USC University of Southern California

Running Deep Learning Applications on HPC System

Hao Ji, Data Scientist

USC Center for Advanced Research Computing

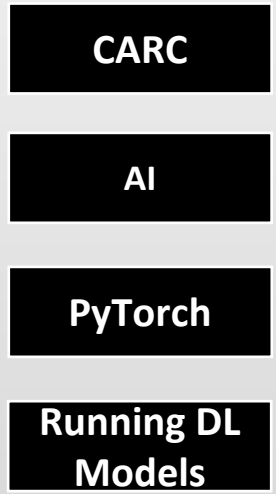
Content

Section 1: Intro to CARC

Section 2: History of AI

Section 3: Install PyTorch on Cluster

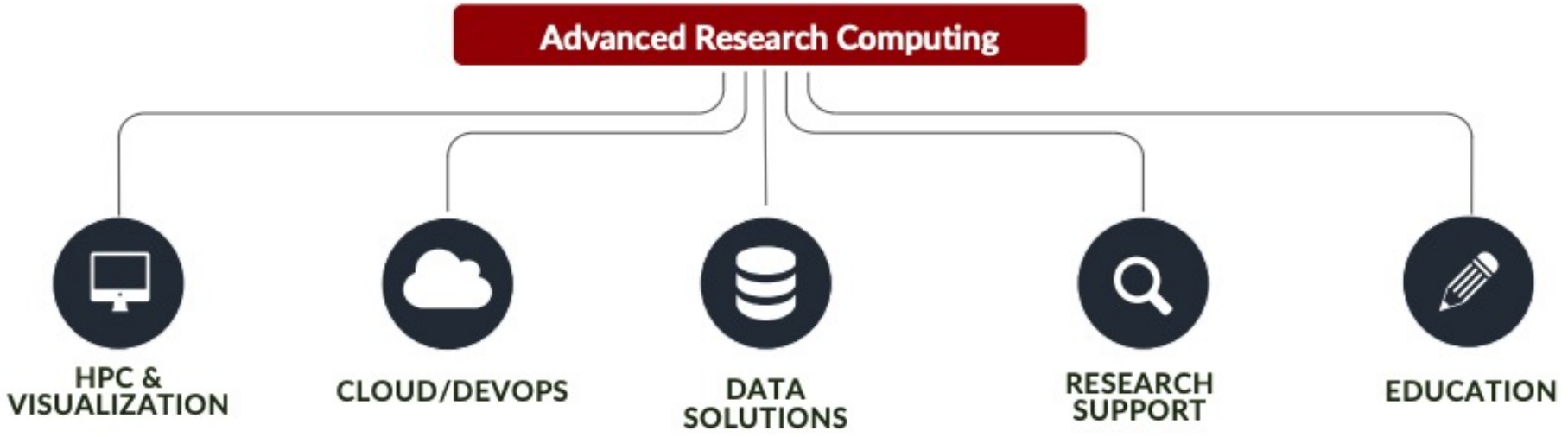
Section 4: Running Deep Learning Models



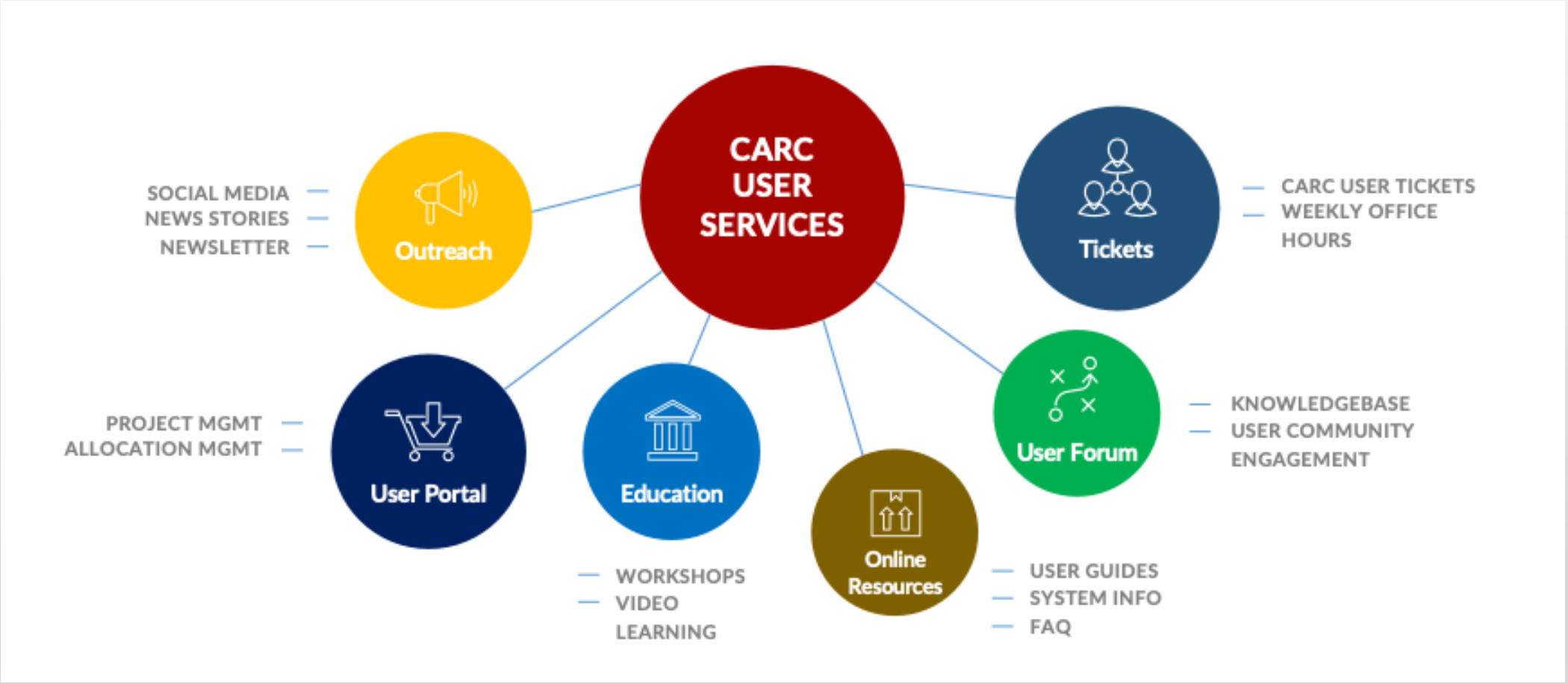
Intro to CARC

CENTER FOR ADVANCED RESEARCH COMPUTING: MISSION & VISION

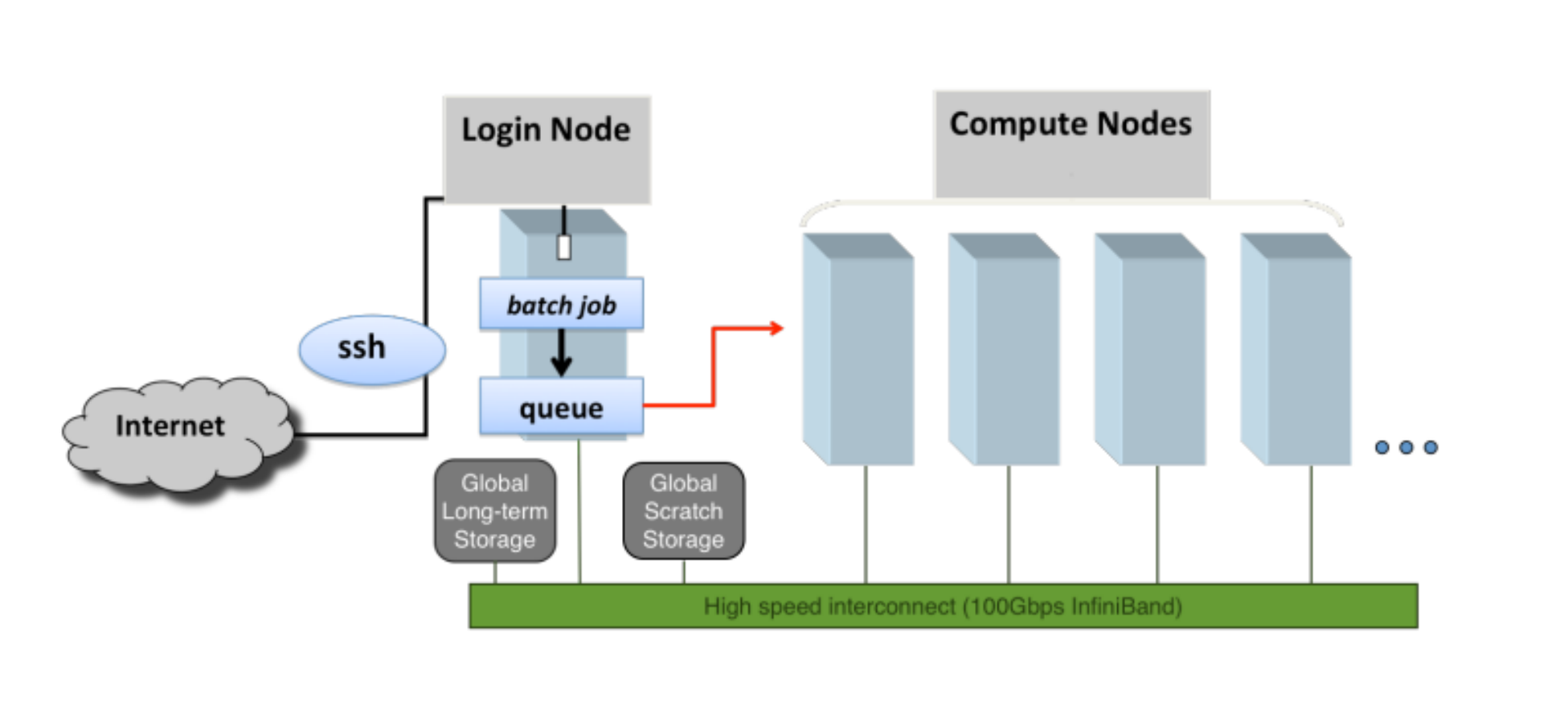
- USC Center for Advanced Research Computing (CARC) aims to provide advanced computational research support to USC faculty and students.
- CARC aims to be an institutional resource and a long-term research partner of USC research community



Intro to CARC



CARC Cluster

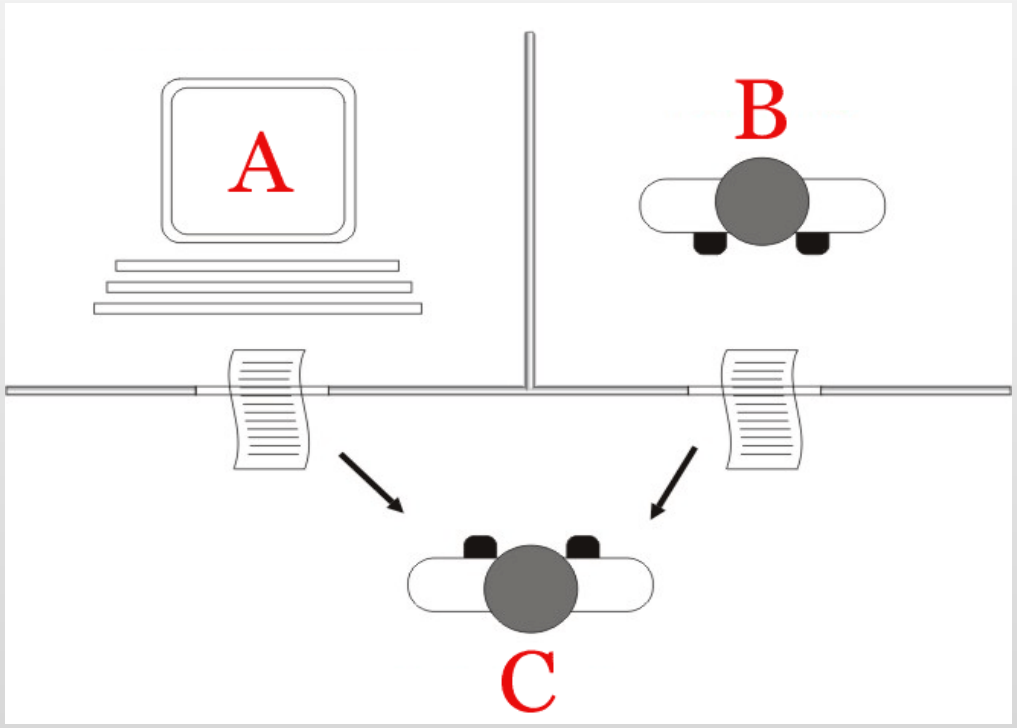


History of AI

Pre-1950s: Foundations of AI

The formal study of AI began with mathematicians and philosophers like Alan Turing.

In 1950, Alan Turing proposed the famous "Turing Test," a measure of a machine's ability to exhibit human-like intelligence. This concept laid the groundwork for future AI research.



The "standard interpretation" of the Turing test, in which player C, the interrogator, is given the task of trying to determine which player – A or B – is a computer and which is a human. The interrogator is limited to using the responses to written questions to make the determination.

History of AI

1950s and 1960s: Early Research and Symbolic AI

The term "artificial intelligence" was coined in 1956 at the Dartmouth Conference, where the field of AI research was officially established.

During this period, researchers focused on symbolic AI, using rules and logic to mimic human problem-solving processes.

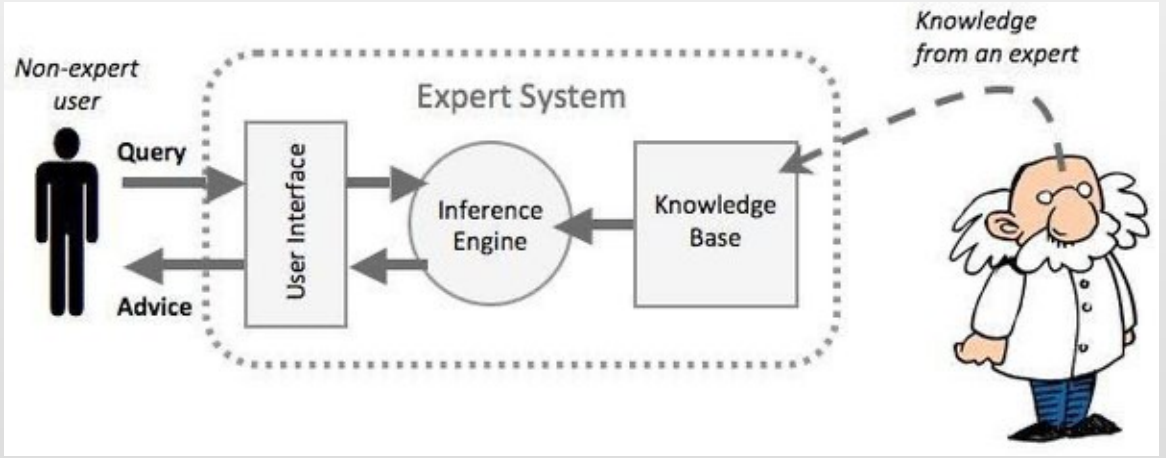
History of AI

1970s and 1980s: Knowledge-Based Systems AI

Research saw a shift towards knowledge-based systems and expert systems.

These systems utilized large knowledge bases to reason and draw conclusions about specific domains.

Though promising, early expert systems faced limitations due to the complexity of representing all human knowledge in a formalized manner.

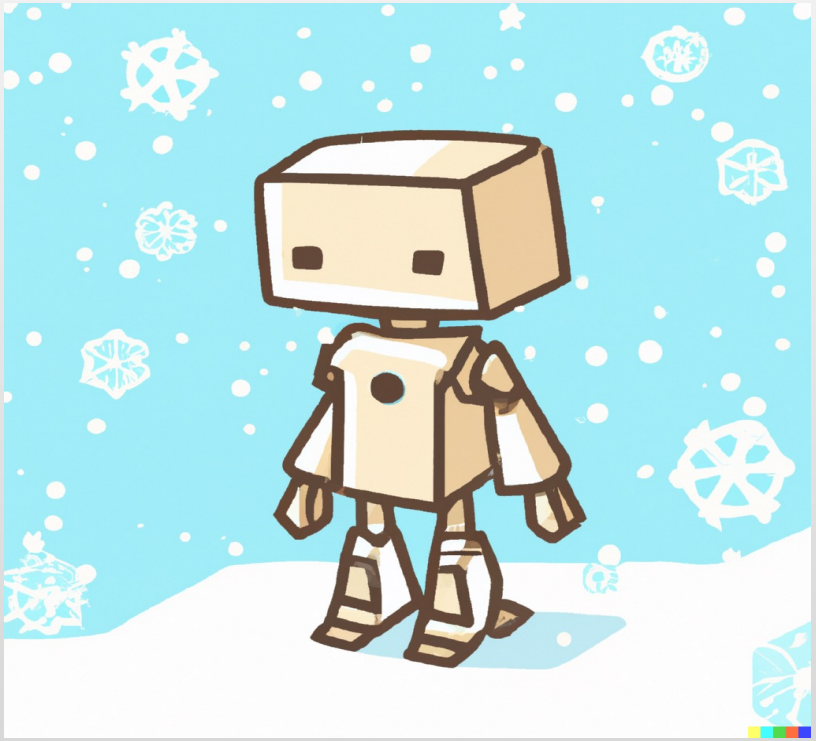


History of AI

1980s and 1990s: AI Winter

During this period, AI faced a decline in interest and funding due to unmet expectations and the inability to deliver on grand promises.

Progress in AI did not match initial optimism, leading to a phase known as the "AI Winter."



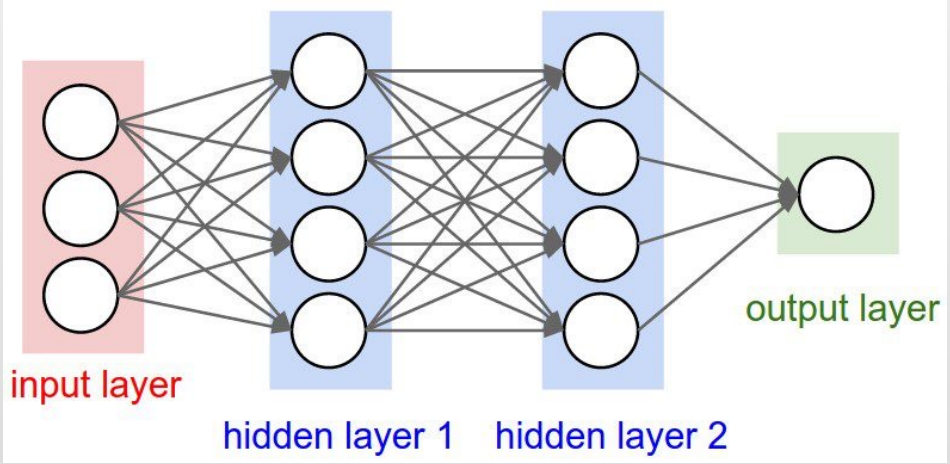
History of AI

Late 1990s and 2000s: Machine Learning and Neural Networks

The resurgence of AI came with the advent of machine learning techniques, particularly neural networks.

Improved computational power and access to vast amounts of data allowed neural networks to excel in tasks like image recognition and language processing.

Support vector machines and other machine learning algorithms also gained popularity.

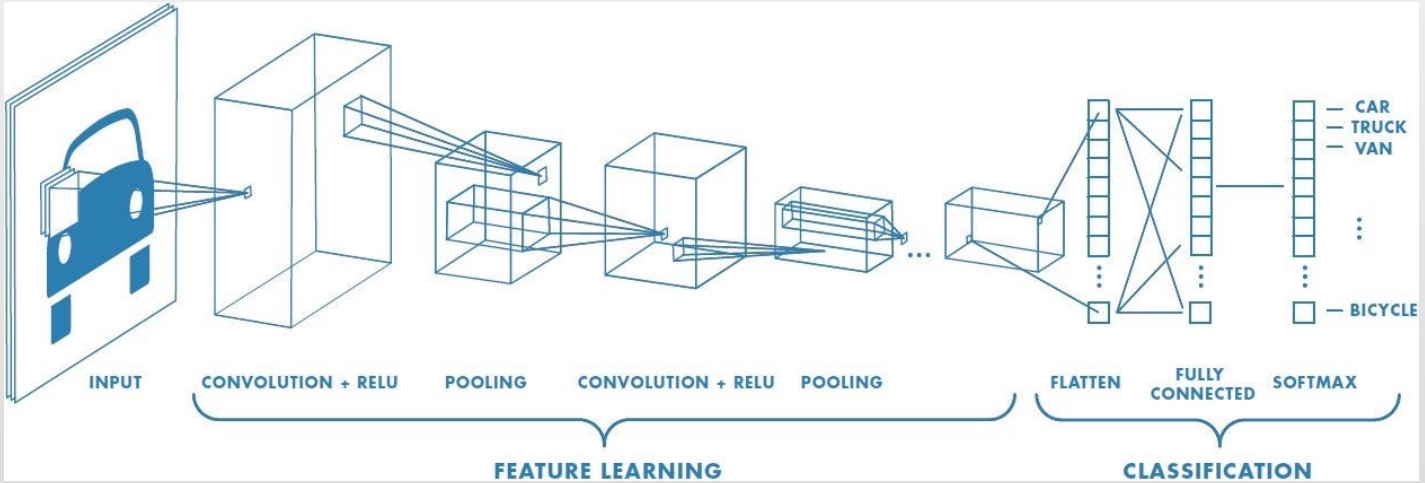


History of AI

2010s: Deep Learning and AI Breakthroughs

Deep learning, a subset of machine learning using artificial neural networks with multiple layers, became the driving force behind many AI breakthroughs.

Applications of AI proliferated in various fields, including natural language processing, computer vision, robotics, and autonomous vehicles.



History of AI

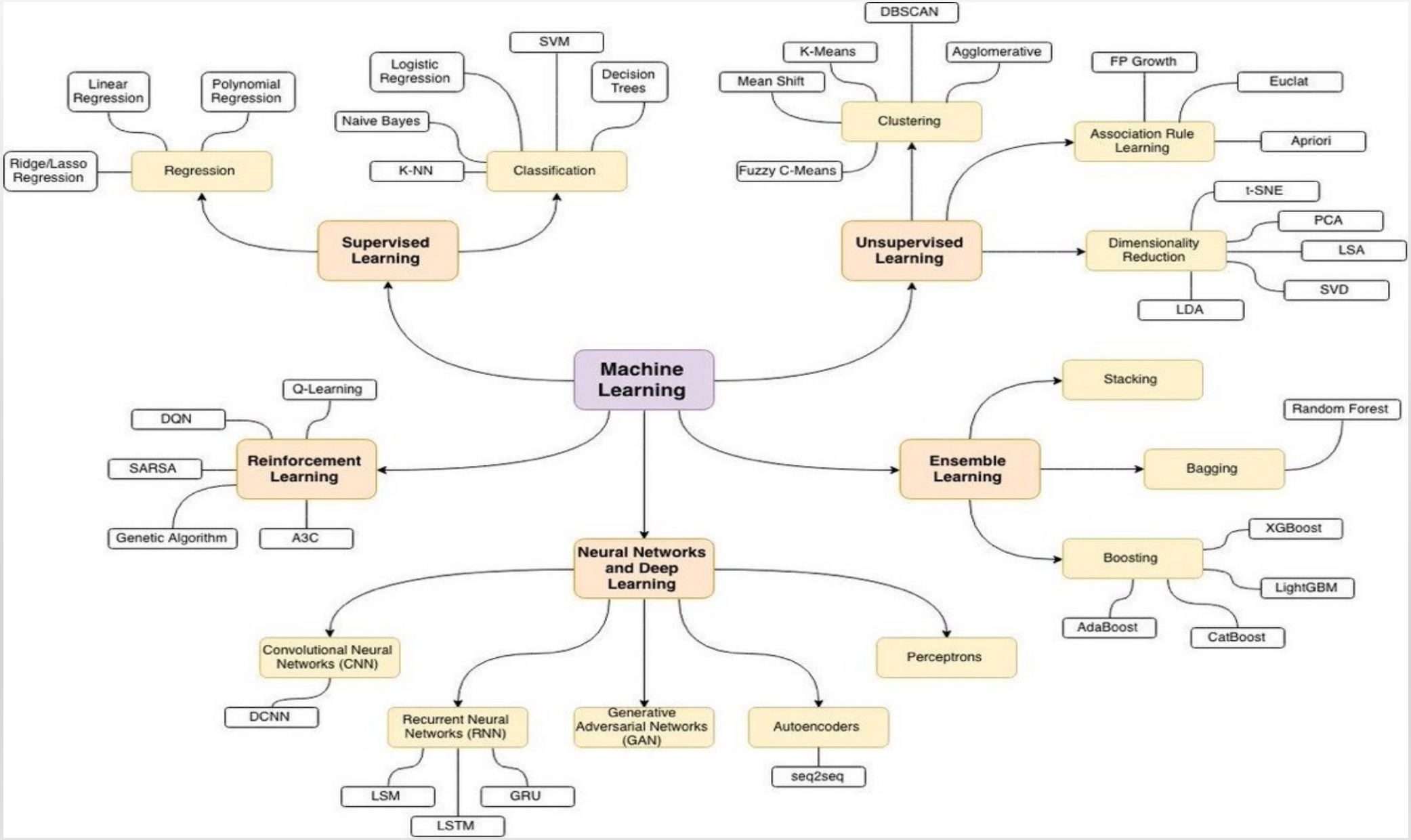
Present and Beyond

AI continues to evolve rapidly, with ongoing research in areas like reinforcement learning, explainable AI, Large Language Model and AI ethics.

Integration of AI in everyday life through virtual assistants, smart devices, and recommendation systems is becoming increasingly common.

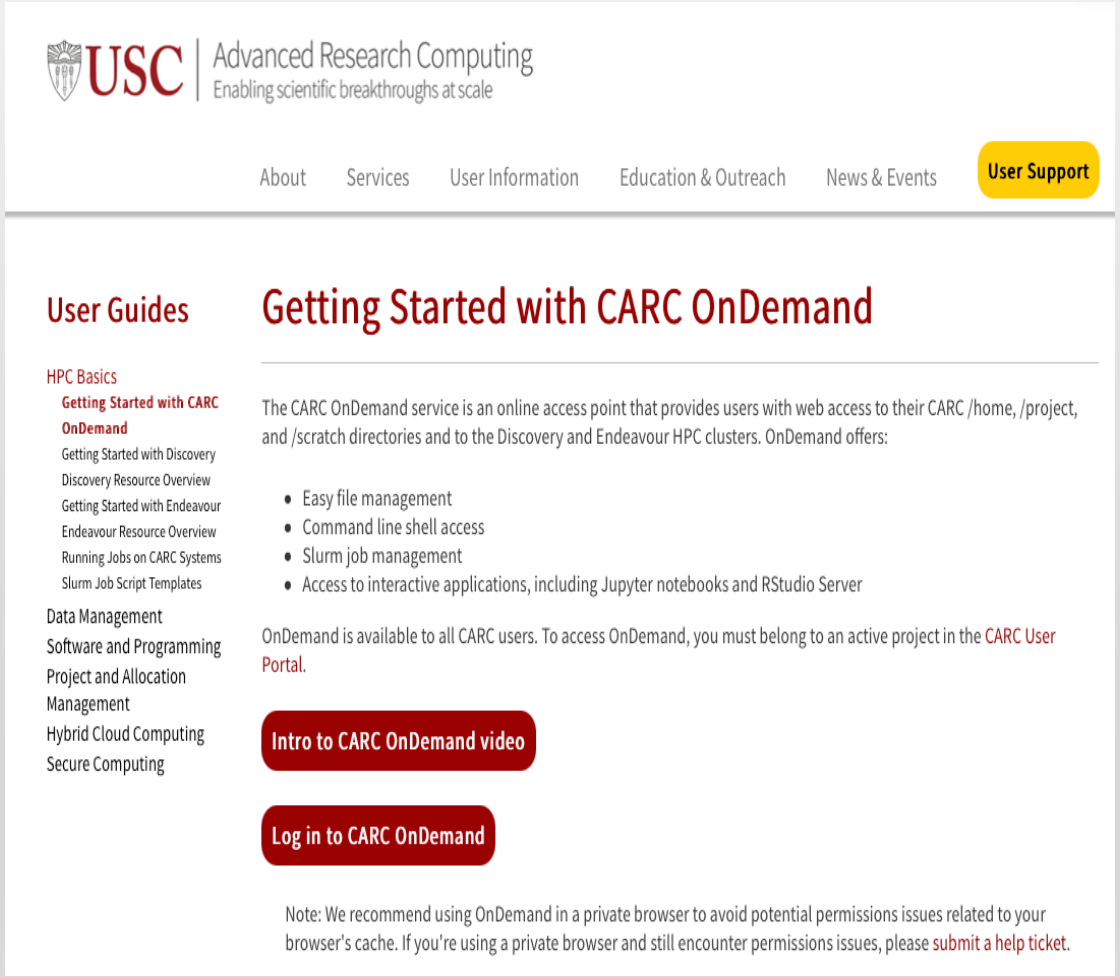
However, concerns about AI's societal impact, privacy, and ethical implications persist.

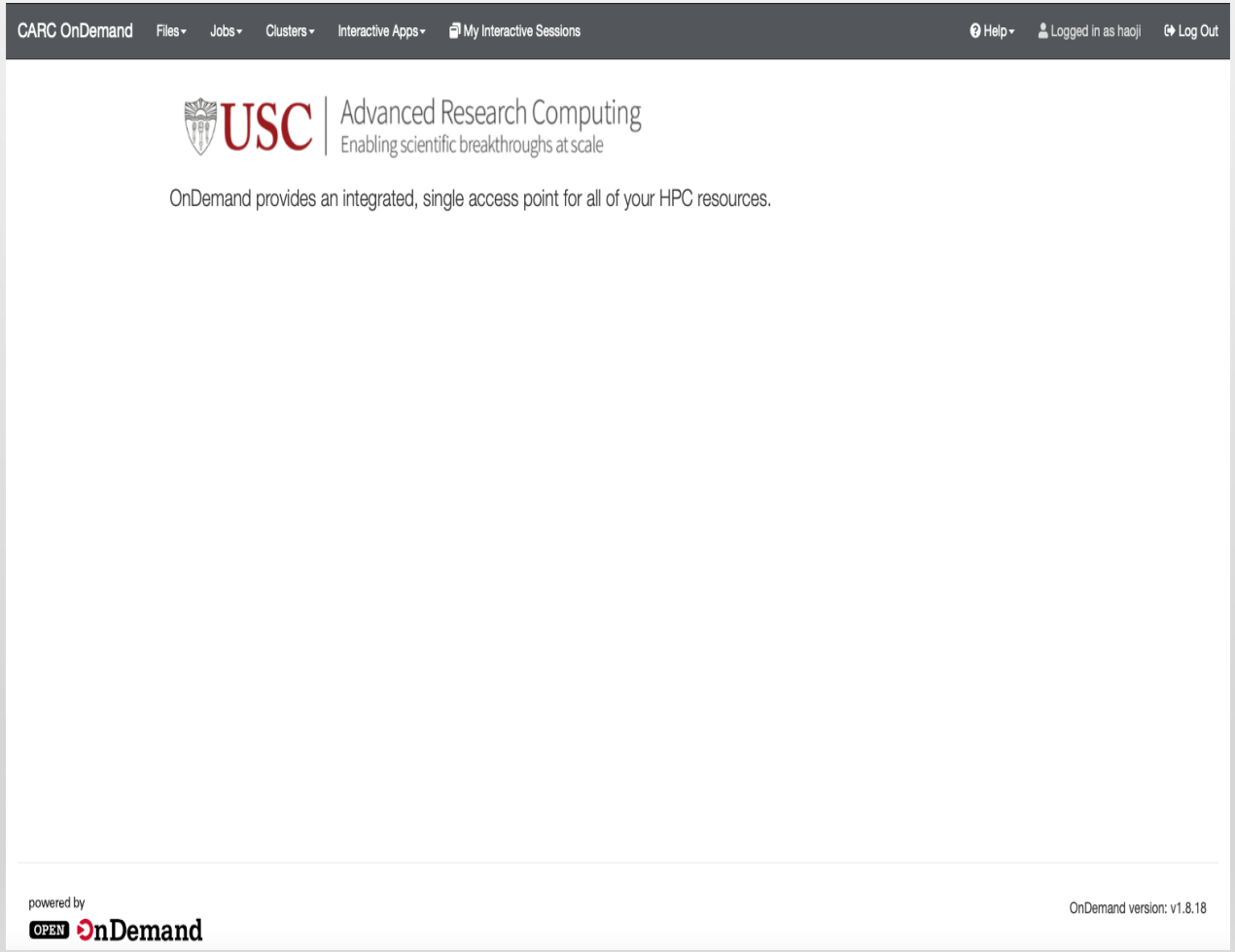
Introduction of Machine Learning



CARC OnDemand

Web Address: https://carc-ondemand.usc.edu

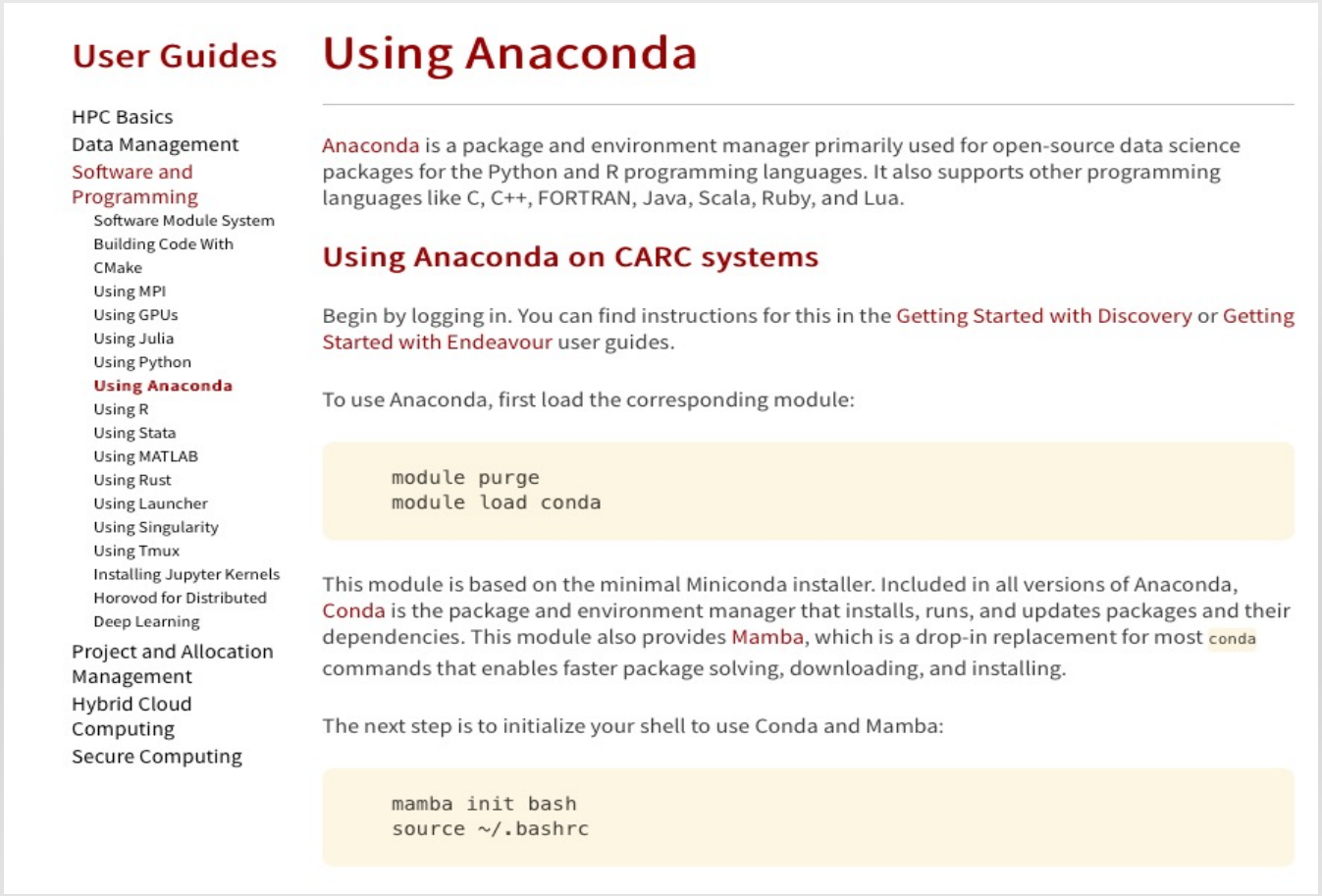




Using Anaconda on CARC

Anaconda: package and environment manager primarily used for open-source data science packages for the Python and R programming languages.

```
salloc --account=hpcsuppt_613 --partition=gpu --nodes=1 --ntasks=1 --cpus-per-task=8 --time=1:00:00 --mem=32GB --gres=gpu:p100:1 --reservation=hacksc
```



User Guides

- HPC Basics
- Data Management
- Software and Programming
  - Software Module System
  - Building Code With CMake
  - Using MPI
  - Using GPUs
  - Using Julia
  - Using Python
  - **Using Anaconda**
  - Using R
  - Using Stata
  - Using MATLAB
  - Using Rust
  - Using Launcher
  - Using Singularity
  - Using Tmux
  - Installing Jupyter Kernels
  - Horovod for Distributed Deep Learning
- Project and Allocation Management
- Hybrid Cloud Computing
- Secure Computing

Using Anaconda

Anaconda is a package and environment manager primarily used for open-source data science packages for the Python and R programming languages. It also supports other programming languages like C, C++, FORTRAN, Java, Scala, Ruby, and Lua.

Using Anaconda on CARC systems

Begin by logging in. You can find instructions for this in the Getting Started with Discovery or Getting Started with Endeavour user guides.

To use Anaconda, first load the corresponding module:

```
module purge
module load conda
```

This module is based on the minimal Miniconda installer. Included in all versions of Anaconda, Conda is the package and environment manager that installs, runs, and updates packages and their dependencies. This module also provides Mamba, which is a drop-in replacement for most `conda` commands that enables faster package solving, downloading, and installing.

The next step is to initialize your shell to use Conda and Mamba:

```
mamba init bash
source ~/.bashrc
```

https://www.carc.usc.edu/user-information/user-guides/software-and-programming/anaconda

Using Anaconda on CARC

https://pytorch.org

START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

PyTorch Build	Stable (2.1.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	ROCm 5.6	CPU
Run this Command:	conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia			

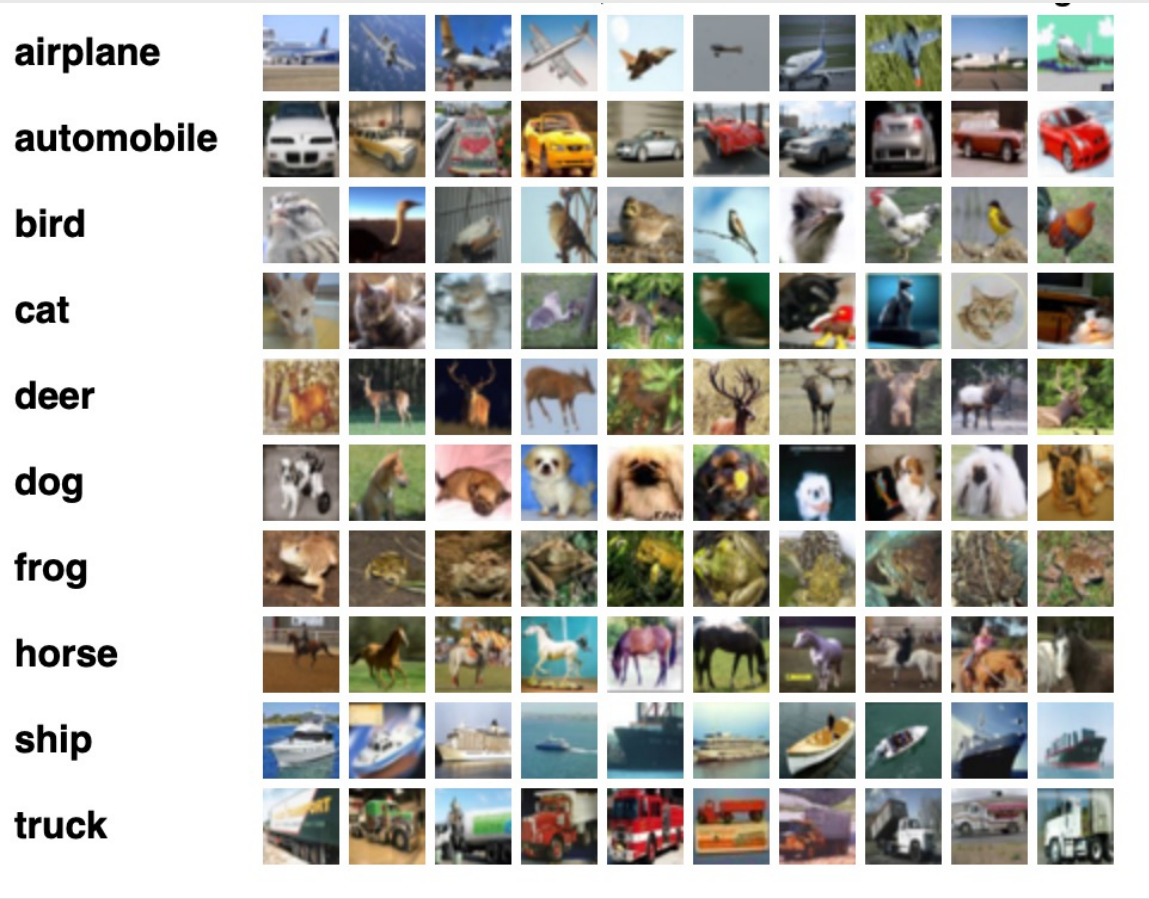
Using Anaconda on CARC

Install PyTorch:
1 mamba create –n torch_benchmark python=3.11
2 mamba activate torch_benchmark
3 mamba install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia

Test installation: type python and import torch

An example of Training

The problem we're going to solve today is to train a model to classify CIFAR10 datasets.
CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class.
There are 50000 training images and 10000 test images.
We will use Resnet50 model as a basis for training



Slurm script for job submission

Sample Job Script for running training jobs on GPU partition

git clone https://github.com/jihao2021/running-DL-applications-in-HPC.git

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem=32g
#SBATCH --time=02:00:00
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --account=hpcsuppt_613
#SBATCH --reservation=hacksc

module purge
eval "$(conda shell.bash hook)"
conda activate torch_benchmark
python CIFAR10_ResNet50.py
```

Common types of GPU

https://www.carc.usc.edu/user-information/user-guides/hpc-basics/discovery-resources

Partition	CPU model	CPU frequency	CPUs/node	GPU model	GPUs/node	Memory/node	Nodes
gpu	xeon-6130	2.10 GHz	32	V100	2	184 GB	29
gpu	xeon-2640v4	2.40 GHz	20	P100	2	123 GB	38
gpu	epyc-7282	2.80 GHz	32	A40	2	248 GB	12
gpu	epyc-7513	2.60 GHz	64	A100	2	248 GB	12

Common types of GPU

https://www.carc.usc.edu/user-information/user-guides/hpc-basics/discovery-resources

GPU specifications in the GPU partition

There are four kinds of GPUs in the GPU partition: A100, A40, V100, P100. The following is a summary table for the GPU specifications:

GPU model	Architecture	Memory	Memory Bandwidth	Base Clock Speed	Cuda Cores	Tensor Cores	Single Precision Performance (FP32)	Double Precision Performance (FP64)
A100	ampere	40GB	1.6TB/s	765MHz	6912	432	19.5TFLOPS	9.7TFLOPS
A40	ampere	48GB	696GB/s	1305MHz	10752	336	37.4TFLOPS	584.6GFLOPS
V100	volta	32GB	900GB/s	1230MHz	5120	640	14TFLOPS	7TFLOPS
P100	pascal	16GB	732GB/s	1189MHz	3584	n/a	9.3TFLOPS	4.7TFLOPS