



Leibniz Supercomputing Centre
of the Bavarian Academy of Sciences and Humanities

AI Training Series

Introduction to the LRZ AI Systems

05.11.2024 | Ajay Navilarekal, Darshan Thummar

1. Introduction to the LRZ AI Systems

.....

- ❑ Overview of the LRZ AI Systems
- ❑ Access to the LRZ AI Systems
- ❑ NVIDIA NGC Cloud
- ❑ Introduction to Enroot Containers
- ❑ Interactive and Batch Jobs
- ❑ Open on Demand
- ❑ Exercise: Run a job and extend an Enroot container

2. Data Distributed Training

.....

- ❑ Introduction to Convolutional Neural Networks
- ❑ Exercise: Train VGG-19 on a GPU
- ❑ Introduction to Distributed Training
- ❑ Exercise: Train VGG-199 on 2 GPUs using DDP

3. Fully Sharded Data Parallel

.....

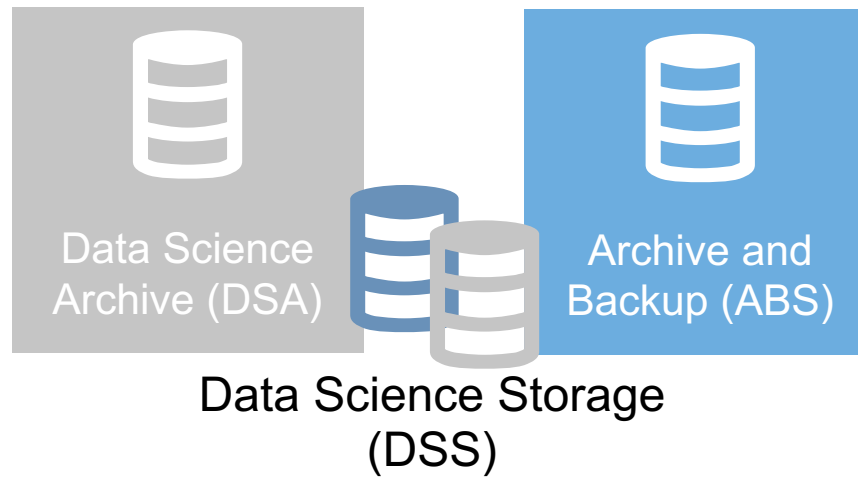
- ❑ Introduction to Fully Sharded Data Parallel
- ❑ Exercise: Train VGG-199 on 2 GPUs using FSDP

Planned breaks

- 11:30 – 11:45 Coffee Break I
- 12:30 – 13:30 Lunch break
- 15:00 – 15:15 Coffee Break II

1. Introduction to the LRZ AI Resources

Overview of the LRZ Systems



Multi-purpose cluster systems might be used for AI workloads as well, but have different focus

LRZ Linux Cluster

CoolMUC-2 Teramem-2 CoolMUC-3

[lxlogin\[1-3\].lrz.de](http://lxlogin[1-3].lrz.de)
lxlogin8.lrz.de

Designed and Configured for AI

LRZ AI Systems

- “Big Data” CPU nodes
- HPE P100 node
- V100 nodes
- DGX-1 P100, DGX-1 V100
- Multiple DGX A100

login.ai.lrz.de
<https://login.ai.lrz.de>

Flexible system that copes with almost any workload

LRZ Compute Cloud

LRZ Compute Cloud
(w/ some GPUs)

<https://cc.lrz.de>

1. Introduction to the LRZ AI Resources

Overview of the LRZ AI Systems



	HGX H100 Architecture	DGX A100 Architecture	DGX A100 Architecture MIG	DGX-1 V100 Architecture	DGX-1 P100 Architecture	HPE Intel Skylake + Nvidia Node	V100 GPU Nodes	CPU Nodes
Number of Nodes	30	4	1	1	1	1	4	12
CPU cores per node	96	252	252	76	76	28	19	18 / 28 / 38 / 94
Memory per node	768GB	2 TB	1 TB	512 GB	512 GB	256 GB	368 GB	min. 360 GB
GPUs per node	4 NVIDIA H100	8 NVIDIA A100	8 NVIDIA A100 (16 MIG partitions)	8 Nvidia Tesla V100	8 Nvidia Tesla P100	4 Nvidia Tesla P100	2 Nvidia Tesla V100	--
Memory per GPU	94 GB	80 GB	40 GB (20GB per MIG partition)	16 GB	16 GB	16GB	16 GB	--
SLURM Partition	lrz-hgx-h100- 92x4	lrz-dgx-a100- 80x8	lrz-dgx-a100- 40x8-mig	lrz-dgx-1- v100x8	lrz-dgx-1- p100x8	lrz-hpe- p100x4	lrz-v100x2	lrz-cpu
Nodes	lrz-hgx-h100- [001-030]	lrz-dgx-a100- [001-002,004- 005]	lrz-dgx-a100- 003	dgx-002	dgx-001	p100-001	gpu-[001- 003,005]	cpu-[001-012]

1. Introduction to the LRZ AI Resources

Storage on the LRZ AI Systems



Storage Pool	Designated Use	Top-level Directory	Size Limit	Automated Backup	Expiration	Additional Information
Home directory	unified home directory with the LRZ Linux Cluster, created when LRZ Linux Cluster access is granted Not suitable for heavy and/or high-frequency I/O operations - use the AI Systems DSS instead.	/dss/dsshome1 /.../<user>	100 GB	yes, backup to tape and file system snapshots	lifetime of LRZ project	File Systems and IO on Linux-Cluster
AI Systems DSS	high-bandwidth, low latency I/O, access is granted upon request through the LRZ Servicedesk	/dss/dssfs04	up to 4 TB	no	until further notice	
Linux Cluster DSS	general purpose, long-term data storage	/dss/dssfs02 /dss/dssfs03	up to 10 TB (or 20TB+ with associated costs)	yes for paid DSS / no for the free tier	lifetime of data project	File Systems and IO on Linux-Cluster (or DSS on demand offer with associated costs)
Exclusive/private DSS systems	specified by the system owner, can be purchased, implemented and housed exclusively for a private group of dedicated users	/dsslegfs01 /dsslegfs02 /dssmcm1fs01	specified by the system owner	specified by the system owner	specified by the system owner	Data Science Storage ("joint project offer")

Access to the LRZ AI Systems – How to access?

- User requirements to get the access:
 1. Own / get a Linux Cluster account:
<https://doku.lrz.de/display/PUBLIC/Access+and+Login+to+the+Linux-Cluster>
 2. Submit a service request to [LRZ Servicedesk](#) – select "AI topics" and "LRZ AI Systems - Request for Access" from the drop-down lists. Request has to include Linux Cluster account username and a description of the intended usage.

- Login node login.ai.lrz.de accessible via SSH:

```
$ ssh --login_name=xxyyzz login.ai.lrz.de
```

- Make sure you are connected to the Munich Scientific Network (MWN).
- Provide your LRZ Linux Cluster credentials to log in.

```
$ ssh --login_name=xyyyzz login.ai.lrz.de
```

Executes in the login node

```
$ sinfo
```

```
$ squeue --user=xyyyzz
```

```
$ salloc --partition=lrz-v100x2 --gres=gpu:1
```

```
$ srun --pty bash
```

```
$ scancel job_id
```

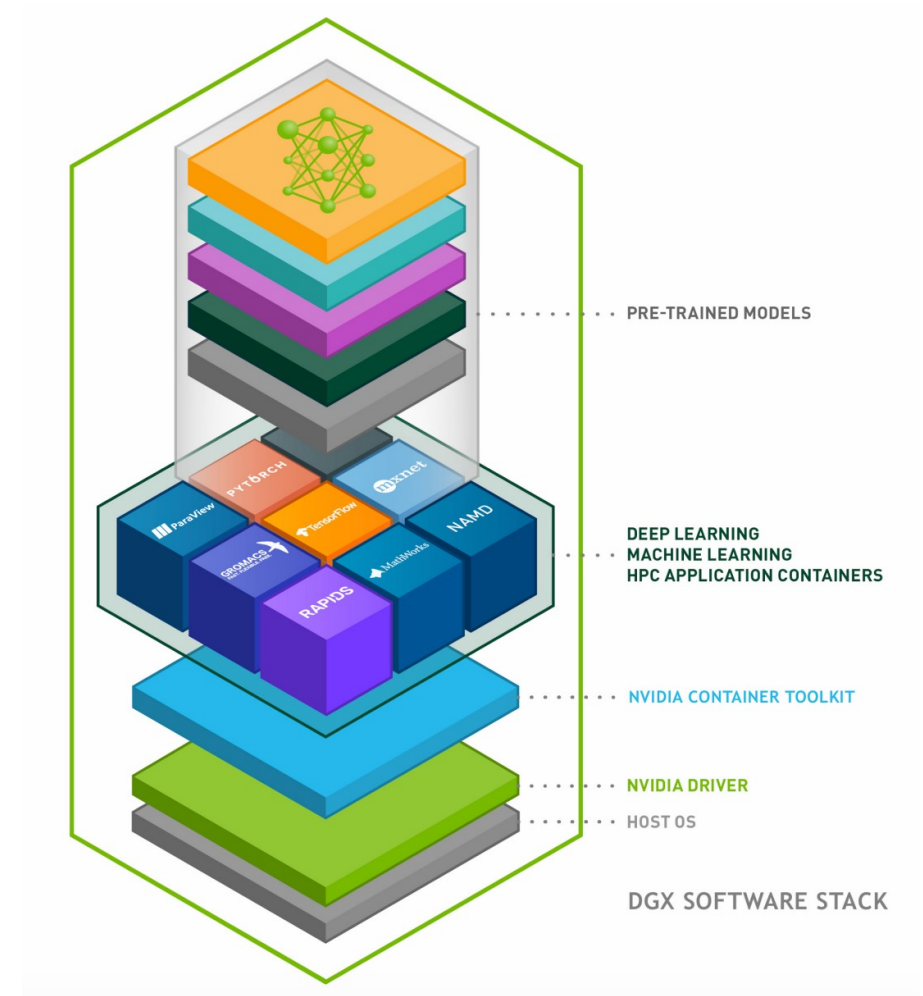
```
di82hod@login-1:~$ sinfo
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
lrz-v100x2*	up	14-00:00:0	1	mix	gpu-005
lrz-v100x2*	up	14-00:00:0	3	alloc	gpu-[001-003]
lrz-hpe-p100x4	up	14-00:00:0	1	idle	p100-001
lrz-dgx-1-p100x8	up	14-00:00:0	1	drain*	dgx-001
lrz-dgx-1-v100x8	up	14-00:00:0	1	alloc	dgx-002
lrz-dgx-a100-80x8	up	14-00:00:0	2	mix	lrz-dgx-a100-[002,004]
lrz-dgx-a100-80x8	up	14-00:00:0	2	alloc	lrz-dgx-a100-[001,005]
lrz-dgx-a100-40x8-mig	up	14-00:00:0	1	mix	lrz-dgx-a100-003
lrz-cpu	up	14-00:00:0	6	mix	cpu-[001-002,004-005,007,009]
lrz-cpu	up	14-00:00:0	1	alloc	cpu-003
lrz-cpu	up	14-00:00:0	2	idle	cpu-[006,008]
mcml-dgx-a100-40x8	up	14-00:00:0	2	mix	mcml-dgx-[004,008]
mcml-dgx-a100-40x8	up	14-00:00:0	6	alloc	mcml-dgx-[001-003,005-007]
test-v100x2	up	14-00:00:0	1	idle	gpu-004

```
di82hod@login-1:~$ salloc -p lrz-hpe-p100x4 --gres=gpu:2 --time=02:00:00
salloc: Pending job allocation 162333
salloc: job 162333 queued and waiting for resources
salloc: job 162333 has been allocated resources
salloc: Granted job allocation 162333
di82hod@login-1:~$ srun --pty bash
di82hod@p100-001:~$ exit
exit
di82hod@login-1:~$ scancel 162333
di82hod@login-1:~$ salloc: Job allocation 162333 has been revoked.
di82hod@login-1:~$
```

Nvidia NGC Containers

- The NGC catalogue provides access to **GPU accelerated** software that speeds up end-to-end workflows with performance optimized **containers**, **pretrained AI models**, and **SDKs** that can be deployed on any NVIDIA's GPU powered systems.
- The **NVIDIA Container Toolkit** includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPUs.
- The **NVIDIA CUDA Toolkit**, incorporated within each GPU-accelerated container in NGC, is the development environment for creating high performance NVIDIA GPU-accelerated applications.
- <https://catalog.ngc.nvidia.com>



1. Introduction to the LRZ AI Resources

Nvidia NGC Containers



The screenshot shows the NVIDIA NGC AI Development Catalog search results for 'pytorch'. The interface includes a search bar with 'pytorch' entered, a 'Sort: Relevance' dropdown, and a 'Back to home page' link. The results are displayed in a grid of four cards:

- PyTorch**: A GPU accelerated tensor computational framework. Functionality can be extended with common Python libraries such as NumPy and SciPy...
- Merlin PyTorch**: The Merlin PyTorch container allows users to do pre-processing and feature engineering with NVTabular, and then train a deep-learning based recommender...
- PyTorch Lightning**: Lightweight framework for training models at scale, without the boilerplate. Train on any number of GPUs or nodes without changing your code...
- NVIDIA L4T PyTorch**: PyTorch is a GPU accelerated tensor computational framework with a Python front end. This container contains PyTorch and torchvision pre-installed in a...

On the left, there are filters for 'NVIDIA AI Enterprise Support' (Yes/No) and 'Entity Type' (Model, Resource, Container, Collection, Helm Chart). Below that, 'Use Case' filters include Automatic Speech Recognition, Natural Language Processing, Text to Speech, Question Answering, Recommendation, and Language Modeling.

The screenshot shows the details page for the '23.03-py3' PyTorch container. The page includes a 'Get Container' dropdown and a 'Deploy to Vertex AI' button. A tooltip displays the image path: 'nvcr.io/nvidia/pytorch:23.03-py3'. The main content area is divided into sections:

- Description**: PyTorch is a GPU accelerated tensor computational framework. Functionality can be extended with common Python libraries such as NumPy and SciPy. Automatic differentiation is done with a tape-based system at the functional and neural network layer levels.
- Prerequisites**: Using the PyTorch NGC Container requires the host system to have the following installed:
 - Docker Engine
 - NVIDIA GPU Drivers
 - NVIDIA Container Toolkit
- Running PyTorch Using Docker**: To run a container, issue the appropriate command as explained in the [Running A Container](#) chapter in the [NVIDIA Containers For Deep Learning Frameworks User's Guide](#) and specify the registry, repository, and tags. For more information about using NGC, refer to the [NGC Container User Guide](#).

1. Introduction to the LRZ AI Resources

Nvidia NGC Containers – Setting up credentials



NVIDIA NGC | CATALOG Welcome Guest

NGC Catalog

Deploy performance-optimized AI/HPC software containers, pre-trained AI models, and Jupyter Notebooks that accelerate AI developments and HPC workloads on any GPU-powered on-prem, cloud and edge systems. Instantly experience end-to-end workflows with [access to free hands-on labs on NVIDIA LaunchPad](#), and learn about enterprise support for NVIDIA accelerated software [here](#).

[Register for NGC](#)

NVIDIA NGC: AI Development Catalog

Displaying 0 results

Sorting & Filters

Getting Started

NVIDIA NGC | SETUP Maja Piskac orwp0ei9x4xt

Setup

Generate API Key

Generate your own API key to use the NGC service through the Docker client.

[Get API Key](#)

CLI

The NGC command line interface (NGC CLI) can run deep learning jobs on NVIDIA Docker containers.

[Documentation](#) [Downloads](#)

NVIDIA NGC | CATALOG Maja Piskac orwp0ei9x4xt

NVIDIA LaunchPad

Instantly experience end-to-end workflows with free hands-on labs.

[Get Started](#)

NVIDIA NGC: AI Development Catalog

Displaying 0 results

Sorting & Filters

Getting Started

NVIDIA NGC | SETUP Maja Piskac orwp0ei9x4xt

Setup > API Key

[Generate API Key](#)

API

API Information

Your API Key authenticates your use of NGC service when using NGC CLI or the Docker client. Anyone with this API Key has access to all services, actions, and resources on your behalf.

Click Generate API Key to create your own API Key. If you have forgotten or lost your API Key, you can come back to this page to create a new one at any time.

Usage

Use your API key to log in to the NGC registry by entering the following command and following the prompts:

NGC CLI

```
$ ngc config set
```

Docker™

For the username, enter '\$oauthtoken' exactly as shown. It is a special authentication token for all users.

```
$ docker login nvcr.io
```

Username: \$oauthtoken
Password: <Your Key>

Nvidia NGC Containers – Setting up credentials

- Create the file *enroot/credentials* within your \$HOME and insert the following lines in it:

```
machine nvcr.io login $oauthtoken password < KEY>  
machine authn.nvidia.com login $oauthtoken password <KEY>
```

- Where <KEY> is the API key generated and copied in the previous step.
- Introduce a new line after < KEY>.
- Now you can import containers from Nvidia NGC on compute nodes of LRZ AI Systems, e.g. a Pytorch container, with:

```
$ enroot import docker://nvcr.io/nvidia/pytorch:24.08-py3
```

- Enroot container runtime operates completely in user space.
- It allows to run containers defined by container images from the NVIDIA NGC Cloud or from the Docker Hub.
- Not available on the login node, but on the compute nodes!
- The Enroot Workflow:
 - I. **Import** an Enroot Container Image – resulting in **sqsh** file
 - II. Create an Enroot Container with **create**,
 - III. Run software inside an existing Enroot Container with **start**.

Executes in the login node

```
$ salloc --partition=lrz-hpe-p100x4 --gres=gpu:1
```

```
$ srun --pty bash
```

Executes in the allocated compute node

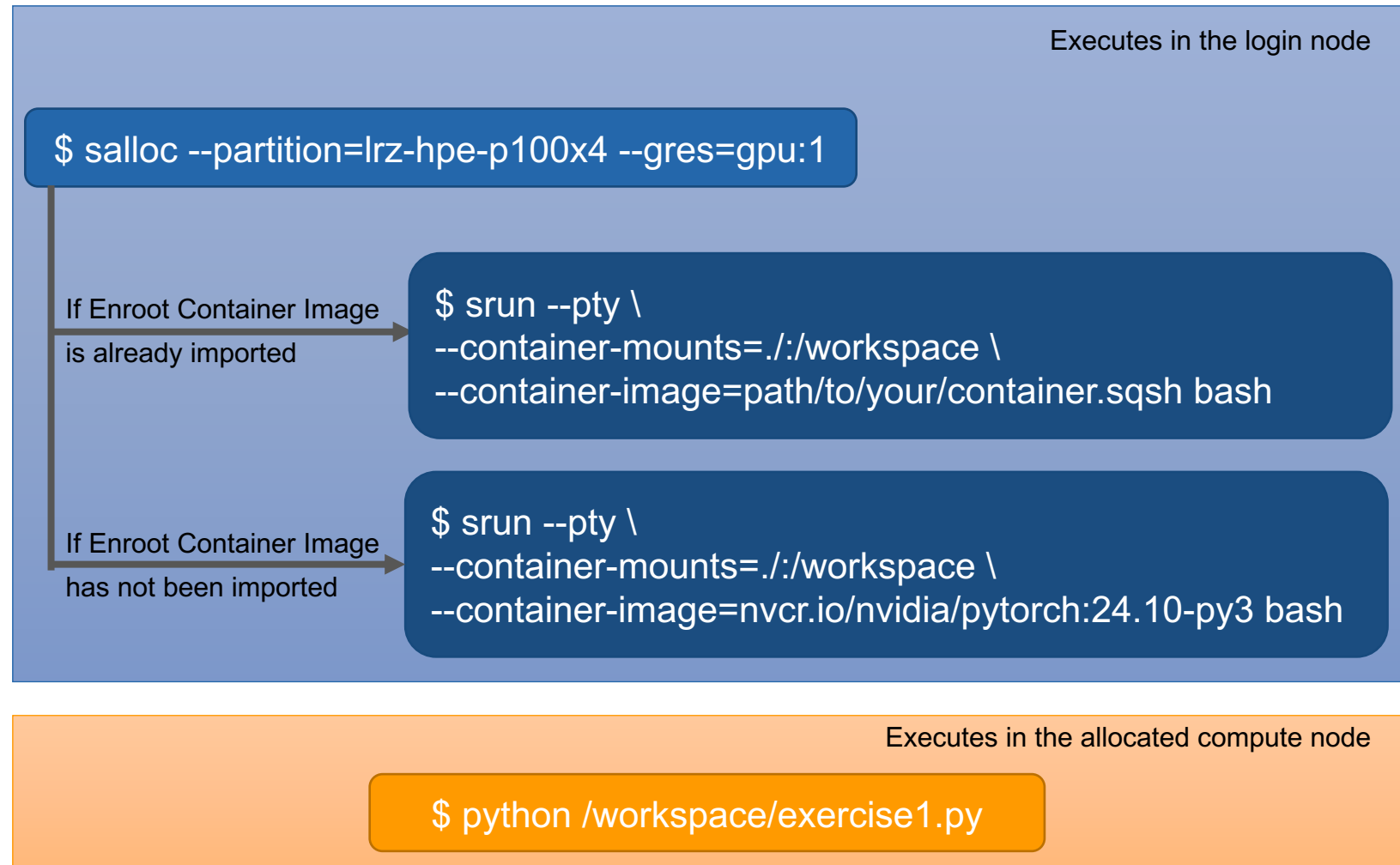
```
$ enroot import docker://nvcr.io/nvidia/pytorch:24.10-py3
```

```
$ enroot create nvidia+pytorch+24.10-py3.sqsh
```

```
$ enroot start nvidia+pytorch+24.10-py3
```

Running Applications as Interactive Jobs

- Interactive jobs are submitted to an existing allocation of resources using the **srun** command.
- We can **mount** existing data from outside of the container into container.
- Enroot container creation and job submission in a single step can be done via a plugin called *pyxis*.



Running Applications as Batch Jobs



- Batch jobs are the preferred and quicker way of using the LRZ AI Systems.
- Batch job is queued and executed when the resources are available.
- It does the allocation and running of the job for you (instead of `salloc` and `srun`).
- The **`sbatch`** command submits jobs described in a **`sbatch` script file**.
- You need to specify the partition and number of GPUs that you want to use.
- Two additional required arguments: *output* and *error messages* file.

```
#!/bin/bash
#SBATCH -p lrz-hpe-p100x4
#SBATCH --gres=gpu:1
#SBATCH -o exercise1.out
#SBATCH -e exercise1.err

srun \
--container-mounts='./:/workspace' \
--container-image='nvcr.io/nvidia/pytorch:24.10-py3' \
python /workspace/exercise1.py
```

Executes in the login node

```
$ sbatch exercise1.sbatch
```

Dealing with base images from catalogues other than NGC

- If your image does not supply the CUDA Toolkit, do not install it within the image, because this fixes paths to the existing NVIDIA driver on the target machine and might crash if the NVIDIA driver is upgraded.
- Instead add the following **environment variables** within the container, and the container runtime will copy within the container the needed libraries. Refer to <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/docker-specialized.html> for more information on the accepted values of these variables.

Executes in the allocated resource

```
$ enroot import docker://some/image-no-cuda
```

```
$ enroot create --name base_container some+image-no-cuda.sqsh
```

```
$ enroot start base_container bash
```

```
echo "NVIDIA_DRIVER_CAPABILITIES=compute,utility" >> /etc/environment  
echo "NVIDIA_REQUIRE_CUDA=cuda>=9.0" >> /etc/environment  
echo "NVIDIA_VISIBLE_DEVICES=all" >> /etc/environment
```

```
$ exit
```

```
$ enroot export --output base_image.sqsh base_container
```

Creating an extended Enroot image

- If your workload depends on a package not provided by the used image.

Executes in the allocated resource

```
$ enroot create --name custom_container nvidia+pytorch+24.08-py3.sqsh
```

```
$ enroot start custom_container
```

```
$ pip install --no-cache-dir lightning  
$ HOROVOD_GPU_OPERATIONS=NCCL pip install --no-cache-dir horovod
```

```
$ exit
```

```
$ enroot export --output custom_container.sqsh custom_container
```

- For installing some applications you need to be root within the container (e.g., installing software using the apt package manager in Debian and Ubuntu-based containers.) In this case, add the `--root` flag.

Executes in the allocated resource

```
$ enroot start --root my_container
```

```
$ apt update  
$ apt install python3-dev
```


Access to AI Systems through interactive web servers

- Jupyter Notebook, JupyterLab, RStudio Server and TensorBoard
- Available at <https://login.ai.lrz.de>
- To start e.g. a Jupyter Notebook session select from the top panel:
"Interactive Apps" => "Jupyter Notebook"
- For a typical use-case:
 - select the type of resources (CPU only or CPU + single GPU)
 - specify your workload (a combination of CPU core and RAM requirements)
 - select the container environment you want to work with (e.g. available PyTorch or Tensorflow container, or a custom container)
 - finally, specify the number of hours you plan to work (be aware that your session will be shut down when this time limit is reached, and any unsaved work will then be lost).

1. Introduction to the LRZ AI Resources

Access to AI Systems through interactive web servers



The image displays a collage of overlapping screenshots from the LRZ AI resources interface. The top-left screenshot shows the 'Systems Web UI (TEST INSTANCE)' with a navigation menu (Files, Jobs, Clusters, Interactive Apps) and a sidebar listing 'Interactive Apps' (Jupyter Notebook, RStudio Server, TensorBoard) and 'Servers'. The bottom-right screenshot shows a Jupyter Notebook interface with a code cell containing Python code for importing packages and defining hyper-parameters.

```
In [3]: # Import packages
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms

# Hyper-parameters
image_width = 32
image_channels = 3
conv1_out_channels = 50
conv2_out_channels = 75
kernel_size = 5
pool_size = 2
fc1_out_channels = 50
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
dim_1 = int((image_width-kernel_size+1)/pool_size)
dim_2 = int((dim_1-kernel_size+1)/pool_size)
```

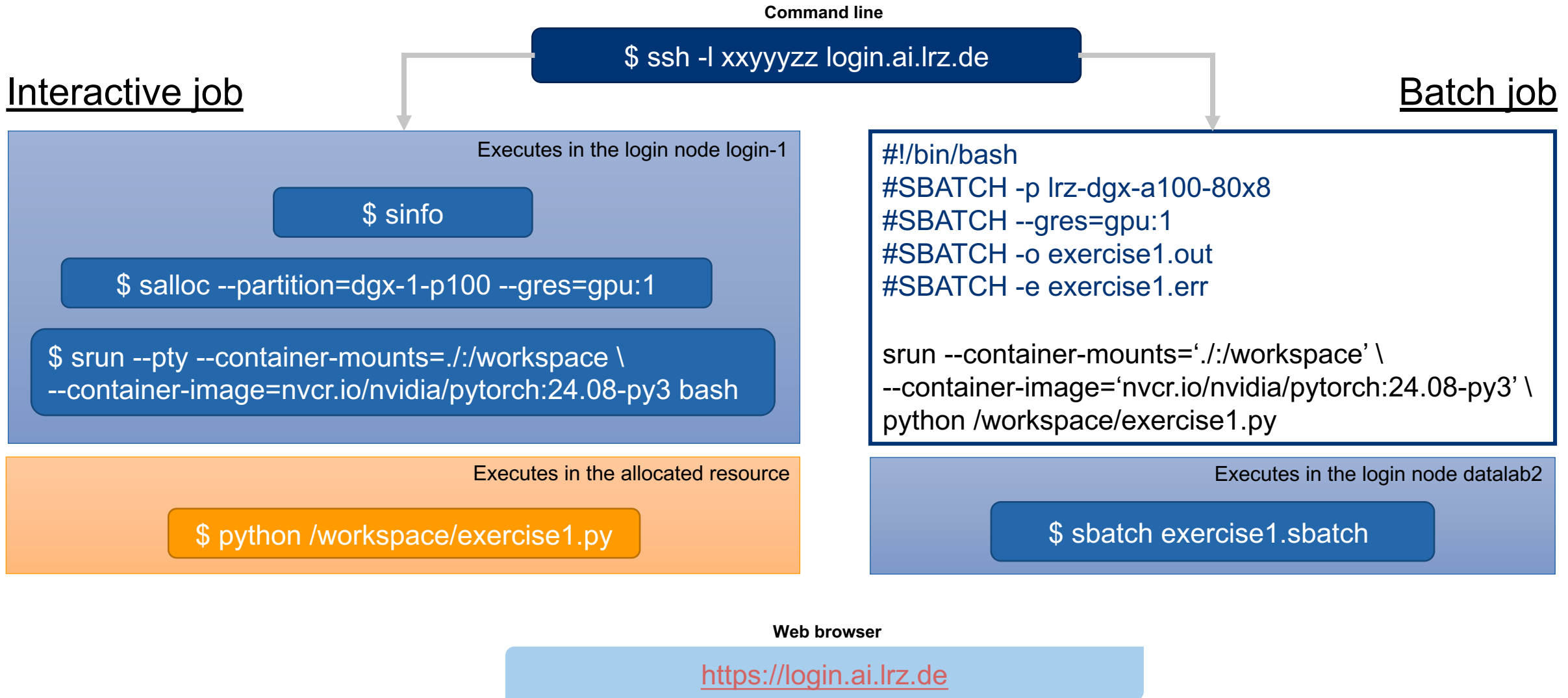
Public Datasets and Containers on the LRZ AI Systems

- Dedicated DSS container for storing public datasets and Enroot container images of interest to researchers.
- Procedure to request the addition of public datasets / Enroot images:
 - make sure the dataset / Enroot image is licensed for public usage and requires no individual license nor registration, and in case of an Enroot image make sure the image is not provided by the Nvidia NGC, Dockerhub or another public repository directly
 - open a ticket with the [LRZ Servicedesk](#), providing the location of the dataset / Dockerfile for building the image, and a justification for public interest (including the expected target audience)
 - provide instructions for downloading the dataset (ideally shell script) / building the image (if non-standard).

Dataset	Location	Version	Licence
AlphaFold	/dss/dssfs04/pn69za/pn69za-dss-0004/datasets/alphafold_2024	Last update March 2024 following the instructions here https://github.com/deepmind/alphafold#genetic-databases	https://github.com/deepmind/alphafold#license-and-disclaimer
COCO-Stuff	dss/dssfs04/pn69za/pn69za-dss-0004/datasets/cocostuff/	2020 Update (train2017.zip, val2017.zip, annoations_trainval2017.zip, stuff_annotations_trainval2017.zip)	https://cocodataset.org/#termsofuse
Visual Genome	dss/dssfs04/pn69za/pn69za-dss-0004/datasets/visualgenome/	Version 1.4 of dataset completed as of July 2017	Creative Commons Attribution 4.0 International License

1. Introduction to the LRZ AI Resources

Summary



The background of the slide is a photograph of a modern, multi-story building with a facade of vertical slats and large windows. The image is overlaid with a semi-transparent blue filter. In the foreground, there are trees and a sidewalk. A dark blue horizontal bar is positioned across the middle of the image, containing the title text.

Hands-On Exercise 0

Creating an extended Enroot image



1. Write a job script to import and extend a container
2. Execute and create your own container

Job script info:

1. partition: lrz-hgx-h100-92x4
2. reservation: aits
3. gpu resources: 1

1. Introduction to the LRZ AI Systems

.....

- ❑ Overview of the LRZ AI Systems
- ❑ Access to the LRZ AI Systems
- ❑ NVIDIA NGC Cloud
- ❑ Introduction to Enroot Containers
- ❑ Interactive and Batch Jobs
- ❑ Open on Demand
- ❑ Exercise: Run a job and extend an Enroot container

2. Data Distributed Training

.....

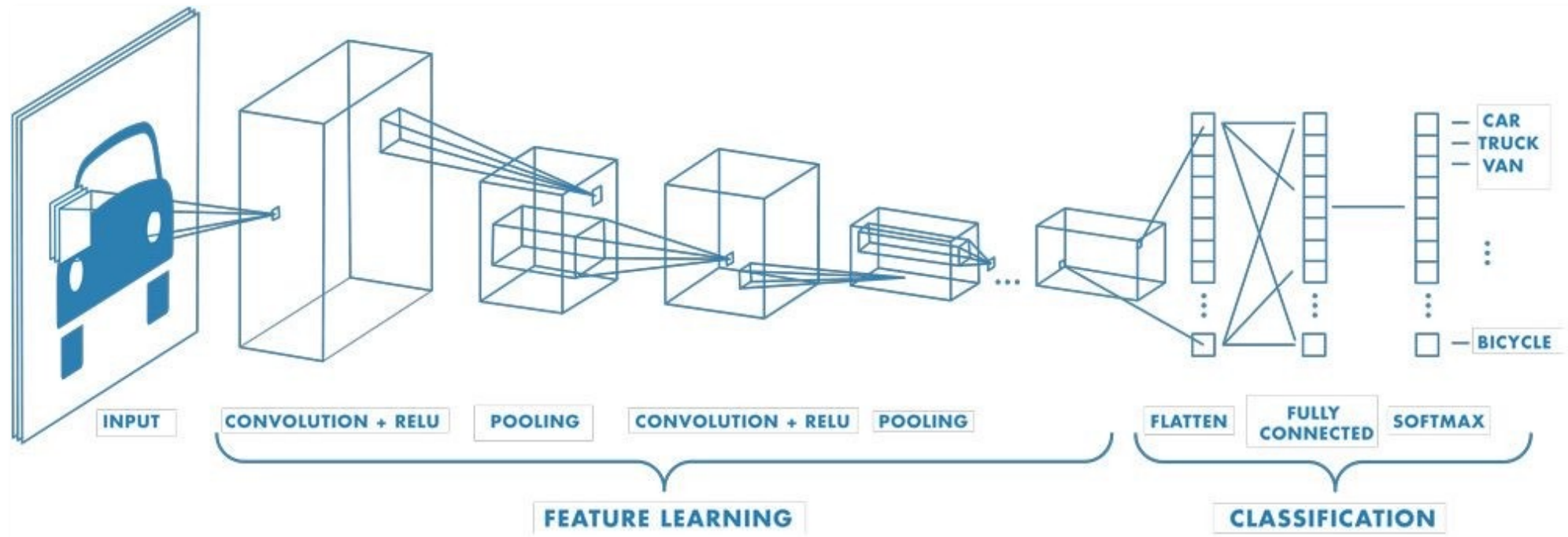
- ❑ Introduction to Convolutional Neural Networks
- ❑ Exercise: Train VGG-19 on a GPU
- ❑ Introduction to Distributed Training
- ❑ Exercise: Train VGG-199 on 2 GPUs using DDP

3. Fully Sharded Data Parallel

.....

- ❑ Introduction to Fully Sharded Data Parallel
- ❑ Exercise: Train VGG-199 on 2 GPUs using FSDP

Introduction to Convolutional Neural Networks



<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

2. Data Distributed Training

Convolutional Neural Network Architectures – VGG (Visual Geometry Group)



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- Main idea: increasing depth of CNNs
- 3 × 3 conv. layers with a stride of 1 - small receptive fields (compared to prev. 11 × 11 with a stride of 4 in AlexNet)
- 1 × 1 conv. to make the decision function more non-linear without changing the receptive fields
- ReLU activation function
- ImageNet dataset – 4 days of training
- 19 layers – 144M parameters

<https://arxiv.org/pdf/1409.1556.pdf>

2. Data Distributed Training

CIFAR10 Data

```
# download the data
```

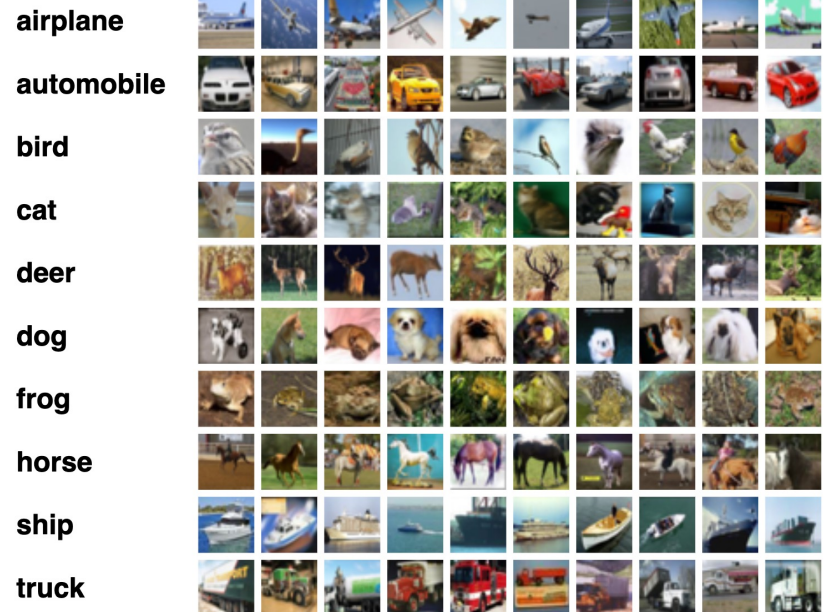
```
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, transform=transforms.ToTensor(), download=True)
```

```
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, transform=transforms.ToTensor(), download=True)
```

```
# transform to DataLoader
```

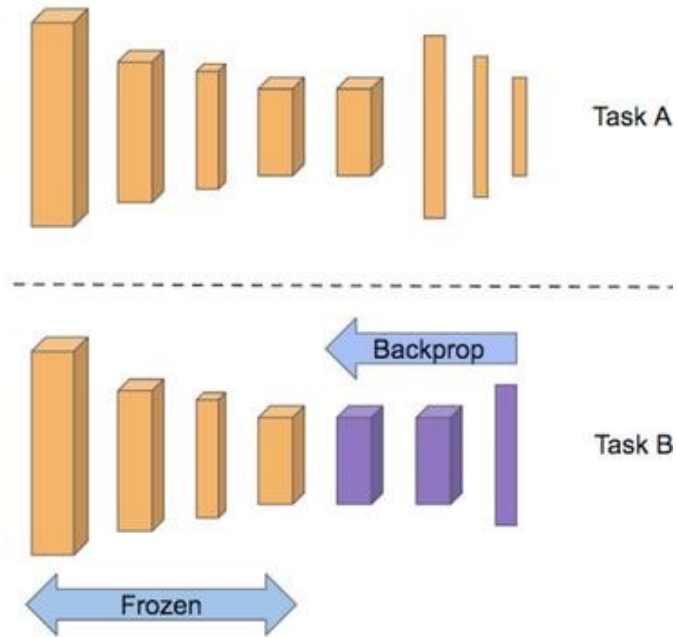
```
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
```

```
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```



Input size: 32x32x3

Transfer learning – Pretrained Convolutional Neural Networks



- Pre-trained model is a saved network that was previously trained on a large dataset.
- Feature Extraction: Use the feature maps from the pre-trained model to detect features in the new samples. Add a new classifier, which will be trained from scratch to make predictions.
- Fine-Tuning: Unfreeze a few top layers of a pretrained model and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us "fine-tune" the higher-order feature maps to make them more relevant for the specific task.

```
import torchvision.models as models
model = models.resnet34(weights='IMAGENET1K_V1')
```

```
import torchvision.models as models
model = models.vgg19(weights='IMAGENET1K_V1')
```

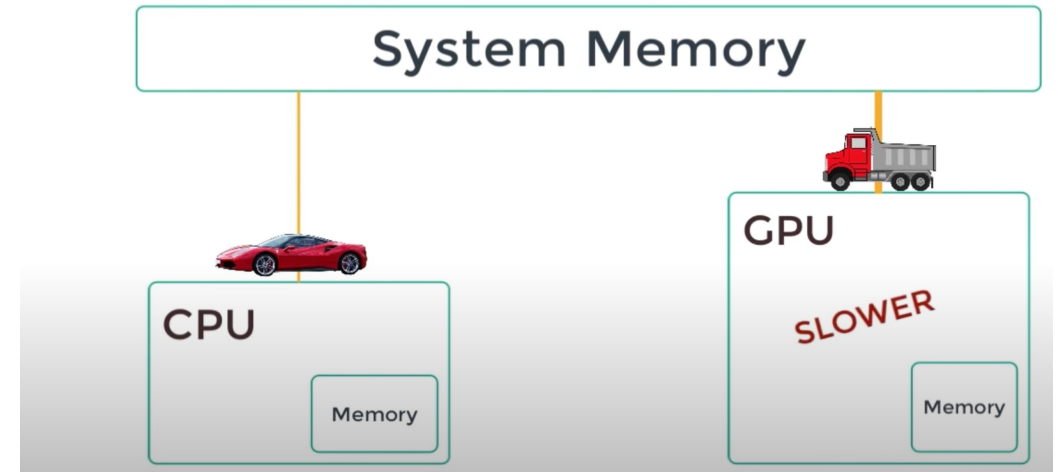
2. Data Distributed Training

Neural Networks and GPUs - Why?

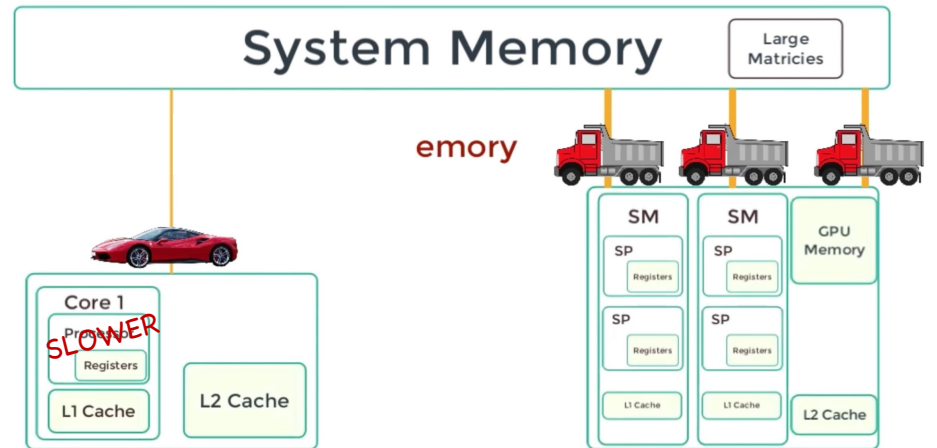
- GPUs in comparison to CPUs:
 - GPU allows parallel running of repetitive calculations within an application
 - CPU can be thought of as the taskmaster of the entire system, coordinating a wide range of general-purpose computing tasks
 - GPU performs a narrower range of more specialized tasks (e.g., matrix multiplications)

- CPUs are faster than GPUs in scalar multiplications.
- GPUs are faster than CPUs in matrix multiplications.

CASE 1: Scalar Multiplication

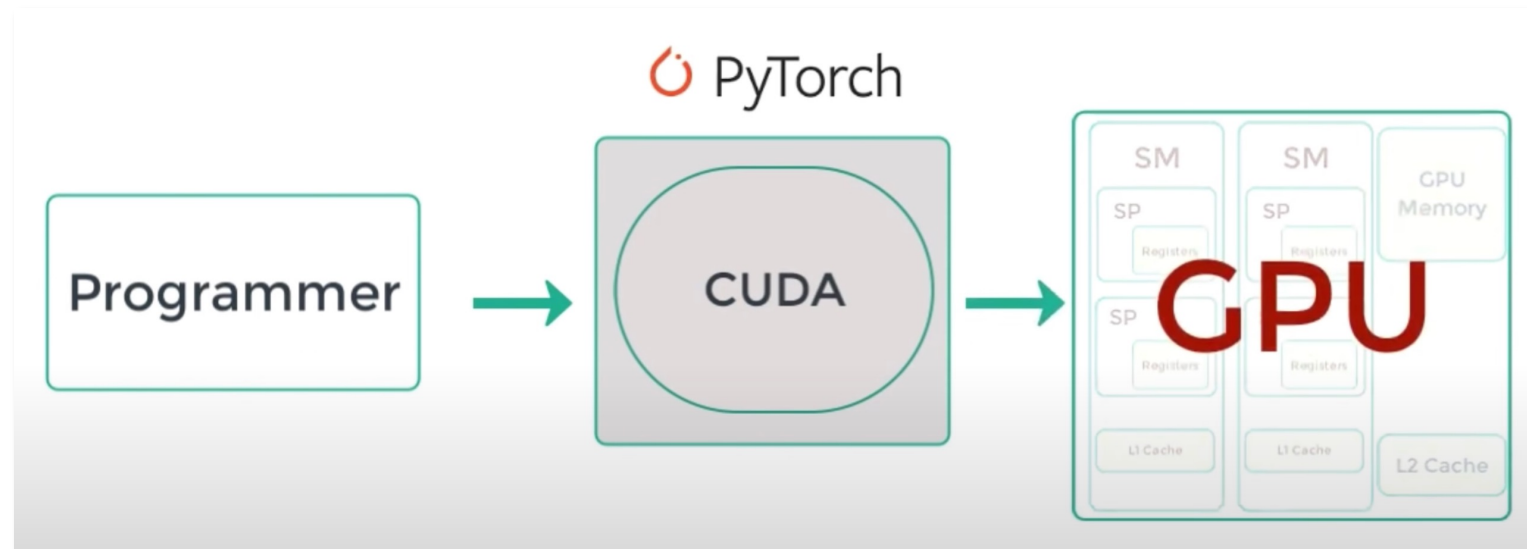


CASE 2: Matrix Multiplication



SP: Streamlined Processor
SM: Streamlined Multiprocessor

Neural Networks and GPUs - How?



```
# Device configuration - cpu  
device = torch.device('cpu')
```

```
# Device configuration - gpu  
device = torch.device('cuda')
```

```
# Model to device  
model = model.to(device)
```

```
# Data to device  
images = images.to(device=device)  
labels = labels.to(device=device)
```


The background of the slide is a photograph of a modern, multi-story building with a facade of vertical slats and large glass windows. The image is overlaid with a semi-transparent blue filter. In the foreground, there are trees and a street with a few people and a sign.

Hands-On Exercise 1

VGG-19 + CIFAR10 on a GPU



1. Make a batch-size vs epoch time table
2. Pull the GitHub repo: <https://github.com/LRZ-BADW/ai-systems.git>
3. Explore the code
4. Write a job script to run the code
5. Play with the batch size and note the time

Batch Size	1 GPU	2 GPU

Job script info:

- partition: lrz-hgx-h100-92x4
- reservation: aits
- gpu resources: 1

2. Data Distributed Training

Increasing Amount of Data Available for Deep Learning

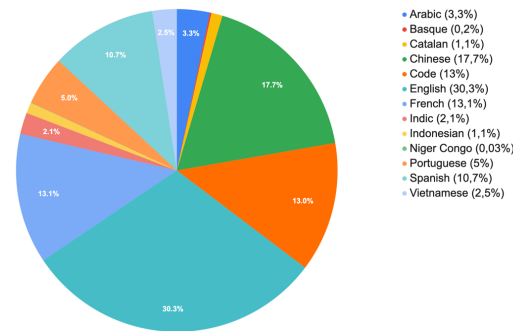
ImageNet



14,197,122 images
(150 GB)

<https://paperswithcode.com/dataset/imagenet>

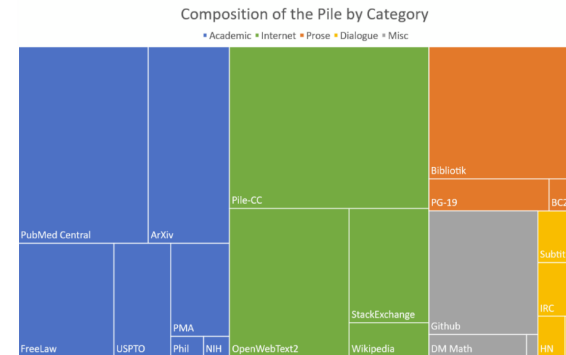
BigScience Multilingual Dataset for Language Modeling



350 billion tokens
(1.5 TB)
46 languages

<https://bigscience.huggingface.co/blog/building-a-tb-scale-multilingual-dataset-for-language-modeling>

Pile: Dataset of Diverse Text for Language Modeling



800GB

<https://arxiv.org/abs/2101.00027>

Common Crawl



2.95 billion web pages
(351.844 TB)

<https://commoncrawl.org/>

2. Data Distributed Training

Pytorch Distributed Data Parallel

- Based on package *torch.distributed* for synchronizing gradients.
- DDP registers a hook for each parameter that fires when the corresponding gradient is computed in the backward pass. That signal is used to trigger gradient synchronization across processes.
- Runs across *multiple GPUs* and across *multiple machines/nodes*.
- Near-linear scalability

Li et al., 2020, arXiv:2006.15704

<https://pytorch.org/docs/master/notes/ddp.html>

https://pytorch.org/tutorials/intermediate/ddp_tutorial.html#basic-use-case

```
import torch.distributed as dist
import torch.multiprocessing as mp
from torch.nn.parallel import DistributedDataParallel as DDP
```

```
def setup(rank, world_size):
    #environment variables for using torch.distributed
    os.environ['MASTER_ADDR'] = 'localhost'
    os.environ['MASTER_PORT'] = '12355'

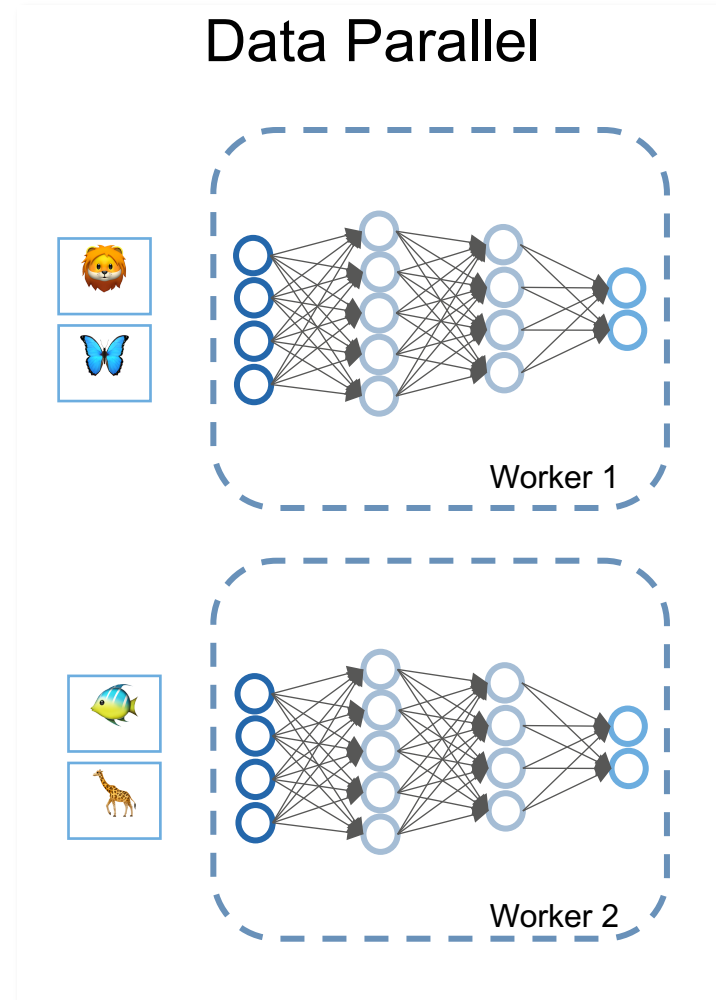
    #initialize the process group
    dist.init_process_group("nccl", rank=rank, world_size=world_size)
```

```
def main(rank, world_size, args):
    #setup DDP
    setup(rank, world_size)
    #create a model.....
    #wrap model in DDP
    ddp_model = DDP(model, device_ids=[rank])
    #define loss function and optimizer...
    #forward pass...
    #backward pass and update parameters....
    #cleanup
    dist.destroy_process_group()
```

```
if __name__=="__main__":

    world_size = torch.cuda.device_count()
    batch_size = int(batch_size / world_size)
    mp.spawn(main, args=(world_size, args), nprocs=world_size,
join=True)
```

Data Distributed Parallel Training of NNs



The background of the slide is a photograph of a modern, multi-story building with a facade of vertical slats and large glass windows. The image is overlaid with a semi-transparent blue filter. In the foreground, there are trees and a sidewalk.

Hands-On Exercise 2

VGG-19 + CIFAR10 on 2 GPUs - DDP



1. Refer to batch-size vs epoch time table
2. Pull the GitHub repo: <https://github.com/LRZ-BADW/ai-systems.git>
3. Explore the code
4. Write a job script to run the code
5. Play with the batch size and note the time

Batch Size	1 GPU	2 GPU

Job script info:

- partition: lrz-hgx-h100-92x4
- reservation: aits
- gpu resources: 2

1. Introduction to the LRZ AI Systems

.....

- ❑ Overview of the LRZ AI Systems
- ❑ Access to the LRZ AI Systems
- ❑ NVIDIA NGC Cloud
- ❑ Introduction to Enroot Containers
- ❑ Interactive and Batch Jobs
- ❑ Open on Demand
- ❑ Exercise: Run a job and extend an Enroot container

2. Data Distributed Training

.....

- ❑ Introduction to Convolutional Neural Networks
- ❑ Exercise: Train VGG-19 on a GPU
- ❑ Introduction to Distributed Training
- ❑ Exercise: Train VGG-199 on 2 GPUs using DDP

3. Fully Sharded Data Parallel Training

.....

- ❑ Introduction to Fully Sharded Data Parallel
- ❑ Exercise: Train VGG-199 on 2 GPUs using FSDP

3. Fully Sharded Data Parallel Training

Fully Sharded Data Parallel

- Inspired by ZeRO Stage 3 from DeepSpeed
- Ideal for training large models that do not fit into a single GPU
- Model parameters, gradients and optimizer states are sharded across GPUs

```
import torch.distributed as dist
import torch.multiprocessing as mp
from torch.distributed.fsdp import FullyShardedDataParallel as FSDP

def setup(rank, world_size):
    #environment variables for using torch.distributed
    os.environ['MASTER_ADDR'] = 'localhost'
    os.environ['MASTER_PORT'] = '12355'

    #initialize the process group
    dist.init_process_group("nccl", rank=rank, world_size=world_size)

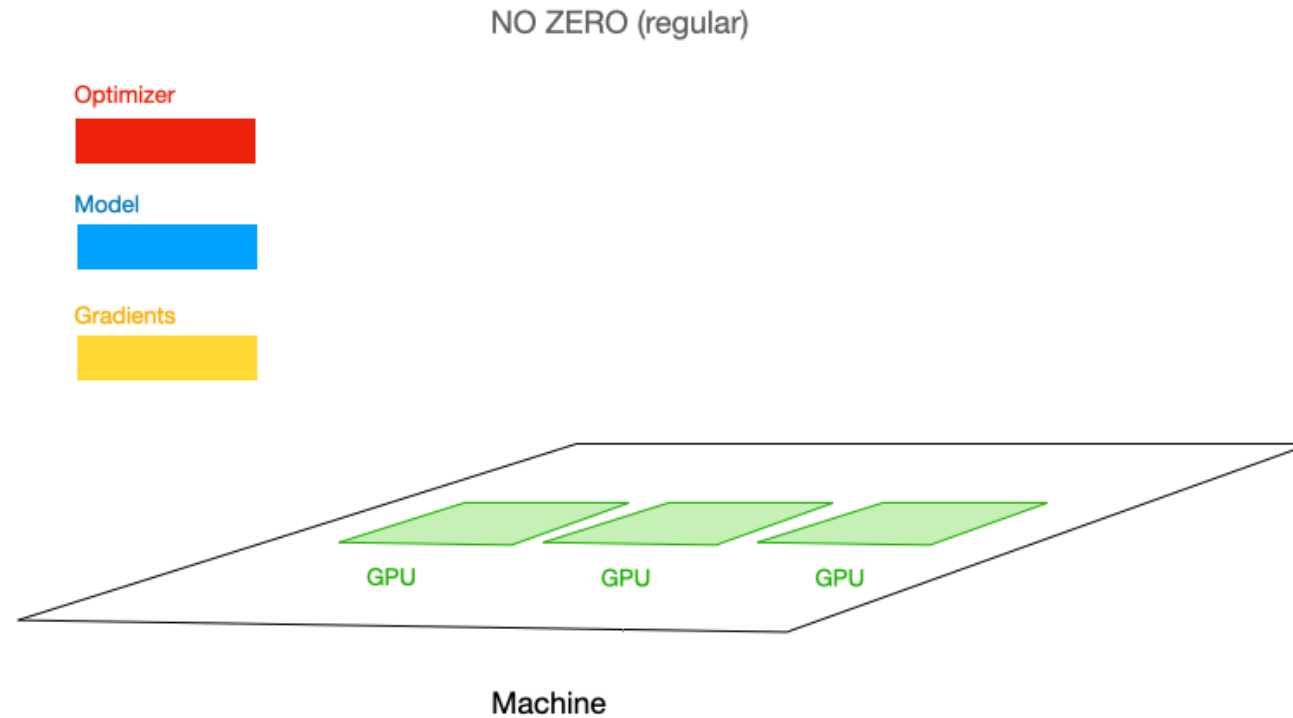
def main(rank, world_size, args):
    #setup FSDP
    setup(rank, world_size)
    #create a model.....
    #wrap model in FSDP
    fsdp_model = FSDP(model)
    #define loss function and optimizer...
    #forward pass...
    #backward pass and update parameters....
    #cleanup
    dist.destroy_process_group()

if __name__=="__main__":

    world_size = torch.cuda.device_count()
    batch_size = int(batch_size / world_size)
    mp.spawn(main, args=(world_size, args), nprocs=world_size,
join=True)
```

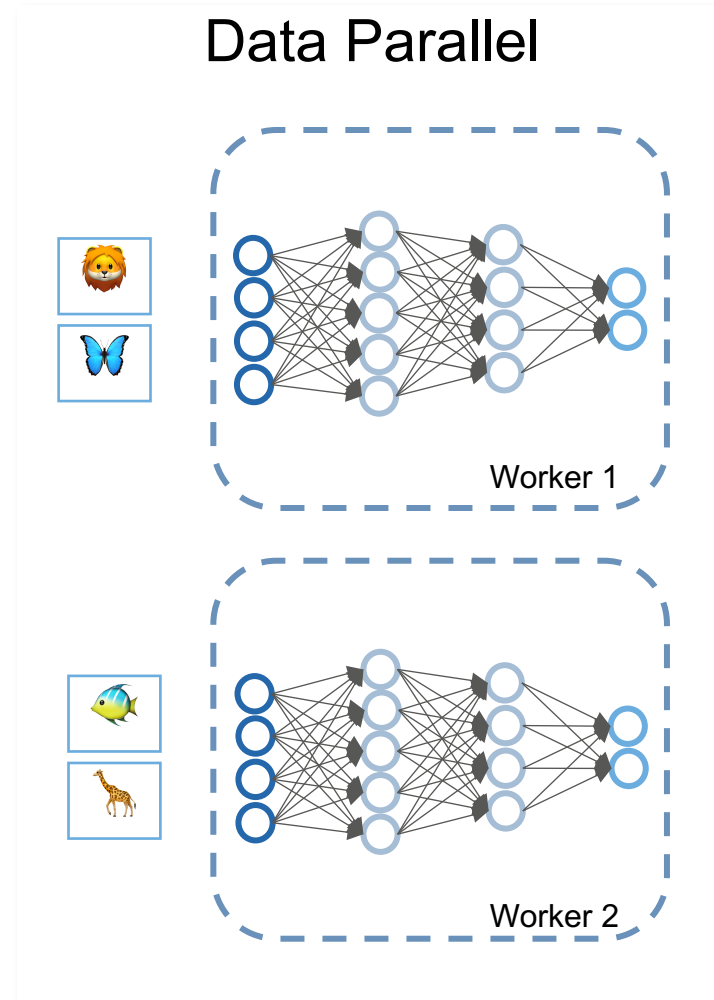
3. Fully Sharded Data Parallel Training

Fully Sharded Data Parallel



3. Fully Sharded Data Parallel Training

Data Distributed Parallel Training of NNs



The background of the slide is a photograph of a modern, multi-story building with a facade of vertical slats and large glass windows. The image is overlaid with a semi-transparent blue filter. In the foreground, there are trees and a sidewalk.

Hands-On Exercise 3

VGG-19 + CIFAR10 on 2 GPUs - FSDP



1. Refer to batch-size vs epoch time table
2. Pull the GitHub repo: <https://github.com/LRZ-BADW/ai-systems.git>
3. Explore the code
4. Write a job script to run the code
5. Play with the batch size and note the time

Batch Size	1 GPU	2 GPU

Job script info:

- partition: lrz-hgx-h100-92x4
- reservation: aits
- gpu resources: 2

A dark blue rectangular box with the text 'The End!' in white, bold, sans-serif font. The box is positioned in the lower-left quadrant of the image, overlapping the building facade.

The End!

1. Introduction to the LRZ AI Systems

.....

- ❑ Overview of the LRZ AI Systems
- ❑ Access to the LRZ AI Systems
- ❑ NVIDIA NGC Cloud
- ❑ Introduction to Enroot Containers
- ❑ Interactive and Batch Jobs
- ❑ Open on Demand
- ❑ Exercise: Run a job and extend an Enroot container

2. Data Distributed Training

.....

- ❑ Introduction to Convolutional Neural Networks
- ❑ Exercise: Train VGG-19 on a GPU
- ❑ Introduction to Distributed Training
- ❑ Exercise: Train VGG-19 on 2 GPUs using DDP

3. Fully Sharded Data Parallel Training

.....

- ❑ Introduction to Fully Sharded Data Parallel
- ❑ Exercise: Train VGG-19 on 2 GPUs using FSDP

The background of the slide is a photograph of a modern, multi-story building with a facade of vertical slats and large glass windows. The image is overlaid with a semi-transparent blue filter. In the foreground, there are several trees and a sidewalk. A dark blue rectangular box is positioned in the lower-left area of the image, containing the text 'Announcements' in white.

Announcements

Hackathon



- Half day event
- BYOC – Bring Your Own Code
- Hands-on & mentoring on scaling/parallelizing your code
- Write us at – Ajay.Navilarekal@lrz.de or Darshan.Thummar@lrz.de

Announcements
Feedback Survey



<https://survey.lrz.de/index.php/885115?lang=en>