MindSpore：一种全场景覆盖的深度学习计算框架

摘要

MindSpore 是一种全新的深度学习计算框架，旨在实现易开发、高效执行、全场景覆盖三大目标。为了实现易开发的目标，MindSpore 采用基于源码转换（Source Code Transformation，SCT）的自动微分（Automatic Differentiation，AD）机制，该机制可以用控制流表示复杂的组合。函数被转换成函数中间表达（Intermediate Representation，IR），中间表达构造出一个能够在不同设备上解析和执行的计算图。在执行前，计算图上应用了多种软硬件协同优化技术，以提升端、边、云等不同场景下的性能和效率。MindSpore 支持动态图，更易于检查运行模式。由于采用了基于源码转换的自动微分机制，所以动态图和静态图之间的模式切换非常简单。为了在大型数据集上有效训练大模型，通过高级手动配置策略，MindSpore 可以支持数据并行、模型并行和混合并行训练，具有很强的灵活性。此外，MindSpore 还有“自动并行”能力，它通过在庞大的策略空间中进行高效搜索来找到一种快速的并行策略。

在本文中，我们将介绍 MindSpore 的架构和几个主要的特性，这些特性不同于现有的训练框架。此外，我们还会介绍 MindSpore 在华为昇腾系列芯片上实现的卓越性能。

1 引言

深度学习研究和应用在近几十年得到了爆炸式的发展，在图像识别[1]、语音识别与合成[2]、游戏[3]、语言建模与分析[4]等方面取得了巨大的成功。深度学习框架[5–9]的不断发展使得在大型数据集上训练神经网络模型时，可以方便地使用大量的计算资源。

目前有两种主流的深度学习框架：一种是在执行之前构造一个静态图，定义所有操作和网络结构，典型代表是 TensorFlow[5]，这种方法以牺牲易用性为代价，来提高训练期间的性能；另一种是立即执行的动态图计算，典型代表是 PyTorch[6]。通过比较可以发现，动态图更灵活、更易调试，但会牺牲性能。因此，现有深度学习框架难以同时满足易开发、高效执行的要求。

本文提出了一种新的深度学习框架——MindSpore，该框架旨在实现三个目标：易开发、高效执行和全场景覆盖。MindSpore 由几个主要组件组成：MindExpression（ME）、MindCompiler（MC）、MindData（MD）、MindRE 和 MindArmour（MA）。表 1 总结了 MindSpore 的技术贡献。

表 1 MindSpore 的特点对实现三个目标的贡献

特点 / 目标	MindExpression&MindCompiler				MindRE	MindData			
	SCT AD	动态图	自动并行	图管理器	运行时系统	训练看板	分析器	自动增强	自动数据加速
易开发	√	√	√	-	-	√	-	√	-
高效执行	√	-	√	√	-	-	√	-	√
全场景覆盖	-	-	-	√	√	-	-	-	-

注：√指主要贡献；-指次要贡献。

1. ME 为用户提供了 Python 编程范式，MC 提供基于中间表达的 JIT 编译优化能力，他们具有以下三个突出特点。
   - **自动微分**：采用基于源码转换的自动微分机制，在训练或推理阶段，将一段 Python 代码转换为数据流图。因此，用户可以方便地使用 Python 原生控制逻辑来构建复杂的神经网络模型。
   - **自动并行**（Automatic Parallelization）：由于大规模模型和数据集的不断增加，跨分布式设备并行化深度神经网络（Deep Neural Network，DNN）训练已经成为一种常见做法。然而，当前框架（如 TensorFlow[5]、Caffe[10]和 MXNet[7]）用于并行化训练的策略仍然简单直接，而且通常是次优的。MindSpore 并行化训练任务，透明且高效。“透明”是指用户只需更改一行配置，提交一个版本的 Python 代码，就可以在多个设备上运行这一版本的 Python 代码进行训练。“高效”是指该算法以最小的代价选择并行策略，降低了计算和通信开销。
   - **动态图**：MindSpore 支持动态图，无须引入额外的自动微分机制（如算子重载微分机制），从而大大增加了动态图和静态图的兼容性。
2. MD 负责数据处理，并提供工具来帮助开发者调试和优化模型。通过自动数据加速技术实现了高性能的流水线，以进行数据处理。各种自动增强策略的出现，使用户不必再寻找合适的数据增强策略。训练看板将多种数据集成在一个页面，方便用户查看训练过程。分析器可以打开执行黑匣子，收集执行时间和内存使用的相关数据，从而有针对性地进行性能优化。
3. MA 负责提供工具，以帮助开发者防御对抗性攻击，实现隐私保护的机器学习。在形式方面，MA 提供了以下功能：生成对抗代码、评估模型在特定对抗环境中的性能、开发出更健壮的模型。MA 还支持丰富的隐私保护能力，如差分隐私[11]、机密人工智能计算[12]、可信协同学习[13、14]等。
4. MindRE 负责 AI 网络的执行，抽象和适配各底层硬件的操作接口，触发网络执行。支持端、云多种硬件环境的运行时系统

本文的其余内容如下：第 2 章介绍了 MindSpore 的架构；第 3 章通过阐述自动微分、自动并行和动态图支持的设计细节，介绍了 MindSpore 的核心模块——ME&MC；第 4 章和第 5 章分别介绍了 MD 和 MA 的相关细节；第 6 章介绍了 MindSpore 支持的端云协同架构；为了说明高效执行这一特性，第 7 章阐述了使用 ResNet50 作为基准来评估“自动并行”特性，还介绍了 MindSpore 的训练和推理性能；最后，第 8 章对我们的工作进行了总结，并介绍了未来的重点研究方向。

2 MindSpore 概述

2.1 MindSpore 架构

如图 1 所示，MindSpore 架构图组成。

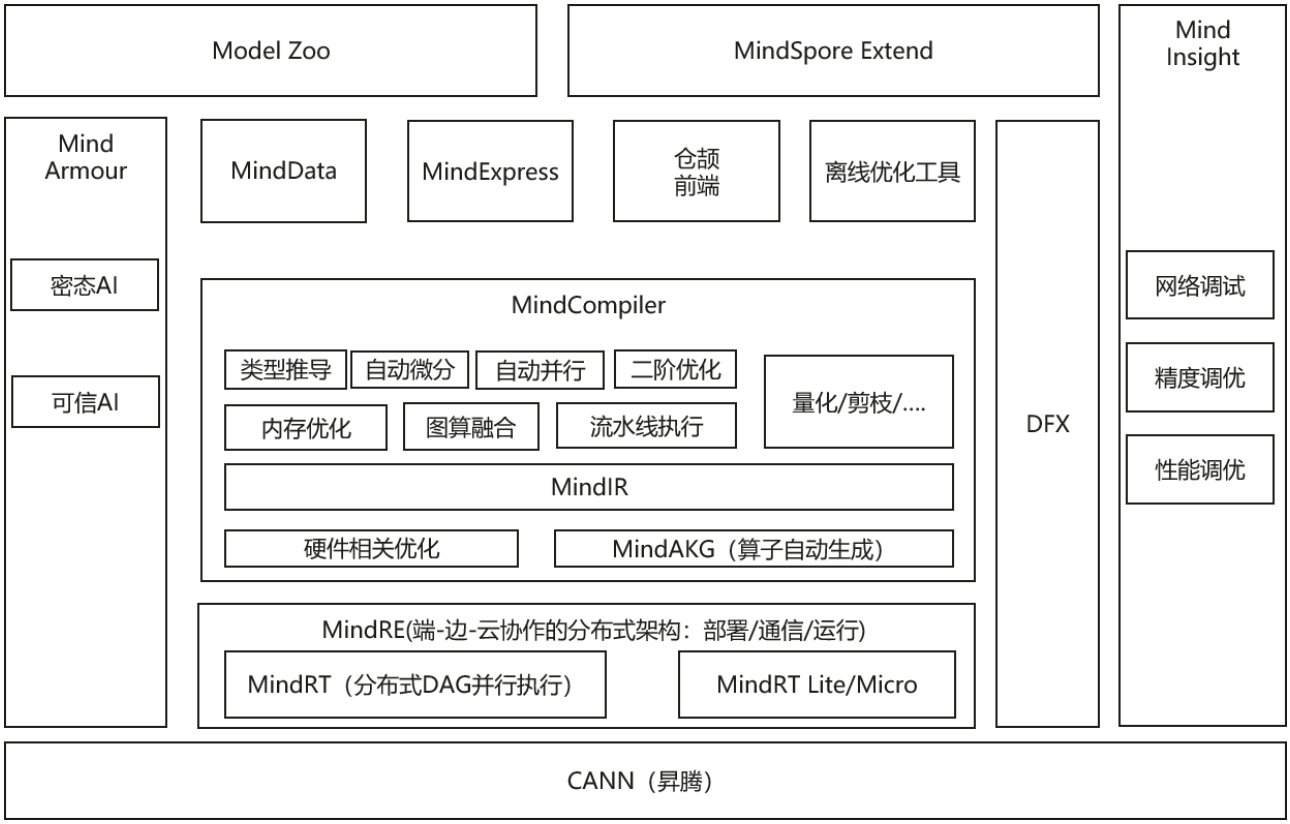


图 1 MindSpore 架构

ME 提供 Python 接口，用于定义用户级应用软件编程接口（Application Programming Interface，API），用于构建和训练神经网络。由于 MindSpore 采用基于源码转换策略的自动微分机制，因此用户可以 Python 方式编程。一种典型的情况是，用户可以用常规的 Python 方式构建复杂的模型和流程控制，如 if、else、while 等。

MC 是高性能执行和自动微分的核心，自动微分基于源码转换方式。如果用户选择以 PyNative 模式运行，则逐个下发算子运行。如果模型以图模式运行，则 MC 会使用流水线根据 Python 代码生成计算图，通过解析 Python 代码，生成抽象语法树（Abstract Syntax Tree，AST），然后将其转换为 A-Normal-Form（ANF）图[15]。ANF 是图形化的，而不是语法化的，因此从算法上操作起来要容易得多[16]。如果用户需要训练神经网络，流水线自动生成反向计算节点，并添加到 ANF 图中。流水线在构造完整图之后进行许多优化（例如内存复用、算子融合、常数消除等）。如果用户需要在分布式环境中训练模型，流水线将应用自动并行提供的优化策略（参见 3.2 节）。后端的虚拟机通过会话管理计算图，调用后端来运行图，并控制图的生命周期。

MD（MindData）中的数据处理在训练过程中完成数据流水线，包括数据加载、数据论证、数据转换等。MD 也提供简单的 API，支持 CV/NLP/GNN 等全场景的数据处理能力。同时，在这个过程中，如何提高数据处理能力以匹配人工智能芯片的算力，是保证人工智能芯片发挥极致性能的关键。MD 中的 MindInsight 有四个模块：训练看板、溯源、分析器和调试器。分析器可以打开执行黑匣子，以收集执行时间和内存使用的相关数据。

调试器是图执行模式下的调试工具，用户可以在训练过程中查看图的内部结构和节点的输入/输出。MindInsight 对训练过程生成的日志文件进行分析。开发者可以轻松地可视化训练过程，在图形用户界面（Graphical User Interface，GUI）上与 MindInsight 对比不同的路径。

MA 帮助用户开发更健壮的模型，保护用户在训练和推理数据中的隐私。MA 中的对抗性攻击防御有三个主要模块：攻击、防御和评估。攻击模块是一个在黑盒和白盒攻击场景下生成对抗样本的工具。防御模块从攻击模块接收对抗样本，并在训练过程中用这些样本来提高模型的鲁棒性。评估模块提供了多种评估指标，允许开发者更容易可视化其模型的鲁棒性。MA 为了实现隐私保护机器学习，实现了一系列差分隐私感知优化器，这些优化器在训练过程中自动将噪声添加到生成的梯度中。

2.2 编程范式

MindSpore 为用户提供 Python 编程范式。借助基于源码转换的自动微分，用户可以使用原生 Python 控制语法和其他一些高级 API，如元组（Tuple）、列表（List）和 Lambda 表达。

为避免用户混淆，MindSpore 引入了尽可能少的接口和概念。在单机平台上训练简单的神经网络时，用户只需要了解以下五个组件：

张量（Tensor）：是一个包含相同数据类型元素的多维矩阵。与其他一些训练框架不同，MindSpore 中没有标量（Scalar）变量（Variable）的概念。要计算一个张量的梯度，该张量的 requires_grad 自动微分属性应该设置为 True。可以用 Numpy 初始化张量，也可以将张量的值转换为一个 Numpy 对象。

数据集（Dataset）：是一个单独的异步流水线，该流水线为在训练中将张量无延迟地馈入网络的剩余部分做准备。

算子（Operator）：是构成神经网络的基本计算单元。MindSpore 支持大多数常用的神经网络算子（如卷积、批标准化、激活等）和数学算子（如加法、乘法等）。此外，MindSpore 支持自定义算子，允许用户添加新的算子来适配特定的硬件平台，或者合并现有算子来生成复合算子。

单元（Cell）：是张量和算子的集合，是所有神经网络单元的基本类。一个单元也可以包含其他单元，允许它们嵌套在同一个树状结构中。用户通过在单元中定义 construct 函数来表达神经网络的计算逻辑。construct 函数中的计算将在每个调用中执行。

模型（Model）：是 MindSpore 中的一个高级 API，它封装了一些低级 API，从而使用户能够更加方便地使用推理和训练功能。如果用户熟悉低级 API，并希望对计算过程建立细粒度的控制，则不一定需要使用模型。

从用户的角度来看，用 MindSpore 编写程序的核心是构建一个对应于神经网络的单元。这个过程首先从输入张量开始，可以是常量张量或参数张量。然后，用户使用 MindSpore 提供的不同算子构造一个单元。最后，用户可以使用模型封装这个单元来训练神经网络，也可以直接将输入数据传递给单元来执行推理任务。

代码 1 展示了 Python 中 MindSpore 程序。该程序展示了定义以及训练 LeNet[17]神经网络的过程。代码前 6 行，导入 MindSpore 合适的库。第 7 行到第 25 行定义了与 LeNet 神经网络相对应的 LeNet5 单元。_init_函数实例化 LeNet 使用的所有算子。construct 函数定义了 LeNet 的计算逻辑。第 26 行和第 27 行从 Mnisit 数据集读取数据，并生成一个迭代器 ds 用作训练的输入。第 28 行将 LeNet5 类实例化为 network。用 SoftmaxCrossEntropyWithLogits 函数来计算损失（loss）（第 29 行），并用 momentum 来优化参数（第 30 行到第 31 行），loss 和 optimizer 用来创建模型（Model）对象。最后，用 epoch 来控制迭代次数，调用模型的训练方法，并在每个 eval_step 对模型进行评估。

代码 1 LeNet5 的 MindSpore 实现

```
1  import mindspore.nn as nn
2  from mindspore.ops import operations as P
3  from mindspore.network.optim import Momentum
4  from mindspore.train import Model
5  from mindspore.nn.loss import SoftmaxCrossEntropyWithLogits
6  import mindspore.dataset as de
7  class LeNet5(nn.Cell):
8   def __init__(self):
9     super(LeNet5, self).__init__()
10    self.conv1 = nn.Conv2d(1, 6, 5, pad_mode='valid')
11    self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
12    self.fc1 = nn.Dense(16 * 5 * 5, 120)
13    self.fc2 = nn.Dense(120, 84)
14    self.fc3 = nn.Dense(84, 10)
```

```
15      self.relu = nn.ReLU()
16      self.max_pool2d = nn.MaxPool2d(kernel_size=2)
17      self.flatten = P.Flatten()
18    def construct(self, x):
19      x = self.max_pool2d(self.relu(self.conv1(x)))
20      x = self.max_pool2d(self.relu(self.conv2(x)))
21      x = self.flatten(x)
22      x = self.relu(self.fc1(x))
23      x = self.relu(self.fc2(x))
24      x = self.fc3(x)
25      return x
26  ds = de.MnistDataset(dataset_dir="./MNIST_Data")
27  ds = ds.batch(batch_size=64)
28  network = LeNet5()
29  loss = SoftmaxCrossEntropyWithLogits()
30  optimizer = nn.Momentum(network.trainable_params(),
31  learning_rate=0.1, momentum=0.9)
32  model = Model(network, loss, optimizer)
33  model.train(epoch=10, train_dataset=ds)
```

3 MindExpression&MindCompiler

3.1 基于源码转换的自动微分

目前主流的深度学习框架有三种自动微分技术：

- 基于静态计算图的转换：在编译时将网络转换为静态数据流图，然后将链式规则转换为数据流图，实现自动微分。
- 基于动态计算图的转换：以算子重载的方式记录前向执行时网络的操作轨迹，然后将链式规则应用到动态生成的数据流图中，实现自动微分。
- 基于源码的转换：该技术是从函数式编程框架演化而来，对中间表达（程序在编译过程中的表达形式），以即时（Iust-In-Time，JIT）编译的形式进行自动微分变换，支持复杂的流程控制场景、高阶函数和闭包。基于源码转化的自动微分如图 2 所示。

TensorFlow 早期采用静态计算图，而 PyTorch 采用动态计算图。静态图可以利用静态编译技术优化网络性能，但是组建或调试网络非常复杂。使用动态图非常方便，但很难在性能上达到极限优化。

MindSpore 开发了一种新的策略，即基于源码转换的自动微分。一方面，它支持流程控制的自动微分，因此构建像 PyTorch 这样的模型非常方便。另一方面，MindSpore 可以对神经网络进行静态编译优化，从而获得良好的性能。

MindSpore 自动微分的实现可以理解为对程序本身进行符号微分，因为 MindSpore IR 是函数式的中间表达，它与基本代数中的复合函数有直观的对应关系，只要已知基础函数的求导公式，就能推导出由任意基础函数组成的复合函数的求导公式。MindSpore IR 中每个原语操作可以对应为基础代数中的基础函数，这些基础函数可以构建更复杂的流程控制。

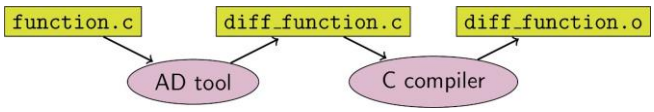


图 2 基于源码转换的自动微分

3.2 自动并行

随着深度学习的发展，为了实现更高的准确率和更丰富的应用场景，训练数据集和深度神经网络模型的规模日益增大。特别是自然语言处理（Natural Language Processing，NLP）领域，数据集的范围从 200MB 到 541TB 不等。模型的尺寸也从 BERT[18]的 3.4 亿个参数，Transformer-xl[19]的 8 亿个参数，GPT-2[19]的 150 亿个参数，到最近 NVIDIA Megatron-LM[20]的 80 多亿个参数。因此，在大型数据集上训练大型模型，不仅需要深度学习框架支持数据并行和模型并行，还需要支持混合并行。

当前的主流框架（如 TensorFlow[5]、Caffe[10]和 MXNet[7]）大多通过手动切分深度神经网络模型来实现模型并行，但手动切分模型难度非常大，对开发者的要求非常高，需要有丰富经验的专家来操作。实现混合并行（数据并行和模型并行同时进行）又极大增加了开发的复杂度。最近的研究成果[21–25]提出了简化混合并行的方法，但这些方法在几个方面都存在局限性。首先，这些方法在整个模型中固定了对张量维数的切分策略，这可能导致切分策略是次优的，因为模型的不同部分适用不同的策略。第二，[22，24]不适合用于许多用于语言建模和行人重新识别（Re-identification，ReID）的深度神经网络，这些网络往往是非线性网络。第三，[21]将划分策略搜索问题表述为混合整数程序，并使用现有的求解程序来求解，当处理大型模型时，速度会变得非常慢。此外，文献[21，24，25]旨在仅优化通信或内存开销，可能不会减少训练时间。

MindSpore 的目标是在模型训练过程中允许并行转换。为此，在并行化策略搜索中引入了张量重排布（Tensor Redistribution，TR），这使输出张量的设备布局在输入到后续算子之前能够被转换，如 3.2 图 3 中红色矩形所示。但是，对于复杂大型模型的搜索并行策略，在考虑张量重排布时，需要克服的挑战主要有两个。首先，既然张量重排布将通信算子（例如 AllGather）引入数据流图，那么如何像普通算子一样自动地对通信算子求导呢？对于每个相应的前向算子，都需要获取反向算子，用来更新可训练参数。目前的框架需要专家在反向阶段手动添加 SEND 和 RECV 源语来传递梯度，这对模型开发者来说是一项具有挑战性的工作，尤其是在模型比较复杂的情况下。其次，随着张量重排布对策略空间的极大扩展，如何为复杂的大型模型高效地找到一个好的策略？在功能和效率方

面，该算法需要为具有非线性结构的大模型快速找到一种策略。在性能方面，算法返回的策略应该会缩短端到端的训练时间。需要对运行成本进行仔细建模，这一过程也增加了人工成本。

MindSpore 针对上述两个挑战，推出了解决方案。为了实现通信算子的自动微分，MindSpore 定义了通信算子的反向算子。例如，AllGatheris 的反向算子为 ReduceScatter，SEND 的反向算子为 RECV 后跟一个 ADD。定义这些反向算子十分重要，因为 Auto-diff 过程可以一次性地区分整个前向图，而无须跳过任何算子，这也是为什么 Auto-diff 是 Auto-parallel 后面一步的原因。针对第二个挑战，在同时考虑计算和通信开销的情况下，建立一个代价模型来选择一个好策略。为了快速地为复杂大图找到一个好策略，提出了几种方法：一种是支持多图操作的算法，将原始图转换成线性图；一种是策略分离机制，在保证返回解的精度的同时，有效地缩小搜索空间。例如，ResNet50 在 8 台设备上搜索并行策略的时间在 1s 内，而返回的解决方案确实缩短了训练时间。例如，当模型较大（类的数量超过 128K）时，返回的解决方案与原始数据并行策略相比减少了大约 55%的训练时间。

代码 2 半自动并行配置中的并行过渡

```
class Submodel(nn.Cell):
def _init_(self, shape):
self.bn = BatchNorm(set_strategy={[4, 1]})
self.matmul = MatMul(set_strategy={[1, 1], [1, 4]})
self.W = Parameter(Tensor(shape), require_grad=True)
def construct(self, X):
 Y = self.bn(X)
 Z = self.matmul(y, self.W)
 return Z
```

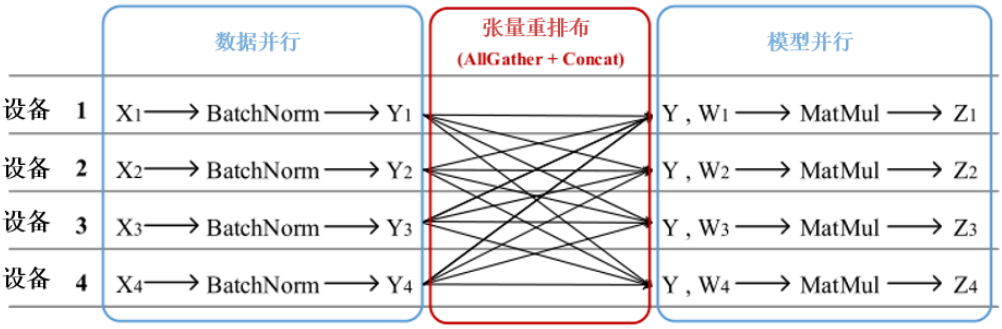


图 3 数据并行性向模型并行性转换

MindSpore 较灵活，它支持用户指定的高级策略配置，称之为半自动并行（semi-auto-parallel）。在代码 2 和图 3 中，展示了一个从数据到模型的并行转换的例子。该子模型的结构为 BatchNorm 算子后跟一个 MatMul 算子，广泛应用于 ResNet、ReID 等分类任务。在 BatchNorm 算子中，X 按行拆分为四部分，数据可以并行，效率非常高。在 MatMul 算子中，可学习参数的权重 W 被分成四部分，

模型可以并行，由于参数数量较多，这部分的模型并行更有效。由于 BatchNormi 的输出布局与 MatMul 的输入布局不同，所以框架插入了一个张量重排布（该例中为 AllGather 和 ConCat），这一过程对用户是透明的。用户也不必关注哪个设备运行了模型的哪个部分，框架会自动安排。然而，不同的模型结构在每个算子中具有不同大小的参数，如图 4 所示，并且它们使用于不同的切分策略。在图 4（3）中，将第一算子配置为模型并行，将后续算子配置为数据并行，也需要插入张量重排布，这样可以获得更好的性能。

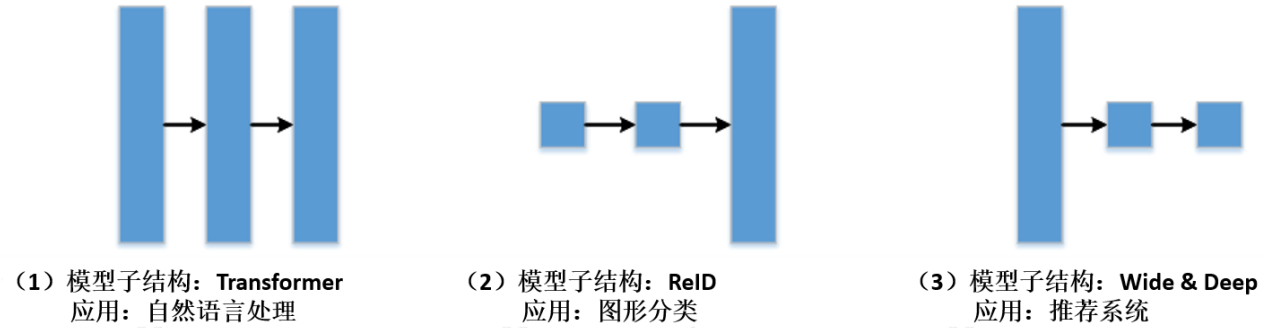


图 4 三种广泛使用的子结构

注：每个矩形代表一个层级（算子），其高度表示该层级中可学习参数的相对大小

在训练新模型时，多次配置 set_strategy，耗时耗力。在这种情况下，如果配置了自动并行（auto-parallel），则不需要指定 set_strategy，该算法将找到一个有效的策略。例如，当 ResNet 中的分类数量超过 130K 时，算法返回的策略导致在 50ms 内训练一个迭代。相比之下，原始数据并行训练一次迭代超过 111ms。更详细的评估，见 8.1 节。

3.3 动态图

由于编译器能获得静态图的全局信息，所以静态图在大多数情况下都表现出更好的运行性能。而动态图可以保证更好的易用性，使用户能够更加方便地构建和修改模型。为了同时支持静态图和动态图，大多数先进的训练框架需要维护两种自动微分机制，即基于 Tape 的自动微分机制和基于图的自动微分机制。从开发者的角度来看，维护两套自动微分机制成本较高；从用户的角度来看，在静态模式和动态模式之间切换也相当复杂。

MindSpore 采用了基于源码转换的自动微分机制，同时支持静态图和动态图，高效易用。在 MindSpore 中，称动态图为 Pynative 模式，因为代码使用 Python 解释器在这种模式下运行。如代码 3 所示，从静态图模式切换到动态图模式只需要一行代码，反之亦然。此外，为了提高 Pynative 模式的运行效率，MindSpore 支持 staging 机制，如代码第 4 行所示。在函数（fc_relu）前添加 ms_function 装饰器，该函数将以静态图模式编译和运行。

代码 3 LeNet5 的 MindSpore 实现

```
1  from mindspore import context
```

```
import numpy as np
class LeNet5(nn.Cell):
  @ms_function
  def fc_relu(self, x):
    x = self.relu(self.fc2(x))
    x = self.fc3(x)
  def construct(self, x):
    x = self.max_pool2d(self.relu(self.conv1(x)))
    x = self.max_pool2d(self.relu(self.conv2(x)))
    x = self.flatten(x)
    x = self.relu(self.fc1(x))
    x = fc_relu(x)
    return x
data=np.ones((batch_size,3,224,224),np.float32)*0.01
net = LeNet5()
# switch to Pynative mode
context.set_context(mode=context.PYNATIVE_MODE)
pynative_out = net(data)
# switch back to static graph mode
context.set_context(mode=context.GRAPH_MODE)
graph_out = net(data)
```

静态图和动态图的架构如 3.3 图 5 所示。在动态图模式中，框架遍历模型的所有算子，为每个算子生成计算图，并将计算图传递给后端进行前向计算。在完成前向计算后，框架为模型生成前向和反向计算图，并将它们发送到后端中执行。由于 MindSpore 使用基于源码转换的自动微分机制，因此在生成反向图时，可以省略用户为检查其模型而设置的所有代码，如 pdb 和 print。

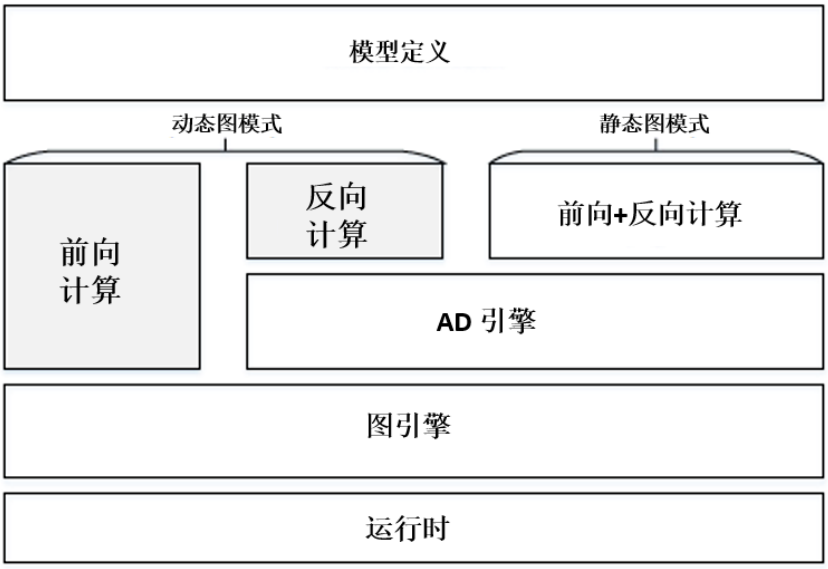


图 5 静态图和动态图的架构

3.4 二阶优化

随着深度学习的发展，深度学习在许多应用领域都表现出了优异的性能，包括图像识别、目标检测和自然语言处理等领域。而优化器是深度学习中非常重要的一部分，基于深度学习的快速发展，有关优化器的研究也越来越多。

常见的优化算法可分为一阶优化算法和二阶优化算法。梯度下降算法（Gradient Descent, GD）是机器学习中最经典的一阶优化算法，也是众多机器学习算法中最常用的优化算法。常用的一阶优化算法（比如 SGD 算法）参数更新规则如下：$\theta = \theta - \eta \nabla_\theta$，其中$\theta$是需要更新的参数，$\eta$是学习率，$\nabla_\theta$是损失函数对于参数的梯度。

此外，通过引入动量和自适应学习率衰减等策略，GD 产生了许多变体，例如 Momentum、Nesterov、AdaGrad、RMSprop、Adadelta 和 Adam 等。这些改进后的优化算法可以利用随机梯度的历史信息来自适应地更新步长，使得它们更容易调参，而且方便使用。但考虑到神经网络的损失函数是高度非凸函数，而且曲面的曲率不平衡，因此使用更多的信息来指导参数更新会得到更好的收敛速度，例如二阶矩阵信息。

二阶优化算法利用目标函数的二阶导数进行曲率校正来加速一阶梯度下降。其收敛速度更快，能高度逼近最优值，几何上下降路径也更符合真实的最优下降路径。与一阶优化算法相比，二阶优化算法则是先将∇_θ与一个矩阵G^{-1}相乘，产生如下的更新规则：$\theta = \theta - \eta G^{-1} \nabla_\theta$，其中 G 即为二阶信息矩阵，不同的二阶优化算法中的 G 定义不见相同，常见的二阶优化算法有牛顿法，自然梯度法等，分别对应的二阶信息矩阵 G 为 Hessian 矩阵，Fisher 矩阵。

Hessian 矩阵是一个由多变量实值函数的所有二阶偏导数组成的方块矩阵。Hessian 矩阵可以表示为：$H_{ij} = \frac{\partial^2 f}{\partial \theta_i \partial \theta_j}$ ，其中 f 即为损失函数，θ是需要更新的参数。Fisher 矩阵是最大似然函数求导的协方差矩阵。Fisher 矩阵可以表示为：$\mathrm{F} = \mathrm{E}_{(x,y)\sim p(x,y|\theta)}[\frac{\partial \log \mathrm{p}(y|x,\theta)}{\partial \theta}\frac{\partial \log \mathrm{p}(y|x,\theta)^T}{\partial \theta}]$ ，其中θ是需要更新的参数，模型的联合分布为 $\mathrm{p}(\mathrm{x},\mathrm{y}|\theta) = p(y|x,\theta)q(x)$，$\mathrm{q}(x)$是$x$的样本分布，与参数$\theta$无关。$\log \mathrm{p}(y|x,\theta)$ 是损失函数，该损失函数是对数似然函数。

二阶优化算法虽然收敛速度快，但是计算二阶矩阵的逆的时间复杂度为$O(n^3)$，当模型参数量为 n_θ 时，对应的二阶信息矩阵的大小为 $n_\theta \times n_\theta$。在深度学习模型中，n_θ 常常在数百万的量级，此时二阶信息矩阵的逆无法计算。因此如何降低二阶信息矩阵求逆的计算复杂度成为关键问题。

MindSpore 针对该问题，提出了自研算法 THOR（Trace-based Hardware-driven layer-ORiented Natural Gradient Descent Computation），THOR 基于自然梯度法对二阶算法进行改进。主要有三个改进点。

3.4.1 矩阵更新频率

实验发现，Fisher 矩阵的 F 范数（Frobenius norm）在前期变化剧烈，后期逐渐变稳定，从而假设$\{F^k\}_{k=1}^n$是一个马尔可夫过程，可以收敛到一个稳态分布π，其中F^k代表第 k 个迭代时的 Fisher 矩阵。因此，在训练过程中逐步增大 Fisher 矩阵的更新间隔，可以在不影响收敛速度的情况下，减少训练时间。

3.4.2 分块更新

将 Fisher 矩阵按层解耦，分别针对每一层的 Fisher 矩阵做实验，发现有些层的 Fisher 矩阵趋于稳态的速度更快，因此想到，更加细粒度的去调整每一层的更新频率。使用二阶信息矩阵的迹作为判断条件，当迹的变化情况小于某一阈值时更新该层的 Fisher 矩阵，否则沿用上一个迭代得 Fisher 矩阵，具体更新公式如下：

$$\Delta^k = \frac{\left|\left|tr\left(F_i^k + \lambda I\right)\right| - \left|tr\left(F_i^{k-1} + \lambda I\right)\right|\right|}{\left|tr\left(F_i^k + \lambda I\right)\right|}$$

$$\begin{cases} \text{更新} F_i^k \text{，} & \text{当} \Delta^k \in (w_1, +\infty) \\ \text{不更新} F_i^k \text{，沿用上一个迭代的} F_i^{k-1} \text{，} & \text{当} \Delta^k \in [w_2, w_1] \\ \text{停止更新} F_i^k \text{，后期都使用} F_i^{k-1} \text{，} & \text{当} \Delta^k \in [0, w_2] \end{cases}$$

3.4.3 硬件感知矩阵切分

THOR 假设了 Fisher 矩阵层之间是解耦的以及每个网络层中的输入和输出块之间也是独立的，例如将每层网络的输入输出切分为 n 个块，这 n 个块之间即是独立的，根据该假设对二阶信息矩阵做进一步的切分，从而提高了计算效率。

其中 n 是矩阵信息损失和硬件性能中取的平衡点。首先根据 Fisher 矩阵中维度最大的那一层，确定矩阵切分维度，在 Ascend 910 上，拿 ResNet-50 举例，确定矩阵切分维度为 [1,16,32,64,128,256,512,1024,2048],然后计算每个矩阵切分维度下的矩阵损失和性能数据，得到下图，得到交叉点为 106，与 128 最接近，最后确定矩阵切分维度为 128。

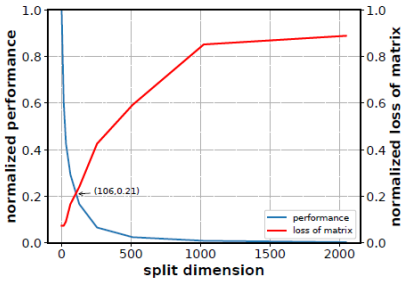


图 6 矩阵切分维度获取示意图

代码 4 MindSpore 中如何使用 THOR 训练网络

```
from mindspore.train.train_thor import ConvertModelUtils
from mindspore.nn.optim import THOR
elif cfg.optimizer == "Thor": #创建 THOR 优化器
    from src.utils import get_bert_thor_lr, get_bert_thor_damping
    lr = get_bert_thor_lr()
    damping = get_bert_thor_damping()
    optimizer = THOR(network, lr, damping, cfg.Thor.momentum,
                     cfg.Thor.weight_decay, cfg.Thor.loss_scale, cfg.batch_size,
                     decay_filter=lambda x: 'layernorm' not in x.name.lower() and 'bias' not in x.name.lower())
    context.set_context(max_call_depth=10000)
```

```
11    model = Model(net_with_grads)
12    model = ConvertModelUtils().convert_to_thor_model(model, network=net_with_grads, optimizer=optimizer,
frequency=cfg.Thor.frequency) #保存 THOR 所需的二阶信息
13    model.train(new_repeat_count, ds, callbacks=callback,
14                    dataset_sink_mode=(args_opt.enable_data_sink == "true"), sink_size=args_opt.data_sink_steps)
```

4 MindData

4.1 数据处理

数据处理是一个单独的异步流水线，它为张量馈入模型做好准备。在流水线内，数据被组织成一系列具有不同列的行，所有列都用列名标识，并且可以独立访问。流水线总是从一个源数据集算子开始，该算子从磁盘（如 MindDataset）读取数据，并包含选择 shuffling 和 sharding 策略的标记。为了访问流水线中的数据，可使用迭代器（Python 访问）或设备队列（直接发送到加速器设备）。

数据处理的体系结构本质上是流水线且并行的，管道的运行默认是异步的，但用户也可以在图中插入同步点，以便流水线算子能够实时反馈循环。未来，流水线将进行动态调整，以充分利用所有可用的资源，包括用于图像处理的硬件加速器或用于缓存的可用内存，用户只需配置默认参数以获得良好的性能，而无须进行手动调优。

为了让用户实现快速迁移，数据处理支持用户已有的 Python 数据增强作为自定义算子以 pyfunc 方式接入，同时现有的 Python 数据集类可以作为参数传递给 GeneratorDataset 接入。不断涌现的新网络对数据处理的灵活性提出了新要求。数据处理支持使用用户自定义的函数（或调度）来更改参数，例如 mini-batch 大小；还支持用户对整个 mini-batch 进行自定义转换，以进行 mini-batch 级的图像大小或多行操作（如图像混合）。为了获得更多样的增强，每个样本的增强可以从一个数据增强操作列表中随机选择。另外，可以通过外部搜索（即 Fast AutoAugment）在候选数据增强操作列表中进行选择，最近关于 randomAugment 和 uniformAugment 的研究表明，对很多数据集来说，只随机从大量数据增强策略中选择合适的，不采用耗费大量搜索时间的自动数据增强方法，同样能获得很好的结果。在训练中收集的度量的反馈（如 loss）也可以传回至数据集，以在数据处理中执行动态调整，如 Adversarial AutoAugment 中所示。

4.2 MindRecord

MindRecord 数据集格式将用户的训练数据按不同类型、不同页面进行存储，建立轻量化、高效的索引，并提供一套接口，方便用户将训练数据转换成 MindRecord 格式，然后使用 MindDataset 将训练数据读入数据集。同时，可以快速从数据集读取重要的元数据（即数据集大小或数据布局）以提高性能或简化用户访问。MindRecord 能够支持小块数据的高效顺序 I/O，还可以根据用例需要支持高效的随机行访问和下沉过滤。随着新用例的出现，进一步优化的功能会下沉到该数据集。

4.3 MindInsight

MindInsight 提供训练看板、溯源、性能分析和调试器的功能。这些功能可以帮助开发者在训练模型的过程中发现模型训练过程中出现的偏差，发现超参、数据增强等因素的影响，并对模型进行调试和性能分析。通过 MindInsight，开发者可以更好地观察和理解训练过程，这样可以提升模型优化效率并优化开发者体验。MindInsight 架构如 4.3 图 7 所示。

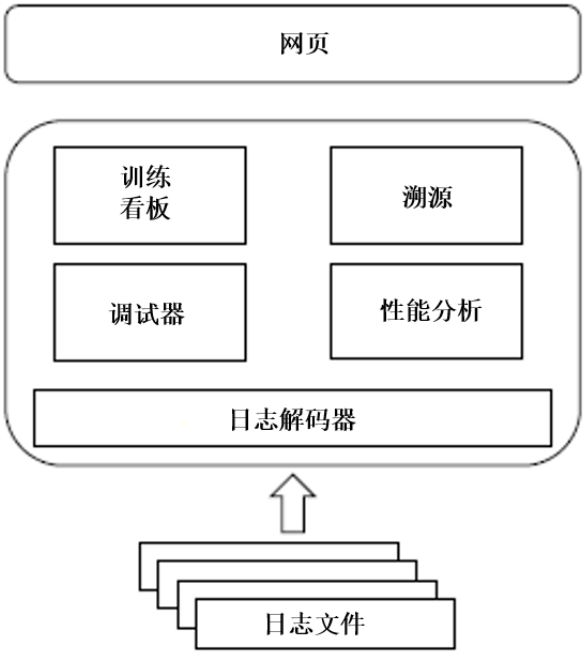


图 7 MindInsight 架构

MindInsight 以模型训练中生成的日志文件作为输入。通过文件解析、信息提取、数据缓存、图表绘制等一系列过程，将二进制训练信息转换成易于理解的图表，并展示在网页上。

4.3.1 训练看板

MindInsight 支持训练过程可视化。训练看板支持在页面上查看训练过程。训练看板包括训练标量信息、参数分布图、计算图、数据图、数据抽样等模块。

训练看板是 MindInsight 在训练过程呈现方式上的创新。通过在一个页面中集成多种类型的数据，用户只需要打开训练看板就能概览训练情况。图 8 展示了训练看板的一个示例。

图 8 MindInsight 训练看板

4.3.2 溯源

MindInsight 还支持溯源可视化。通过将多个训练的溯源信息整合到表格和图形中，用户可以轻松选择最优的数据处理流水线和超参设置。溯源可视化包括模型溯源可视化和数据溯源可视化。模型溯源函数可以记录模型训练的关键参数，如损失函数、优化器、迭代次数、精度等。MindInsight 中显示多次训练的超参和评估指标，帮助用户选择最优超参。未来，将逐步引入辅助超参推荐，帮助用户快速优化超参。图 9 展示了模型溯源可视化的一个示例。

数据溯源可视化可以记录每次模型训练的数据处理流水线。多次训练的数据处理模式显示在 MindInsight 上，帮助用户选择最优的数据处理流水线。图 10 展示了数据溯源可视化的一个示例。

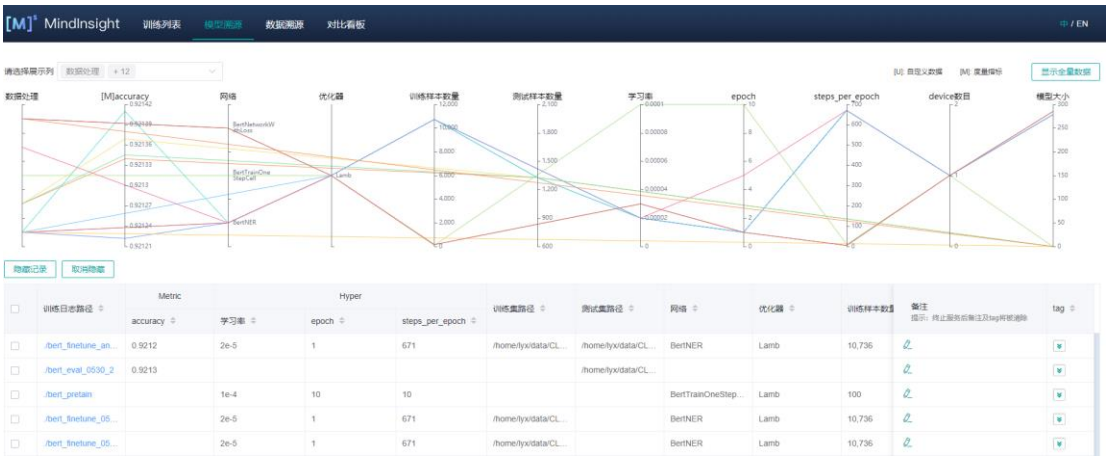


图 9 MindInsight 模型溯源

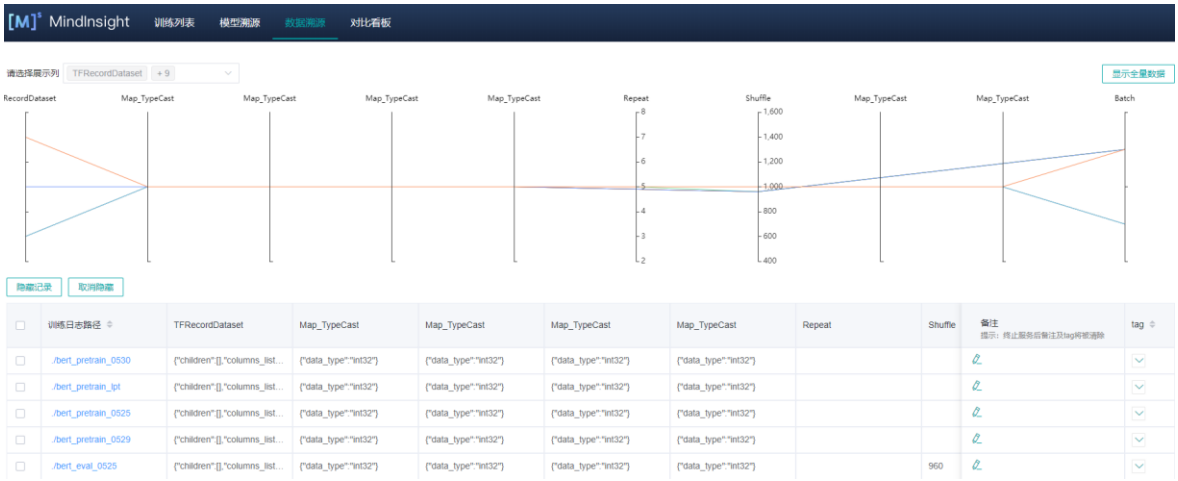


图 10 MindInsight 数据溯源

4.3.3 性能分析

为了满足工程师对神经网络性能优化的要求，我们设计并实现了性能分析工具。它可以打开 MindSpore 神经网络的执行过程，收集各算子的时间、内存等数据。MindInsight 会进一步对性能数据进行整理、分析，呈现多维度、多层次的性能分析结果。这为神经网络性能优化提供了方向。性能数据展示维度包括：

1. **迭代轨迹**

   迭代轨迹将神经网络的执行分解为多个阶段，包括数据读取、前向和反向计算、Allreduce 等，工程师可以快速发现哪个阶段是目前的性能瓶颈。

2. **算子性能**

   算子性能将聚合并整理算子执行时间的数据，这样工程师可以很容易找出耗时的算子。

3. **时间轴**

   时间轴可以显示芯片侧执行流和执行任务的状态，帮助工程师进行细粒度的性能分析。

4. **MindData Profiling**

   可以帮助用户定位和分析训练数据获取过程中的瓶颈，工程师可以通过提高瓶颈算子的线程数等方法来提高性能。

4.3.4 调试器

神经网络训练中经常出现数值误差情况，如无穷大等，用户希望分析训练无法收敛的原因。但是，由于计算被封装为黑盒，以图的方式执行，工程师很难定位其中的错误。MindInsight 调试器是图模式下训练调试的工具，用户可以在训练过程中查看图的内部结构以及节点的输入/输出，例如查看一个张量的值，查看图中的节点对应的 Python 代码等。此外，用户还可以选择一组节点设置条件断点，实时监控节点的计算结果。

5 MindArmour

5.1 对抗性攻击防御

对抗性攻击[26，27]对机器学习模型安全的威胁日益严重。攻击者可以通过向原始样本添加人类不易感知的小扰动来欺骗机器学习模型[28，29]。为了防御对抗性攻击，MA 提供了攻击（对抗样本生成)、防御（对抗样本检测和对抗性训练)、评估（模型鲁棒性评估和可视化）等功能。

给定模型和输入数据，攻击模块提供简单的 API，能够在黑盒和白盒攻击场景下生成相应的对抗样本。这些生成的对抗样本被输入防御模块，以提高机器学习模型的泛化能力和鲁棒性。防御模块还实现了多种检测算法，能够根据恶意内容或攻击行为来区分对抗样本和正常样本。评估模块提供了多种评估指标，开发者能够轻松地评估和可视化模型的鲁棒性。

5.2 隐私保护人工智能

隐私保护也是人工智能应用的一个重要课题。MA 考虑了机器学习中的隐私保护问题，并提供了相应的隐私保护功能。针对已训练模型可能会泄露训练数据集中的敏感信息问题[30，31]，MA 实现了一系列差分隐私优化器，自动将噪声加入反向计算生成的梯度中，从而为已训练模型提供差分隐私保障。特别地，优化器根据训练过程自适应地加入噪声，能够在相同的隐私预算下实现更好的模型可用性。同时提供了监测模块，能够对训练过程中的隐私预算消耗进行动态监测。用户可以像使用普通优化器一样使用这些差分隐私优化器。

6 端云协同架构

MindSpore 旨在构建一个从端侧到云侧全场景覆盖的人工智能框架，将支持“端云”协同能力，包括模型优化、端侧训练和推理、联邦学习等过程，如图 11 所示。

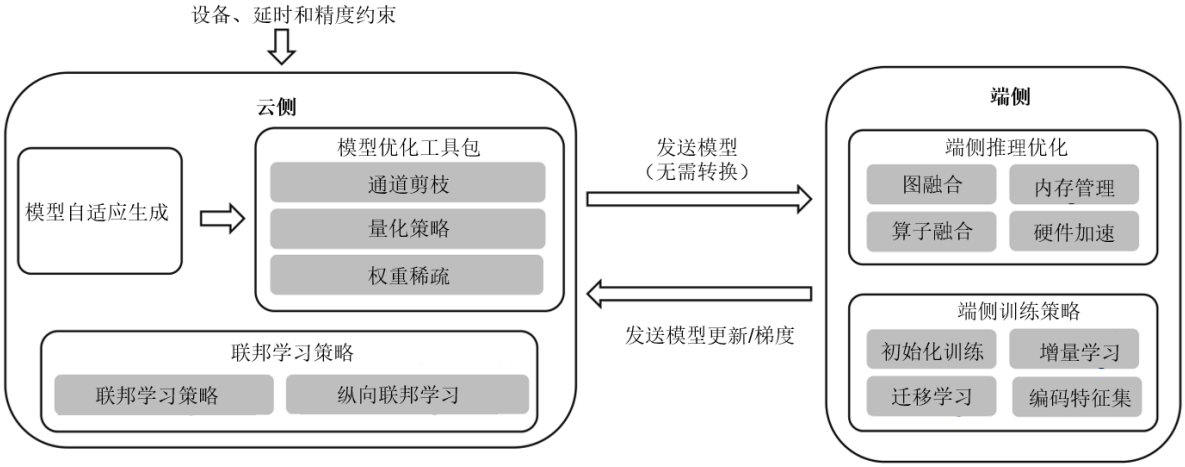


图 11 MindSpore 端云协同架构

1. **模型生成与优化工具包**

   移动和边缘设备通常资源有限，如电源和内存。为了帮助用户利用有限的资源部署模型，MindSpore 将支持一系列优化技术，如模型自适应生成、量化策略等，如图 11 左侧所示。模型自适应生成是指应用神经架构搜索（Neural Architecture Search，NAS[32]）技术来生成在不同设备下时延、精度、模型大小均满足需求的模型。量化策略是指通过以更少位数的数据类型来近似表示 32 位有限范围浮点数据类型的过程，MindSpore 支持训练后量化和量化感知训练。

2. **端侧训练和联邦学习**

   虽然在大型数据集上训练的深度学习模型在一定程度上是通用的，但是在某些场景中，这些模型仍然不适用于用户自己的数据或个性化任务。

   MindSpore 计划提供端侧训练方案，允许用户训练自己的个性化模型，或对设备上现有的模型进行微调，同时避免了数据隐私、带宽限制和网络连接等问题。端侧将提供多种训练策略，如初始化训练策略、迁移学习、增量学习等。MindSpore 还将支持联邦学习，通过向云侧发送模型更新/梯度来共享不同的数据，如图 11 所示。基于联邦学习，模型可以学习更多的通用知识。

3. **移动和边缘设备部署**

   MindSpore 提供轻量级的计算引擎，支持模型在设备上高效执行。在将预先训练好的模型部署到设备侧时，通常需要进行模型转换。然而，这个过程可能导致性能降低和精度损失。在 MindSpore 中，端侧推理模式能够兼容云上训练好的模型，因此，在设备上部署已经训练好的模型时，无须进行转换，这样避免了潜在的性能损失。此外，MindSpore 还内置了针对设备的各种自动优化，例如图和算子融合、精细复杂的内存管理、硬件加速等，如图 11 右侧所示。

7 MindSpore Serving

7.1 模块简介

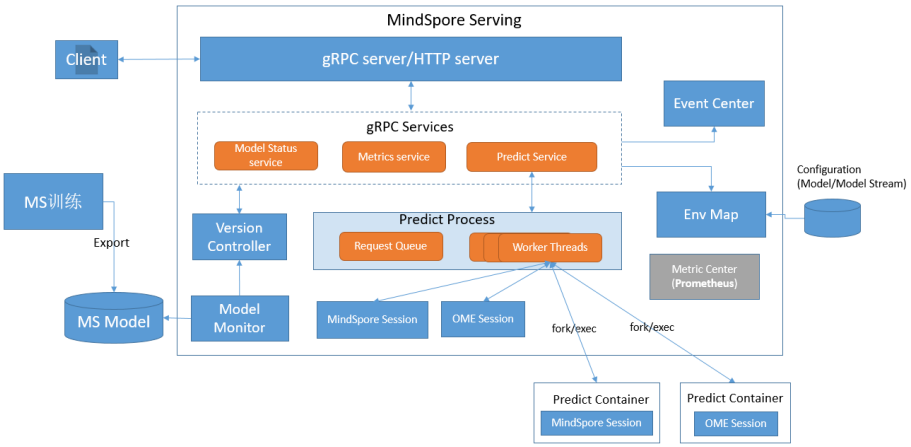
MindSpore Serving 是一个轻量级、高性能的推理服务模块，旨在帮助 MindSpore 开发者在生产环境中高效部署在线推理服务。当用户使用 MindSpore 完成模型训练后，导出 MindSpore 模型，即可使用 MindSpore Serving 创建该模型的推理服务。

7.2 功能

MindSpore Serving 提供如下功能：

- 加载模型文件生成推理引擎，提供推理功能；
- 预测请求和处理结果的消息交互，支持 gPRC 和 RESTful 两种请求方式；
- 预测接口调用，执行预测，返回预测结果；
- 模型的生命周期管理；

- 服务的生命周期管理；
- 多模型多版本的管理。



MindSpore Serving 架构中各模块功能如下：

- gRPC server：处理 gRPC 请求，支持同步、异步方式；
- HTTP server：处理 HTTP 请求，支持同步、异步方式；
- Service：MS Serving 提供的服务的基本执行单元，通过 gRPC 定义其接口；典型的 Service 包含：预测、模型版本查询等。
- Predict Process：推理请求的处理模块，包含有推理请求的 cache 队列以及处理请求的 worker 线程池；worker 线程数与卡的数目相同；每个 worker 线程对应一个推理 session；推理 session 可以被 Predict Container 管理（进程级别隔离），或者直接由 MindSpore Serving 管理（线程级别隔离）；
- Version Controller：负责 MindSpore Model 的加载以及版本管理，版本策略可配置。
- Model Monitor：基于 period polling 方式检测 MindSpore Model。

8 评估

在本节中，我们评估了 MindSpore 在自动并行方面的性能，还在华为昇腾系列芯片集群上进行了实验，并与当前主流的深度学习框架进行了性能对比。结果表明，我们的系统具有以下特点：（1）高通量；（2）加速比随着集群的增加保持稳定。此外，我们提供了几种模型的推理性能，并获得了比主流的深度学习框架更好的性能。

8.1 自动并行

我们在一个有 8 台设备的昇腾服务器上进行了实验。使用标准的 ResNet50，分别在原始数据并行和自动并行下训练。原始数据并行和自动并行的比较结果如图 12 所示，分类数目从 1K 到 1024K 左右。我们观察到当分类数目小于 32K 时，两种模式的迭代次数几乎相同，因为该算法找到了数据并行（Data-Parallelism，DP）策略。当分类数目超过 64K 后，自动并行模式相比数据并行模式，在性能上有了较大提升。这是因为当分类数目超过 64K，该算法返回的策略是混合并行（Hybrid Parallelism，HP）(表示模型头部的算子是数据并行的，而 MatMul 算子是模型并行的，如图 3 所示)。这一过程节省了很大的通信开销，因为模型并行避免了 MatMul 算子中由可学习参数同步引起的大量 AllReduce。

当分类数目大于 256K 时，由于“内存溢出”（Out of Memory，OOM）失败，数据并行模式甚至不能运行，而自动并行模式下，ResNet50 训练成功，迭代时间略有增加。

图 12 使用原始数据并行和自动并行训练的 ResNet50 的性能比较

8.2 性能测试

8.2.1 训练性能

我们在昇腾系列芯片的集群上完成了 ResNet50 和 BERT-large 模型的训练，并与 TensorFlow 进行了对比。TensorFlow 使用 NVIDIA DGX-2(16x V100 32G)，我们通过配置不同的 AI 设备数来对比 MindSpore 和 TensorFlow 的训练性能。如图 13 和图 14 所示，随着 AI 设备数的增加，MindSpore 可以获得比 TensorFlow 更大的吞吐量。

如图 15 所示，使用昇腾系列芯片的 MindSpore 在训练 ResNet-50v1.5 和 BERT-large 时都可以获得超过 93%的加速比。而使用 GPU 的 TensorFlow 在训练 ResNet50 时只能达到 79%的加速比，在训练 BERT-large 时只能达到 86%的加速比。

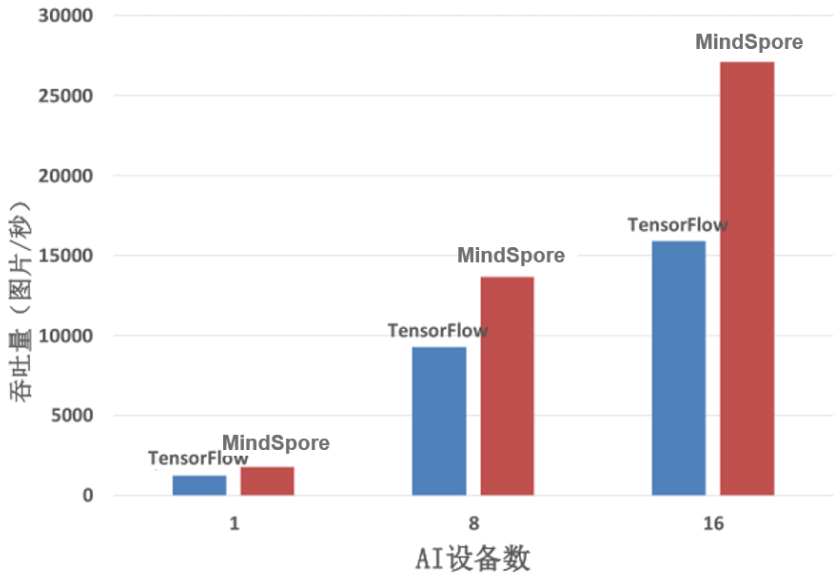


图 13 用 MindSpore 和 TensorFlow 训练 ResNet-50v1.5 的吞吐量的比较

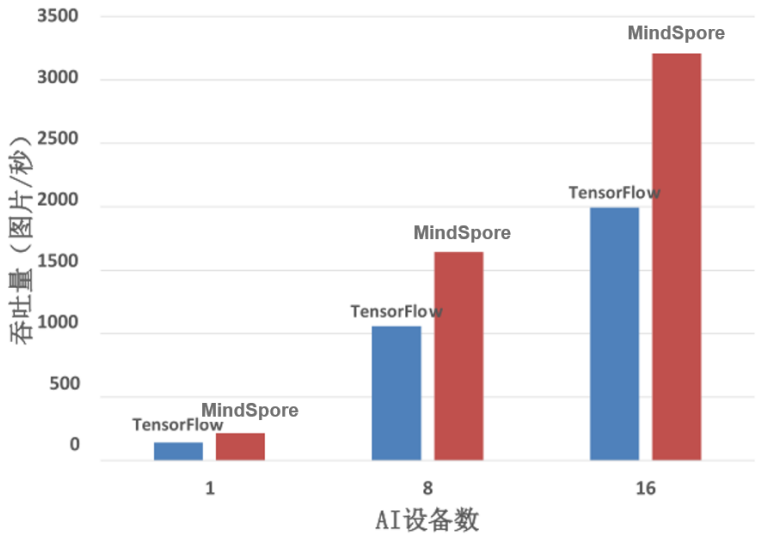


图 14 用 MindSpore 和 TensorFlow 训练 BERT-large 的吞吐量的比较

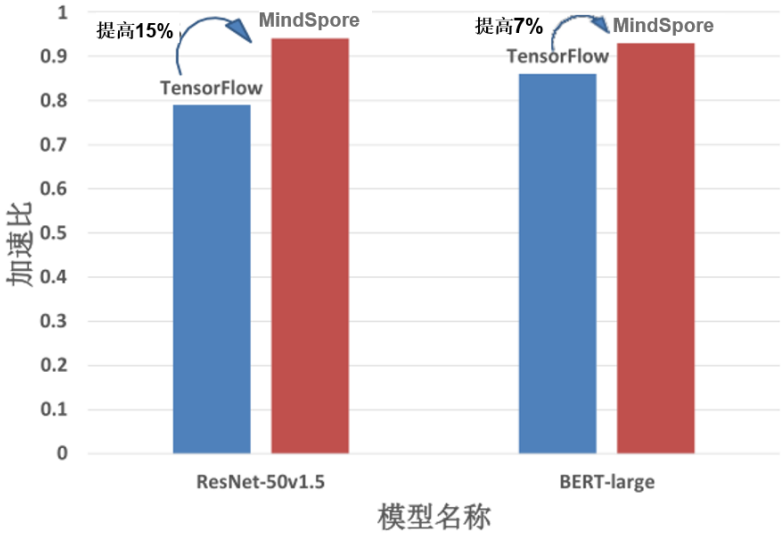


图 15 用 MindSpore 和 TensorFlow 在 ResNet-50v1.5 和有 16 个 AI 设备的 BERT-large 上训练的加速比的比较

8.2.2 推理性能

我们还使用了不同的轻量级模型在华为 Mate 30 手机上进行了推理实验，如表 2 所示。在 CPU 上执行，并将推理延迟与 TensorFlow 进行比较。结果表明，MindSpore 比 TensorFlow 具有更短的推理时延。

表 2 用 MindSpore 和 TensorFlow Lite 在华为 Mate 30 手机进行推理的性能比较

模型	**线程数**	**MindSpore (单位：毫秒)**	**TensorFlow Lite (单位：毫秒)**
inception v4	1	657.921	787.012
	2	345.307	431.289
	4	231.397	312.81
mobilenet v1 1.0 224 frozen	1	33.028	37.471
	2	17.156	20.4
	4	11.761	13.871
mobilenet v1 1.0 224 quant frozen	1	17.216	56.246
	2	9.614	39.333
	4	6.508	31.902
nasnet mobile	1	59.885	70.212
	2	39.121	47.017

	4	32.559	33.539
squeezenet	1	40.308	53.488
	2	21.776	30.313
	4	16.049	21.298

9 结论及未来工作方向

在本文中，我们介绍了新的深度学习框架——MindSpore，重点介绍了 MindSpore 的重要组成部分：ME、MD 和 MA 以及 MindSpore 的几个新特性，即自动并行、自动微分和端边云协同训练，这使 MindSpore 能够实现易开发、高效执行、全场景覆盖三大目标。此外，MindSpore 还提供可视化和防御工具，使训练过程对各种对抗性攻击都可见，并且能够抵抗这些攻击。2020 年 3 月 28 日华为发布了 MindSpore 的第一个版本，现在，MindSpore 已经成功与昇腾系列芯片适配，并应用于华为的各种产品，从智能手机到云。今后，我们计划从几个方面改进现有的 MindSpore 系统。对于 ME 部分，我们希望考虑拓扑感知的调度，以满足多节点集群中不同的通信需求；对于 MD，我们将为人工智能工程师提供更灵活的工具来处理和增强不同类型的数据；对于 MA，我们将尝试在 CV/NLP 领域防御各种对抗性攻击。

参考文献

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Se- nior, Vincent Vanhoucke, Patrick Nguyen, Brian Kings- bury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing maga- zine*, 29, 2012.

[3] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, A Rusu Andrei, and Veness Joel. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research, 3(Feb):1137– 1155, 2003.*

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Ten- sorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[6] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Auto- matic differentiation in pytorch. 2017.

[7] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient ma- chine learning library for heterogeneous distributed sys- tems. *arXiv preprint arXiv:1512.01274*, 2015.

[8] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[9] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clay- ton. Chainer: a next-generation open source frame- work for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty- ninth annual conference on neural information process- ing systems (NIPS)*, volume 5, pages 1–6, 2015.

[10] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadar- rama, and Trevor Darrell. Caffe: Convolutional archi- tecture for fast feature embedding. In Proceedings *of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[11] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371– 380, 2009.

[12] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on* Machine *Learning*, pages 201–210, 2016.

[13] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Sym- posium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[14] Lingchen Zhao, Qian Wang, Qin Zou, Yan Zhang, and Yanjiao Chen. Privacy-preserving collaborative deep learning with unreliable participants. *IEEE Transactions on Information Forensics and Security*, 15:1486–1500, 2019.

[15] Cormac Flanagan, Amr Sabry, Bruce F Duba, and Matthias Felleisen. The essence of compiling with con- tinuations. In *Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and im- plementation*, pages 237–247, 1993.

[16] Bart van Merrienboer, Olivier Breuleux, Arnaud Berg- eron, and Pascal Lamblin. Automatic differentiation in ml: Where we are and where we should be going. In *Advances in neural information processing systems*, pages 8757–8767, 2018.

[17] Yann LeCun et al. Lenet-5, convolutional neural net- works. *URL: http://yann. lecun. com/exdb/lenet*, 20:5, 2015.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirec- tional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[20] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[21] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. Mesh- tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10414–10423. Curran Associates, Inc., 2018.

[22] Zhihao Jia, Sina Lin, Charles R. Qi, and Alex Aiken. Ex- ploring hidden dimensions in accelerating convolutional neural networks. In *Proceedings of the 35th Interna- tional Conference on Machine Learning (ICML)*, pages 2274–2283. PMLR, 2018.

[23] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *Pro- ceedings of the 2nd Conference on Machine Learning and Systems (MLSys)*. ACM, 2019.

[24] Minjie Wang, Chien-chin Huang, and Jinyang Li. Sup- porting very large models using automatic dataflow graph partitioning. In *Proceedings of the Fourteenth EuroSys Conference (EuroSys)*. ACM, 2019.

[25] Linghao Song, Jiachen Mao, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. Hypar: Towards hybrid par- allelism for deep learning accelerator array. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 56–68. IEEE, 2019.

[26] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial exam- ples. *arXiv preprint arXiv:1412.6572*, 2014.

[27] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[28] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Ad- versarial examples in the physical world. *arXiv preprint arXiv:*1607.*02533*, 2016.

[29] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee sympo-* sium *on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[30] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence infor- mation and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[31] Reza Shokri, Marco Stronati, Congzheng Song, and Vi- taly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[32] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.