先进编译实验室
Advanced Compiler

编译论坛

动态shape深度学习算子自动调优DietCode

嘉宾：王一帆

先进编译实验室
Advanced Compiler

DietCode

DietCode: Automatic Optimization for Dynamic Tensor Programs

Bojian Zheng [*1 2 3] **Ziheng Jiang** [*4] **Cody Hao Yu** [2] **Haichen Shen** [5] **Josh Fromm** [6]
Yizhi Liu [2] **Yida Wang** [2] **Luis Ceze** [7 6] **Tianqi Chen** [8 6] **Gennady Pekhimenko** [1 2 3]

Abstract

Achieving high performance for compute-intensive operators in machine learning (ML) workloads is a crucial but challenging task. Many ML and system practitioners rely on vendor libraries or auto-schedulers to do the job. While the former requires large engineering efforts, the latter only supports static-shape workloads in existing works. It is difficult, if not impractical, to apply existing auto-schedulers directly to dynamic-shape workloads, as this leads to extremely long auto-scheduling time.

We observe that the key challenge faced by existing auto-schedulers when handling a dynamic-shape workload is that they cannot construct a unified search space for all the possible shapes of the workload, because their search space is shape-dependent. To address this, we propose *DietCode*, a new auto-scheduler framework that efficiently supports dynamic-shape workloads by constructing a *shape-generic* search space and cost model. Under this construction, all shapes *jointly* search within the same space and update the same cost model when auto-scheduling, which is therefore more efficient compared with existing auto-schedulers.

We evaluate *DietCode* using state-of-the-art machine learning workloads on a modern GPU. Our evaluation shows that *DietCode* has the following key strengths when auto-scheduling an entire model end-to-end: (1) reduces the auto-scheduling time by up to $5.88\times$ less than the state-of-the-art auto-scheduler on the uniformly sampled dynamic shapes ($94.1\times$ estimated if all the possible shapes are included), (2) improves performance by up to 69.5% better than the auto-scheduler and 18.6% better than the vendor library. All these advantages make *DietCode* an efficient and practical solution for dynamic-shape workloads.

1 Introduction

Deep neural networks (DNNs) form an important class of ML algorithms (He et al., 2016; Vaswani et al., 2017; Amodei et al., 2016; Devlin et al., 2019). They are made of tensor operators which are often executed for tens of

ML frameworks rely on heavily optimized, hand-crafted vendor libraries (e.g., oneDNN (oneAPI, 2021) on Intel CPUs; cuDNN (Chetlur et al., 2014) and cuBLAS (NVIDIA, 2021) on NVIDIA GPUs) to provide highly optimized operators (Abadi et al., 2016; Paszke et al., 2019; Chen et al., 2015). Despite delivering high performance, the develop-

目录

01. 背景与动机

02. 主体思路

03. 形状通用搜索空间

04. 成本模型

05. 自动调度器

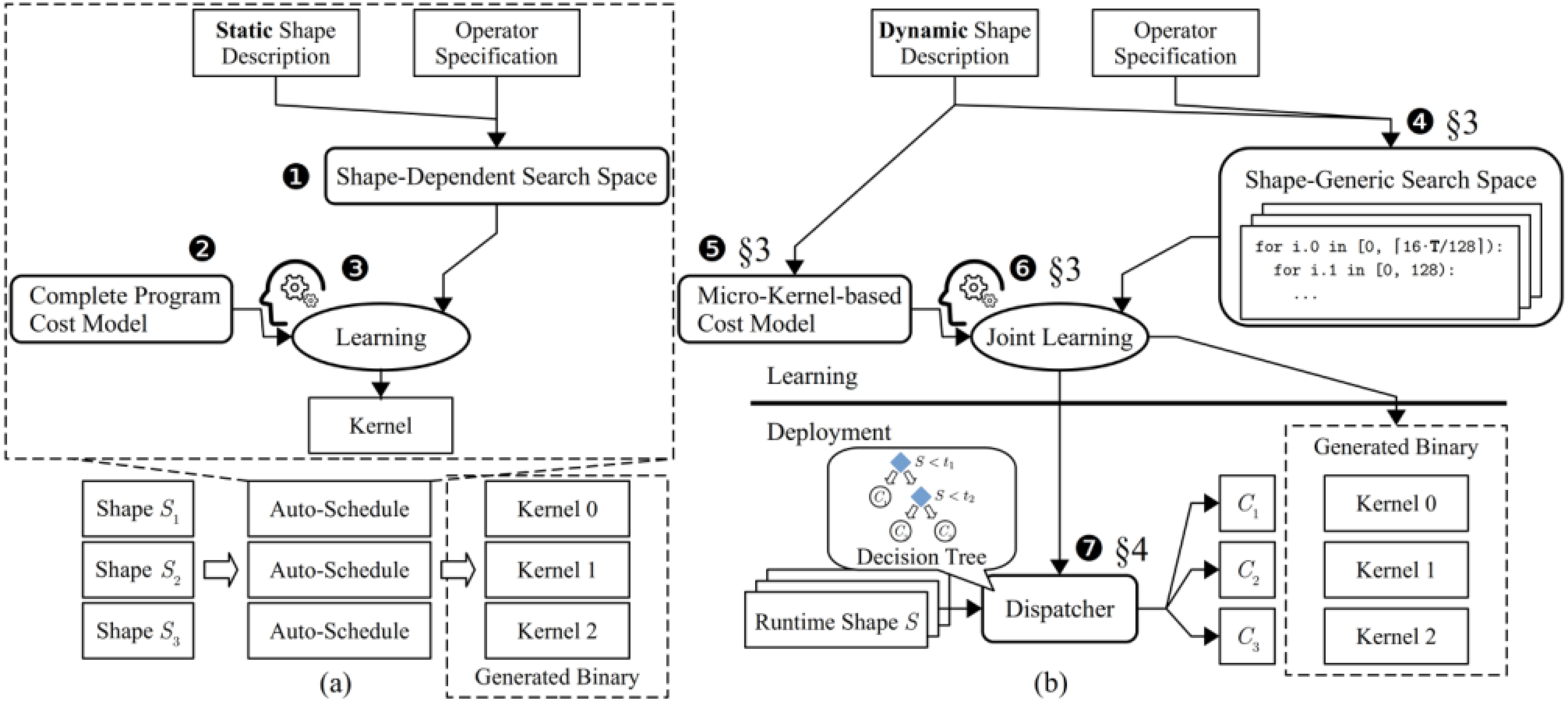
06. 性能

背景与动机

研究背景

深度学习模型已经广泛地应用于日常的生产与生活。算子是模型的最小执行单元，其性能极大地影响着模型整体的执行效率。高性能的算子需要针对目标硬件的架构特点进行优化（称为调度），以充分利用目标硬件的计算能力。而软件对于硬件的适配需要大量软硬件跨度的专家来解决问题。他们会根据硬件的架构、指令等对算子进行适配。

主体思路

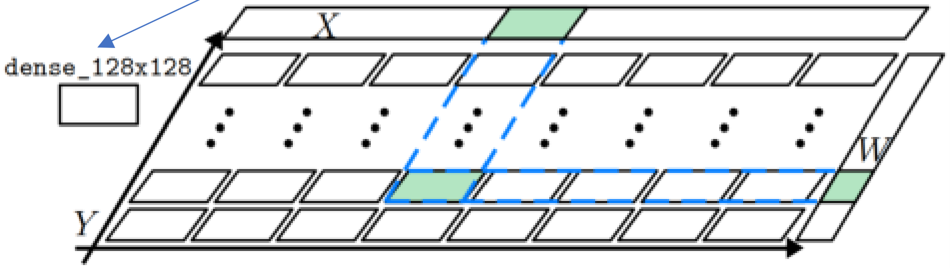

Static Shape Description
Operator Specification
❶ Shape-Dependent Search Space
❷ Complete Program Cost Model
❸ Learning
Kernel
Shape S_1
Shape S_2
Shape S_3
Auto-Schedule
Auto-Schedule
Auto-Schedule
Kernel 0
Kernel 1
Kernel 2
Generated Binary
(a)
Dynamic Shape Description
Operator Specification
❹ §3
Shape-Generic Search Space
for i.0 in [0, ⌈16·T/128⌉):
for i.1 in [0, 128):
...
❺ §3
Micro-Kernel-based Cost Model
❻ §3
Joint Learning
Learning
Deployment
$S < t_1$
$S < t_2$
Decision Tree
Runtime Shape S
❼ §4
Dispatcher
C_1
C_2
C_3
Generated Binary
Kernel 0
Kernel 1
Kernel 2
(b)

形状通用搜索空间


```
for (i = 0; i < T; ++i)
  A[i] = ...
```
(a)

```
for (io = 0; io < ⌈T/t⌉; ++io)
  for (ii = 0; ii < t; ++ii)
    A[io*t+ii] = ...
```
(b)

$$Y = XW^{\top}, X : [128, 768], W : [128, 768]$$



观察到，不同形状的搜索空间实际上可以重叠，因此可以潜在地形成形状通用搜索空间，而不是使用现有的自动调度器逐个调整每个可能的形状。

$$Y = XW^{\top} \quad \begin{array}{ll} \text{(a)} & X : [1024, 768], W : [2304, 768] \\ \text{(b)} & X : [16 \times \mathbf{T}, 768], W : [2304, 768], \\ & \mathbf{T} \in [1, 128] \end{array}$$

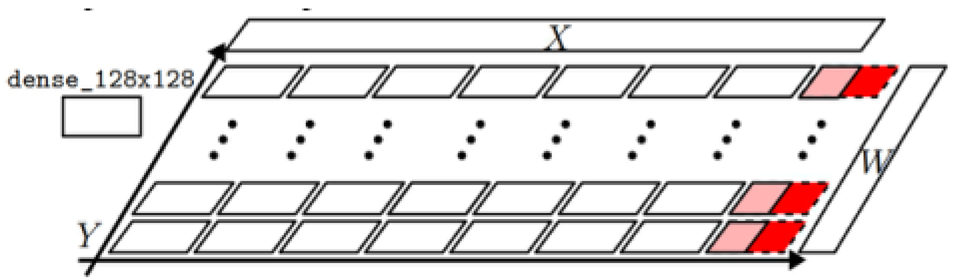
形状通用搜索空间

```
for (i = 0; i < T; ++i)
  A[i] = ...
```

(a)

```
for (io = 0; io < ⌈T/t⌉; ++io)
  for (ii = 0; ii < t; ++ii)
    A[io*t+ii] = ...
```

(b)



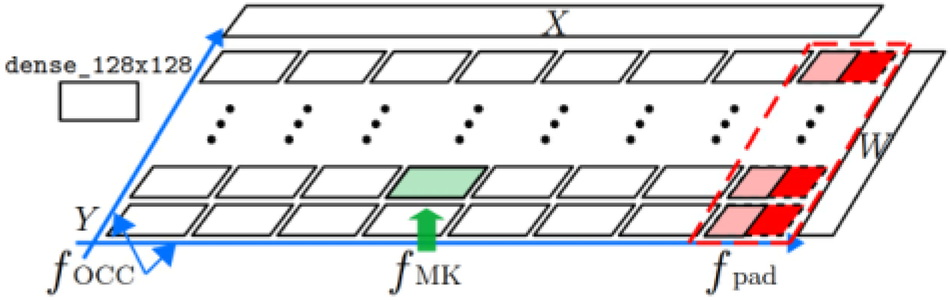

```
for i.0 in [0,T):
  for i.1 in [0,t):
    if i.0*t+i.1 < T:
      X_local = X[...]
    if i.0*t+i.1 < T:
      Y_local = ...
    if i.0*t+i.1 < T:
      Y[...] = Y_local
```

然而，工作负载实例通常不完全适合微内核，如图所示，最后一列的微内核没有完全具体化。在这些情况下，需要在微内核内部注入边界检查，以确保程序不会在无效数据值上运行，但它们也会给程序带来很大的性能下降这是因为这些检查会在生成的程序中引入额外的分支和计算指令。

成本模型

为了准确预测基于微内核的完整程序的性能，我们在成本模型中设计了三个项，分别说明：①微内核的性能；②硬件内核占用惩罚，其与执行该微内核以组成完整程序的次数相关；③填充惩罚。成本模型的数学表达式如下：

$$\mathrm{Cost}_M(P) = f_{\mathrm{MK}}(\mathrm{FeatureExtractor}(M))① \cdot \underbrace{② f_{\mathrm{OCC}}(P/M) \cdot f_{\mathrm{pad}}(P, M)③}_{f_{\mathrm{adapt}}(P,M)}$$



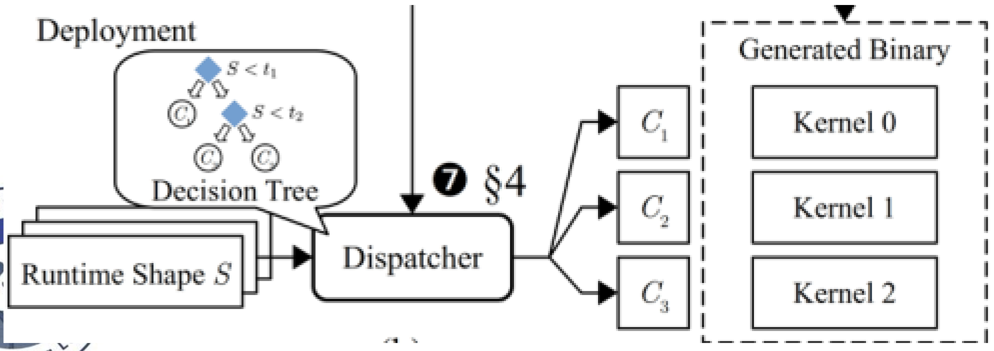
图说明了方程中三个项之间的相关性以及微内核如何形成完整程序，其中我们可以看到方程和完整程序组成之间的一一对应关系。这种对应关系使我们能够准确预测基于微内核的完整程序的性能，从而在联合学习过程中使用进化搜索有效地搜索高性能微内核。

自动调度

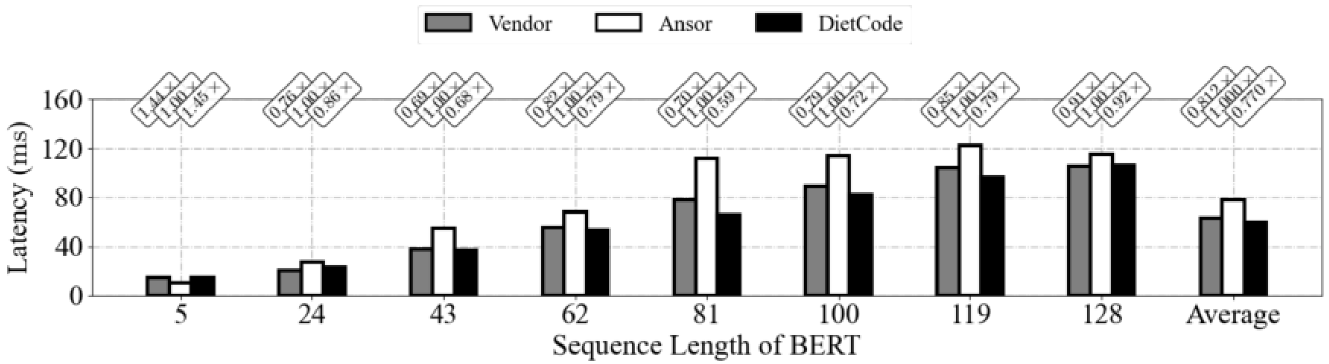
学习过程完成后，DietCode将生成一组微内核作为自动调度结果。为了将所有可能的形状分配给这些微内核，我们让每个形状S根据等式3中的成本公式投票给其最喜欢的微内核

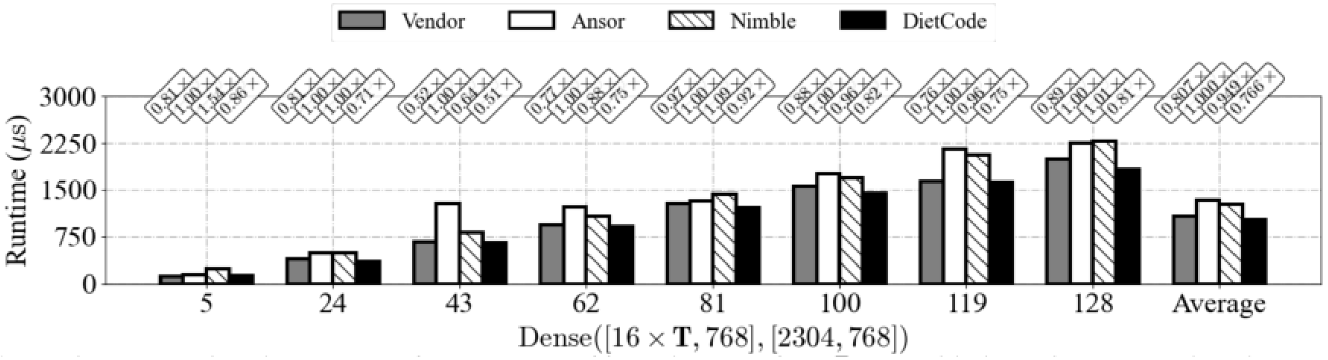
$$\text{vote}(S) = \text{argmax}_M(\text{Cost}_M(P(S, M)))$$

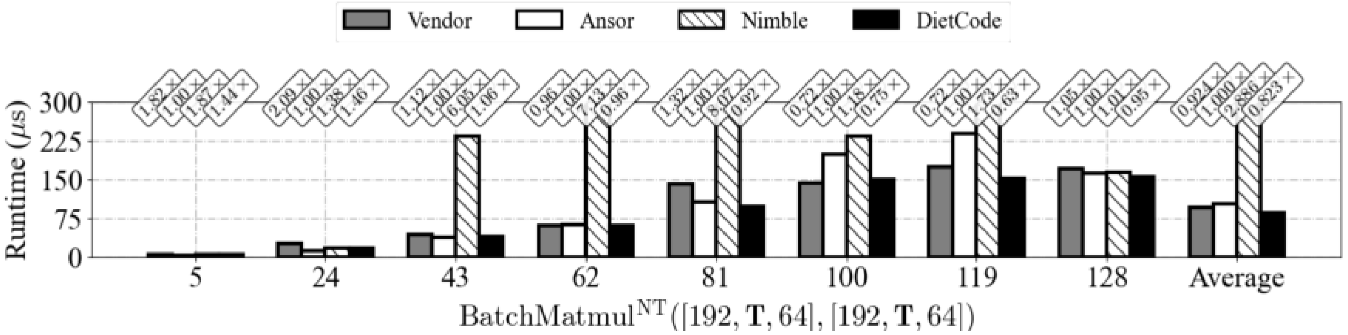
我们使用scikit学习框架训练决策树。决策树的输入是所有可能的形状，输出标签是它们选择的微内核。



性能







评估表明，在DietCode上，可以将自动调度时间显著缩短5.88倍（如果包括所有可能的形状，则预测为94.1倍），同时在端到端的完整最先进DNN模型上，性能比最先进的自动调度程序高69.5%，比供应商库高18.6%。