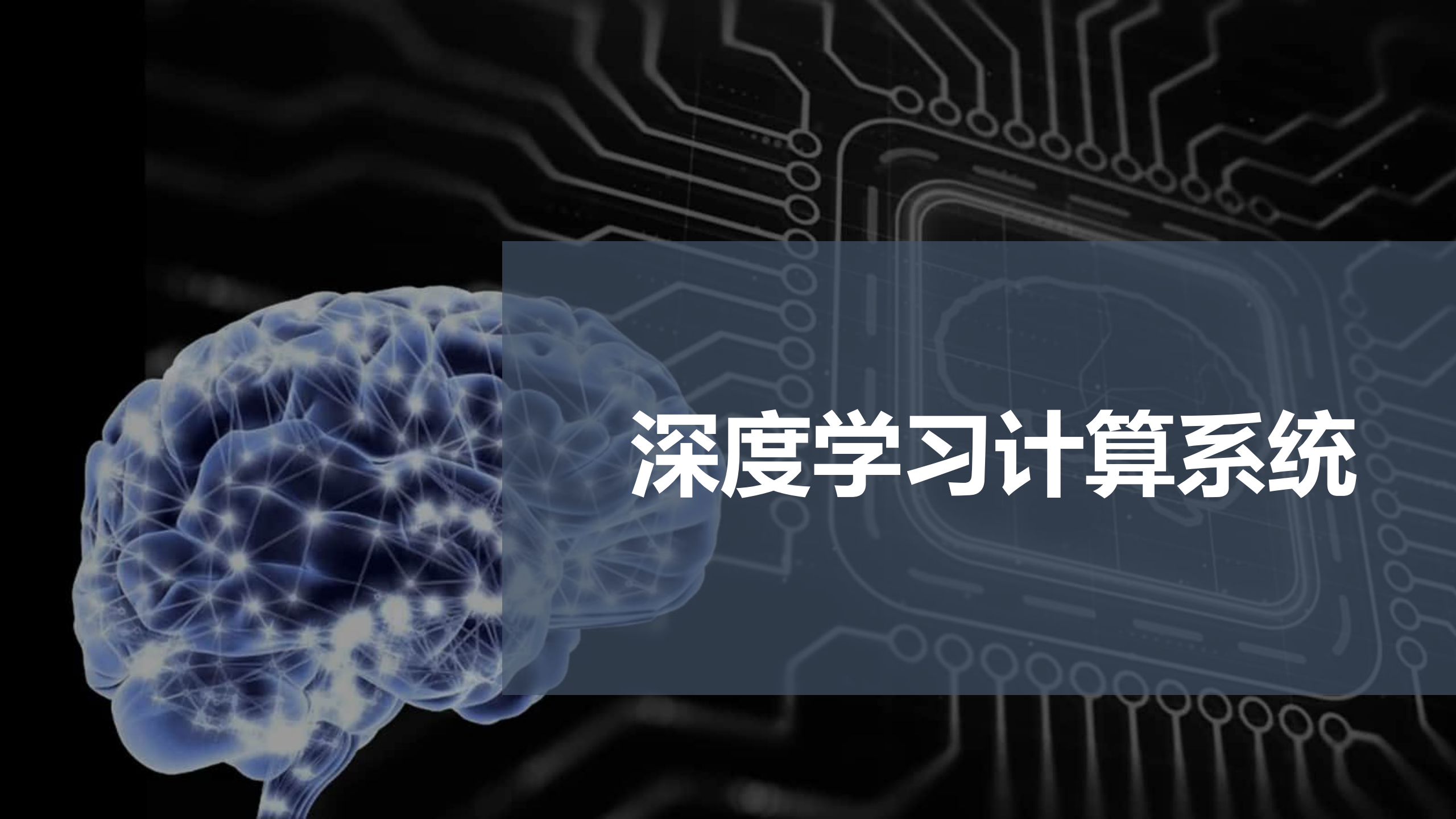
复习

- 正则化是深度学习中的一种常用技术，请简述正则化的意义，并列举几种常用的正则化方法；
- 比较Batch Normalization（批量归一化）和Layer Normalization（层归一化）的差异。

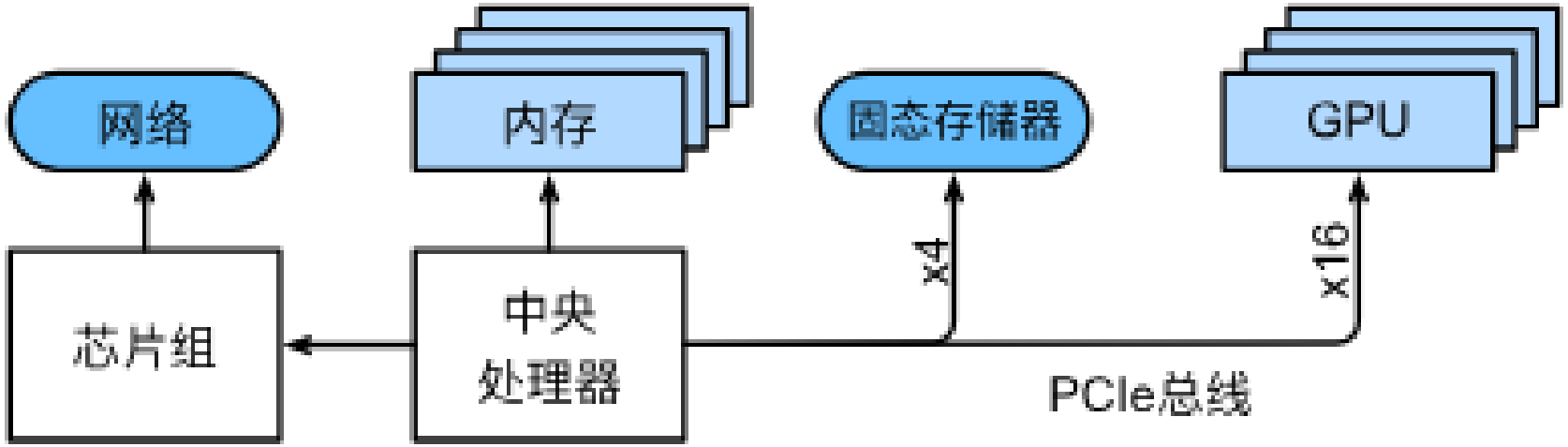


深度学习计算系统

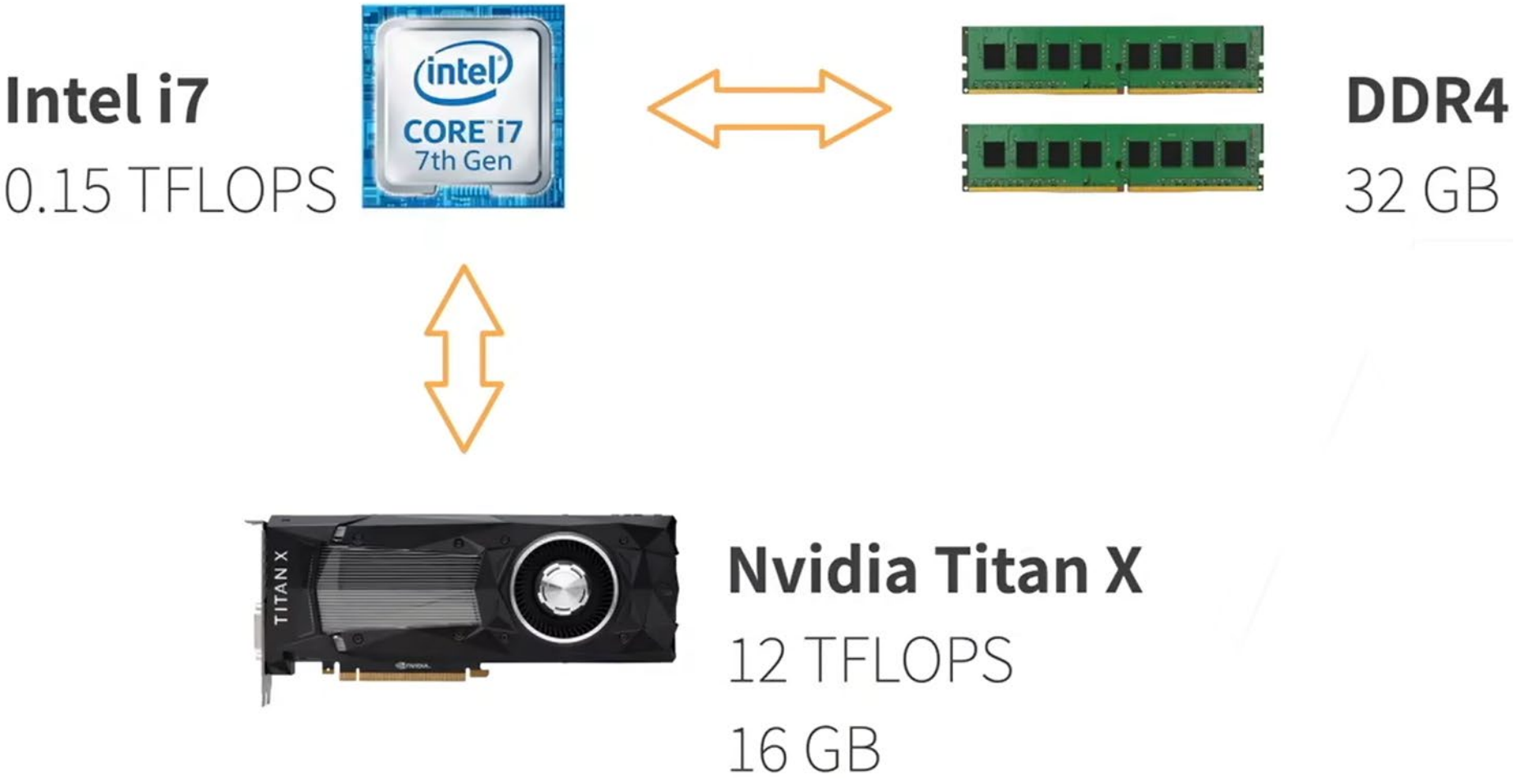
深度学习计算系统

- 深度学习硬件系统
- 深度学习框架
- 深度学习编译
- 深度学习部署
- MLOps

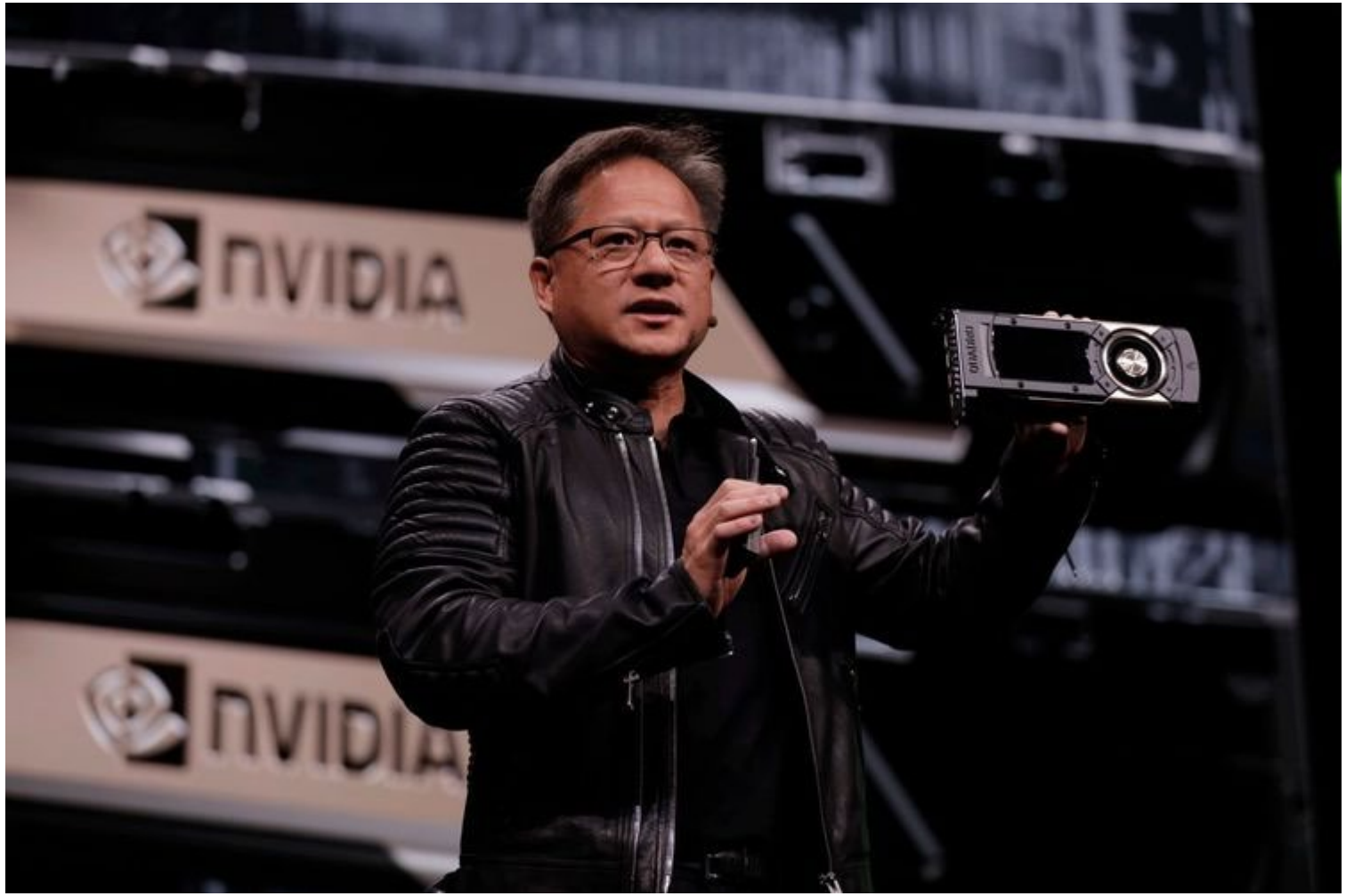
深度学习硬件系统



深度学习硬件系统


Intel i7
0.15 TFLOPS
intel
CORE i7
7th Gen
DDR4
32 GB
TITAN X
Nvidia Titan X
12 TFLOPS
16 GB

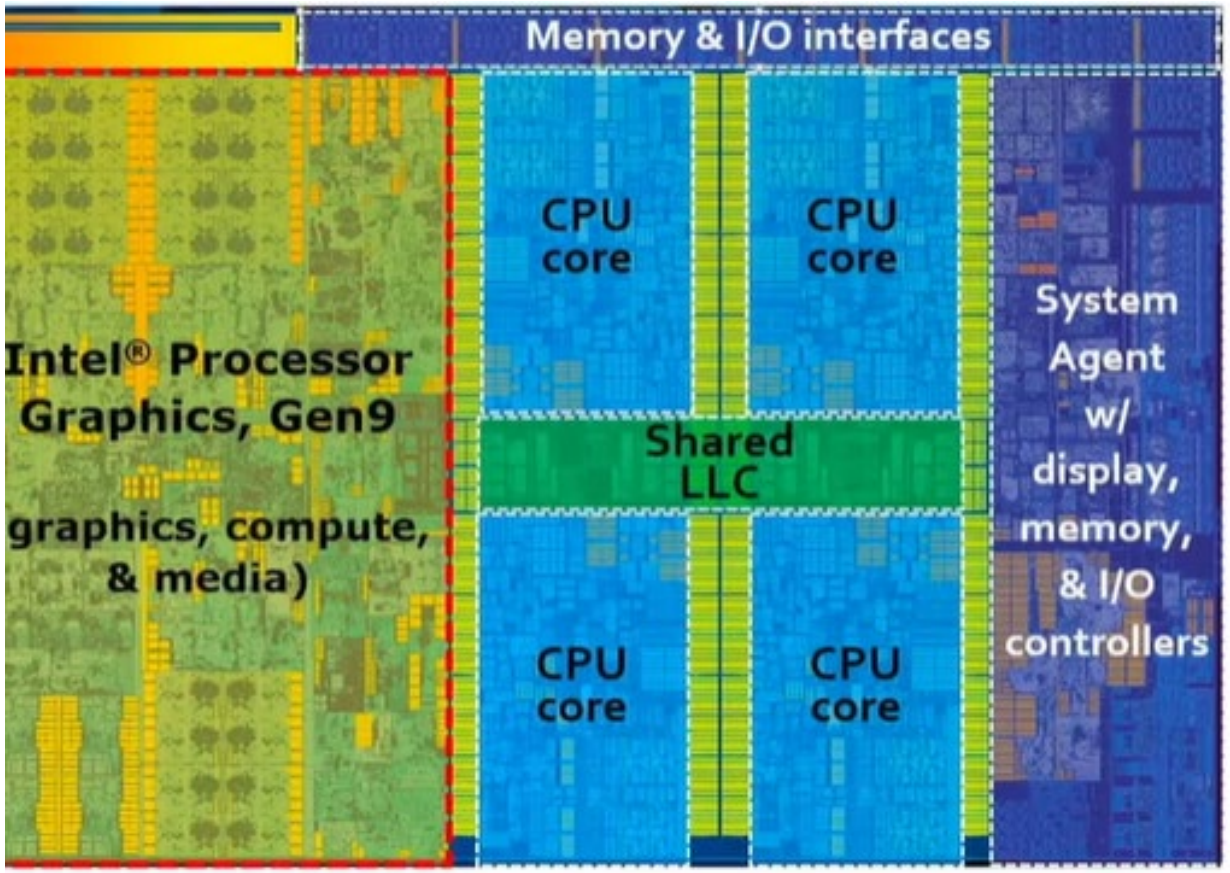
深度学习硬件系统



深度学习硬件系统


分时 五日 日K 周K 月K 季K 年K 120分 60分 30分 15分 5分 1分
区间统计 全屏显示
均线 MA5:654.54 MA10:549.90 MA20:379.09 MA30:330.74 MA60:217.97
前复权
1018.21
844.09
669.97
495.85
321.73
147.61
974.00
雪球

深度学习硬件系统

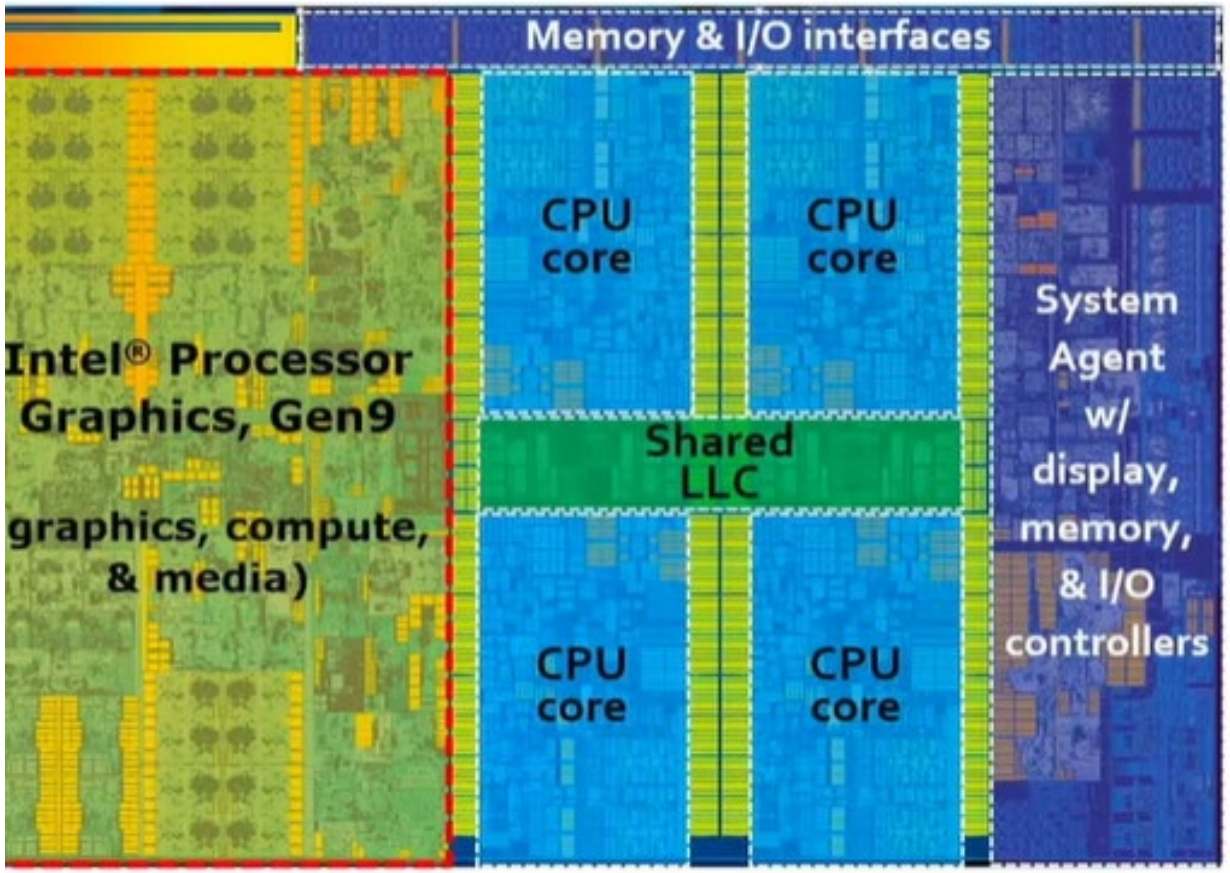


Intel i7-6700K

CPU

- CPU内核
- CPU高速缓存
- 显示处理器（核显）
- 显示、内存、I/O控制器
- 内存、I/O接口

深度学习硬件系统

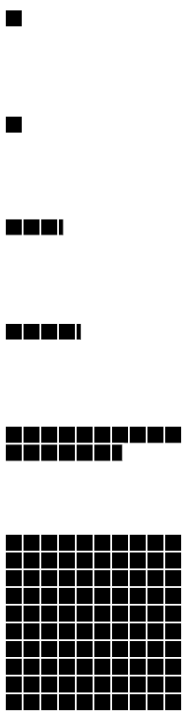


Intel i7-6700K

如何提升CPU利用率?

- 在计算a+b之前，需要准备数据：
  - 主内存→L3→L2→L1→寄存器；
  - L1访问延时：0.5ns；
  - L2访问延时：7ns；
  - 主内存访问延时：100ns；
- 提升空间和时间的内存本地性：
  - 时间：重用数据，保持数据在缓存；
  - 空间：按序读写数据使得可以预读取。

深度学习硬件系统



1ns

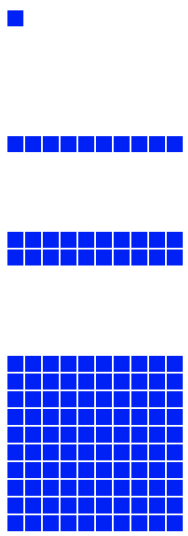
L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = ■

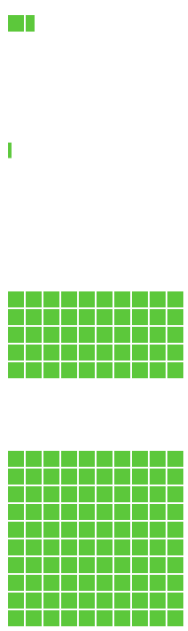


Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs = ■



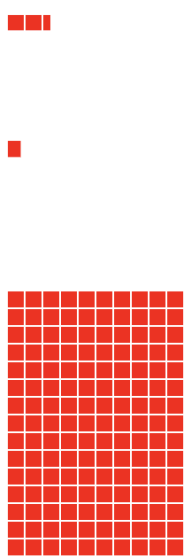
Send 2,000 bytes over commodity network: 44ns

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 3,000ns ≈ 3µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms = ■



Read 1,000,000 bytes sequentially from SSD: 49,000ns ≈ 49µs

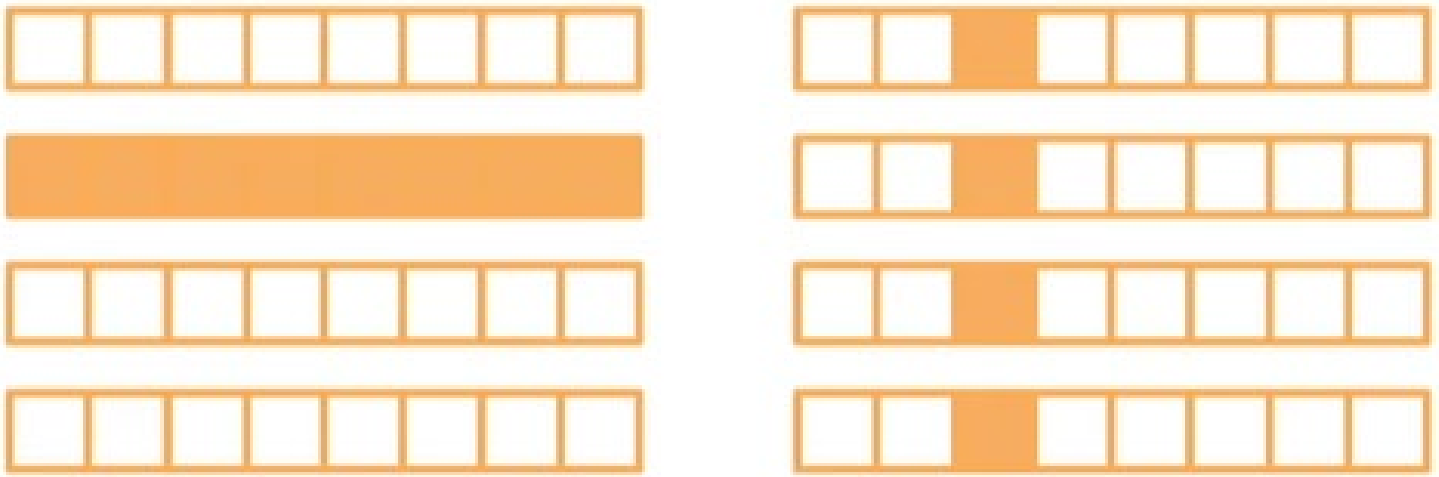
Disk seek: 2,000,000ns ≈ 2ms

Read 1,000,000 bytes sequentially from disk: 825,000ns ≈ 825µs

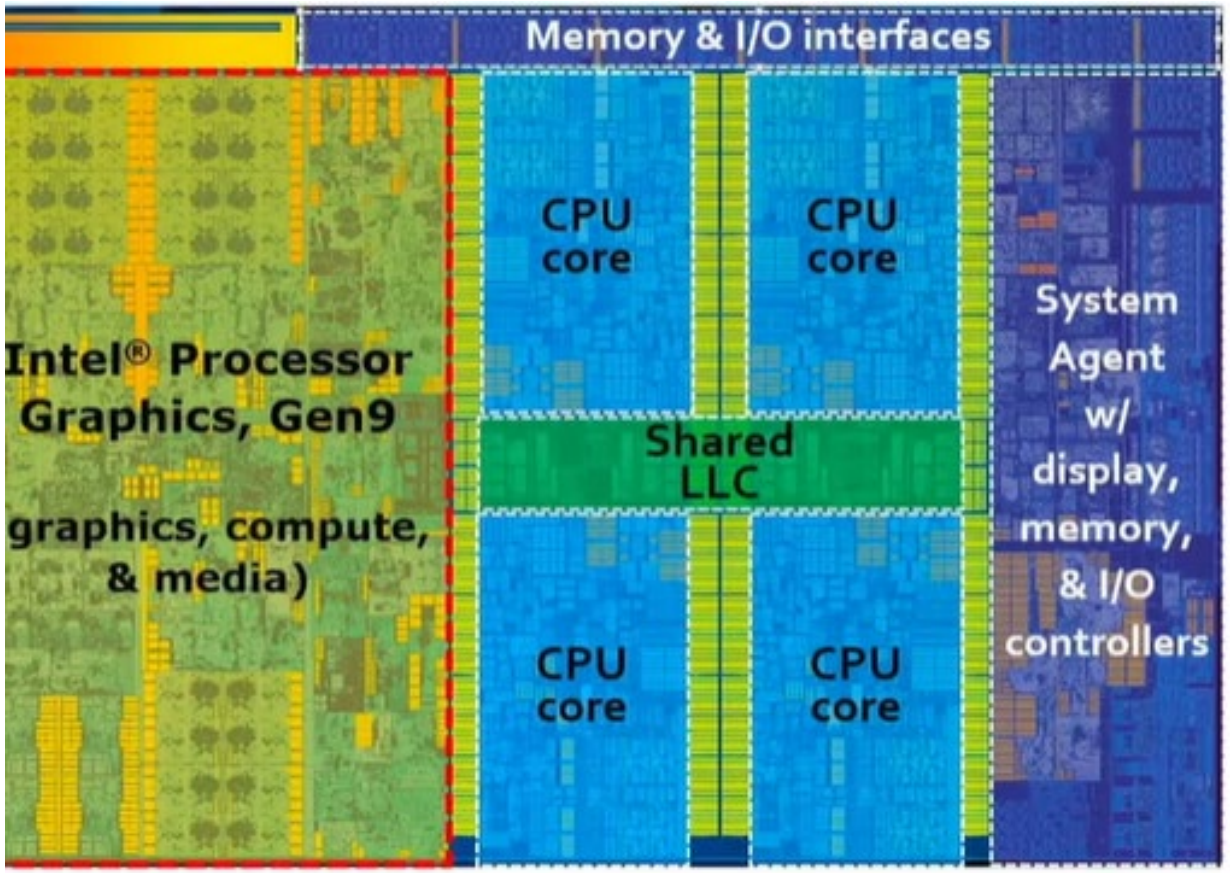
Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

深度学习硬件系统

- 假设矩阵按行存储，那么访问一行比访问一列更快；
  - CPU一次读取64字节（Cache Line）
  - CPU会“聪明地”提前读取下一个（Cache Line）
  - 如果需要按列取矩阵元素，需要进行几次读取？



深度学习硬件系统



Intel i7-6700K

如何提升CPU利用率?

- 另一种策略——并行：
  - 高端CPU通常有几十个核心；
  - 并行利用所有核心；
  - 超线程不一定能提升计算性能，因为它们会共享寄存器，在深度学习计算中，关闭超线程可能还会略微提升性能。

深度学习硬件系统

- 左边比右边慢（Python）：

```
for i in range(len(a)):
    c[i] = a[i] + b[i]
```

```
c = a + b
```

  - 左边调用n次函数，每次调用都有开销；
  - 右边容易并行（类似下面的C++实现）：

```
#pragma omp for
for (i=0; i<a.size(); i++) {
    c[i] = a[i] + b[i]
}
```

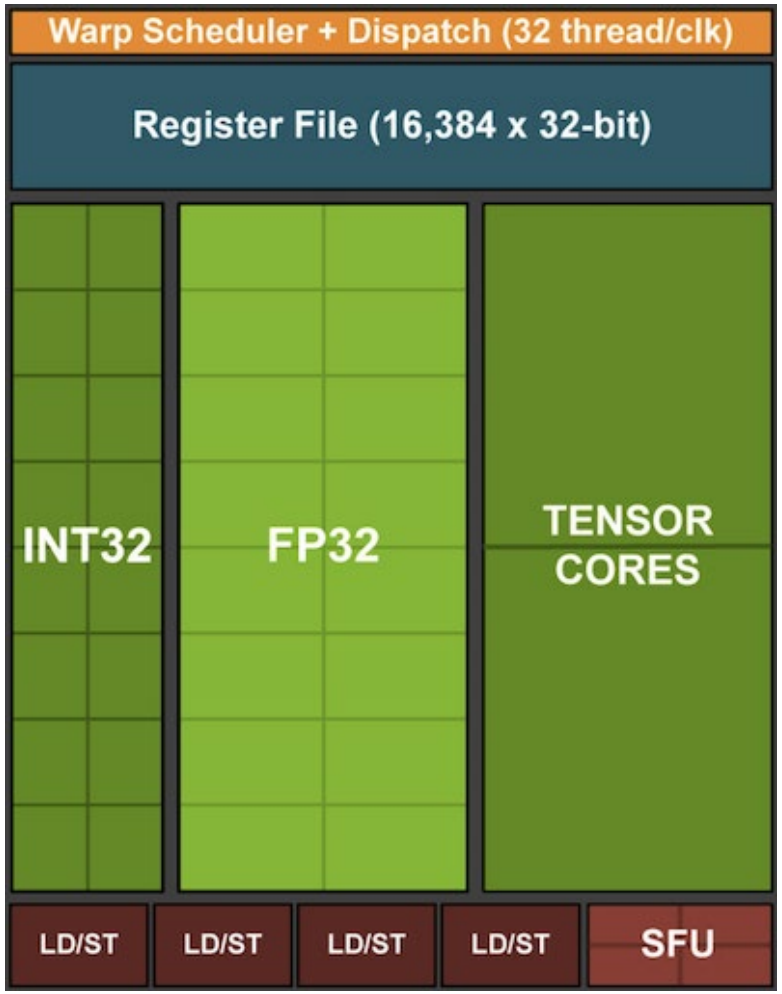
深度学习硬件系统

NVIDIA Titan X (Pascal)

GPU

- 流式多处理器 (Stream Multi-processor, SM)；
- 每个流式处理单元只能执行简单计算指令（例如浮点运算），但是数量多，并行能力比CPU强。

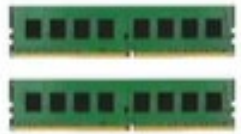
深度学习硬件系统



GPU

- 新架构的NVIDIA GPU专门设计了张量核（Tensor Core）。

深度学习硬件系统



# 核	6 / 64	2K / 4K
TFLOPS	0.2 / 1	10 / 100
内存大小	32 GB / 1 TB	16 GB / 32 GB
内存带宽	30 GB/s / 100 GB/s	400 GB/s / 1 TB/s
控制流	强	弱

CPU——通用，GPU——特化

深度学习硬件系统

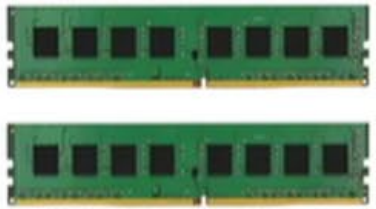
NVIDIA Titan X (Pascal)

提高GPU利用率

- 并行：
  使用数千个计算线程（在较大的模型才能很好地体现GPU并行优势）；
- 内存本地性：
  缓存小，架构更加简单；
- 少用控制语句：
  支持有限，同步开销大。

深度学习硬件系统

30 GB/s



PCIe 3.0 16x: 16 GB/s

480 GB/s

- 不要频繁在CPU和GPU之前传数据：带宽限制，同步开销

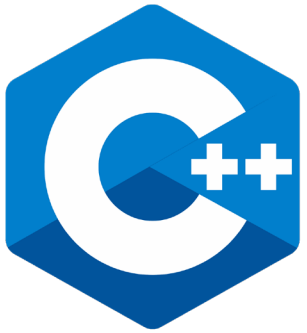
深度学习硬件系统

更多CPU、GPU

- CPU：AMD，ARM
- GPU：AMD，Intel，ARM，Qualcomm...

CPU/GPU高性能计算编程

- **CPU：**
  - C++或其他任何高性能语言
  - 编译器成熟

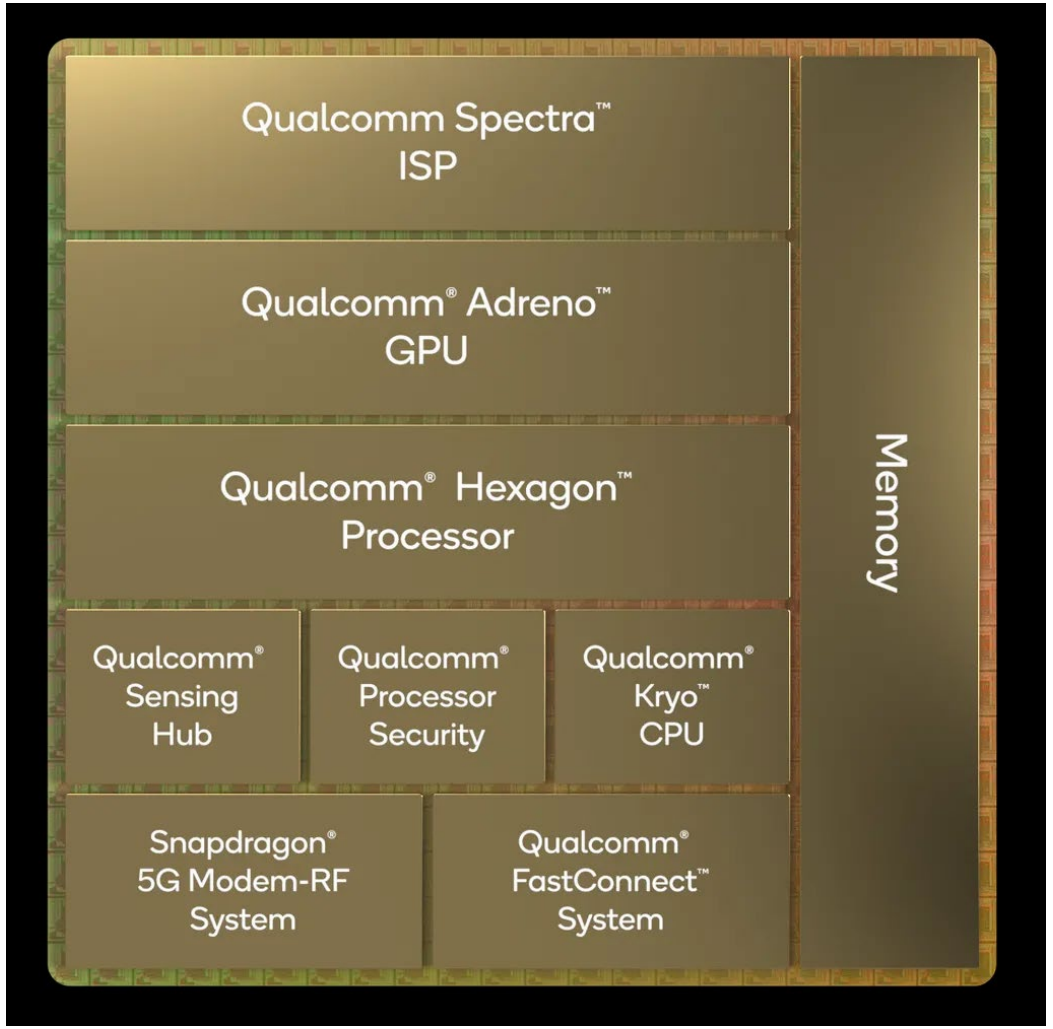


- **GPU：**
  - NVIDIA上用CUDA，编译器和驱动成熟
  - 其他用OpenCL，质量取决于硬件厂商

深度学习硬件系统

DSP：数字信号处理

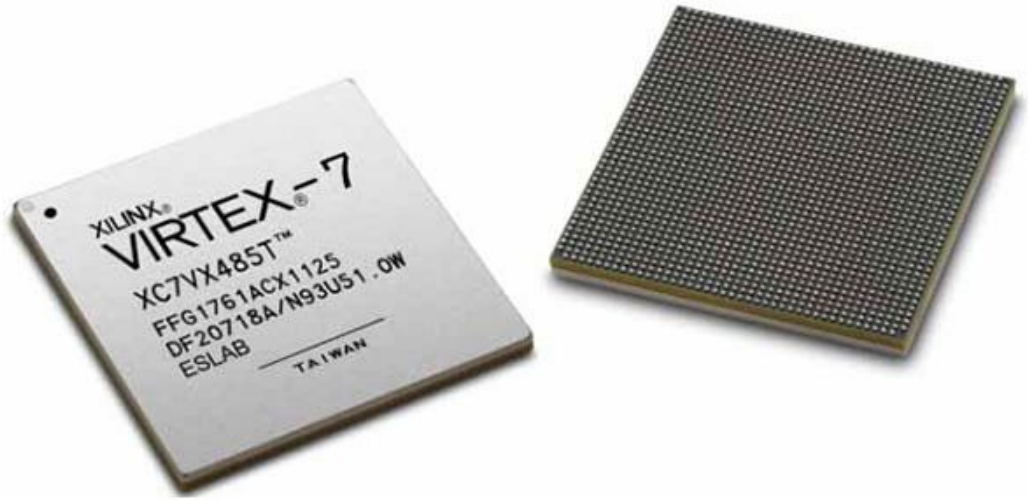
- 为数字信号处理算法设计（滤波、降噪）；
- 低功耗、高性能；
- VLIW：Very long instruction word，一条指令计算上百次累加；
- 编程和调试困难；
- 编译器质量良莠不齐。



深度学习硬件系统

FPGA：可编程阵列

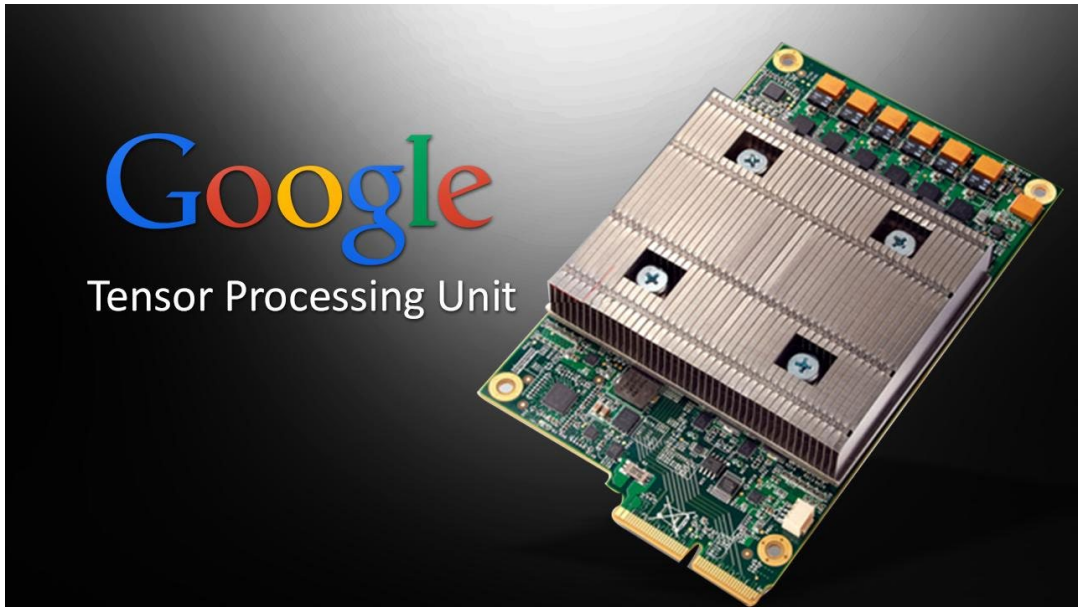
- 有大量可以编程的逻辑单元和可配置连接；
- 可以配置成计算复杂函数：
- 编程语言：VHDL、Verilog；
- 通常比CPU效率更高；
- 工具链质量良莠不齐；
- 一次“编译”需要数小时。



深度学习硬件系统

ASIC：专用集成电路

- 深度学习热门领域：
  - 大公司都在设计制造自己的芯片（Intel、Qualcomm、Google、Amazon、Facebook……）；
- Google TPU是标志性芯片：
  - 能够媲美NVIDIA GPU性能；
  - 在Google大量部署；
  - 核心是systolic array；
- 华为昇腾、寒武纪。



深度学习硬件系统


Ascend 910
Meet the world's most powerful AI proc
算力最强的AI处理器 — 昇腾910正式推

深度学习硬件系统


征程® 6 旗舰
Journey
560 TOPS*
专为新一代城区高阶智能驾驶而生
的车载计算方案
城区高阶智驾规模化落地和体验升级，面临复杂嵌入式超级计算机系统技术和极致工程能力的技术挑战，更需要性能高效且资源均衡的计算方案。征程6专为新一代城区高阶智能驾驶推出性能旗舰版本。高集成度的征程6旗舰，算力560TOPS，对BEV Transformer、大规模交互式博弈等先进算法的支持效率领跑业界。征程6旗舰同时提供更均衡的计算资源分配，单芯片可处理感知、决策、规控全流程数据，同时为客户提供灵活的算法优化空间。征程6旗舰的推出将赋能车企全面落地城区高阶智能驾驶，打造差异化用户体验，助力品牌向上。
Horizon Robotics
Journey™ 6

深度学习硬件系统


灵活性，易用性
intel
CORE i7
7th Gen
Hexagon 685 DSP
xilinx
FPGA
性能和功耗

并行训练和推理

单机多卡并行

- 一台机器可以安装多个GPU（1-16）；
- 在训练和推理时，将一个小批量的计算切分到多个GPU，来达到加速目的；
- 常用的切分方案：
  - 数据并行；
  - 模型并行；
  - 通道并行（数据+模型）。

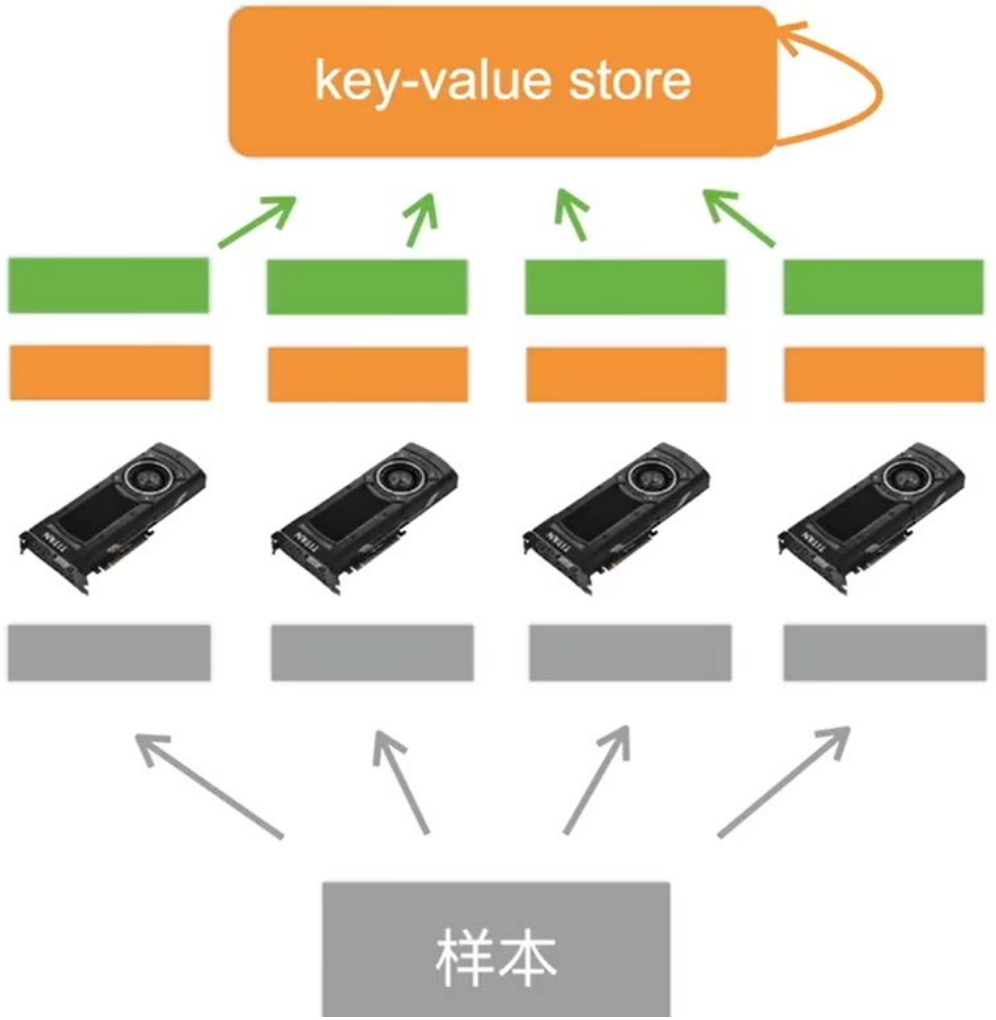
并行训练和推理

数据并行 VS 模型并行

- 数据并行：
  将小批量分成n块，每个GPU拿到完整的参数计算1块数据的梯度；
  - 通常性能更好；
- 模型并行：
  将模型分成n块，每个GPU拿到1块模型计算它的前向和反向结果；
  - 通常用于规模很大的模型。

并行训练和推理

数据并行



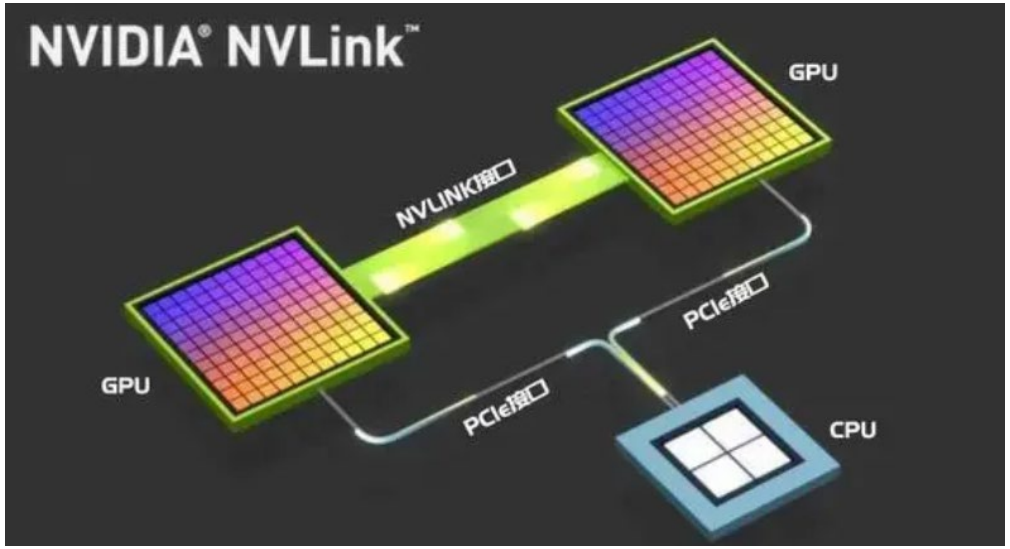
1. 读一个数据块
2. 拿回参数
3. 计算梯度
4. 发出梯度
5. 更新梯度

并行训练和推理

数据并行——PCIe V.S. NVLink

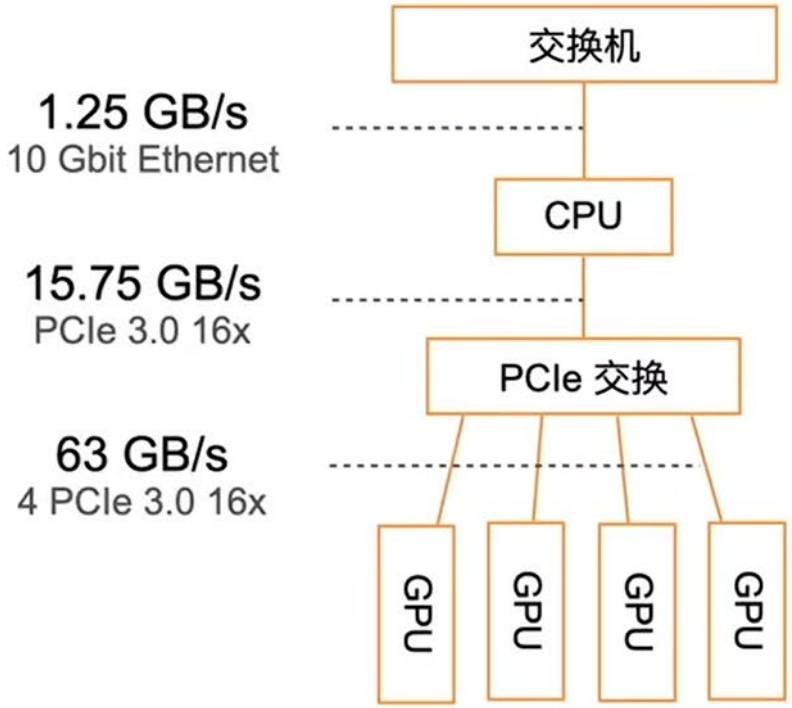
- PCIe：一种基于串行通信的总线标准，采用点对点连接，支持多个设备同时进行高速数据传输；
  - PCIe 4.0的带宽约为32GB/s；

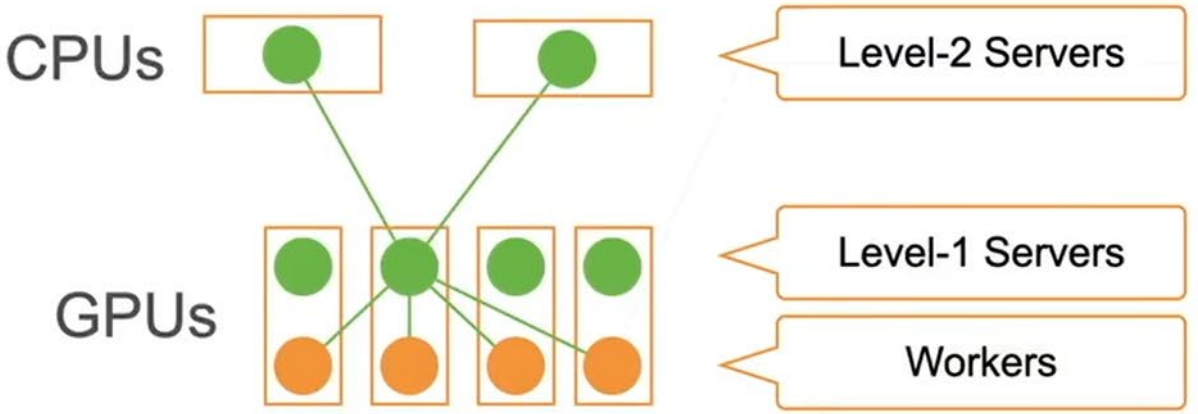
- NVLink：NVidia开发的一种高速连接技术，主要用于连接GPU和其他组件，采用并行通信，较PCIe能提供更高的带宽和更低的延迟；
  - NVLink 2.0的带宽高达300GB/s。



并行训练和推理

多机多卡并行

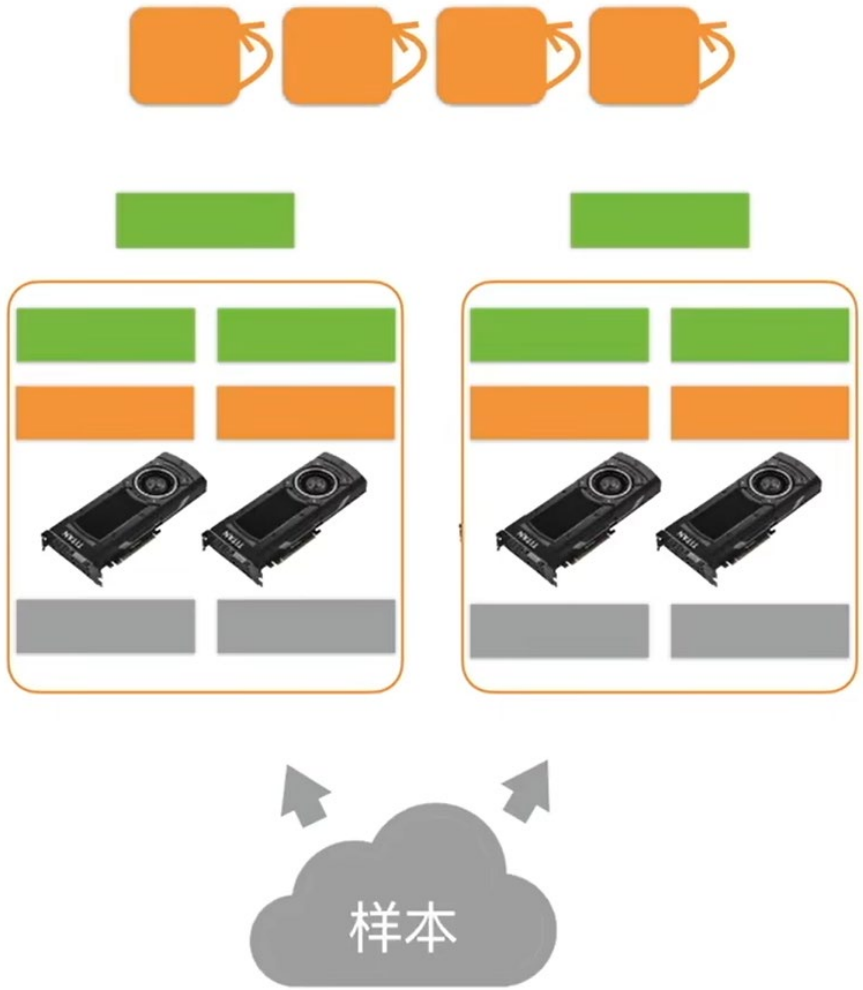




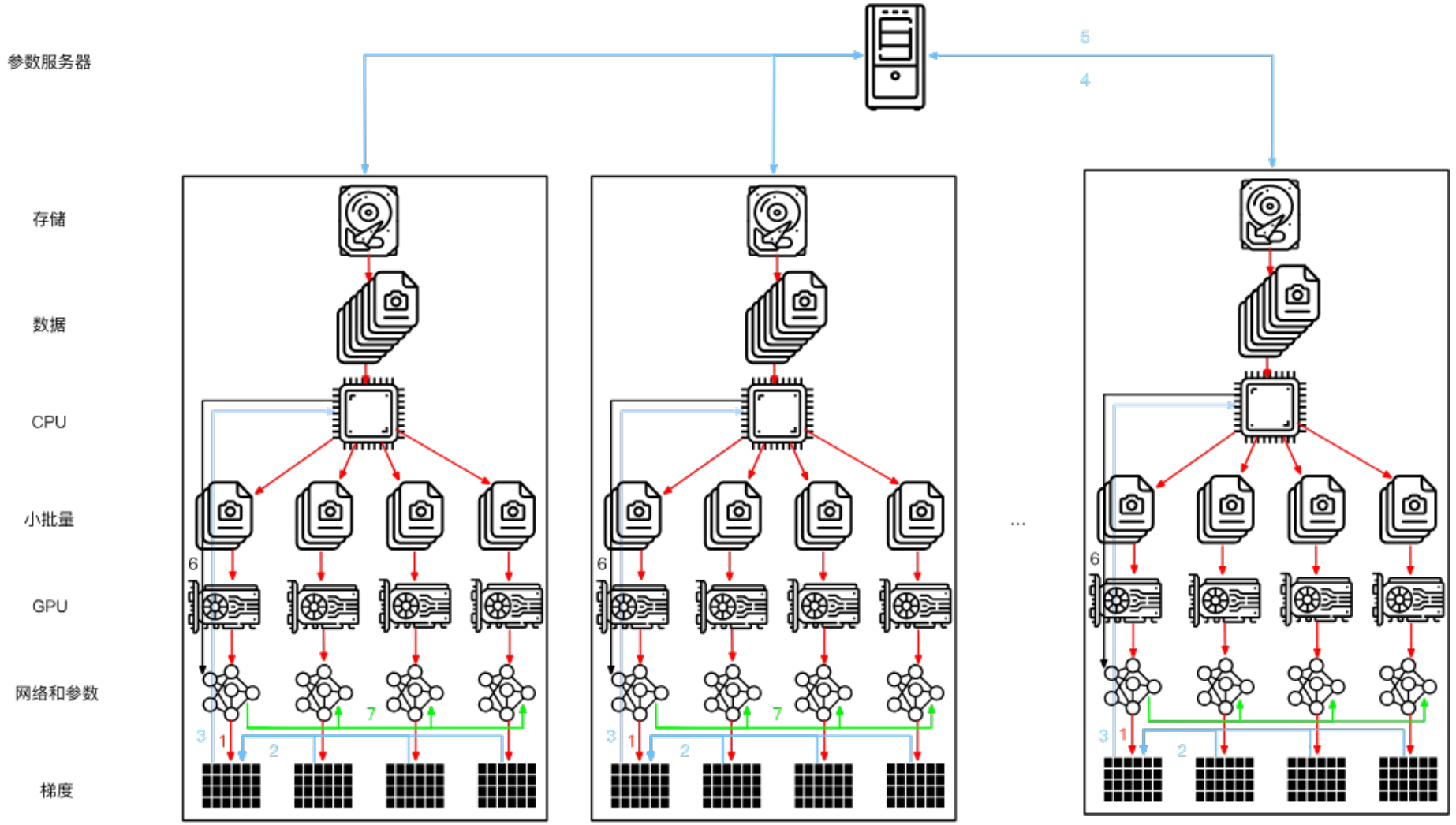
并行训练和推理

多机多卡并行

- 每台计算服务器读取数据的一块；
- 进一步切分数据到每块GPU；
- 每个worker从参数服务器获取模型参数；
- 复制参数到每块GPU；
- 每块GPU计算梯度；
- 将所有GPU上的梯度求和；
- 梯度传回参数服务器；
- 每台服务器更新参数。

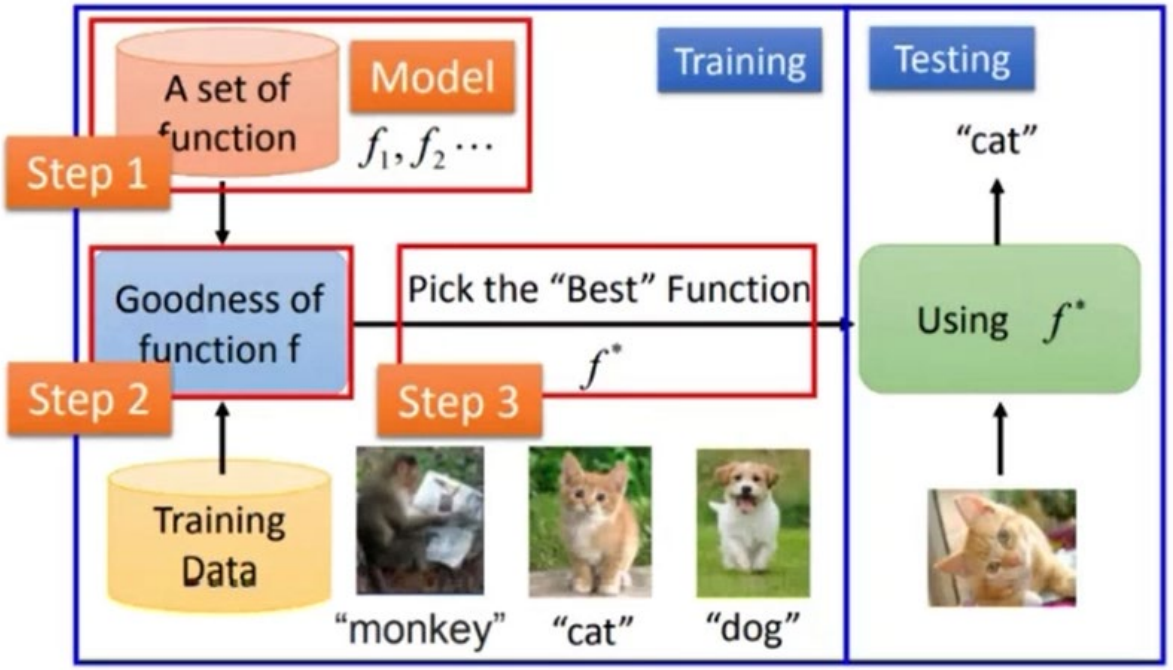


并行训练和推理



深度学习框架

- 软件框架（Software Framework）：为了实现某个业界标准，或完成特定基本任务的软件组件规范；
- 深度学习存在大量重复的基础算法、流程——重复工作。



深度学习框架——发展历程


使用简单
多GPU支持
复杂网络支撑
声明式
Caffe
Chainer
theano
指令式
API复杂
无GPU支持
手动实现网络
torch
opennn
Open Neural Networks Library
自动微分
计算图优化
生态友好
分布式支持
效率优化、可扩展
mxnet
Caffe2
TensorFlow
CNTK
PYTORCH
Keras
并行计算
异构转换
编译层优化
多场景、多任务支持
丰富的套件支持
TensorFlow
2.0 is coming
昇思
MindSpore
飞桨
PyTorch
面向人工智能模型发展趋势
算子优化、统一标准、端云一体
大模型、大任务、全场景
隐私与公平性
2000年~ 萌芽阶段
2012年~ 成长阶段
2015~2018年 稳定阶段
2019~2020年 深化阶段
~?

深度学习框架——发展历程

“石器时代”

~2012年：API不友好，没有GPU支持，尚不完善

MATLAB是由美国MathWorks公司出品的一种用于算法开发、数据分析以及数值计算的高级技术计算语言和交互式环境。

OpenNN 开发于 2003 年在国际工程数值方法中心的名为RAMFLOOD的项目中是一个使用C++编写的开源类库。

Torch是Facebook的开源机器学习库、科学计算框架和基于Lua编程语言的脚本语言，是PyTorch的直系祖先。

```
function [W1, W5, Wo] = MnistConv(W1, W5, Wo, X, D)
%
alpha = 0.01;
beta  = 0.95;
momentum1 = zeros(size(W1));
momentum5 = zeros(size(W5));
momentumo = zeros(size(Wo));
N = length(D);
bsize = 100;
blist = 1:bsize:(N-bsize+1);
% One epoch loop
for batch = 1:length(blist)
  dW1 = zeros(size(W1));
  dW5 = zeros(size(W5));
  dWo = zeros(size(Wo));
  % Mini-batch loop
  %
  begin = blist(batch);
  for k = begin:begin+bsize-
    % Forward pass = inference
    %
    x  = X(:, :, k);              % Input,           28x28
    y1 = Conv(x, W1);             % Convolution,  20x20x20
    y2 = ReLU(y1);
    y3 = Pool(y2);                % Pool,         10x10x20
    y4 = reshape(y3, [], 1);      %                  2000
    v5 = W5*y4;                   % ReLU,             360
    y5 = ReLU(v5);                %
    v  = Wo*y5;                   % Softmax,           10
```

```
    d(sub2ind(size(d), D(k), 1)) = 1;
    % Backpropagation
    %
    e = d - y;                    % Output layer
    delta  = e;
    e5 = Wo' * delta;             % Hidden(ReLU) layer
    delta5 = (y5 > 0) .* e5;
    e4 = W5' * delta5;            % Pooling layer
    e3 = reshape(e4, size(y3));
    e2 = zeros(size(y2));
    W3 = ones(size(y2)) / (2*2);
    for c = 1:20
      e2(:, :, c) = kron(e3(:, :, c), ones([2 2])) .* W3(:, :, c);
    end
    delta2 = (y2 > 0) .* e2;          % ReLU layer
    delta1_x = zeros(size(W1));       % Convolutional layer
    for c = 1:20
      delta1_x(:, :, c) = conv2(x(:, :), rot90(delta2(:, :, c), 2), 'valid');
    end
    dW1 = dW1 + delta1_x;
    dW5 = dW5 + delta5*y4'
    dWo = dWo + delta *y5';
  end
  % Update weights
  %
  dW1 = dW1 / bsize;
  dW5 = dW5 / bsize;
  dWo = dWo / bsize;
  momentum1 = alpha*dW1 + beta*momentum1;
  W1        = W1 + momentum1;
  momentum5 = alpha*dW5 + beta*momentum5;
  W5        = W5 + momentum5;
  momentumo = alpha*dWo + beta*momentumo;
  Wo        = Wo + momentumo;
end
```

```lua
require 'nn';
net = nn.Sequential()
net:add(nn.SpatialConvolution(1, 6, 5, 5)) - 1个图像输入channel, 6个输出channels, 5x5卷积核
net:add(nn.SpatialMaxPooling(2,2,2,2))     -- 1个2x2窗口的最大池化操作
net:add(nn.SpatialConvolution(6, 16, 5, 5))
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.View(16*5*5))                   -- 将16x5x5的3D张量变为1D的16*5*5
net:add(nn.Linear(16*5*5, 120))            -- 全连接层(输入和权重之间的矩阵乘法)
net:add(nn.Linear(120, 84))
net:add(nn.Linear(84, 10))                 -- 10是网络输出的数量net:add(nn.LogSoftMax())

print(net:__tostring());
mlp=nn.Parallel(2,1);     -- iterate over dimension 2 of input
mlp:add(nn.Linear(10,3)); -- apply to first slice
mlp:add(nn.Linear(10,2))  -- apply to first second slice
x = torch.randn(10,2)
print(x)
print(mlp:forward(x))
criterion = nn.ClassNLLCriterion() -- 多分类的负对数似然准则
loss = criterion:forward(output, 3) --
gradients = criterion:backward(output, 3)
gradInput = net:backward(input, gradients)
parameters, gradParameters = net:getParameters()
```

深度学习框架——发展历程

“青铜时代”

2013~2015年：CPU多核、GPU支持、不同编程风格

Caffe

Caffe是以C++/CUDA代码为主的深度学习框架，需要进行编译安装，支持命令行、Python和MATLAB接口，单机多卡、多机多卡等都可以很方便的使用

theano

Theano是一个Python库和优化编译器的开源项目，用于操作和评估数学表达式，尤其是矩阵值表达式。其计算是使用NumPy语法表示，并且编译后可在CPU/GPU架构上高效运行

Chainer是一个开源的深度学习框架，完全在 NumPy 和 CuPy Python 库的基础上用 Python 编写,因其采用边运行边定义 方案以及在大型系统上的性能而著称

```
layer {
  name: "mnist"
  type: "Data"
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
  }
  top: "data"
  top: "label"
}
layer {
  name: "pool1"
  type: "Pooling"
  pooling_param {
    kernel_size: 2
    stride: 2
    pool: MAX
  }
  bottom: "conv1"
  top: "pool1"
}
```

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
```

深度学习框架——发展历程

“铁器时代”

2015~2018年：效率优化、分布式支持、蓬勃发展

深度学习框架——发展历程

“罗马时代”

2018~2019年：大规模训练、并行计算、可用性提升

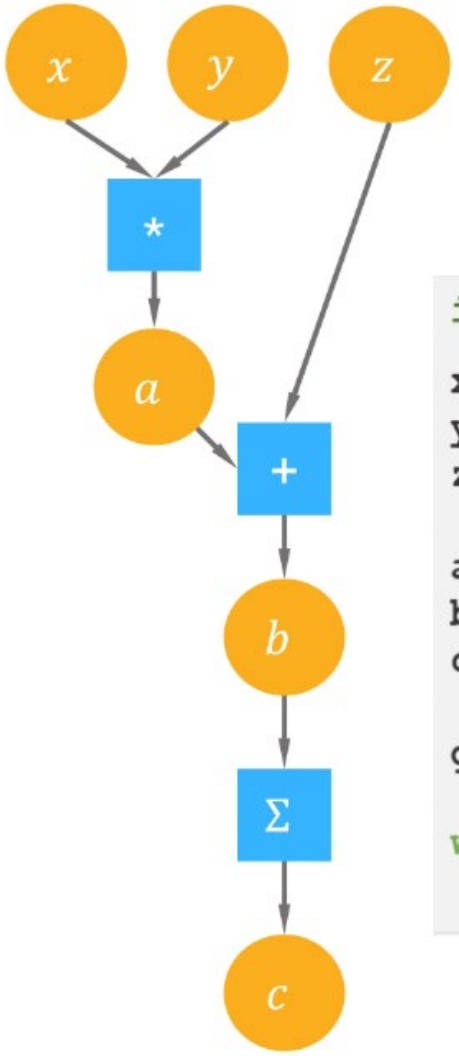
深度学习框架——发展历程

“工业时代”

2020年~：多场景多任务支持、编译层优化、丰富的套件支持

- 基于编译器的算子优化
- 编程接口优化
- 自动分布式训练
- 全场景、异构设备支持
- 动态图、静态图融合
- 训练推理一体化

深度学习框架



TensorFlow Program

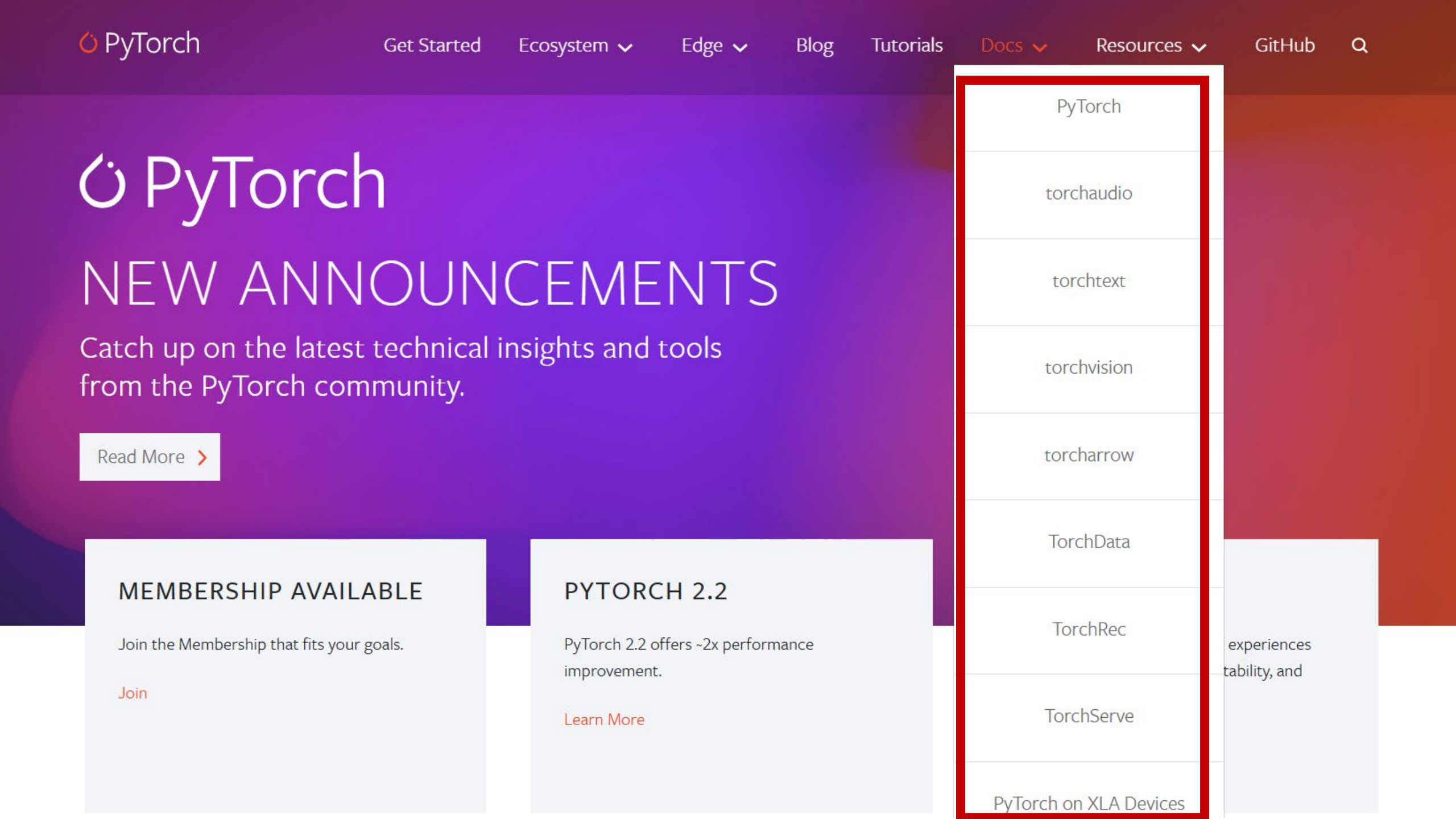
```
import tensorflow as tf

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x,grad_y,grad_z = tf.gradients(c, [x,y,z])

with tf.Session() as sess:
  sess.run([grad_z], feed_dict=values)
```


PyTorch
Get Started
Ecosystem
Edge
Blog
Tutorials
Docs
Resources
GitHub
PyTorch
torchaudio
torchtext
torchvision
torcharrow
TorchData
TorchRec
TorchServe
PyTorch on XLA Devices
PyTorch
NEW ANNOUNCEMENTS
Catch up on the latest technical insights and tools from the PyTorch community.
Read More
MEMBERSHIP AVAILABLE
Join the Membership that fits your goals.
Join
PYTORCH 2.2
PyTorch 2.2 offers ~2x performance improvement.
Learn More
experiences
tability, and

深度学习框架——框架开发

README | Code of conduct | Apache-2.0 license | Security

python 3.9 | 3.10 | 3.11 | 3.12 | pypi package 2.16.1 | DOI 10.5281/zenodo.4724125 | openssf best practices passing
openssf scorecard 8.2 | oss-fuzz coverage failing | oss-fuzz build failing | OSSRank #9 (Top 1%) | Contributor Covenant v1.4 adopted
TF Official Continuous 8 passed, 0 failed | TF Official Nightly 13 passed, 3 failed

Documentation

api reference

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

TensorFlow was originally developed by researchers and engineers working within the Machine Intelligence team at

Languages

C++ 56.6% Python 26.6%
MLIR 5.9% Starlark 4.1%
HTML 2.7% Go 1.3% Other 2.8%

深度学习框架——框架开发

README | Code of conduct | License | Security

PyTorch is a Python package that provides two high-level features:

- Tensor computation (like NumPy) with strong GPU acceleration
- Deep neural networks built on a tape-based autograd system

You can reuse your favorite Python packages such as NumPy, SciPy, and Cython to extend PyTorch when needed.

Our trunk health (Continuous Integration signals) can be found at hud.pytorch.org.

- More About PyTorch
  - A GPU-Ready Tensor Library
  - Dynamic Neural Networks: Tape-Based Autograd
  - Python First
  - Imperative Experiences

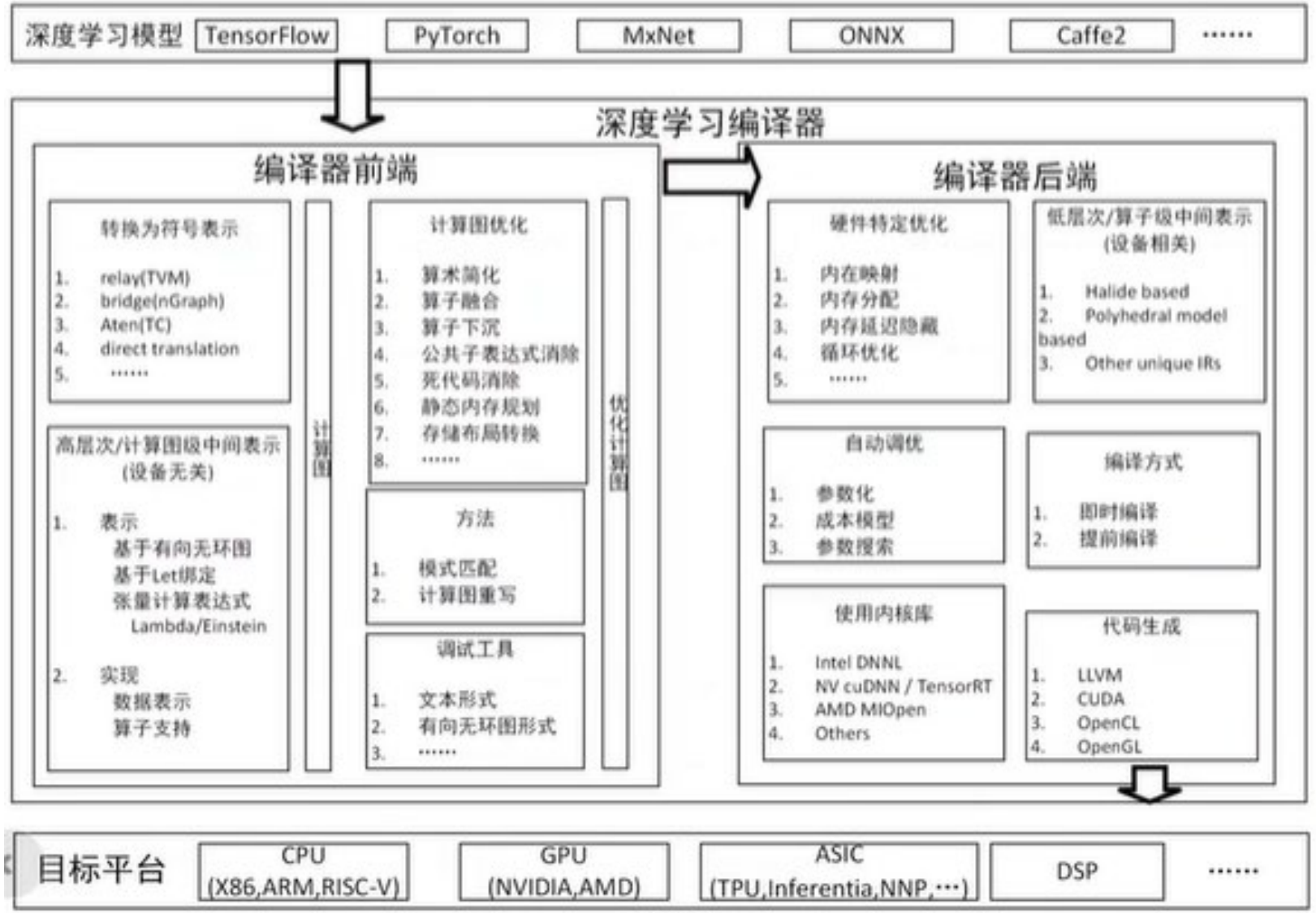
Languages

- Python 50.0%
- C++ 40.6%
- Cuda 3.7%
- C 2.0%
- Objective-C++ 1.3%
- CMake 0.8%
- Other 1.6%

深度学习框架——框架开发

- 现代C++
- CUDA
- C++代码封装（提供Python端接口）
- CMake——自动构建系统

深度学习编译系统


深度学习模型
TensorFlow
PyTorch
MxNet
ONNX
Caffe2
……
深度学习编译器
编译器前端
转换为符号表示
1. relay(TVM)
2. bridge(nGraph)
3. Aten(TC)
4. direct translation
5. ……
高层次/计算图级中间表示
(设备无关)
1. 表示
基于有向无环图
基于Let绑定
张量计算表达式
Lambda/Einstein
2. 实现
数据表示
算子支持
计算图
计算图优化
1. 算术简化
2. 算子融合
3. 算子下沉
4. 公共子表达式消除
5. 死代码消除
6. 静态内存规划
7. 存储布局转换
8. ……
方法
1. 模式匹配
2. 计算图重写
调试工具
1. 文本形式
2. 有向无环图形式
3. ……
优化计算图
编译器后端
硬件特定优化
1. 内在映射
2. 内存分配
3. 内存延迟隐藏
4. 循环优化
5. ……
低层次/算子级中间表示
(设备相关)
1. Halide based
2. Polyhedral model based
3. Other unique IRs
自动调优
1. 参数化
2. 成本模型
3. 参数搜索
编译方式
1. 即时编译
2. 提前编译
使用内核库
1. Intel DNNL
2. NV cuDNN / TensorRT
3. AMD MIOpen
4. Others
代码生成
1. LLVM
2. CUDA
3. OpenCL
4. OpenGL
目标平台
CPU
(X86,ARM,RISC-V)
GPU
(NVIDIA,AMD)
ASIC
(TPU,Inferentia,NNP,…)
DSP
……

深度学习部署

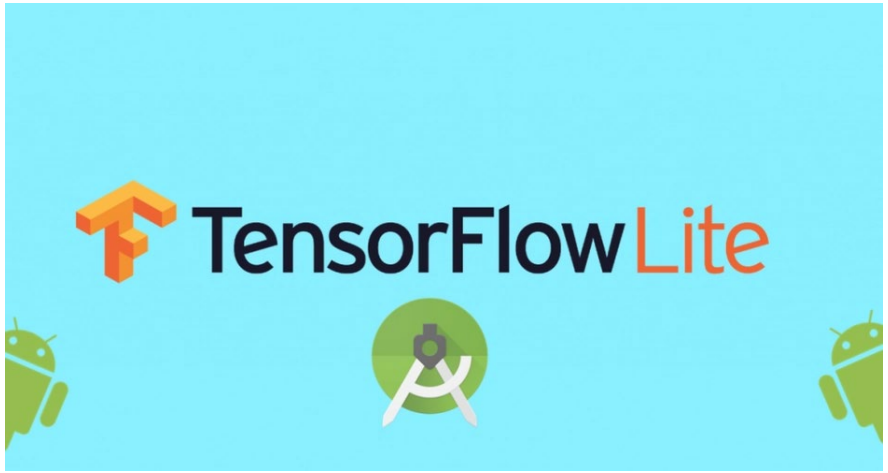
部署方式：

- 训练框架部署：如使用完整的TensorFlow、PyTorch等；
  - 需要安装整个框架；
  - 推理性能差；
  - 很多冗余功能；
  - 内存占用大。

深度学习部署

部署方式：

- 训练框架部署引擎：
  TensorFlow Serving、
  TensorFlow Lite、TorchServe等；
  - 只支持自己框架训练的模型；
  - 支持的硬件、OS有限。



TORCHSERVE

TorchServe is a performant, flexible and easy to use tool for serving PyTorch

What's going on in TorchServe?

- High performance Llama 2 deployments with AWS Inferentia2 using To
- Naver Case Study: Transition From High-Cost GPUs to Intel CPUs and o
- Run multiple generative AI models on GPU using Amazon SageMaker m
  75% in inference costs

深度学习部署

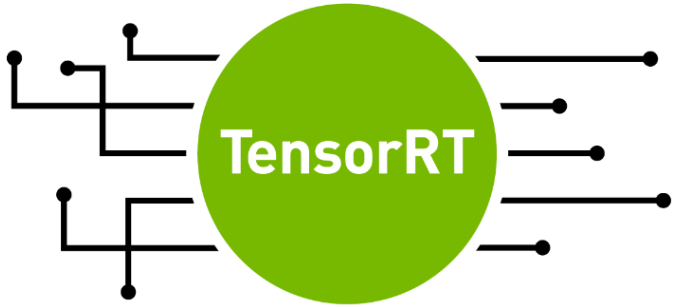
部署方式：

- 手动模型重构：
  手写C/C++代码，实现计算图，导入权重；
  - “从头造轮子”；
  - 需要开发者对模型有充分的了解；
  - 有较高的技术难度。

深度学习部署

部署方式：

- 深度学习推理引擎：
  ONNX、OpenVINO、NVIDIA Tensor RT，移动端NCNN；
  - 支持加载主流框架的模型文件；
  - 性能良好，能够满足不同环境下的应用需求。

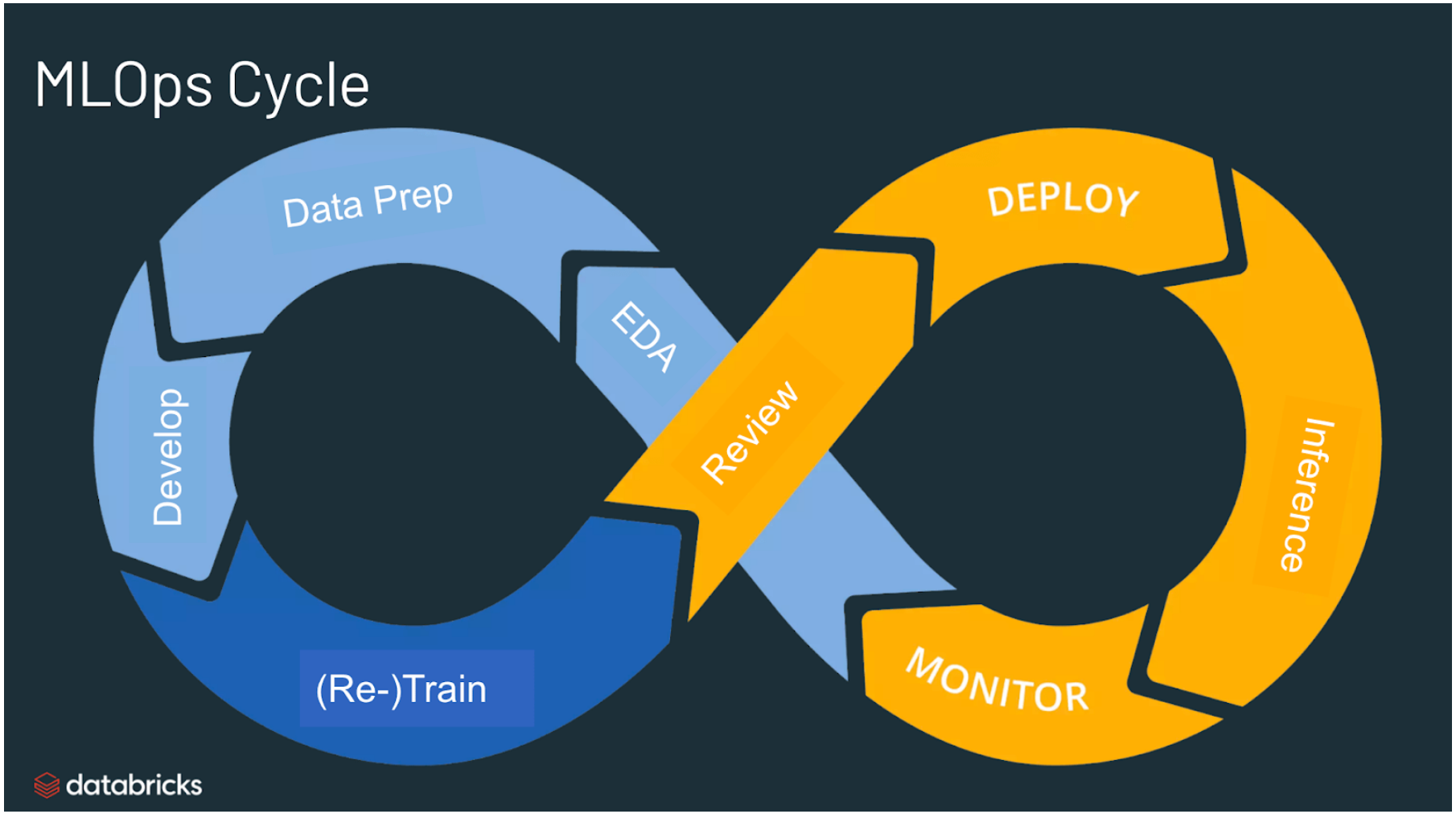


深度学习部署

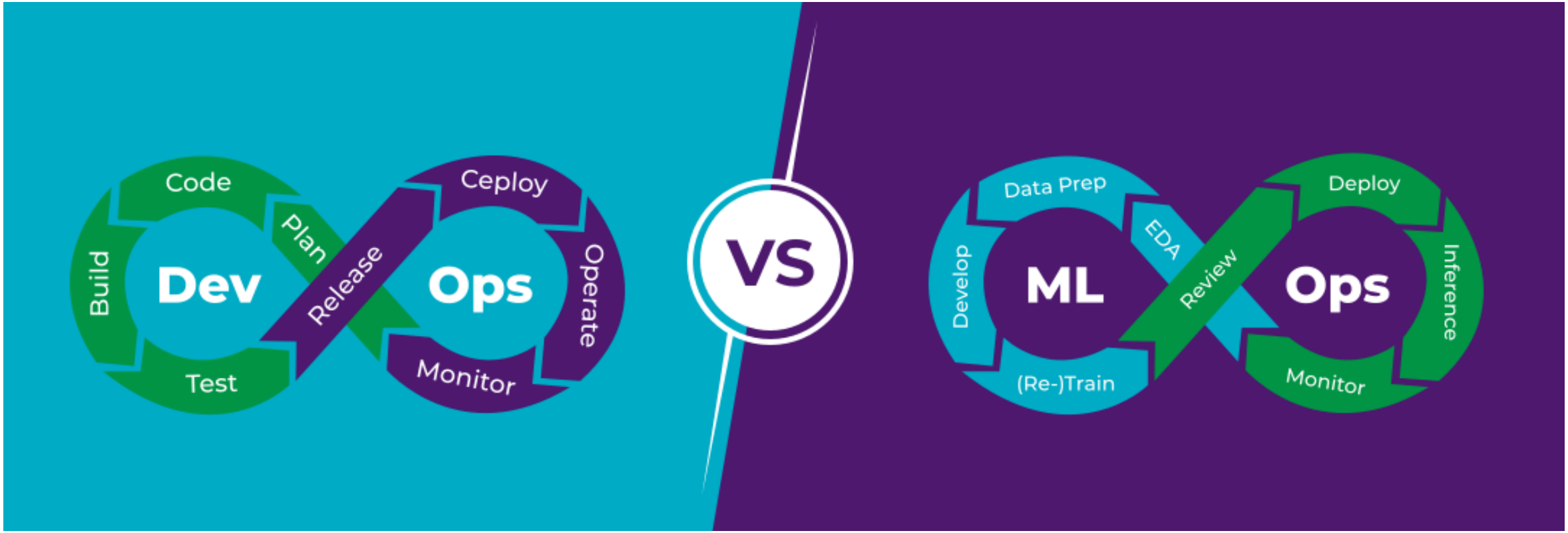
部署方式：

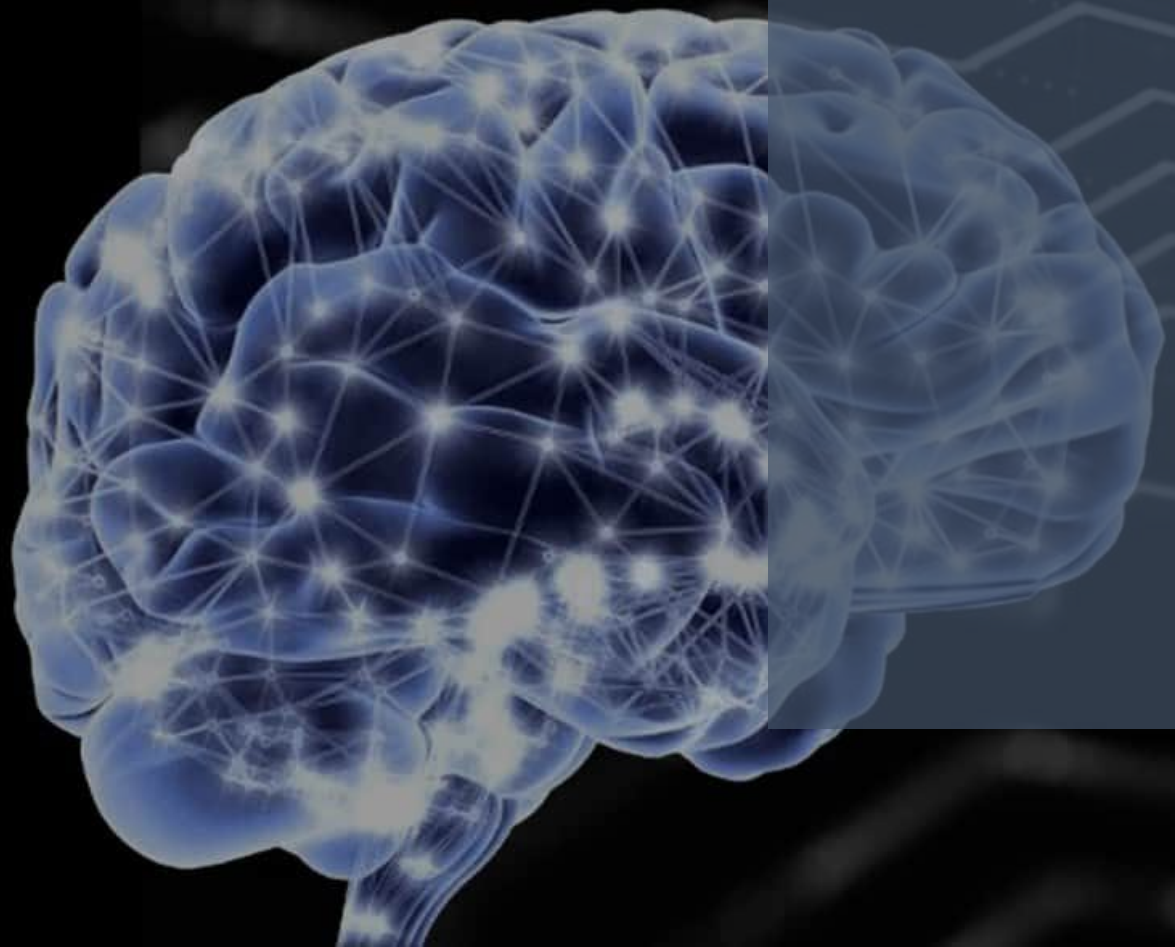
- 训练框架部署引擎：
  TensorFlow Lite、TorchServe等；
  - 比完整的训练框架精简；
  - 只支持自己框架训练的模型；
  - 支持的硬件、OS有限。

MLOps


MLOps Cycle
Data Prep
EDA
Develop
(Re-)Train
Review
DEPLOY
Inference
MONITOR
databricks

MLOps


Code
Plan
Build
Dev
Release
Test
Ceploy
Operate
Ops
Monitor
VS
Data Prep
EDA
Develop
ML
Review
(Re-)Train
Deploy
Inference
Ops
Monitor



谢谢！

文献阅读/专题研讨

- 以小组为单位，进行深度学习领域的**经典文献阅读**或**专题研讨**
- 小组汇报：
  - PPT介绍（8-10分钟）
  - 答辩、讨论（3-5分钟）
- 每组选择1个主题，并在以下文档登记（每个小组选择的主题不能重复，先到先得）：
  https://docs.qq.com/sheet/DVFVzY2NMQXFSS2FI?tab=BB08J2
- 评分要点：
  - 结构清晰、内容详略得当、回答问题准确
  - 注意控制时间（时间不足或超时酌情扣分）

文献阅读/专题研讨

日期	周次	小组数量	小组编号
2024.05.27（周一）	第13周	4个小组	
2024.06.03（周一）	第14周	4个小组	
2024.06.05（周三）	第14周	4个小组	
2024.06.17（周一）	第16周	4个小组	
2024.06.19（周三）	第16周	5个小组	