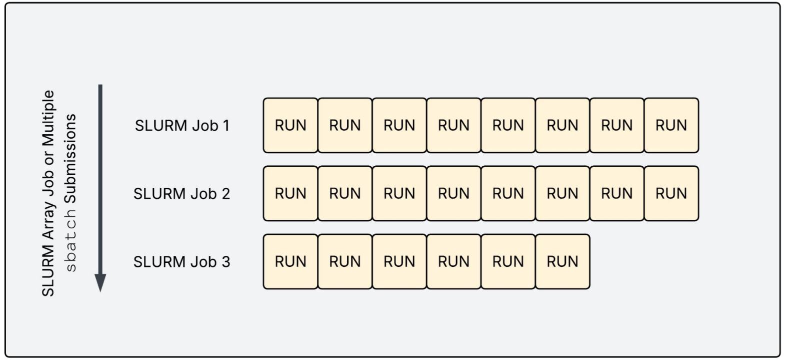
Sweep agents are responsible for running an experiment with a set of hyperparameter values that you defined in your sweep configuration.

Several ways to do this - we use array jobs to parallelize hyperparameter runs over slurm jobs



```
#SBATCH --array=1-12%4

wandb agent --count 1 your_entity/your_project/your_sweep_id
```

W&B Sweep with Slurm

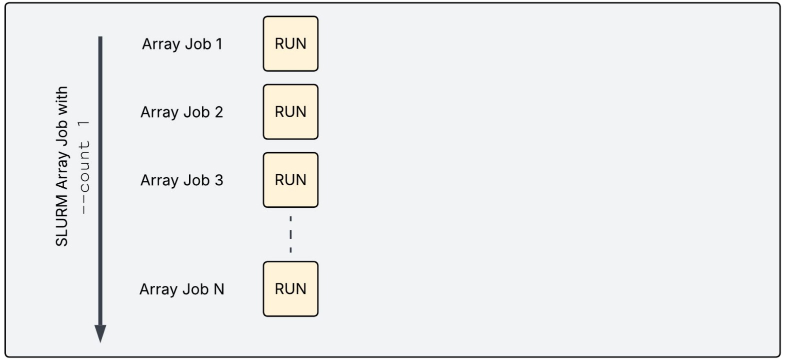


SLURM Array Job or Multiple `sbatch` Submissions

SLURM Job 1 RUN RUN RUN RUN RUN RUN RUN RUN

SLURM Job 2 RUN RUN RUN RUN RUN RUN RUN RUN

SLURM Job 3 RUN RUN RUN RUN RUN RUN



SLURM Array Job with `--count 1`

Array Job 1 RUN

Array Job 2 RUN

Array Job 3 RUN

Array Job N RUN

Exercise: Deploying sweep

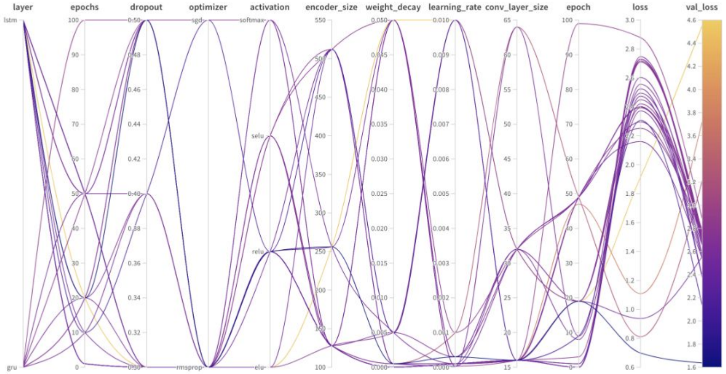
Find the relevant files in workshop_exercises/wandb_sweep

wandb_sweep is the same code we've been working with

1) Update sweep_config.yaml to grid search over:
   a) Learning rates of 0.01, 0.001
   b) Batch size of 32, 64, and 128
   c) Finetuning or not (true and false in yaml)
2) Update config.yaml so each run trains for only 3 epochs
3) Change so array job runs 4 jobs at once
4) Initialize sweep
5) Submit slurm job. Check how many jobs are running at once. How many runs should there be in total?
6) Look at sweep interface

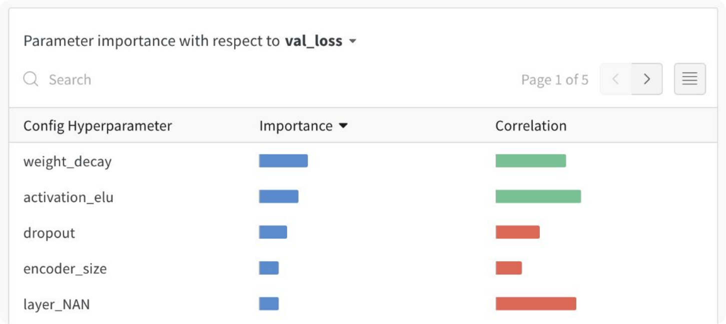
Sweep Visualizations

Parallel Coordinates Plot



Maps hyperparameter values to model metrics

Hyperparameter Importance Plot



Parameter importance with respect to **val_loss** ▾

Search

Page 1 of 5

Config Hyperparameter	Importance ▾	Correlation
weight_decay	🟦	🟩
activation_elu	🟦	🟩
dropout	🟦	🟥
encoder_size	🟦	🟥
layer_NAN	🟦	🟥

Shows which hyperparameters were best predictors of metrics

Correlation: linear correlation between hyperparameter and chosen metric

Feature importance: trains random forest with hyperparameters as inputs and metric as target output, reports feature importance values from this

Live Demo of W&B Interface

Sample Sweeps Project:
https://wandb.ai/anmolmann/pytorch-cnn-fashion/sweeps/pmqye6u3?nw=default

Agenda

1 Introduction

2 Getting set up

3 Data loading & processing

4 Experiment tracking

5 Hyperparameter sweeps & evaluation

6 Checkpointing & resuming

7 Distributed training

8 And beyond!

What is checkpointing and why do it?

Checkpointing: Saving the current state of a training process so that training can be resumed later without starting over

Have you encountered any of these errors in your ML workflow?

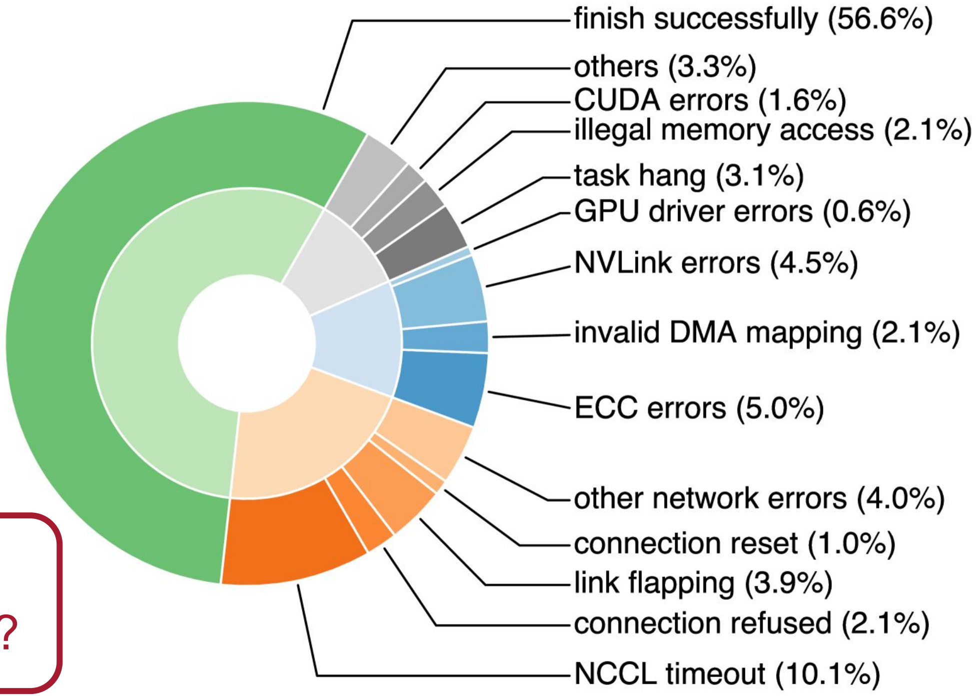
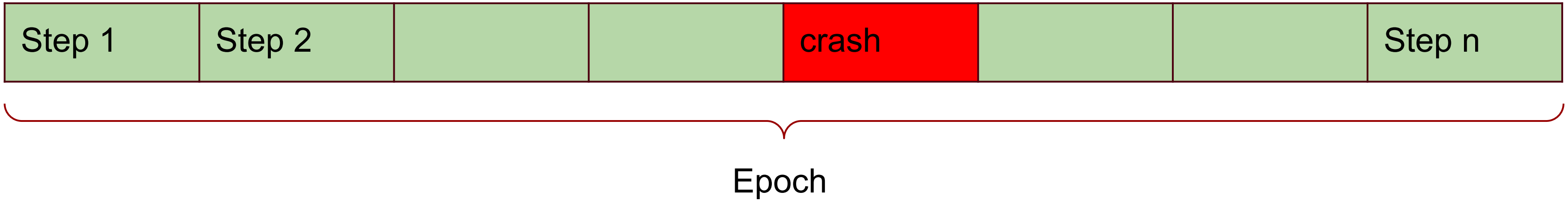


Image from He et al 2023, https://arxiv.org/html/2401.00134v1

Failure handling - checkpoint



Step 1	Step 2			crash			Step n

Epoch

As part of checkpoint, save biases, gradients, optimizer states, epoch number, sometimes scheduler

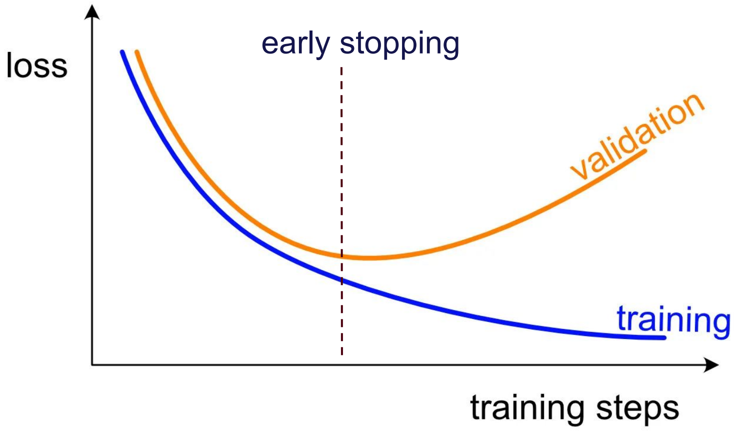
Should we checkpoint after each epoch or each step?
Save the best models or just the latest?

Failure handling - checkpoint

What to checkpoint: Model weights and biases, optimizer state, gradients, epoch # and scheduler.

When to checkpoint: Every epoch for most cases; every step for fast or unstable training.

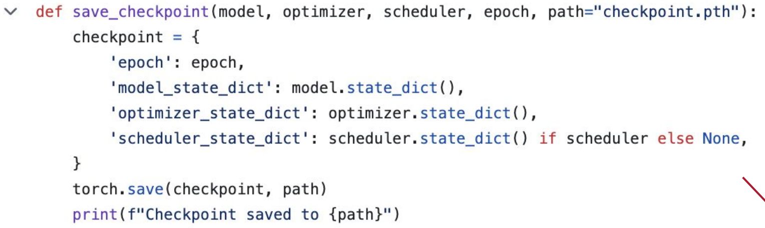
Last vs. best checkpoint: Use the last for resuming; best for evaluation or deployment.



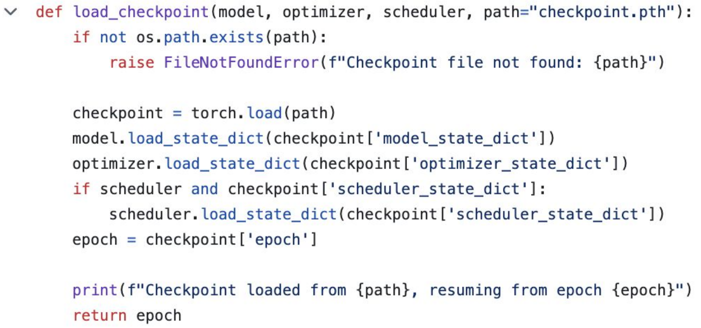
https://medium.com/@damian0815/fine-tuning-stable-diffusionwith-validation-3fe1395ab8c3

Adding checkpointing to our code

Find the relevant files in workshop_exercises/checkpointing

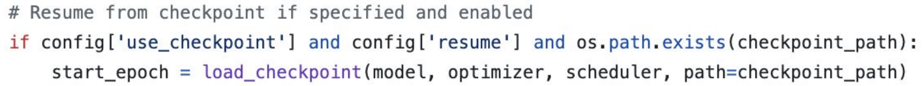


```python
def save_checkpoint(model, optimizer, scheduler, epoch, path="checkpoint.pth"):
    checkpoint = {
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'scheduler_state_dict': scheduler.state_dict() if scheduler else None,
    }
    torch.save(checkpoint, path)
    print(f"Checkpoint saved to {path}")
```

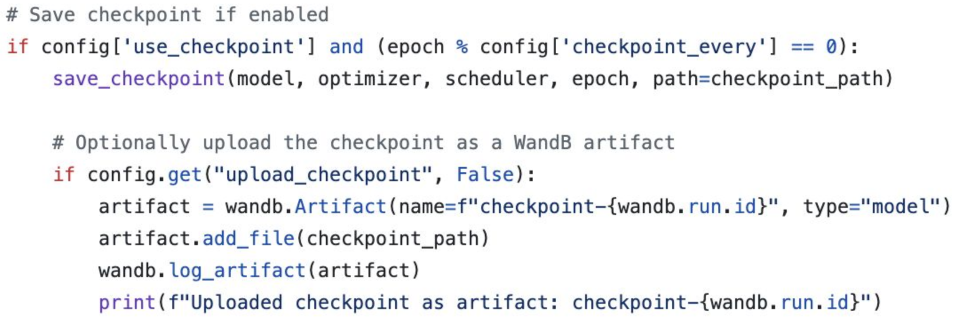


```python
def load_checkpoint(model, optimizer, scheduler, path="checkpoint.pth"):
    if not os.path.exists(path):
        raise FileNotFoundError(f"Checkpoint file not found: {path}")

    checkpoint = torch.load(path)
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    if scheduler and checkpoint['scheduler_state_dict']:
        scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
    epoch = checkpoint['epoch']

    print(f"Checkpoint loaded from {path}, resuming from epoch {epoch}")
    return epoch
```

Calls in main script



```python
# Resume from checkpoint if specified and enabled
if config['use_checkpoint'] and config['resume'] and os.path.exists(checkpoint_path):
    start_epoch = load_checkpoint(model, optimizer, scheduler, path=checkpoint_path)
```



```python
# Save checkpoint if enabled
if config['use_checkpoint'] and (epoch % config['checkpoint_every'] == 0):
    save_checkpoint(model, optimizer, scheduler, epoch, path=checkpoint_path)

    # Optionally upload the checkpoint as a WandB artifact
    if config.get("upload_checkpoint", False):
        artifact = wandb.Artifact(name=f"checkpoint-{wandb.run.id}", type="model")
        artifact.add_file(checkpoint_path)
        wandb.log_artifact(artifact)
        print(f"Uploaded checkpoint as artifact: checkpoint-{wandb.run.id}")
```

Exercise: Try out resuming from checkpoint

Find the relevant files in workshop_exercises/checkpointing

1) Update config.yaml to simulate a crash and resume from checkpoint:
   a) Enable checkpointing
   b) Enable resuming from checkpoint
   c) Set checkpointing frequency
   d) [Optional] Upload checkpoints to W&B
2) Update config.yaml so each run trains for only 10 epochs (batch 64)
3) Initialize sweep
4) Monitor simulated crash by looking into job *.out or *.err files
5) Resume training from checkpoints
6) Locate local checkpoint files inside wandb folder

Agenda

1 Introduction

2 Getting set up

3 Data loading & processing

4 Experiment tracking

5 Hyperparameter sweeps & evaluation

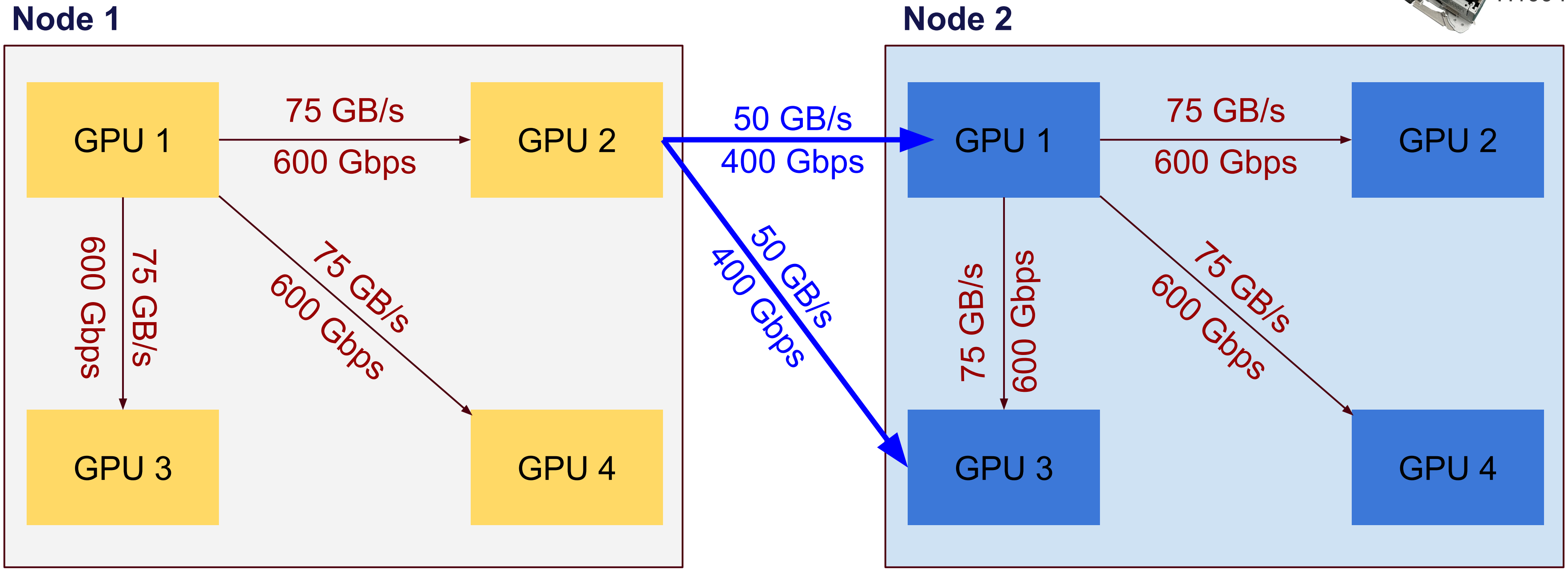
6 Checkpointing & resuming

7 **Distributed training**

8 And beyond!

GPU-to-GPU Communication

H100 Node

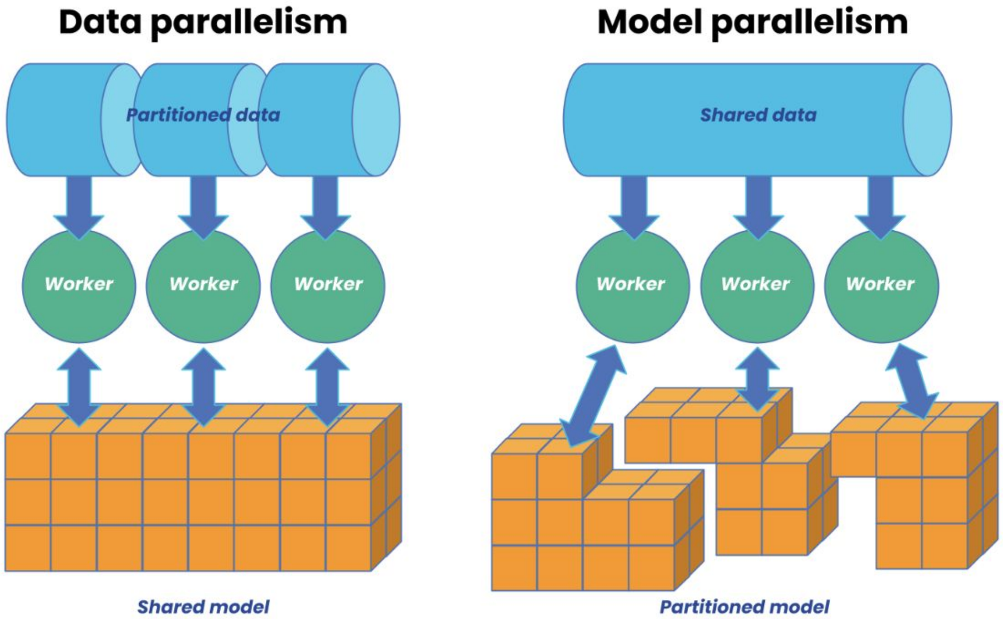


<u>Inside Node (**NVLINK**)</u>: Each GPU talks to other three GPUs at 75 GB/s (single direction). This sums up to 900 GB/s all GPU-GPU bidirectional speed. 75 GB/s * 6 * 2 = 900 GB/s

<u>Outside Node (**InfiniBand Network NDR**)</u>: Each GPU communicates to other GPUs in another node at 400 Gbps (50 GB/s).

Distributed Training

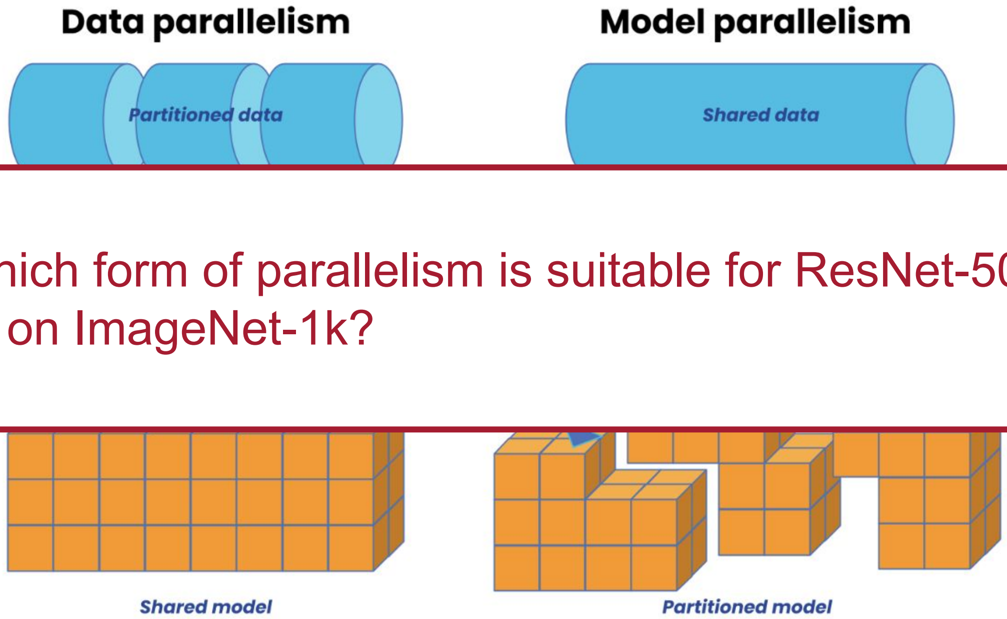
- Data Parallel
- Model Parallel
- Pipeline Parallel
- Fully Sharded Data Parallel (FSDP)
- Hybrid Parallel
- and more



Credit: https://www.anyscale.com/blog/what-is-distributed-training

Distributed Training

- Data Parallel
- Model Parallel
- Pipeline Parallel



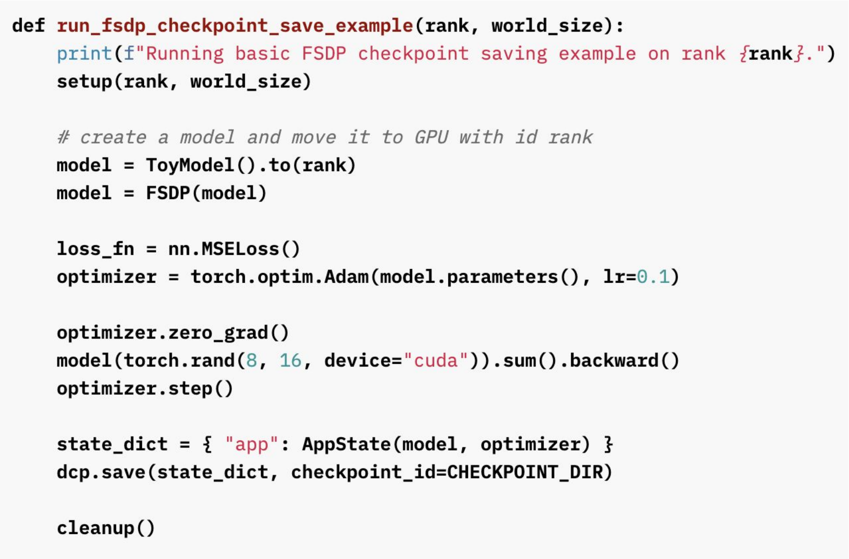
Consider H100 80GB GPU, which form of parallelism is suitable for ResNet-50 on ImageNet-1k?

Credit: https://www.anyscale.com/blog/what-is-distributed-training

PyTorch Distributed Checkpointing

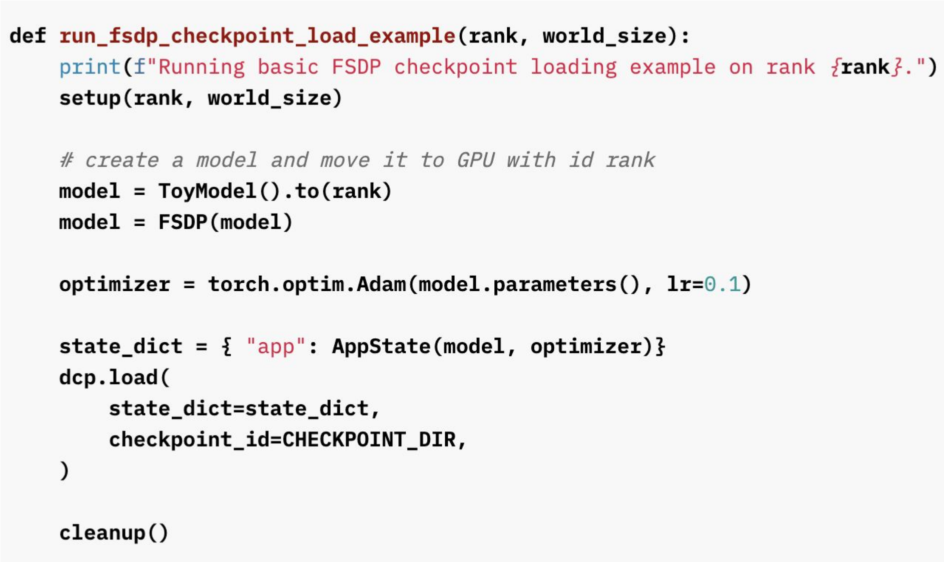
```
import torch.distributed.checkpoint as dcp
```

Save Checkpoint with DCP



```
def run_fsdp_checkpoint_save_example(rank, world_size):
    print(f"Running basic FSDP checkpoint saving example on rank {rank}.")
    setup(rank, world_size)

    # create a model and move it to GPU with id rank
    model = ToyModel().to(rank)
    model = FSDP(model)

    loss_fn = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

    optimizer.zero_grad()
    model(torch.rand(8, 16, device="cuda")).sum().backward()
    optimizer.step()

    state_dict = { "app": AppState(model, optimizer) }
    dcp.save(state_dict, checkpoint_id=CHECKPOINT_DIR)

    cleanup()
```

Load Checkpoint with DCP



```
def run_fsdp_checkpoint_load_example(rank, world_size):
    print(f"Running basic FSDP checkpoint loading example on rank {rank}.")
    setup(rank, world_size)

    # create a model and move it to GPU with id rank
    model = ToyModel().to(rank)
    model = FSDP(model)

    optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

    state_dict = { "app": AppState(model, optimizer)}
    dcp.load(
        state_dict=state_dict,
        checkpoint_id=CHECKPOINT_DIR,
    )

    cleanup()
```

Check out full example here: https://pytorch.org/tutorials/recipes/distributed_checkpoint_recipe.html

Agenda

1 Introduction

2 Getting set up

3 Data loading & processing

4 Experiment tracking

5 Hyperparameter sweeps & evaluation

6 Checkpointing & resuming

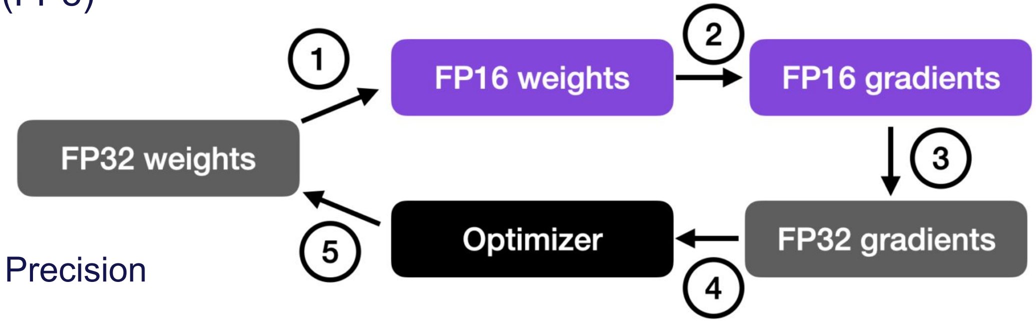
7 Distributed training

8 And beyond!

Compute Resources Optimization

- **Mixed Precision Training**: Save on GPU memory and improve speed
- **Same Node Allocation**: Faster intra-node communication vs inter-node
- **Controlled Checkpointing**: Save on storage and compute
- **Distributed Data Access:** Right storage selection and settings for faster multi-worker access
- **Inference**: Try lower precision inference (FP8)

Lots of extra features, including mixed precision, in the examples in the rest of the repo!



Mixed Precision

https://lightning.ai/pages/community/tutorial/accelerating-large-language-models-with-mixed-precision-techniques/

Monitoring Tools/Frameworks

Tool	GPU Monitoring	Low-Level Profiling	Training Metrics	Best For
Weights & Biases	Yes	No	Yes	Team tracking, PyTorch & TF workflows
PyTorch Profiler	Yes	Yes	Partial	PyTorch performance tuning
NVIDIA Nsight	Yes (deep)	Yes (low-level)	Limited	Kernel-level GPU profiling
MLflow	Limited	No	Yes	Model lifecycle management
TensorBoard	Limited	No	Yes	TensorFlow users

Read more about Nvidia Nsight Tools and PyTorch Profiler on Kempner Computing Handbook

Kempner
INSTITUTE

VE RI TAS
HARVARD
UNIVERSITY

Thank you