东南大学 SOUTHEAST UNIVERSITY

计算机科学与工程学院 School of computer science and engineering

Pruner: A Draft-then-Verify Exploration Mechanism to Accelerate Tensor Program Tuning

一种通过草稿-验证探索方式实现加速张量程序调优的机制

Liang Qiao*
University of Science and Technology of China
Hefei, China
ql1an9@mail.ustc.edu.cn

Jun Shi*
University of Science and Technology of China
Hefei, China
shijun18@ustc.edu.cn

Xiaoyu Hao
University of Science and Technology of China
Hefei, China
hxy2018@mail.ustc.edu.cn

Xi Fang
University of Science and Technology of China
Hefei, China
fangxi@mail.ustc.edu.cn

Sen Zhang
University of Science and Technology of China
Hefei, China
sen@mail.ustc.edu.cn

Minfan Zhao
University of Science and Technology of China
Hefei, China
zmf@mail.ustc.edu.cn

Ziqi Zhu
University of Science and Technology of China
Hefei, China
ta1ly@mail.ustc.edu.cn

Junshi Chen†
University of Science and Technology of China
Hefei, China
cjuns@ustc.edu.cn

Hong An‡†
University of Science and Technology of China
Hefei, China
han@ustc.edu.cn

Xulong Tang
University of Pittsburgh
Pittsburgh, USA
tax6@pitt.edu

Bing Li
NIO
Shanghai, China
libing475211023@sjtu.edu.cn

Honghui Yuan
Xinyang Wang
ethan.song@nio.com
12307130155@fudan.edu.cn
NIO
Shanghai, China

https://github.com/qiaolian9/Pruner

ASPLOS'25

汇报人：王维龙　2025 年 11 月 24 日

安虹，博士，教授，博导，省教学名师

石军，中国科学技术大学博士后研究员

[1] PWDFT-SW: Extending the Limit of Plane-Wave DFT Calculations to 16K Atoms on the New Sunway Supercomputer (TPDS 2025) CCF A

[2] Pruner: A Draft-then-Verify Exploration Mechanism to Accelerate Tensor Program Tuning (ASPLOS 2025) CCF A

[3] NDFT: Accelerating Density Functional Theory Calculations via Hardware/Software Co-Design on Near-Data Computing System (DAC 2025) CCF A

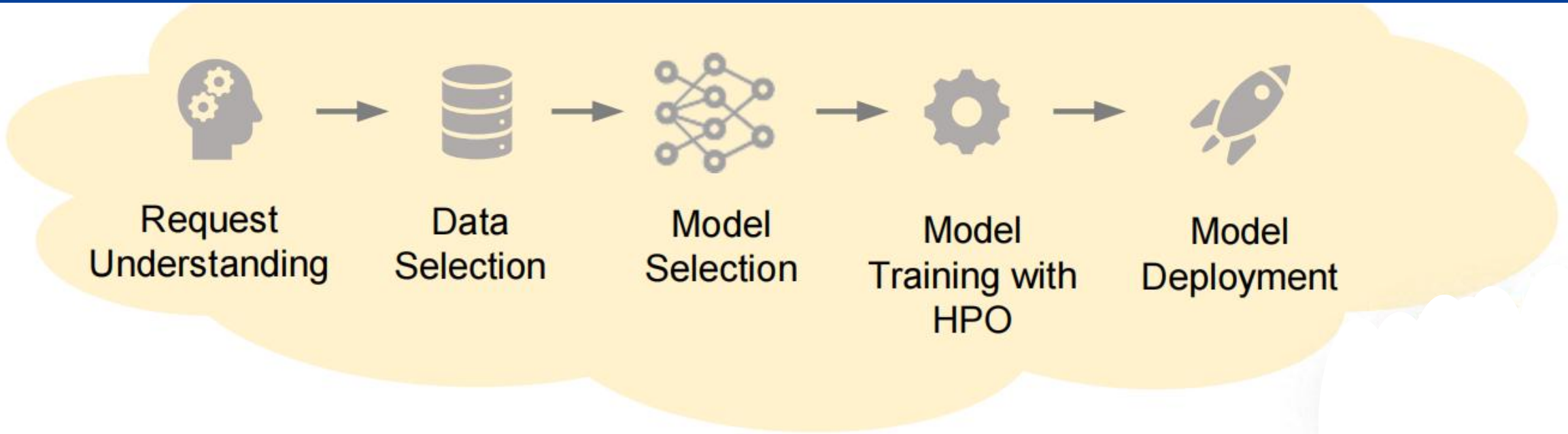
[4] Predictive Accuracy-Based Active Learning for Medical Image Segmentation (IJCAI 2024) CCF A

ACSA Lab | Publications (ustc.edu.cn)

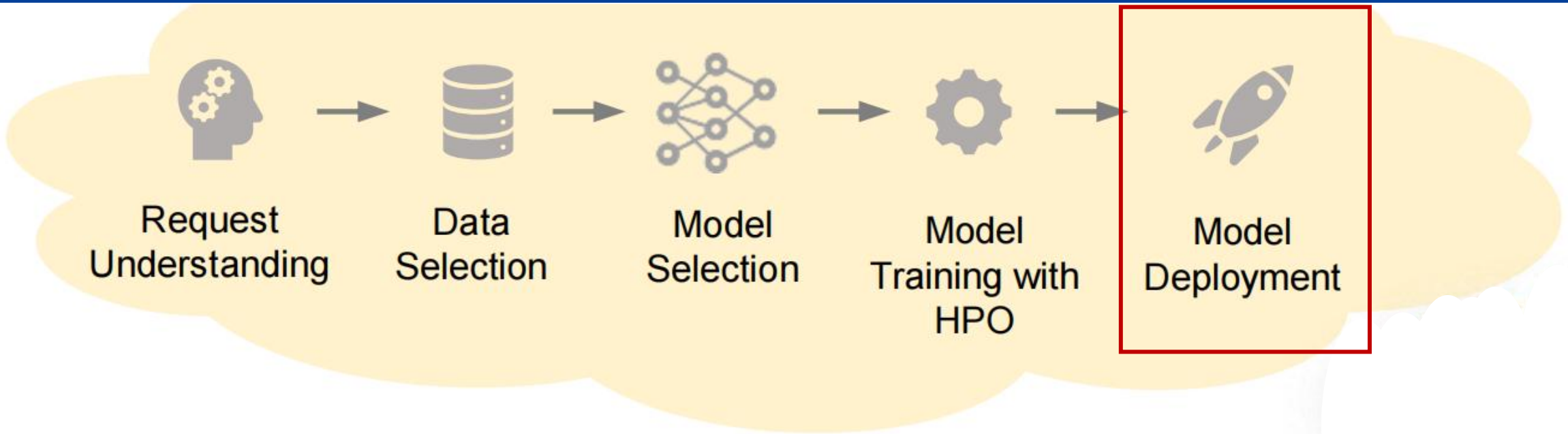
提纲

提纲

AI模型的应用周期

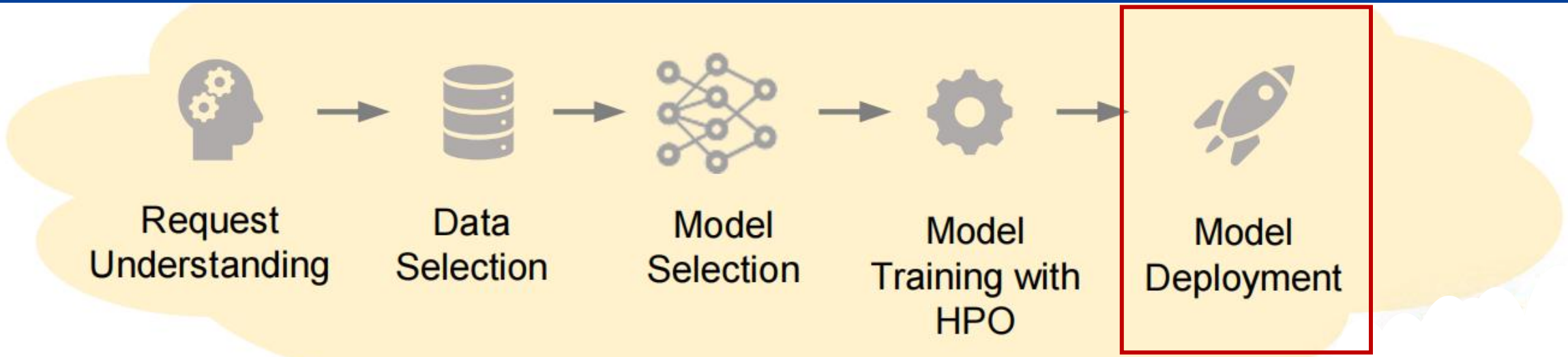

Request Understanding
Data Selection
Model Selection
Model Training with HPO
Model Deployment

AI模型的应用周期



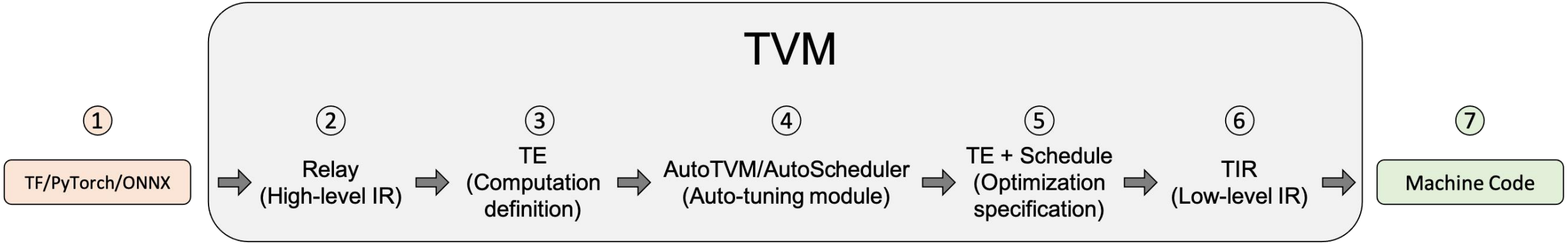
1. 直接部署
2. 编译优化后部署

AI模型的应用周期

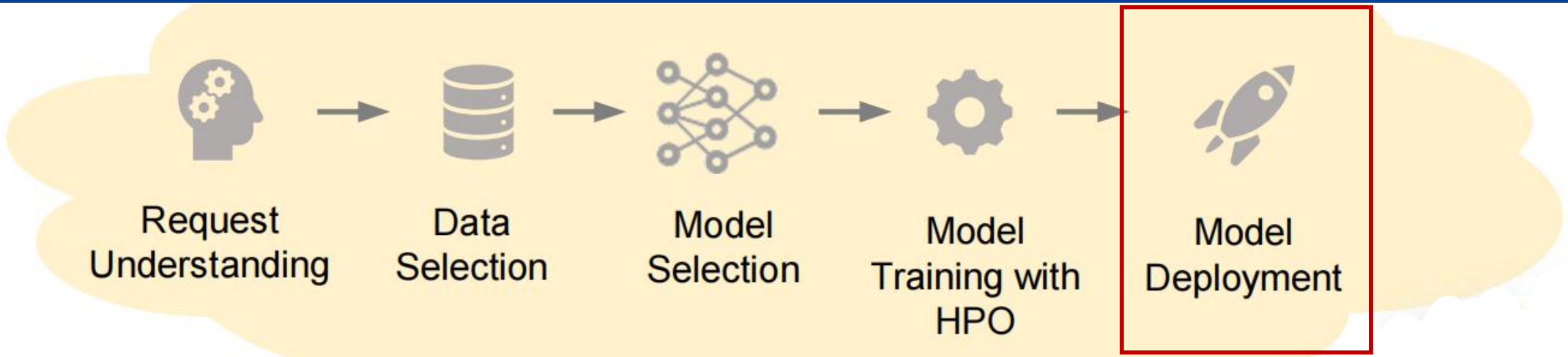


1. 直接部署

2. 编译优化后部署

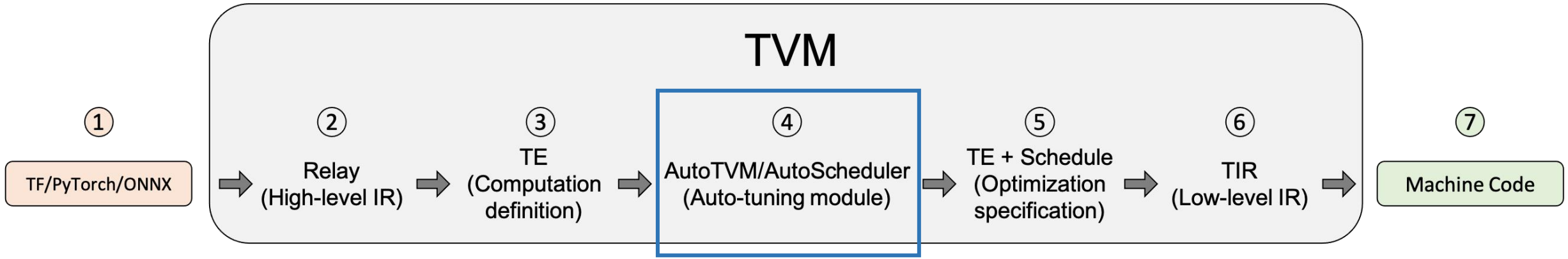


AI模型的编译优化流程

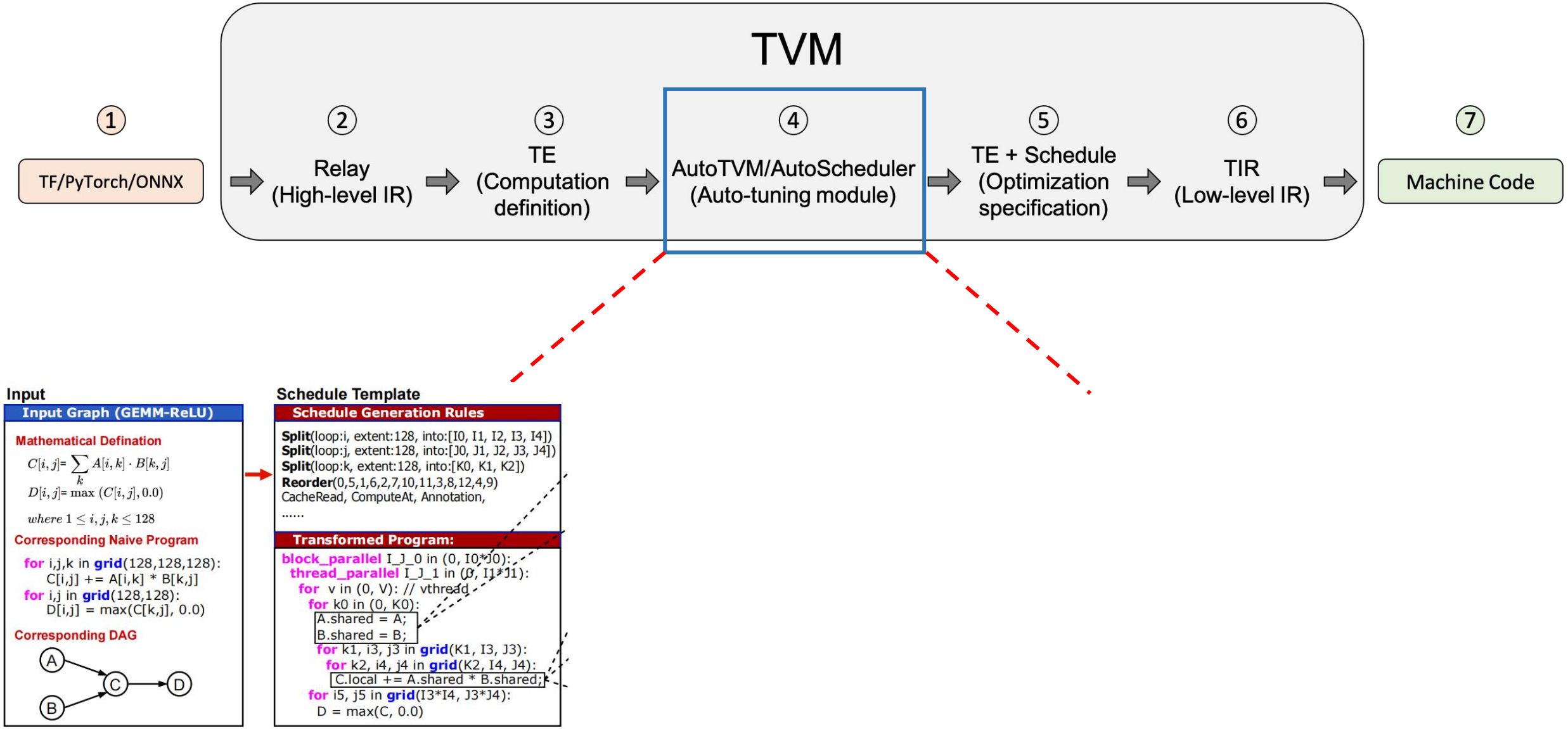


1. 直接部署

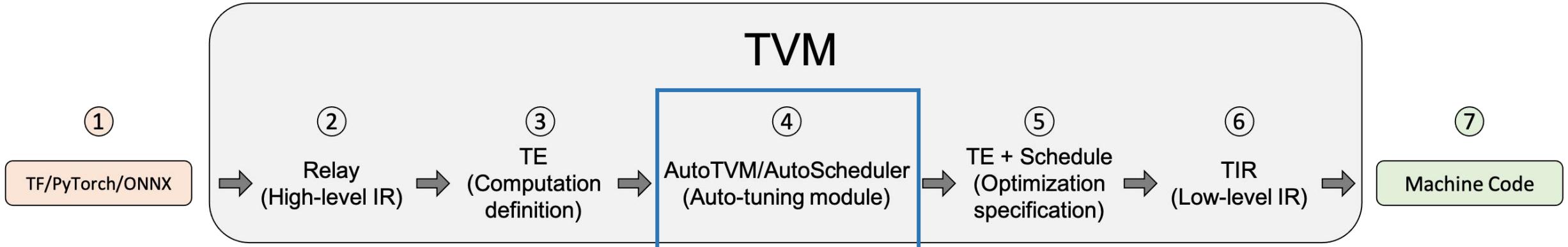
2. 编译优化后部署



例子：GPU中的矩阵计算（通用矩阵乘GEMM-ReLU）



例子：GPU中的矩阵计算（通用矩阵乘GEMM-ReLU）



Input

Input Graph (GEMM-ReLU)

Mathematical Defination

$C[i,j] = \sum_k A[i,k] \cdot B[k,j]$

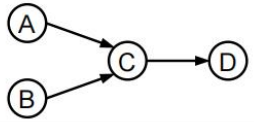
$D[i,j] = \max(C[i,j], 0.0)$

$where\ 1 \le i,j,k \le 128$

Corresponding Naive Program

```
for i,j,k in grid(128,128,128):
    C[i,j] += A[i,k] * B[k,j]
for i,j in grid(128,128):
    D[i,j] = max(C[k,j], 0.0)
```

Corresponding DAG



Schedule Template

Schedule Generation Rules

```
Split(loop:i, extent:128, into:[I0, I1, I2, I3, I4])
Split(loop:j, extent:128, into:[J0, J1, J2, J3, J4])
Split(loop:k, extent:128, into:[K0, K1, K2])
Reorder(0,5,1,6,2,7,10,11,3,8,12,4,9)
CacheRead, ComputeAt, Annotation,
.....
```

Transformed Program:

```
block_parallel I_J_0 in (0, I0*J0):
 thread_parallel I_J_1 in (0, I1*J1):
  for v in (0, V): // vthread
   for k0 in (0, K0):
    A.shared = A;
    B.shared = B;
    for k1, i3, j3 in grid(K1, I3, J3):
     for k2, i4, j4 in grid(K2, I4, J4):
      C.local += A.shared * B.shared;
   for i5, j5 in grid(I3*I4, J3*J4):
    D = max(C, 0.0)
```

Example Input 1:

*** The mathmetical expression:**

$C[i,j] = \sum_k A[i,k] \times B[k,j]$

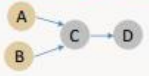
$D[i,j] = \max(C[i,j], 0.0)$

where $0 \le i,j,k < 512$

*** The corresponding naive program:**

```
for i in range(512):
  for j in range(512):
    for k in range(512):
      C[i, j] += A[i, k] * B[k, j]
for i in range(512):
  for j in range(512):
    D[i, j] = max(C[i, j], 0.0)
```

*** The corresponding DAG:**



Generated sketch 1

```
for i.0 in range(TILE_I0):
  for j.0 in range(TILE_J0):
    for i.1 in range(TILE_I1):
      for j.1 in range(TILE_J1):
        for k.0 in range(TILE_K0):
          for i.2 in range(TILE_I2):
            for j.2 in range(TILE_J2):
              for k.1 in range(TILE_I1):
                for i.3 in range(TILE_I3):
                  for j.3 in range(TILE_J3):
                    C[...] += A[...] * B[...]
        for i.4 in range(TILE_I2 * TILE_I3):
          for j.4 in range(TILE_J2 * TILE_J3):
            D[...] = max(C[...], 0.0)
```

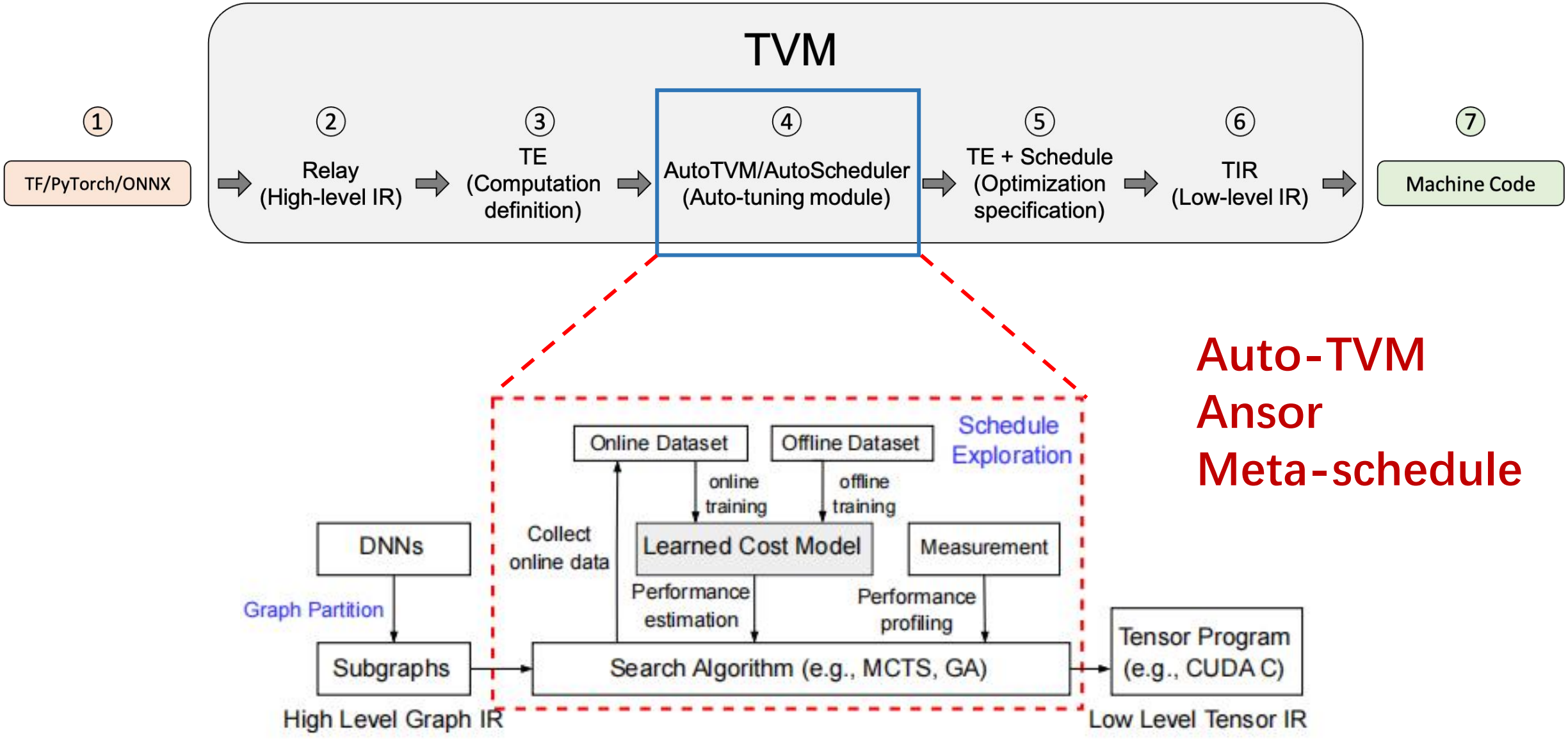
Sampled program 1

```
parallel i.0@j.0@i.1@j.1 in range(256):
  for k.0 in range(32):
    for i.2 in range(16):
      unroll k.1 in range(16):
        unroll i.3 in range(4):
          vectorize j.3 in range(16):
            C[...] += A[...] * B[...]
  for i.4 in range(64):
    vectorize j.4 in range(16):
      D[...] = max(C[...], 0.0)
```

Sampled program 2

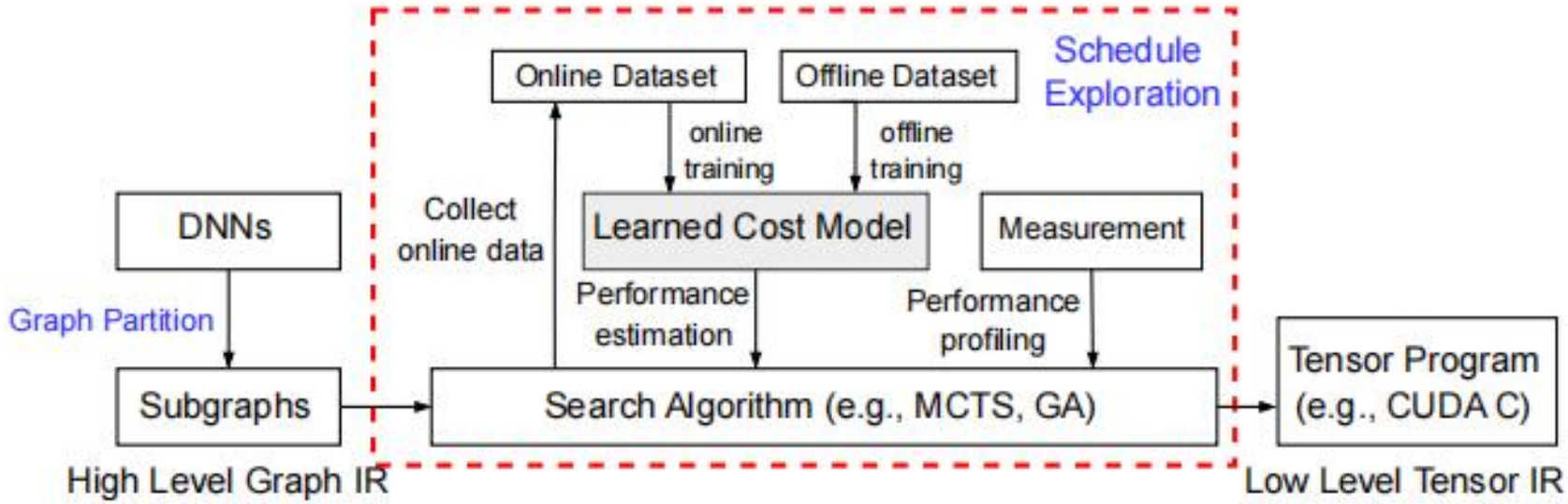
```
parallel i.2 in range(16):
  for j.2 in range(128):
    for k.1 in range(512):
      for i.3 in range(32):
        vectorize j.3 in range(4):
          C[...] += A[...] * B[...]
parallel i.4 in range(512):
  for j.4 in range(512):
    D[...] = max(C[...], 0.0)
```

AI模型的编译优化流程



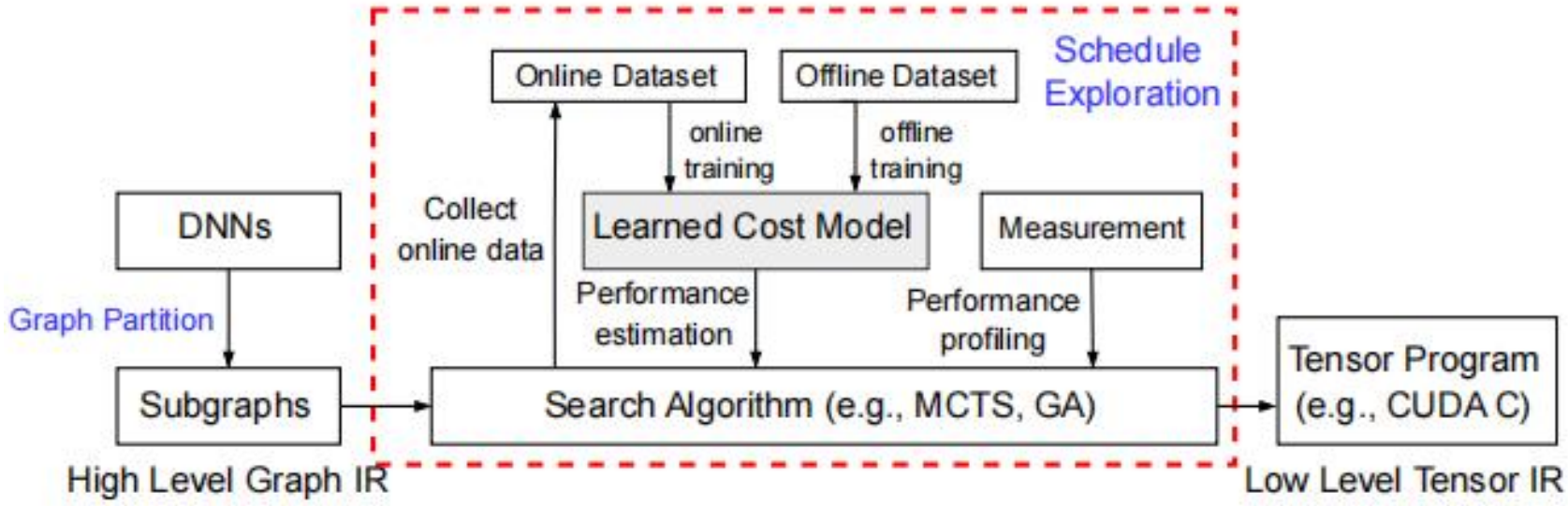
Auto-TVM
Ansor
Meta-schedule

AI模型的编译优化流程



提纲

L1：调度方案搜索成本高



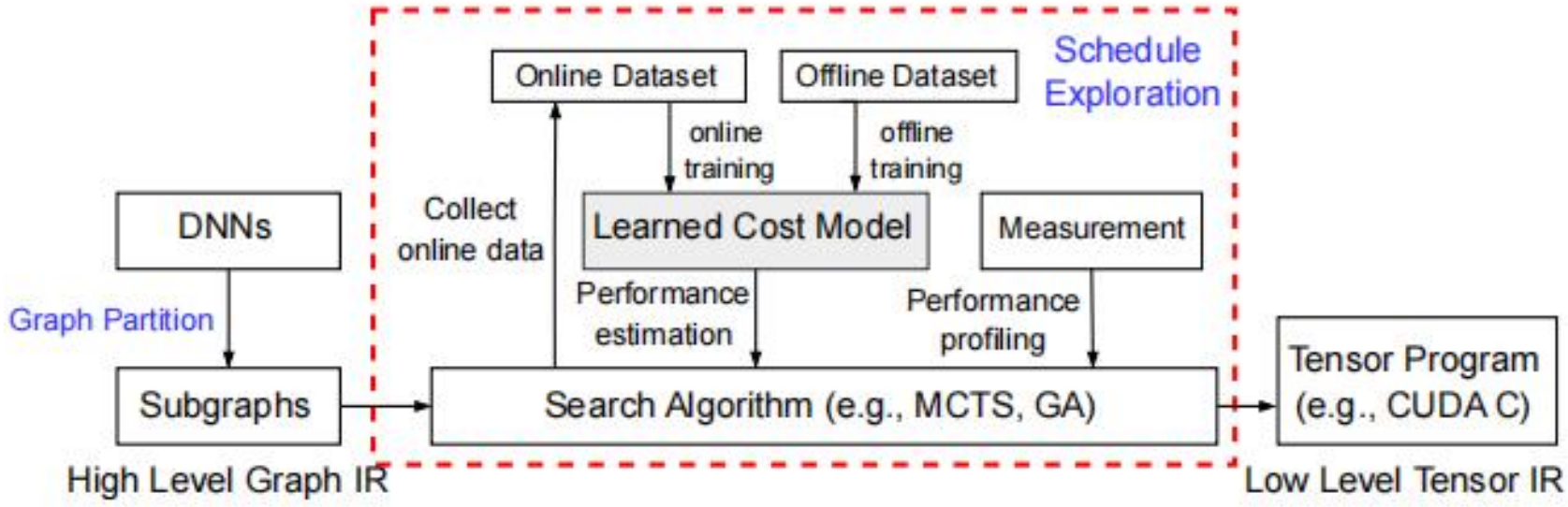
[1] Zheng L, Liu R, Shao J, et al. Tenset: A large-scale program performance dataset for learned tensor compilers[C]//Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1). 2021.

[2] 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.

[3] Zhai Y, Zhang Y, Liu S, et al. Tlp: A deep learning-based cost model for tensor program tuning[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2023: 833-845.

[4] Zhao Y, Sharif H, Adve V, et al. Felix: Optimizing tensor programs with gradient descent[C]//Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. 2024: 367-381.

L1：调度方案搜索成本高



[1] Zheng L, Liu R, Shao J, et al. Tenset: A large-scale program performance dataset for learned tensor compilers[C]//Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1). 2021.
[2] 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.

局限性1: 基于深度学习模型的成本模型，导致探索成本高昂

2.
[4] Zhao Y, Sharif H, Adve V, et al. Felix: Optimizing tensor programs with gradient descent[C]//Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. 2024: 367-381.

L2：不同硬件设备的方案难迁移

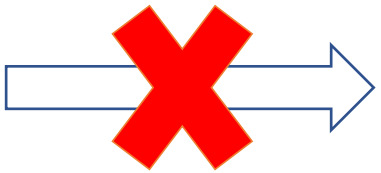
V100

成本模型

L2：不同硬件设备的方案难迁移

V100

成本模型



Jetson Orin Nano

局限性2: 不同硬件设备的方案间迁移效果不佳，甚至能力崩溃

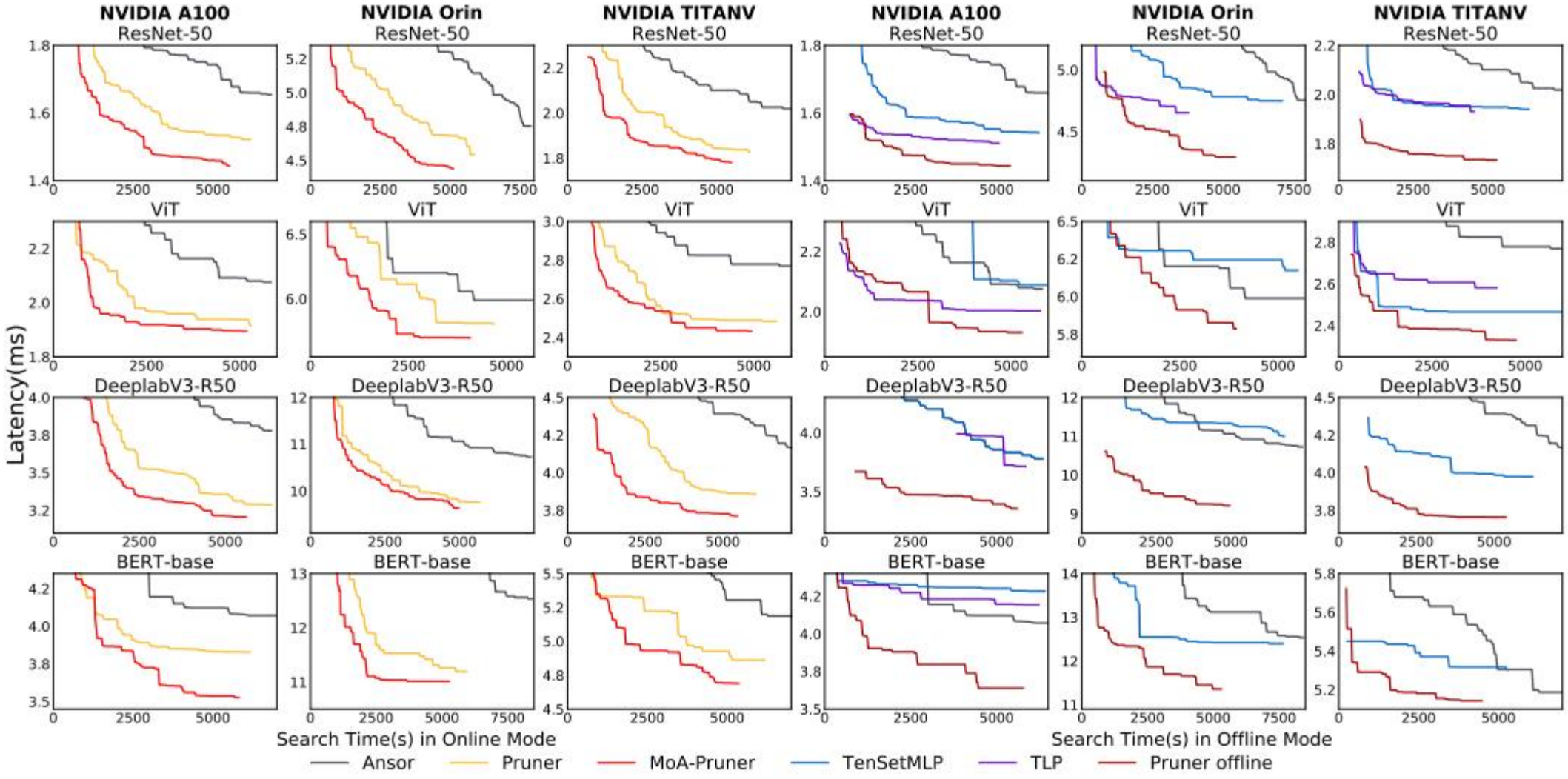
O1：探索机制可以更加高效

Table 1. Tuning costs (min) for Ansor with 2,000 trials on Orin, which means the time cost for space exploration, model training, and kernel hardware measurement, respectively.

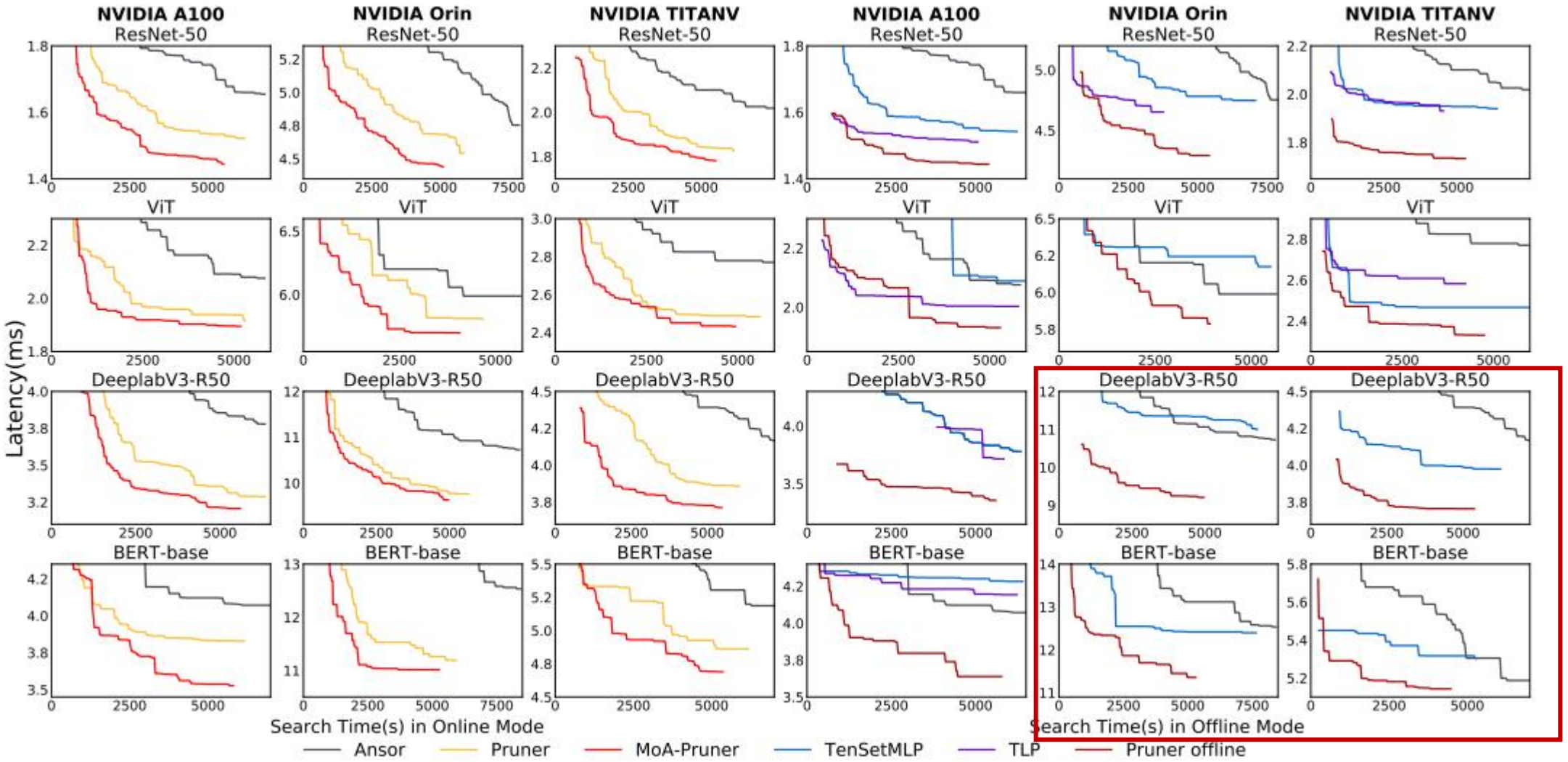
Ansor	R50 [18]	DeTR [8]	I-V3 [35]
Exploration	35	30.31	41.8
Training	5.4	5.6	5.5
Measurement	44.4	50.61	49.4

发现：使用学习成本模型进行空间探索所占时间接近40%。当应用更复杂的成本模型时，所占时间会增加。

O2：构建成本模型困难


NVIDIA A100
NVIDIA Orin
NVIDIA TITANV
NVIDIA A100
NVIDIA Orin
NVIDIA TITANV
ResNet-50
ViT
DeeplabV3-R50
BERT-base
Latency(ms)
Search Time(s) in Online Mode
Search Time(s) in Offline Mode
Ansor
Pruner
MoA-Pruner
TenSetMLP
TLP
Pruner offline

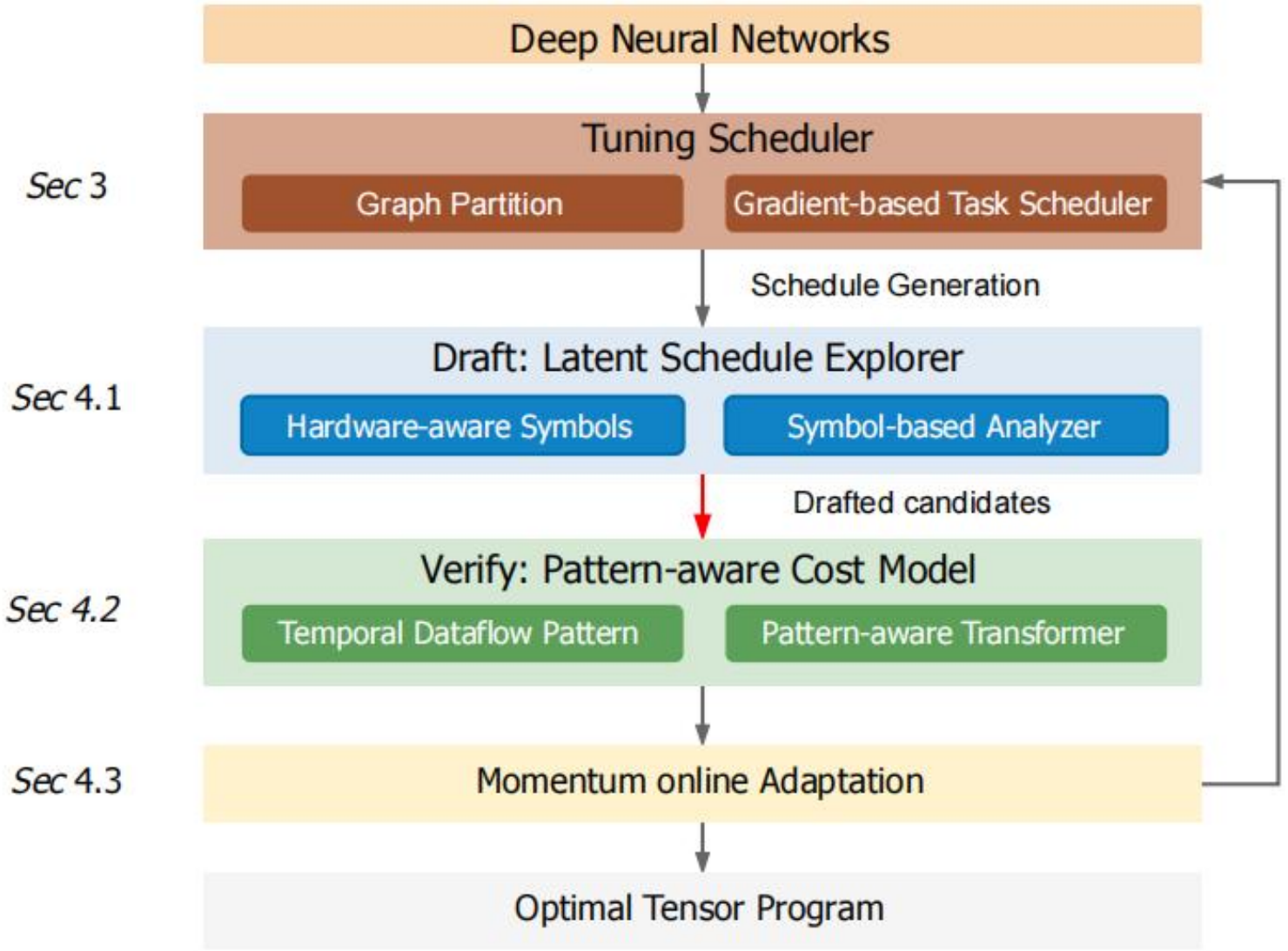
O2：构建成本模型困难



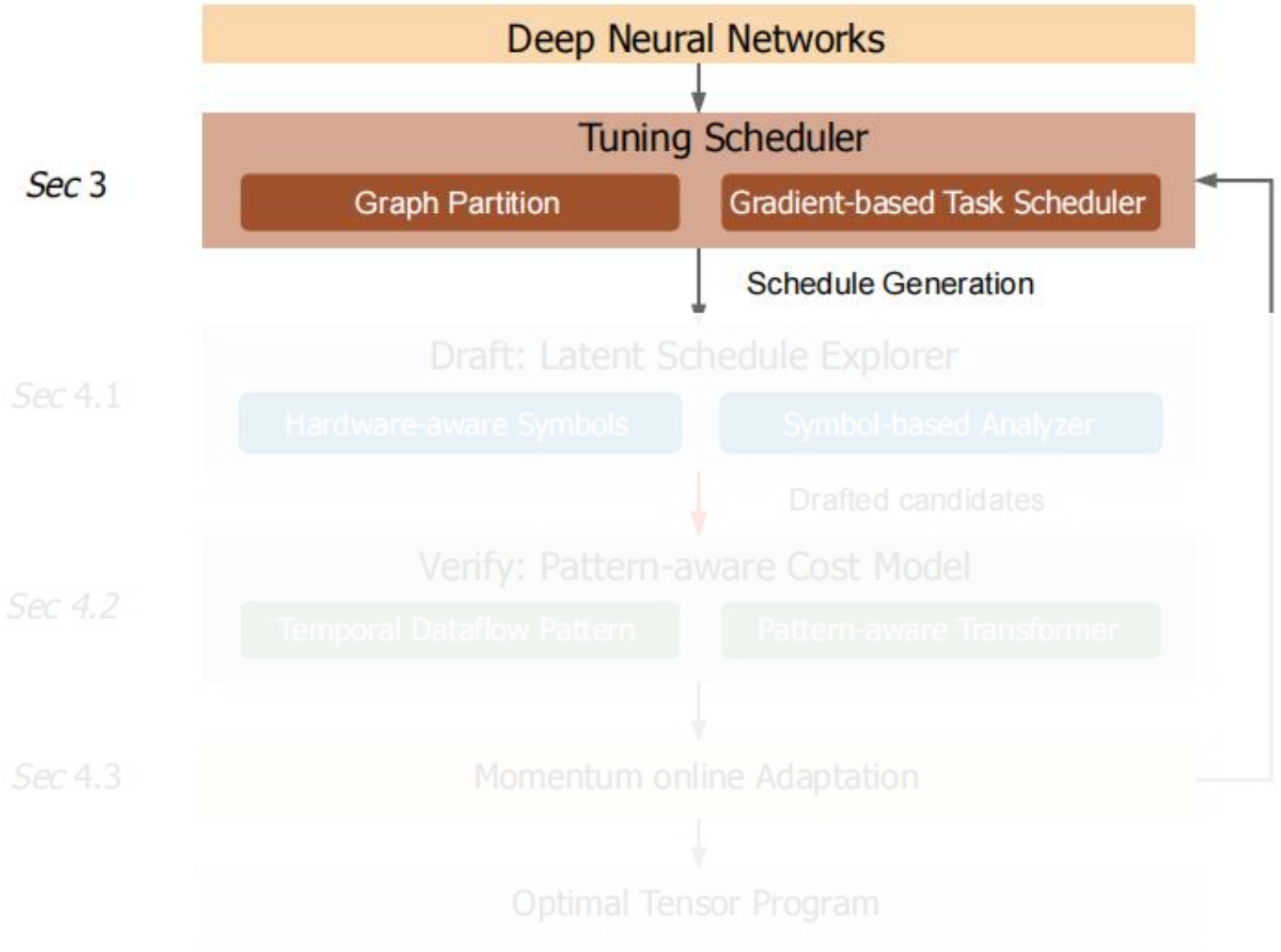
发现：离线训练的成本模型迁移困难，在线的成本模型需要额外微调甚至重训练

提纲

Overview

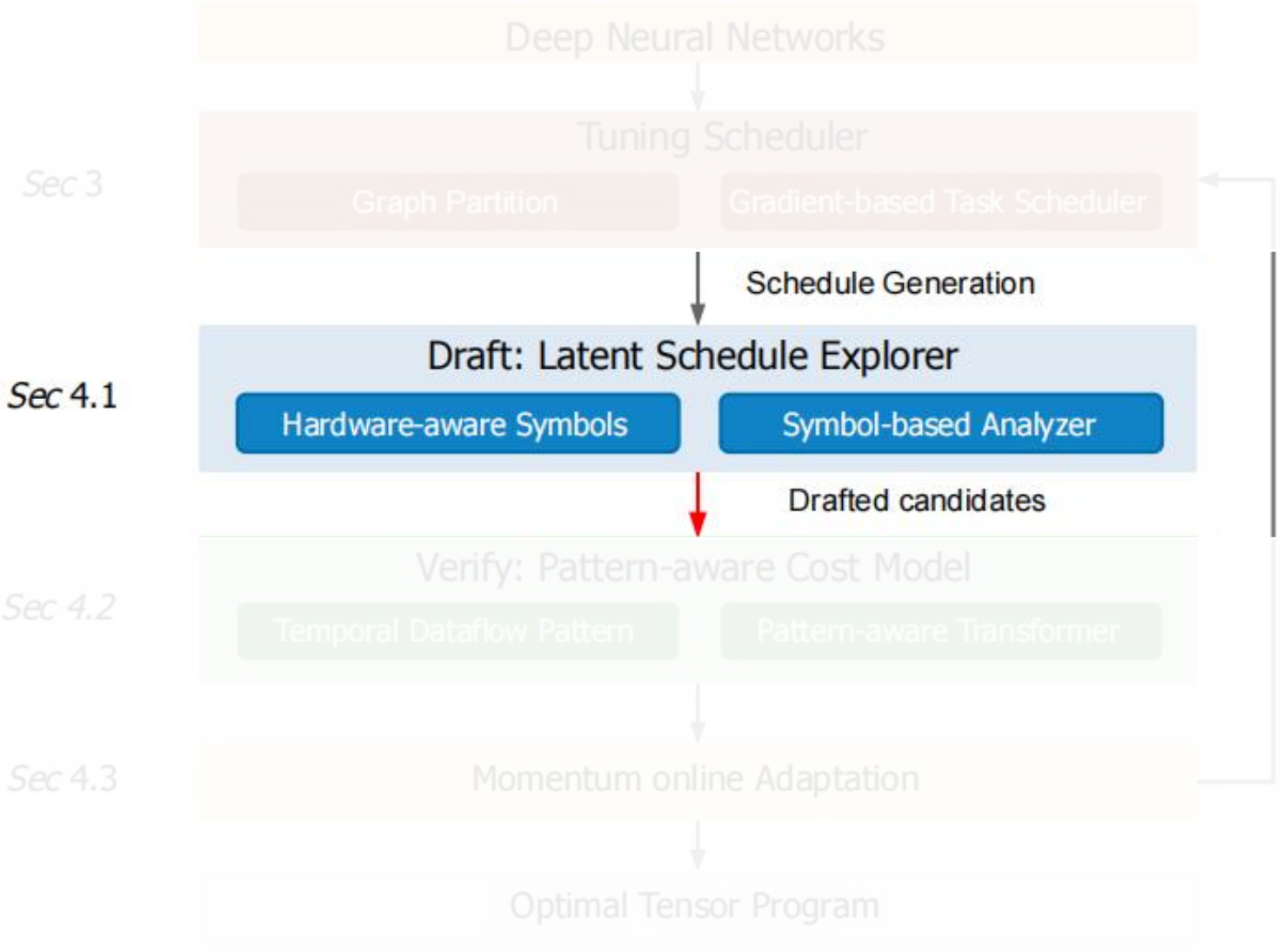

Deep Neural Networks
Sec 3
Tuning Scheduler
Graph Partition
Gradient-based Task Scheduler
Schedule Generation
Sec 4.1
Draft: Latent Schedule Explorer
Hardware-aware Symbols
Symbol-based Analyzer
Drafted candidates
Sec 4.2
Verify: Pattern-aware Cost Model
Temporal Dataflow Pattern
Pattern-aware Transformer
Sec 4.3
Momentum online Adaptation
Optimal Tensor Program

Overview



TVM中Ansor自动或集成的工具实现

Draft: Latent Schedule Explorer



Hardware-aware Symbols

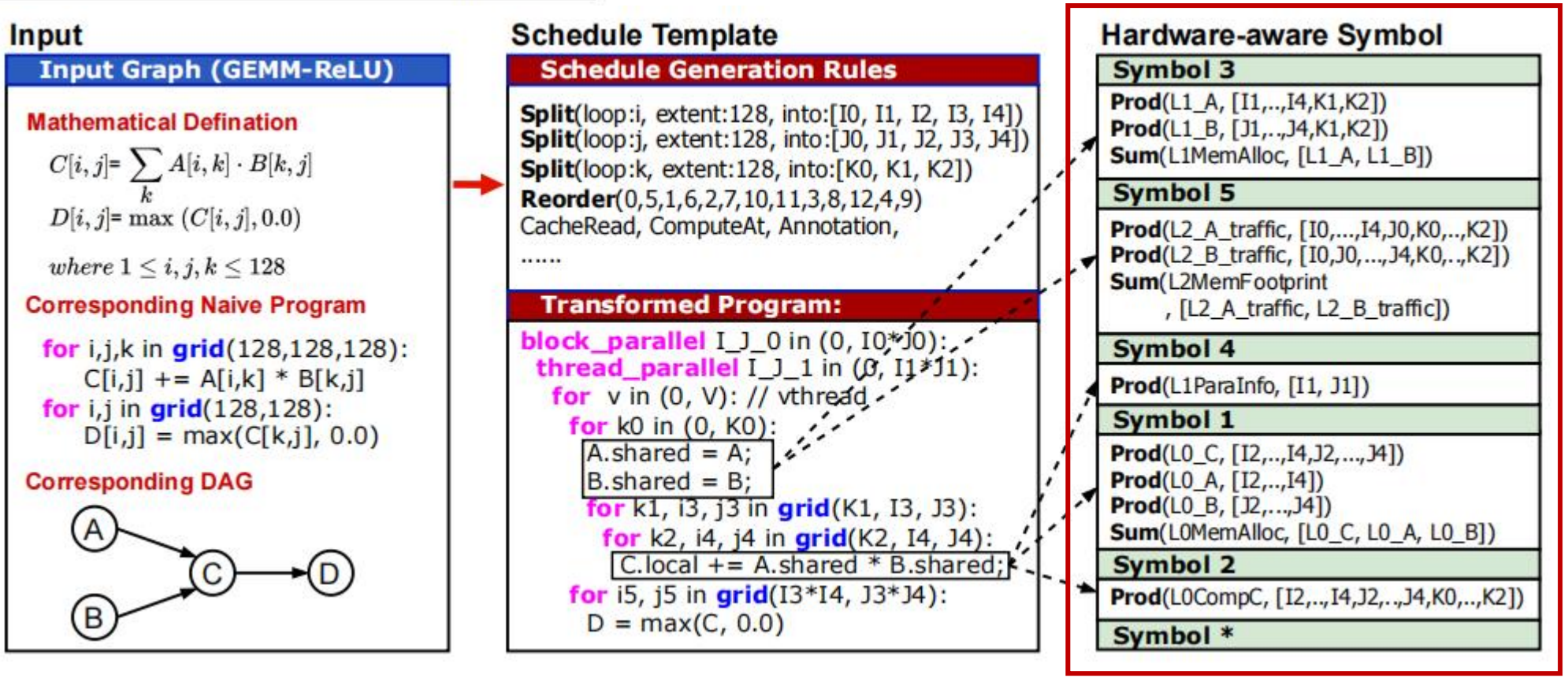
Mem	Symbol
L0	S1-L0MemAlloc, S2-L0CompCount
L1	S3-L1MemAlloc, S4-L1ParaInfo
L2	S5-L2MemFootprint, S6-L2ParaInfo S7-L2TransDim, S8-L2CompCount

L0（2 个）：寄存器的 “S1内存占用 + S2计算量”；
L1（2 个）：共享内存的 “S3内存占用 + S4并行度”；
L2（4 个）：全局内存的 “S5访问量 + S6并行度 + S7传输维度 + S8总计算量”。

Hardware-aware Symbols

Mem	Symbol
L0	S1-L0MemAlloc, S2-L0CompCount
L1	S3-L1MemAlloc, S4-L1ParaInfo
L2	S5-L2MemFootprint, S6-L2ParaInfo S7-L2TransDim, S8-L2CompCount

L0（2 个）：寄存器的 “S1内存占用 + S2计算量”；
L1（2 个）：共享内存的 “S3内存占用 + S4并行度”；
L2（4 个）：全局内存的 “S5访问量 + S6并行度 + S7传输维度 + S8总计算量”。



全局扫描
然后生成
Symbols表示

Symbol-based Analyzer

本质是一个**轻量级经验公式代价模型**

计算相关延迟

$$L_c^i = \frac{S8}{U_p}, \quad L_m^i = \frac{S5}{U_m}, \quad L_{total} = \sum_i (L_c^i + L_m^i) \qquad (1)$$

内存相关延迟

Symbol-based Analyzer

本质是一个**轻量级经验公式代价模型**

计算相关延迟

$$L_c^i = \frac{S8}{U_p}, \quad L_m^i = \frac{S5}{U_m}, \quad L_{total} = \sum_i (L_c^i + L_m^i) \qquad (1)$$

内存相关延迟

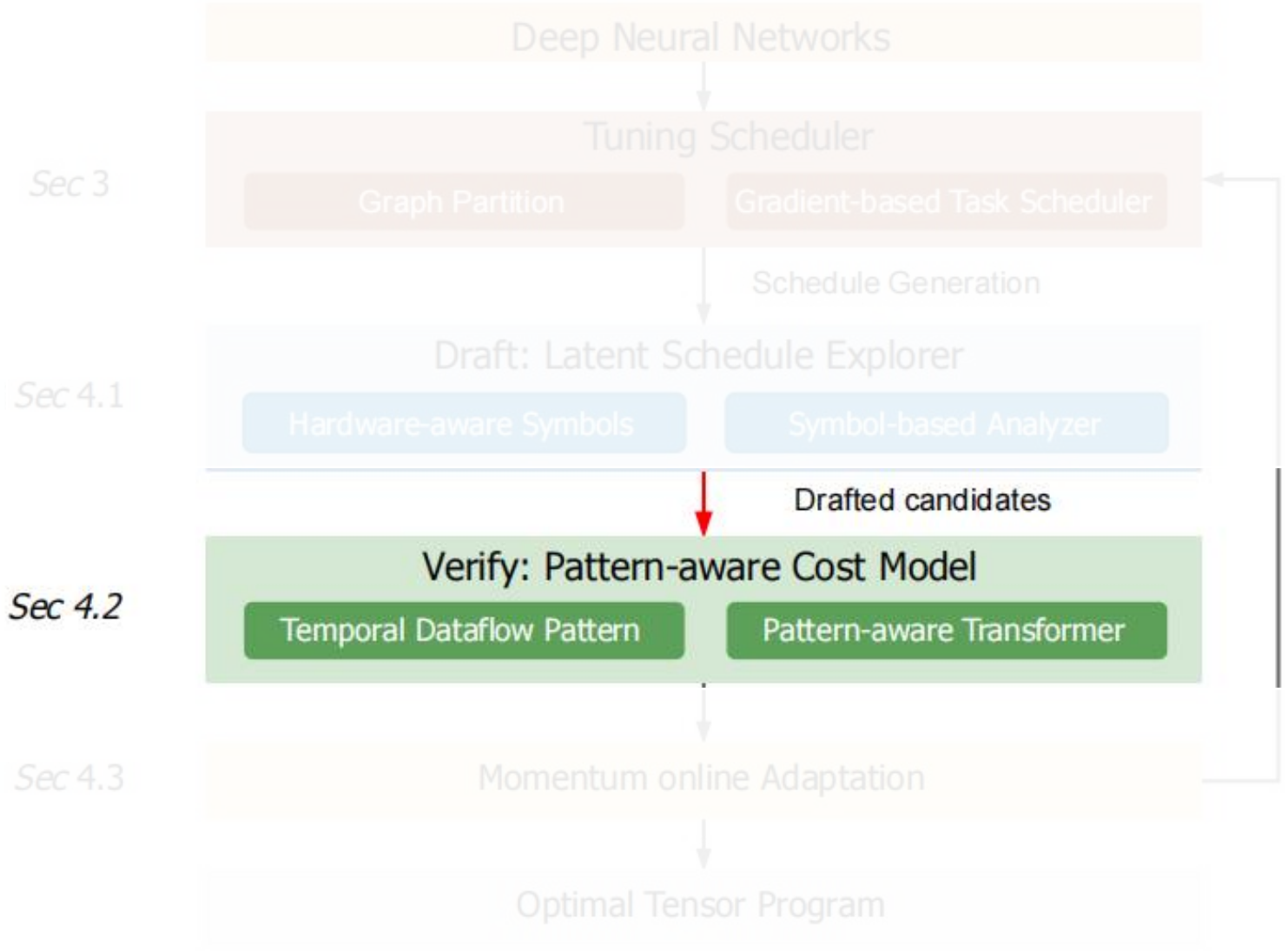
数学公式简单计算

高效筛选有
潜力候选集
(Schedule)

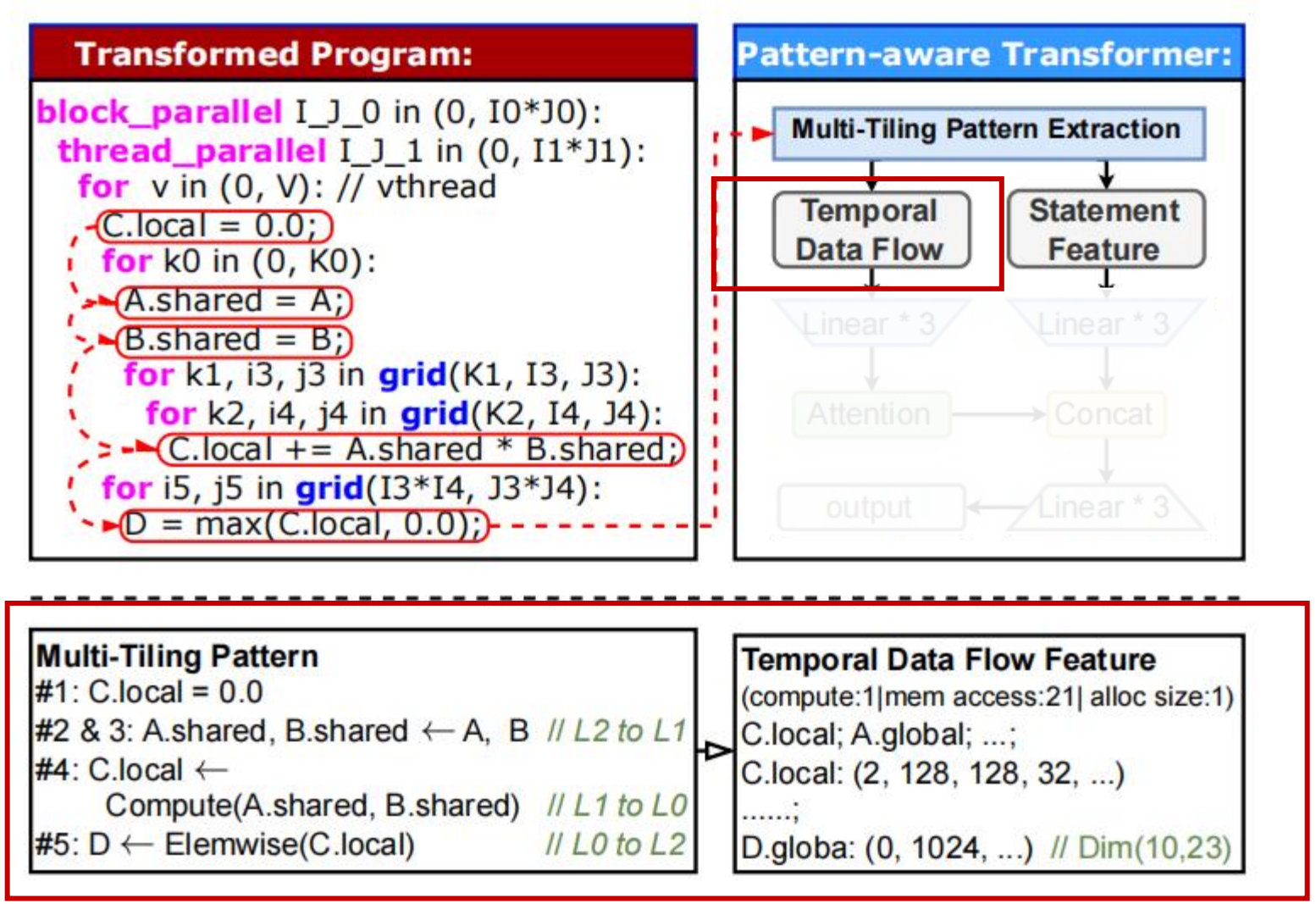
深度学习模型

低效筛选有
潜力候选集
(Schedule)

Verify: Pattern-aware Cost Model



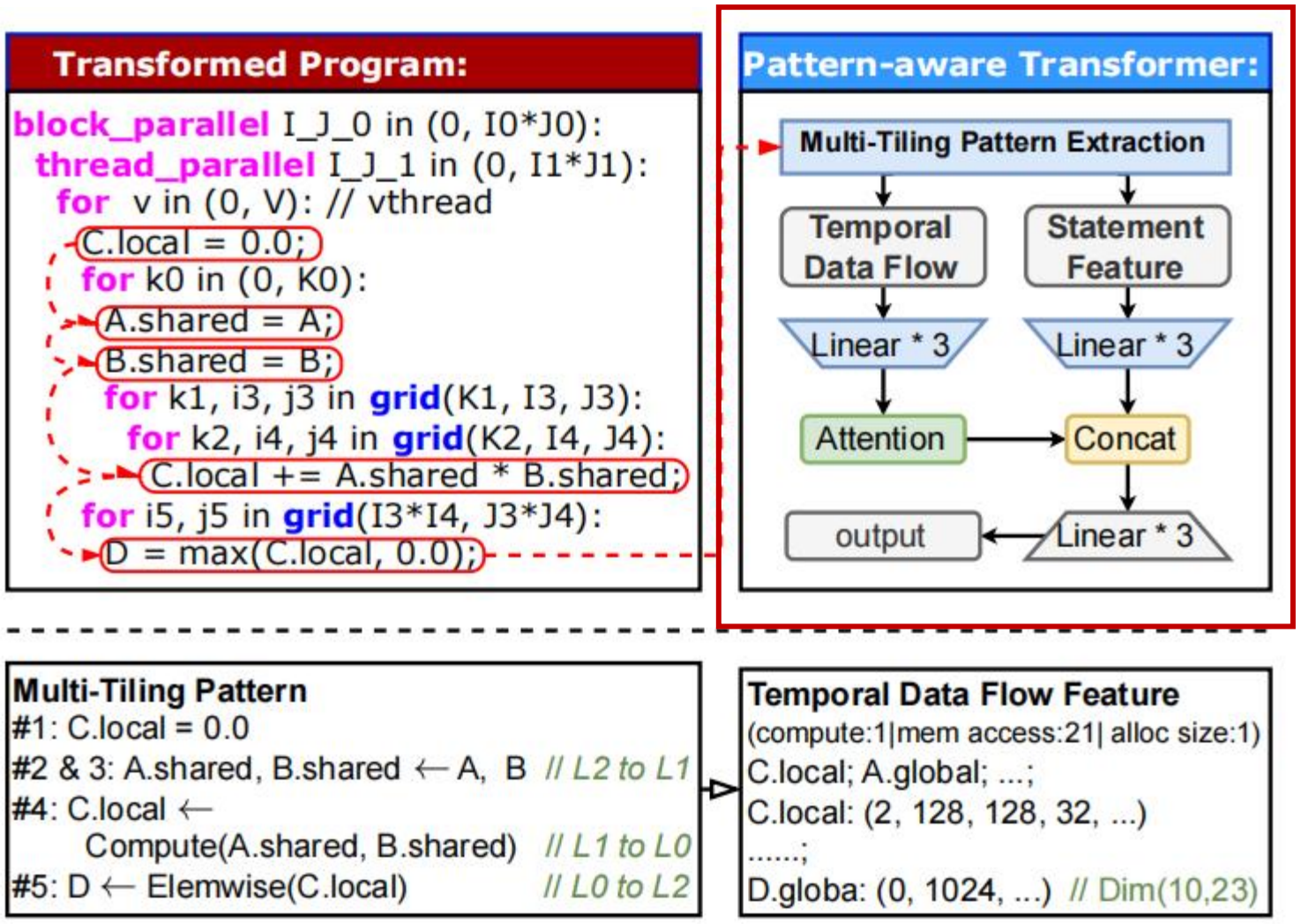
Temporal Dataflow Pattern



Statement-level features（语句级特征）是传统张量程序代价模型重点关注的特征类型

平铺（tiling）表征数据的流动过程（时序），解决传统特征忽略数据流依赖的问题

Pattern-aware Transformer

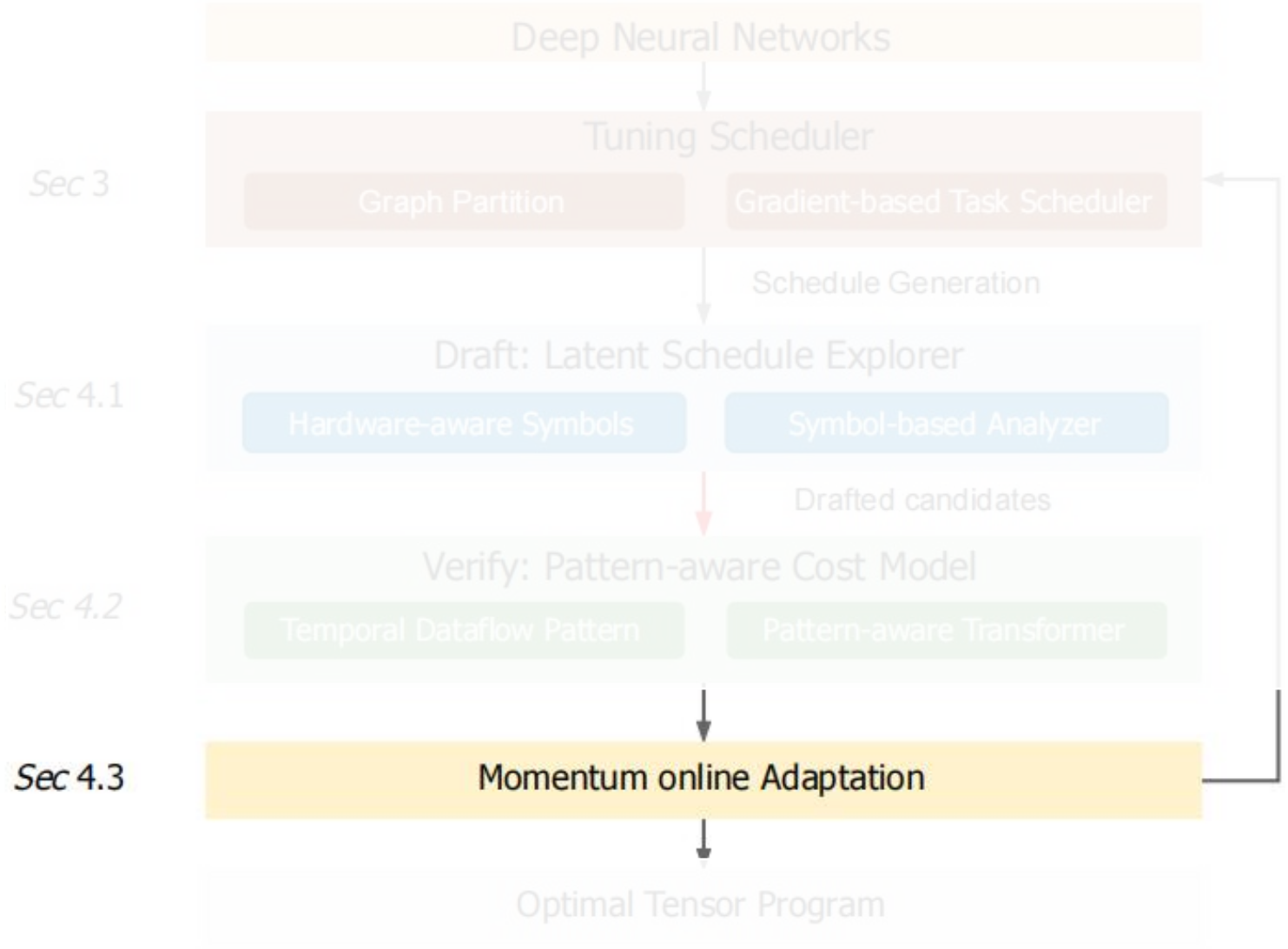


Statement-level features（语句级特征）是传统张量程序代价模型重点关注的特征类型

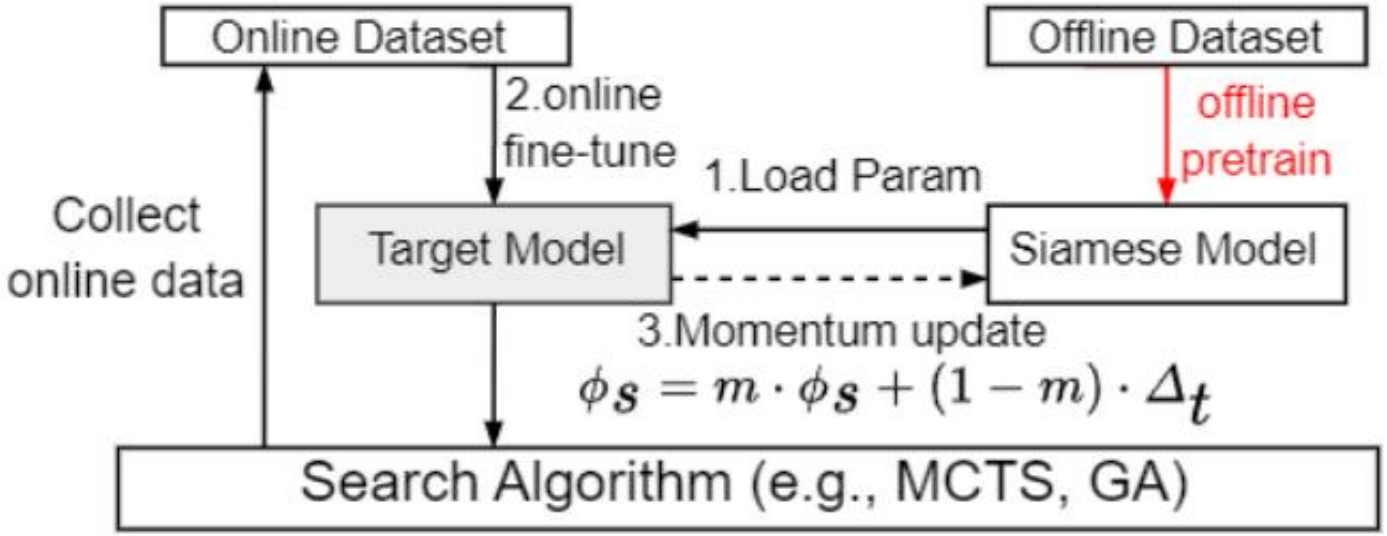
任务的核心**不是预测的延迟更准，而是排序/相对性能更准**

以归一化延迟为目标、LambdaRank 为损失函数，通过线性层输出相对性能评分

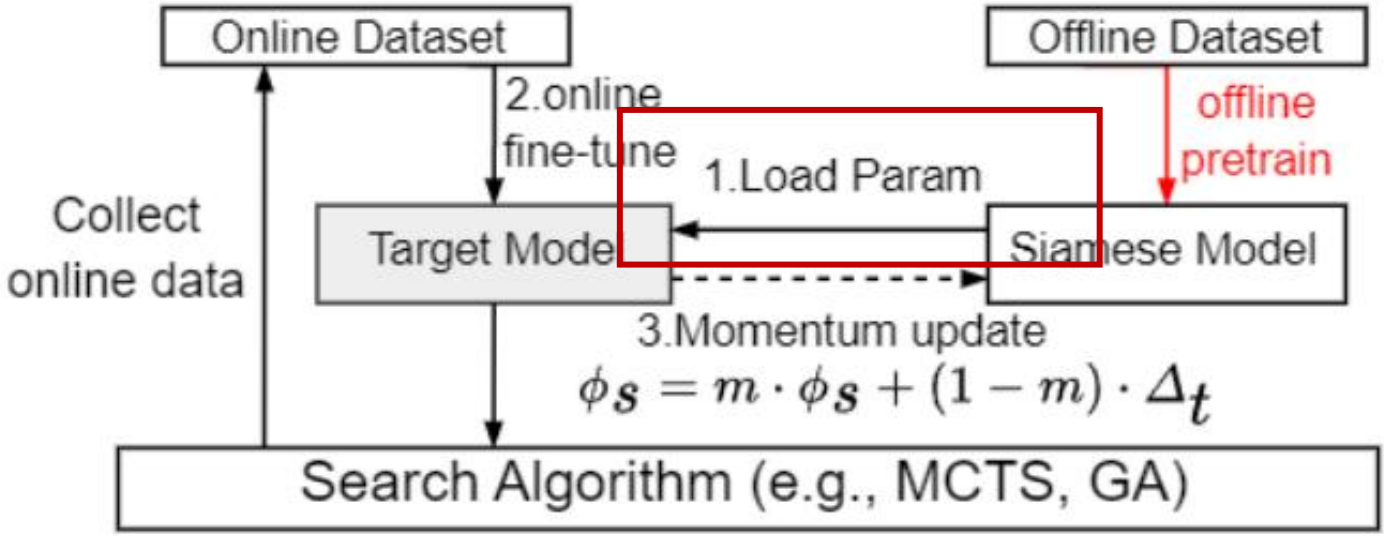
Verify: Pattern-aware Cost Model



Momentum online Adaptation (MoA)-Pruner



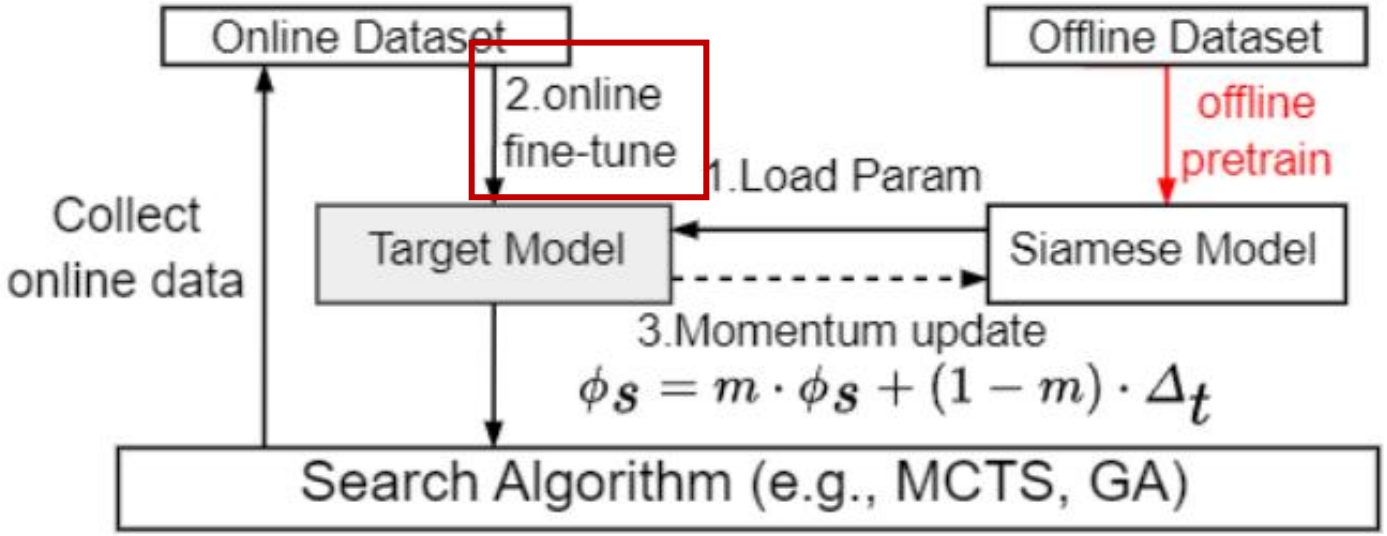
Momentum online Adaptation (MoA)-Pruner



相似架构的预训练的成本模型

1．Load Parm：是为了成本模型的初始权重不是随机的，而是具备一定知识。（可以类似CV领域的使用随机初始权重，和使用ImageNet预训练权重）

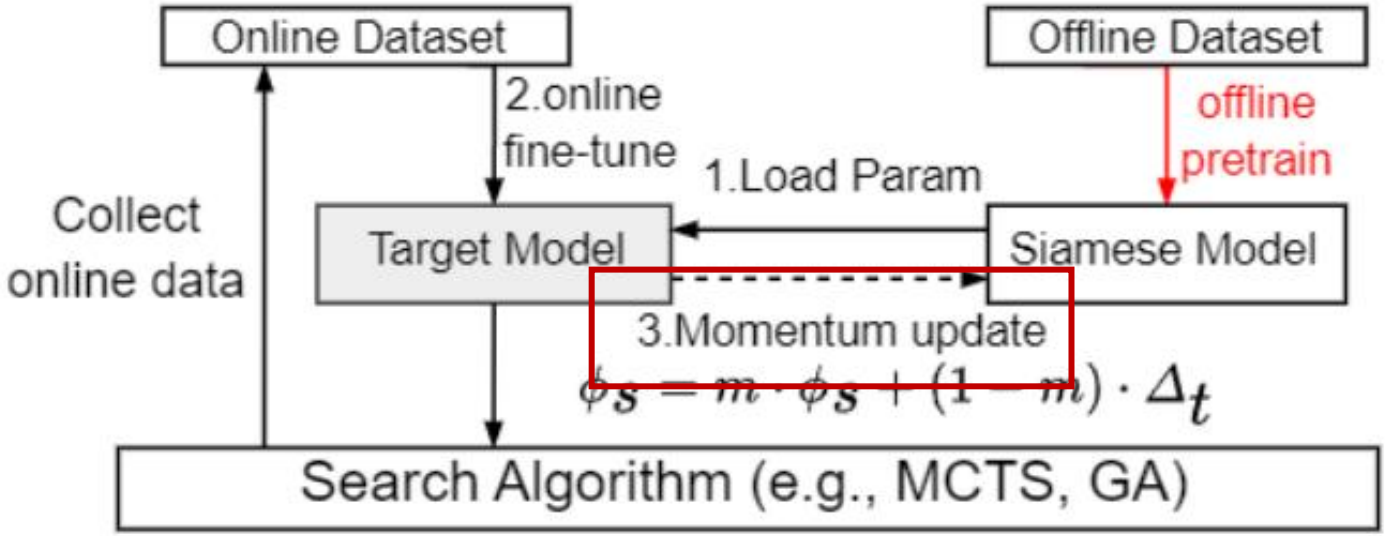
Momentum online Adaptation (MoA)-Pruner



相似架构的预训练的成本模型

1. Load Parm：是为了成本模型的初始权重不是随机的，而是具备一定知识。（可以类似CV领域的使用随机初始权重，和使用ImageNet预训练权重）
2. Online：与传统方案一致，采样+验证+微调

Momentum online Adaptation (MoA)-Pruner



相似架构的预训练的成本模型

1. Load Parm：是为了成本模型的初始权重不是随机的，而是具备一定知识。（可以类似CV领域的使用随机初始权重，和使用ImageNet预训练权重）
2. Online：与传统方案一致，采样+验证+微调
3. Momentum update：反向更新Siamese Model，使其更具通用性，为后续其他设备迁移提供经验

提纲

实验设置

- **DNN** workloads

Table 3. Evaluated DNN models in Pruner, with shapes and optimization precisions.

CNNs	Shape & Precision	Transformers	Shape & Precision
ResNet[18]	(1, 3, 224, 224) & (F)	Bert-B/T[15]	(1 & 4, 128) & (F, H)
WideResNet[43]	(1, 3, 224, 224) & (F)	GPT-2[30]	(1 & 4, 128) & (F, H)
Inception-V3[35]	(1, 3, 299, 299) & (F)	Llama[39]	(1 & 4, 128) & (F, H)
Densenet-121[19]	(1, 3, 224, 224) & (F)	OPT[46]	(1 & 4, 128) & (H)
Mobilenet-V2[32]	(1, 3, 224, 224) & (F)	Mistral[20]	(1 & 4, 128) & (H)
DCGAN[29]	(1, 100) & (F)	ViT[16]	(1, 3, 256, 256) & (F)
Deeplab-V3[9]	(1, 3, 224, 224) & (F)	DeTR[18]	(1, 3, 256, 256) & (F)

Table 4. Details for Transformer-based language models.

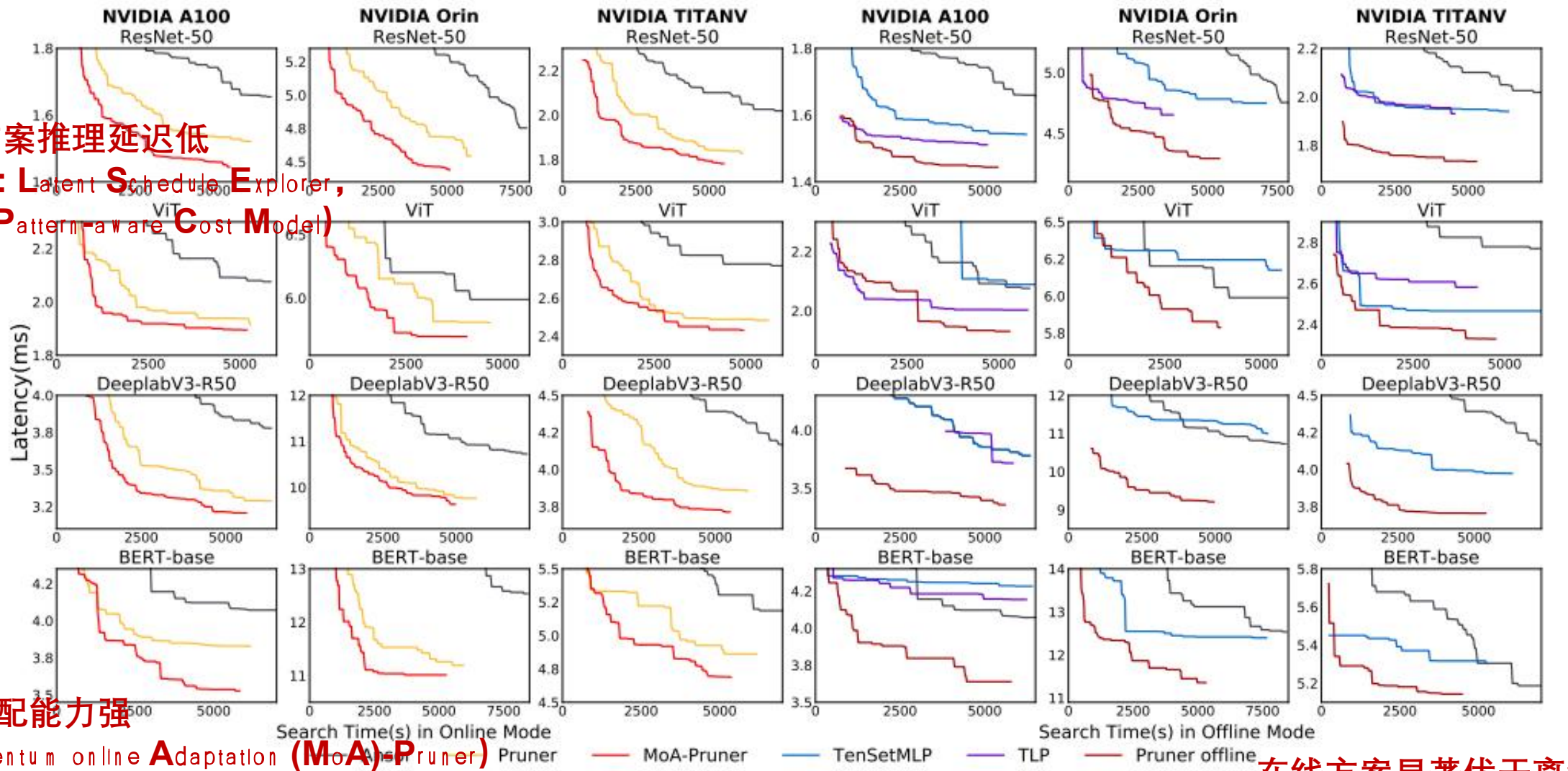
Model	layers	heads	hidden	intermediate
Bert-Tiny	6	8	512	2048
Bert-Base	12	12	768	3072
GPT-2	12	12	768	3072
OPT-1.3b	24	32	2048	8192
Llama	12	12	768	3072
Mistral-7b	32	32	4096	14336

- **P**latforms and baselines
  - A100 （服务器）
  - Titan V （服务器）
  - Jetson Orin-AGX （边缘设备）

- **B**aselines
  - 工业界：Ansor、MetaSchedule
  - 学术界：TenSetMLP [NeurIPS'21]、TLP [ASPLOS'23]、Felix [ASPLOS'24]、Adatune [NeurIPS'20]、 TLM [OSDI'24]。

最优推理延迟

所需方案推理延迟低
（Draft: Latent Schedule Explorer，
Verify: Pattern-aware Cost Model）

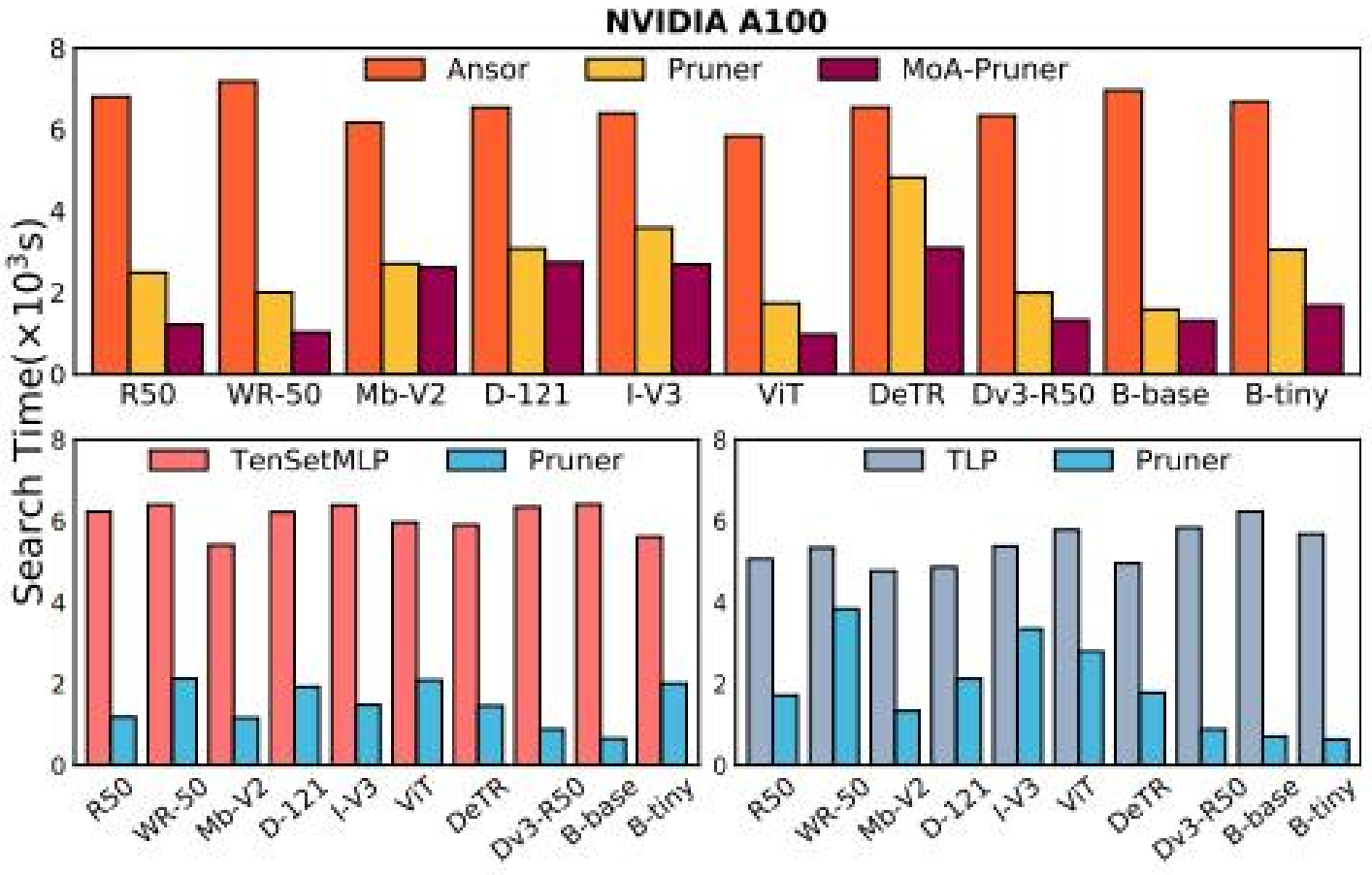


设备适配能力强
（Momentum online Adaptation (MoA)）

在线方案显著优于离线方案

横轴是搜索时间（基于在线/离线训练的成本模型），纵轴是筛选的最优方案的推理延迟

搜索效率

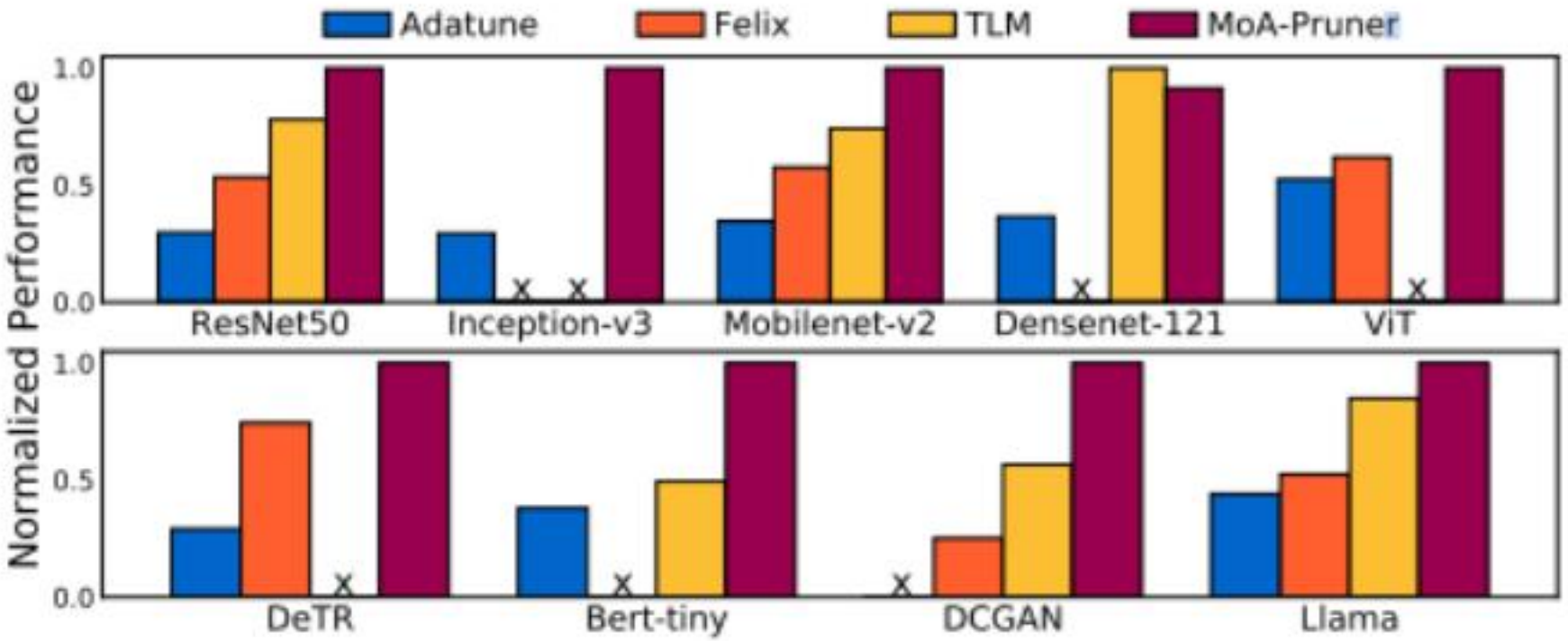


横轴是不同模型，纵轴是搜索时间

搜索时间显著减少

(Draft: Latent Schedule Explorer, Verify: Pattern-aware Cost Model)

最优推理延迟



横轴是不同模型，纵轴是归一化后的模型推理性能

具备更强的泛化性，不是着眼特定算子，而是重点关注设备特征计算

（Momentum online Adaptation (MoA)-Pruner）

总结与思考

总结

- **框架**：提出一种新型的张量程序调度方案探索机制，即采用先草拟后验证”的范式加速搜索过程。
- **技术**：引入了动态在线适应技术，解决了预训练成本模型在跨平台中难迁移的问题，使得在线成本模型能够高效适应任何平台。
- **实验**：通过与现有最先进方法在三个基于GPU的平台上（包括CUDA核心和Tensor核心）的对比分析，突显了Pruner的普适性。全面的实验表明，Pruner在调优质量和效率之间取得了显著的平衡，其性能远超现有方法。并认为Pruner背后的核心思想与现有的基于搜索的方法相辅相成。

思考

1. 这个paper有什么问题，基于这个paper还能做什么?

- 本文重点关注GPU，重点关注了latency，是否可以将问题聚焦与边缘计算场景，并且还可以考虑energy，考虑多目标优化探索。

2. 这个paper提到的idea，能不能用在自己的方向/project上面?

- 本文的方案是一种特别适合和硬件设备相关的，以性能为导向的探索类问题。其理论上也可以用于神经网络超网搜索最优子网的问题，至少可以用于筛选部分探索候选集（无需真实推理即可排除不满足延迟约束的候选网络）。

3. 这个paper能不能泛化，需要较为熟悉这个小方向?

- 本文的面向的设备是GPU。从理论上可知，只要可以抽象出不同设备的共有特征，或者采用双重特征（1重特征表征什么类型设备，2重特征表征具体计算能力特征），是不是就可以泛化到更多的设备场景，甚至实现GPU-→CPU, TPU等。

感谢各位老师和同学！
请大家提出宝贵意见！