计算机科学
COMPUTER SCIENCE

# 深度学习编译器研究综述

刘正煜，张帆，祁晓峰，高彦钊，宋怡景，范旺

引用本文

刘正煜，张帆，祁晓峰，高彦钊，宋怡景，范旺. 深度学习编译器研究综述[J]. 计算机科学，2025，52(8)：29-44.

LIU Zhengyu, ZHANG Fan, QI Xiaofeng, GAO Yanzhao, SONG Yijing, FAN Wang. Review of Research on Deep Learning Compiler [J]. Computer Science, 2025, 52(8): 29-44.

---

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250100062

# 深度学习编译器研究综述

刘正煜[1,2]　张　帆[1]　祁晓峰[1]　高彦钊[1]　宋怡景[3]　范　旺[3]
1 信息工程大学　郑州 450000
2 网络空间安全教育部重点实验室　郑州 450000
3 复旦大学大数据研究院　上海 200000
(704676894@qq.com)

**摘　要**　随着人工智能的快速发展,越来越多的神经网络模型和算法相继涌现。与此同时,摩尔定律逐渐失效,新型加速器和计算机体系架构层出不穷,这推动了神经网络模型在这些新硬件平台上实现高效部署的迫切需求。在此背景下,深度学习编译器应运而生。与传统编译器不同,深度学习编译器将不同的网络模型作为输入,采用多级中间表示设计,逐层优化模型,并在编译器后端针对特定硬件架构进行优化,最终生成优化后的可执行程序。首先,介绍了深度学习编译器的通用框架,包括各个核心组件和总体流程;随后,系统地分类探讨了编译器的各类优化技术,并对近年来的研究进展进行总结,揭示了领域内的热点和发展趋势;最后,整理了现阶段的深度学习编译器研究,并根据现有研究现状展望了未来的研究方向。

**关键词**:深度学习;编译器;编译优化
**中图分类号**　TP314

## Review of Research on Deep Learning Compiler

LIU Zhengyu[1,2],ZHANG Fan[1],QI Xiaofeng[1],GAO Yanzhao[1],SONG Yijing[3] and FAN Wang[3]
1 University of Information Engineering,Zhengzhou 450000,China
2 Key Laboratory of Cyberspace Security,Ministry of Education of China,Zhengzhou 450000,China
3 Institute of Big Data,Fudan University,Shanghai 200000,China

**Abstract**　With the rapid development of artificial intelligence,an increasing number of neural network models and algorithms have been proposed. Meanwhile,as Moore's Law gradually loses its effectiveness,a variety of new accelerators and computer architectures have emerged,creating an urgent demand for the efficient deployment of neural network models on these novel hardware platforms. Against this backdrop,deep learning compilers have emerged. Unlike traditional compilers,deep learning compilers take various network models as input,use a multi-level intermediate representation design to optimize models layer by layer,and perform hardware-specific optimizations in the backend for different hardware architectures,ultimately generating optimized executable programs. This paper first introduces the general framework of deep learning compilers,including their core components and overall process. It then categorizes and discusses the various optimization techniques used in compilers,summarizing recent research progress and highlighting the key research trends in the field. Finally,this paper organizes the current stage of deep learning compiler research and looks forward to future research directions based on the current state of existing research.

**Keywords**　Deep learning,Compiler,Compiler optimization

## 1　引言

随着人工智能技术的飞速发展,深度学习算法和框架的种类不断增加,伴随而来的是海量新算子的提出[1],这使得算子库的开发、维护、优化和测试工作面临前所未有的挑战,工作量飞速增长。同时,近年来硬件领域的突破也愈加显著,尤其是专用加速芯片的崛起,如 TPU 和 NPU 等,这些新型硬件架构在性能上远超传统通用计算平台,带来了深度学习应用的巨大性能提升[2]。然而,不同硬件平台之间存在显著的架构差异,导致性能的可移植性成为一个迫切的需求。为了弥合这一差距,实现前端深度学习模型与后端硬件的协同优化,深度学习编译器应运而生[3]。它的核心功能是将深度学习模型作为输入,经过一系列优化后生成适配各种硬件平台的高效代码,最终输出与特定硬件架构兼容的可执行机器码。通过这种编译优化过程,深度学习编译器不仅能够提升计算性能,还能减少硬件资源的消耗,推动人工智能在多种硬件平

到稿日期:2025-01-09　返修日期:2025-04-29
基金项目:国家重点研发计划(2022YFB4500900)
This work was supported by the National Key R&D Program of China (2022YFB4500900).
通信作者:张帆(17034203@qq.com)

台上的高效部署。

本文从深度学习编译器的整体框架出发，深入探讨了其各类核心组件及优化技术，系统地总结了现有深度学习编译器的主要特点，并调研了近年来的最新研究成果。在此基础上，还对深度学习编译器的未来发展进行了前瞻性展望。第2章详细介绍了深度学习编译器的基本架构和组成部件；第3章则聚焦于中间表示（Intermediate Representation，IR）的相关技术；第4章和第5章分别探讨了前端和后端的优化技术；第6章对业界近几年的研究进展进行了全面分析；第7章则展望了深度学习编译器的未来发展方向。

## 2 深度学习编译器体系架构

深度学习编译器的通用架构如图1所示，通常由编译器前端和后端构成，在这两者之间，IR起到关键的桥梁作用。在编译过程中，编译器对源程序进行的一次完整的处理过程被称为一个PASS，整个编译过程通常会经过多个PASS，每个PASS负责不同的扫描和处理任务，以确保最终生成的目标代码达到最佳效果。

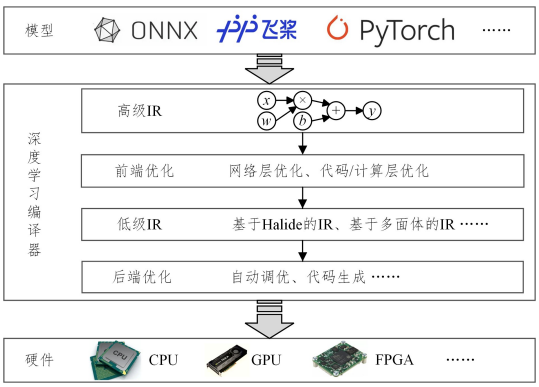


图1 深度学习编译器框架图

Fig. 1 Framework diagram of deep learning compiler

深度学习编译器通常采用多级IR设计，以实现高效的代码优化。在这一处理流程中，模型会被转换为多个层级的IR。编译器前端使用的IR被称为高级IR，它包含与硬件无关的抽象信息，主要用于执行与硬件无关的转换和优化。而编译器后端使用的IR则被称为低级IR，它更贴近目标硬件的实际实现，主要用于执行与硬件相关的优化、代码生成和最终的编译工作。不同级别的IR面临的优化压力因目标硬件的特性、应用需求和可用资源的不同而有所区别。

通过这种分层设计，深度学习编译器能够灵活应对不同复杂度的模型，并在多种硬件平台上实现高效的代码生成和执行。这种架构不仅显著提升了编译器的可维护性和可扩展性，还为深度学习应用的广泛部署与性能优化提供了有力的支持。

表1列出了深度学习编译器中的各类优化技术及近年来的相关研究成果，后文将对其进行具体介绍。

表1 编译优化技术汇总

Table 1 Summary of compilation optimization techniques

| 类别 | 具体内容 | | 研究进展 |
|---|---|---|---|
| IR相关 | 设计新型IR | | 文献[4-9] |
| | 其他 | | 文献[10-11] |
| 前端优化技术 | 网络层优化 | 算子相关 | 文献[12-16] |
| | | 布局转换 | 文献[16-21] |
| | | 内存优化 | 文献[22-23] |
| | | 稀疏化 | 文献[24-27] |
| | | 其他 | 文献[28-29] |
| | 其他 | | 文献[30-33] |
| 后端优化技术 | 调度优化 | 循环优化 | 文献[18,34-40] |
| | | 指令优化 | 文献[41] |
| | | 存储优化 | 文献[13,16,28,41-44] |
| | 基于搜索的自动调优 | 搜索空间 | 文献[45-46] |
| | | 成本模型 | 文献[47-48] |
| | | 搜索算法 | 文献[49-50] |
| | | 其他 | 文献[34,51-53] |
| | 多面体模型 | | 文献[34,39,54-61] |
| | 代码生成、程序执行 | JIT/AOT | 文献[62-66] |
| | | 其他 | 文献[67-68] |
| | 后端架构扩展 | 异构 | 文献[12,16-17,57,69-71] |
| | | 其他 | 文献[18,44,72-74] |
| | 并行化 | | 文献[28,75-76] |
| | 控制流优化 | | 文献[16,77] |
| | 其他 | | 文献[78] |

## 3 中间表示

### 3.1 高级IR

高级IR用于表示神经网络的计算图，通过计算图展现程序的控制流和数据流，因此也被称为图IR。它比源代码更接近程序结构，具备较高的抽象层次，便于理解和调试。通常，高级IR保留了高级语言的一些特性，如循环和条件语句等，但不包含与特定硬件平台相关的信息。

图2展示了Glow中的多级IR示例[79]，XLA HLO和Glow高级IR的张量计算采用基于函数的表达形式，TVM采用基于lambda表达式的张量表达式表示张量计算[80]。

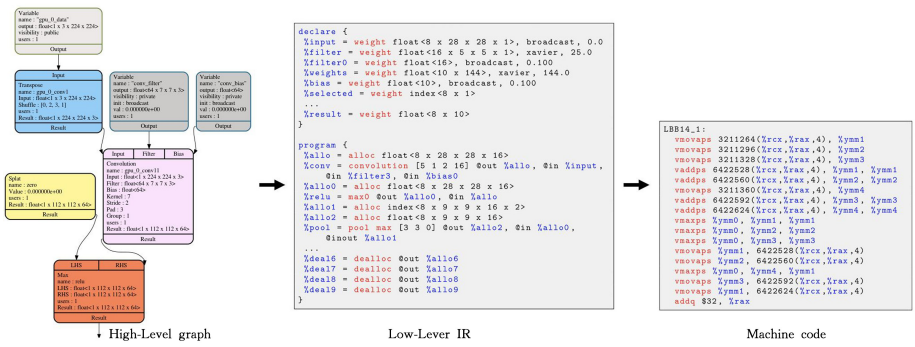


图2 Glow中的多级IR

Fig. 2 Multi-level IR in Glow

3.1.1 计算图的构成

计算图用于表示深度学习网络模型在训练和推理过程中的计算逻辑与状态。在编译器中,计算图通常采用有向无环图表示,其中节点代表算子,边则描述了各个计算之间的依赖关系。借助计算图,深度学习编译器能够分析算子之间的相互关系和依赖,并利用这些信息指导优化过程。

3.1.2 计算图的分类

1)静态计算图

静态计算图是一种特殊的数据结构,其中模型的结构和计算流程在定义后即被固定,不可更改。静态计算图广泛应用于 TensorFlow 和 Theano 等框架,适用于需要对模型结构和计算流程进行优化与静态分析的场景。在静态图模式下,模型通过前端语言定义并形成完整的程序表达后,并不通过前端语言的解释器执行,而是将整个模型交给深度学习框架处理。框架首先分析 API 代码,获取网络层之间的连接拓扑关系,以及参数设置和损失函数等信息,然后使用静态数据结构来描述拓扑结构和其他神经网络组件。

2)动态计算图

与静态计算图不同,动态计算图的结构和计算流程能够根据输入数据动态变化。动态计算图常见于 PyTorch 等动态图框架中,每次前向传播时都会重新构建计算图。动态计算图由于其灵活性,使得模型定义和调试更加便捷,特别适用于需要灵活性和易于调试的场景。在动态图模式下,框架通过前端语言自身的解释器解析代码,并利用深度学习框架的算子分发功能即时执行算子并输出结果。动态图采用命令式编程范式,使得使用前端语言构建神经网络模型更加简洁易懂。静态图与动态图的比较如表 2 所列。

表 2 计算图特性对比

Table 2 Comparison of calculation chart characteristics

| 特性 | 静态图 | 动态图 |
|---|---|---|
| 即时获取中间结果 | × | √ |
| 代码调试难易 | 难 | 简单 |
| 控制流实现方式 | 特定的语法 | 前端语言语法 |
| 性能 | 优化策略多,性能较佳 | 图优化受限,性能较差 |
| 内存占用 | 较少 | 较多 |
| 部署方式 | 可直接部署 | 不可直接部署 |

3)其他

混合计算图结合了静态计算图和动态计算图的优点,既保留了静态图的高效性,又具备动态图的灵活性。在混合计算图中,模型的一部分结构可以静态定义,而另一部分则根据需求动态生成。这种方法通常用于如 TensorFlow Serving 等混合框架,特别适合将训练好的模型部署到生产环境中。

高阶计算图是在计算图中引入更高阶概念的表达形式,如函数、模块或子图。它使得模型可以以更模块化、更抽象的方式进行表示,从而提高了模型的可重用性和可理解性。例如,TensorFlow 2.0 中的 Functional API 允许用户以模块化的方式构建复杂的计算图,简化了模型的构建与管理。

## 3.2 低级 IR

低级 IR 在更细粒度的层面上表示模型,接近目标代码,可以针对不同硬件平台执行特定的优化和代码生成。通常,低级 IR 包含关于寄存器、存储器和指令的更多细节,并与特定的硬件体系结构紧密相关,因此它包含了更多硬件平台的相关信息。这使得编译器能够充分利用硬件特性进行优化。

与高级 IR 相比,低级 IR 通过更精细的表示来描述深度学习模型的计算,提供了调节计算和内存访问的接口,从而实现了针对目标硬件的优化。在本节中,低级 IR 的实现方式被分为 3 类:基于 Halide[81] 的 IR、基于多面体的 IR 以及其他 IR。

3.2.1 基于 Halide 的 IR

Halide 是一种开源的领域特定语言,最初用于并行化图像处理,现已被广泛应用于深度学习模型的计算表达,且具备良好的可扩展性。Halide 的核心思想是将计算和调度分离,采用 Halide 编译器时,它不会直接给出具体的实现方案,而是通过自动搜索和尝试多种调度选项来决定最佳方案。Halide 中的内存引用和循环嵌套受到与轴对齐的有界框的限制,因此它无法用复杂的图(如非矩形图)来表示计算。然而,深度学习中的计算通常具有规则性,因此可以通过 Halide 高效表达。

TVM 中的 Relay IR 借鉴了 Halide 的设计理念,但在多个方面进行了改进。首先,TVM 去除了 Halide IR 对 LLVM 的依赖,重新构建了项目模块和 Halide IR 的设计结构,优化了代码组织,使得图 IR 和前端语言的编译过程更加简洁易懂。其次,通过运行时分发机制,开发者可以方便地添加自定义算子,提高了可重用性。最后,TVM 将变量定义从字符串匹配简化为指针匹配,确保每个变量都以静态单赋值(Static Single Assignment,SSA)形式进行表示,从而提升了变量管理和代码的可维护性。

3.2.2 基于多面体的 IR

多面体模型是一种用于并行编译的数学抽象,它利用线性规划、仿射变换等数学方法,优化具有边界和分支的静态控制流循环代码;通过改变循环顺序、消除依赖关系并增加并行性,来实现代码优化。与 Halide 不同,多面体模型允许内存引用和循环嵌套的边界采用任何形状的多面体,这种灵活性使其在通用编译器中得到了广泛应用。然而,正是这种灵活性,使得多面体模型在与调优机制的集成方面面临挑战。由于多面体模型能够有效处理深度嵌套的循环,因此许多深度学习编译器(如 MLIR[82] 中的 Affine)将多面体模型作为其低级 IR 的一部分,从而提升了编译效率和优化能力。

3.2.3 其他 IR

在设计某种深度学习编译器的低级 IR 时,可以根据具体需求选择不同的设计思路。例如,Glow 的低级 IR 既未采用 Halide,也未使用多面体模型,而是基于指令的表达式,专门用于操作通过地址引用的张量。XLA 的 HLO IR 则可被视为介于高级 IR 和低级 IR 之间的一个层次,因为 HLO 的粒度足够细,能够表示与硬件相关的具体信息。

## 3.3 IR 相关研究现状

表 3 列出了近年来一些具有代表性的编译器 IR 的研究进展。大多数关于 IR 的研究都基于 MLIR,这是由于 MLIR 具有出色的可扩展性和丰富的生态系统。

表 3 IR 相关研究调研

Table 3 IR related studies research

| 来源 | 年份 | 研究团队 | 编译组件 | 类型 |
|---|---|---|---|---|
| 文献[8] | 2024 | 美国茨大学 | MLIR | 设计新 IR |
| 文献[9] | 2023 | 加州大学、慕尼黑工业大学等 | LLVM | 以 IR 为核心的机器学习 |
| 文献[4] | 2022 | 波特兰州立大学 | TVM | 设计面向卷积核的 IR |
| 文献[10] | 2022 | 爱丁堡大学、滑铁卢大学 | MLIR | 设计新的 IR 定义语言 |
| 文献[11] | 2022 | 海得拉巴国际信息技术研究所、爱丁堡大学 | MLIR | 设计新 SSA |
| 文献[5] | 2021 | 爱丁堡大学、微软 | MLIR | 设计新 IR |
| 文献[6] | 2021 | 埃因霍温大学、巴黎高等师范学院等 | MLIR | 设计渐进式提升的多级 IR |
| 文献[7] | 2020 | 爱丁堡大学、格拉斯哥大学 | — | 设计新 IR |
| 文献[27] | 2019 | 麻省理工学院、Adobe | — | 设计新 IR |

在新型 IR 的设计方面，David 等[8]设计了一个新型的多级 Datalog IR，其支持 IR 扩展并保证可执行性，可将数据日志方言翻译成扩展的 IR。Martin 等[5]使用 MLIR 提出了一种基于功能的 Rise IR，这种 IR 将程序语义捕捉为通用计算模式的组合，支持基于重写的优化，并能与 MLIR 中的其他方言结合使用。Pizzuti 等[7]提出了一种通用的高级 IR，旨在实现稀疏数据结构的高效计算。该 IR 利用高级 IR 中的密集构建块，采用依赖类型的形式编码稀疏表示，并以 CSR 格式建模稀疏矩阵，显式表示多个密集数组之间的关系，包括行长度、列索引和非零值。Kjolstad 等[27]开发了一种称为具体索引符号的 IR，用于张量运算，能够指定子计算的发生时机及存储位置。

在其他方面，Aiden 等[9]研发了一个基于 IR 的机器学习模型。Chelini 等[6]提出了一种渐进式提升的 IR 设计策略，利用该策略将低级抽象提升至高级抽象，从而提高针对特定领域的编译性能。Fehr 等[10]分析了过去两年作为 LLVM 的一部分开发的 28 个特定于领域的 IR，并展示了如何使用新的 IR 定义语言表示这些 IR，同时提出了基于 SSA 的新 IR 定义语言 IRDL，重新定义了 MLIR 中的常用方言，改变了编译器 IR 的结构。Bhat 等[11]则开发了一种新的 SSA 构造区域，通过经典的基于 SSA 的推理表达函数优化。

## 4 前端优化

深度学习编译器的前端通过从深度学习框架中获取模型并将其转换为计算图表示，开始对计算图进行优化。这一过程通常被称为计算图优化或图层优化。一旦模型被导入并转化为计算图，编译器便能够确定每个操作的输入和输出张量的形状，进而基于这些形状信息对计算图进行相应的优化。由于前端优化不依赖于特定的硬件后端，因此优化后的计算图结果可以适用于多种不同的硬件目标。图 3 给出了几类来源于微软人工智能系统小组的计算图优化的示例。

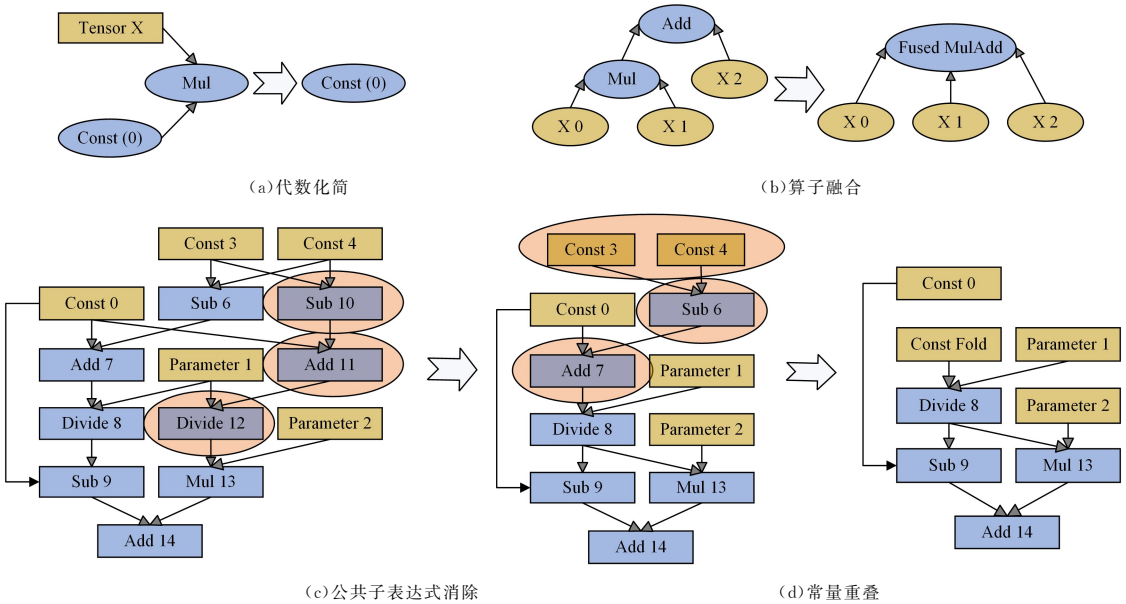


图 3 计算图优化

Fig. 3 Computational graph optimization

深度学习编译器的前端优化通过一系列 PASS 实现对计算图节点的遍历与图转换。在此过程中，编译器从计算图中捕获特定特征，并基于这些特征对计算图进行重写，以便进行后续的优化。除了预定义的 PASS，深度学习编译器还支持开发者自定义 PASS。各个优化 PASS 之间的执行顺序并没有严格规定，但通常遵循一定的规律。例如，首先执行网络层相关的优化，包括算子融合、布局转换和内存分配；然后进行代码和计算层面的优化，如常量折叠、公共子表达式消除、死代码消除以及代数化简。基于常见的优化流程，本章将前端优化分为网络层相关优化和代码/计算层相关优化两大类。

### 4.1 网络层相关优化

#### 4.1.1 算子融合

算子融合是指将网络模型中的多个小算子合并为一个更大的算子,从而有效减少模型训练过程中频繁的数据读取、内存访问以及内核调度的开销[83]。此外,算子融合还可以通过优化循环嵌套结构,为后续的计算优化提供更多机会[84]。以TVM为例,它将算子划分为4类,即映射算子、约简算子、复杂算子和不可融合算子,并根据支配树的结构,为每一类算子设计相应的融合规则。在图3(b)中,Add算子和Mul算子融合为一个Fused MulAdd算子。

#### 4.1.2 算子下沉

算子下沉是指将计算图中的算子移至数据源更靠近的位置,从而缩短数据传输距离,减少内存占用,并提高模型的执行效率。通过这种优化,像转置这样的操作可以被移动到Relu或洗牌等操作之后,促进相似算子的靠近,从而创造更多代数简化的机会。这不仅提升了计算效率,还有助于更好地利用硬件资源。

#### 4.1.3 布局转换

布局转换是指在计算图中寻找最优的数据存储布局。需要注意的是,在这一阶段并不直接进行实际的布局转换,而是将布局转换的节点插入到计算图中,实际的布局转换将在编译器后端优化时执行。

在深度学习中,不同的数据类型采用不同的存储方式。多维数据通常通过多维数组进行存储。以卷积神经网络(CNN)中的特征图为例,其通常使用四维数组来保存,维度分别为:1) $N$(Batch数量):批处理的数量,例如图像的数量;2) $H$(Height):特征图的高度,即垂直方向的像素数;3) $W$(Width):特征图的宽度,即水平方向的像素数;4) $C$(Channels):特征图的通道数,例如彩色RGB图像的通道数为3。由于数据必须以线性方式存储,因此这4个维度在存储时有特定的顺序。不同的深度学习框架可能采用不同的布局顺序,常见的布局方式包括NCHW和NHWC,如图4所示。

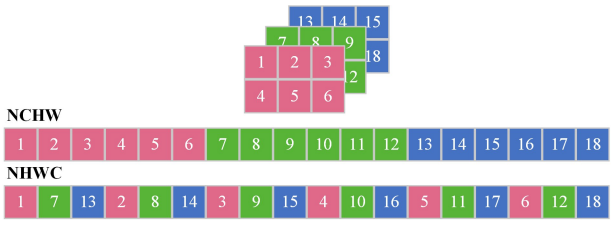


图4 数据布局方式对比

Fig. 4 Comparison of data layout methods

#### 4.1.4 内存划分

在深度学习模型训练过程中,显存需求往往非常庞大。例如,InceptionV4在设置batch size为32进行ImageNet训练时,需要约40 GB的显存;BERT模型拥有768个隐藏层,在batch size为64时需要73 GB显存;而使用ImageNet训练Wide ResNet-152时,batch size为64的情况下,显存需求达到180 GB。这些显存空间大致可以分为静态内存和动态内存两类。通过优化内存分配策略,可以显著提升程序的性能和效率,同时有效减少内存使用量。

### 4.2 代码/计算层相关优化

#### 4.2.1 常量折叠

在传统编译器中,常量折叠和常量传播是常见的优化技术。常量折叠通过在编译时对常量或常量表达式进行计算,来简化代码;而常量传播则是在代码中将表达式中的常量替换为相应的字面量或常量表达式,再应用常量折叠进一步优化代码结构。

在深度学习编译器中,常量折叠的优化同样应用于计算图中,可以将那些在编译时能够确定输出值的节点直接替换为常量,并简化计算图结构。例如,对两个形状为($N$,$C$,$H$,$W$)的四维常量张量进行加法操作,其结果是确定的,这时可以将加法操作合并为一个常量,并在编译时生成该常量。这样,既不需要为加法节点分配额外的存储资源,也避免了在计算图执行过程中反复计算加法操作,可以直接访问已计算好的结果,从而提升了执行效率。在图3(d)中,Const 3和Const 4两个常量形成了一个新的Const Fold。

#### 4.2.2 公共子表达式消除

在传统编译器中,公共子表达式消除是一种常用的优化技术。其主要思想是将多个相同的表达式替换为一个共享变量,该变量存储该表达式的计算结果,从而减少了重复计算带来的开销。此外,编译器还会进行成本效益分析,判断重复计算该表达式的开销是否大于存储计算结果的成本。

在深度学习编译器中,公共子表达式消除遵循类似的思路,区别在于其优化对象是计算图。通过在计算图中识别和消除具有相同结构的子图,可以有效简化计算图结构,从而减少不必要的计算开销。这一优化手段有助于提高计算效率,特别是在处理复杂深度学习模型时,能够显著降低计算资源的消耗。在图3(c)的消除过程中,Sub 10,Add 11和Divide 12这3个算子被消除优化。

#### 4.2.3 死代码消除

在传统编译器中,死代码消除的主要目的是移除对程序执行结果没有任何影响的代码。通过去除这些无用代码,可以减小程序的体积,也能避免在执行过程中进行无关的计算,从而提升程序的执行效率,缩短运行时间。常见的死代码消除方法是通过预处理器判断代码是否需要编译,并根据某个条件值来决定是否包括该程序段。这样不仅能节省计算资源,还能减少内存占用,提高程序的可读性和维护性。

在深度学习编译器中,死代码消除的目标是删除计算图中的“死节点”,即那些不参与最终结果计算的节点。这一优化能够避免为无效节点分配存储空间或进行无意义的计算,从而提高计算图的执行效率,简化图结构,并为后续的图优化PASS铺平道路。死节点通常并非在网络模型定义时引入,而是在其他图优化PASS中产生的,因此,死代码消除往往作为后续优化步骤的一部分进行应用。常见的死节点包括无用的控制流、空操作指令、零维张量以及与训练过程相关但推理阶段不需要的子图等。

#### 4.2.4 代数化简

代数化简是一种通过应用交换律、结合律等数学规律来优化计算图中算子的执行顺序,或去除冗余算子,从而提高整体计算效率的技术。代数化简可以通过子图替换的方式

实现。一种方法是首先设计一个通用的子图替换框架，然后对各个规则进行实例化；另一种方法则是为每个具体规则定制优化逻辑。

根据不同的优化目标，代数化简通常分为 3 类：算术化简、运行时化简和广播化简。算术化简着重于简化运算过程中的数学表达式，减少不必要的计算步骤；运行时化简优化的是在程序执行过程中可以提前确定结果的操作，从而减少运行时的开销；而广播化简则关注于调整张量操作的形状，使得数据传输和计算更加高效。通过代数化简，编译器能够更有效地优化计算图，从而不仅提升了计算性能，还能够减少内存消耗和执行时间，进一步增强深度学习模型的运行效率。

### 4.3 前端优化相关研究现状

近些年的前端优化技术主要集中在网络层优化，具体如表 4 所列。

表 4 前端优化技术调研

Table 4 Front-end optimization techniques research

| 来源 | 年份 | 开发团队 | 编译组件 | 优化类型 |
|---|---|---|---|---|
| 文献[12] | 2024 | 北京软件开发环境实验室、北京航空航天大学 | MLIR | 算子融合 |
| 文献[16] | 2024 | 中国科学技术大学等 | LLVM | 算子融合、布局转换 |
| 文献[17] | 2024 | 爱丁堡大学 | LLVM | 布局转换 |
| 文献[28] | 2023 | 清华大学 | TVM | 图重写 |
| 文献[23] | 2023 | 北京大学、微软研究院 | TVM | 内存优化 |
| 文献[29] | 2023 | 清华大学、北京科技大学等 | TVM | 计算图构建 |
| 文献[30] | 2022 | 瑞士苏黎世联邦理工学院、卡内基梅隆大学 | LLVM | 静态分析 |
| 文献[31] | 2022 | 达姆施塔特工业大学、华为慕尼黑研究中心 | MLIR | 图划分 |
| 文献[13] | 2022 | 清华大学 | TVM | 冗余算子消除 |
| 文献[24] | 2021 | 清华大学、联发科公司 | TVM | 稀疏化 |
| 文献[22] | 2020 | 加州大学圣地亚哥分校、高通人工智能研究院等 | — | 内存分配 |
| 文献[25] | 2020 | 亚马逊、埃奇科尔蒂克斯公司 | TVM | 量化 |
| 文献[18] | 2020 | 佐治亚理工学院、赛灵思 | Halide | 布局转换 |
| 文献[26] | 2019 | 美国北卡罗来纳州立大学、橡树岭国家实验室 | TensorFlow | 剪枝 |
| 文献[32] | 2019 | 印度理工学院马德拉斯分校 | Soot 框架 | 静态分析、堆分析 |
| 文献[27] | 2019 | 麻省理工学院、Adobe | — | 稀疏 |
| 文献[19] | 2019 | 罗切斯特大学 | LLVM | 布局转换 |
| 文献[20] | 2019 | 亚马逊 | TVM | 布局转换 |
| 文献[14-15] | 2018 | 阿里巴巴 | XLA | 算子融合 |
| 文献[33] | 2018 | 罗切斯特大学 | LLVM | 局部性分析 |
| 文献[21] | 2018 | 都柏林圣三一大学 | GCC 7.2 | 布局转换 |

在算子优化方面，Long 等[14-15]提出了一种新型的算子融合方法——FusionStitching，用于在复杂计算图中高效地融合算子，以减少计算和内存访问开销，提高模型执行效率。

在布局转换方面，Lavaee 等[19]设计了一个 Codestitcher，这是一种过程间基本块代码布局优化器，能够重新排序可执行文件中的基本块，从而改善缓存和 TLB 性能。Liu 等[20]则提出了一种针对计算图中所有卷积运算的布局优化方法，将卷积布局改为 NCHW[$x$]$c$ 形式，其中 $c$ 表示通道分割的子维度，$x$ 则表示子维度的分割大小。通过在硬件特定优化过程中自动调整这些参数，并结合硬件细节（如缓存行大小、矢量化单元大小和内存访问模式），可以实现全局探索并优化布局。Anderson 等[21]介绍了数据布局转换中的最优原语选择问题，并提出了一些有效的解决方案。

在内存优化方面，Ahn 等[22]提出并实现了一种内存感知调度技术，旨在最小化边缘设备上的峰值激活内存占用，为内存受限设备的内存规划提供了新的研究方向。

在稀疏化优化方面，Liao 等[24]采用 im2col plus GEMM 的方法，为 TVM 添加了对稀疏卷积的支持，显著提高了稀疏数据处理的效率。Kjolstad 等[27]则引入了名为工作空间的临时张量，避免了低效的稀疏数据结构访问，进一步提升了计算性能。

此外，Thakur 等[32]提出了一种三阶段分析方法，其在不损失精度的前提下，为大型程序扩展了基于整个程序值上下文的堆分析，增强了对复杂程序行为的理解。Chen 等[33]则描述了一种基于静态并行采样的局部性分析新方法，这项技术能够帮助优化平铺、程序共定位和缓存提示选择等程序优化策略，并有助于分析写局部性和并行局部性，从而进一步提高程序的执行效率。

## 5 后端优化

深度学习编译器的后端主要负责将高级 IR 转化为低级 IR，并针对特定硬件进行优化。由于参数空间庞大，手动寻找最优方案往往既繁琐又耗时，因此如今广泛采用自动调优技术来对这些参数空间进行高效搜索。目前，自动调度优化方法主要包括基于搜索的自动调度优化（如 Ansor[85]）和基于多面体编译技术的自动调度优化（如 MindAKG）。此外，一些硬件厂商还专门为自家硬件架构构建了内核库，其中提供了针对特定硬件的优化方案，从而进一步提升了编译效率与硬件利用率。

### 5.1 基于搜索的自动调优

基于搜索的自动调优常被称为 Auto-Tuning。在传统编译器中，Auto-Tuning 指的是为特定程序和目标架构寻找最优的编译优化方法，包括选择哪些优化技术、确定优化参数集，并安排优化方法的应用顺序，以实现最佳性能。优化方法主要可以分为优化选择和优化顺序两种，优化选择关注于选择哪些优化技术，而优化顺序则着重于如何排列这些技术的应用顺序。

在深度学习编译器中，Auto-Tuning 呈现出两个特点。1）更低的实验开销：优化集中在算子或内核级别，而非整个

程序的优化。同时,要求训练和推理的模拟速度足够快,以降低实验的开销。2)特定的领域结构:针对神经网络算子或内核级别的优化,特别是考虑到深度学习模型中高度的循环化、张量化以及并行化特征,进行定制化的优化。

深度学习编译器中的 Auto-Tuning 的最终目标是使机器自动生成的内核与人工手写的优化内核在性能上达到一致。因此,需要构建一个足够大的搜索空间,确保所有可能的人工优化都涵盖其中,并通过快速搜索算法寻找最优解。通常,Auto-Tuning 过程包括 3 个关键步骤:搜索空间参数化、成本模型构建和搜索算法。

#### 5.1.1 搜索空间参数化

在调度优化问题中,搜索空间通常由可参数化变换的各个可能参数及其取值组合构成,因此,需要通过调度原语来对这些参数进行参数化表示。数据参数、目标参数和优化选项是其中几个关键参数。数据参数主要描述数据的规格,例如输入张量的形状等;目标参数则反映了在优化调度和代码生成过程中需要考虑的硬件特定特征和约束,例如在针对 GPU 的优化中,需要指定共享内存、寄存器大小等硬件相关参数。

在 TVM 中,既有预定义的调度,也允许用户自定义调度相关参数。XLA 则更倾向于通过参数化的优化来进行调度,这不仅与性能紧密相关,还能够在后续调优时以较低的成本进行调整。例如,小批量维度通常映射到 CUDA 中的网格维度,这一参数在自动调优过程中可以被进一步优化,以提升性能表现。

#### 5.1.2 成本模型构建

成本模型用于评估某一参数化调度下的性能,以指导搜索最优的调度策略。其主要评估标准包括运行时间、内存占用和编译后指令数等。实现成本模型的方式主要有 3 种:黑盒模型、基于机器学习的模型和基于模拟的预定义模型。

黑盒模型仅关注最终的执行时间,而不考虑编译任务的具体特征。其优点是模型建立简单,能够快速实现。然而,由于缺乏对任务特征的指导,这种方法可能导致较高的开销,并且无法有效地找到最优解。

基于机器学习的成本模型通过统计方法来预测性能,随着新配置的探索,模型可以自我更新,从而不断提高预测精度。TVM 和 XLA 都采用了机器学习方法,其中 TVM 使用了梯度提升树模型,而 XLA 则采用了前馈神经网络模型。

预定义模型通过构建一个基于编译任务特征的理想化模型,来评估任务的整体性能。这种方法相对高效,在应用时产生的计算开销较小。然而,与基于机器学习的模型相比,预定义模型通常需要大量的工程投入来确保模型的准确性和广泛适用性,特别是在每次面对新的深度学习模型和硬件平台时,需要重新构建模型。虽然预定义模型应用得较少,但在一些特定场景下,由于其开销较低,它依然具有优势。

#### 5.1.3 搜索算法

在确定了初始化参数和搜索空间后,接下来就是在这个搜索空间中找到能够实现最佳性能的参数配置。初始化选项可以随机设置,也可以基于已有的配置,如用户指定的参数或历史最佳配置。在搜索空间的定义上,通常会在自动调优开始之前明确指定。TVM 等编译器框架允许开发人员根据领域知识来定义搜索空间,并为每个硬件目标提供自动化的搜索空间提取机制,以便更加精准地进行优化。

常见的搜索算法包括遗传算法、模拟退火算法和基于机器学习的优化算法等。在 TVM 的自动调度技术中,采用了模拟退火算法,利用其在大规模搜索空间中高效寻找最优解的优势来加速调优过程。

### 5.2 基于多面体模型的自动调优

多面体编译技术的核心在于调度变换,它指的是在满足依赖关系的前提下,将一种调度方式转换为另一种调度方式的过程。在程序自动并行化的领域,这种变换旨在提升程序的并行性和数据局部性。对于多面体编译工具而言,循环嵌套中的语句首先被表示为多维空间中的几何体——空间多面体,然后进行后续的分析和变换。多面体模型上的调度变换实际上是多维空间几何的基变换过程。针对多面体模型的调度算法大致可分为 3 类。

#### 5.2.1 Feautrier 调度算法

Feautrier[86-87] 首先针对多面体模型中的调度变换问题展开了深入研究。他提出的算法将循环迭代抽象为执行序列,并将循环迭代在空间多面体上的坐标表示为对应的时间节点。由于输入的循环嵌套是串行程序,因此每个循环迭代都有唯一的坐标表示,意味着每个迭代都对应着一个唯一的时间节点。

Feautrier 调度算法的核心思想是在满足依赖关系的前提下,为每个循环迭代计算一个新的执行时间节点。经过 Feautrier 调度算法的变换后,循环迭代将根据这些时间节点按序执行。然而,Feautrier 调度算法可能会修改某些循环迭代的时间节点,使得不同的循环迭代拥有相同的时间节点。此时,这些具有相同时间节点的循环迭代可以并行执行,通过将它们分配到不同的处理器上来提升程序的并行性。

#### 5.2.2 基于平面划分的调度算法

Feautrier 调度算法的前提是处理器之间的通信或同步开销相对于计算开销可以忽略不计,通常其计算结果趋向于内层循环并行。然而,在现代多级存储架构的环境下,这一前提条件已不再完全适用,因为通信和同步开销变得更加显著。

为了突破这一局限性,Lim 等[88-89] 提出了一种以外层并行为目标的调度算法。该算法的核心思想是将矩阵抽象为多面体,并将构成该矩阵的每一行视为不同的划分平面。调度变换的过程转化为计算每个划分平面上所有元素的过程。在这个过程中,调度变换必须满足依赖关系,因此每个划分平面与依赖距离向量的乘积必须大于或等于零,从而形成多个不等式,这些不等式构成了计算线性整数规划问题的基础。该算法利用线性整数规划工具来求解一个满足所有不等式的整数解集合,即矩阵的所有元素。在计算划分平面的过程中,算法尽可能地在内层循环满足依赖关系,这样外层划分平面对应的循环便能并行执行,从而提高并行粒度,减少通信和同步的频率,最终优化程序的性能。

#### 5.2.3 Pluto 算法

Lim 等提出的划分算法虽能提高并行性,并减少通信与同步开销,但该优化方法仅将通信数据量优化到某个阶数,未能实现常数系数的最小化。针对这一问题,Bondhugula 等[90]

设计了一种专注于通信数据量优化的代价模型，并基于该模型依次求解调度变换所需的划分平面。

与 Lim 等基于划分平面进行优化的思路类似，该算法同样旨在计算出寻求变基的矩阵中的所有元素。但不同之处在于，Bondhugula 等在原有多个不等式的基础上，额外加入了一个新的不等式，这个不等式由循环边界等程序常数参数所限定，构成了该算法的代价模型。虽然这一代价模型的引入使得线性整数规划过程中的未知变量数量增加，但最终计算出的矩阵与 Lim 等所得到的矩阵不同，得到的调度变换结果也显著减少了通信量。这一调度算法是 Pluto 编译器的核心内容，因此被称为 Pluto 算法。此外，由于该算法中引入了代价模型，它也常被称为基于代价模型的调度变换算法。

### 5.3 程序执行

生成目标程序后，通常有两种运行方式：静态编译和动态解释。静态编译，即提前编译（Ahead of Time，AOT），是指程序在执行之前完全编译为机器码；而动态解释则是指程序边编译边运行，这种方式通常称为即时编译（Just in Time，JIT）。

JIT 编译器能够在运行时动态生成可执行代码，并根据运行时的知识进行优化，从而提升代码性能。相比之下，AOT 编译器则是在程序执行前完成所有可执行二进制文件的生成，这使得 AOT 编译器能够在静态分析时具备更广泛的优化空间，还能用于嵌入式平台的交叉编译。尽管 JIT 编译可以提高动态编程语言的执行效率，但与提前编译的语言相比，其性能仍然存在差距。现有的 JIT 编译器通过类型专门化利用类型单态，并使用运行时检查确保程序的正确性，但这些检查通常会带来额外的性能开销。表 5 对 JIT 和 AOT 两种运行方式进行了对比。

表 5 JIT 与 AOT 的对比

Table 5 Comparison of JIT and AOT

| 运行方式 | 优点 | 缺点 |
|---|---|---|
| JIT | 1）可以根据当前硬件情况实时编译生成最优机器指令<br>2）可以根据当前程序的运行情况生成最优的机器指令序列<br>3）当程序需要支持动态链接时，只能使用 JIT<br>4）可以根据进程中内存的实际情况调整代码，使内存能够得到更充分的利用 | 1）编译需要占用运行时资源，会导致进程卡顿<br>2）编译占用运行时间，对某些代码编译优化不能完全支持，需在流畅和时间间权衡<br>3）编译准备和识别频繁使用的方法需要占用时间，初始编译不能达到最高性能 |
| AOT | 1）在程序运行前编译，可以避免在运行时的编译性能消耗和内存消耗<br>2）可以在程序运行初期就达到最高性能<br>3）可以显著加快程序的启动 | 1）不在程序运行前编译，会使程序安装时间增加<br>2）牺牲高级语言的一致性问题<br>3）将提前编译的内容保存，会占用更多的内存 |

### 5.4 后端优化相关研究现状

近些年的后端优化的研究进展如表 6 所列。

在调度优化方面，循环优化和存储优化是研究的两个主要方向。针对循环优化，Peng 等[34]提出了一种自动调优方法，其针对常用多面体编译器 Pluto 在默认循环调度和分块大小上的性能不足，计算多种合法置换，并通过置换和分块大小构成的配置空间来优化循环程序。Rocha 等[35]提出了一种新的直线码循环滚动技术 RoLAG，其利用自下而上的方式对齐 SSA 图以识别同构代码，并将这些代码滚动到一个循环中，从而提升程序效率。Behroozi 等[36]介绍了一种配置文件引导的编译器方法 Loner，其通过优化空闲的向量数据路径来提升标量整数循环的性能。Rocha 等[37]提出了一种新的矢量化感知循环展开方法 VALU，旨在与 SLP 矢量化紧密耦合，从而实现更高效的循环展开。Zhao 等[39]提出了一种基于仿射关系的自动编译方法，专门用于并行化和优化动态计数循环。Sioutas 等[40]提出的优化算法，通过分析循环嵌套的算法描述，决定是否应优先优化时间局部性或空间局部性，同时还考虑了硬件预取策略。

表 6 后端优化技术调研

Table 6 Backend optimization techniques research

| 来源 | 年份 | 研发团队 | 编译组件 | 优化类型 |
|---|---|---|---|---|
| 文献[12] | 2024 | 北京软件开发环境国家重点实验室、北京航空航天大学 | MLIR | 扩展后端架构 |
| 文献[49] | 2024 | 美国佐治亚理工学院 | LLVM，CompilerGym | 相位排序（使用 RL） |
| 文献[16] | 2024 | 中国科学技术大学等 | LLVM | 控制流优化、扩展后端架构、访存优化 |
| 文献[17] | 2024 | 爱丁堡大学 | LLVM | 扩展后端架构 |
| 文献[42] | 2024 | 并行计算机工程技术研究中心 | — | 存储优化 |
| 文献[51] | 2024 | 中国科学院大学等 | MLIR | 自动调优、自动配置机制 |
| 文献[28] | 2023 | 清华大学 | TVM | 数据移动、内存优化 |
| 文献[75] | 2023 | ONERA、Google | MLIR | 并行化 |
| 文献[77] | 2023 | 清华大学、微软研究院 | PyTorch，Rammer | 控制流优化 |
| 文献[47] | 2023 | 郑州大学、信息工程大学 | TVM | 自动调优、成本模型 |
| 文献[34] | 2023 | 中国科学院重庆绿色智能技术研究院、中国科学院大学重庆学院 | Pluto | 自动调优、多面体技术、循环优化 |
| 文献[74] | 2023 | Sophgo Inc. | MLIR | 扩展后端架构 |
| 文献[28] | 2023 | 清华大学 | TVM | 并行化 |
| 文献[67] | 2022 | 多伦多大学、微软研究院等 | TVM，Rammer | 内核生成 |
| 文献[43] | 2022 | 美国佐治亚理工学院 | LLVM | 内存优化、访存优化 |
| 文献[54] | 2022 | 华为公司 | MindSpore | 多面体技术 |
| 文献[35] | 2022 | 爱丁堡大学、曼彻斯特大学 | LLVM | 循环滚动 |
| 文献[13] | 2022 | 清华大学 | TVM | 内存优化 |

（续表）

| 来源 | 年份 | 研发团队 | 编译组件 | 优化类型 |
|---|---|---|---|---|
| 文献[36] | 2022 | 美国密歇根大学安娜堡分校 | LLVM | 优化标量整数循环 |
| 文献[76] | 2022 | 加州大学伯克利分校、上海交通大学等 | Jax+XLA | 并行化 |
| 文献[52] | 2022 | 美国密歇根大学安娜堡分校、韩国首尔汉阳大学等 | GCC，TVM，OpenCL | 自动调优策略 |
| 文献[48] | 2022 | 韩国浦项科技大学 | PyTorch | 自动调优的成本模型 |
| 文献[50] | 2022 | 西北大学、英国利兹大学等 | LLVM | 使用 RL、自动调优 |
| 文献[55] | 2021 | 瑞士苏黎世联邦理工学院、美国卡内基梅隆大学 | LLVM | 多面体技术 |
| 文献[56] | 2021 | 法国里昂大学等 | — | 多面体技术 |
| 文献[72] | 2021 | 韩国浦项科技大学、韩国首尔延世大学 | LLVM | 扩展后端架构 |
| 文献[57] | 2021 | 郑州数学工程与高等计算国家重点实验室、华为 | MindSpore | 多面体技术、扩展后端架构 |
| 文献[53] | 2021 | 华盛顿大学、德克萨斯大学 | TVM | 自动调优、自动生成量化机器学习推理有效代码 |
| 文献[45] | 2021 | 巴西马林加州立大学、巴西米纳斯联邦大学 | LLVM | 优化搜索空间（遗传算法） |
| 文献[62] | 2021 | 加拿大蒙特利尔麦吉尔大学 | — | JIT/AOT |
| 文献[69] | 2020 | 瑞士苏黎世联邦理工学院、摩德纳大学 | LLVM | 扩展后端架构 |
| 文献[46] | 2020 | 北京大学、微软研究院等 | — | 自动调度空间 |
| 文献[18] | 2020 | 佐治亚理工学院、赛灵思公司 | Halide | 循环优化、扩展后端架构 |
| 文献[37] | 2020 | 英国爱丁堡大学、英特尔等 | LLVM | 循环展开、矢量化 |
| 文献[38] | 2020 | 加州大学伯克利分校、英特尔实验室 | LLVM | 使用 RL，处理循环矢量化 |
| 文献[70] | 2019 | 中国科学院计算技术研究所、澳大利亚悉尼新南威尔士大学 | LLVM | 扩展后端架构 |
| 文献[58] | 2019 | 美国麻省理工大学、意大利米兰理工大学等 | — | 多面体技术 |
| 文献[68] | 2019 | 德国埃尔朗根-纽伦堡大学等 | Hipacc | 内核融合 |
| 文献[63] | 2019 | 瑞士甲骨文实验室等 | Graal | JIT |
| 文献[71] | 2019 | 数学工程与先进计算国家重点实验室 | Open64 | 扩展后端架构 |
| 文献[78] | 2019 | 美国斯坦福大学 | Spatial | 多目标优化 |
| 文献[41] | 2019 | 英特尔公司 | LLVM | 指令优化、内存优化等 |
| 文献[44] | 2018 | 美国斯坦福大学、瑞士洛桑联邦理工学院 | — | 内存优化、任务划分、扩展后端架构 |
| 文献[64] | 2018 | 脸书公司 | GCC | JIT |
| 文献[59] | 2018 | 德国萨尔州大学 | Polly | 多面体技术 |
| 文献[65] | 2018 | 美国罗切斯特大学、莱斯大学等 | | JIT |
| 文献[66] | 2018 | 约翰尼斯开普勒大学、奥地利林茨甲骨文实验室 | GraalVM | JIT |
| 文献[39] | 2018 | 法国巴黎高级学院 | PPCG | 循环优化、多面体技术 |
| 文献[40] | 2018 | 荷兰埃因霍温理工大学 | Halide | 循环优化 |
| 文献[60] | 2018 | 印度科学研究所、法国巴黎高等师范学院等 | Pluto | 多面体技术 |
| 文献[61] | 2018 | 法国国家信息与自动化研究所、瑞士苏黎世联邦理工学院 | LLVM，Polly | 多面体技术 |
| 文献[73] | 2018 | 英特尔公司 | LLVM | 扩展后端架构 |

在存储优化方面，Damani 等[43]提出了一种新的编译器内存访问优化方法，其能够静态地识别并合并位于同一位置的内存访问，从而减少线程迁移的次数，提升内存访问效率。

在基于搜索的自动调优方面，Yang 等[47]提出了一种基于特征重要性的自动调优方法，使得模型在推理过程中获得了高达 15%的性能提升。Han 等[49]利用休眠信息来指导搜索过程，显著缩小了搜索空间，同时保留了最优解。通过排除休眠变换，这种方法能够有效修剪搜索空间，并确保最优解不受影响。Zhang 等[51]设计了一种面向深度学习编译优化的自动配置机制 AutoConfig，该机制能够根据不同任务和硬件平台，通过静态信息提取和动态开销测量相结合的方式进行综合分析，从而实现算法的自动选择与调优。Park 等[52]提出了一种智能自动调优策略 SRTuner，其利用公开的重要优化交互信息来搜索最佳优化配置。Cowan 等[53]将量化值的布局选择集成到机器学习编译器的调度阶段，并允许根据平铺和并行化决策进行优化，从而提升了编译效率和执行性能。

在多面体模型的相关研究中，Bastoul 等[54]将多面体框架解耦为线性和非线性分量，并引入了约束树抽象。该约束树能够通过非线性优化器生成，并被注入到多面体优化过程中，以帮助构建更优的解。

在代码生成及执行方面，Krolik 等[62]提出了一种基于 GPU 的 AOT/JIT 混合方案 r3d3，该方案有效减少了中间数据的写出和代码编译时间。Brock 等[65]提出了一种编译策略 PAYJIT，该策略根据方法大小调整编译热度阈值，优化了编译速度。David 等[66]提出了一种新的方法 DBDS，旨在确定应执行哪些复制操作以提升性能。该方法基于复制模拟，允许编译器评估每个潜在复制的成功度量，从而选择最有希望的候选进行优化。Qiao 等[68]则通过将融合问题转化为图划分问题，提出了一种基于最小分割技术的递归搜索方法，用于解决可融合核的选择问题。

## 6 深度学习编译器研究现状

### 6.1 工业界的深度学习编译器

工业界的深度学习编译器如表 7 所列。

TVM 是一个开源的深度学习编译器，它提供统一的 IR 堆栈和自动化代码生成方法，支持 CPU，GPU 和 ARM 等多种硬件架构。该项目于 2017 年由华盛顿大学的陈天奇等人发起，最初主要聚焦于对 CPU 和 GPU 等通用硬件的优化。2019 年左右，TVM 逐渐扩展到 FPGA 和 ASIC 等硬件平台，并引入了自动调优技术。TVM 的自动调优过程包括两个核心模块：基于模板的 AutoTVM 和无模板的 AutoScheduler（Ansor）。在后端支持方面，TVM 不仅支持常见的 CPU 和 GPU 硬件，还通过 VTA 模块实现了对自定义硬件后端的支持。

nGraph 由英特尔于 2017 年推出，最初主要支持 CPU 和 GPU 后端。随后，nGraph 逐步扩展，开始支持英特尔的 Nervana 神经处理器以及 FPGA 等硬件平台。

Glow 由 Facebook AI Research 团队于 2018 年开发，是一款专为多平台生成高度优化代码的深度学习编译器。Glow 的架构设计简洁，基本上可以视为硬件加速器之上的抽象层，其核心目标是为模型提供高效的编译与执行引擎。如今，Glow 已成为 PyTorch 机器学习框架的重要组成部分，支持从云端到台式机，再到微控制器等多种软硬件平台。

XLA(Accelerated Linear Algebra)是 TensorFlow 的图优化编译器，并在 TensorFlow 2.0 版本中作为正式特性发布。TensorFlow 的计算模型基于计算图，该架构的一个显著优点是，在定义计算图时，开发者不必提前指定张量的形状和数值，而是可以在图执行时通过分析操作之间的依赖关系动态推导张量的形状。这种灵活性对深度学习模型开发者非常友好，但给编译器带来了挑战。由于张量形状的动态特性，编译器很难确定如何高效地编译相关操作，也很难在执行前准确估算所需的内存大小，影响了内存的高效利用。

MindSpore 是华为于 2019 年 8 月正式发布的深度学习框架，其整体架构分为 4 个层次：模型层、表达层、编译优化层和运行时层。在这个框架中，编译器 MindCompiler 是核心组件，其通过全场景统一的中间表示(MindIR)这个媒介，将前端模型表达编译成更高效的底层执行代码。

MLIR(Multi-Level Intermediate Representation)是由谷歌开发的一种用于编译器开发的统一软件框架，它提供了一种可扩展的、中立的 IR，能够支持多种编程语言和硬件平台。通过这种灵活的 IR，MLIR 使得编译器能够在不同的层次上进行优化，并轻松地适应不同的硬件架构。基于 MLIR 框架，近年来涌现出多款成熟的深度学习编译器，其中包括 Triton，MegCC，MLIR-AIE 和 TPU-MLIR[74]等。

除了上述编译器之外，工业界还涌现出了一些其他的编译器。例如，寒武纪针对其 MLU 芯片的 CNCC 编译器、ACT 实验室的 NPiler 和阿里巴巴的 AGSpeed 等，这些编译器也在特定硬件平台上提供了优化支持。

表 7 工业界中的深度学习编译器

Table 7 Deep learning compilers in industry

| 名称 | 开发团队 | 年份 | IR | 前端支持 | 后端支持 | 自动调优 |
|---|---|---|---|---|---|---|
| MegCC | 旷视 | 2022 | Grapg IR，Kernel IR | MegEngine，ONNX | CPU | — |
| BladeDISC | 阿里巴巴 | 2022 | MHLO IR | TensorFlow，PyTorch | CPU，GPU 等 | 支持 |
| Triton | OpenAI | 2021 | Triton IR | PyTorch，TensorFlow，MXNet 等 | GPU | AutoTune |
| Mind Compiler | 华为 | 2019 | Mind IR | TensorFlow，PyTorch | NPU，GPU，CPU 及嵌入式设备等 | MindSpore AKG，AutoAugment |
| Glow | Facebook | 2018 | — | TensorFlow，PyTorch，Caffe2，ONNX 等 | CPU，GPU | 不支持 |
| XLA | Google | 2017 | HLO IR | Tensorflow，JAX，Julia，PyTorch 和 Nx 等 | CPU，GPU，TPU | 支持 |
| TVM | 陈天奇等 | 2017 | Realy IR，T IR | PyTorch，TensorFlow，MXNet，ONNX，Keras，TFLite，CoreML，YOLO，PaddlePaddle，OneFlow | CPU，GPU，ARM，FPGA 等 | AutoTVM，AutoScheduler |

### 6.2 学术界的深度学习编译器

近些年，学术界出现了大量端到端深度学习编译器的研究，如表 8 所列。

MagPy[91]不以计算图作为输入，而是直接面向用户编写的 Python+PyTorch 程序，通过监控程序执行来收集关键的运行时信息，自动生成完整的算子图。实验结果表明，MagPy 可以成功地将 1191 个真实用户程序中的 93.40%实例化为完整的算子图。

POWERFUSION[28]是一款高性能张量编译器，专注于通过计算与数据移动的联合优化，为内存密集型运算符生成高效代码。POWERFUSION 使用 GIR 表示深度神经网络程序，该表示包括描述计算任务、数据移动模式和并行策略的原语。这些信息进一步被整合为指令级数据流图，支持在不同硬件平台上通过搜索多种内存访问模式和计算操作，生成内存高效的代码，从而实现整体优化。此外，GIR 的设计目标是通过显式表达数据移动模式和细粒度数据依赖性，优化内存性能。针对 GIR，提出了一套优化策略，包括基于 GIR 的搜索与转换方法，旨在最大限度地减少内存访问并提升内存效率。

TVM_T[29]是基于开源编译器框架 TVM 开发的第一个端到端编译器，专门支持神经网络训练的优化与加速。与传统的 TVM 编译器主要面向推理优化不同，TVM_T 针对神经网络的训练过程进行了全面的优化，涵盖了从模型定义到训练执行的整个流程。

PIMFlow[92]是一种专门针对支持 PIM(Processing-in-Memory)技术的 GPU 内存设计的加速框架，旨在优化 CNN 模型的执行效率。PIMFlow 通过转换模型图，创建跨 GPU 和 PIM 节点之间的并行性，探索多种任务和数据并行执行策略，从而显著缩短模型的执行时间。该框架不仅支持高效的任务分配和数据分布，还为 DRAM 与 PIM 之间的协同工作提供了优化的代码生成后端和执行引擎。通过精细的资源调度和内存计算协同，PIMFlow 实现了计算与存储的高效结合，推动了深度学习任务在异构内存架构上的加速和优化。

Welder[23]，Cocktailer[77]，Roller[67]和 Rammer[46]是微软研究院与海内外合作者近些年来设计的 4 款深度学习编译器，旨在提升硬件并行利用率、编译效率及全局访存效率，保证控制流的高效执行等。

Rammer 在编译时为 DNN 生成了有效的静态时空调度，最大限度地减少了调度开销。同时，为计算任务和硬件加速器提出的几个新的、与硬件无关的抽象，使 Rammer 获得了更丰富的调度空间，实现了算子间和算子内的协同调度，从而可以全面利用并行性，更好地探索空间并找到有效的调度，大幅提高硬件利用率。Roller 在考虑内存特性的前提下，像铺地

板一样把高维的张量数据平铺到二维的内存中，找到最优的块(tile)大小。同时，它还封装了与底层加速器的硬件特性一致的张量形状，通过限制形状选择来实现高效编译。Roller也已被用于微软内部开发的自定义DNN内核上。在实际开发中验证了Roller可以显著缩短开发周期的优越性能。Welder通过链接不同的算子，可以让数据块以流水线的方式处理，大大降低访存量，全面优化由通用算子组成的端到端DNN模型的内存访问效率。Cocktailer可以在硬件加速器上共同优化控制流和数据流的执行，并通过一种新的抽象来统一包括控制流和数据流的人工智能模型的表示；使用启发式策略找到了有效的调度方案，且能够自动将控制流移动到设备内核中，进而实现了跨控制流边界的优化。

表8 学术界中的深度学习编译器

Table 8 Deep learning compilers in academia

| 来源 | 年份 | 开发团队 | 组件/框架 | 目标硬件 | 开源 | 类型 |
|---|---|---|---|---|---|---|
| MagPy[91] | 2024 | 清华大学 | PyTorch | CPU(AMD EPYC 7742)<br>GPU(NVIDIA A100PCIE-40GB) | √ | 面向Python |
| POWERFUSION[28] | 2023 | 清华大学 | TVM | GPU(NVIDIA A100，AMD MI100)<br>加速卡(MLU-370) | × | 张量编译器 |
| TVM_T[29] | 2023 | 清华大学、<br>北京科技大学等 | TVM | CPU(Intel Xeon Gold 6271C)<br>GPU(NVIDIA RTX 2080Ti) | × | — |
| PIMFlow[92] | 2023 | 韩国浦项科技大学 | TVM | 存算一体架构 | × | — |
| Welder[23] | 2023 | 北京大学、<br>微软研究院 | TVM，Rammer，Roller | CPU(Intel Xeon)<br>GPU(NVIDIA V100等、AMD MI50)<br>加速卡(Graphcore IPU) | √ | — |
| Cocktailer[77] | 2023 | 清华大学、<br>微软研究院 | PyTorch，Rammer | CPU(Intel Xeon)<br>GPU(NVIDIA V100，AMD MI100) | √ | — |
| Roller[67] | 2022 | 微软研究院等 | 多伦多大学、<br>TVM，Rammer | CPU(Intel Xeon)<br>GPU(NVIDIA V100，K80，AMD MI50)<br>加速卡(Graphcore IPU) | √ | — |
| SPNC[31] | 2022 | 达姆施塔特工业大学、<br>华为慕尼黑研究中心 | MLIR | CPU(AMD Ryzen，Intel Xeon)<br>GPU(Nvidia RTX 2070 super) | × | 针对和积网络 |
| 文献[4] | 2022 | 美国波特兰州立大学 | TVM | 加速卡(NVDLA，BM1880) | × | — |
| KunlunTVM[13] | 2022 | 清华大学 | TVM | CPU(Xeon Gold 6271C)<br>加速卡(Kunlun K200) | × | 异构编译器 |
| SuperSonic[50] | 2022 | 西北大学、利兹大学等 | LLVM，TVM | CPU(Intel Xeon，AMD EPYC 7532)<br>GPU(NVIDIA RTX 2080 Ti) | √ | — |
| Alpa[76] | 2022 | 加州大学伯克利分校、<br>上海交通大学等 | Jax+XLA | GPU(NVIDIA V100) | √ | — |
| 文献[56] | 2021 | 法国里昂大学、<br>Xtrem Logic SAS France | — | FPGA(Xilinx VCU1525板上的XCVU9P-L2FSGD2104E) | × | 数据感知过程网络 |
| Rammer[46] | 2020 | 北京大学、<br>微软研究院等 | — | CPU(Intel Xeon)<br>GPU(NVIDIA V100，AMD MI50)<br>加速卡(Graphcore IPU) | √ | — |
| NeuroVectorizer[38] | 2020 | 加州大学伯克利分校、<br>英特尔实验室 | LLVM | — | √ | — |
| 文献[93] | 2019 | 墨西哥锡那罗亚库<br>利亚坎自治大学 | — | — | × | 多目标编译器 |
| TIRAMISU[58] | 2019 | 麻省理工大学、<br>米兰理工大学等 | — | CPU(Intel Xeon E5-2680 v3)<br>GPU(NVIDIA Tesla K40) | × | 多面体模型 |
| Wootz[26] | 2019 | 北卡罗来纳州立大学、<br>橡树岭国家实验室 | TensorFlow | CPU(AMD Opteron 6274)<br>GPU(NVIDIA K20X) | × | — |

文献[31]和文献[56]分别针对和积网络和数据感知过程网络构建了编译支持。Wang等[4]提出了一个扩展TVM，以支持具有卷积核的深度学习加速器的通用框架。KunlunTVM[13]在昆仑芯片上支持训练和推理任务。实验结果表明，与现有支持昆仑芯片的框架飞桨相比，KunlunTVM实现了高达5倍的训练性能提升，并且该编译器中所提出的算子消除等方法可扩展到TVM中的不同后端中。SuperSonic[50]用于自动化RL架构搜索和参数调优过程，使RL更容易集成到编译器中。Alpa[76]设计了许多编译PASS，通过生成统一数据、操作符和pipe并行性的执行计划，自动化大型深度学习模型的模型并行训练。NeuroVectorizer[38]提出一种使用深度强化学习的循环矢量化处理方法。Wootz[26]注重于CNN的剪枝过程，设计了一种基于压缩的算法来有效地识别要预训练的CNN层集，揭示了在一组修剪过的CNN模型的训练中可组合性的存在，并指出了计算重用的机会，最大限度地提高了它们在CNN修剪中的重用效益。

## 7 总结与展望

本文从深度学习编译器的通用架构出发，进行了深入的分析和全面的探讨。首先，介绍了当前深度学习编译器普遍采用的通用架构，重点分析了多级中间表示、前端和后端的设计结构，并详细阐述了各类前后端优化技术的应用与发展。接着，回顾了近年来编译优化技术的相关研究，深入剖析了当前编译优化的研究热点与发展趋势。最后，总结并介绍了工业界中几款成熟的深度学习编译器，同时对学术界的最新研究成果进行了汇总与调研。

在详细调研深度学习编译器现状的基础上，本文对其未来发展做出如下展望。

1)解决生态碎片化问题

当前的深度学习编译器生态面临着严重的碎片化问题。随着各大厂商推出自有芯片,如何将模型转换为能够在不同硬件上执行的机器码,并充分发挥硬件算力,成为了一个亟需应对的挑战。为了满足这一需求,每家厂商几乎都推出了自己的编译器和框架,甚至是完整的软件栈。然而,各个厂商之间缺乏统一的标准和范式,导致深度学习编译器生态的高度碎片化。谷歌提出的 MLIR 在一定程度上缓解了这一问题,提供了一个统一的解决方案框架,但具体的实施与应用仍需工业界和学术界共同努力,才能在实际应用中实现真正的整合与优化。

2)构建新体系架构的编译支持及优化

当前,随着硬件加速器的迅猛发展,许多新型计算架构不断涌现,如拟态计算、存算一体和量子计算等。近年来,针对这些新兴架构的研究逐渐增多。然而,如何将日益复杂的模型高效地部署到这些新型计算体系中,仍然是一个亟待解决的问题,需要进一步的探索与创新。

3)分布式并行计算

在当前大模型发展的趋势下,模型大小已经超过加速器内存容量,单个设备上无法加载完整的模型。因此,如何对模型进行高效划分,并将划分后的任务合理映射到硬件架构上,使用分布式并行的方法最大化地利用硬件资源,仍然是一个亟待解决的核心问题。目前针对深度学习模型的分布式并行方法有数据并行、流水线并行和张量并行等,这些并行策略各有优劣,分布式并行计算仍需要继续深入研究与探索。

4)多面体技术

将多面体模型与自动调优技术相结合,设计深度学习编译器,是一个具有广阔前景的研究方向。一方面,通过重用已有的配置,自动调优可以有效降低多面体 JIT 编译的开销。另一方面,多面体模型可用于自动调度,从而缩小自动调优过程中的搜索空间,提高编译效率和性能。

5)对动态图的支持

许多深度学习框架(如 PyTorch 和 TensorFlow 2.0)支持动态图,这为模型的构建和调试带来了更大的灵活性。然而,目前大多数深度学习编译器主要集中在静态图的优化上,因此,如何更好地支持动态图,已成为满足不同框架和应用需求的一个关键挑战。

6)自动调优

现有的自动调优技术主要聚焦于单个算子的优化。然而,局部最优的结果并不一定能带来全局最优。例如,对于应用于不同数据布局的两个相邻算子,可以通过联合调优进行优化,而无需在两者之间进行额外的内存转换,这样可以避免不必要的性能损失。

7)稀疏计算

随着深度学习模型的复杂性和规模的增加,计算资源的消耗也越来越大。稀疏计算可以通过有效利用数据和计算的稀疏性,降低计算量、存储需求以及计算延迟。深度学习编译器正通过优化稀疏矩阵存储格式(如 CSR、块稀疏)和结合硬件特性,显著提升了稀疏计算效率。例如,百度的 Paddle 在 API 设计上,稀疏计算的使用与稠密计算保持高度一致,用户可以通过简单的 API 调用实现稀疏计算的功能。在未来,如何将稀疏计算与这些动态计算图相结合,通过智能算法自动选择稀疏化策略以及不同硬件上稀疏化的执行方式,仍有待继续深入研究与探索。

8)知识蒸馏

知识蒸馏作为一种模型压缩技术,其核心目标是通过知识迁移过程将复杂教师模型中的隐式知识迁移至轻量化学生模型,以在降低模型复杂度的同时保持其推理性能,例如,DeepSeek-R1 系列模型在训练过程中便使用了知识蒸馏技术[94]。然而,现有研究多聚焦于算法层面的优化,而如何将知识蒸馏与深度学习编译器深度结合,实现模型压缩与硬件执行效率的协同优化,仍存在诸多挑战。未来潜在研究方向包括编译器驱动的蒸馏架构搜索和动态编译与知识迁移的耦合优化等。

## 参 考 文 献


[1] ZHANG X, SUN N, FANG C, et al. Predoo: precision testing of deep learning operators[C]// Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021: 400-412.

[2] SUN R Y. Optimization for Deep Learning: An Overview[J]. Journal of the Operations Research Society of China, 2020(3): 1-46.

[3] LI M, LIU Y, LIU X, et al. The Deep Learning Compiler: A Comprehensive Survey[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(3): 708-727.

[4] WANG Y, XIE F. Extending Tensor Virtual Machine to Support Deep-Learning Accelerators with Convolution Cores[C]// 2022 26th International Conference on Engineering of Complex Computer Systems (ICECCS). 2022: 189-194.

[5] LÜCKE M, STEUWER M, SMITH A. Integrating a functional pattern-based IR into MLIR[C]// Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2021: 12-22.

[6] CHELINI L, DREBES A, ZINENKO O, et al. Progressive raising in multi-level IR[C]// Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2021: 15-26.

[7] PIZZUTI F, STEUWER M, DUBACH C. Generating fast sparse matrix vector multiplication from a high level generic functional IR[C]// Proceedings of the 29th International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2020: 85-95.

[8] KLOPP D, ERDWEG S, PACAK A. A Typed Multi-level Datalog IR and Its Compiler Framework[C]// Proceedings of the ACM on Programming Languages. 2024.

[9] GROSSMAN A, PAEHLER L, PARASYRIS K, et al. Compile: A large ir dataset from production sources[J]. arXiv: 2309.15432, 2023.

[10] FEHR M, NIU J, RIDDLE R, et al. IRDL: an IR definition language for SSA compilers[C]// Proceedings of the 43rd ACM

SIGPLAN International Conference on Programming Language Design and Implementation. New York, NY, USA: Association for Computing Machinery, 2022: 199-212.

[11] BHAT S, GROSSER T. Lambda the ultimate SSA: optimizing functional programs in SSA [C] // Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2022: 168-178.

[12] LI M, LIU Y, CHEN B, et al. Building a domain - specific compiler for emerging processors with a reusable approach[J]. SCIENCE CHINA Information Sciences, 2023, 67(1): 112101. 1-112101. 19.

[13] ZENG J, KOU M, YAO H. KunlunTVM: A Compilation Framework for Kunlun Chip Supporting Both Training and Inference [C] // Proceedings of the Great Lakes Symposium on VLSI 2022. New York, NY, USA: Association for Computing Machinery, 2022: 299-304.

[14] LONG G, YANG J, LIN W. FusionStitching: Boosting Execution Efficiency of Memory Intensive Computations for DL Workloads[J]. arXiv: 1911. 11576, 2019.

[15] LONG G, YANG J, ZHU K, et al. FusionStitching: Deep Fusion and Code Generation for Tensorflow Computations on GPUs [J]. arXiv: 1811. 05213, 2018.

[16] SHEN L, ZHOU W H, WANG F, et al. swLLVM: An Optimizing Compiler for the New Generation of Sunway Supercomputers[J]. Journal of Software, 2024, 35(5): 2359-2378.

[17] MAVROGEORGIS N, VASILADIOTIS C, MU P, et al. UNIFICO: Thread Migration in Heterogeneous-ISA CPUs without State Transformation[C] // Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2024: 86-99.

[18] CHATARASI P, NEUENDORFFER S, BAYLISS S, et al. Vyasa: A High-Performance Vectorizing Compiler for Tensor Convolutions on the Xilinx AI Engine[J]. arXiv: 2006. 01331, 2020.

[19] LAVAEE R, CRISWELL J, DING C. Codestitcher: inter-procedural basic block layout optimization[C] // Proceedings of the 28th International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2019: 65-75.

[20] LIU Y, WANG Y, YU R, et al. Optimizing CNN Model Inference on CPUs[C] // 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019: 1025-1040.

[21] ANDERSON A, GREGG D. Optimal DNN primitive selection with partitionedboolean quadratic programming [C] // Proceedings of the 2018 International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery, 2018: 340-351.

[22] AHN B H, LEE J, LIN J M, et al. Ordering Chaos: Memory-Aware Scheduling of Irregularly Wired Neural Networks for Edge Devices[C] // Proceedings of Machine Learning and Systems. 2020: 44-57.

[23] SHI Y, YANG Z, XUE J, et al. WELDER: Scheduling Deep Learning Memory Access via Tile-graph [C] // 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23). 2023.

[24] LIAO HH, LEE C L, LEE J K, et al. Support Convolution of CNN with Compression Sparse Matrix Multiplication Flow in TVM[C] // 50th International Conference on Parallel Processing Workshop. New York, NY, USA: Association for Computing Machinery, 2021: 1-7.

[25] JAIN A, BHATTACHARYA S, MASUDA M, et al. Efficient Execution of Quantized Deep Learning Models: A Compiler Approach[J]. arXiv: 2006. 10226. 2020.

[26] GUAN H, SHEN X, LIM S H. Wootz: a compiler-based framework for fast CNN pruning via composability[C] // Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: Association for Computing Machinery, 2019: 717-730.

[27] KJOLSTAD F, AHRENS W, KAMIL S, et al. Tensor Algebra Compilation with Workspaces[C] // 2019 IEEE/ACM International Symposium on Code Generation and Optimization(CGO). 2019: 180-192.

[28] MA Z, WANG H, XING J, et al. PowerFusion: A Tensor Compiler with Explicit Data Movement Description and Instruction-level Graph IR[J]. arXiv: 2307. 04995, 2023.

[29] ZENG J, KOU M Y, ZHENG X Y, et al. TVM_T: A High-Performance Neural Network Training Compiler Based on TVM [J]. Science China Information Sciences, 2023, 53(12): 2458-2471.

[30] RIVERA J, FRANCHETTI F, PÜSCHEL M. A compiler for sound floating-point computations using affine arithmetic[C] // Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2022: 66-78.

[31] SOMMER L, AXENIE C, KOCH A. SPNC: an open-source MLIR-based compiler for fast sum-product network inference on CPUs and GPUs[C] // Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2022: 290-300.

[32] THAKUR M, NANDIVADA V K. Compare less, defer more: scaling value-contexts based whole-program heap analyses[C] // Proceedings of the 28th International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2019: 135-146.

[33] CHEN D, LIU F, DING C, et al. Locality analysis through static parallel sampling[C] // Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: Association for Computing Machinery, 2018: 557-570.

[34] PENG C, LIU Q Z, CHEN C B. Loop Permutation and Auto-Tuning under the Polyhedral Model[J]. Computer Engineering and Science, 2023, 45(12): 2121-2134.

[35] ROCHA R C O, PETOUMENOS P, FRANKE B, et al. Loop rolling for code size reduction [C] // Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2022: 217-229.

[36] BEHROOZI A, PARK S, MAHLKE S. Loner: utilizing the CPU vectordatapath to process scalar integer data[C]// Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2022: 205-217.

[37] ROCHA R C O, PORPODAS V, PETOUMENOS P, et al. Vectorization-aware loop unrolling with seed forwarding[C]// Proceedings of the 29th International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2020: 1-13.

[38] HAJ-ALI A, AHMED N K, WILLKE T, et al. NeuroVectorizer: end-to-end vectorization with deep reinforcement learning [C]// Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery, 2020: 242-255.

[39] ZHAO J, KRUSE M, COHEN A. A polyhedral compilation framework for loops with dynamic data-dependent bounds [C]// Proceedings of the 27th International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2018: 14-24.

[40] SIOUTAS S, STUIJK S, CORPORAAL H, et al. Loop transformations leveraging hardware prefetching[C]// Proceedings of the 2018 International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery, 2018: 254-264.

[41] CHANDRASEKHAR A, CHEN G, CHEN P Y, et al. IGC: the open source Intel graphics compiler[C]// Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization. Washington, DC, USA: IEEE Press, 2019: 254-265.

[42] FANG Y F, LI Y B, DONG E M, et al. Memory Access and Communication Fusion Compilation Optimization for the Sunway Many-Core Processor[J]. Journal of Software, 2024, 35(6): 1-20.

[43] DAMANI S, BARUA P, SARKAR V. Memory access scheduling to reduce thread migrations[C]// Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2022: 144-155.

[44] KOEPLINGER D, FELDMAN M, PRABHAKAR R, et al. Spatial: a language and compiler for application accelerators[C]// Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: Association for Computing Machinery, 2018: 296-311.

[45] SILVA A FD, DE LIMA B N B, PEREIRA F M Q. Exploring the space of optimization sequences for code-size reduction: insights and tools[C]// Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2021: 47-58.

[46] MA L, XIE Z, YANG Z, et al. RAMMER: Enabling Holistic Deep Learning Compiler Optimizations with rTasks[C]// Operating Systems Design and Implementation. 2020.

[47] YANG H, LIU Q R, FAN W, et al. Research on Automatic Scheduling Optimization of Deep Learning Based on Feature Importance[J]. Computer Science, 2024, 51(7): 22-28.

[48] RYU J, PARK E, SUNG H. One-shot tuner for deep learning compilers[C]// Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2022: 89-103.

[49] HAN R, KIM H. Exponentially Expanding the Phase-Ordering Search Space via Dormant Information[C]// Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2024: 250-261.

[50] WANG H, TANG Z, ZHANG C, et al. Automating reinforcement learning architecture design for code optimization[C]// Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2022: 129-143.

[51] ZHANG H B, ZHOU X L, XING M J, et al. AutoConfig: An Automatic Configuration Mechanism for Deep Learning Compiler Optimization[J]. Journal of Software, 2024, 35(6): 2668-2686.

[52] PARK S, LATIFI S, PARK Y, et al. SRTuner: effective compiler optimization customization by exposing synergistic relations [C]// Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2022: 118-130.

[53] COWAN M, MOREAU T, CHEN T, et al. Automatic generation of high-performance quantized machine learning kernels [C]// Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery, 2020: 305-316.

[54] BASTOUL C, ZHANG Z, RAZANAJATO H, et al. Optimizing GPU deep learning operators with polyhedral scheduling constraint injection[C]// Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2022: 313-324.

[55] RIVERA J, FRANCHETTI F, PÜSCHEL M. An interval compiler for sound floating-point computations[C]// Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2021: 52-64.

[56] ALIAS C, PLESCO A. Data-aware process networks[C]// Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2021: 1-11.

[57] ZHAO J, LI B, NIE W, et al. AKG: automatic kernel generation for neural processing units using polyhedral transformations [C]// Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. New York, NY, USA: Association for Computing Machinery, 2021: 1233-1248.

[58] BAGHDADI R, RAY J, ROMDHANE M B, et al. Tiramisu: a polyhedral compiler for expressing fast and portable code[C]// Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization. Washington, DC, USA:

IEEE Press,2019:193-205.

[59] DOERFERT J,SHARMA S,HACK S. Polyhedral expression propagation[C]// Proceedings of the 27th International Conference on Compiler Construction. New York,NY,USA:Association for Computing Machinery,2018:25-36.

[60] ACHARYA A,BONDHUGULA U,COHEN A. Polyhedral auto-transformation with no integer linear programming[C]// Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA:Association for Computing Machinery,2018:529-542.

[61] KRUSE M,GROSSER T. DeLICM:scalar dependence removal at zero memory cost[C]// Proceedings of the 2018 International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery, 2018: 241-253.

[62] KROLIK A,VERBRUGGE C,HENDREN L. r3d3:optimized query compilation on GPUs[C]// Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea: IEEE Press, 2021:277-288.

[63] PROKOPEC A,DUBOSCQ G,LEOPOLDSEDER D,et al. An optimization-driven incremental inline substitution algorithm for just-in-time compilers[C]// Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization. Washington,DC,USA:IEEE Press,2019:164-179.

[64] OTTONI G. HHVM JIT:a profile-guided,region-based compiler for PHP and Hack[C]// Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York,NY,USA:Association for Computing Machinery,2018:151-165.

[65] BROCK J,DING C,XU X,et al. PAYJIT:space-optimal JIT compilation and its practical implementation[C]// Proceedings of the 27th International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery, 2018:71-81.

[66] LEOPOLDSEDER D,STADLER L,WÜRTHINGER T,et al. Dominance-based duplication simulation(DBDS):code duplication to enable compiler optimizations[C]// Proceedings of the 2018 International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery,2018:126-137.

[67] ZHU H,WU R,DIAO Y,et al. Roller:Fast and Efficient Tensor Compilation for Deep Learning[C]// The 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22). 2022.

[68] QIAO B,REICHE O ,HANNIG F,et al. From loop fusion to kernel fusion:a domain-specific approach to locality optimization [C]// Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization. Washington,DC, USA:IEEE Press,2019:242-253.

[69] KURTH A, WOLTERS K, FORSBERG B, et al. Mixed-data-model heterogeneous compilation and OpenMP offloading[C]// Proceedings of the 29th International Conference on Compiler Construction. New York,NY,USA:Association for Computing Machinery,2020:119-131.

[70] LIU Y, HUANG L, WU M, et al. PPOpenCL: a performance-portable OpenCL compiler with host and kernel thread code fusion[C]// Proceedings of the 28th International Conference on Compiler Construction. New York, NY, USA: Association for Computing Machinery,2019:2-16.

[71] LI Y B,ZHAO R C,HAN L,et al. A Parallel Compilation Framework for Heterogeneous Many-Core Processors[J]. Journal of Software,2019,30(4):981-1001.

[72] KIM C,JEONG S,CHO S,et al. Thread-aware area-efficient high-level synthesis compiler for embedded devices[C]// Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization. Virtual Event, Republic of Korea:IEEE Press,2021:327-339.

[73] BAGHSORKHI S S,MARGIOLAS C. Automating efficient variable-grained resiliency for low-power IoT systems[C]// Proceedings of the 2018 International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery,2018:38-49.

[74] HU P,LU M,WANG L,et al. TPU-MLIR:A Compiler For TPU Using MLIR[J]. arXiv:2210.15016,2022.

[75] ESSADKI M, MICHEL B, MAUGARS B, et al. Code Generation for In-Place Stencils[C]// Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery,2023:2-13.

[76] ZHENG L,LI Z,ZHANG H,et al. Alpa:Automating Inter- and {Intra-Operator} Parallelism for Distributed Deep Learning [C]// 16th USENIX Symposium on Operating Systems Design and Implementation(OSDI 22). 2022:559-578.

[77] ZHANG C,MA L,XUE J,et al. COCKTAILER:Analyzing and Optimizing Dynamic Control Flow in Deep Learning[C]// 17th USENIX Symposium on Operating Systems Design and Implementation(OSDI'23). 2023.

[78] NARDI L,KOEPLINGER D,OLUKOTUN K. Practical Design Space Exploration[C]// 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems(MASCOTS). 2019:347-358.

[79] ROTEM N,FIX J,ABDULRASOOL S,et al. Glow:Graph Lowering Compiler Techniques for Neural Networks[J]. arXiv:1805.00907,2018.

[80] CHEN T,MOREAU T,JIANG Z,et al. TVM:An automated End-to-End optimizing compiler for deep learning[C]// 13th USENIX Symposium on Operating Systems Design and Implementation(OSDI'18). 2018:578-594.

[81] RAGAN-KELLEY J, BARNES C, ADAMS A, et al. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines[J]. ACM SIGPLAN Notices,2013,48(6):519-530.

[82] LATTNER C, AMINI M, BONDHUGULA U, et al. MLIR: A Compiler Infrastructure for the End of Moore's Law[J]. arXiv: 2002.11054,2020.

[83] VASILACHE N, ZINENKO O, THEODORIDIS T, et al. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions[J]. arXiv:1802.04730, 2018.

[84] ROESCH J, LYUBOMIRSKY S, KIRISAME M, et al. Relay: A High-Level Compiler for Deep Learning[J]. arXiv:1904.08368, 2019.

[85] ZHENG L, JIA C, SUN M, et al. Ansor: Generating {High-Performance} Tensor Programs for Deep Learning[C]// 14th USENIX Symposium on Operating Systems Design and Implementation(OSDI 20). 2020:863-879.

[86] FEAUTRIER P. Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time | International Journal of Parallel Programming[EB/OL]. [2024-05-26]. https://springer.longhoe.net/article/10.1007/BF01379404.

[87] FEAUTRIER P. Some efficient solutions to the affine scheduling problem. I. One-dimensional time[J]. International Journal of Parallel Programming, 1992, 21(5):313-347.

[88] LIM A W, LAM M S. Maximizing parallelism and minimizing synchronization with affine transforms[C]// Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. New York, NY, USA: Association for Computing Machinery, 1997:201-214.

[89] LIM A W, CHEONG G I, LAM M S. An affine partitioning algorithm to maximize parallelism and minimize communication [C]// Proceedings of the 13th International Conference on Supercomputing. Rhodes Greece: ACM, 1999:228-237.

[90] BONDHUGULA U, BASKARAN M, KRISHNAMOORTHY S, et al. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model[C]// Compiler Construction. Berlin: Springer, 2008:132-146.

[91] ZHANG C, DONG R, WANG H, et al. MAGPY: compiling eager mode DNN programs by monitoring execution states[C]// Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference. 2024:683-698.

[92] SHIN Y, PARK J, CHO S, et al. PIMFlow: Compiler and Runtime Support for CNN Models on Processing-in-Memory DRAM [C]// Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization. New York, NY, USA: Association for Computing Machinery, 2023:249-262.

[93] CASTRO-LOPEZ O, VEGA-LOPEZ I F. Multi-target compiler for the deployment of machine learning models[C]// Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization. Washington, DC, USA: IEEE Press, 2019:280-281.

[94] GUO D, YANG D, ZHANG H, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning[J]. arXiv:2501.12948, 2025.

**LIU Zhengyu**, born in 1997, doctoral student, is a member of CCF (No. H4565G). His main research interest is distributed parallel and compiler.

**ZHANG Fan**, born in 1981, professor, Ph.D supervisor. His main research interests include computer architecture and network security.

(责任编辑:柯颖)