华为昇腾神经网络加速器性能评测与优化

鲁蔚征[1),2),3)] 张 峰[2),3)] 贺寅烜[3)] 陈跃国[1),2),3)] 翟季冬[4)] 杜小勇[1),2),3)]

[1)](中国人民大学大型科学仪器共享平台 北京 100872)

[2)](数据工程与知识工程教育部重点实验室(中国人民大学) 北京 100872)

[3)](中国人民大学信息学院 北京 100872)

[4)](清华大学计算机科学与技术系 北京 100084)

摘 要 华为昇腾是一款新型神经网络加速器.与 GPU 相比,昇腾加速器专门面向神经网络计算,设计了专用计算单元,核心算力集中在低精度,基于昇腾的软件栈与 GPU 有所差异.现有研究大多专注于 GPU 上的深度学习负载性能分析和优化,由于昇腾平台推出不久且具有新的体系结构特征,其实际表现仍有待探索.为深入挖掘昇腾的性能和优化方法,本文对其进行了系统性的评测和分析,包括:(1)基于标准数据集在四个端到端神经网络(ResNet、Transformer、DeepFM 和 LSTM)上对昇腾和 GPU 的性能和功耗进行了对比;(2)研究了昇腾上深度学习框架、算子和混合精度训练优化策略;(3)测试三个计算密集型算子(全连接、卷积和 RNN)的浮点计算能力、硬件利用率和访存性能.评测结果表明:华为昇腾加速器适合进行稠密型神经网络工作负载,且功耗低于 GPU;使用昇腾进行模型训练,需要将神经网络模型从 32 位精度量化到 16 位精度.针对昇腾的体系结构和编译软件栈特点,本文提出如下优化策略:深度学习框架开发时应进行整图编译构建,进行算子融合;算子开发时应合理设置分块大小,尽量使用低精度实现算子;模型训练时要合理设置混合精度参数.

关键词 深度学习;神经网络加速器;华为昇腾;高性能计算;评测基准

中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2022.01618

Evaluation and Optimization for Huawei Ascend Neural Network Accelerator

LU Wei-Zheng[1),2),3)] ZHANG Feng[2),3)] HE Yin-Xuan[3)] CHEN Yue-Guo[1),2),3)]
ZHAI Ji-Dong[4)] DU Xiao-Yong[1),2),3)]

[1)](*Office of Research Infrastructure, Renmin University of China, Beijing* 100872)

[2)](*Key Laboratory of Data Engineering and Knowledge Engineering of Ministry of Education, Renmin University of China, Beijing* 100872)

[3)](*School of Information, Renmin University of China, Beijing* 100872)

[4)](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract The great success achieved by deep neural networks (DNNs) mainly relies on the computation ability provided by modern chips. Nvidia's high performance and general-purpose Graphics Processing Units (GPUs) are widely used to build deep learning tools and software. There is an industry-wide trend towards domain specific neural network accelerators to extend deep learning performance. For example, Google has released Tensor Processing Unit (TPU) and has deployed TPUs in the data center; MIT proposed an energy-efficient reconfigurable accelerator for deep convolution neural networks. In addition to these accelerators, Huawei has developed the Ascend accelerator, including Ascend 910 for training and Ascend 310 for inference. Ascend

收稿日期:2021-05-24;在线发布日期:2021-11-22.本课题得到国家重点研发计划项目(2018YFB1004401)、国家自然科学基金(U1711261,62172419)、教育部产学融合协同育人(华为昇腾)项目资助.**鲁蔚征**,硕士,工程师,中国计算机学会(CCF)会员,主要研究方向为高性能计算和数据科学.E-mail:luweizheng@ruc.edu.cn.**张 峰**(通信作者),博士,副教授,中国计算机学会(CCF)专业会员,主要研究方向为大数据系统和高性能计算.E-mail:fengzhang@ruc.edu.cn.**贺寅烜**,本科生,主要研究方向为高性能计算.**陈跃国**,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为大数据系统和知识图谱.**翟季冬**,博士,副教授,主要研究方向为高性能计算和性能分析与优化.**杜小勇**,博士,教授,中国计算机学会(CCF)会士,主要研究领域为数据库系统和信息检索.

accelerators feature super computing power, high integration and fast network bandwidth. Take Ascend 910 as an example, it delivers 256T half precision FLOPS, 32 GB memory with 1200 GB/s bandwidth and 100G RoCE v2 network adapter. Compared with GPU, Ascend is mainly for neural networks. Differences between Ascend and GPU are: (1) Ascend uses task-specific processing units which is mainly for neural networks; (2) the computing power is based on lower precision; (3) the compiler software stack on Ascend is different from GPU. The main goal of deep learning is to train a statistical model based on train dataset and the fitted model should make high quality predictions on unseen data, which is referred to as generalization. From the perspective of hardware design, task-specific processing units can greatly speed up some particular workloads and lower precision enables faster training for a single iteration. However, task-specific processing units may not meet the need of a wide variety of deep learning models and lower precision hardware requires special software-level optimization methods. Previous benchmarks and analyses focused on deep learning with GPU platform. Ascend has its special and novel features and its potential remains unknown. To thoroughly understand its performance and optimization method, we conduct a systematic evaluation on Huawei Ascend and analyze optimization methods for faster training. Our contributions include: (1) we compare the performance results between Ascend and GPU on four end-to-end neural networks (ResNet, Transformer, DeepFM and LSTM) on well-known public datasets; (2) we analyze optimization methods on Ascend including deep learning framework developing, operator tile strategy and mixed precision training; (3) we measure hardware utilization and memory access pattern on three compute-intensive operators (fully-connected, convolution and RNN). To the best of our knowledge, we are the first to conduct comprehensive analysis on Ascend. Ascend is suitable for dense neural network workloads, its power consumption is lower than GPU when training and neural networks should be quantized from 32-bit to 16-bit precision. Based on the characteristics of the architecture and compiler software stack, to achieve better performance, we propose the following optimization strategies: when developing deep learning frameworks, we should compile the whole computation graph of neural network models so that operators can be fused. When developing operators, we should configure the tile size carefully with lower precision. When training models, we should adopt mixed precision configurations within a reasonable range. Ascend is not suitable for sparse workload. There are some internal errors when allocating extreme big size memory.

Keywords　deep learning; accelerator; Huawei Ascend; HPC; benchmark

1　引　言

近年来，深度学习取得突破性进展很大程度上得益于芯片和算力的支撑. 英伟达的图形处理器(Graphics Processing Unit，GPU)凭借其高性能和通用性的特点，一直在深度学习训练任务中占据主要地位，主流的深度学习工具如 TensorFlow[1]、PyTorch[2]等都会基于英伟达 GPU 产品进行研发. 为了获取更高的性能，学术界和工业界开始将研究重点从通用计算架构(General Purpose Architecture)转向领域专用架构(Domain Specific Architecture，DSA)[3]. 各厂商正积极研发神经网络加速器(Neural Network Accelerator)，以及基于神经网络加速器的编译器、库、优化工具和上层软件：Google 推出了张量处理器(Tensor Processing Unit，TPU)并将其大量部署在云数据中心[4]，Google 开源的深度学习框架 TensorFlow 可以充分利用 TPU 的算力；MIT 提出了一种高能效、可重配置的神经网络加速器 Eyeriss[5]，可以加速卷积神经网络的训练；寒武纪基于 DianNao 系列指令集[6]推出深度学习加速器.

除了以上提到的神经网络加速器，华为发布了昇腾(Ascend)系列加速器，包括面向训练场景的昇腾 910 和面向推理场景的昇腾 310. 昇腾加速器具

有高算力、高集成度和高速网络互连特性.以单张昇腾 910 加速器为例,它提供了峰值达 256T FLOPS(Floating Point Operations per Second)半精度浮点计算能力、32GB 高带宽内存(High Bandwidth Memory,HBM)、1200GB/s 内存带宽、多加速器间 100G RoCE v2 高速网络.相较于 GPU,昇腾作为一款新型神经网络加速器,有如下特点:(1)芯片架构专门面向神经网络计算,设置了多个专用计算单元;(2)以较低精度换取较高算力;(3)所提供的编译软件栈有别于 GPU.

神经网络训练任务的最终目的是在已有训练数据基础上构建一个统计模型,拟合后的模型在新数据上具有良好的泛化能力.从加速器硬件设计的角度,较低精度使得单次迭代的速度更快,但有可能导致模型无法拟合到一个较好的结果,或者使得迭代次数增多[7].现有的研究主要专注于 GPU 上的低精度训练[8]和性能分析[9-10],昇腾的芯片架构、加速库和深度学习框架都区别于 GPU,其实际表现有待进一步研究.例如,昇腾的低精度算力能否满足不同业务场景下的模型拟合要求,软硬件栈在各类任务上的性能如何等.

为弥补昇腾相关研究的空白,探索昇腾上的优化技巧,本文对昇腾加速器进行了系统性地分析,并重点研究了昇腾上的加速优化方法.本文的工作主要包括:(1)分别在昇腾和 GPU 上,对四个具有高度代表性的端到端神经网络(ResNet[11]、Transformer[12]、DeepFM[13]、LSTM[14])在准确度和吞吐量进行了对比和分析,这四个端到端模型涵盖了图像分类、机器翻译、点击率预估和情感分析应用场景;(2)重点研究了深度学习框架在昇腾软件栈上的适配开发,算子开发的分块策略和精度选择,混合精度设置等与昇腾体系结构和软件栈密切相关的优化方法;(3)分别在昇腾和 GPU 上,对三种计算密集型算子(全连接、卷积、RNN(Recurrent Neural Network))进行了浮点计算能力、设备利用率、访存分析.

本文是业内首次对昇腾处理器进行深度评测的文章,主要结论如下:

(1)华为昇腾加速器适合进行稠密型神经网络工作负载.使用昇腾进行模型训练,需要将神经网络模型从 32 位精度量化到 16 位精度.

(2)为充分利用硬件算力,在昇腾上进行深度学习框架开发时应进行整图编译构建,进行算子融合;进行算子开发时应合理设置分块大小,尽量使用低精度;模型训练时应使用混合精度训练.

(3)昇腾不适合稀疏型工作负载;昇腾软件栈在极大内存条件下出现内存分配错误,仍需优化;昇腾在空载时功耗较高.

本文的第 2 节将简述深度神经网络训练的基本原理、已有评测基准和昇腾加速器架构、软件栈及深度学习框架;第 3 节将介绍本项研究的动机和挑战;第 4 节从总体思路、性能指标、端到端模型、单算子四个层面介绍评测方法;第 5 节对性能数据进行分析;第 6 节介绍昇腾加速器上加速模型训练和保证准确性的优化方法;第 7 节介绍本文发现;第 8 节介绍相关工作;第 9 节对全文进行总结.

2 研究背景

2.1 深度学习训练

模型泛化能力. 深度学习的主要目的是在已有训练数据集上训练模型,使用该模型在未知数据上进行高质量的预测,即模型必须有泛化能力[15].为了量化模型在未知数据上的预测能力,模型训练时通常需要使用一个验证数据集来验证模型的预测能力,验证数据集通常区别于训练数据集.当模型在验证集上的效果表现良好时,会停止迭代.

典型训练过程. 训练数据集的特征经过神经网络后,使用损失函数与标签比对,得到损失值;训练时需要选择一种最优化算法,通过多轮迭代,优化损失值.训练过程中的最优化算法一般基于随机梯度下降(Stochastic Gradient Descent,SGD)[16]或其变体,如 Adam[17].限于硬件内存大小,优化算法每次只使用训练数据集的一小批次数据,该批次的数据量大小被称为 Batch Size.一次完整的优化迭代包括前向和反向两步:一批次的训练数据先经过神经网络前向得到损失值,根据损失值和链式求导法则反向计算网络中各个参数的梯度,然后基于梯度更新参数[18].遍历整个训练集,所有数据均参与迭代的过程被称为 Epoch.

模型与算子. 在特定的业务场景上,深度学习需要构建某种特定的神经网络.神经网络通常由多个算子组成,常见的算子包括全连接、卷积、池化、归一化、激活函数等[15].例如,用于图像分类的卷积神经网络通常由卷积、池化、全连接、归一化和激活函数等算子组成.

2.2 神经网络加速器硬件设计

多种计算单元. 不同算子的数学计算有一定差异,也对芯片的设计提出了一定的要求.全连接和卷

积算子会大量使用乘累加(Multiply-Accumulate, MAC)[19]，乘累加可以通过特定的电路设计将计算并行化，神经网络加速器会设计专用的 MAC 计算单元以加速乘累加. MAC 计算单元可以成倍地加速乘累加，但 MAC 计算单元的缺点是专用性强，只能进行乘累加. 神经网络还有很多其他非 MAC 计算，例如，RNN 除了使用乘累加，还需要使用非线性的 tanh 或 ReLU 激活函数[15]. 因此，为了加速器的灵活性，神经网络加速器还会添加一些非 MAC 计算单元，以适配不同的数学计算.

速度与泛化之间的平衡. 越来越多的训练数据和越来越大的神经网络模型对加速器的算力和存储提出了更高的要求，使用更低的精度可以适应这些要求. 当前神经网络训练常用的精度格式有 32 位浮点(FP32，又称为单精度)和 16 位浮点(FP16，又称为半精度). 芯片在低精度下可以节省更多存储空间，获得更快的访问速度；同时，在设计芯片时，低精度意味着单位面积上晶体管数量更多，单位时间内的可访问的操作数更多. FP16 可以提供更快的训练速度，但也有一定负面影响，需要使用额外的技术或更多的迭代次数来保证模型的泛化能力，具体如下：

(1) FP16 的数值空间更窄，容易发生数值上下溢出的问题. 直接使用 FP16 进行训练会导致模型无法拟合，必须使用一定技巧保证模型的数值准确性，例如混合精度训练[8]：使用 FP16 完成绝大多数计算密集型计算，使用 FP32 完成精度敏感型计算，包括对重要权重参数进行 FP32 备份、损失放大(Loss Scale)、一些特殊层仍使用 FP32 计算等.

(2) FP16 精度还可以在加速器内存有限的情况下使用更大的 Batch Size 进行训练，增大 Batch Size 可提升硬件的利用率，加快单次迭代速度，但也会降低单次迭代的准确性[20].

2.3　神经网络训练评测基准

评测基准(Benchmark)可以有效评测计算机软硬件的性能特征，评测基准一方面可以比较不同软硬件系统，另一方面可以评价同一个系统不同版本的优化改进情况. 传统的高性能计算通常使用 LINPACK 衡量软硬件系统的数值计算性能[21]，但在神经网络训练领域设计一个各方面都十分均衡的评测基准有一定挑战，主要因为：(1) 神经网络训练任务需要考虑准确性(泛化能力)、运行时间、吞吐量、内存带宽、功耗等不同指标，单纯的数值计算速度无法全面衡量加速器的性能；(2) 神经网络训练任务的技术栈复杂，涉及到硬件、编译器、库、深度学习框架和上层应用；(3) 神经网络模型正在飞速迭代，神经网络训练中所使用的超参数设置、权重初始化、优化器、混合精度、多卡并行等技巧也在不断演进. 目前，业内还没有公认的评测标准. 现有的评测基准集主要分为两大类：基于公开数据集和端到端模型的评测基准集，如 MLPerf[22]、DAWNBench[23]；针对某些计算密集型的基础算子的微评测基准集，比如 DeepBench①.

基于公开数据集和端到端模型的评测基准集通常会选择多个神经网络模型，在公开数据集上进行端到端的训练，通过比较准确度、运行时间、吞吐量等指标来衡量软硬件的性能. 这类评测基准集需要选择当前流行的神经网络模型，使用深度学习框架进行开发. 随着深度学习社区飞速发展，新的模型提出后，当前已有的模型可能逐渐废弃，因此整个评测基准集需要参与者长期维护. 不同加速器会对特定类型的模型加速，在提交评测结果时，加速器厂商往往选择对自身有利的部分，刻意忽略不擅长的模型. 此外，单纯从端到端的评测中，评测人员难以深入洞察硬件与深度学习软件栈之间的问题.

针对特定基础算子的微评测基准集通常选取矩阵乘法、卷积等算子，使用硬件厂商提供的底层库和编译工具实现这些算子，对运行时间进行性能评价. 例如，DeepBench 使用英伟达的 cuDNN、英特尔的 MKL 来实现各个基础算子. 这类评测基准集可以有效评估算子在特定硬件上的表现，但无法对整个模型进行评估，算子与算子之间的依赖以及整个模型层面的优化非常影响神经网络训练性能.

2.4　GPU

GPU 最初为图像处理而设计，因其拥有较多计算核心，研究人员常用 GPU 进行通用的并行计算，用以加速各类计算密集型任务. 这类可进行通用计算的 GPU 也被称为 GPGPU(General Purpose GPU). 神经网络包含大量的乘累加计算，可利用 GPU 提供的众多计算核心进行加速.

在业界各类 GPU 产品中，英伟达的软硬件生态较为完善. 英伟达提供了 CUDA(Compute Unified Device Architecture)平台，研发人员可基于 CUDA 平台开发并行计算程序；此外，英伟达提供了面向神经网络的 cuBLAS 和 cuDNN 库. 主流的深度学习框架会基于英伟达提供的软硬件进行优化. 例如，在

① DeepBench. https://github.com/baidu-research/DeepBench, 2021.2.16

目前英伟达的产品线中，Tesla V100 是神经网络训练任务中经常使用的一款加速器[23]. Tesla V100 基于英伟达公司的 Volta 架构，主要有两类计算单元：CUDA Core 和 Tensor Core. CUDA Core 可进行浮点和整数计算，V100 的 FP32 CUDA Core 算力为 15.7 T FLOPS. Tensor Core 专为乘累加设计，在 FP16 和 FP32 混合精度下，一个时钟周期内完成一次 4×4 矩阵计算，其算力为 125T FLOPS. 对于无法使用 Tensor Core 完成的计算任务，CUDA Core 是后备选项. 为更好地使用混合精度训练，英伟达协助深度学习框架进行了适配. 例如针对 PyTorch 提供了插件，研发人员可由 FP32 模式快速切换至混合精度模式.

2.5 华为昇腾

昇腾系列加速器是华为公司推出的一款面向神经网络加速的 DSA 硬件，被称为 Neural Network Processing Unit(NPU). 昇腾加速器是一个片上系统(System on Chip)，共有两类 AI 计算引擎：AI Core 和 AI CPU，其中 AI Core 提供神经网络算力，AI CPU 用于承担非矩阵类计算.

图 1 为 AI Core 所基于的华为自研 DaVinci 架构. AI Core 有三类计算单元，三类计算单元分别支持不同类型的深度学习计算任务：专门负责乘累加的矩阵计算单元(Cube Unit)，负责激活函数等非乘累加计算的向量计算单元(Vector Unit)和高度灵活的标量计算单元(Scalar Unit)[24]. 矩阵计算单元使用 FP16 精度，算力强大，但只能完成乘累加计算. 相比矩阵计算单元，向量计算单元的指令更加丰富，可以完成激活函数等非乘累加类计算. 标量计算单元主要完成循环控制、分支判断等标量计算.

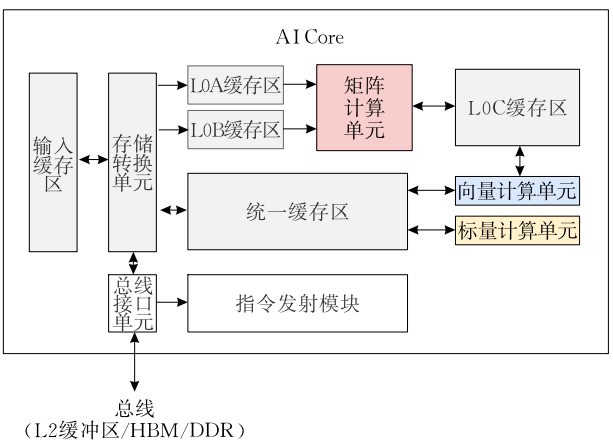


图 1 AI Core 架构示意图

在数据传输方面，图 1 的箭头表示数据传输的方向. 其中，AI Core 通过总线接口单元从核外存储系统中读取或写回数据，存储转换单元可将输入数据转换成 AI Core 中计算单元所兼容的数据格式，各个缓存区暂存了各种形式的中间数据[24]. 以全连接和激活函数为例，在最优调度的前提下，数据首先经总线从核外系统进入存储转换单元和输入缓存区，再经 L0A 和 L0B 缓存区进入矩阵计算单元，计算结果经 L0C 缓存区后，输入到向量计算单元，执行激活函数计算，最后经过统一缓存区，将数据写入核外存储系统. 值得注意的是，AI Core 中的所有数据如果需要向外传输，都必须经过统一缓存区，才能够被写回核外存储系统.

以上提到的计算、数据传输和调度需由昇腾上的 CANN(Compute Architecture for Neural Networks)软件栈控制，CANN 允许开发者在昇腾硬件上进行软件开发. 在深度学习框架层面，华为开源了 MindSpore 编程框架，MindSpore 支持华为昇腾和英伟达 GPU；同时，华为也基于 CANN 对 TensorFlow 和 PyTorch 进行了适配，使得 TensorFlow 和 PyTorch 用户在不改变使用习惯的前提下，也能在昇腾上训练神经网络.

2.6 编译软件栈与深度学习框架

深度学习框架是人工智能领域重要的高性能计算软件，它对底层软硬件栈进行了封装，为深度学习算法研发人员提供简单易用的编程接口. 当前，几乎所有神经网络模型的训练与推理任务均基于深度学习框架来完成[23].

深度学习框架需要定义张量和算子，张量用于存储计算过程中的数据，算子用于定义各类数学计算[1-2]. 在 GPU 平台上，开发者想在深度学习框架中实现一个算子，一般可以：调用 cuBLAS、cuDNN 等英伟达封装好的库函数；或者编写 CUDA 内核函数[1-2]. 华为昇腾的 CANN 软件栈与 GPU 有所区别. 在算子层面，CANN 内置了常用算子，可直接调用，类似 GPU 上的 cuBLAS 和 cuDNN；如果开发者希望自定义算子，需要使用 CANN 中的 TBE(Tensor Boost Engine). TBE 是一种基于 TVM[25] 的 Python API，开发者在 TBE 中完成算子数学逻辑的开发，使用自动或手动的方式进行调度. 在深度学习框架适配开发时，开发者需要使用 Ascend CL(Ascend Computing Language)来管理设备、上下文与内存，定义张量，将深度学习框架上层 API 定义的计算图编译为昇腾上的可执行文件. 由于基于 TBE 自定义的算子仅为数学计算和调度逻辑，在每次训练或推理时，深度学习框架需要调用 CANN 软件栈中的编译器，将算子或计算图编译为昇腾上的可执行文件.

在 TBE 中，有两种算子开发方式：DSL(Domain Specific Language)和 TIK(Tensor Iterator Kernel). DSL 与 TVM 较为相似，各类接口已经高度封装，使用预置的自动调度进行数据搬运. TIK 更接近底层硬件架构，提供了更精细的缓存管理、数据搬运、过程优化等接口. 某些场景下，TIK 相比 DSL 能取得更优性能，支持更复杂的算子逻辑.

3　研究动机与挑战

3.1　主要动机

本文对华为昇腾加速器进行了系统性地评测和分析，主要研究动机如下：

第一，昇腾实际表现有待探索. 根据公开披露的性能指标①，华为昇腾提供了较强的算力，然而指标无法代表在实际场景下的表现. 尽管已有研究机构在 MLPerf 上提交了昇腾在图像分类场景下的表现，但昇腾在不同业务场景下端到端模型训练性能和功耗、不同深度学习框架的适配方式以及其与 GPU 平台的对比仍然处于空白研究阶段.

第二，新硬件对模型训练提出了新要求. 昇腾采用了自主设计的芯片体系结构、指令集和软件栈，在较低精度下可达到较高算力. 如何基于昇腾的体系结构，合理调用软件栈使得充分发挥昇腾的算力仍需探索. 低精度对模型训练引入了更多新问题，包括如何设置 Loss Scale 才能保证模型拟合、混合精度能否适配不同业务场景、哪些算子设置为低精度等.

第三，基准评测可有助于软硬件栈后续优化. 昇腾生态目前仍处于起步发展阶段，其软硬件栈仍有较多可优化空间. 基准评测可深度挖掘加速器软硬件栈之间的性能瓶颈，帮助厂商有针对性地调优. 已有的评测基准集或者关注训练时间，无法深刻洞察性能瓶颈；或者使用底层语言，实现难度较大.

3.2　挑战

为了系统性地评测分析昇腾加速器，本文设计了跨平台评测基准集，专门用来对加速器进行性能分析，寻找优化空间. 在设计评测基准集并寻找优化空间的过程中，仍然面临诸多挑战，主要有三点：

第一，评测分析工作要能够挖掘硬件特性. 昇腾加速器有区别于 GPU 的体系结构和软件栈，且主要基于低精度. 评测和分析工作要深入理解昇腾的体系结构和软件栈. 评测结果能够给加速器使用者提供使用建议，包括：架构师和算法研发人员针对工作负载场景选择何种加速器硬件、深度学习框架开发者如何基于编译软件栈进行适配开发、如何利用昇腾的低精度开发算子等.

第二，跨硬件平台的评测工作实现难度大. 已有的模型训练和评测工作主要集中在 GPU 平台上，昇腾平台的软硬件栈仍处在完善阶段，设计跨硬件平台的评测基准既需要保证不同硬件平台上各类训练参数一致，又需要保证评测基准能够发挥出硬件平台本身的算力性能. 从深度学习框架的角度来看，不同深度学习框架的使用方式和算法实现也各有差异. 例如，同样的算子，GPU 和昇腾上的实现方式有所区别；同样的数据集，不同深度学习框架对数据集的加载方式有一定差异.

第三，评测的工作负载需要精心选择. 评测工作负载中需要包括端到端模型训练，以代表真实的训练场景. 评测基准中需要考虑单算子的性能表现. 评测基准中要覆盖不同的评价指标，包括准确度、训练时间、功耗、硬件性能利用率、访存、带宽等. 基于所选择的工作负载，评测基准要从不同的维度(不同计算类型、不同训练数据、评测指标)对加速器进行广泛测试和分析.

4　评测方法

4.1　总体思路

根据第 3 节所提出的研究动机与挑战，本文对昇腾处理器进行了系统的评测分析，重点研究了昇腾处理器上的加速优化方法，整个评测思路如图 2 所示.

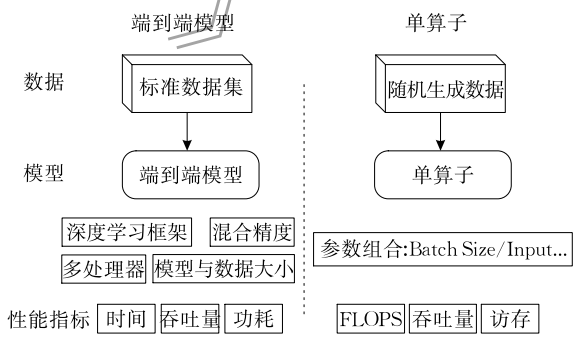


图 2　评测总体思路

为应对跨硬件平台问题，对工作负载进行评测时，本文使用相同的深度学习框架、相同的超参数. 本文选择了已经对 GPU 和昇腾加速器均适配过的 TensorFlow、PyTorch 和 MindSpore 深度学习框架，相同的深度学习框架在数据集加载、CPU 与加

① 昇腾社区官网. https://www.hiascend.com/hardware/accelerator-card, 2021,5,16

速器间数据传输方式、优化器和学习率等实现上一致.为应对深度学习训练随机性大的问题,本文选择较大的数据集,因为较大的数据集更具有说服力.从工作负载的选取角度,本文分别选择了端到端模型和单算子两类不同类型的负载.端到端模型可测试加速器实际的拟合和泛化能力,单算子可挖掘软硬件之间的关系.本文也从不同维度对端到端和单算子工作负载进行了分析:端到端模型从不同的深度学习框架、不同训练数据等维度进行了测试;单算子从不同 Batch Size 等组成的参数组合进行了分析.针对昇腾处理器低精度特点,本文重点研究了不同混合精度设置和不同 Batch Size 下的性能表现.通过收集每次训练的时间、吞吐量、FLOPS 等性能指标数据,对昇腾处理器有了更为全面的认识后,本文可为昇腾使用者提供使用建议.

4.2 性能指标

为横向评测不同神经网络加速器在指定神经网络模型上的性能,需要使用通用的性能指标进行对比,常用指标包括达到准确度阈值的时间、吞吐量、FLOPS 等.

4.2.1 达到准确度阈值的时间

神经网络训练任务主要目的在于使模型达到某个准确度阈值,对于一款性能表现未知的加速器,其首要目的是解决端到端训练的模型拟合问题.部分评测基准集关注单轮迭代的速度或时间,这种评价指标较为片面.例如,增加 Batch Size 可充分利用加速器的算力,加快单轮迭代速度,但较大的 Batch Size 会导致模型收敛速度变慢.本文认为,达到验证集准确度阈值的时间能够综合衡量软硬件系统在训练任务上的性能.

不同应用场景下的准确度评价标准不同.本文进行图像分类、机器翻译、点击率预估和情感分析四类任务的评测,其中,图像分类使用预测准确度 TOP1 作为评价指标,机器翻译使用 BLEU[26] 作为评价指标,点击率预估使用 AUC[27] 作为评价指标,情感分析使用分类准确度作为评价指标,本文在每轮 Epoch 训练结束后使用验证数据集对模型进行准确度计算.

4.2.2 吞吐量

吞吐量,即每秒处理的样本数,是另一种经常使用的端到端训练性能评价指标.在验证集准确度不容易快速获取的情况下,吞吐量可以反映软硬件系统的性能.

在图像分类场景下,吞吐量一般为每秒处理的图片数(图片/s);在机器翻译场景下,吞吐量一般为每秒处理的句子对(句子对/s);在点击率预估和情感分析场景下,吞吐量一般为每秒处理的样本数(样本/s).

4.2.3 FLOPS

FLOPS 常用来从硬件层面衡量芯片单位时间内所完成的浮点计算数.芯片厂商通常会基于芯片的额定频率和核数计算 FLOPS 理论峰值.在实际应用场景中,受限于芯片的访存系统和应用的算法设计,芯片很难达到 FLOPS 理论峰值.为获取某个计算任务在某种芯片上的 FLOPS,需要首先获得该计算任务的 FLOPs,即该计算任务的浮点操作数;还需要以秒为单位记录该计算任务的运行时间 time,那么 $\text{FLOPS}=\frac{\text{FLOPs}}{\text{time}}$.一个神经网络模型是一种由多个算子组成的计算图,先得到计算图中每个算子的 FLOPs,再递归地遍历整个计算图,将所有算子的 FLOPs 聚合汇总,即可得到整个计算图的 FLOPs.

现有的深度学习框架对 FLOPs 的支持并不完善.TensorFlow 虽然能计算常见算子的 FLOPs,但 RNN 类模型(包括 RNN、GRU、LSTM)等算子的 FLOPs 计算有误.PyTorch 在 1.8 版本前不提供 FLOPs 相关接口,1.8 版本也仅针对乘累加和卷积算子提供了接口.本文系统性地对常用算子进行了 FLOPs 估计,所覆盖的算子包括全连接、卷积、池化、循环神经网络、激活函数(ReLU 等)、丢弃.例如,按照卷积运算的定义,一个二维卷积算子 Conv2d 的 FLOPs 如下:

$$FLOPs=C_{\text{in}}\times K^2\times H\times W\times C_{\text{out}},$$

其中,C_{in} 为输入通道数,K 为卷积核大小,H 和 W 为输出特征图的高和宽,C_{out} 为输出通道数.

4.3 端到端模型的性能分析

使用端到端模型进行分析,可以验证深度学习框架与神经网络加速器的适配是否完善.本文使用了图像分类、机器翻译、点击率预估和情感分析四种应用场景进行端到端模型的分析,所使用的模型、数据集和准确度阈值情况如表 1 所示.

表 1 端到端模型工作负载概览

模型	应用场景	数据集	准确度阈值
ResNet[11]	图像分类	CIFAR-10[28]	90.0%
		ImageNet 2012[29]	75.9%
Transformer[12]	机器翻译	WMT16 en-de[30]	BLEU 25.0
DeepFM[13]	推荐系统	Criteo①	AUC 0.8075
LSTM[14]	情感分析	IMDB[31]	83.5%

① Display Advertising Challenge. https://www.kaggle.com/c/criteo-display-ad-challenge/, 2021,8,20

对比单算子，影响端到端模型训练速度的超参数因素较多，包括混合精度、优化器等. 本文尽量保证同一个工作负载在不同硬件平台上的深度学习框架和超参数设置保持一致，以避免这些因素所带来的偏差和干扰. 例如，在图像分类的工作负载上，本文基于 MindSpore 框架，设计相同的网络模型，使用同样的训练数据在 GPU 和昇腾硬件平台进行训练. 同时，针对同一工作负载，本文使用相同的优化器、参数初始化和学习率调整策略等.

由于神经网络加速器趋于选择较低的精度以换取更快的速度，混合精度训练成为主流，而 GPU 提供的精度选择更多，包括 FP64、FP32 和 FP16. 本文尽量使用 FP32 与 FP16 组成的混合精度模式对模型进行训练，同时也研究了混合精度下不同设置对准确度和训练时间的影响.

4.3.1　图像分类

图像分类是评价神经网络加速器最常用的任务之一. 给定一张图像和一个神经网络模型，模型会预测该图像最可能的类别. 图像分类模型也经常作为各类其他计算机视觉任务（如目标检测）的特征抽取器.

本文的图像分类任务使用 ResNet[11] 模型. ResNet 模型由多个残差块组成，每个残差块都使用了卷积、批量归一化和 ReLU 等算子. ResNet 允许设置残差块的数量，不同数量的残差块决定了 ResNet 的总层数不同，针对 ImageNet[29] 224×224 尺寸的输入，常见的模型有 ResNet50、ResNet101；针对 CIFAR-10[28] 32×32 尺寸的输入，常见的模型有 ResNet56.

4.3.2　机器翻译

机器翻译可以将源语言文本序列翻译成目标语言文本序列. 基于神经网络的机器翻译经常使用编码器-解码器（Encoder-Decoder）结构，编码器用来分析源语言文本序列，解码器用来生成目标语言文本序列. 本文主要实现了 Transformer 模型[12]，基于 WMT16 英语-德语（WMT16 en-de）翻译数据集[30]进行了评测. Transformer 抛弃了 RNN 类算子，主要使用了注意力层，它的 Encoder 和 Decoder 均包含 6 个基础块，每个基础块均包含了注意力层，注意力层主要由全连接和 Softmax 等算子组成.

4.3.3　点击率预估

点击率预估算法被广泛应用在互联网搜索、推荐和广告服务中. 点击率预估算法以用户的基本信息和各类偏好为特征，对某项内容是否会被用户点击进行预估. 本文使用 DeepFM 模型[13]进行评测. DeepFM 由三类计算构成：稀疏全连接层、因子分解机和稠密全连接层.

4.3.4　情感分析

情感分析是对带有主观情感色彩的文本进行分析和归类. 常用于互联网平台用户生成数据的分析，如口碑分析等. 本文使用面向情感分类的 LSTM 模型[14]对 IMDB 电影评论数据集[31]进行了情感分析，训练好的模型可以预测一条电影评论的褒贬口碑.

LSTM 是 RNN 的一种变体，它可以捕捉一段序列中的依赖关系. 尽管在部分应用场景下，RNN 类模型的准确度已经被其他预训练模型所超越，本文仍将 LSTM 模型作为工作负载. 因为相比预训练模型，LSTM 类模型计算复杂度更低，训练和部署成本较小，很多任务仍然基于 RNN 和其变体，有必要对 RNN 结构进行评测. 本文实现的 LSTM 模型中，主要使用了词嵌入、LSTM 和全连接等算子.

4.3.5　工作负载的选择

在工作负载和数据集的选择上，本文也进行了精心考虑：在应用场景层面，所选择的图像分类、机器翻译、点击率预估、情感分析四类应用场景，分别代表计算机视觉类、推荐系统类和自然语言处理类任务；在计算模式层面，ResNet 和 Transformer 为稠密计算场景，DeepFM 计算相对稀疏，LSTM 后序时间步的输入依赖前序时间步，四个模型涵盖了神经网络经常使用的算子.

针对图像分类任务，ResNet 是图像分类领域业界领先的模型之一，也是各类基准评测中经常使用的模型. 很多计算机视觉类研究会在 ResNet 的基础上进行改进，各类目标检测等任务也会选择 ResNet 作为特征抽取器. 因此，本文认为 ResNet 非常适合作为图像分类任务的基准. 同时，本文选择 CIFAR-10 和 ImageNet 2012 作为数据集，两个数据集代表了不同数据规模.

自然语言处理任务涵盖的场景丰富，模型多样，当前主流的自然语言处理任务的基础算子都会基于注意力机制或 RNN. 注意力机制在越来越多的自然语言处理和计算机视觉任务上流行，Transformer 可代表注意力机制类模型，LSTM 模型可代表 RNN 类模型.

点击率预估算法一般主要使用因子分解机和全连接构建神经网络，相比 ResNet 和 Transformer，点击率预估算法的模型层数较少. DeepFM 融合了稀疏全连接层、因子分解机和稠密全连接层，可代表

搜索、推荐和广告场景中的模型.

4.4 单算子性能分析

除了对端到端模型进行评测,本文也对三种常用的计算密集型算子进行了全面的评测,包括全连接、卷积和 RNN. 全连接是神经网络中最常使用的线性变换算子,卷积经常用于图像分类和文本处理等任务中,RNN 经常用于机器翻译、情感分析等任务中,以上三种算子为计算密集型,因此使用这三种算子进行单算子性能分析非常具有代表性.

评测过程中,本文使用参数化的评测方法,分析了算子在不同参数组合下的性能. 例如,卷积算子的常用输入参数有批大小(Batch Size)、输入特征图大小(Input Size)、输入通道数(Input Channels)、输出通道数(Output Channels)、卷积核大小(Filter Size). 表 2 展示了对卷积算子评测时使用的参数组合. 需要注意的是,本文中的参数组合均基于 8 的倍数,因为基于 8 的倍数,才能更好地利用昇腾 910 和 Tesla V100 的 MAC 计算单元.

表 2 卷积算子参数组合列表

	最小值	最大值	增量
Batch Size	128	512	×2
Input Size	112	336	×2
Input Channels	32	128	×2
Output Channels	64	256	×2
Filter Size	1	5	+2

本文按照参数组合中的 Batch Size、Input Size 等输入维度的大小,随机生成了算子的训练数据. 评测程序在初始化时会产生一定的时间开销,为避免初始化时间开销的影响,保证所收集到的运行时间数据的准确性,在正式进行性能评测前,先进行了预热,包括:将输入数据从主存拷贝到加速器上,进行 10 轮的迭代. 正式的性能评测使用了 100 次迭代. 迭代包括前向和反向两种场景. 每次运行结束后,收集了 FLOPS、模型参数量、内存访问量、吞吐量等性能指标. 使用参数化的评测方法可以测试算子在不同场景下的性能表现,为算子调优提供依据.

5 性能分析与比较

5.1 硬件详情

本文使用了一台搭载 8 块华为昇腾 910 的节点和搭载 2 块英伟达 Tesla V100 GPU 的节点进行测试. 华为昇腾 910 单张卡的 HBM 存储大小为 32 GB,带宽为 1200 GB/s,FP16 下的 FLOPS 理论峰值为 256T. 昇腾 910 节点配备的是 192 核心的华为鲲鹏 920 CPU,768 GB 主存,后文以 NPU 指代此平台. 英伟达 Tesla V100 单张卡的 HBM 存储大小为 32 GB,带宽为 900 GB/s,Tensor Core 的 FLOPS 理论峰值为 125T. Tesla V100 节点配备的是 64 核心的 Intel Xeon 5218 CPU,256 GB 主存,后文以 GPU 指代此平台. 两种加速器以及其所在的计算节点的性能参数如表 3 所示.

表 3 实验平台软硬件参数对比

	NPU	GPU
CPU	华为鲲鹏 920	Intel Xeon 5218
主存	768 GB	256 GB
外置存储	SSD	SSD
加速器型号	华为昇腾 910	英伟达 Tesla V100
加速器存储	32 GB 1200 GB/s	32 GB 900 GB/s
加速器 FLOPS	256 T	125 T
并行计算库	CANN 5.0.2	CUDA 10.1
深度学习框架	tensorflow-ascend 5.0.2 昇腾适配版	tensorflow-gpu v1.15
	pytorch-ascend 5.0.2 昇腾适配版	pytorch-gpu v1.5
	mindspore-ascend v1.2	mindspore-gpu v1.2

由于上述两台测试节点上的 CPU、主存等配置无法保证一致,这些因素都可能对端到端模型训练速度产生影响,本文在测试过程中也使用了一些方法尽量避免这些因素所带来的扰动,包括:(1)使用真实数据集进行端到端训练时,使用深度学习框架提供的缓存机制,尽量将训练数据缓存到内存;(2)使用随机生成数据进行测试时,先进行预热,在评测加速器性能前后调用同步函数,并使用加速器提供的事件接口监测运行是否结束.

5.2 端到端模型性能

5.2.1 计算速度分析

NPU 适合稠密计算场景. 对于卷积和稠密型矩阵乘计算较多的场景,NPU 相比 GPU 性能提升明显.

ResNet 和 Transformer 模型是两类稠密型计算典型代表,其中 ResNet 的核心计算为卷积、批量归一化和 ReLU 激活函数组成的残差块,Transformer 的核心计算为矩阵乘法和 Softmax 激活函数组成的多头注意力块.

对于 ResNet 模型,本文在 CIFAR-10[28] 和 ImageNet 2012[29] 数据集上进行了端到端的模型训练. CIFAR-10 数据集的图片尺寸为 32×32,训练集共 50000 张图片,验证集共 10000 张图片. ImageNet 2012 数据集的图片尺寸为 224×224,训练集共 1281167 张图片,验证集共 50000 张图片. 针对

ResNet 模型设计如下三种场景，测试不同输入尺寸和不同数据集大小下 NPU 的性能：在尺寸为 32×32 的 CIFAR-10 上使用 ResNet56 进行训练，准确度如图 3(a)所示；将 CIFAR-10 的图片尺寸扩充到 224×224，并使用 ResNet50 进行训练，准确度如图 3(b)所示；在尺寸为 224×224 的 ImageNet 2012 上使用 ResNet50 进行训练，准确度如图 3(c)所示. 图 3(a)至图 3(c)横轴均为时间，纵轴均为验证集上 TOP1 准确度. 在 32×32 尺寸下，以 90%的 TOP1 准确度为阈值，NPU 平台的性能为 GPU 平台的 2.70 倍. 当将图片扩充到 224×224，NPU 平台是 GPU 的 2.34 倍. 而使用数据量较大的 ImageNet 数据集进行训练时，以 75.9%(MLPerf v1.0 标准)的准确度为阈值，NPU 平台的性能是 GPU 的 2.89 倍.

对于 Transformer 模型，本文在 WMT16 en-de 数据集[30]上进行了端到端的模型训练. 如图 4(a)所示，达到 BLEU 为 25.0 的准确度阈值，NPU 是 GPU 的 1.67 倍.

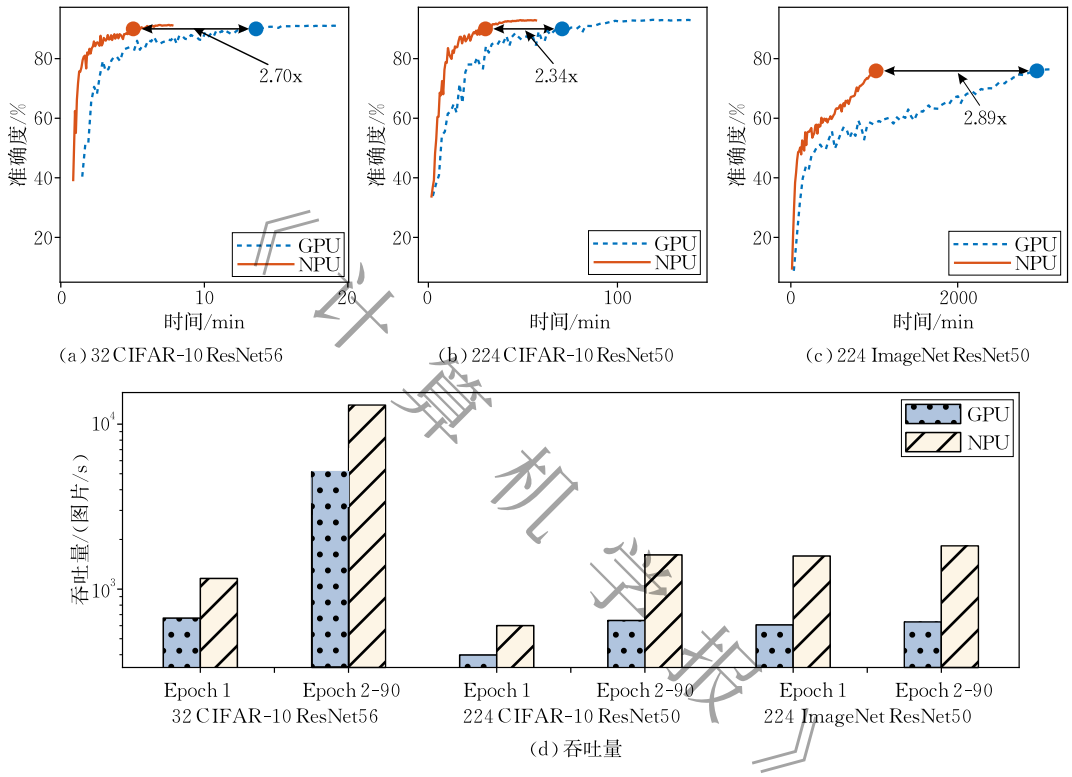


图 3　在 NPU 和 GPU 上训练 ResNet 模型时的性能表现((a)在 32×32 CIFAR-10 上训练 ResNet56；(b)在 224×224 CIFAR-10 上训练 ResNet50；(c)在 224×224 ImageNet 上训练 ResNet50；(d)吞吐量，包括不同场景下首轮 Epoch 与非首轮 Epoch 平均值)

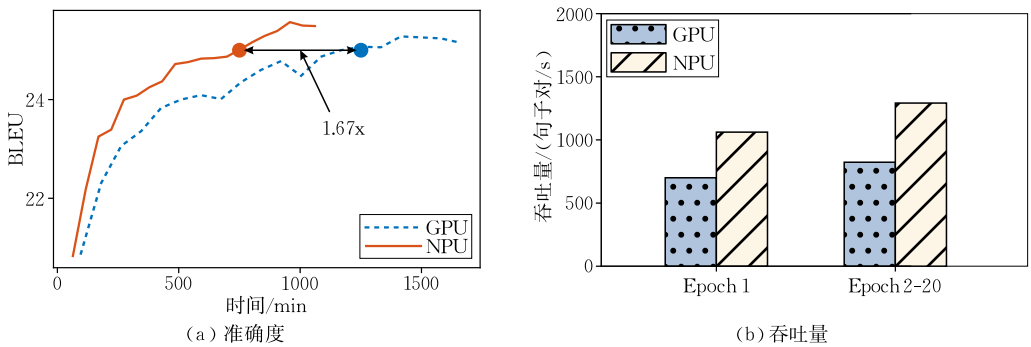


图 4　在 NPU 和 GPU 上训练 Transformer 模型时的性能表现(包括首轮 Epoch 与非首轮 Epoch 吞吐量平均值)

NPU 的初始化较长. NPU 需要在首轮 Epoch 创建上下文、申请内存、生成计算图以及将计算图编译为二进制可执行文件，初始化耗时较长.

本文进一步分析了 ResNet 三种场景下不同 Epoch 的吞吐量，如图 3(d)所示. 图 3(d)包含了每种训练场景下首轮 Epoch 吞吐量和非首轮 Epoch

的平均吞吐量.一次训练首轮 Epoch 要对数据集进行预处理,在正式进行模型训练前,对于 MindSpore 这样的基于静态图的深度学习框架需将上层接口转化生成计算图.对于 NPU 平台,要调用 CANN 软件栈,在 NPU 设备上创建上下文,申请内存、将深度学习框架所生成的计算图编译为二进制可执行文件.32×32 的 CIFAR-10 上,首轮 Epoch 的吞吐量仅为非首轮 Epoch 的 8.89%,主要因为,NPU 节点在首轮 Epoch 中花费较长时间初始化.将 CIFAR-10 扩充到 224×224 后,扩充后的 CIFAR-10 在 NPU 上首轮 Epoch 为非首轮 Epoch 吞吐量的 27.27%.ImageNet 上首轮 Epoch 是非首轮 Epoch 吞吐量的 81.44%.因为 ImageNet 样本数量远大于 CIFAR-10,模型训练的时间远大于初始化时间,初始化时间占比较低.

NPU 在稀疏计算场景下表现不及 GPU. 对于稀疏类矩阵乘和 RNN,由于 NPU 体系结构和 CANN 软件栈的特殊性,整体表现不及 GPU.

DeepFM 和 LSTM 是两类稀疏型计算典型代表.DeepFM 涉及三类计算中,线性全连接层和因子分解机属于稀疏型计算,稠密的全连接层虽然是稠密型计算,但层数较少.LSTM 等 RNN 类模型在一个 RNN 单元内进行线性矩阵乘和激活函数计算,后序时间步的输入依赖前序时间步的输出,在计算模式上有访存依赖,不易将计算并行化.

对于 DeepFM 模型,本文在 Criteo 数据集上进行了训练.如图 5(a)所示,达到 AUC 为 0.8075 的准确度阈值,NPU 速度为 GPU 的 73%;非首轮 Epoch 吞吐量,NPU 为 GPU 的 63.16%,忽略数据预处理和 NPU 初始化等因素,NPU 的性能不及 GPU.

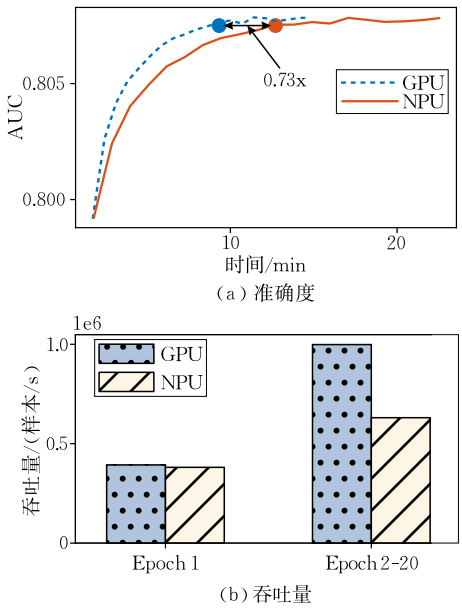


图 5 在 NPU 和 GPU 上训练 DeepFM 模型时的性能表现(包括首轮 Epoch 与非首轮 Epoch 平均值)

对于 LSTM 模型,本文在 IMDB 电影评论数据集[31]上进行了训练.如图 6(a)所示,达到准确度为 83.5%的准确度阈值,NPU 速度为 GPU 的 0.13.NPU 速度较慢的原因,主要由于:(1)首轮 Epoch 的吞吐量 NPU 仅为 GPU 的 7%,NPU 需要花较多时间进行 NPU 的初始化,尤其是计算图的编译,在当前的软硬件栈中,NPU 在时间步循环场景下的编译时间较长;(2)非首轮 Epoch 的吞吐量 63%,忽略数据预处理和 NPU 初始化等因素,NPU 的性能不及 GPU.

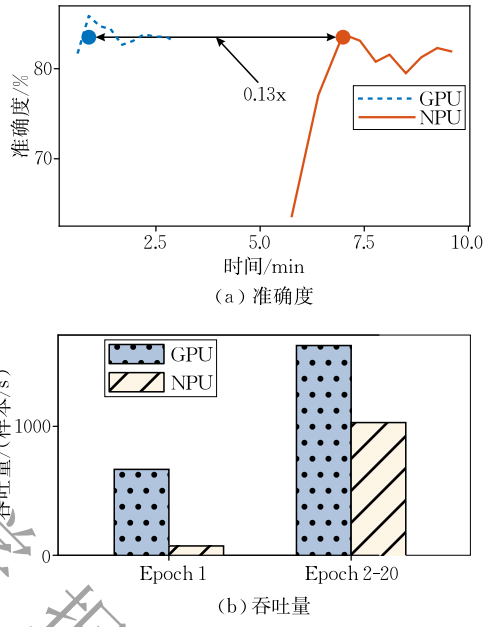
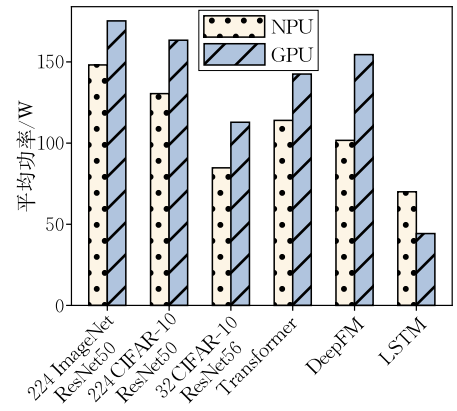


图 6 在 NPU 和 GPU 上训练 LSTM 模型时的性能表现(包括首轮 Epoch 与非首轮 Epoch 吞吐量平均值)

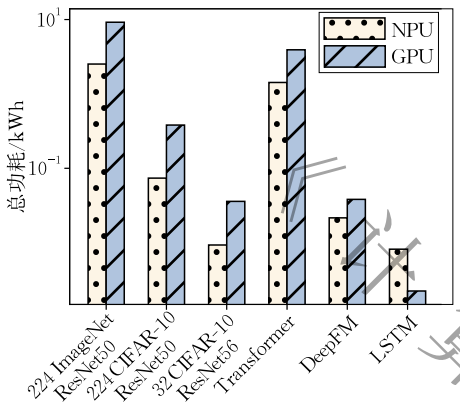
LSTM 模型结构中没有大量稠密型计算,IMDB 数据集相比 ImageNet 和 WMT16 en-de 较小,总体训练时间为分钟级.综合 IMDB 上的 LSTM 以及 CIFAR-10 上的 RestNet56,当数据集较小或模型计算量较小时,模型训练所需时间较短,NPU 上的初始化时间占整个端到端训练时间比重较高,端到端模型训练相比 GPU 没有速度优势.

5.2.2 功耗分析

根据硬件厂商提供的信息,NPU 的额定最大功率为 300 W,GPU 的额定最大功率为 250 W,在实际工作负载上,加速器不一定能达到最大功率.本文使用硬件厂商提供的系统管理接口收集了实际工作负载中加速器功率数据,计算出训练达到准确度阈值过程中的平均功率和总功耗,图 7(a)为不同工作负载的平均功率,图 7(b)为不同工作负载的总功耗.



(a) 平均功率



(b) 总功耗

图 7　模型训练达到准确度阈值的平均功率和总功耗

在稠密计算场景，NPU 的功耗比 GPU 低. 在 ResNet、Transformer 稠密计算型工作负载上，NPU 算力性能得到充分发挥，训练速度更快，总功耗相比 GPU 明显降低. 在 ResNet ImageNet 场景下，NPU 的平均功率为 148.2 W，是 GPU 的 84.5%；NPU 训练速度快、耗时短，总功耗为 2.52 kWh，是 GPU 的 27.4%. 在 Transformer WMT16 en-de 场景下，NPU 的平均功率为 114.0 W，是 GPU 的 80.0%，总功耗为 1.43 kWh，是 GPU 的 26.4%.

当无法发挥 NPU 算力时，NPU 的功耗比 GPU 高. 在无任何计算任务的空载情况下，NPU 平均功率为 67.5 W，而 GPU 的平均功率为 22.5 W，NPU 的空载平均功率是 GPU 的 3 倍. 这意味着，当模型无法充分利用 NPU 的算力时，NPU 由于其空载功率高，总功耗较高. 例如，在 LSTM 场景下，NPU 平均功耗为 70 W，是 GPU 的 1.58 倍，总功耗是 GPU 的 3.65 倍. NPU 空载功率高，主要因为单张 NPU 加速器内集成了 AI CPU、AI Core、HBM 和 RoCE 高速网络等多个单元.

5.3　单算子性能

5.3.1　内存分配

本文分别在 NPU 和 GPU 上使用了多组参数组合测试了全连接、卷积、RNN 三个算子的性能. 在部分参数组合下，由于算子申请的内存空间较大，会发生内存分配错误，导致运行不成功. 表 4 为三个算子在两个平台下的运行成功情况.

表 4　三个算子在不同参数组合下运行成功情况

	参数组合总数	NPU 成功次数	GPU 成功次数
全连接	192	180	192
卷积	243	155	207
RNN	144	122	144

以全连接算子为例，随着 Batch Size、Input 和 Ouput 变大，模型的输入和模型本身将占用越来越多的内存，超出一定上限，将引起内存分配错误. 同样的问题也发生在了卷积和 RNN 算子上，从表 4 中可以看到，GPU 运行成功的次数均高于 NPU，而本次评测使用的 GPU 和 NPU 的内存大小均为 32 GB. 可见，NPU 在极大内存情况下上容易发生内存分配错误. 因此，本文认为 NPU 上软件栈在处理较大内存上仍需完善.

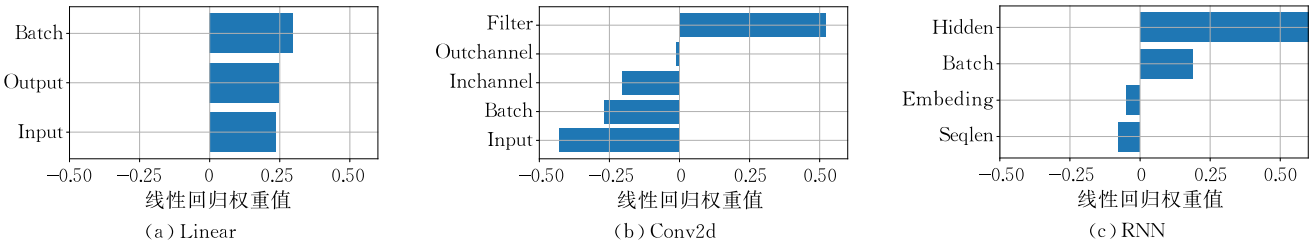
5.3.2　FLOPS

本文分析了三个算子在 NPU 上的 FLOPS 利用情况. 对于全连接算子，FLOPS 实测最高可达 210.17 T，FLOPS 利用率为 82%. 对于卷积算子，FLOPS 实测最高可达 216.96 T，FLOPS 利用率为 84.7%.

为了量化 FLOPS 实测值和参数组合之间的关系，本文进行了线性回归分析，用以发现哪些参数影响 FLOPS. 例如，对于全连接算子，线性回归共有 3 个参数 Input、Output 和 Batch Size，线性回归模型为

$$FLOPS = w_0 \times Input + w_1 \times Output + w_2 \times BatchSize.$$

w_0 至 w_2 是线性回归中的特征权重，可以反映特征对目标值 FLOPS 的影响程度. 如果 w_0 为正，表明该特征对目标值 FLOPS 有正影响. 图 8(a)展示了全连接算子各参数对 FLOPS 的影响，Input、Output 和 Batch Size 的权重均为正，即 Input、Output 和 Batch Size 越大，NPU 上全连接算子的 FLOPS 实测值越高. 图 8(c)展示了 RNN 各参数对 FLOPS 的影响，隐层大小(Hidden)和 Batch Size 的权重为正，对 FLOPS 有正向影响. Seqlen 为序列长度，由于 RNN 结构中后续时间步依赖前序时间步的计算结果，计算无法并行化，Seqlen 越长，越影响算力的发挥. 根据图 8 中分析，当算子本身更宽，训练数据的 Batch Size 更大时，算子可充分利用 NPU 的算力.

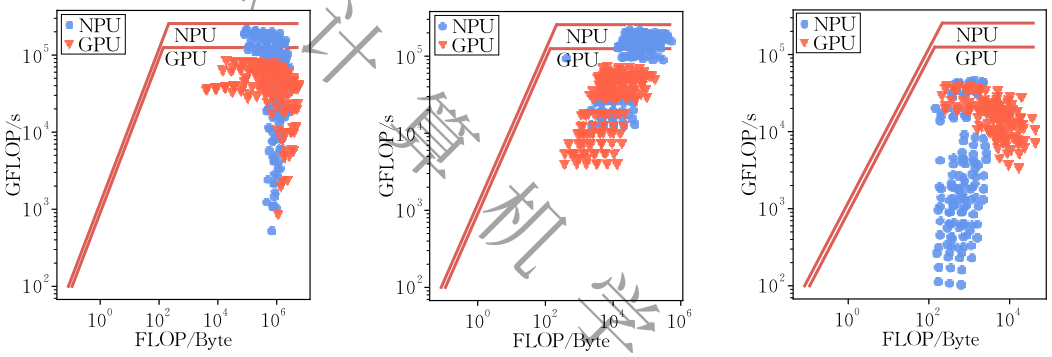


(a) Linear (b) Conv2d (c) RNN

图 8 NPU 单算子参数组合线性回归分析

5.3.3 访存

结合 5.3.2 节中的 FLOPS 分析，本文使用 Roofline 模型[32]分析了三个算子的访存特性，如图 9 所示. 图 9 显示，NPU 在全连接和卷积算子上几乎无访存瓶颈，但在 RNN 算子上，NPU 的 FLOP/Byte 低于 GPU，在访存瓶颈附近. NPU 上的 RNN 性能低于 GPU，主要因为 RNN 是访存密集型计算：单个 RNN 计算单元内先进行线性矩阵乘，其计算结果将输出到 AI Core 核外存储系统，激活函数计算还需再将矩阵乘的结果从核外存储系统中搬运进 AI Core 内，由向量计算单元执行；RNN 后续时间步依赖前序时间步的输出，后续时间步需等待前序时间步完成计算，后续时间步需将前序时间步的计算结果从 AI Core 核外存储再次搬运回 AI Core 内. 频繁的数据搬运导致 NPU 上的 RNN 性能低于 GPU.



(a) Linear (b) Conv2d (c) RNN

图 9 NPU 与 GPU 访存 Roofline 模型

5.3.4 NPU 与 GPU 对比

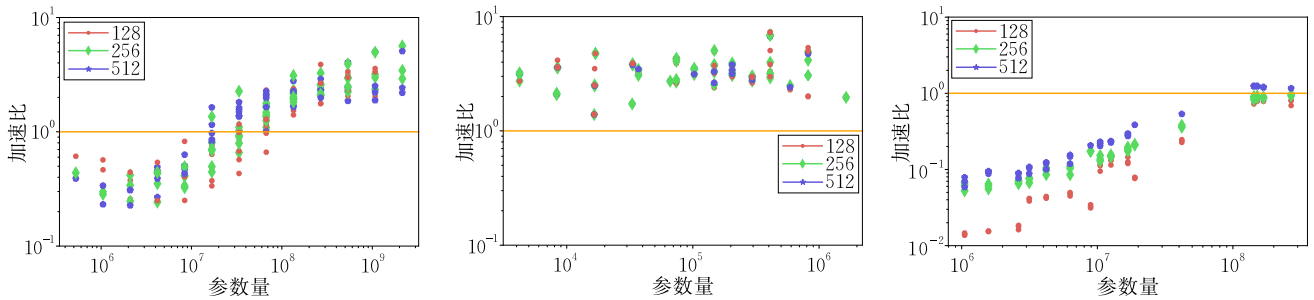
对于全连接、卷积和 RNN 三个算子，在不同参数组合下，本文分析了 NPU 和 GPU 之间的吞吐量比值. 图 10 为三个算子在相同参数组合下 NPU 与 GPU 的吞吐量比值，纵轴为两者比值，横线表示比值为 1，即吞吐量性能持平，横轴为算子权重参数量的大小，不同标记代表不同 Batch Size.

NPU 与 GPU 的吞吐量比值见表 5. 结合表 5 和图 10，可以发现：

(1) 对于全连接，当模型参数量较小时，NPU 的表现不如 GPU，当模型参数量较大时，NPU 的速度比 GPU 有较大提升.

(2) 对于卷积，NPU 在各种参数组合下均优于 GPU.

(3) 对于 RNN，由于 NPU 芯片体系结构和软件栈未进行 RNN 的访存优化，NPU 在大部分参数



(a) Linear (b) Conv2d (b) RNN

图 10 NPU 与 GPU 吞吐量实测比值(纵轴为比值，横轴为算子参数量，不同标记代表不同 Batch Size)

表 5　NPU 与 GPU 的吞吐量比值

	最大值	最小值
全连接	5.64	0.23
卷积	7.37	1.38
RNN	1.24	0.02

组合下的性能低于 GPU.

经过上述对比后发现，NPU 的矩阵计算单元更适合进行卷积计算. 对于全连接计算，在输入数据规模较大时才能充分利用矩阵计算单元的算力. 在有访存压力的 RNN 场景，NPU 仍需优化.

5.4　带宽性能

本文对 NPU 和 GPU 上主机与设备间、设备内数据传输数据时的带宽进行了分析. 图 11(a)为主机与设备间数据传输的性能对比，D2H(Device To Host)为设备端向主机端传输数据，H2D(Host To Device)为主机端向设备端传输数据，当传输数据大小较小时(小于 512 K)，GPU 带宽较高，当传输数据大小较大(大于 512 K)，NPU 的带宽较高. 图 11(b)为设备内输出传输的性能对比，GPU 性能均优于 NPU.

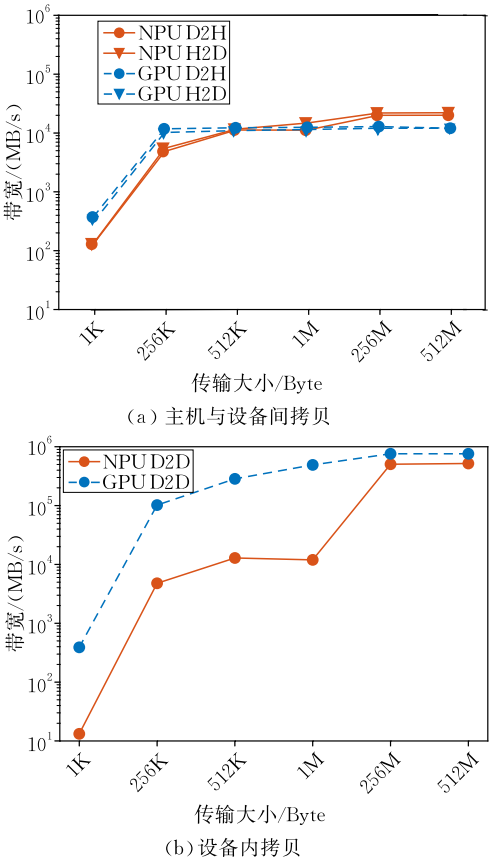


图 11　NPU 与 GPU 带宽性能对比

6　NPU 性能优化

6.1　深度学习框架适配优化

现有的深度学习框架多基于 GPU 平台开发，NPU 上的 CANN 与 GPU 上的 CUDA 有所不同，在 NPU 上适配开发深度学习框架，既要保证现有深度学习框架的使用习惯不变，又要充分发挥 NPU 的算力优势. 为分析 NPU 上深度学习框架的适配开发特性，本文使用 MindSpore、TensorFlow、PyTorch 三种不同的深度学习框架在 NPU 平台上进行了性能对比，并深入分析了 NPU 上深度学习框架的优化策略.

三种深度学习框架中，TensorFlow 和 PyTorch 已经相对成熟，形成了自身特色的软件设计模式，NPU 的适配要在已有代码库上根据 NPU 软件栈进行开发；MindSpore 从一开始就支持 NPU 和 GPU 平台. 图 12 为三种深度学习框架在 ImageNet ResNet50 和 CIFAR-10 ResNet56 场景下的表现，图 12(a)为训练达到准确度阈值所需时间，图 12(b)为非首轮 Epoch 吞吐量. 从所需时间和吞吐量上来

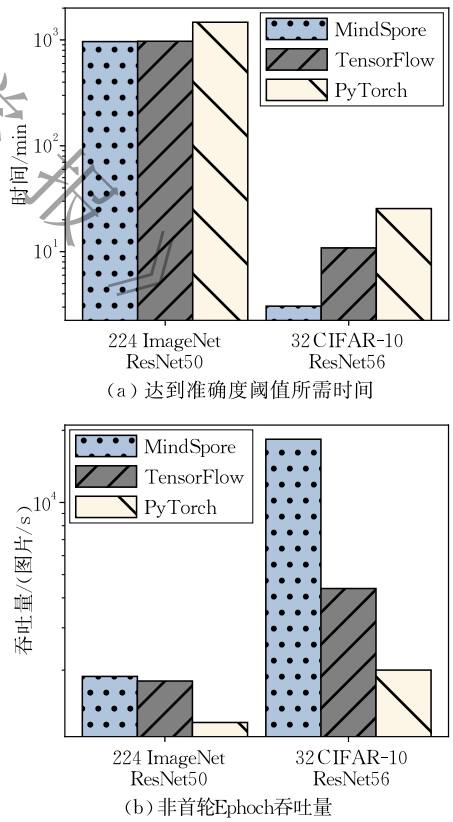


图 12　MindSpore、TensorFlow、PyTorch 三种深度学习框架在 NPU 平台上的性能对比

看，MindSpore 均表现最优.

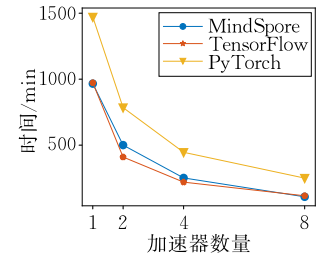
整图编译构建才能发挥 NPU 性能. MindSpore 和 TensorFlow 将整个计算图编译构建，相比 PyTorch 按照单个算子编译构建有较大性能提升.

三种深度学习框架中，MindSpore 和 TensorFlow 为静态图模式，框架首先构建整个模型的计算图，再调用 CANN 软件栈中的编译器，将计算图编译为二进制可执行文件. 编译过程中将启动一系列编译优化，如算子融合.

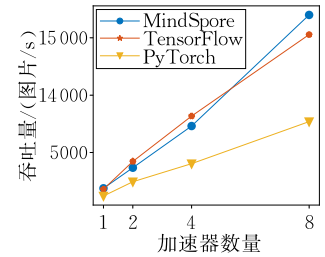
PyTorch 为动态图模式，计算图的构建与计算图的计算是同时发生的，即构建单个算子后立即执行单个算子的计算. NPU 上的 PyTorch 对计算图中的每个算子单独构建，算子与算子之间独立，未进行整图算子融合优化. 本文评测所使用的 ResNet 使用的残差块结构由卷积、批量归一化和 ReLU 算子组成. 芯片中的计算单元要想发挥算力，必要条件是保证数据能够及时准确地出现在计算单元的输入缓存区. 如果编译时进行算子融合且在缓存区空间允许的情况下，卷积的计算结果将停留在 AI Core 内的缓存区，可直接输送给向量计算单元进行批量归一化和 ReLU 计算；如不进行算子融合，卷积的计算结果将输出到 AI Core 之外的核外存储系统，批量归一化需再次从核外存储系统将数据搬运回 AI Core 内部，频繁的数据搬运意味着计算单元需要等待更长时间才能访问到数据. 另外，本文评测发现，算子编译也需一定时间，单个算子的编译至少需要 2 s 以上，当计算图中算子较多时，对每个算子单独编译构建也将占用大量时间，导致 PyTorch 的首轮 Epoch 训练时间很长.

6.2 多加速器并行加速

为测试 NPU 的横向扩展能力，我们使用三种深度学习框架在单机不同加速器数量下基于 ImageNet ResNet 场景评测 NPU 的多卡可扩展性. 硬件层面，多张昇腾加速器之间搭载了 100 G RoCE v2 网络；软件层面，CANN 提供了集合通信库. 评测主要基于数据并行，即对数据集按照加速器数量平均切分，每张加速器都加载整个模型，并基于数据集的一部分训练，每次迭代后进行模型参数的聚合. 评测结果如图 13(a)所示，从 1 张加速器到 8 张加速器，三种深度学习框架中 MindSpore 和 TensorFlow 都展示了良好的横向扩展性，8 张加速器时，MindSpore 达到准确度阈值仅需 110 min；图 13(b)直观地显示出 MindSpore 可以近乎线性地利用多张 NPU 算力.



(a) 达到准确度阈值所需时间



(b) 平均吞吐量

图 13 多加速器可扩展性分析

6.3 算子优化

6.3.1 分块

合理分块可提升 NPU 算子性能. 在开发 NPU 上的算子时，根据算子数学计算逻辑和芯片体系结构进行分块以充分利用 NPU 算力.

NPU AI Core 中的缓存区空间有限，使用 TIK 进行算子开发时，一般需要对数据进行分块，将分块数据依次从 AI Core 核外存储系统经各类缓存区搬运至 ***L0*** 缓存区. 算法 Y 为 NPU 上基于 TIK 接口的半精度矩阵乘法. 算法输入为加速器内存上的两个矩阵 $\boldsymbol{A}[m,k]$ 和 $\boldsymbol{B}[k,n]$，算法根据分块参数 *m_tile_size*、*k_tile_size* 和 *n_tile_size* 进行三层循环. NPU 的矩阵计算单元在进行一次矩阵乘法时需要 16 个半精度浮点数对齐，最内侧循环先将加速器主存上的矩阵搬运至 ***L1*** 缓存区，且 ***L1*** 缓存区上的 ***A_L1*** 和 ***B_L1*** 按照 16 对齐，接着调用 TIK 中的矩阵乘法接口完成计算. 分块参数 *m_tile_size*、*k_tile_size* 和 *n_tile_size* 影响了循环次数，进而影响数据搬运次数.

算法 Y. NPU 矩阵乘法.

输入：矩阵 $\boldsymbol{A}[m,k]$，矩阵 $\boldsymbol{B}[k,n]$

输出：矩阵 $\boldsymbol{C}[m,n]$

$m\_cycle\_times = m / m\_tile\_size$

$n\_cycle\_times = n / n\_tile\_size$

$k\_cycle\_times = k / k\_tile\_size$

FOR *n_idx* FROM 0 TO *n_cycle_times*

  FOR *m_idx* FROM 0 TO *m_cycle_times*

    FOR *k_idx* FROM 0 TO *k_cycle_times*

$$data\_move(\boldsymbol{A}[m,k],\boldsymbol{A\_L1}[k\_tile\_size/16,m\_tile\_size,16])$$
$$data\_move(\boldsymbol{B}[k,n],\boldsymbol{B\_L1}[k\_tile\_size/16,n\_tile\_size,16])$$
$$\boldsymbol{C\_L0}=\mathrm{matmul}(\boldsymbol{A\_L1},\boldsymbol{B\_L1})$$
$$data\_move(\boldsymbol{C\_L0},\boldsymbol{C}[m,n])$$

本文以 $\boldsymbol{A}[16,256]$ 和 $\boldsymbol{B}[256,1024]$ 大小的矩阵进行了测试，将 n_tile_size 和 k_tile_size 设置成不同的大小，剔除数据在主机与设备之间的拷贝时间，仅分析 AI Core 上的计算耗时，结果如图 14 所示. 根据图 14，在缓存区大小允许的情况下，适当增加分块大小，可减少数据搬运次数，加速计算过程；当分块大小足够大时（$k\_tile\_size>64$，$n\_tile\_size>256$），更大的分块并没有带来更多性能提升，因为分块越大，计算单元无法一次完成分块的计算，需要在调度指令的控制下，循环更多次才能完成计算任务.

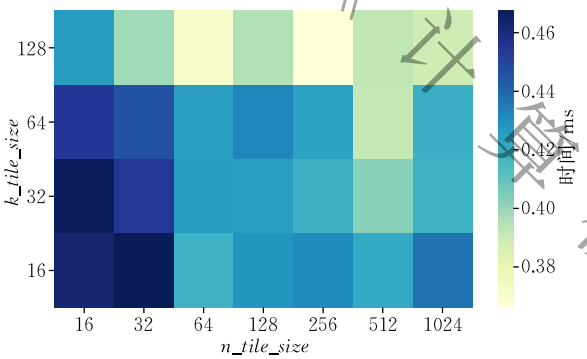


图 14　矩阵乘法分块大小对计算耗时影响

6.3.2　低精度

NPU 上的算子尽量使用 FP16 或更低精度. NPU 中矩阵计算单元只能进行 FP16 或更低精度计算，FP32 计算将会被调度到算力较低的向量计算单元，且 FP16 和 FP32 之间的转化会有性能损耗. 尽量使用 FP16 或更低精度能发挥 NPU 算力.

本文以批量归一化（Batch Normalization，简称 BatchNorm）为案例，分析了 NPU 上 FP16 精度的优势. BatchNorm 常用在卷积神经网络中以解决深度网络中计算不稳定的问题[33]，本文在 ImageNet ResNet50 场景上对 BatchNorm 分别使用 FP16 和 FP32 进行计算，如表 6 所示，BatchNorm 为 FP32 时，吞吐量为 FP16 的 49.8%，达到准确度阈值的时间为 1.97 倍. 在 ResNet50 中，BatchNorm 的输入为卷积的输出. 在 NPU 中，矩阵计算单元进行卷积计算，其计算结果为 FP16，如果 BatchNorm 设置为 FP32，矩阵计算单元计算单元的输出需要被转换成 FP32，类型转换本身耗时，且增加了调度时间.

表 6　ResNet50 下 BatchNorm 使用 FP16 与 FP32

	吞吐量(image/s)	达到阈值的时间/min
FP32	1091	1458
FP16	2192	740

6.4　混合精度训练优化

合理设置 Loss Scale. NPU 平台依赖 FP16 精度下的算力，FP16 在训练过程中存在数值表征范围过小而溢出的问题. 考虑到部分权重的梯度数值较小，文献[8]提出的混合精度训练技术在前向阶段对损失成倍放大，根据链式法则，反向时梯度不至于过小而向下溢出. 放大的倍数被称为 Loss Scale，通常是 8 的倍数. 在设置 Loss Scale 时，有 Loss Scale 固定值策略和 Loss Scale 动态更新策略.

ResNet50 上可以使用 Loss Scale 固定值策略. 本文分别从 1 到 1024 选择不同的 Loss Scale 值进行了测试. 测试发现，如果 Loss Scale 设置小于 32 则存在数值溢出的风险，模型在预测集上的准确度接近 0. Loss Scale 设置较小会造成模型在后续迭代均为无效计算，所训练的模型不具有泛化能力，对后续训练产生的负面影响不可逆. Loss Scale 较大的负面影响较小，因为即使设置得较大，损失放大后发生数值向上溢出，程序可以很容易检测出上溢，上溢后放弃本批次梯度更新，直接进入下一次循环即可. 除了对模型的收敛能力进行检验，本文也发现，当 Loss Scale 在 32 至 1024 之间时，Loss Scale 对模型达到准确度阈值的时间几乎无影响. 对于 ResNet 模型，建议将 Loss Scale 设置为 1024.

对于 Transformer 模型，如果未添加 NPU 平台上精度的特殊处理，模型很快溢出，无法完成训练. 如希望在 NPU 平台上训练 Transformer，需使用改进的混合精度优化器和 Loss Scale 动态更新策略. Loss Scale 动态更新策略是指，训练开始时设置 Loss Scale 初始值，当检测到数值上溢，则缩小 Loss Scale 值，并放弃本批次梯度更新.

7　发现与讨论

7.1　主要发现

经过上述详细评测分析与优化讨论，本文将主要发现总结如下.

对于期望从 GPU 迁移到昇腾平台的潜在用户和使用者，主要结论为：

(1) 昇腾加速器适合进行稠密计算型工作负载，不适合稀疏计算型工作负载. 稠密计算型工作负

载指包含多层卷积或多层稠密全连接的网络.稀疏计算型工作负载指 RNN 或层数较少的全连接网络.从端到端模型来看,在 ImageNet 2012 数据集上训练 ResNet50 模型,达到 75.9%准确度阈值,昇腾 910 训练速度为 Tesla V100 的 2.89 倍(图 3(c));在 WMT16 en-de 数据集上训练 Transformer 模型,达到 25.0 BLEU,昇腾 910 速度为 Tesla V100 的 1.67 倍(图 4(a)).在 Criteo 数据集上训练 DeepFM 模型以及在 IMDB 数据集上训练 LSTM 模型,昇腾 910 的性能均不及 Tesla V100.从单算子分析来看,昇腾加速器适合进行卷积和全连接计算(图 10(a)、图 10(b)),不适合 RNN 类计算(图 10(c)).与 V100 相比,昇腾 910 在卷积算子上可获得 7.37 倍的加速效果,在全连接算子上可获得 5.64 倍的加速效果(表 5).更大的 Batch Size 和更宽的模型和算子可充分利用昇腾的算力(图 8、图 10).

(2)昇腾的强大算力由 FP16 精度的矩阵计算单元提供,需要使用混合精度训练才能充分利用其算力,混合精度训练容易产生数值溢出的问题.使用者需要合理设置 Loss Scale.不同模型和训练场景下的 Loss Scale 设置有所区别,例如,ResNet50 模型的 Loss Scale 建议设置为 1024,小于 32 的 Loss Scale 会导致模型无法拟合.

(3)在功耗方面,稠密性计算场景昇腾的功耗远低于 GPU(图 7),ImageNet ResNet 和 WMT16 en-de Transformer 场景的总功耗昇腾分别是 GPU 的 27.4%和 26.4%;在稀疏性计算场景以及空载时,昇腾的基础功耗高.

对于深度学习框架开发者来说,主要优化策略和方向为:

(1)昇腾上的 CANN 软件栈与 GPU 上的 CUDA 开发和使用方式有所区别.基于昇腾和 CANN 开发深度学习框架时,尽量基于 CANN 的编译器整图编译构建,利用编译器的算子融合,方能释放昇腾算力.使用深度学习框架启动训练或推理任务,任务启动后 CANN 软件栈需要耗费一定时间初始化和编译构建计算图,使得首轮 Epoch 耗时较长.

(2)昇腾上进行算子开发时需要进行分块,分块不宜过小,否则需要频繁地搬运数据,影响计算性能(图 14).算子尽量使用 FP16 或者更低精度进行实现,以 BatchNorm 算子为例,使用 FP16 可以获得更快的加速效果(表 6).

对于昇腾体系结构和编译软件栈设计人员来说,仍需解决的问题有:

昇腾在稀疏型计算仍与 GPU 有所差距.昇腾加速器内部融合了针对非神经网络计算的 AI CPU、神经网络类计算的 AI Core 以及高速网络,加速器整体性能得到了提升,但也增加了空载功耗,在无计算任务时,昇腾的功耗远高于 GPU.昇腾的软件栈仍存在提升空间,如输入数据规模大时出现内存问题.

7.2 讨论

重要性. 本文首次对国产昇腾神经网络加速器进行了系统性地评测和分析.本文设计了评测基准程序集,测试了四个具有代表性的端到端神经网络模型及相应功耗;深入分析了三个常用算子在不同参数组合下硬件利用率和访存特性;针对昇腾体系结构和软件栈特点,深入分析了深度学习框架的适配方式,提出了合理设置数据分块和低精度实现算子的优化策略.读者可通过本文对昇腾加速器的性能、适用场景、优化方向有一个全面的认识;期望从 GPU 迁移到昇腾平台的潜在用户、深度学习平台架构设计师可从本文获取关于昇腾在不同应用场景下的速度和准确度信息;深度学习框架开发者和算子开发者可根据本文提出的优化策略,有针对性地适配开发昇腾上的深度学习框架和算子.加速器厂商可根据性能评测结果对下一代软硬件栈采取进一步优化措施.

未来工作. 优化昇腾上访存密集型应用,例如 RNN 类计算.

8 相关工作

近年来,深度学习在图像分类[11,34]、目标检测[35]、情感分析[31]、机器翻译[12]等各类任务上的准确度超越传统方法.深度学习的飞速发展得益于芯片算力的提升.AlexNet[34]首次使用 GPU 训练深度神经网络后,GPU 开始成为深度学习算力基础设施的关键部分,为各类应用提供训练和推理支持.基于 GPU 的深度学习软件栈得到飞速发展,PyTorch[2]等各类深度学习框架均在 GPU 平台上进行优化.

然而,深度学习的代价是极高的计算复杂度,为提高计算效率,方便模型部署,一些研究者提出了针对深度学习的软硬件优化方法.一些研究在异构计算的框架下,设计了面向神经网络的专用芯片,如 TPU[4]、Eyeriss[5]等;一些研究分析了深度学习中数值精度,提出使用 FP32 和 FP16 相结合的混合精度训练方法[8,19];一些研究提出充分利用 CPU 与加速器融合架构中内存共享等特点设计机器学习系统[36-38];文献[25]提出了一种针对深度学习的编译

器，以适配不同类型的加速器硬件.

在深度学习软硬件系统飞速发展的背景下[39]，华为推出了昇腾系列加速器.昇腾是一种异构计算框架下的神经网络加速器，采用华为自研的 DaVinci 架构，包含矩阵计算单元、向量计算单元和标量计算单元等多种计算单元[24].昇腾上需要依托 CANN 软件栈来完成算子开发、设备管理、内存分配等功能.基于 CANN，华为对 TensorFlow 和 PyTorch 深度学习框架进行了适配，开发者可快速从 GPU 平台迁移到昇腾平台.同时，华为推出了跨 GPU 和昇腾的深度学习框架 MindSpore.

深度学习软硬件系统不断涌现，业界需要公开的评测基准集来测试不同软硬件的性能.在神经网络训练领域，MLPerf[22] 和 DAWNBench[23] 重点关注端到端模型训练的准确性，DeepBench 关注某些计算密集型算子的性能表现；ParaDnn[40] 使用随机生成的训练数据测试不同硬件平台的算力利用率、吞吐量和带宽；文献[9]对 GPU 上多款深度学习框架进行对比和评测.本文设计的跨平台评测基准是对已有评测基准的补充，可对昇腾加速器进行更为细致的分析和评测.

9　结　论

华为昇腾是一款新型神经网络加速器，截止目前，没有针对昇腾的系统性评测、分析和优化研究.为填补昇腾性能数据、软件栈调用方式和优化方向等方面空缺，本文系统性地对昇腾进行了测试，本文在昇腾和 GPU 上分别测试了四种具有代表性的端到端神经网络模型，详细分析了昇腾和 GPU 在三个计算密集型算子上的硬件利用率和访存表现，重点分析了昇腾上不同深度学习框架、算子以及不同混合精度训练设置等加速优化方法.对评测结果进行总结后发现，昇腾加速器适合进行稠密计算型深度学习训练，功耗相对较低.为充分发挥昇腾的算力，深度学习框架开发者需要注意昇腾芯片体系结构和 CANN 软件栈的特点，充分利用整图编译构建、数据分块和算子融合等特性.

致　谢　本文所使用的计算资源由华为公司和中国人民大学校级计算平台提供.感谢华为公司金雪峰、宁伟康、谭涛、贺冬冬等人对昇腾体系结构、软件栈和 MindSpore 框架的详细解读；感谢北京大学樊春提供运维支持！

参　考　文　献

[1] Abadi M, Barham P, Chen J, et al. TensorFlow: A system for large-scale machine learning//Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI). Savannah, USA, 2016: 265-283

[2] Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library//Advances in Neural Information Processing Systems 32 (NeurIPS). Vancouver, Canada, 2019: 8024-8035

[3] Wang Chao, Wang Teng, Ma Xiang, et al. Research progress on FPGA-based machine learning hardware acceleration. Chinese Journal of Computers, 2020, 43(6): 1161-1182 (in Chinese)
(王超，王腾，马翔等. 基于 FPGA 的机器学习硬件加速研究进展. 计算机学报，2020，43(6)：1161-1182)

[4] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit//Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA). Toronto, Canada, 2017: 1-12

[5] Chen Y-H, Krishna T, Emer J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid State Circuits, 2017, 52(1): 127-138

[6] Chen T, Du Z, Sun N, et al. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. ACM SIGARCH Computer Architecture News, 2014, 42(1): 269-284

[7] De Sa C, Feldman M, Ré C, et al. Understanding and optimizing asynchronous low-precision stochastic gradient descent//Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA). New York, USA, 2017: 561-574

[8] Micikevicius P, Narang S, Alben J, et al. Mixed precision training//Proceedings of the 6th International Conference on Learning Representations (ICLR). Vancouver, Canada, 2018

[9] Shi S, Wang Q, Xu P, et al. Benchmarking state-of-the-art deep learning software tools//Proceedings of the 2016 7th International Conference on Cloud Computing and Big Data (CCBD). Macau, China, 2016: 99-104

[10] Bahrampour S, Ramakrishnan N, Schott L, et al. Comparative study of deep learning software frameworks//Proceedings of the 4th International Conference on Learning Representations (ICLR) Workshop Track. San Juan, Puerto Rico, 2016

[11] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition//Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, USA, 2016: 770-778

[12] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need//Advances in Neural Information Processing Systems 30 (NeurIPS). Long Beach, USA, 2017: 5998-6008

[13] Guo H, Tang R, Ye Y, et al. DeepFM: A factorization-machine based neural network for CTR prediction//Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI). Melbourne, Australia, 2017: 1725-1731

[14] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735-1780

[15] Goodfellow I, Bengio Y, Courville A. Deep Learning. Cambridge, USA: MIT Press, 2016

[16] Robbins H, Monro S. A stochastic approximation method. Annals of Mathematical Statistics, 1951, 22(3): 400-407

[17] Kingma D P, Ba J. Adam: A method for stochastic optimization //Bengio Y, LeCun Y eds. Proceedings of the 3rd International Conference on Learning Representations (ICLR). San Diego, USA, 2015

[18] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors. Nature, 1986, 323 (6088): 533-536

[19] Sze V, Chen Y-H, Yang T-J, et al. Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE, 2017, 105(12): 2295-2329

[20] McCandlish S, Kaplan J, Amodei D, et al. An empirical model of large-batch training. arXiv preprint arXiv: 1812.06162, 2018

[21] Dongarra J, Luszczek P, Petitet A. The LINPACK benchmark: Past, present and future. Concurrency and Computation: Practice and Experience, 2003, 15(9): 803-820

[22] Mattson P, Reddi V J, Cheng C, et al. MLPerf: An industry standard benchmark suite for machine learning performance. IEEE Micro, 2020, 40(2): 8-16

[23] Coleman C, Kang D, Narayanan D, et al. Analysis of DAWNBench, a time-to-accuracy machine learning performance benchmark. ACM SIGOPS Operating Systems Review, 2019, 53(1): 14-25

[24] Liang Xiao-Yao. Ascend AI Processor Architecture and Programming: Principles and Applications of CANN. Beijing: Tsinghua University Press, 2019(in Chinese)
(梁晓峣. 昇腾 AI 处理器架构与编程：深入理解 CANN 技术原理及应用. 北京：清华大学出版社，2019)

[25] Chen T, Moreau T, Jiang Z, et al. TVM: An automated end-to-end optimizing compiler for deep learning//Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Carlsbad, USA, 2018: 578-594

[26] Papineni K, Roukos S, Ward T, et al. BLEU: A method for automatic evaluation of machine translation//Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL). Philadelphia, USA, 2002: 311-318

[27] Fawcett T. An introduction to ROC analysis. Pattern Recognition Letters, 2006, 27(8): 861-874

[28] Krizhevsky A. Learning Multiple Layers of Features from Tiny Images [M. S. dissertation]. University of Toronto, Toronto, Canada, 2009

[29] Russakovsky O, Deng J, Su H, et al. ImageNet large scale visual recognition challenge. International Journal of Computer Vision, 2015, 115(3): 211-252

[30] Bojar O, Chatterjee R, Federmann C, et al. Findings of the 2016 conference on machine translation//Proceedings of the First Conference on Machine Translation (WMT). Berlin, Germany, 2016: 131-198

[31] Maas A L, Daly R E, Pham P T, et al. Learning word vectors for sentiment analysis//Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL). Portland, USA, 2011: 142-150

[32] Williams S, Waterman A, Patterson D. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Communications of the ACM, 2009, 52(4): 65-76

[33] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift// Proceedings of the 32nd International Conference on Machine Learning (ICML). Lille, France, 2015: 448-456

[34] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks//Advances in Neural Information Processing Systems 25 (NeurIPS). Lake Tahoe, USA, 2012: 1106-1114

[35] Liu W, Anguelov D, Erhan D, Szegedy C, et al. SSD: Single shot multibox detector//Proceedings of the 14th European Conference on Computer Vision (ECCV). Amsterdam, Netherlands, 2016: 21-37

[36] Zhang F, Zhai J, He B, et al. Understanding co-running behaviors on integrated CPU/GPU architectures. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(3): 905-918

[37] Zhang Feng, Zhai Ji-Dong, Chen Zheng, et al. Survey on performance analysis, optimization, and applications of heterogeneous fusion processors. Journal of Software, 2020, 31(8): 325-346(in Chinese)
(张峰，翟季冬，陈政等. 面向异构融合处理器的性能分析、优化及应用综述. 软件学报，2020, 31(8): 325-346)

[38] Zhang F, Zhai J, Shen X, et al. POCLib: A high-performance framework for enabling near orthogonal processing on compression. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(2): 459-475

[39] Du Xiao-Yong, Lu Wei, Zhang Feng. History, present, and future of big data management systems. Journal of Software, 2019, 30(1): 127-141(in Chinese)
(杜小勇，卢卫，张峰. 大数据管理系统的历史、现状与未来. 软件学报，2019, 30(1): 127-141)

[40] Wang Y, Wei G, Brooks D. A systematic methodology for analysis of deep learning hardware and software platforms// Proceedings of the Machine Learning and Systems 2020 (MLSys). Austin, USA, 2020: 30-43

LU Wei-Zheng, M. S., engineer. His research interests include high performance computing and data science.

ZHANG Feng, Ph. D., associate professor. His research interests include big data system and high performance computing.

HE Yin-Xuan, B. S. candidate. His research interest is high performance computing.

CHEN Yue-Guo, Ph. D., professor. His research interests include big data system and knowledge graph.

ZHAI Ji-Dong, Ph. D., associate professor. His research interests include high performance computing, performance analysis and optimization.

DU Xiao-Yong, Ph. D., professor. His research interests include database system and information retrieval.

Background

The great success achieved by deep neural networks (DNNs) mainly relies on the computation ability provided by modern chips. Nvidia's high performance and general-purpose Graphics Processing Units (GPUs) are widely used to build deep learning tools and software[1-2]. There is an industry-wide trend towards domain specific neural network accelerators to extend deep learning performance[3]. For example, Google has released Tensor Processing Unit (TPU) and has deployed TPUs in data center[4]; MIT proposed an energy-efficient reconfigurable accelerator for deep convolutional neural networks[5].

In addition to these accelerators, Huawei has developed the Ascend accelerator, including Ascend 910 for training and Ascend 310 for inference. Ascend accelerators feature super computing power, high integration and fast network bandwidth. Take Ascend 910 as an example, it delivers 256T half precision FLOPS, 32GB memory with 1200 GB/s bandwidth and 100G RoCE v2 network adapter. Ascend is a new type of neural network accelerator. Differences between Ascend and GPU are: (1) the chip uses task-specific processing units which is mainly for neural networks; (2) the computing power is based on lower precision; (3) the compiler software stack on Ascend is different from GPU.

The main goal of deep learning is to train a statistical model based on train dataset and the fitted model should make high quality predictions on unseen data, which is referred to as generalization. From the perspective of hardware design, lower precision enables faster speed for a single iteration. However, lower precision could prevent convergence or need more iterations[7]. Previous studies focused on low precision training[8] and performance analysis[9-10] on GPU platform. Can lower precision of Ascend meet the need of different scenarios? What's actual performance of the hardware and software stack on Ascend? The potential of Ascend remains unknown.

To thoroughly understand its feature and performance, we conduct a systematic evaluation on Huawei Ascend and analyze optimization methods for faster training. Our contributions include: (1) We compare the performance results of Ascend and GPU with four end-to-end neural networks (ResNet[11], Transformer[12], DeepFM[13] and LSTM[14]) on well-known public datasets; (2) We analyze optimization methods on Ascend including deep learning framework developing, operator tile strategy and lower precision training configurations; (3) We measure hardware utilization and memory access pattern on three compute-intensive operators (fully-connected, convolution and RNN).

To the best of our knowledge, we are the first to conduct comprehensive evaluation on Ascend. The main findings are as follows: (1) Ascend is suitable for dense neural network training and neural networks should be quantized from 32-bit to 16-bit precision. (2) To fully utilize the Ascend hardware, deep learning framework should compile the whole computation graph of neural network, making operators fused. When implementing operators, tile size should be carefully configured and lower precision is preferred. Models should be trained with mixed precision mode. (3) Ascend is not suitable for sparse workload. There are some internal errors when allocating extreme big size memory. The power consumption is relatively high when there is no workload.

This paper is supported by the National Key R&D Program of China under Grant No. 2018YFB1004401 and the National Natural Science Foundation of China under Grant Nos. U1711261 and 62172419.