Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities

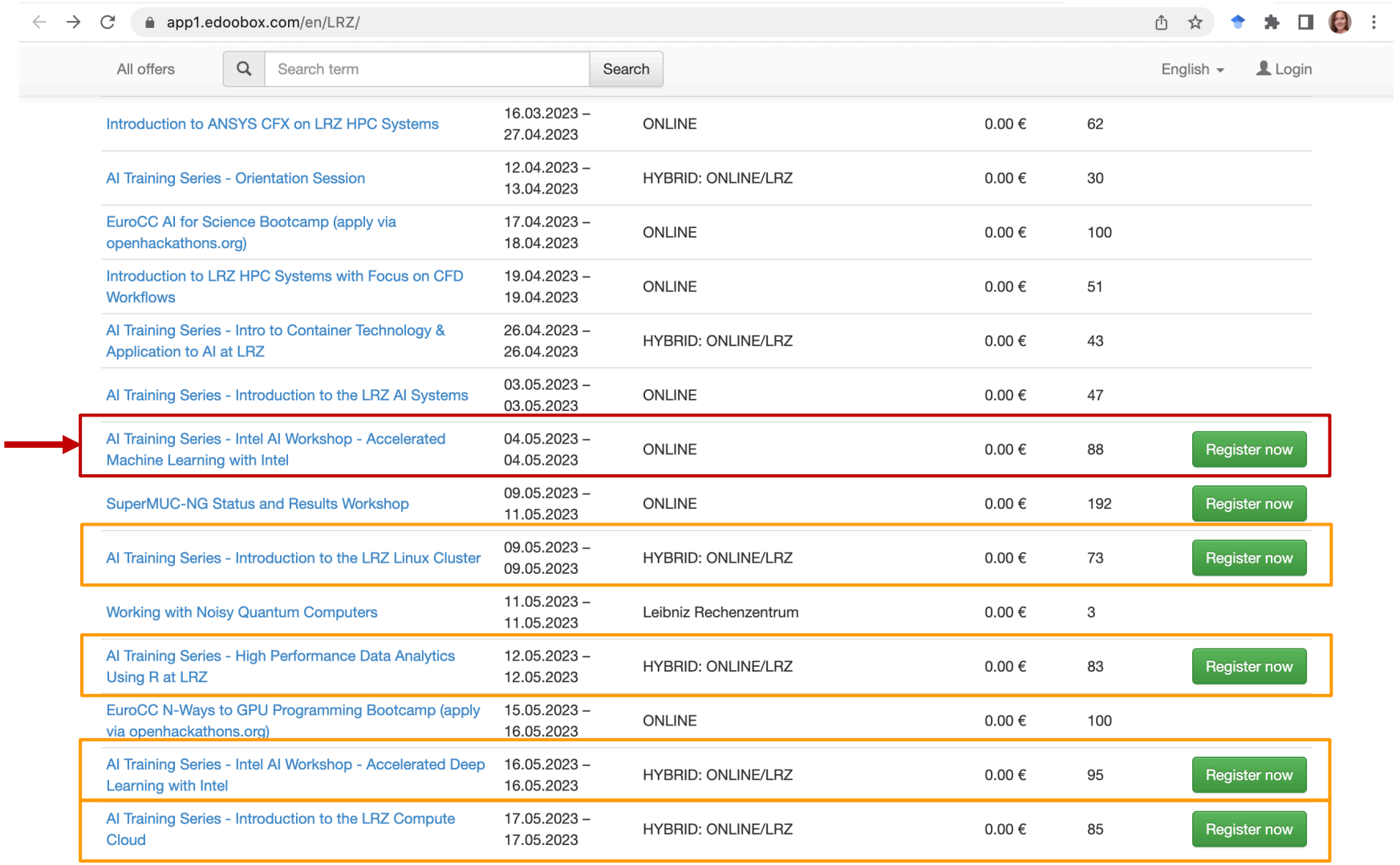
AI Training Series
Introduction to the LRZ AI Systems

03.05.2023 | Maja Piskac

ANNOUNCEMENT

https://app1.edoobox.com/en/LRZ/

Registration deadline is today at noon!



Introduction to ANSYS CFX on LRZ HPC Systems	16.03.2023 – 27.04.2023	ONLINE	0.00 €	62	
AI Training Series - Orientation Session	12.04.2023 – 13.04.2023	HYBRID: ONLINE/LRZ	0.00 €	30	
EuroCC AI for Science Bootcamp (apply via openhackathons.org)	17.04.2023 – 18.04.2023	ONLINE	0.00 €	100	
Introduction to LRZ HPC Systems with Focus on CFD Workflows	19.04.2023 – 19.04.2023	ONLINE	0.00 €	51	
AI Training Series - Intro to Container Technology & Application to AI at LRZ	26.04.2023 – 26.04.2023	HYBRID: ONLINE/LRZ	0.00 €	43	
AI Training Series - Introduction to the LRZ AI Systems	03.05.2023 – 03.05.2023	ONLINE	0.00 €	47	
AI Training Series - Intel AI Workshop - Accelerated Machine Learning with Intel	04.05.2023 – 04.05.2023	ONLINE	0.00 €	88	Register now
SuperMUC-NG Status and Results Workshop	09.05.2023 – 11.05.2023	ONLINE	0.00 €	192	Register now
AI Training Series - Introduction to the LRZ Linux Cluster	09.05.2023 – 09.05.2023	HYBRID: ONLINE/LRZ	0.00 €	73	Register now
Working with Noisy Quantum Computers	11.05.2023 – 11.05.2023	Leibniz Rechenzentrum	0.00 €	3	
AI Training Series - High Performance Data Analytics Using R at LRZ	12.05.2023 – 12.05.2023	HYBRID: ONLINE/LRZ	0.00 €	83	Register now
EuroCC N-Ways to GPU Programming Bootcamp (apply via openhackathons.org)	15.05.2023 – 16.05.2023	ONLINE	0.00 €	100	
AI Training Series - Intel AI Workshop - Accelerated Deep Learning with Intel	16.05.2023 – 16.05.2023	HYBRID: ONLINE/LRZ	0.00 €	95	Register now
AI Training Series - Introduction to the LRZ Compute Cloud	17.05.2023 – 17.05.2023	HYBRID: ONLINE/LRZ	0.00 €	85	Register now

Agenda

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 with an Enroot container

2. Fundamentals of Deep Learning

- Introduction to Neural Networks
- Introduction to Convolutional Neural Networks
- Exercises: Train CNNs on a GPU
- Introduction to Transformers
- Exercise: Train a Transformer on a GPU

3. Distributed Training of Neural Networks

- Data Parallel Training
- Exercise: DP Training of CNN on 2 GPUs
- Model Parallel Training: Pipeline Parallel and Tensor Parallel
- Exercise: PP Training of Transformer on 2 GPUs

Planned breaks

- 11:15 – 11:30 Coffee Break I
- 12:30 – 13:30 Lunch break
- 14:45 – 15:00 Coffee Break II

Overview of the LRZ Systems

Data Science Storage
(DSS)

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

LRZ Linux Cluster

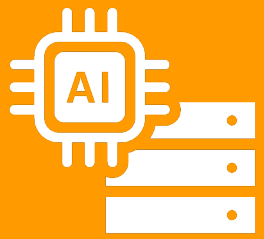
CoolMUC-2 Teramem-2 CoolMUC-3

lxlogin[1-3].lrz.de
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://datalab3.srv.lrz.de

Flexible system that copes with almost any workload

LRZ Compute Cloud

LRZ Compute Cloud
(w/ some GPUs)

https://cc.lrz.de

Overview of the LRZ AI Systems

	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	4	1	1	1	1	4	9
CPU cores per node	256	256	80	80	64	20	40 / 32 / 20
Memory per node	2 TB	1 TB	512 GB	512 GB	256 GB	368 GB	min. 360 GB
GPUs per node	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	80 GB	40 GB (20GB)	16 GB	16 GB	16GB	16 GB	--
SLURM Partition	lrz-dgx-a100-80x8	lrz-dgx-a100-40x8	lrz-dgx-1-v100x8	lrz-dgx-1-p100x8	lrz-hpe-p100x4	lrz-v100x2	lrz-cpu
Nodes	lrz-dgx-a100-[001-002,004-005]	lrz-dgx-a100-003	dgx-002	dgx-001	p100-001	gpu-[001-003,005]	cpu-[001-009]

DGX A100

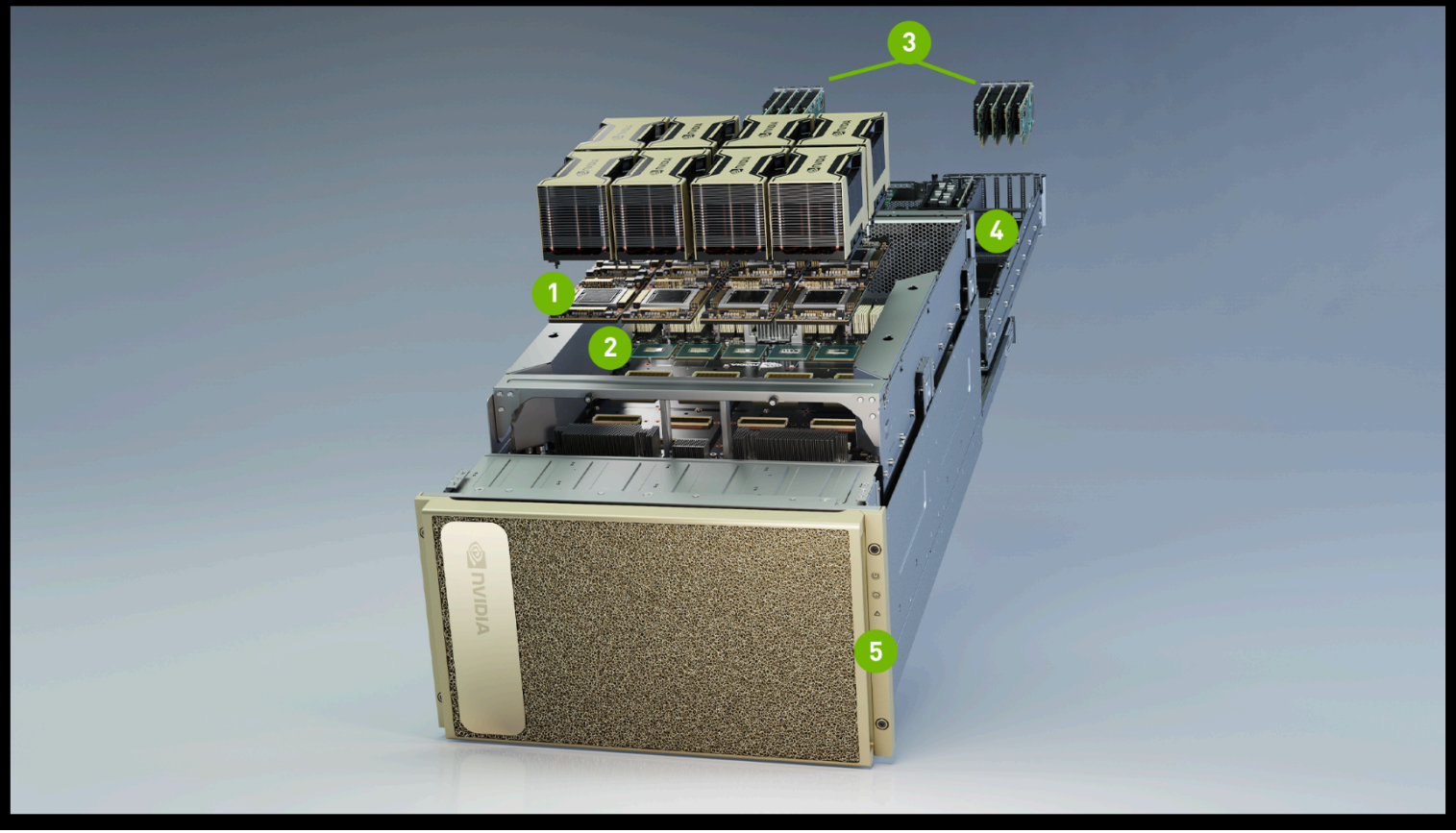
1. **8x NVIDIA A100 GPUs with up to 640GB total GPU memory**
   12 NVIDIA NVLinks® per GPU, 600GB/s of GPU-to-GPU bidirectional bandwidth

2. **6x NVIDIA NVSwitches™**
   4.8TB/s of bidirectional bandwidth, 2X more than previous-generation NVSwitch

3. **10x NVIDIA ConnectX-7 200Gb/s network interface**
   500GB/s of peak bidirectional bandwidth

4. **Dual 64-CORE AMD CPUs and 2TB system memory**
   3.2X more cores to power the most intensive AI jobs

5. **30TB Gen4 NVMe SSD**
   50GB/s of peak bandwidth, 2X faster than Gen3 NVMe SSDs



Storage on the LRZ AI Systems

Storage Pool	Designated Use	Top-level Directory	Size Limit	Automated Backup	Expiration	Additional Information
Home directory	infrequently changing user files, scripts, configuration files, etc. 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 5 TB	No	Until further notice	
Linux Cluster DSS	general purpose, long-term data storage	/dss/dssfs02 /dss/dssfs03	up to 10 TB	No	Lifetime of data project	File Systems and IO on Linux-Cluster
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 /dssmcmlfs01	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. Submitt 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 a description of the intended usage.

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

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

- Resources are allocated and jobs submitted to the partitions using a job scheduler Slurm.

https://slurm.schedmd.com/overview.html

Access to the LRZ AI Systems – sinfo, salloc, srun & scancel

$ ssh --login_name=xxyyyzz login.ai.lrz.de

Executes in the login node login-1

$ sinfo

$ squeue --user=xxyyzzz

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

$ srun --pty bash

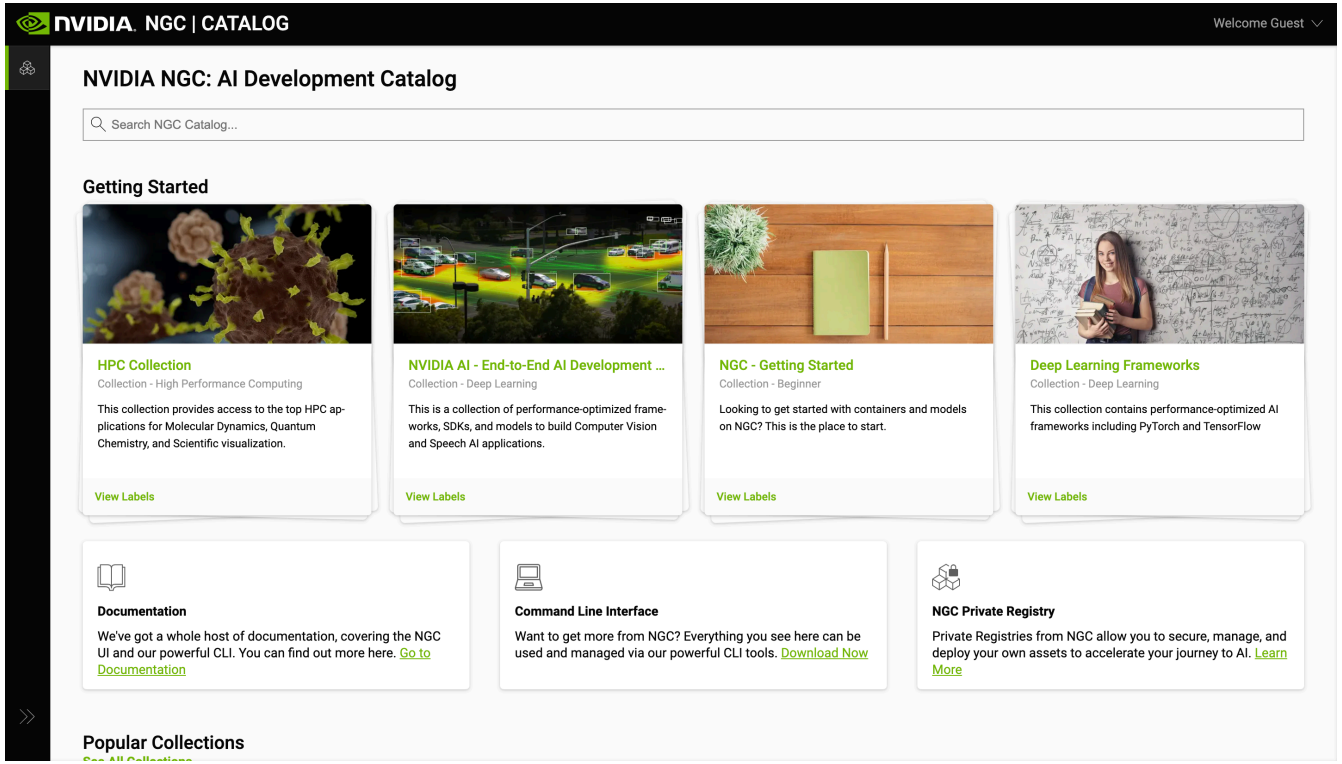
$ scancel job_id

```
di82hod — ssh login.ai.lrz.de — 82×28
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:~$
```

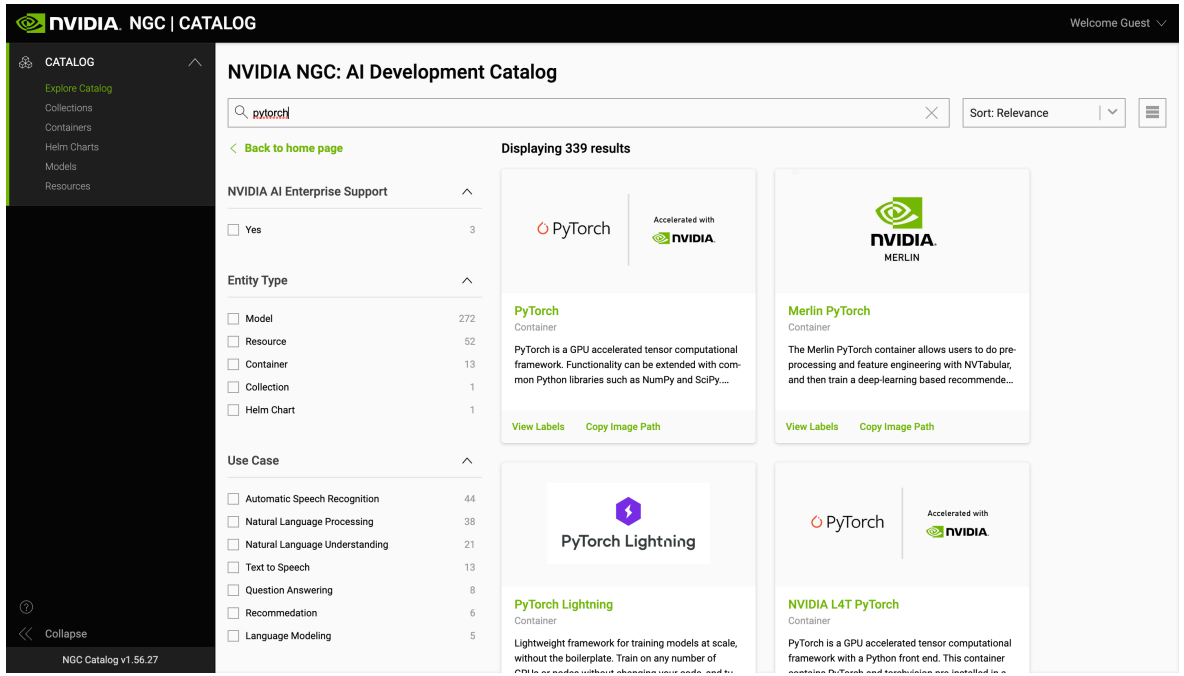
Nvidia NGC Containers

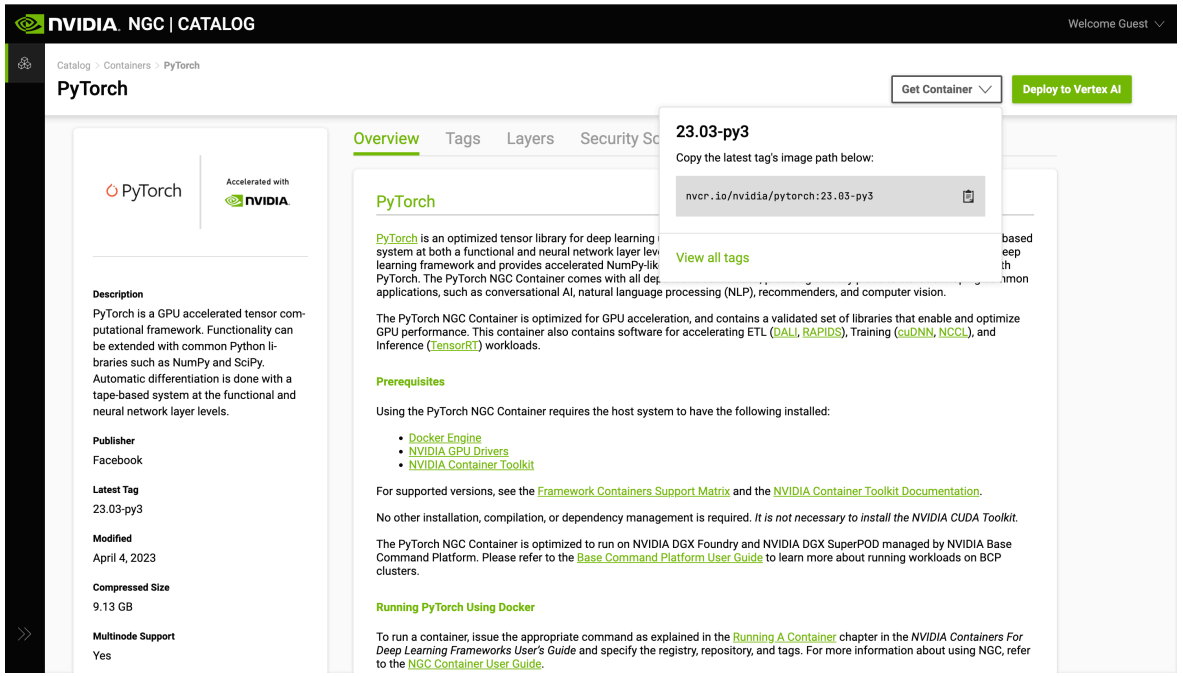
- The NGC catalog provides access to GPU-accelerated software that speeds up end-to-end workflows with performance-optimized containers, pretrained AI models, and SDKs that can be deployed on any NVIDIA's GPU-powered on-prem, cloud and edge systems.

- https://catalog.ngc.nvidia.com

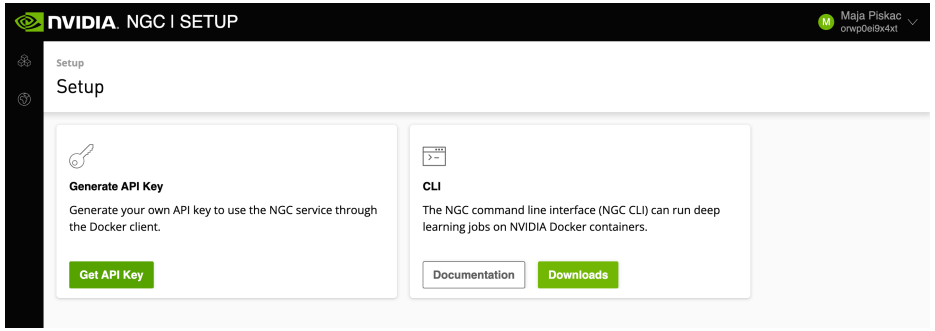


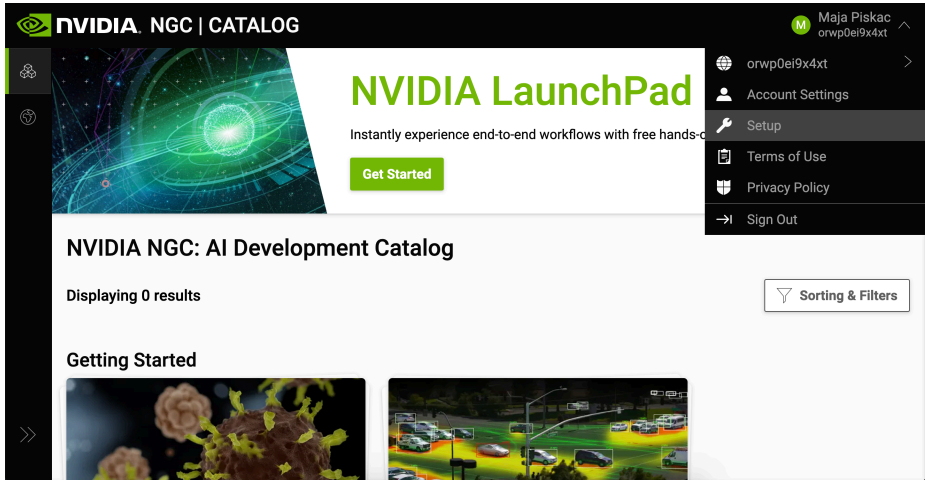
Nvidia NGC Containers

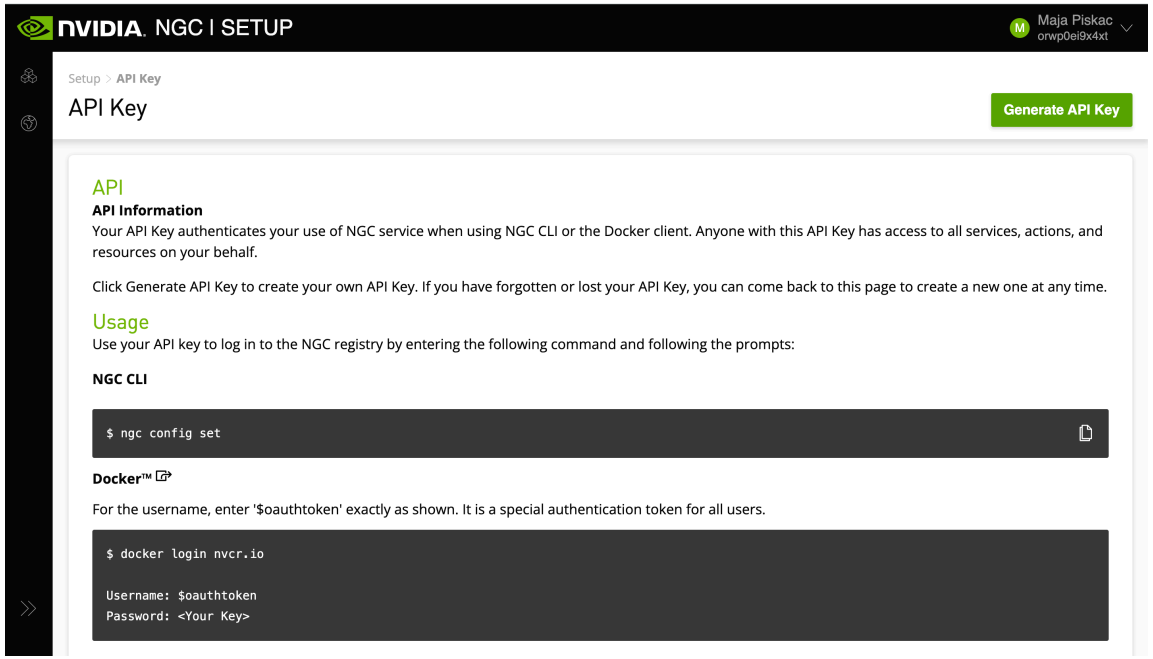




Nvidia NGC Containers – Setting up credentials







Nvidia NGC Containers – Setting up credentials

- Create the file *enroot/.credentials* within your $HOME and append the following line to it:

```
machine nvcr.io login $oauthtoken password <Your Key>
```

- Introduce a new line after <Your Key>

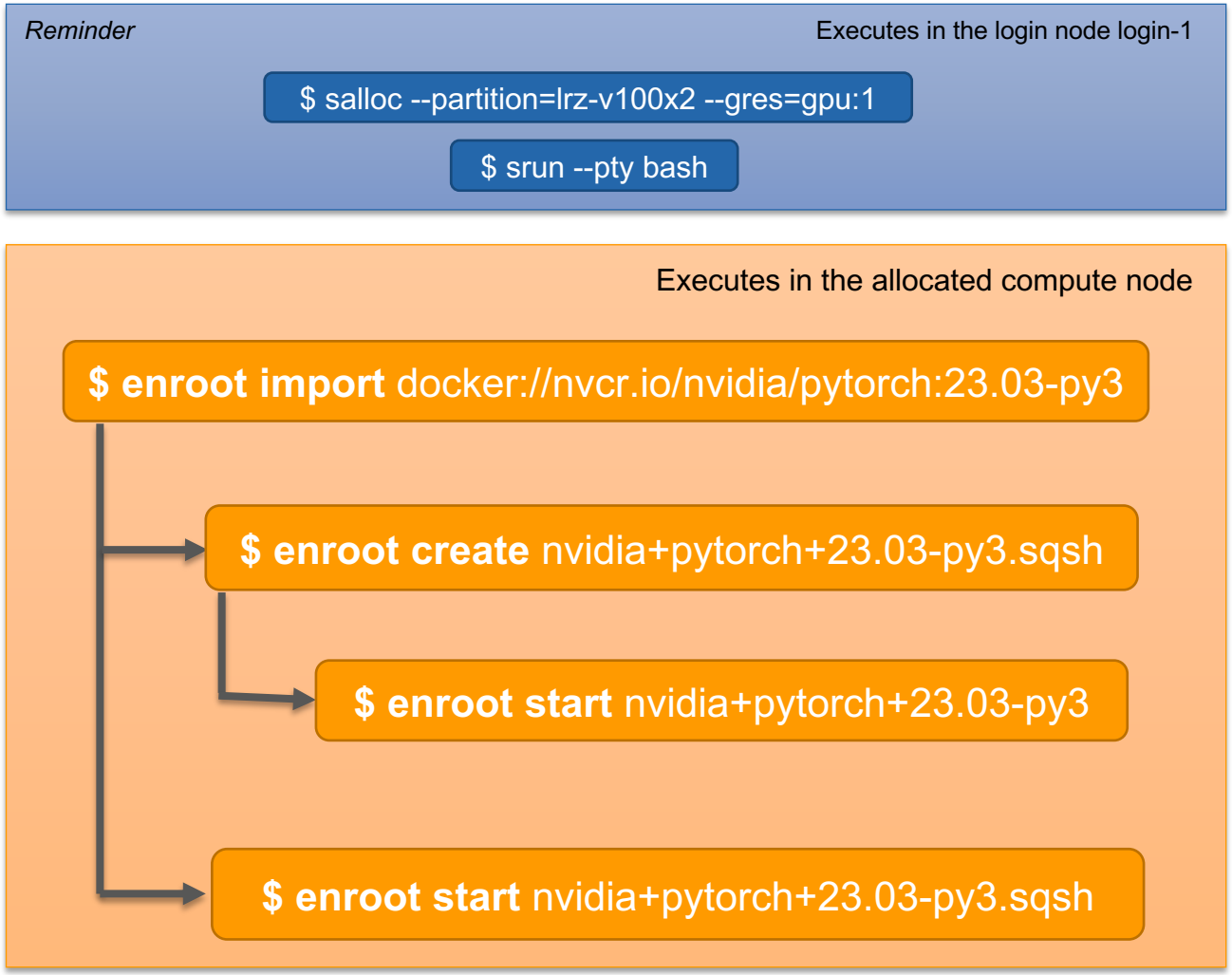
- Now you can import containers from Nvidia NGC, e.g. latest Pytorch container, with the following line:

```
$ enroot import docker://nvcr.io/nvidia/pytorch:23.03-py3
```

- These containers supply the cuda toolkit, cudnn libraries, and Nvidia dependencies.

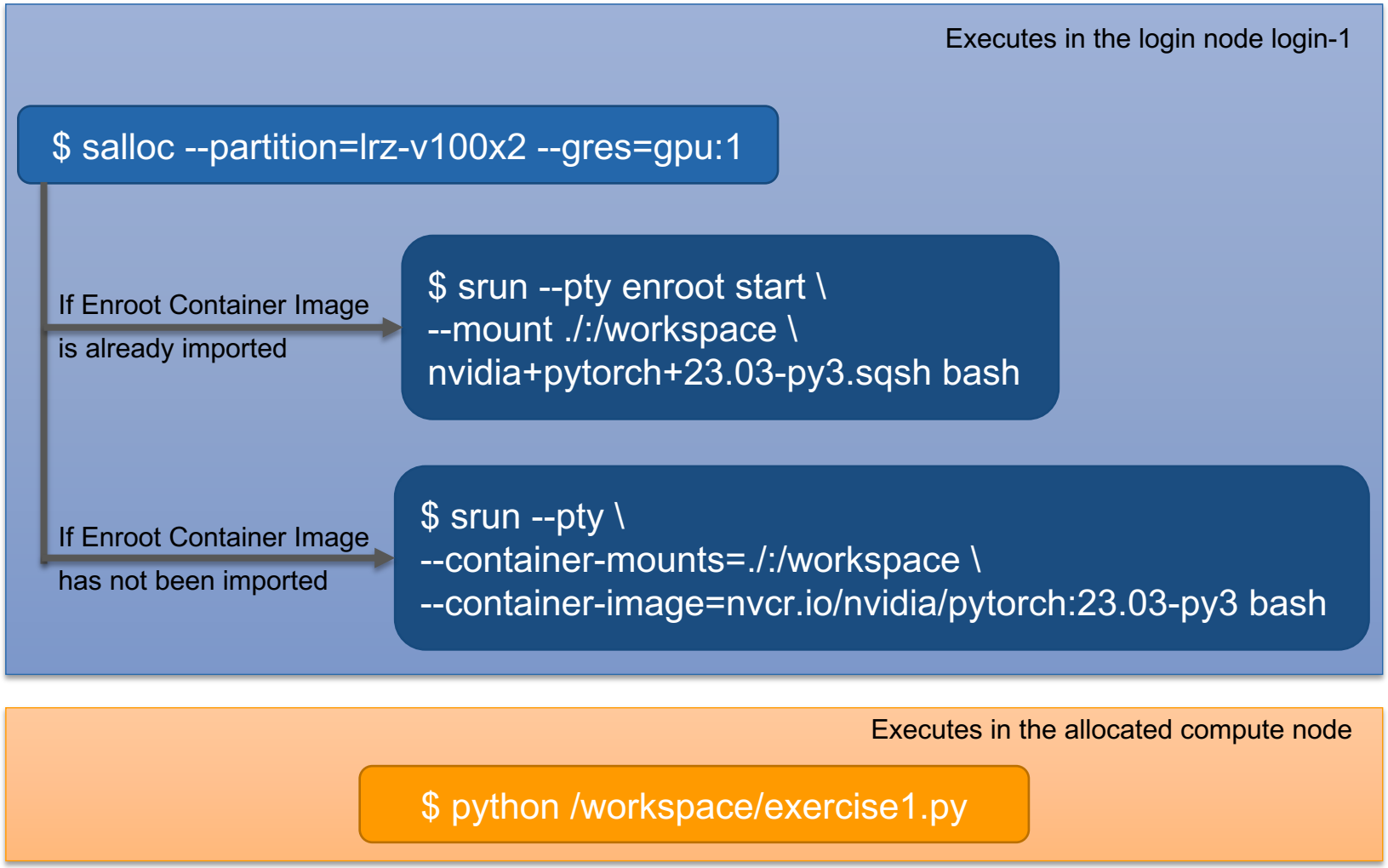
Introduction to Enroot: The Software Stack Provider for the AI Systems

- 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**.



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 the 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 or nodes that you want to use.
- Two additional required arguments: *output* and *error messages* file.

```
#!/bin/bash
#SBATCH -p lrz-dgx-a100-80x8
#SBATCH --gres=gpu:1
#SBATCH -o exercise1.out
#SBATCH -e exercise1.err

srun \
--container-mounts='./:/workspace' \
--container-image='nvcr.io/nvidia/pytorch:23.03-py3' \
python /workspace/exercise1.py
```

Executes in the login node login-1

```
$ sbatch exercise1.sbatch
```

Dealing with base images from catalogues other than NGC

- If your chosen image does not supply the cuda toolkit, do not install it within the image! Let the Enroot runtime deal with it by adding environment variables within the container.

Executes in the allocated resource

```
$ enroot import docker://someone/image-no-cuda
$ enroot create --name base_container someone+image-no-cuda.sqsh
$ enroot start base_container bash
echo "NVIDIA_DRIVER_CAPABILITIES=compute,utility,video" >> /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 plt_container base_image.sqsh
$ enroot start plt_container
$ pip install matplotlib
$ exit
$ enroot export --output pytorch+plt.sqsh plt_container
$ enroot start pytorch+plt.sqsh
```

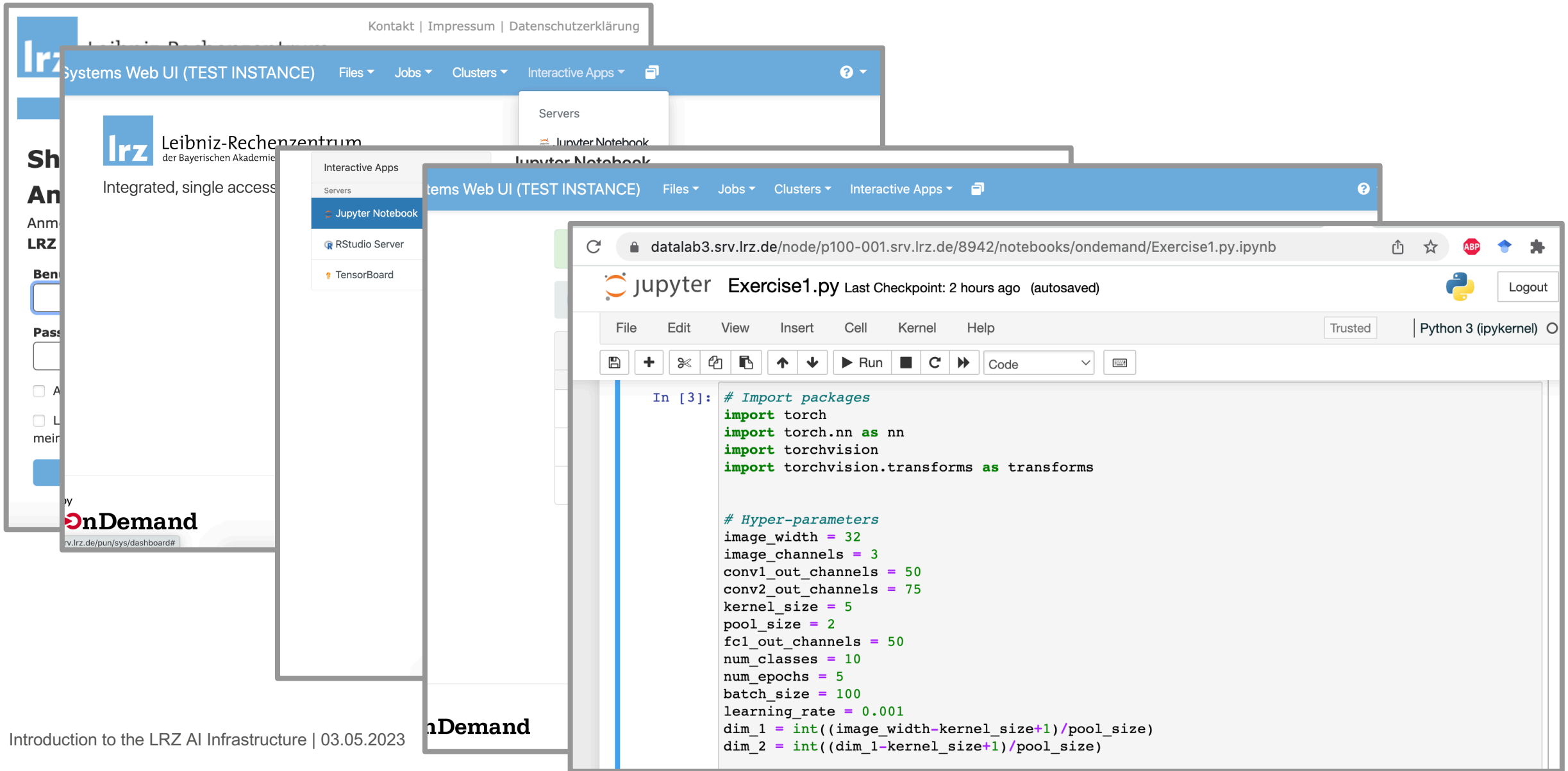
Access to AI Systems through interactive web servers

- Jupyter Notebook, JupyterLab, RStudio Server and TensorBoard

- Available at https://datalab3.srv.lrz.de

- 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
  - 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, any unsaved work will then be lost).

Access to AI Systems through interactive web servers



Public Datasets and Containers on the LRZ AI Systems

- LRZ AI Systems offer a dedicated Data Science Storage (DSS) container aimed at storing public datasets as well as Enroot container images of interest to more than one researcher.

Dataset	Location	Version	Licence
AlphaFold	/dss/dssfs04/pn69za/pn69za-dss-0004/datasets/alphafold/	Last update April 2022 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 12, 2017	Creative Commons Attribution 4.0 International License

Container	Location	Version
AlphaFold	/dss/dssfs04/pn69za/pn69za-dss-0004/containers/shared/alphafold.sqsh	Built from the Dockerfile here https://github.com/deepmind/alphafold/tree/main/docker

Summary

Command line

```
$ ssh -l xxyyyzz login.ai.lrz.de
```

Interactive job

Executes in the login node login-1

```
$ sinfo
```

```
$ salloc --partition=dgx-1-p100 --gres=gpu:1
```

```
$ srun --pty --container-mounts=./:/workspace \
--container-image=nvcr.io/nvidia/pytorch:23.03-py3 bash
```

Executes in the allocated resource

```
$ python /workspace/exercise1.py
```

Batch job

```
#!/bin/bash
#SBATCH -p lrz-dgx-a100-80x8
#SBATCH --gres=gpu:1
#SBATCH -o exercise1.out
#SBATCH -e exercise1.err

srun --container-mounts='./:/workspace' \
--container-image='nvcr.io/nvidia/pytorch:23.03-py3' \
python /workspace/exercise1.py
```

Executes in the login node datalab2

```
$ sbatch exercise1.sbatch
```

Web browser

https://datalab3.srv.lrz.de

Exercise: Start an interactive job and create an Enroot container

1. Ssh into the login node login.ai.lrz.de and download the exercises from https://doku.lrz.de/display/PUBLIC/AI+Infrastructure+Material.

2. Allocate and run a job with 1 GPU.

3. Import a Pytorch container image from NGC (or Pytorch Docker container), create and start the container.

4. Run exercise1.py as an interactive job.

Agenda

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 with an Enroot container

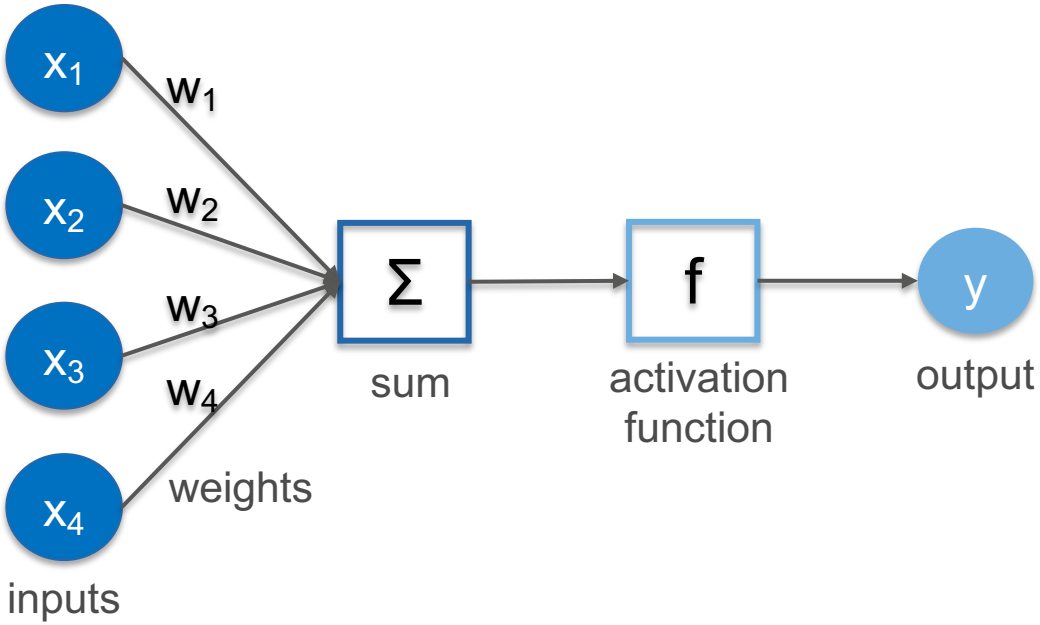
2. Fundamentals of Deep Learning

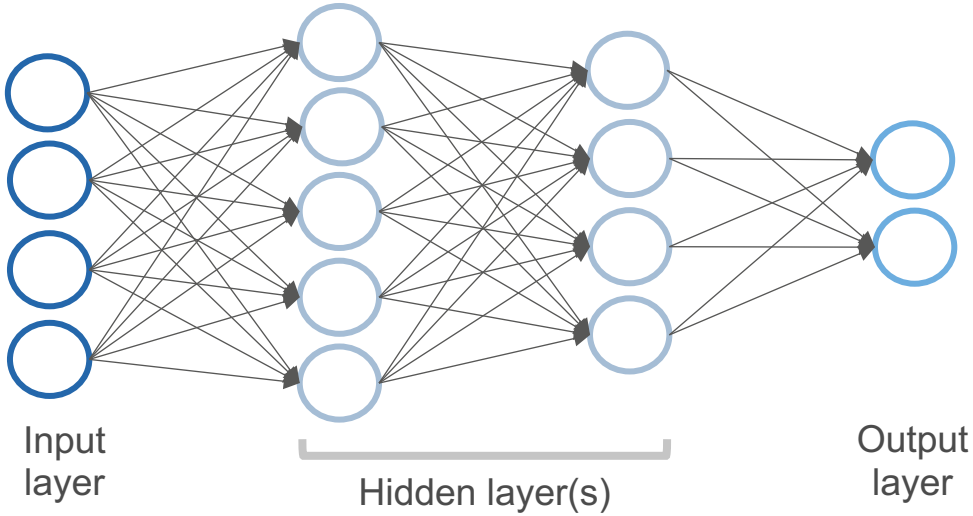
- ❑ Introduction to Neural Networks
- ❑ Introduction to Convolutional Neural Networks
- ❑ Exercises: Train CNNs on a GPU
- ❑ Introduction to Transformers
- ❑ Exercise: Train a Transformer on a GPU

3. Distributed Training of Neural Networks

- ❑ Data Parallel Training
- ❑ Exercise: DP Training of CNN on 2 GPUs
- ❑ Model Parallel Training: Pipeline Parallel and Tensor Parallel
- ❑ Exercise: PP Training of Transformer on 2 GPUs

Introduction to Neural Networks

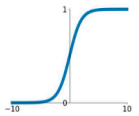




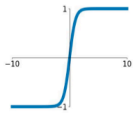
$$y = f(x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4 + b)$$

$$\theta = \{w_1, w_2, w_3, w_4\}$$

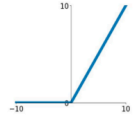
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



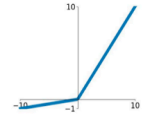
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

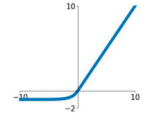


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



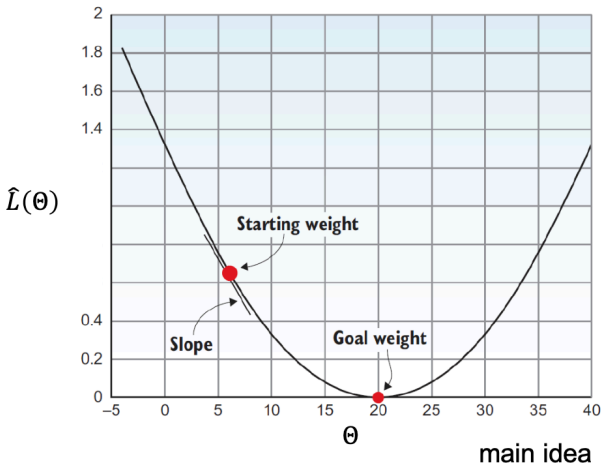
Training Neural Networks

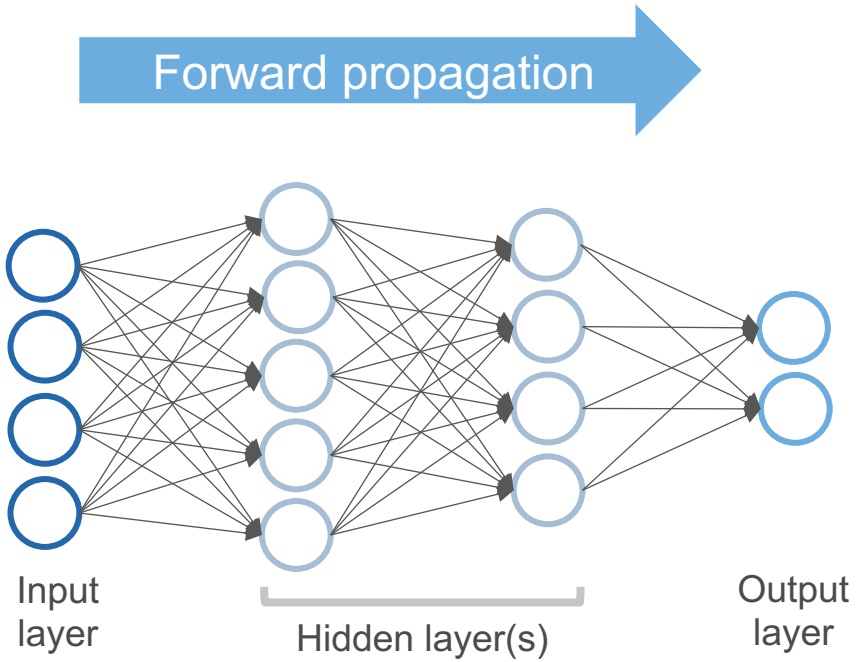
➢ Find weights that minimize the loss function on a given subset of the training data.

Optimizer SGD

$$\theta_{new} = \theta_{old} - \eta * g(\theta_{old})$$

$$g(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla l(\theta; z_i)$$





Loss function

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta; z_i)$$

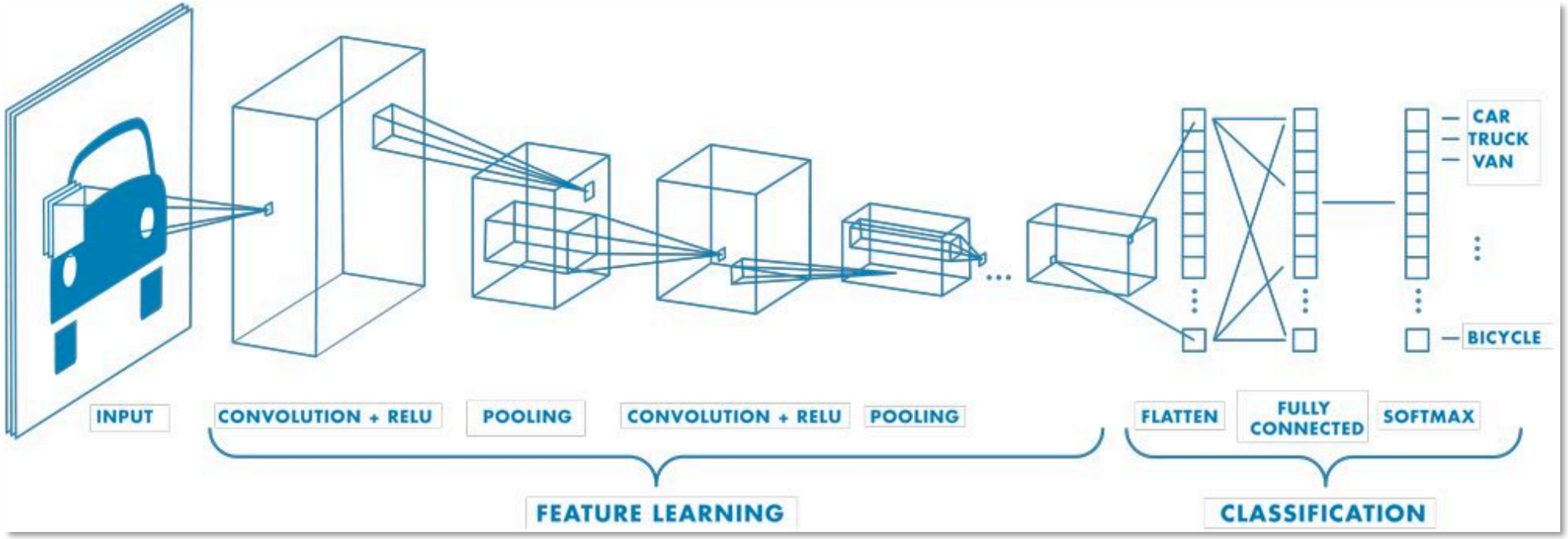
$$MeanSquaredError(y, y^p) = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^p)^2$$

$$CrossEntropy(y, y^p) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{M} y_{i,c} * \log(y_{i,c}^p)$$

Hyper Parameters

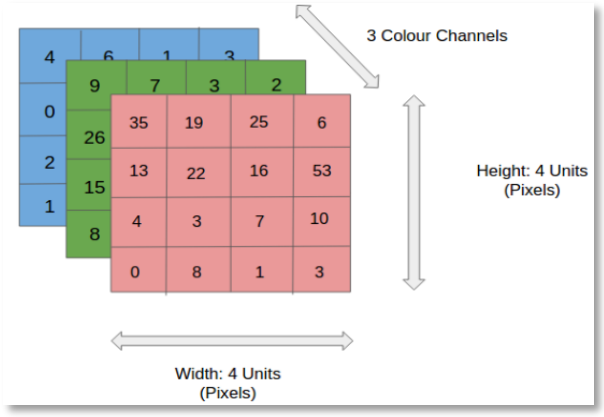
- epochs
- batch size n
- learning rate η

Introduction to Convolutional Neural Networks

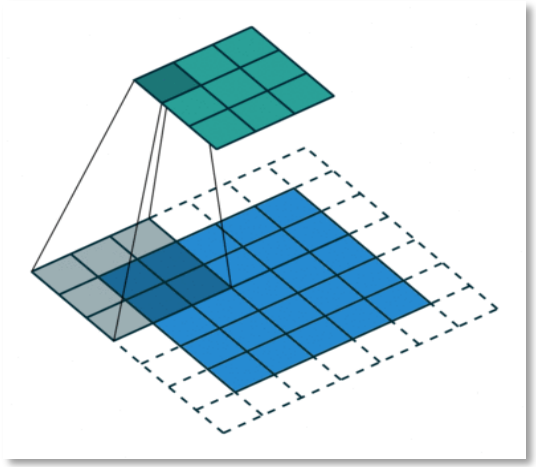


https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

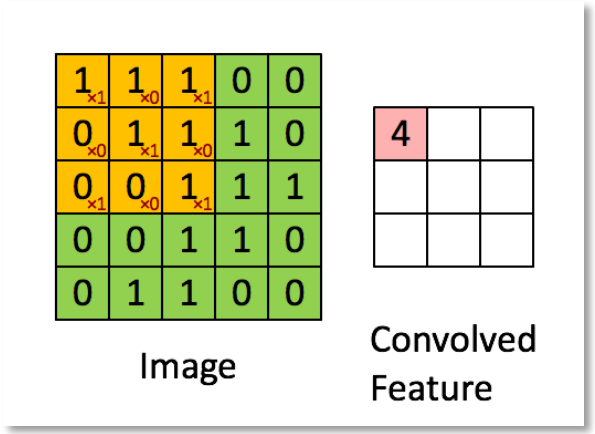
Introduction to Convolutional Neural Networks



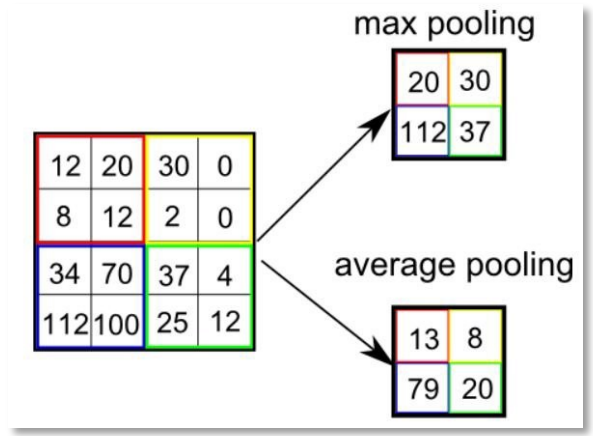
Input image



Convolution with stride=2 and padding=1



Convoluting 5x5x1 image with 3x3x1 kernel to get 3x3x1 convolved feature



Types of pooling

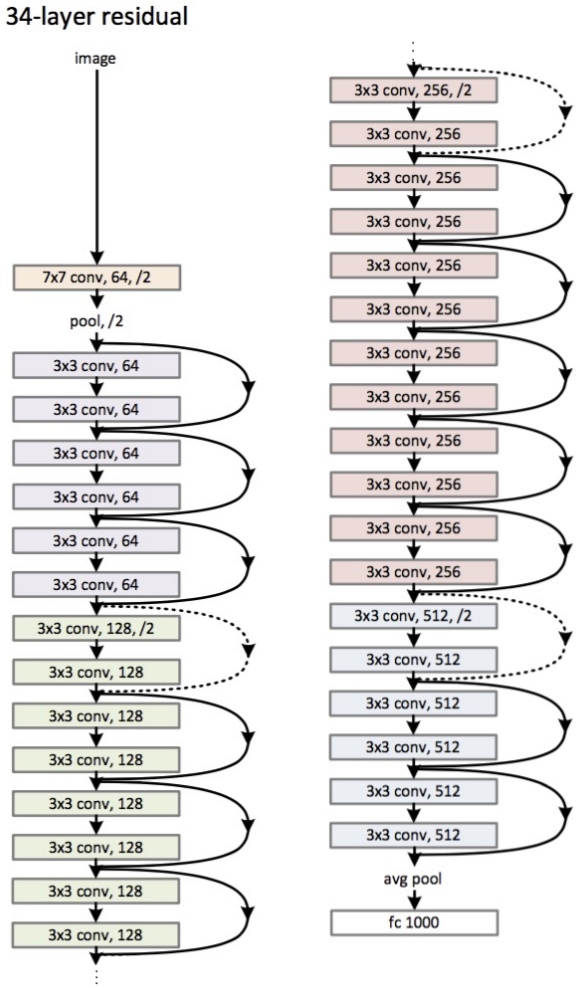
Convolutional Neural Network Architectures – VGG (Visual Geometry Group)

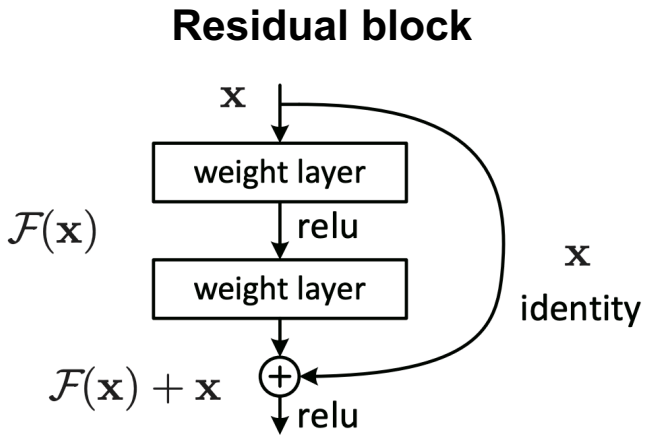
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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 **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

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

Convolutional Neural Network Architectures – ResNet

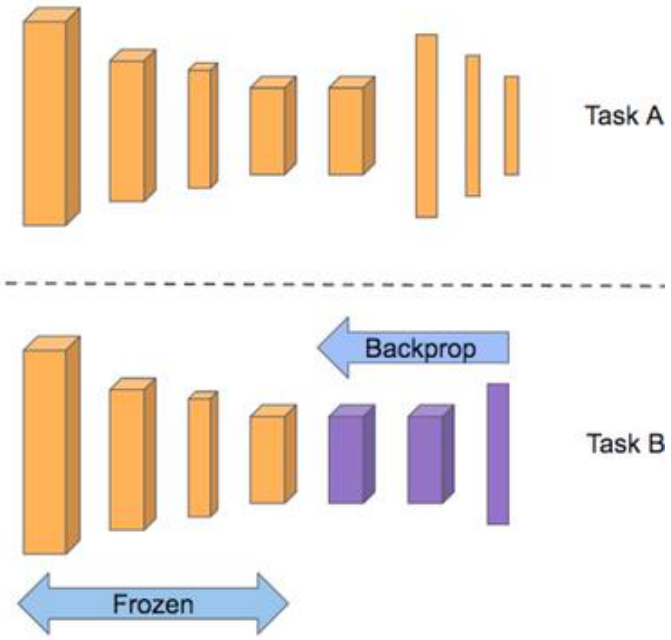




- Main idea: “identity shortcut connection”
- Avoids vanishing gradient problem
- The network gets the input along with the learning on the residual and if the input function was the appropriate function, it can change the weights of the residual function to be zero.
- ImageNet dataset

https://arxiv.org/pdf/1512.03385.pdf

Transfer learning – Pretrained Convolutional Neural Networks



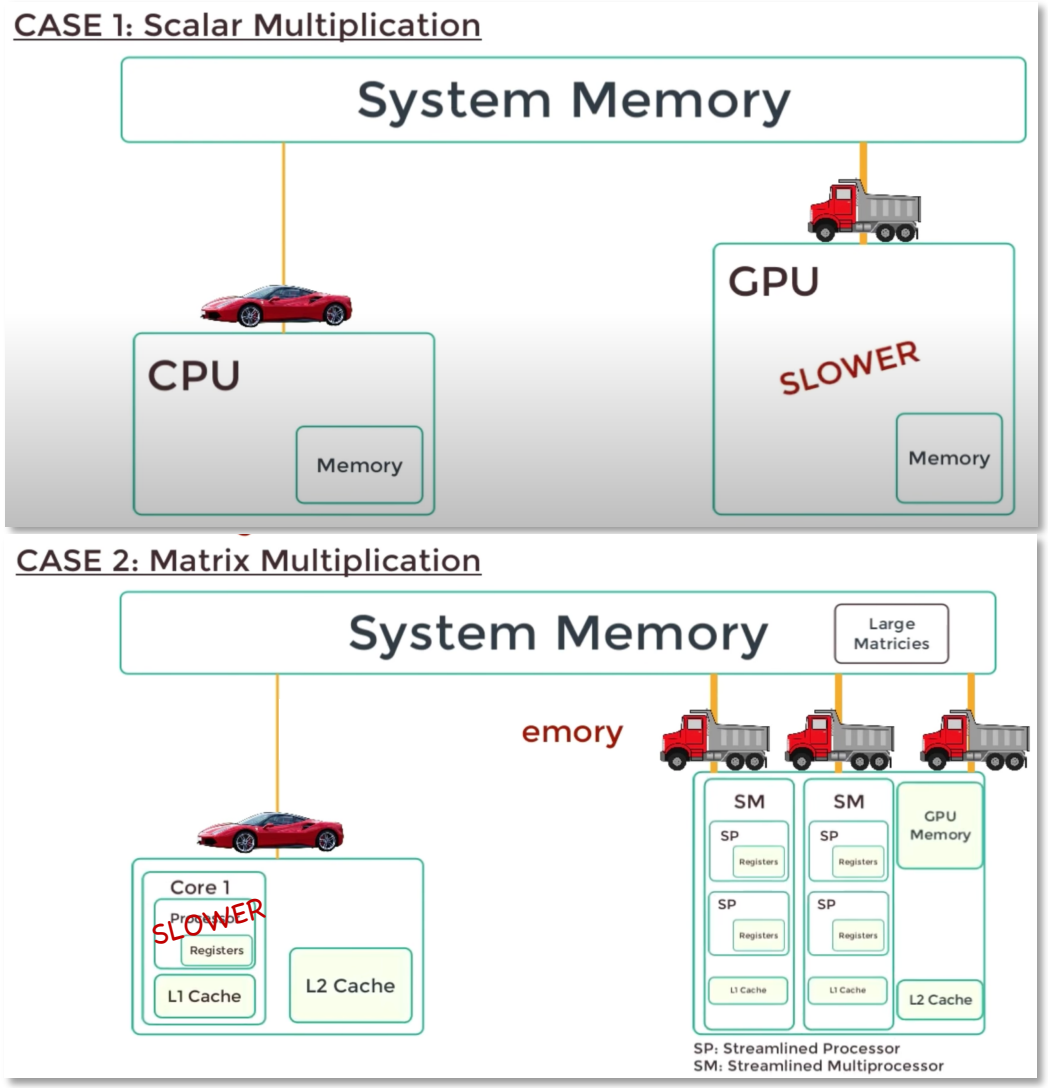
- Pre-trained model is a saved network that was previously trained on a large dataset.
- Feature Extraction: Use the feature maps from the pre-trained model to detect features in the new samples. Add a new classifier, which will be trained from scratch to make predictions.
- Fine-Tuning: Unfreeze a few 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')
```

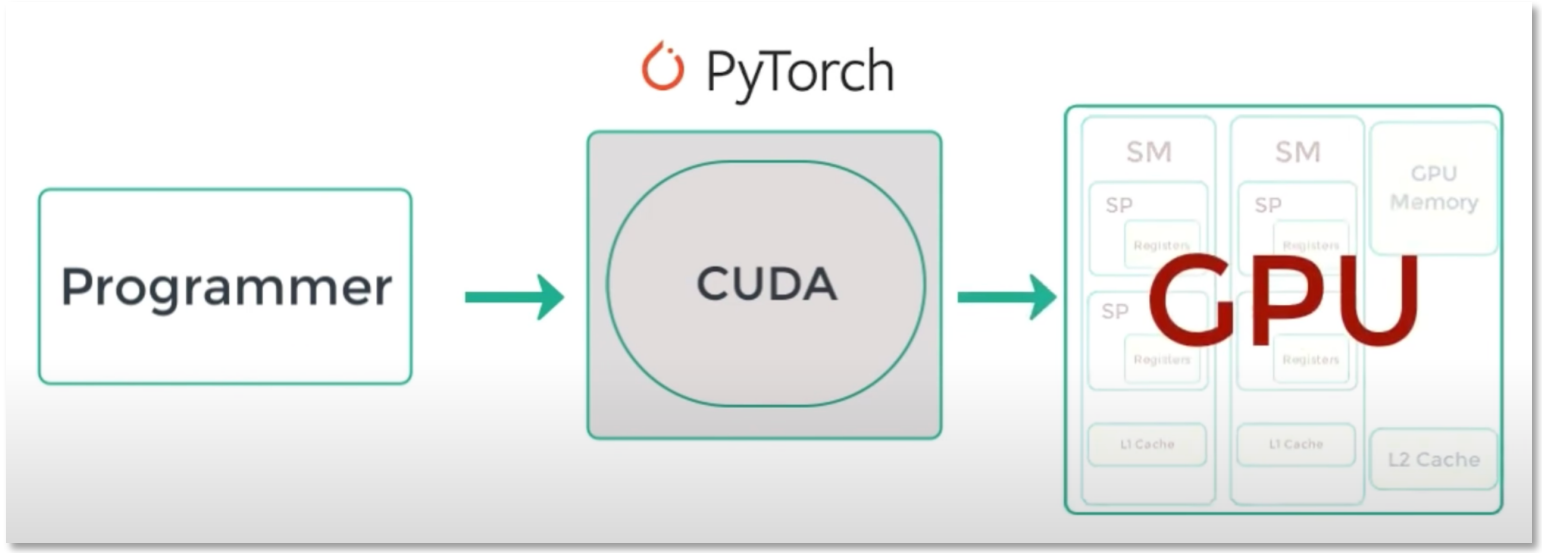
```
import torchvision.models as models
model = models.vgg19(weights='IMAGENET1K_V1')
```

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.



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)
```

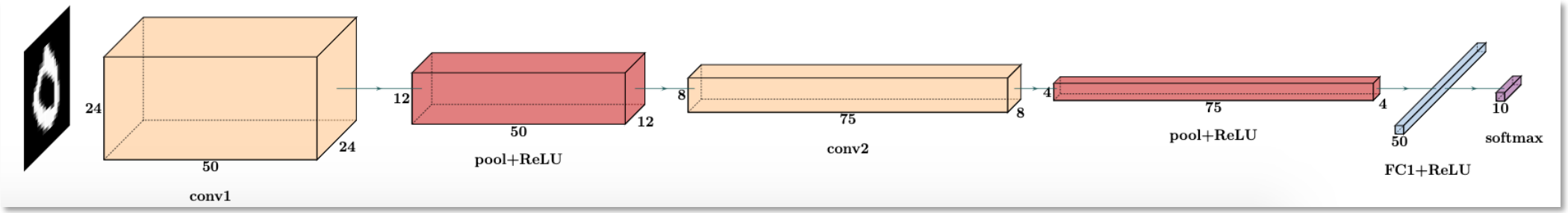
CNN Exercises in Pytorch

Simple Convolutional Neural Network

```
class Model(nn.Module):
    def __init__(self, image_width, image_channels, num_classes):
        super().__init__()
        self.conv1 = nn.Conv2d(image_channels, conv1_out_channels, kernel_size)
        self.conv2 = nn.Conv2d(conv1_out_channels, conv2_out_channels, kernel_size)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(pool_size)
        self.fc1 = nn.Linear(conv2_out_channels*(dim_2**2), fc1_out_channels)
        self.fc2 = nn.Linear(fc1_out_channels, num_classes)

    def forward(self, x):
        x = self.relu(self.pool(self.conv1(x)))
        x = self.relu(self.pool(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

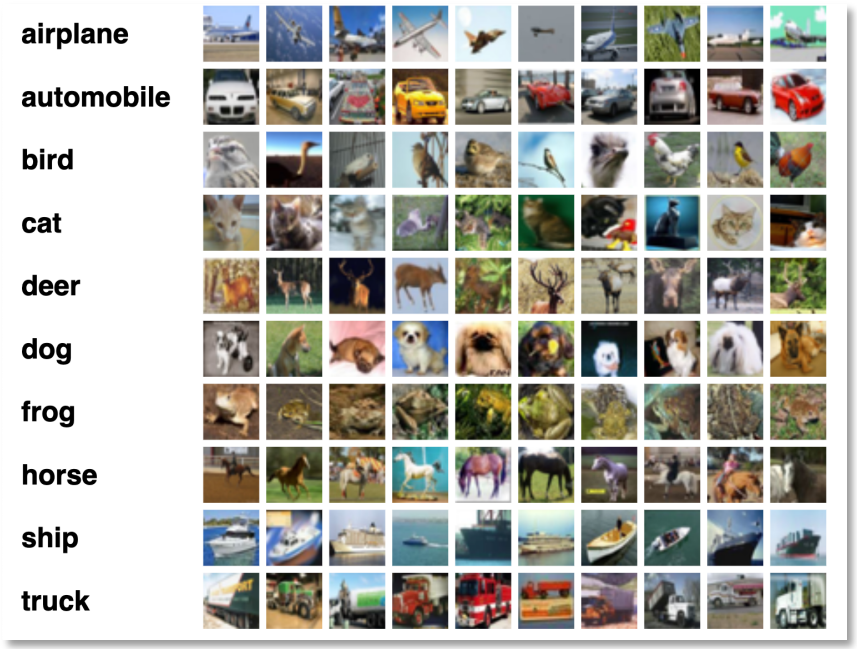
```
# Hyper-parameters
image_width = 28
image_channels = 1
conv1_out_channels = 50
conv2_out_channels = 75
kernel_size = 5
pool_size = 2
fc1_out_channels = 50
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```



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

https://www.cs.toronto.edu/~kriz/cifar.html

exercise1.py: Simple Convolutional Neural Network on a CPU vs on a GPU (155685 parameters)

Output:

Device: cpu

Epoch [2/20], Step [49920/50000], Loss: 1.3028
Epoch [4/20], Step [49920/50000], Loss: 1.3503
Epoch [6/20], Step [49920/50000], Loss: 0.9295
Epoch [8/20], Step [49920/50000], Loss: 0.8239
Epoch [10/20], Step [49920/50000], Loss: 0.7265
Epoch [12/20], Step [49920/50000], Loss: 0.6835
Epoch [14/20], Step [49920/50000], Loss: 0.8085
Epoch [16/20], Step [49920/50000], Loss: 0.6006
Epoch [18/20], Step [49920/50000], Loss: 0.5492
Epoch [20/20], Step [49920/50000], Loss: 0.7049

Test set: Accuracy: 69.6 %

Elapsed time for training and testing: **396.54 s**

(396.54 s ≈ **6.6 min**)

Output:

Device: cuda

Epoch [2/20], Step [49920/50000], Loss: 1.1545
Epoch [4/20], Step [49920/50000], Loss: 1.0863
Epoch [6/20], Step [49920/50000], Loss: 1.0183
Epoch [8/20], Step [49920/50000], Loss: 0.8718
Epoch [10/20], Step [49920/50000], Loss: 0.6980
Epoch [12/20], Step [49920/50000], Loss: 0.6620
Epoch [14/20], Step [49920/50000], Loss: 0.6058
Epoch [16/20], Step [49920/50000], Loss: 0.6141
Epoch [18/20], Step [49920/50000], Loss: 0.4936
Epoch [20/20], Step [49920/50000], Loss: 0.5196

Test set: Accuracy: 69.8 %

Elapsed time for training and testing: **130.16 s**

(130.16 s ≈ **2 min**)

exercise1_resnet34.py: Pretrained ResNet34 on CIFAR10 data on a CPU vs on a GPU

(21.8M parameters)

Output:

Device: cpu

Epoch [2/20], Step [49920/50000], Loss: 1.2576
Epoch [4/20], Step [49920/50000], Loss: 0.9185
Epoch [6/20], Step [49920/50000], Loss: 0.8305
Epoch [8/20], Step [49920/50000], Loss: 0.6098
Epoch [10/20], Step [49920/50000], Loss: 0.5429
Epoch [12/20], Step [49920/50000], Loss: 0.4359
Epoch [14/20], Step [49920/50000], Loss: 0.2560
Epoch [16/20], Step [49920/50000], Loss: 0.2965
Epoch [18/20], Step [49920/50000], Loss: 0.3191
Epoch [20/20], Step [49920/50000], Loss: 0.3405

Test set: Accuracy: 80.52 %

Elapsed time for training and testing: **2685.67 s**

(2685.67 s ≈ **45 min**)

Output:

Device: cuda

Epoch [2/20], Step [49920/50000], Loss: 0.6397
Epoch [4/20], Step [49920/50000], Loss: 0.4903
Epoch [6/20], Step [49920/50000], Loss: 0.2941
Epoch [8/20], Step [49920/50000], Loss: 0.1125
Epoch [10/20], Step [49920/50000], Loss: 0.1250
Epoch [12/20], Step [49920/50000], Loss: 0.0748
Epoch [14/20], Step [49920/50000], Loss: 0.0519
Epoch [16/20], Step [49920/50000], Loss: 0.1285
Epoch [18/20], Step [49920/50000], Loss: 0.0511
Epoch [20/20], Step [49920/50000], Loss: 0.0567

Test set: Accuracy: 81.26 %

Elapsed time for training and testing: **325.72 s**

(325.72 s ≈ **5.4 min**)

exercise1_vgg19.py: Pretrained VGG19 on CIFAR10 data on a CPU vs on a GPU
(143.7M parameters)

Output:

Device: cpu

Epoch [2/20], Step [49920/50000], Loss: 1.2591
Epoch [4/20], Step [49920/50000], Loss: 0.8687
Epoch [6/20], Step [49920/50000], Loss: 0.6782
Epoch [8/20], Step [49920/50000], Loss: 0.3366
Epoch [10/20], Step [49920/50000], Loss: 0.5239
Epoch [12/20], Step [49920/50000], Loss: 0.1858
Epoch [14/20], Step [49920/50000], Loss: 0.2304
Epoch [16/20], Step [49920/50000], Loss: 0.3175
Epoch [18/20], Step [49920/50000], Loss: 0.1343
Epoch [20/20], Step [49920/50000], Loss: 0.2626

Test set: Accuracy: 82.61 %
Elapsed time for training and testing: **7786.24 s**

(7786.24 s ≈ **130 min**)

Output:

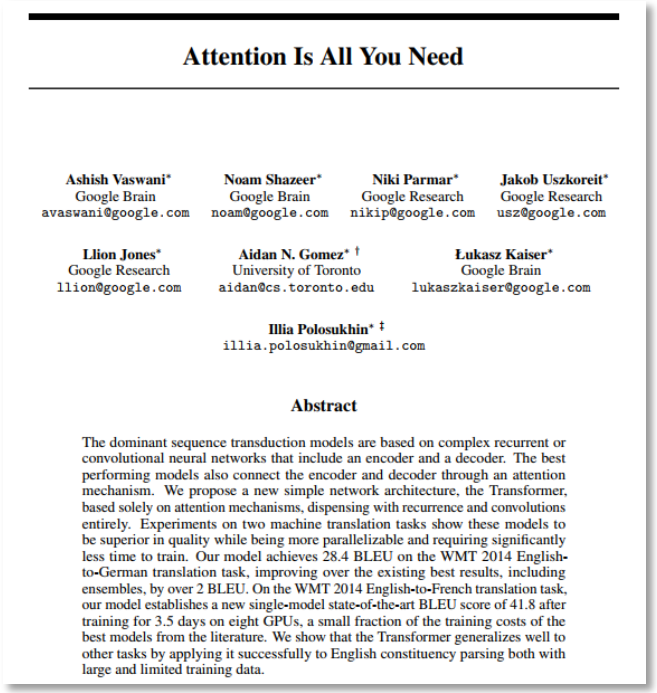
Device: cuda

Epoch [2/20], Step [49920/50000], Loss: 1.2958
Epoch [4/20], Step [49920/50000], Loss: 0.6183
Epoch [6/20], Step [49920/50000], Loss: 0.6047
Epoch [8/20], Step [49920/50000], Loss: 0.3839
Epoch [10/20], Step [49920/50000], Loss: 0.4468
Epoch [12/20], Step [49920/50000], Loss: 0.2408
Epoch [14/20], Step [49920/50000], Loss: 0.2059
Epoch [16/20], Step [49920/50000], Loss: 0.2823
Epoch [18/20], Step [49920/50000], Loss: 0.1345
Epoch [20/20], Step [49920/50000], Loss: 0.1415

Test set: Accuracy: 82.84 %
Elapsed time for training and testing**: 383.44 s**

(383.44 s ≈ **6.4 min**)

Transformers

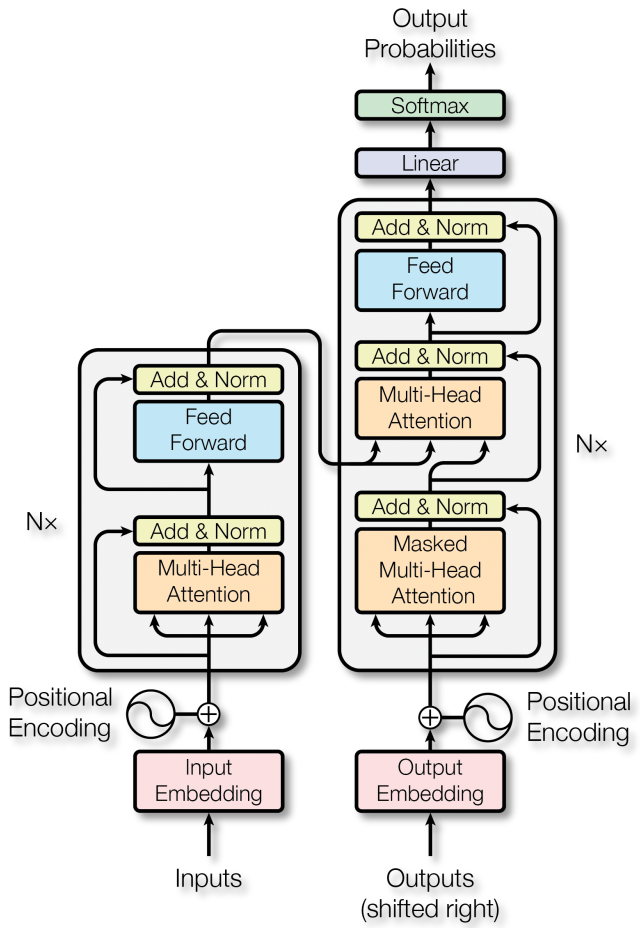


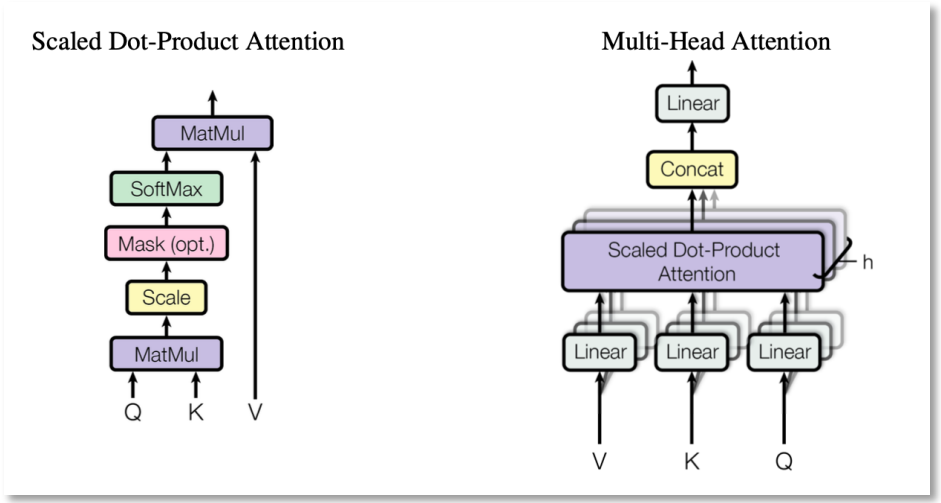
Attention Is All You Need

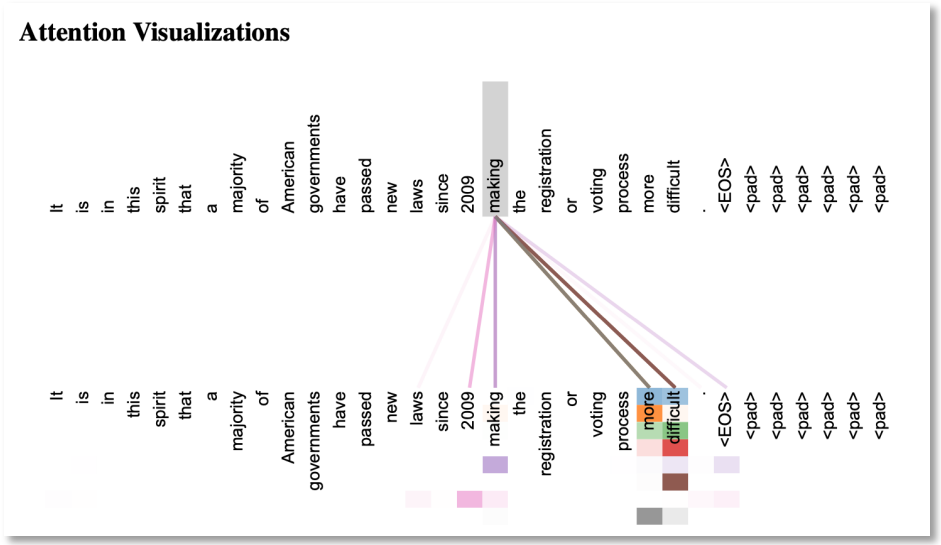
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.







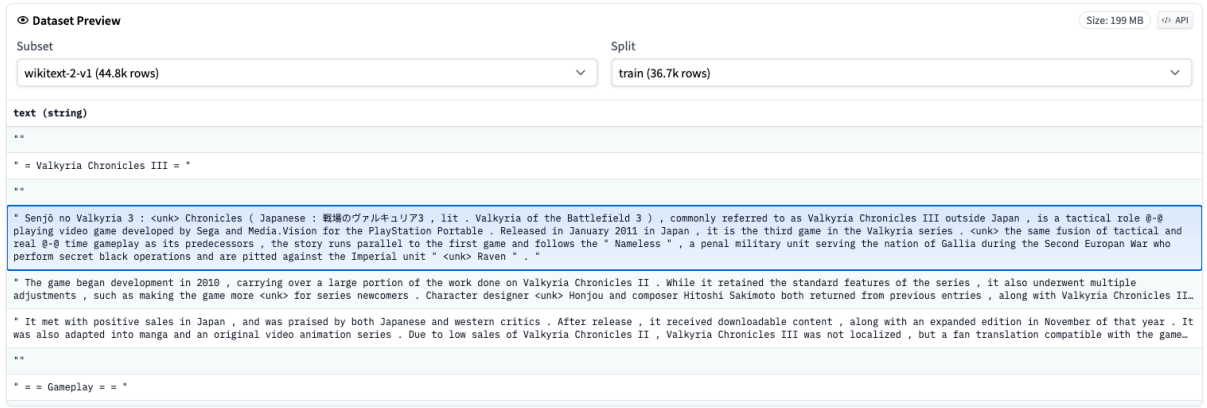
Transformer Exercise

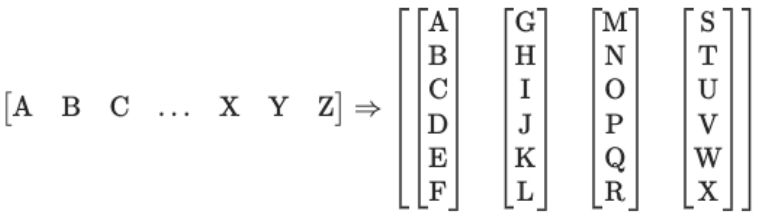
```
class TransformerModel(nn.Module):
    def __init__(self, ntoken, d_model, nhead, d_hid, nlayers, dropout):
        super().__init__()
        self.encoder = nn.Embedding(ntoken, d_model)
        self.d_model = d_model
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        encoder_layers = nn.TransformerEncoderLayer(d_model, nhead, d_hid, dropout)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layers, nlayers)
        self.decoder = nn.Linear(d_model, ntoken)

    def forward(self, src, src_mask):
        """ src: Tensor, shape ``[seq_len, batch_size]``
            src_mask: Tensor, shape ``[seq_len, seq_len]``
            output: Tensor, shape ``[seq_len, batch_size, ntoken]`` """
        src = self.encoder(src) * math.sqrt(self.d_model)
        src = self.pos_encoder(src)
        output = self.transformer_encoder(src, src_mask)
        output = self.decoder(output)
        return output
```

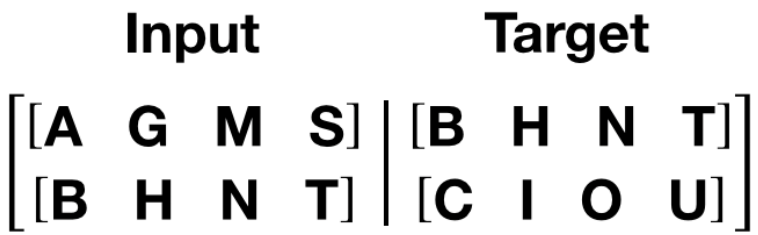
- Training a sequence-to-sequence model based on **nn.Transformer**.
- The nn.Transformer module is modularized – a single component, like TransformerEncoder can be easily adapted / composed.
- We train a **nn.TransformerEncoder** to assign a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words.
- Square attention mask is required with the input sequence, because the self-attention layers are only allowed to see the earlier positions in the sequence.
- PositionalEncoding class is defined to use sine and cosine functions of different frequencies to inject the information on the position of the tokens in the input sequence.

WikiText-2 Data

- Using **torchtext** to generate **WikiText-2** dataset.
- WikiText language modeling dataset is a collection of tokens extracted from the set of verified good and featured Wikipedia articles.
- Number of lines per split:
  - train 36718, valid 3760, test 4358.
- Given a 1-D vector of sequential data, function **batchify()** arranges it into batch_size columns and trims to fit. Batching enables more parallelizable processing, however the model treats each column independently, e.g., the dependence of G and F on the example on the right.
- Function **get_batch()** generates a pair of input-target sequences for the transformer model. It subdivides the source data into chunks of length bptt, e.g., with bptt=2, we'd get the input and target as shown on the right.







https://pytorch.org/tutorials/beginner/transformer_tutorial.html

exercise1_nlp.py: Transformer on a CPU vs on a GPU (77,879,406)

- Embedding dimension of 1024, hidden size of 1024, 4 attention heads and 3 transformer layers.

Output:

Device: cpu

| end of epoch 1 | time: 1618.58s | valid loss 5.75 | valid ppl 313.30
| end of epoch 2 | time: 1635.81s | valid loss 5.69 | valid ppl 296.32
| end of epoch 3 | time: 1493.84s | valid loss 5.57 | valid ppl 263.73
| End of training | test loss 5.48 | test ppl 240.33

(4748.23 s ≈ **80 min**)

Output:

Device: cuda

| end of epoch 1 | time: 79.06s | valid loss 5.75 | valid ppl 314.83
| end of epoch 2 | time: 70.25s | valid loss 5.58 | valid ppl 265.83
| end of epoch 3 | time: 69.92s | valid loss 5.49 | valid ppl 242.84
| End of training | test loss 5.40 | test ppl 221.61

(219.23 s ≈ **3.7 min**)

Exercise: Start an interactive job and create an extended Enroot image

1. Extend the existing Pytorch container with the following python packages:
   - pip install --no-cache-dir torchdata portalocker torchvision
     (for exercise1_nlp.py, Transformer exercises)
   - HOROVOD_GPU_OPERATIONS=NCCL pip install --no-cache-dir horovod
     (for exercise2.py, Horovod exercise which will be shown later in the slides)

2. Run exercises exercise1_resnet34.sbatch, exercise1_vgg19.sbatch, and/or exercise1_nlp.sbatch as batch jobs.

3. Run exercise exercise1_nlp.ipynb as a Jupyter Notebook on https://datalab3.srv.lrz.de/.

Agenda

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 with an Enroot container

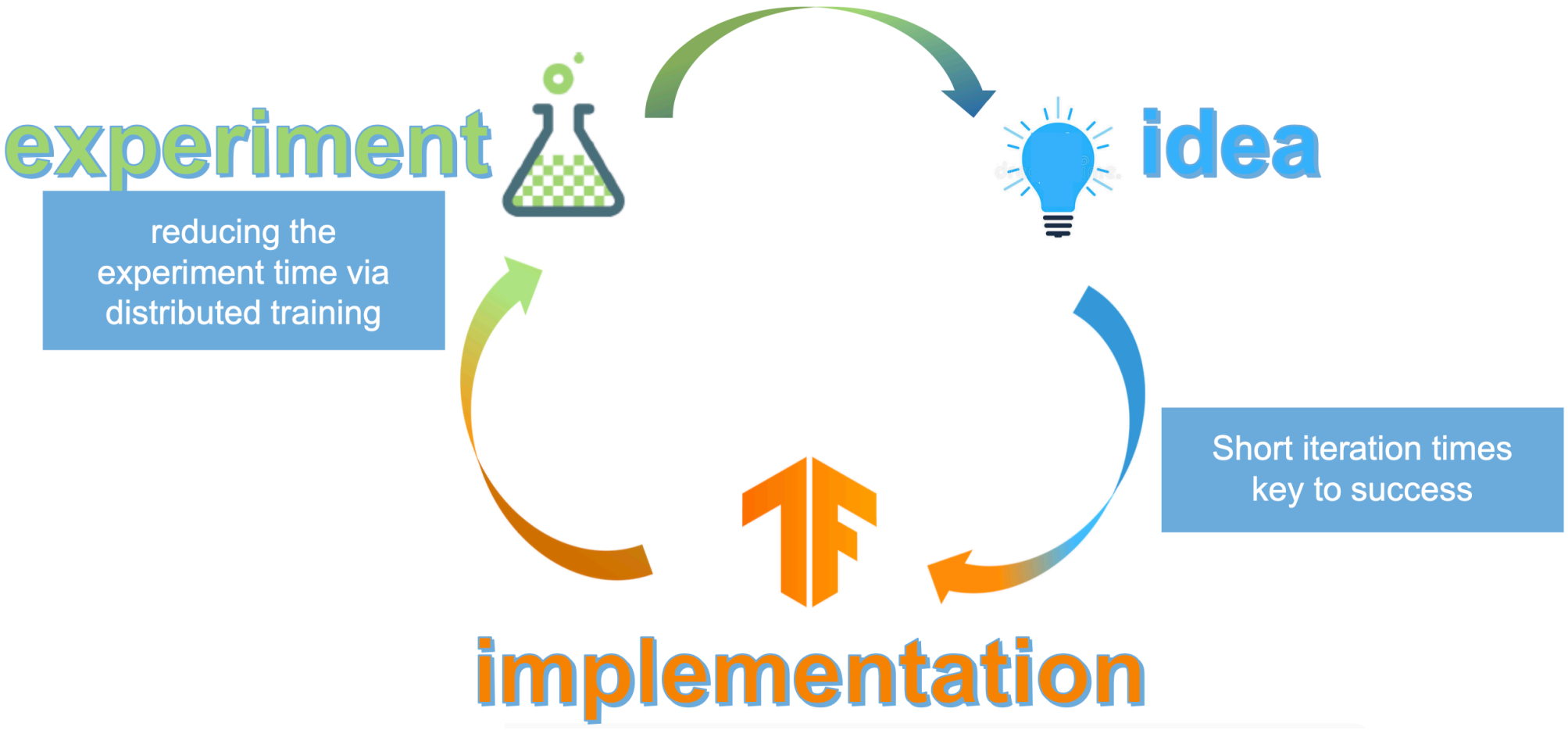
2. Fundamentals of Deep Learning

- ✓ Introduction to Neural Networks
- ✓ Introduction to Convolutional Neural Networks
- ✓ Exercises: Train CNNs on a GPU
- ✓ Introduction to Transformers
- ✓ Exercise: Train a Transformer on a GPU

3. Distributed Training of Neural Networks

- ❑ Data Parallel Training
- ❑ Exercise: DP Training of CNN on 2 GPUs
- ❑ Model Parallel Training: Pipeline Parallel and Tensor Parallel
- ❑ Exercise: PP Training of Transformer on 2 GPUs

Experimental Science Requires Short Iteration Times



Trends of Parallel Architectures in Deep Learning

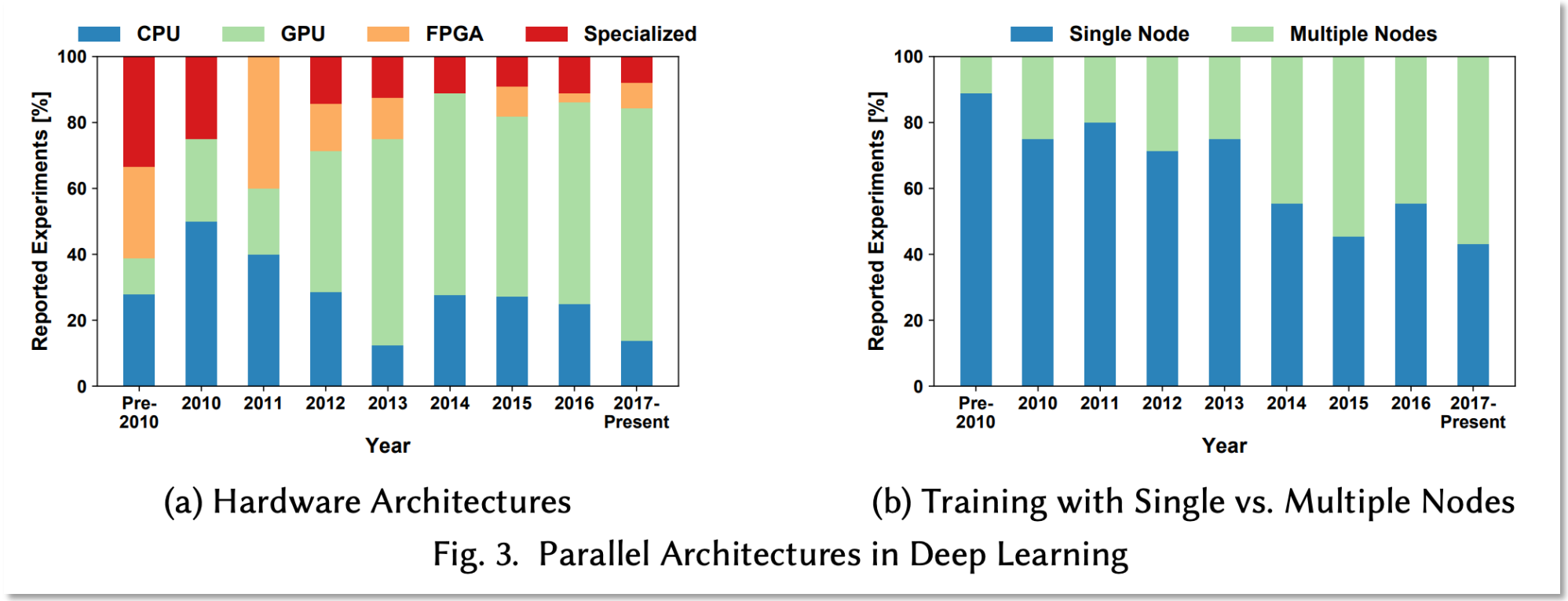


Fig. 3. Parallel Architectures in Deep Learning

Source: Ben-Nun and Hoefler, 2018, arXiv:1802.09941

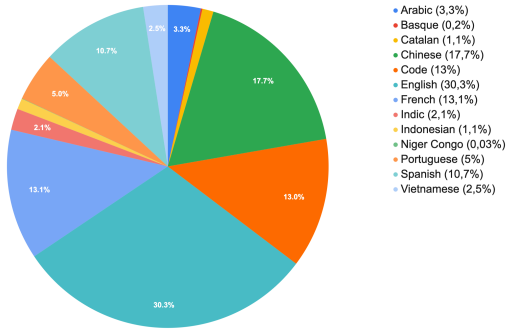
Increasing Amount of Data Available for Deep Learning

ImageNet

14,197,122 images
(150 GB)

https://paperswithcode.com/dataset/imagenet

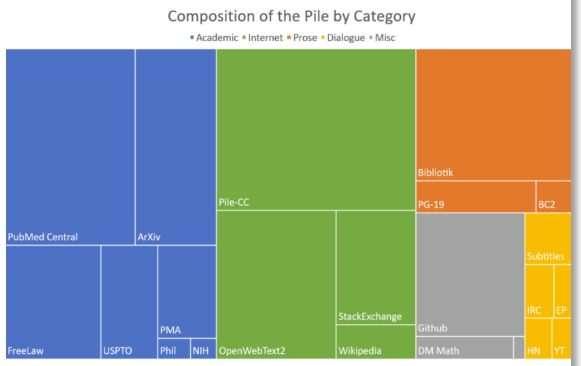
BigScience Multilingual Dataset for Language Modeling



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

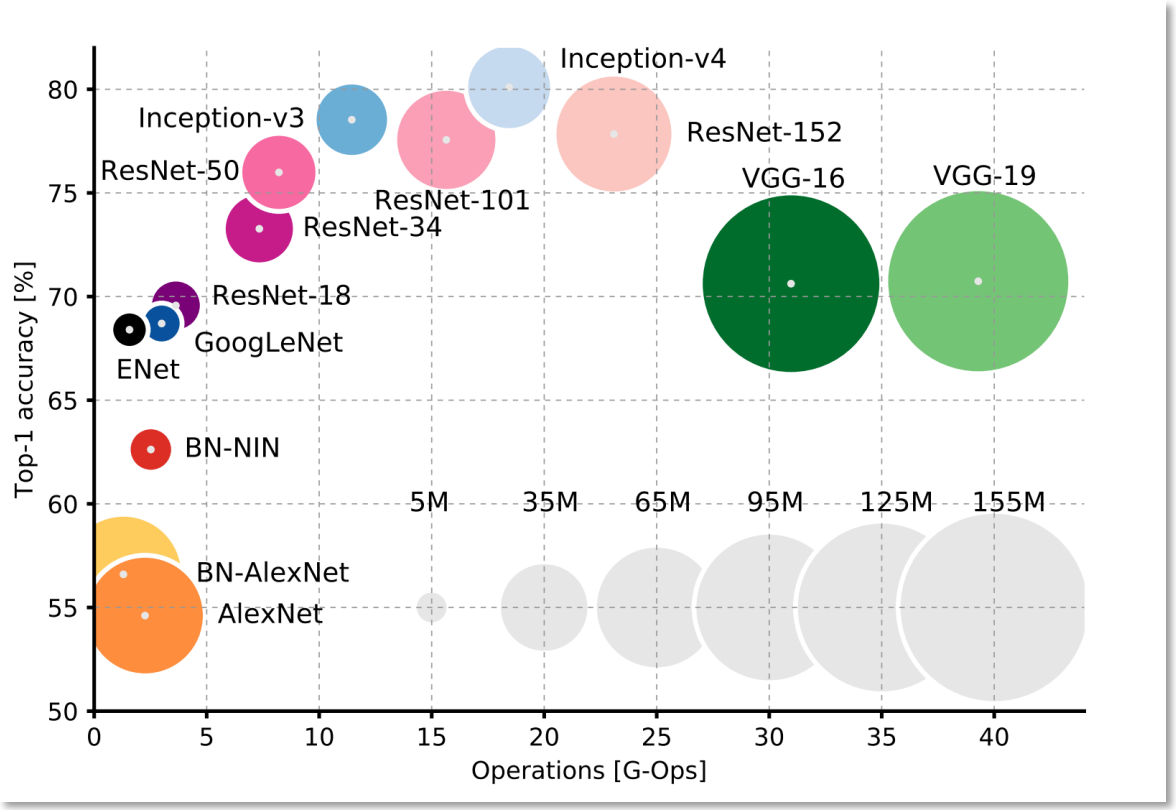
https://arxiv.org/abs/2101.00027

Common Crawl

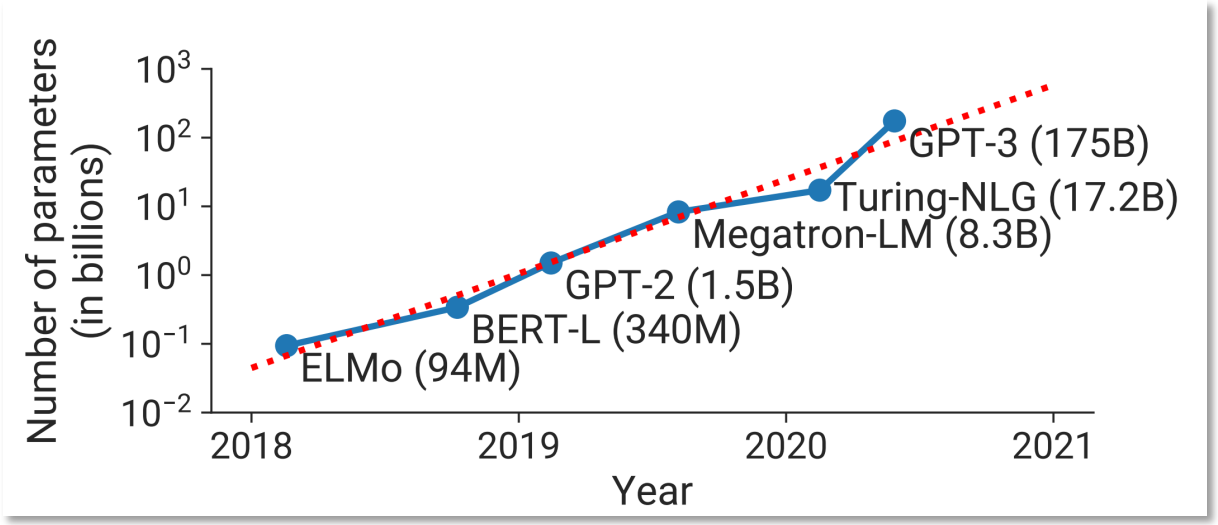
2.95 billion web pages
(351.844 TB)

https://commoncrawl.org/

Increasing Complexity of Deep Learning Models



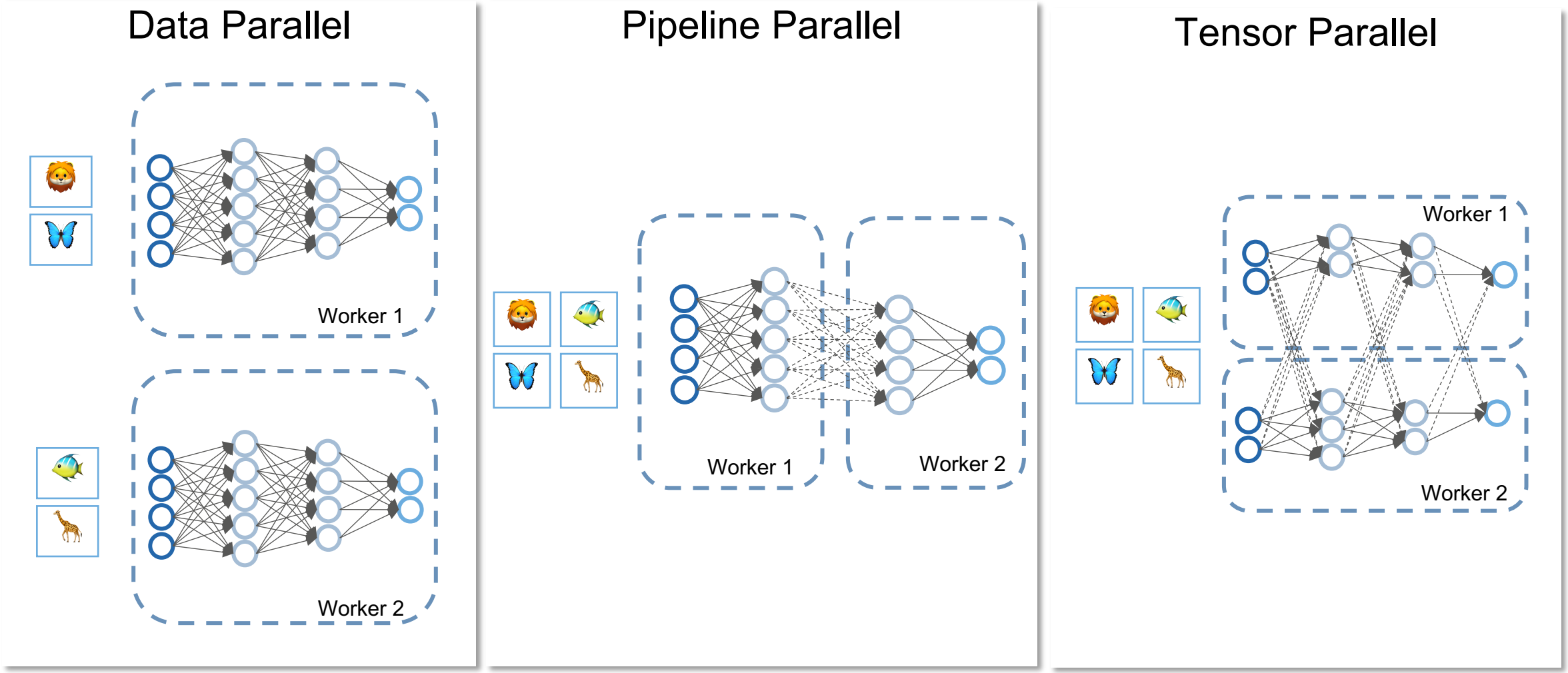
Source: Canziani et al., 2017, arXiv:1605.07678



Source: Narayanan et al., 2021, arXiv:2104.04473

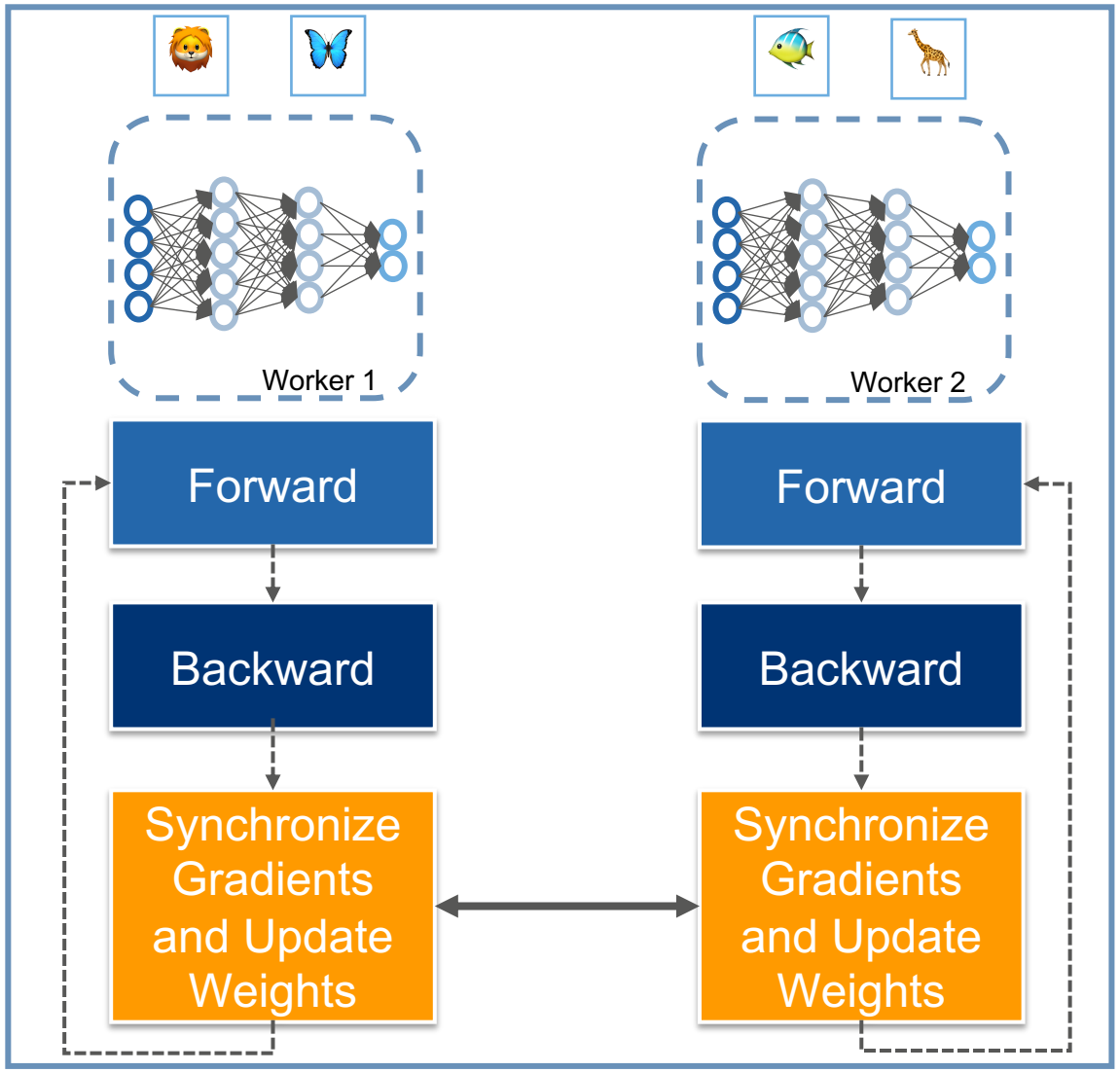
BigScience's BLOOM – 176B
Google's PaLM – 540B
GPT-4 – reportedly 1T

Overview of the Techniques for Distributed Training of NNs



Data Parallel Training of Neural Networks

- Every worker has a copy of the whole model and a subset of the data.
- Each worker generates its own weights, biases, gradients and optimizer states.
- At the end of every iteration, workers aggregate the gradients across all workers, average gradients, and then apply those averaged gradients (using e.g. *allreduce)*.
- **Collective (all-to-all)** communication of weight gradients.



Horovod

- Distributed open-source deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet.
- Runs across *multiple GPUs* and across *multiple machines/nodes*.
- *Fast* – scales up to hundreds of GPUs with upwards of 90% scaling efficiency.
- *Easy* – a few lines of codes.
- *Portable* – different frameworks.


```
#1 initialization
import horovod.torch as hvd
hvd.init()
```

```
#2 pin each GPU to a single process
if torch.cuda.is_available():
    torch.cuda.set_device(hvd.local_rank())
```

```
#3 distributed optimizer
opt = #chose your optimizer of preference
opt = hvd.DistributedOptimizer(opt)
```

```
#4 broadcast the initial variable states from rank 0 to all other processes
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)
```

```
#5 differentiate among different workers (e.g. check pointing on worker 0)
if hvd.rank() == 0:
    checkpoint.save(checkpoint_dir)
```

```
# run training with 2 GPUs on a single machine
$ horovodrun -np 2 python train.py
# run training with 2 GPUs on two machines (1 GPUs each)
$ horovodrun -np 2 -H hostname1:1,hostname2:1 python train.py
```

Sergeev and Del Balso, 2018., arXiv:1802.05799

https://horovod.readthedocs.io/en/stable/pytorch.html#

exercise2.py: Data Parallel Training of ResNet34 on 2 GPUs with Horovod

```
# Initialize Horovod and pin each GPU to a single process
hvd.init()
if torch.cuda.is_available():
    torch.cuda.set_device(hvd.local_rank())

# Partition datasets among workers using DistributedSampler
sampler = DistributedSampler(data, num_replicas=hvd.size(), rank=hvd.rank())
loader = torch.utils.data.DataLoader(data, batch_size, sampler)

# Use distributed optimizer and broadcast parameters
optimizer = hvd.DistributedOptimizer(optimizer, model.named_parameters())
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)

# Horovod: average metric values across workers.
def metric_average(val, name):
    tensor = val.clone().detach()
    avg_tensor = hvd.allreduce(tensor, name=name)
    return avg_tensor.item()
```

- Install in Pytorch container with: HOROVOD_GPU_OPERATIONS=NCCL pip install --no-cache-dir horovod

```
$ horovodrun -np 2 python exercise2.py
```

Output:

```
[1,1]<stdout>:Epoch [2/10], Step [24960/25000], Loss: 0.5788
[1,0]<stdout>:Epoch [2/10], Step [24960/25000], Loss: 0.4635
[1,1]<stdout>:Epoch [4/10], Step [24960/25000], Loss: 0.3478
[1,0]<stdout>:Epoch [4/10], Step [24960/25000], Loss: 0.2552
[1,1]<stdout>:Epoch [6/10], Step [24960/25000], Loss: 0.2159
[1,0]<stdout>:Epoch [6/10], Step [24960/25000], Loss: 0.1682
[1,1]<stdout>:Epoch [8/10], Step [24960/25000], Loss: 0.1184
[1,0]<stdout>:Epoch [8/10], Step [24960/25000], Loss: 0.0652
[1,0]<stdout>:Epoch [10/10], Step [24960/25000], Loss: 0.1116
[1,1]<stdout>:Epoch [10/10], Step [24960/25000], Loss: 0.0584

[1,0]<stdout>:Test set: Accuracy: 81.08%
[1,0]<stdout>:Elapsed time: 126.31 s ≈ 2 min
```

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 using 256 GPUs.

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 example(rank, world_size):
        # create default process group – nccl for building with cuda
        dist.init_process_group(backend="nccl", rank=rank,
                                                world_size=world_size)
        # create a model and wrap it with DDP
        model = nn.Linear(10, 10).to(rank)
        ddp_model = DDP(model, device_ids=[rank])
        # define loss function and optimizer
        loss_fn = nn.MSELoss()
        optimizer = torch.optim.SGD(ddp_model.parameters(), lr=0.01)
        # forward pass
        outputs = ddp_model(torch.randn(20, 10).to(rank))
        labels = torch.randn(20, 10).to(rank)
        # backward pass and update parameters
        loss_fn(outputs, labels).backward()
        optimizer.step()
        # cleanup
        dist.destroy_process_group()

if __name__=="__main__":
        # environment variables for using torch.distributed
        os.environ["MASTER_ADDR"] = "localhost"
        os.environ["MASTER_PORT"] = "29500"
        world_size = 2   # world_size = gpus * nodes
        mp.spawn(example, args=(world_size,), nprocs=world_size)
```

exercise3.py: Data Parallel Training of VGG19 on 2 GPUs with Pytorch DDP

```
# environment variables for using torch.distributed
world_size = gpus * nodes
os.environ['MASTER_ADDR'] = 'localhost'
os.environ['MASTER_PORT'] = '8888'
mp.spawn(train_and_test, gpus, args=(gpus, nodes, nr, world_size))

def train_and_test(gpu, gpus, nodes, nr, world_size):
    rank = nr * gpus + gpu  # nr is ranking within the nodes
    dist.init_process_group(backend='nccl', init_method='env://', world_size, rank)
    # Set the seed to initiate the model with the same weights on each worker
    torch.manual_seed(0)
    # Set the model and loss to GPU
    model.cuda()
    criterion = nn.CrossEntropyLoss().cuda()
    # Wrap the model
    model = nn.parallel.DistributedDataParallel(model, device_ids=[gpu])
    # Partition datasets among workers using DistributedSampler
    sampler = DistributedSampler(data, num_replicas=world_size, rank=rank)
    loader = torch.utils.data.DataLoader(data, batch_size, sampler)
```

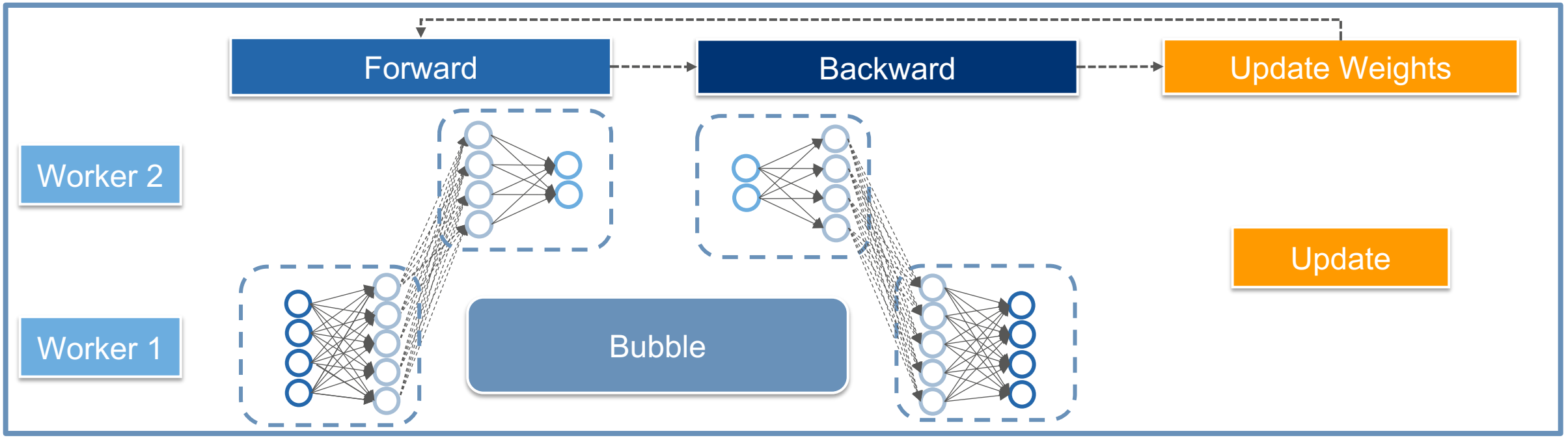
```
$ python exercise3.py
```

Output:

```
Epoch [2/10], Step [24960/25000], Loss: 1.5240
Epoch [2/10], Step [24960/25000], Loss: 1.4402
Epoch [4/10], Step [24960/25000], Loss: 0.7013
Epoch [4/10], Step [24960/25000], Loss: 0.7268
Epoch [6/10], Step [24960/25000], Loss: 0.5393
Epoch [6/10], Step [24960/25000], Loss: 0.4175
Epoch [8/10], Step [24960/25000], Loss: 0.3391
Epoch [8/10], Step [24960/25000], Loss: 0.4450
Epoch [10/10], Step [24960/25000], Loss: 0.3889
Epoch [10/10], Step [24960/25000], Loss: 0.3106

Test set: Accuracy: 81.76%
Elapsed time:  234.65 s  ≈ 4 min
```

Pipeline Parallel Training of Neural Networks

- Each worker has one or more layers, and the corresponding parameters of the neural network. There is a single copy of weight parameters shared between the workers.
- Goal: Make feasible models, that are larger than the memory capacity of a single GPU.
- **Point-to-point** communication of intermediate activations and gradients.



Pipeline Parallel Training of Neural Networks - Different Schemes

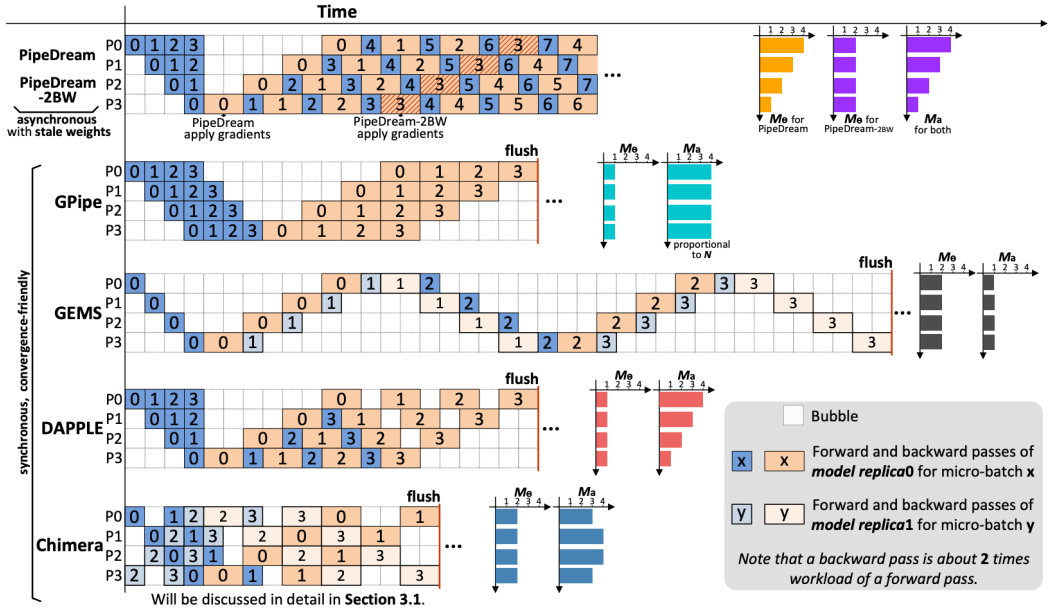


Figure 2: Pipeline parallelism schemes, with four pipeline stages (D=4) and four micro-batches (N=4) within a training iteration, except that PipeDream updates the model after each backward pass on a micro-batch.

Table 1: The list of symbols frequently used in the paper.

D	The number of pipeline stages (*depth*)
W	The number of replicated pipelines (*width*) for data parallelism [1]
P	The number of workers ($= W * D$)
B	Micro-batch size
N	The number of micro-batches executed by each worker within a training iteration
$\hat{B}$	Mini-batch size ($= B * N * W$)
M_θ	Memory consumption for the weights of one stage
M_a	Memory consumption for the activations of one stage

[1] This paper considers the cases where all pipeline stages have balanced workload, and therefore are equally replicated to combine with data parallelism.

- Goal: Minimizing bubbles in the pipeline.

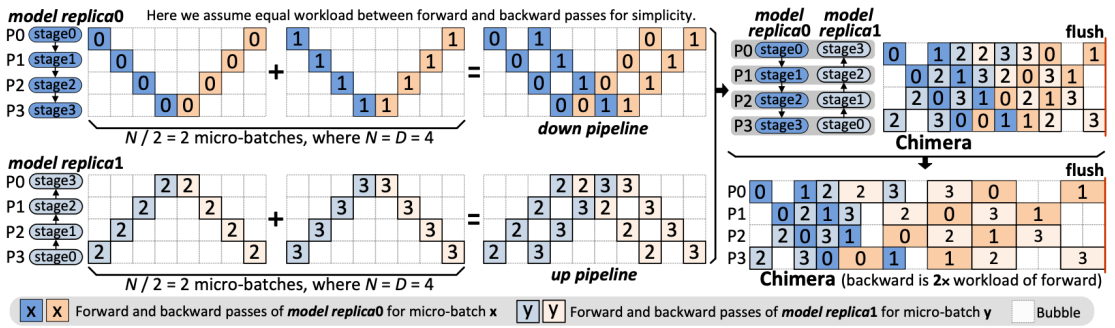


Figure 3: Model replicas and bidirectional pipelines scheduling of Chimera.

Li and Hoefler, 2021,
http://htor.inf.ethz.ch/publications/img/shigang-chimera.pdf

Pipeline Parallelism with Pytorch Pipe

- Micro-batch pipeline parallelism based on Gpipe scheme.
- Depends on the Remote Procedure Call (RPC) framework from *torch.distributed* for communication.
- Currently runs only on a *single machine/node.*
- *Experimental and subject to change! Not yet fully compatible with CUDA!*

```
from torch.distributed import rpc
from torch.distributed.pipeline.sync import Pipe

# initialize Remote Procedure Call (RPC) framework
os.environ['MASTER_ADDR'] = “localhost”
os.environ['MASTER_PORT'] = “29500”
rpc.init_rpc(name='worker', rank=0, world_size=1)

# build the pipe with each layer on different GPU
fc1 = nn.Linear(16, 8).cuda(0)
fc2 = nn.Linear(8, 4).cuda(1)
model = nn.Sequential(fc1, fc2)
model = Pipe(model, chunks=8)
input = torch.rand(16, 16).cuda(0)

# rref is a reference to a value of a tensor on a remote worker
output_rref = model(input)
outputs = output_rref.local_value()
loss = criterion(outputs, targets.cuda(1))
```

https://pytorch.org/docs/stable/pipeline.html

Kim et al., 2020, https://arxiv.org/abs/2004.09910

Exercise: Pipeline Parallel Training of a Transformer on 2 GPUs

```
nlayers = 12
num_gpus = 2
partition_len = ((nlayers - 1) // num_gpus) + 1
# Add encoder in the beginning.
tmp_list = [Encoder(ntokens, emsize, dropout).cuda(0)]
module_list = []
# Add all the necessary transformer blocks.
for i in range(nlayers):
    transformer_block = TransformerEncoderLayer(emsize, nhead, nhid, dropout)
    if i != 0 and i % (partition_len) == 0:
        module_list.append(nn.Sequential(*tmp_list))
        tmp_list = []
    device = i // (partition_len)
    tmp_list.append(transformer_block.to(device))
# Add decoder in the end.
tmp_list.append(Decoder(ntokens, emsize).cuda(num_gpus - 1))
module_list.append(nn.Sequential(*tmp_list))

from torch.distributed.pipeline.sync import Pipe
# Build the pipeline.
chunks = 8
model = Pipe(torch.nn.Sequential(*module_list), chunks = chunks)
```

- Using an embedding dimension of 4096, hidden size of 4096, 16 attention heads and 12 transformer layers.
- Model with ~1.4 billion parameters.
- Split the model such that half of the nn.TransformerEncoderLayer is on 1st GPU and the other half is on the 2nd.

```
$ python exercise3.py
```

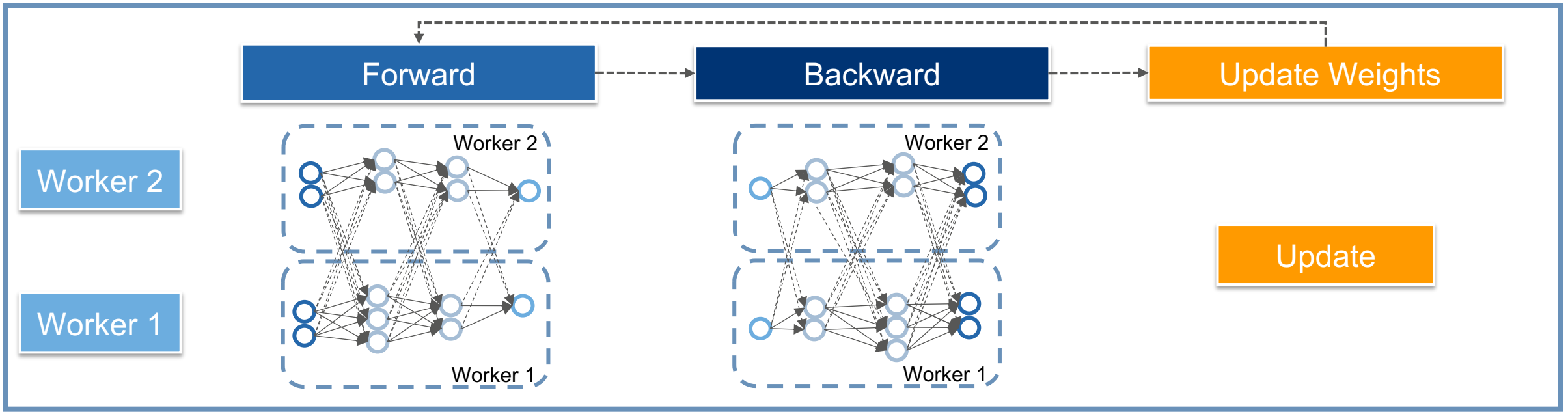
Output:

```
Total parameters in model: 1,444,261,998
end of epoch 1 | time: 28.42s | valid loss 1.22
end of epoch 2 | time: 27.36s | valid loss 0.25
end of epoch 3 | time: 27.49s | valid loss 0.23
End of training | test loss 0.20
```

https://pytorch.org/tutorials/intermediate/pipeline_tutorial.html

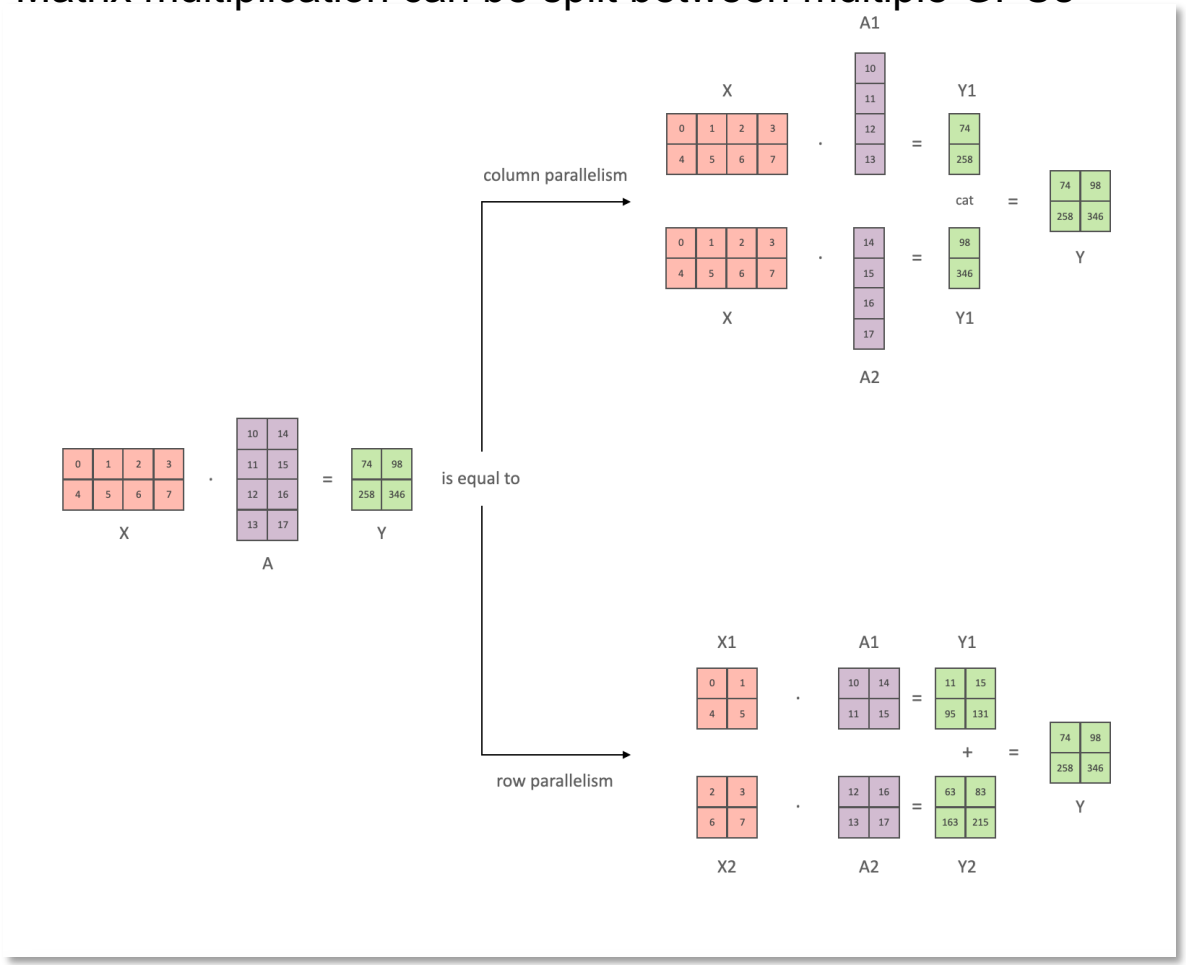
Tensor Parallel Training of Neural Networks

- Each operator or layer of the neural network split between the workers.
- There is a single copy of weight parameters shared between the workers.
- Goal: Make feasible models that are larger than the memory capacity of a single GPU.
- **Collective (all-to-all)** communication of <u>partial</u> activations and gradients of each worker.

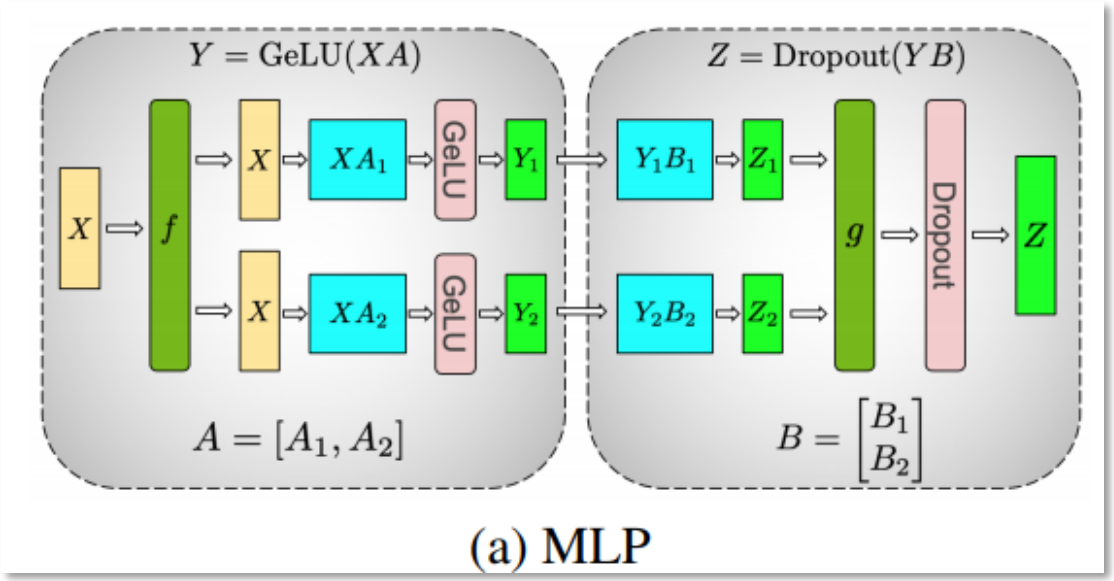


Tensor Parallel Training of Neural Networks

Matrix multiplication can be split between multiple GPUs



Source: https://huggingface.co/docs/transformers/parallelism#naive-model-parallelism-vertical-and-pipeline-parallelism



Source: Narayanan et al., 2021, arXiv:2104.04473

Combination of Data, Pipeline and Tensor Parallelism – Megatron-LM

How do we navigate this configuration space?

- Degree of pipeline, tensor, and data parallelism
- Pipelining schedule
- Global batch size
- Microbatch size

→ Each of these choices influence amount of communication, size of pipeline bubble, memory footprint

Tensor, pipeline, and data parallelism can be composed to scale with a trillion parameters to thousands of GPUs.

Source: Narayanan et al., 2021, arXiv:2104.04473
https://github.com/NVIDIA/Megatron-LM
https://www.youtube.com/watch?v=MO_CESBOXgo

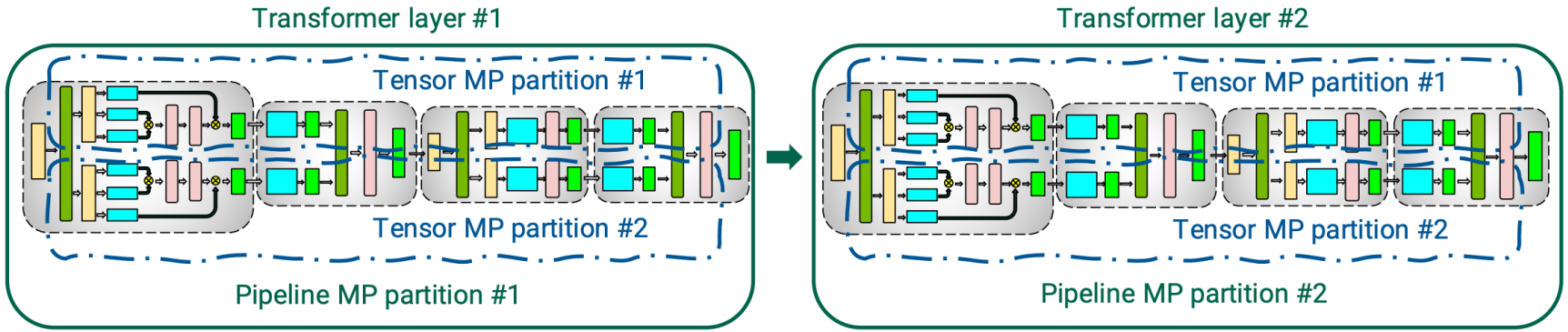


Figure 2: Combination of tensor and pipeline model parallelism (MP) used in this work for transformer-based models.

Further Topics

- Pytorch tutorials on distributed training of transformer models:
  - Data and pipeline parallelism: https://pytorch.org/tutorials/advanced/ddp_pipeline.html
- Mesh TensorFlow data and model parallelism:
  - GitHub: https://github.com/tensorflow/mesh
  - Paper: Shazeer et al., 2018, https://arxiv.org/pdf/1811.02084.pdf
- DeepSpeed (PyTorch deep learning optimization library by Miscrosoft):
  - Pipeline parallelism: https://www.deepspeed.ai/tutorials/pipeline/
  - ZeRO data parallelism: https://deepspeed.readthedocs.io/en/latest/zero3.html
- FairScale (PyTorch library for high performance large scale training by Facebook Research):
  - Pipeline parallelism: https://fairscale.readthedocs.io/en/latest/tutorials/pipe.html
  - ZeRO data parallelism: https://github.com/facebookresearch/fairscale/#optimizer-state-sharding-zero

Agenda

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 with an Enroot container

2. Fundamentals of Deep Learning

- ✓ Introduction to Neural Networks
- ✓ Introduction to Convolutional Neural Networks
- ✓ Exercises: Train CNNs on a GPU
- ✓ Introduction to Transformers
- ✓ Exercise: Train a Transformer on a GPU

3. Distributed Training of Neural Networks

- ✓ Data Parallel Training
- ✓ Exercise: DP Training of CNN on 2 GPUs
- ✓ Model Parallel Training: Pipeline Parallel and Tensor Parallel
- ✓ Exercise: PP Training of Transformer on 2 GPUs

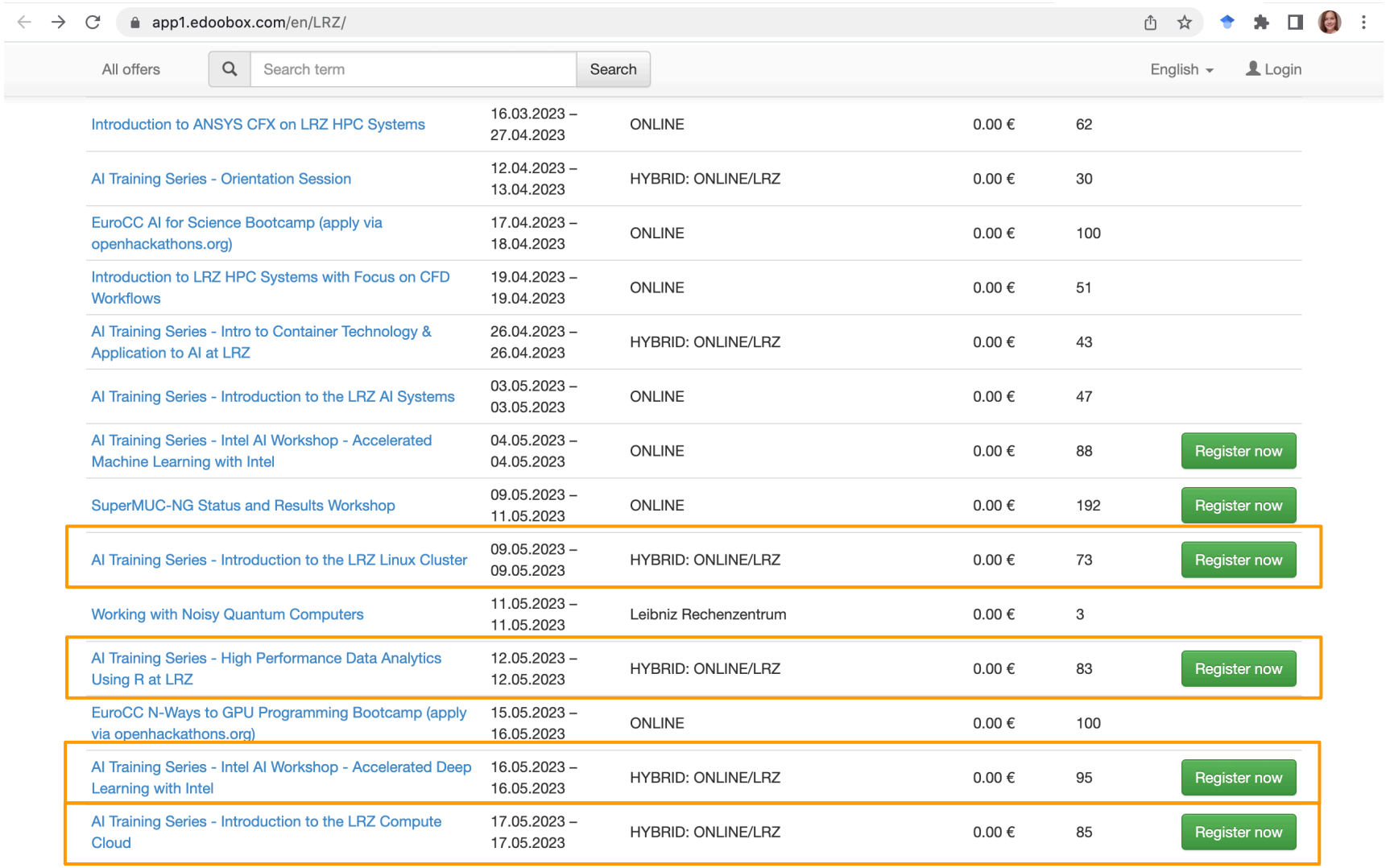
Course Evaluation – Module 5: Introduction to the LRZ AI Systems

Please visit
https://survey.lrz.de/index.php/538854?lang=en
and rate this course.

Your feedback is highly appreciated!
Thank you!

ANNOUNCEMENT

https://app1.edoobox.com/en/LRZ/



Introduction to ANSYS CFX on LRZ HPC Systems	16.03.2023 – 27.04.2023	ONLINE	0.00 €	62	
AI Training Series - Orientation Session	12.04.2023 – 13.04.2023	HYBRID: ONLINE/LRZ	0.00 €	30	
EuroCC AI for Science Bootcamp (apply via openhackathons.org)	17.04.2023 – 18.04.2023	ONLINE	0.00 €	100	
Introduction to LRZ HPC Systems with Focus on CFD Workflows	19.04.2023 – 19.04.2023	ONLINE	0.00 €	51	
AI Training Series - Intro to Container Technology & Application to AI at LRZ	26.04.2023 – 26.04.2023	HYBRID: ONLINE/LRZ	0.00 €	43	
AI Training Series - Introduction to the LRZ AI Systems	03.05.2023 – 03.05.2023	ONLINE	0.00 €	47	
AI Training Series - Intel AI Workshop - Accelerated Machine Learning with Intel	04.05.2023 – 04.05.2023	ONLINE	0.00 €	88	Register now
SuperMUC-NG Status and Results Workshop	09.05.2023 – 11.05.2023	ONLINE	0.00 €	192	Register now
AI Training Series - Introduction to the LRZ Linux Cluster	09.05.2023 – 09.05.2023	HYBRID: ONLINE/LRZ	0.00 €	73	Register now
Working with Noisy Quantum Computers	11.05.2023 – 11.05.2023	Leibniz Rechenzentrum	0.00 €	3	
AI Training Series - High Performance Data Analytics Using R at LRZ	12.05.2023 – 12.05.2023	HYBRID: ONLINE/LRZ	0.00 €	83	Register now
EuroCC N-Ways to GPU Programming Bootcamp (apply via openhackathons.org)	15.05.2023 – 16.05.2023	ONLINE	0.00 €	100	
AI Training Series - Intel AI Workshop - Accelerated Deep Learning with Intel	16.05.2023 – 16.05.2023	HYBRID: ONLINE/LRZ	0.00 €	95	Register now
AI Training Series - Introduction to the LRZ Compute Cloud	17.05.2023 – 17.05.2023	HYBRID: ONLINE/LRZ	0.00 €	85	Register now

Thank you for your attention! ☺

Maja Piskač
Leibniz Supercomputing Centre
LRZ AI & Big Data Team
maja.piskac@lrz.de