Lecture 10: Machine Learning Compiler and System

Some Notes

- First round of team meeting on Dec 1 and Dec 2.
- Lab 2 grades have been posted.
- Lab 3 is due this Sunday.
- Will have the office hour on Thursday 6-7pm.
- Midterm grade will be posted by this weekend.
- Will add extra-credit quiz

Recap

- Speculative Decoding
- Distributed DNN Training
- Distributed DNN Inference
- Federated Learning

Topics

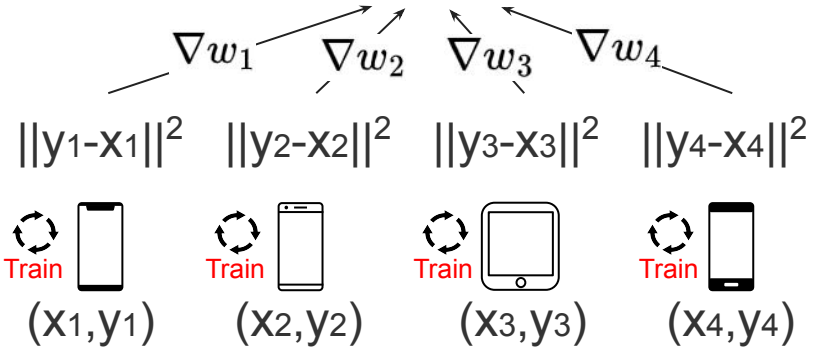
- Federated Learning (Continue)
- Machine Learning Compiler
- Machine Learning System

Federated Learning

- Training data: (x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4)

$$\sum_{i=1}^{4} ||y_i - F(x_i)||^2$$

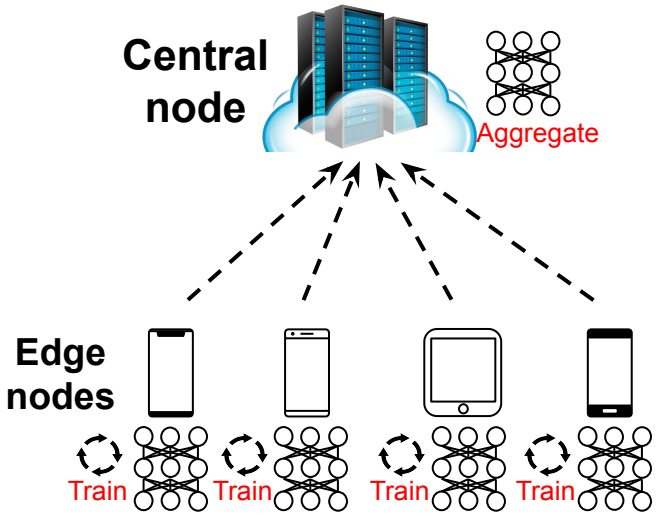
$$\nabla w = \frac{\nabla w_1 + \nabla w_2 + \nabla w_3 + \nabla w_4}{4}$$



- Non-iid training data distribution
- Heterogeneity among the edge devices
- Communication error

NYU SAI LAB

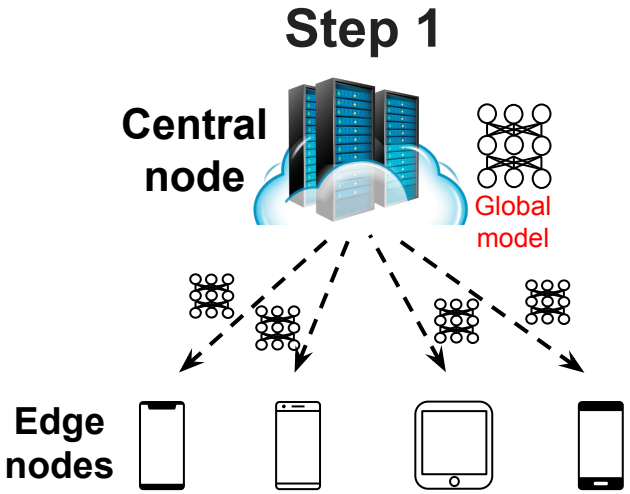
Federated Learning



- Federated learning is a machine learning technique that allows the training of models across multiple decentralized nodes holding local data samples, **without exchanging their data**.
- This approach enhances privacy, user can train the powerful DNN model **without sharing the dataset**.

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg

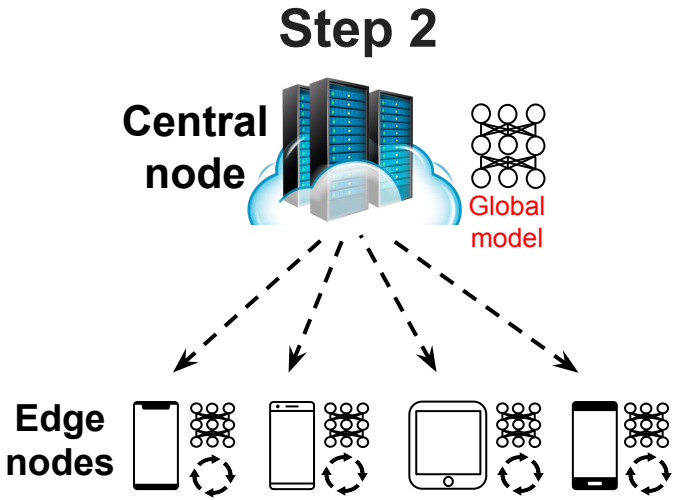


- A global model is initialized on the central node and sent to all participating nodes .

$$w_i = w_{global} \quad \text{For each i}$$

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg

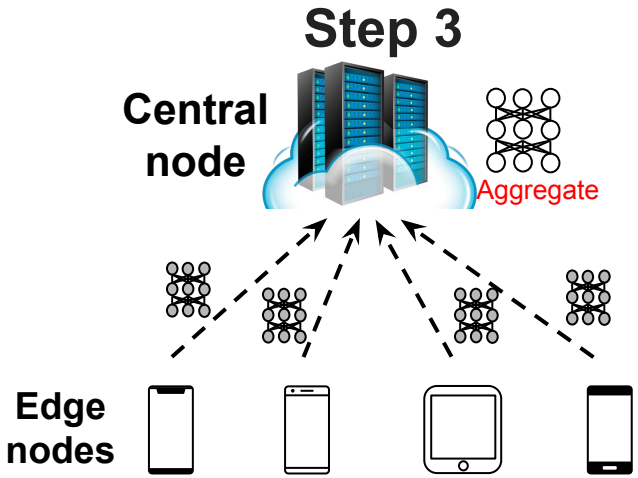


$$w_i' = \underset{w_i}{min} L(F(D_i), Y_i)$$

- Each node i trains the global model locally using its own data for a few epochs.
- The length of local training process may vary.

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg

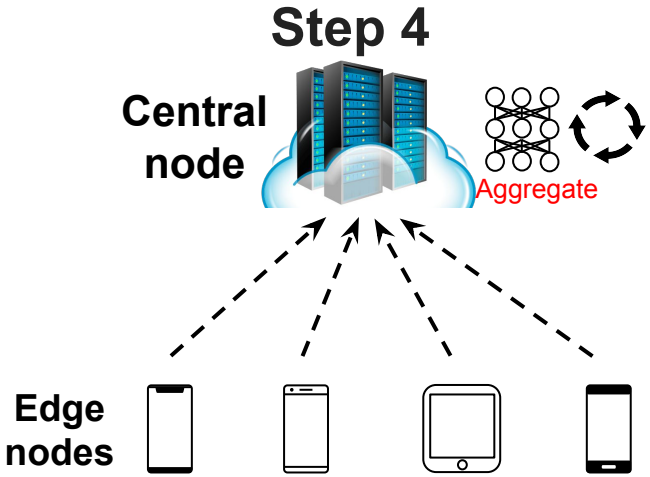


$$\Delta w_i = w - w_i$$

- Local updates are sent from each node to the central node.

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg



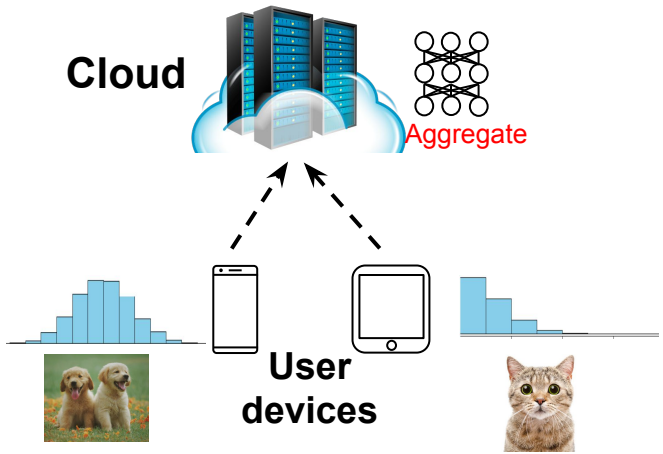
- The central node aggregates the local updates to update the global model.

$$w + \frac{1}{N} \sum_i \Delta w_i$$

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

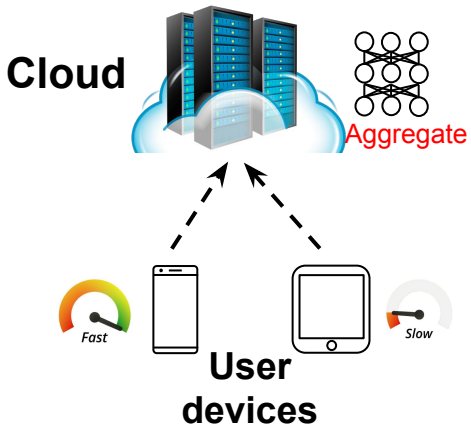
Federated Learning Problems: Non-IID

- However, in FL, the data distributed across different devices or clients is not drawn from the same statistical distribution.
- Unlike the scenario distributed training, where the training data are randomly distributed. For FL, the data stored in each device is highly biased.
- This may lead to significant accuracy degradation for the global model.



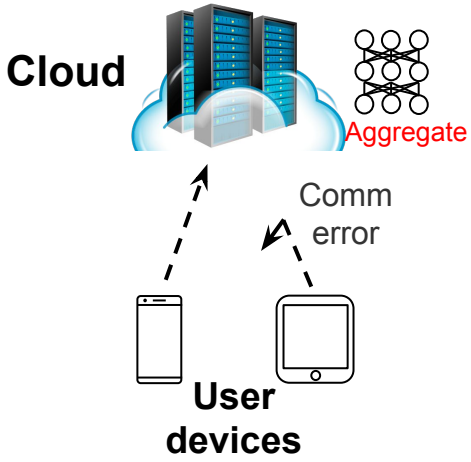
NYU SAI LAB

Federated Learning Problems: Heterogeneity



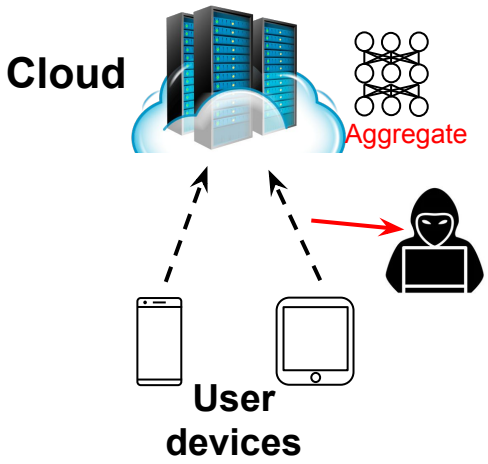
- Different edge device may have different processing speed.
- This will cause the total latency of each training round bottlenecked by the straggler, leading to a slow convergence of the training process.

Federated Learning Problems: Communication



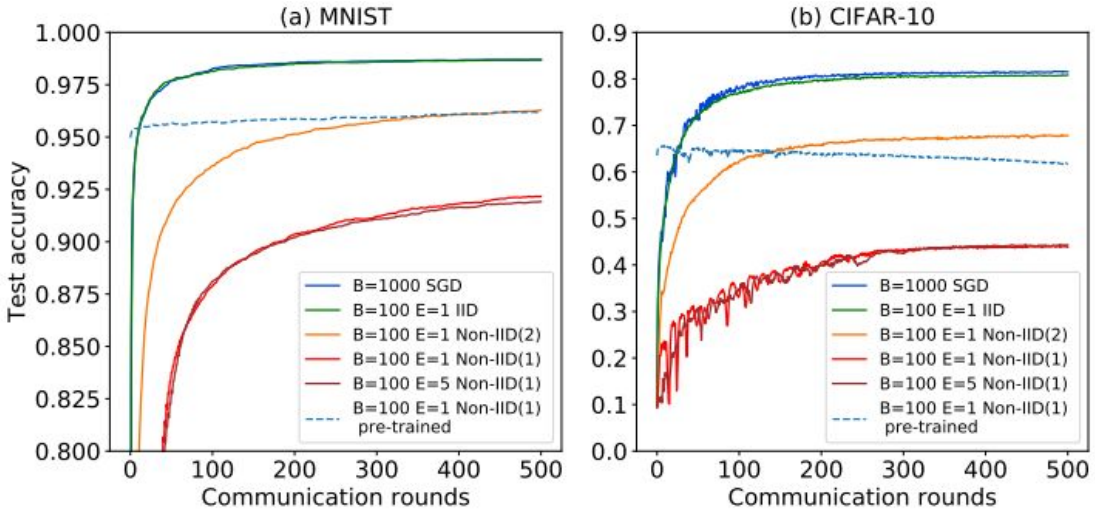
- The communication between edge devices and central cloud may incur transmission loss or error.
- This will impact the training latency and accuracy.

Federated Learning Problems: Privacy



- The attacker can leverage the transmitted gradient to reconstruct the original input training data.
- This will lead to privacy leakage.

Federated Learning with Non-iid Data

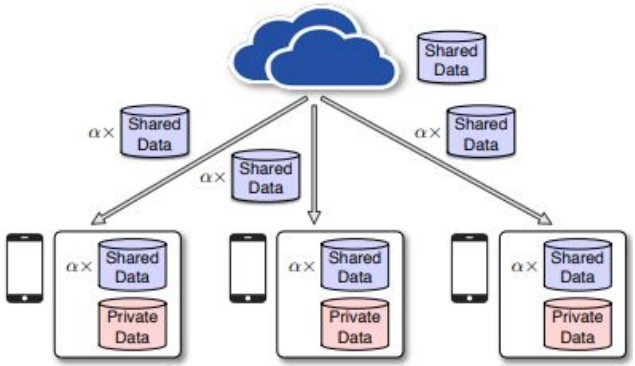


- The training sets are evenly partitioned into 10 clients.
- For IID setting, each client is randomly assigned a uniform distribution over 10 classes.
- For non-IID setting, the data is sorted by class and divided to create two extreme cases: (a) 1-class non-IID, where each client receives data partition from only a single class, and (b) 2-class non-IID, where the sorted data is divided into 20 partitions and each client is randomly assigned 2 partitions from 2 classes.

Zhao, Yue, et al. "Federated learning with non-iid data." *arXiv preprint arXiv:1806.00582* (2018).

Federated Learning with Non-iid Data

- We propose a data-sharing strategy to improve FedAvg with non-IID data by creating a small subset of data which is globally shared between all the edge devices.
- Experiments show that test accuracy can be increased by ~30% on CIFAR-10 dataset with only 5% globally shared data.



NYU SAI LAB

Zhao, Yue, et al. "Federated learning with non-iid data." *arXiv preprint arXiv:1806.00582* (2018).

FedProx

Algorithm 2 `FedProx` (Proposed Framework)

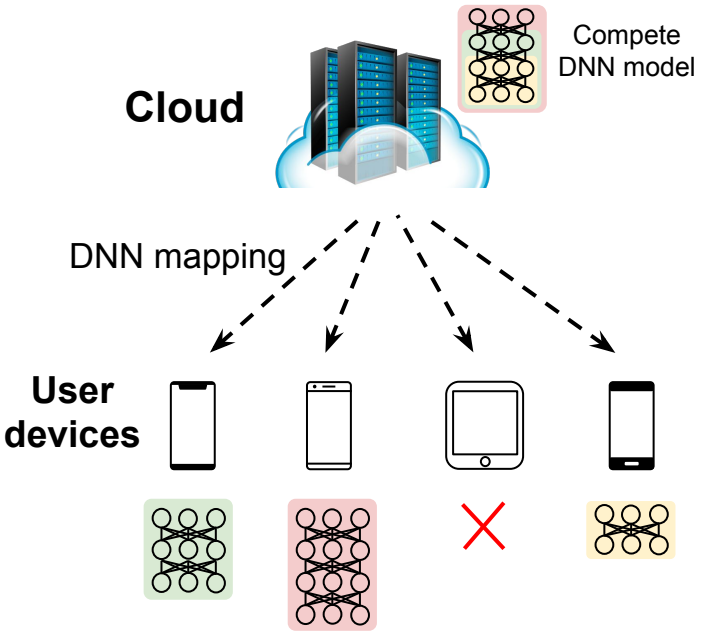
Input: $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ **do**
  Server selects a subset S_t of K devices at random (each device k is chosen with probability p_k)
  Server sends w^t to all chosen devices
  Each chosen device $k \in S_t$ finds a w_k^{t+1} which is a γ_k^t-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w \; h_k(w;\; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
  Each device $k \in S_t$ sends w_k^{t+1} back to the server
  Server aggregates the w's as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
end for

$$\min_w h_k(w;\; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$$

- We add an extra term to minimize the l_2 distance between the initial weight w_t and the learned weight w.
- This loss ensures that the learnt w is not too different from the original w.

Li, Tian, et al. "Federated optimization in heterogeneous networks." *Proceedings of Machine learning and systems* 2 (2020): 429-450.

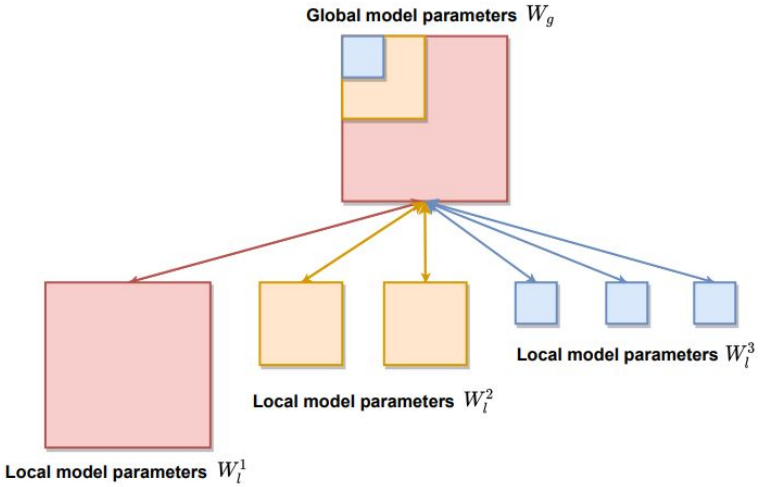
Federated Learning Problems: Heterogeneity



- End devices will have heterogeneous system configuration.
- HeteroFL partitions and assigns the DNN based on the processing power of each device.
- Each device only train a subset of the DNN model.

Diao, Enmao, Jie Ding, and Vahid Tarokh. "Heterofl: Computation and communication efficient federated learning for heterogeneous clients." *arXiv preprint arXiv:2010.01264* (2020).

HeteroFL



- Each edge device will be assigned with part of the neural network to perform local training based on its computational complexity.

NYU SAI LAB

Diao, Enmao, Jie Ding, and Vahid Tarokh. "Heterofl: Computation and communication efficient federated learning for heterogeneous clients." *arXiv preprint arXiv:2010.01264* (2020).

Federated Learning Problems: Communication

$$e(\mathbf{u}, \bar{\mathbf{u}}) = \frac{1}{N} \sum_{j=1}^{N} I(\mathrm{sgn}(u_j) = \mathrm{sgn}(\bar{u}_j))$$

- u_j denotes the sign of the model weight after local updates.
- Our solution dynamically identifies relevant local updates and excludes those irrelevant from being.
- Only the local device with high relevance will transmit their weight to the central server.

Luping, W.; Wei, W.; and Bo, L. 2019. Cmfl: Mitigating communication overhead for federated learning. In 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 954–964. IEEE.

FedMARL

Final model Accuray

Total Training Latency

Total Bandwidth

$$\max_{A} E\left[w_1 Acc(T) - w_2 \sum_{t \in T} H_t - w_3 \sum_{t \in T} B_t\right]$$

$$A = [a_n^t]$$

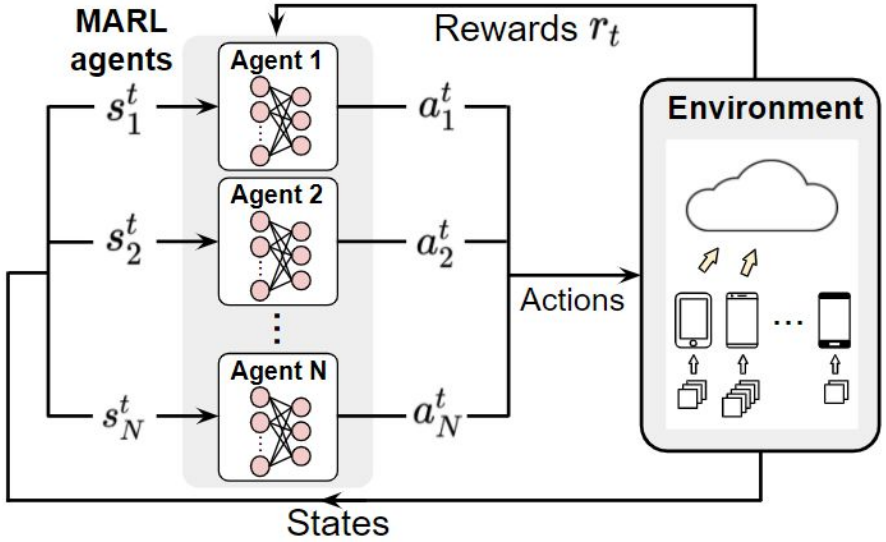
Client Selection

- Our objective is to maximize the accuracy of the global model while minimizing the total processing latency and communication cost.
- w1,w2,w3 are the importance of the objectives controlled by the FL application designers.
- The FL optimization problem is difficult to solve directly. We instead model the problem as a MARL problem.

NYU SAI LAB

Zhang, Sai Qian, Jieyu Lin, and Qi Zhang. "A multi-agent reinforcement learning approach for efficient client selection in federated learning." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. No. 8. 2022.

FedMARL

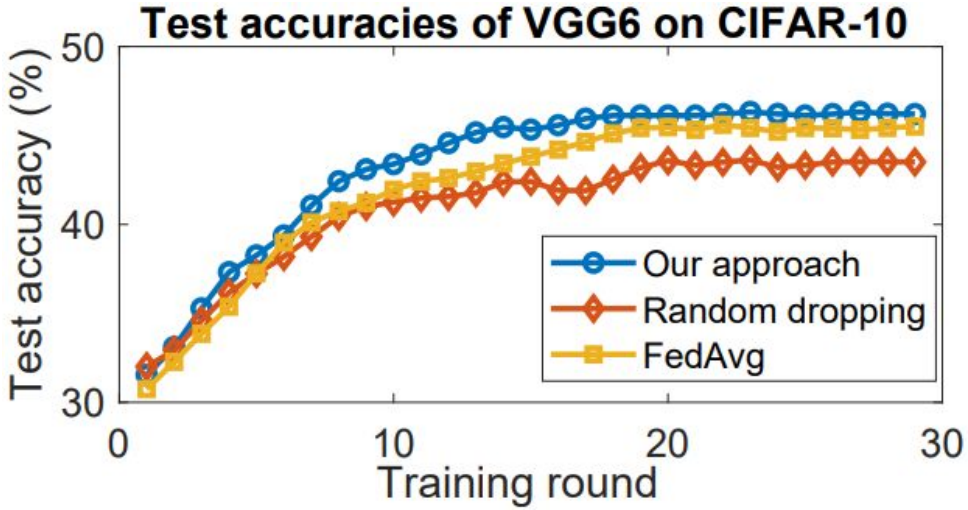


- In FedMarl, each client device n relies on an MARL agent at the central server to make its participation decision. Each MARL agent contains a simple two-layer Multi-layer perceptron (MLP) that is cheap to implement.

NYU SAI LAB

Zhang, Sai Qian, Jieyu Lin, and Qi Zhang. "A multi-agent reinforcement learning approach for efficient client selection in federated learning." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. No. 8. 2022.

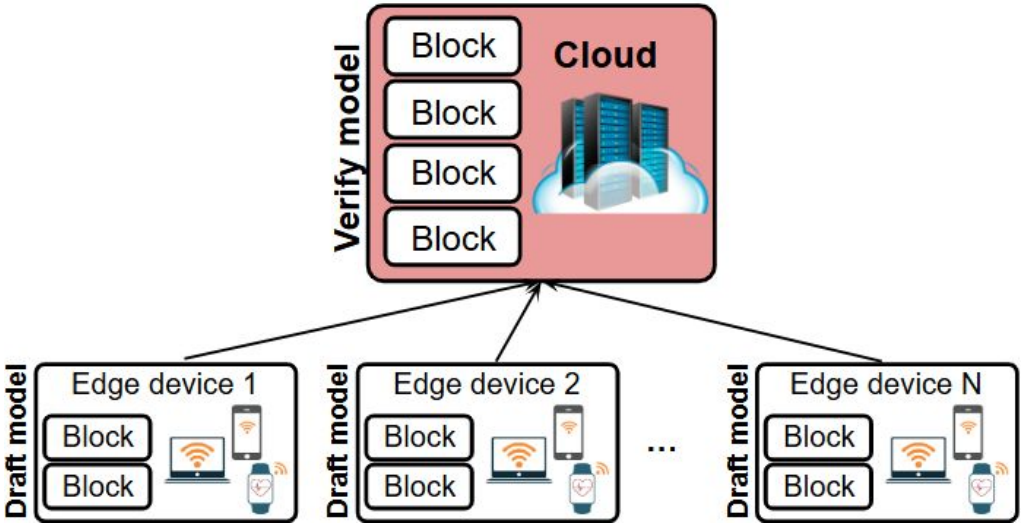
FedMARL



- Every random dropping is better than FedAvg.
- FedMarl is much better than random dropping and FedAvg.

NYU SAI LAB

Distributed LLM Inference with Speculative Decoding

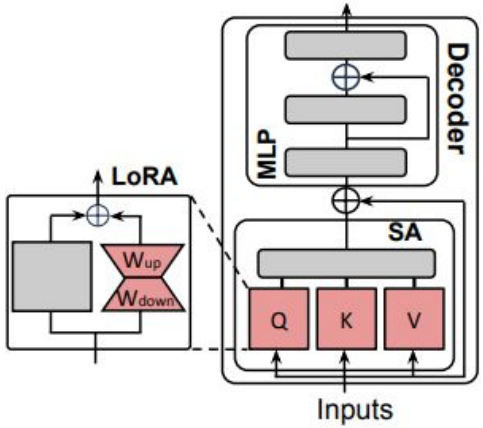


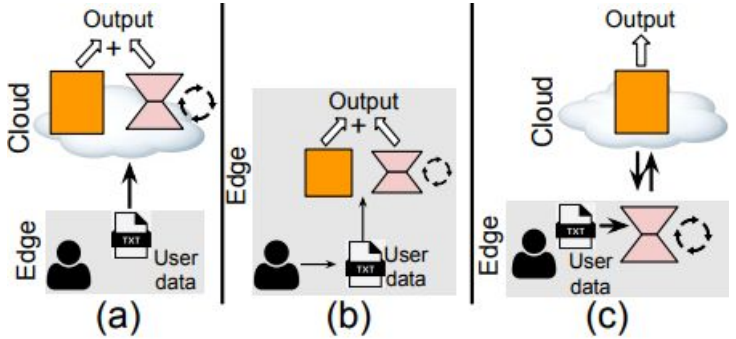
- Speculative decoding provides a lossless solution for distributed LLM execution.
- A small draft model can execute in sensor to produce draft tokens, and then send to aggregator for verification.

NYU SAI LAB

Li, Tianhong, et al. "Autoregressive Image Generation without Vector Quantization." *arXiv preprint arXiv:2406.11838* (2024).

DLoRA: Distributed parameter-efficient fine-tuning solution for large language model





- LLMs need to be finetuned in real time to better adapted to the downstream tasks.
- For resource-limited device, this finetuning process will consume a lot of latency and power.
- We propose distributed LLM finetuning techniques to deployed over the edge devices.

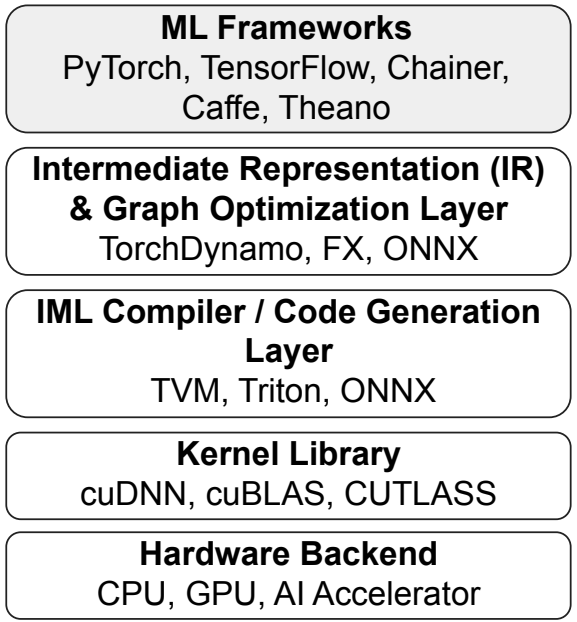
NYU SAI LAB

Gao, Chao, and Sai Qian Zhang. "Dlora: Distributed parameter-efficient fine-tuning solution for large language model." *EMNLP* (2024).

Topics

- Federated Learning (Continue)
- Deep Learning Software/Compiler
- Machine Learning System

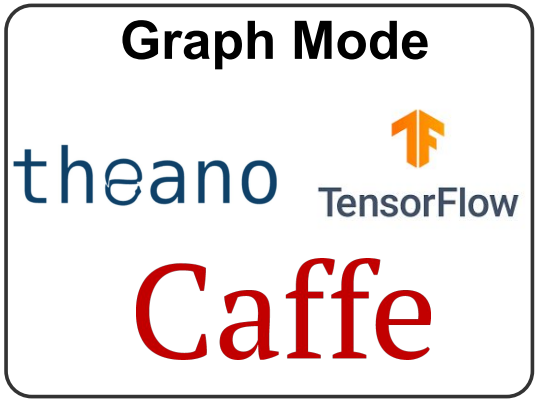
ML Software–Compiler stack

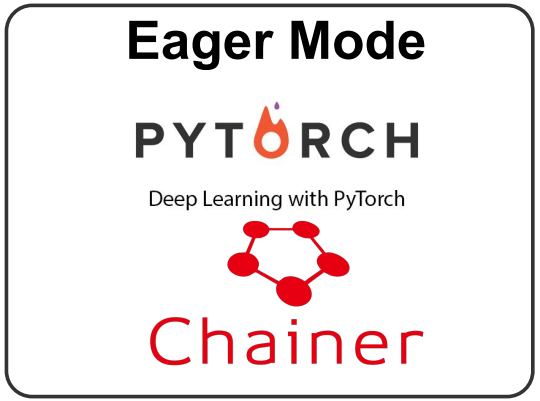


- High-level interfaces where models are defined, trained, and debugged.
- Converts model definitions into an analyzable graph IR for optimization.
- Translates the IR into optimized kernels and executable code for specific devices.
- Executes optimized kernels, manages device memory, and provides primitives.
- The physical devices executing the compiled and optimized models.

NYU SAI LAB

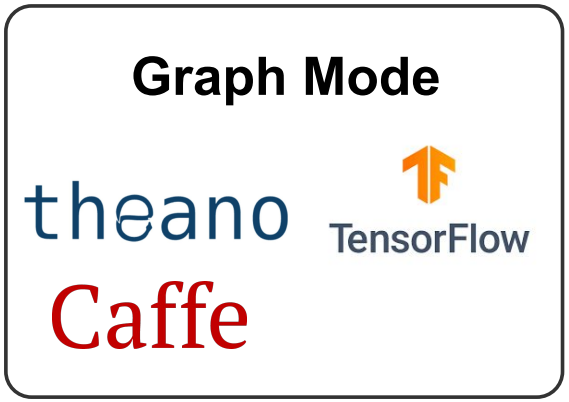
Deep Learning Frameworks





- **Graph mode:** where they expose a graph building API that requires users to first construct a graph and then later execute that graph.
- **Eager mode:** meaning operations are executed immediately as they are called in Python, rather than being added to a static computation graph (as in early TensorFlow).

Deep Learning Frameworks



- **Graph mode:** where they expose a graph building API that requires users to first construct a graph and then later execute that graph.
- **Eager mode:** meaning operations are executed immediately as they are called in Python, rather than being added to a static computation graph (as in early TensorFlow).

Pytorch

```
class LinearLayer(Module):
   def __init__(self, in_sz, out_sz):
      super().__init__()
      t1 = torch.randn(in_sz, out_sz)
      self.w = nn.Parameter(t1)
      t2 = torch.randn(out_sz)
      self.b = nn.Parameter(t2)

   def forward(self, activations):
      t = torch.mm(activations, self.w)
      return t + self.b
```

```
class FullBasicModel(nn.Module):
   def __init__(self):
      super().__init__()
      self.conv = nn.Conv2d(1, 128, 3)
      self.fc = LinearLayer(128, 10)

   def forward(self, x):
      t1 = self.conv(x)
      t2 = nn.functional.relu(t1)
      t3 = self.fc(t1)
      return nn.functional.softmax(t3)
```

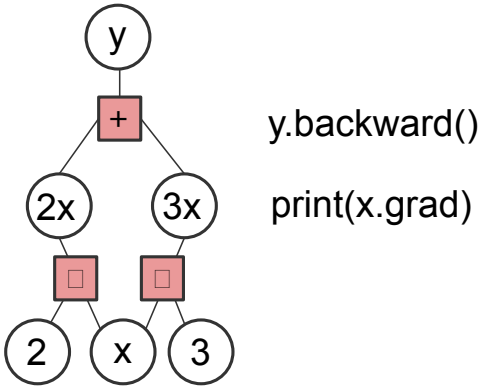
```
discriminator = create_discriminator()
generator = create_generator()
optimD = optim.Adam(discriminator.parameters())
optimG = optim.Adam(generator.parameters())

def step(real_sample):
  # (1) Update Discriminator
  errD_real = loss(discriminator(real_sample), real_label)
  errD_real.backward()
  fake = generator(get_noise())
  errD_fake = loss(discriminator(fake.detach(), fake_label)
  errD_fake.backward()
  optimD.step()
  # (2) Update Generator
  errG = loss(discriminator(fake), real_label)
  errG.backward()
  optimG.step()
```

- Inherit from Chainer.
- DNN models can be programmed. API is defined to make user freely select the layer configuration.
- This “everything is a just a program” philosophy is not limited to just the models, and applies to optimizers and data loaders as well.

Pytorch

- PyTorch allows for bidirectional exchange of data with external libraries.
  - For example, it provides a mechanism to convert between NumPy arrays and PyTorch tensors using the torch.from_numpy() function and .numpy() tensor method.
  - Pytorch has implemented an automatic differentiation functionality to automatically performance the gradient computation.

```
import torch
x = torch.tensor(2.0, requires_grad=True)
y = x*2+x*3
```



NYU SAI LAB

Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019).

Pytorch

- What if the function is not differentiable? How to generate gradient?
  - Subgradients or Piecewise Derivatives
  - Approximation Gradients
  - Customized gradient definition
  - Compilation Error

Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019).

Pytorch

- Subgradients or Piecewise Derivatives

```
import torch
x = torch.tensor(0.0, requires_grad=True)
y = torch.abs(x)
y.backward()
print(x.grad)  # prints 0.0
```

- Straight-Through Estimator (STE)

```
def binary_step_ste(x):
    y = (x > 0).float()
    y.backward = lambda grad: grad  #
approximate grad as 1
```

- Customized gradient definition

```
class CustomOp(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)
        return torch.sign(x)
    @staticmethod
    def backward(ctx, grad_output):
        x, = ctx.saved_tensors
        grad_input = grad_output * (x.abs()
<= 1).float()  # custom surrogate
        return grad_input
```


Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019).

Pytorch

- PyTorch maintains a strict separation between its control (i.e. program branches, loops) and data flow (i.e. tensors and the operations performed on them).
- The resolution of the control flow is handled by Python and optimized C++ code executed on the host CPU, and result in a linear sequence of operator invocations on the device. Operators can be run either on CPU or on GPU.
- Memory allocation is first performed on the CPU, which subsequently manages the allocation and mapping on the GPU.

Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019).

ML Software–Compiler stack

ML Frameworks
PyTorch, TensorFlow, Chainer, Caffe, Theano

Intermediate Representation (IR) & Graph Optimization Layer
TorchDynamo, FX, ONNX

ML Compiler / Code Generation Layer
TVM, Triton, ONNX

Kernel Library
cuDNN, cuBLAS, CUTLASS

Hardware Backend
CPU, GPU, AI Accelerator

- High-level interfaces where models are defined, trained, and debugged.
- Converts model definitions into an analyzable graph Intermediate Representation (IR) for optimization.
- Translates the IR into optimized kernels and executable code for specific devices.
- Executes optimized kernels, manages device memory, and provides primitives.
- The physical devices executing the compiled and optimized models.

TensorFlow

- Open-source ML framework by Google (2015)
- Supports CPU, GPU, TPU acceleration
  - Neural networks (CNNs, RNNs, Transformers)
  - Reinforcement learning
  - Signal processing and scientific computing
- TensorFlow builds a **static computation graph** (predefined dataflow)
  - Graph nodes = operations
  - Edges = data (tensors)

- Optimizable and deployable on different hardware
- Enables **parallelism** and **graph-level optimizations**

Abadi, Martín, et al. "{TensorFlow}: a system for {Large-Scale} machine learning." *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016.

TensorFlow

```
import tensorflow as tf

x = tf.constant(3.0)
y = tf.constant(4.0)
z = x * y  # <-- nothing runs yet, just builds the graph
print(z)
```

```
Tensor("mul:0", shape=(), dtype=float32)
```

```
with tf.Session() as sess:
    print(sess.run(z))  # Output: 12.0
```

```
import tensorflow as tf

x = tf.constant(3.0)
y = tf.constant(4.0)
z = x * y  # executed immediately
print(z)
```

```
tf.Tensor(12.0, shape=(), dtype=float32)
```

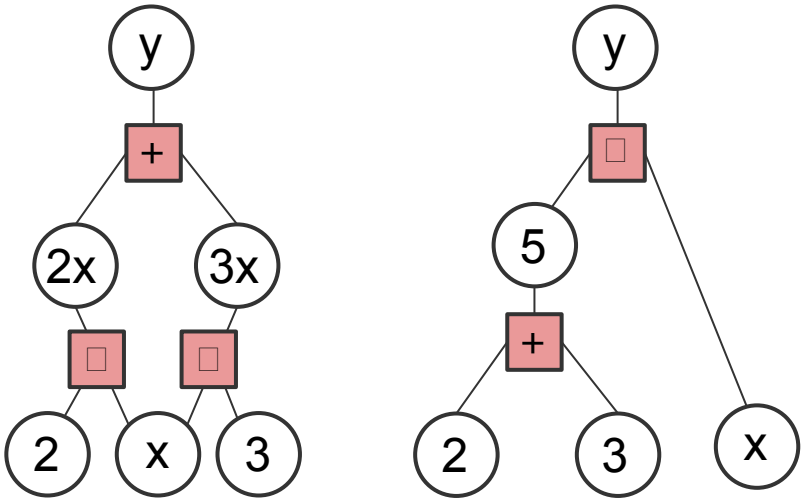

Abadi, Martín, et al. "{TensorFlow}: a system for {Large-Scale} machine learning." *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016.

Graph-level Optimization

```
import torch
x = torch.tensor(2.0, requires_grad=True)
y = x*2+x*3
```

```
import torch
x = torch.tensor(2.0, requires_grad=True)
y = x*(2+3)
```

The downside of eager mode frameworks is that they make it harder to apply graph-level optimizations through compilers.



Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019).

Torch.fx

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class RedundantModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(4, 8)
        self.fc2 = nn.Linear(8, 8)
        self.fc3 = nn.Linear(8, 2)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = F.relu(x)     # redundant ReLU
        x = self.fc2(x)
        x = F.relu(x)
        return self.fc3(x)
```

```
from torch.fx import symbolic_trace

model = RedundantModel()
gm = symbolic_trace(model)
print(gm.graph)
```

```
graph():
    %x = placeholder[target=x]
    %fc1 = call_module[target=fc1](args=(%x,), kwargs={})
    %relu_1 = call_function[target=torch.nn.functional.relu](args=(%fc1,), kwargs={})
    %relu_2 = call_function[target=torch.nn.functional.relu](args=(%relu_1,), kwargs={})
    %fc2 = call_module[target=fc2](args=(%relu_2,), kwargs={})
    %relu_3 = call_function[target=torch.nn.functional.relu](args=(%fc2,), kwargs={})
    %fc3 = call_module[target=fc3](args=(%relu_3,), kwargs={})
    return %fc3
```


Ansel, Jason, et al. "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation." *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 2024.

Torch.fx

```
def remove_redundant_relu(gm):
    new_graph = torch.fx.Graph()
    env = {}

    last_op = None
    for node in gm.graph.nodes:
        if node.op == 'call_function' and node.target == torch.nn.functional.relu:
            # Skip consecutive ReLU nodes
            if last_op == torch.nn.functional.relu:
                continue
        new_node = new_graph.node_copy(node, lambda x: env[x])
        env[node] = new_node
        last_op = node.target if node.op == 'call_function' else None

    gm_optimized = torch.fx.GraphModule(gm, new_graph)
    return gm_optimized
```

Torch.fx provides the flexibility to modify and transform the computation graph, enabling performance and efficiency improvements.

Ansel, Jason, et al. "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation." *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 2024.

ML Software–Compiler stack

ML Frameworks
PyTorch, TensorFlow, Chainer, Caffe, Theano

Intermediate Representation (IR) & Graph Optimization Layer
TorchDynamo, FX, ONNX

ML Compiler / Code Generation Layer
TVM, Triton, ONNX

Kernel Library
cuDNN, cuBLAS, CUTLASS

Hardware Backend
CPU, GPU, AI Accelerator

- High-level interfaces where models are defined, trained, and debugged.
- Converts model definitions into an analyzable graph IR for optimization.
- Translates the IR into optimized kernels and executable code for specific devices.
- Executes optimized kernels, manages device memory, and provides primitives.
- The physical devices executing the compiled and optimized models.

Triton

- An open-source compiler and language (originally by Harvard/OpenAI, now integrated into PyTorch)
- Allows writing custom GPU kernels in Python, achieving CUDA-level performance with much simpler code
- **Automatic tiling & vectorization** for performance portability
- **Fusion-friendly**: easily integrates with PyTorch's graph optimizers

Tillet, Philippe, Hsiang-Tsung Kung, and David Cox. "Triton: an intermediate language and compiler for tiled neural network computations." *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 2019.

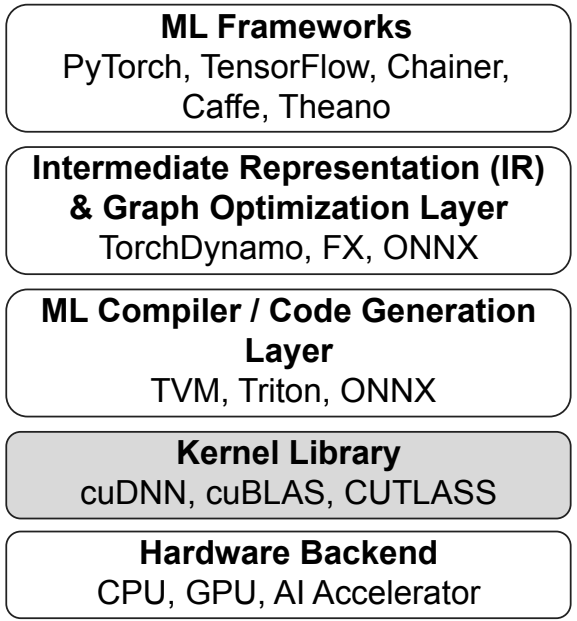
Triton

```
import triton
import triton.language as tl

@triton.jit
def matmul_kernel(a_ptr, b_ptr, c_ptr, M, N, K):
    pid = tl.program_id(0)
    row = pid * 16 + tl.arange(0, 16)
    col = tl.arange(0, 16)
    a = tl.load(a_ptr + row[:, None] * K + tl.arange(0, K))
    b = tl.load(b_ptr + tl.arange(0, K)[:, None] * N + col)
    c = tl.dot(a, b)
    tl.store(c_ptr + row[:, None] * N + col, c)
```

- Writing efficient GPU kernels in CUDA is complex and error-prone.
- Researchers often need custom fused kernels beyond what cuDNN/cuBLAS offer.
- Frameworks like PyTorch needed a flexible but high-performance solution.
- Triton bridges this gap: Python-like syntax with compiler-grade optimization.

Tillet, Philippe, Hsiang-Tsung Kung, and David Cox. "Triton: an intermediate language and compiler for tiled neural network computations." *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 2019.

ML Software–Compiler stack



- High-level interfaces where models are defined, trained, and debugged.
- Converts model definitions into an analyzable graph IR for optimization.
- Translates the IR into optimized kernels and executable code for specific devices.
- Executes optimized kernels, manages device memory, and provides primitives.
- The physical devices executing the compiled and optimized models.

NYU SAI LAB

cuDNN (CUDA Deep Neural Network Library)

- High-performance GPU library for deep learning primitives
- Optimized implementations for:
  - Convolutions, pooling, normalization
  - Activation functions (ReLU, tanh, sigmoid)
  - RNN/LSTM layers
- Automatically used by TensorFlow, PyTorch, and JAX
- Enables Tensor Cores, mixed precision, and algorithm autotuning for speedups

Chetlur, Sharan, et al. "cudnn: Efficient primitives for deep learning." *arXiv preprint arXiv:1410.0759* (2014).

cuBLAS (CUDA Basic Linear Algebra Subprograms)

- GPU-accelerated version of BLAS (Basic Linear Algebra Subroutines)
- Provides fast operations for:
  - Matrix–vector and matrix–matrix multiplications (`GEMM`)
  - Vector scaling, addition, dot products
- Underpins many deep learning operations (e.g., dense layers, attention mechanisms)
- Also supports **FP16 / BF16 precision** for performance on modern GPUs
- cuDNN accelerates *deep learning-specific ops*, while cuBLAS accelerates *general linear algebra*.
- Both are critical layers in the GPU software stack that make frameworks like TensorFlow and PyTorch fast.

Topics

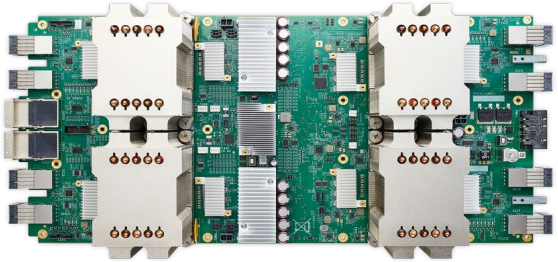
- Federated Learning (Continue)
- Deep Learning Software/Compiler
- Hardware System Overview

Hardware Support for DNN

- GPU is better than CPU in terms of throughput for both Neural Network training and inference.
  - GPU leverages the highly parallelized architecture of its computing units to handle computational intensive operations.
  - GPU has 10x-20x higher throughput than CPU.
- However, GPU:
  - General purpose.
  - Power consumption and latency is high.
  - Does not support sophisticated pruning and quantization algorithms.

Hardware Support for DNN

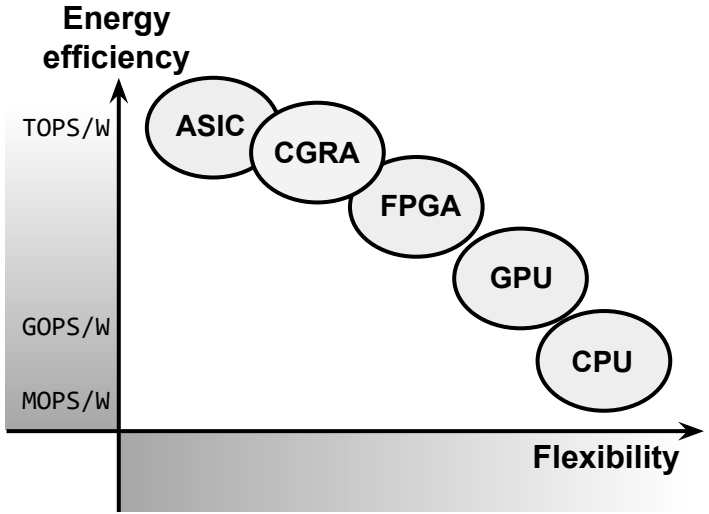
- ASIC-based implementations have been recently explored to accelerate the DNN inference.
  - Google's TPU, Apple's Neural Engine, Cerebras AI chip, …
- FPGA-based accelerators for DNN inference have been recently developed.
  - Has good programmability and flexibility
  - Short development cycles
  - Can be used as a benchmark before implementing on ASIC



Tensor Processing Unit (Google)

Alveo Accelerator Card (Xilinx)

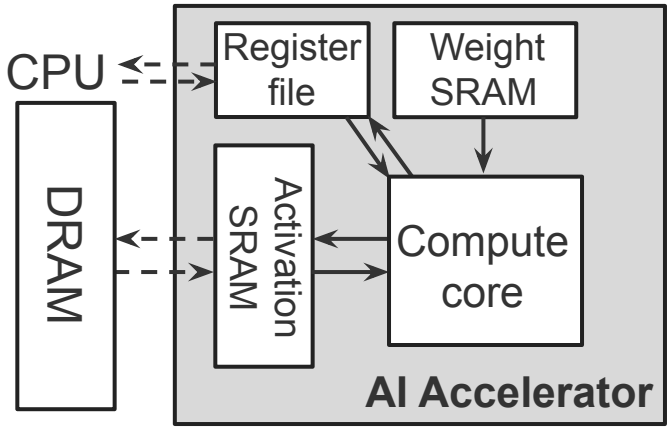
Flexibility & Performance

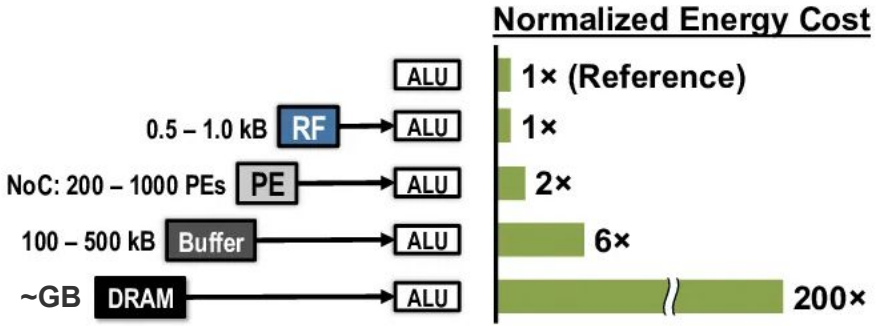


- ASIC offers the highest energy efficiency but is only suitable for specific applications.
- The CPU is a general-purpose processor but has the lowest energy efficiency.

AI Accelerator

- Making any chip is a costly, difficult and lengthy process typically done by teams of 10 to 1000's of people depending on the size and complexity of the chip.

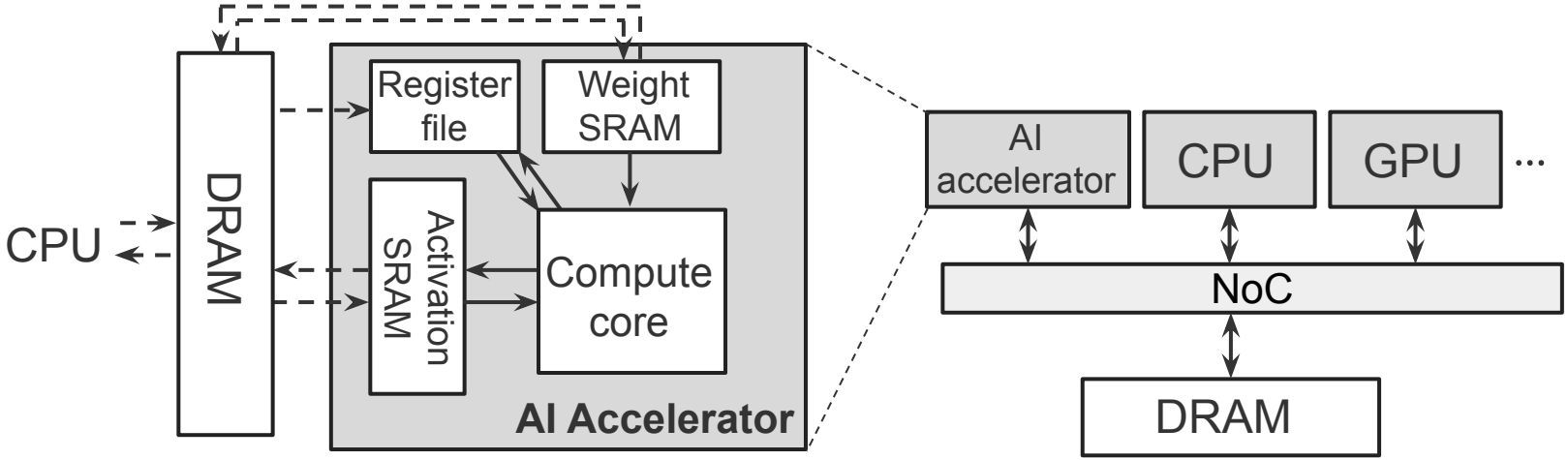




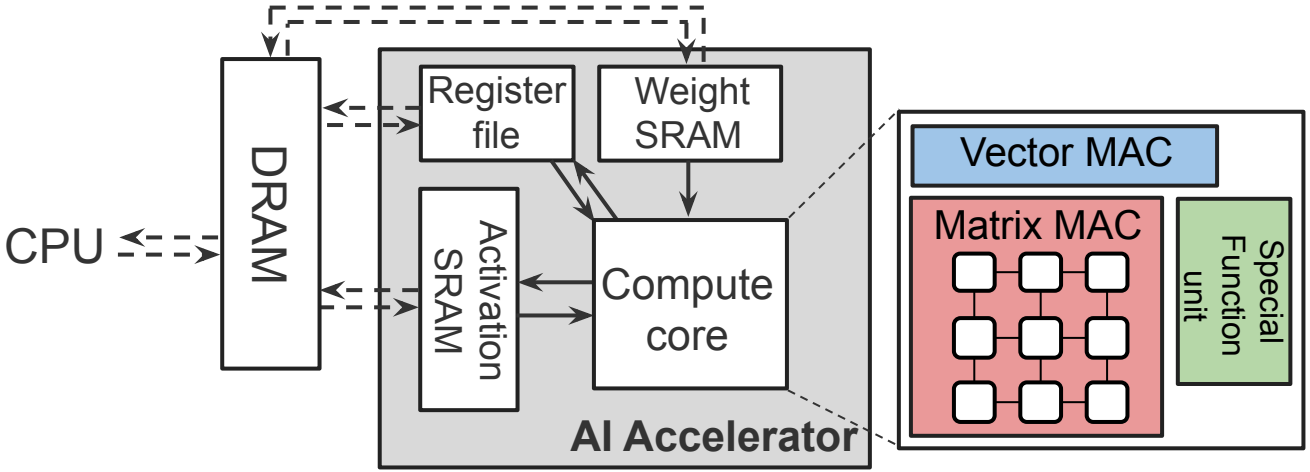
NYU SAI LAB

AI Accelerator

- The AI accelerator can execute part of the machine code that is related to the AI workload.



AI Accelerator



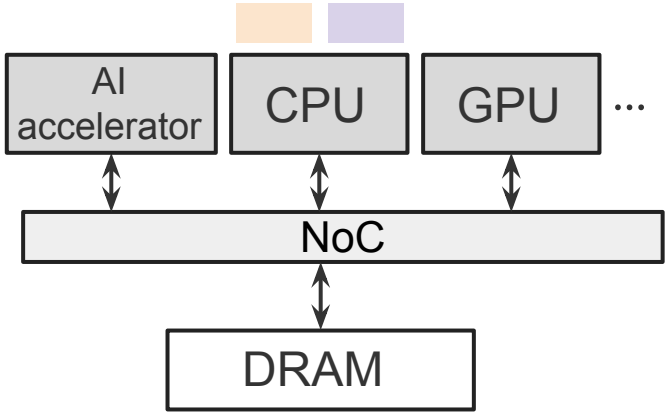
- The compute core consists of Multiply and accumulator (MAC) engine for 2D matrix multiplication.
- It also contains vector multiplier MAC as well as special function unit.

AI Accelerator

```
import torch
import torch.nn as nn
input = torch.randn(1, 1, 5, 5)
conv = nn.Conv2d(1, 1, kernel_size=3)
output = conv(input)
print(output)
```

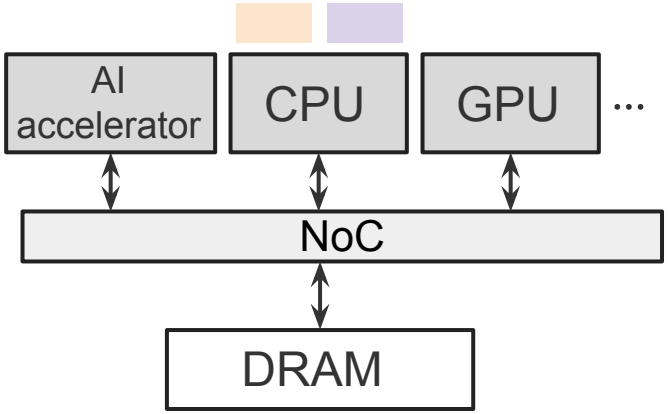
Compiler

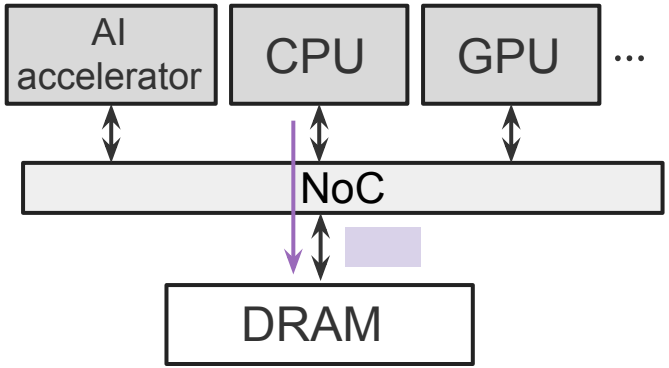
```
01001000 10001001 11011000
01001000 10000011 11000000 00001010
01001000 10000011 11101011 00000101
01001000 00111001 11011000
01110100 00000101
01001000 10001001 11000001
01001000 01101001 11001001 00000010
01001000 10001001 00001111
11101011 00000011
```



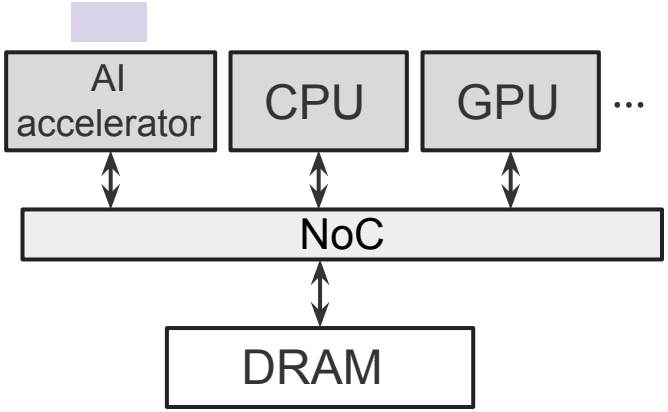
NYU SAI LAB

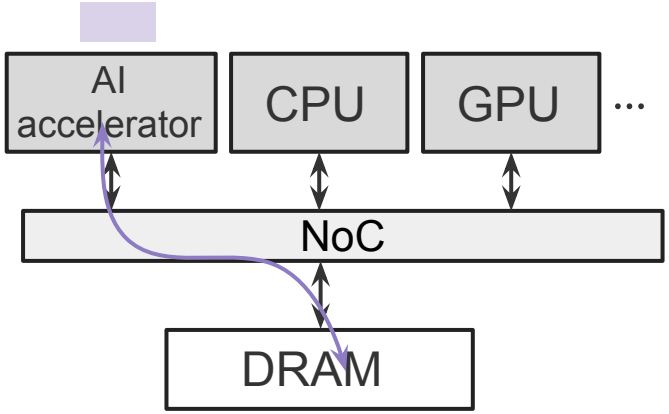
AI Accelerator



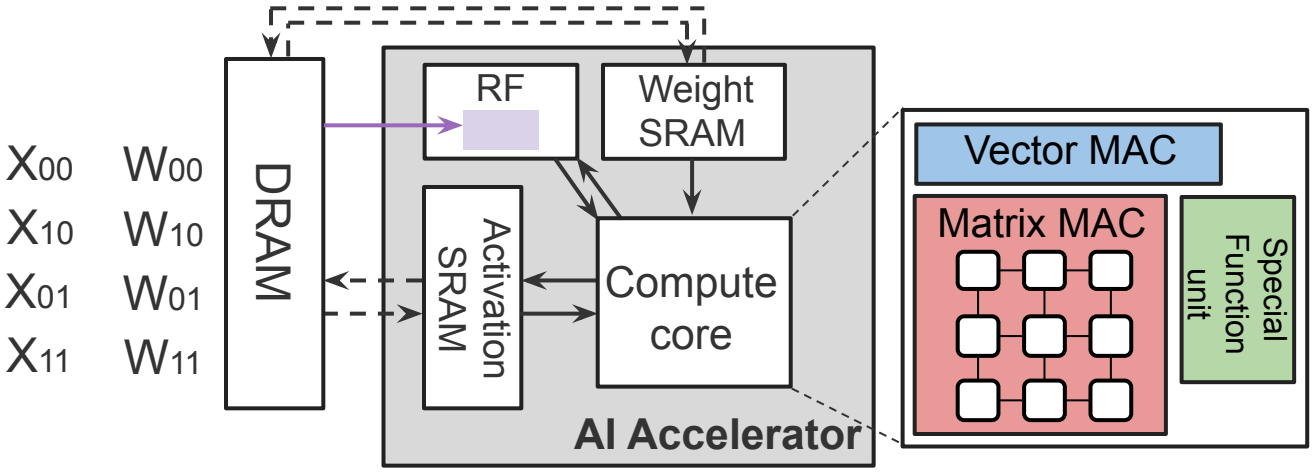


AI Accelerator



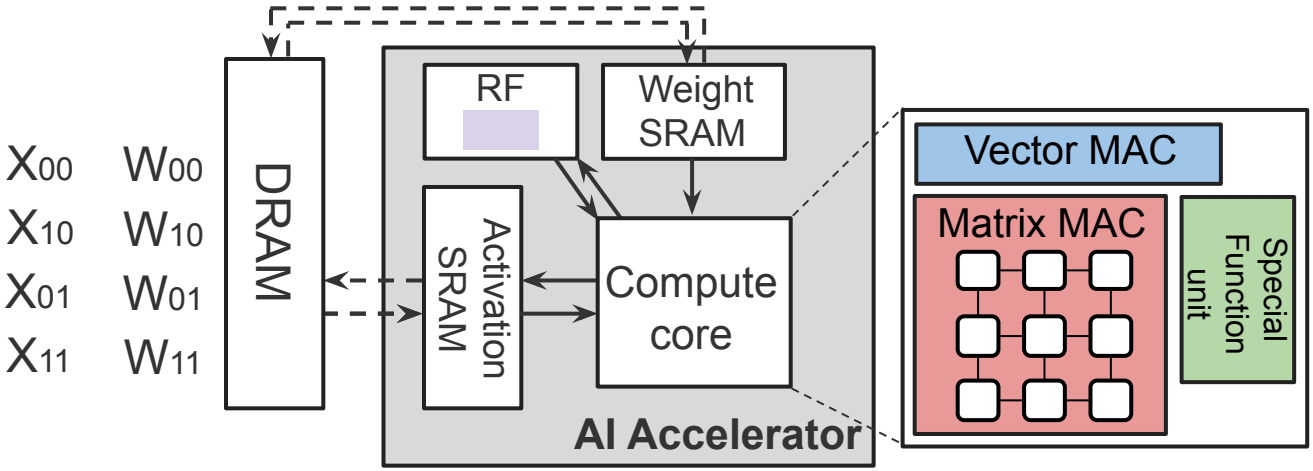


AI Accelerator



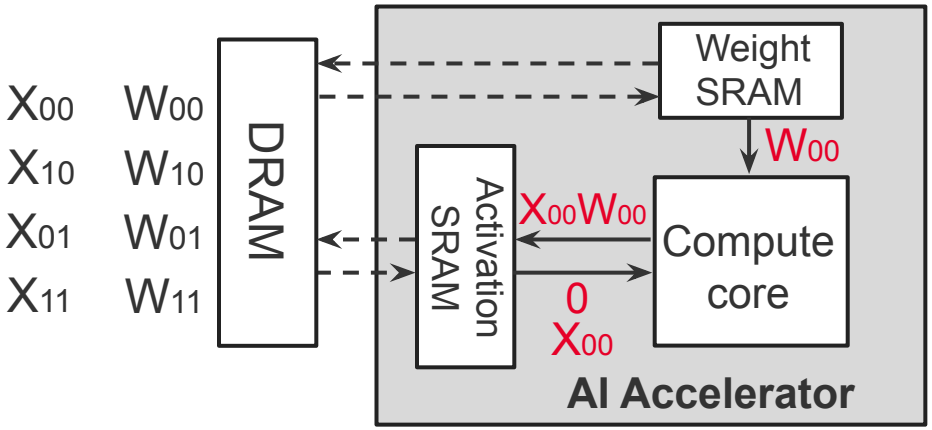
$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

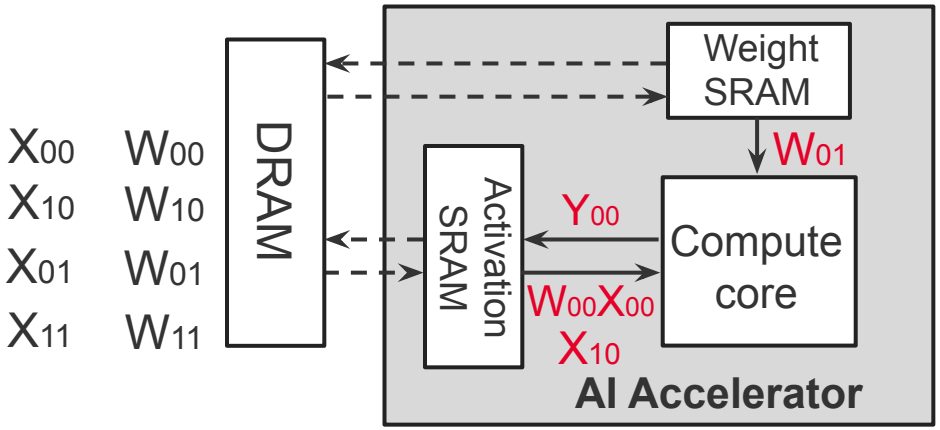
AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

NYU SAI LAB

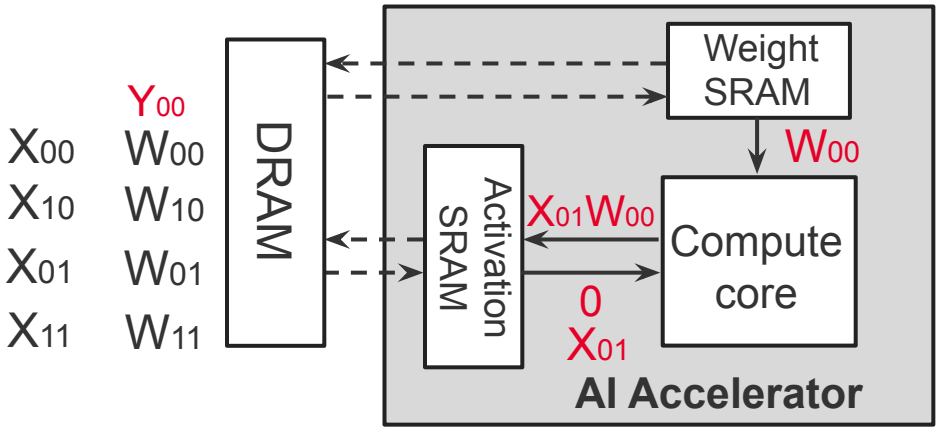
AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

NYU SAI LAB

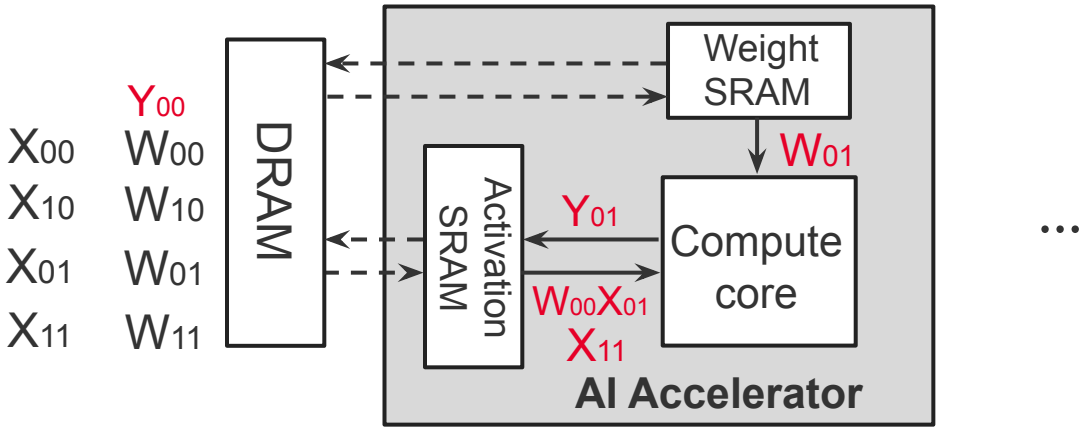
AI Accelerator



$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

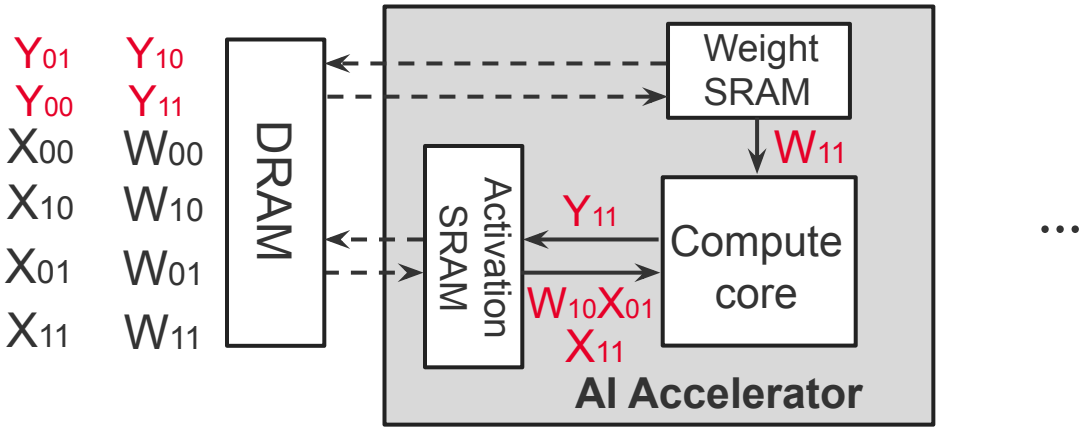
NYU SAI LAB

AI Accelerator



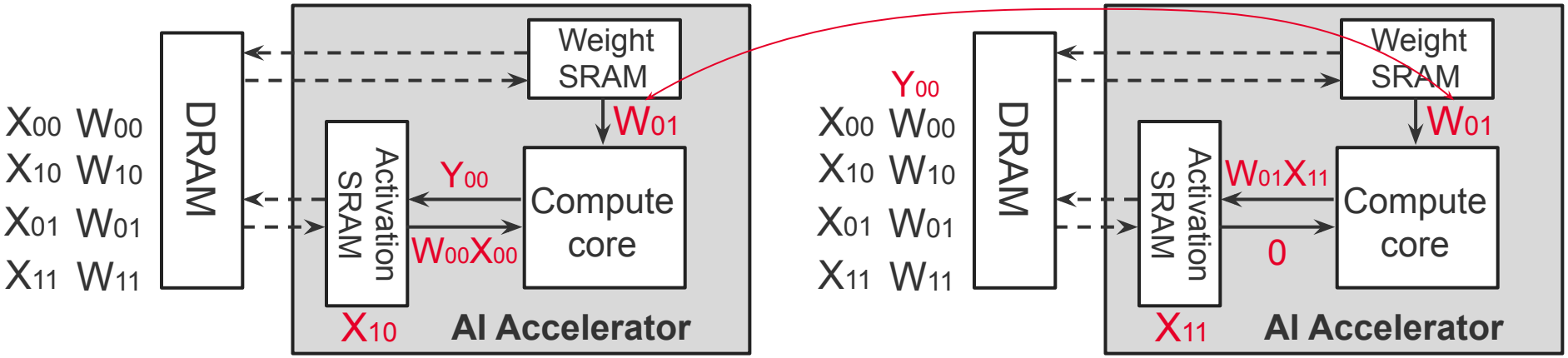
$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

AI Accelerator



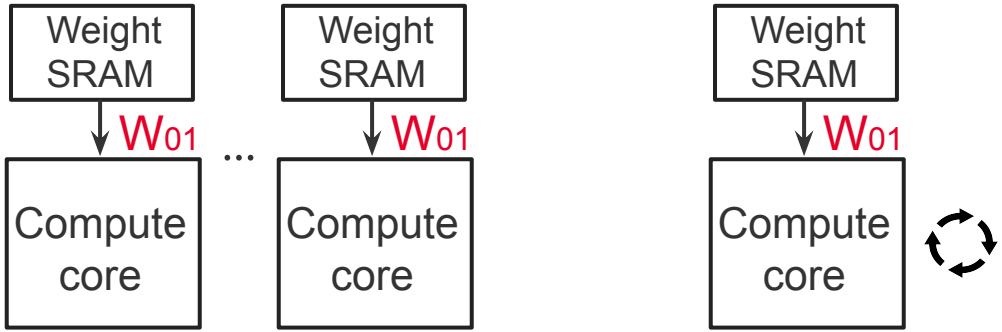
$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} \times \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} = \begin{bmatrix} W_{00}X_{00}+W_{01}X_{10} & W_{00}X_{01}+W_{01}X_{11} \\ W_{10}X_{00}+W_{11}X_{10} & W_{10}X_{01}+W_{11}X_{11} \end{bmatrix} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

Memory Access Reduction



- The computation and memory access pattern can be changed to minimize the computational cost without impacting the results.

Memory Access Reduction



- It is preferable to minimize memory access by maximizing the reuse of loaded data.
- We will explore methods for scheduling neural network layer accesses to minimize memory usage in the next lecture.