Bayesian Code Diffusion for Efficient Automatic Deep Learning Program Optimization

Authors： Isu Jeong and Seulki Lee

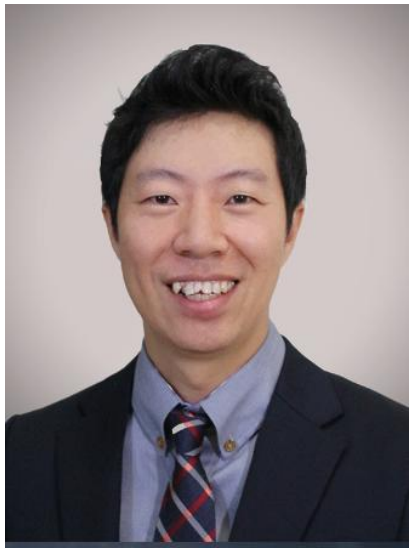
Ulsan National Institute of Science and Technology

OSDI'25

Reporter: Hangshuai He

研究方向:

- On-device machine learning on embedded/mobile/IoT devices
- Embedded Computer Vision, Embedded NLP, Embedded Reinforcement Learning
- Efficient model (deep learning) inference, training, and adaptation



Author：Associate Professor SEULKI LEE

提纲

- *背景*
- *系统设计*
- *实验评估*
- *总结*

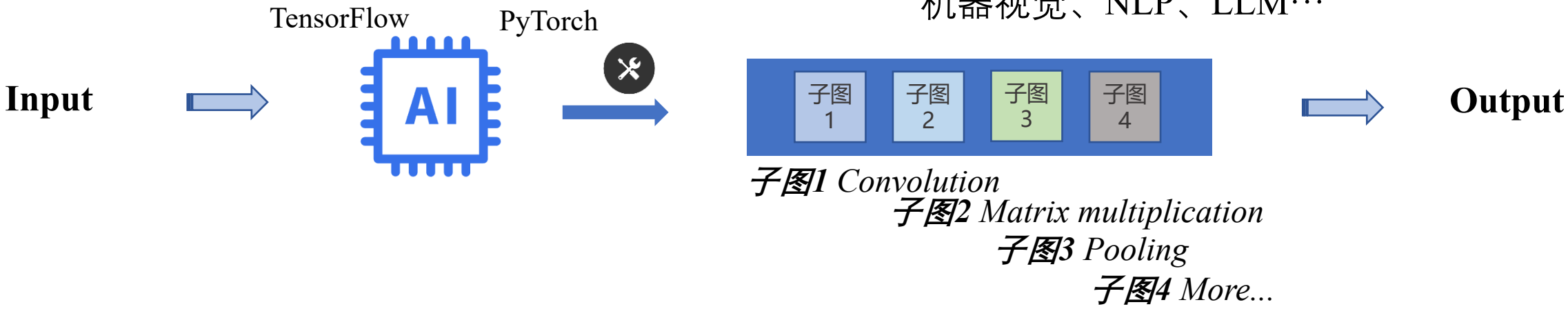
提纲

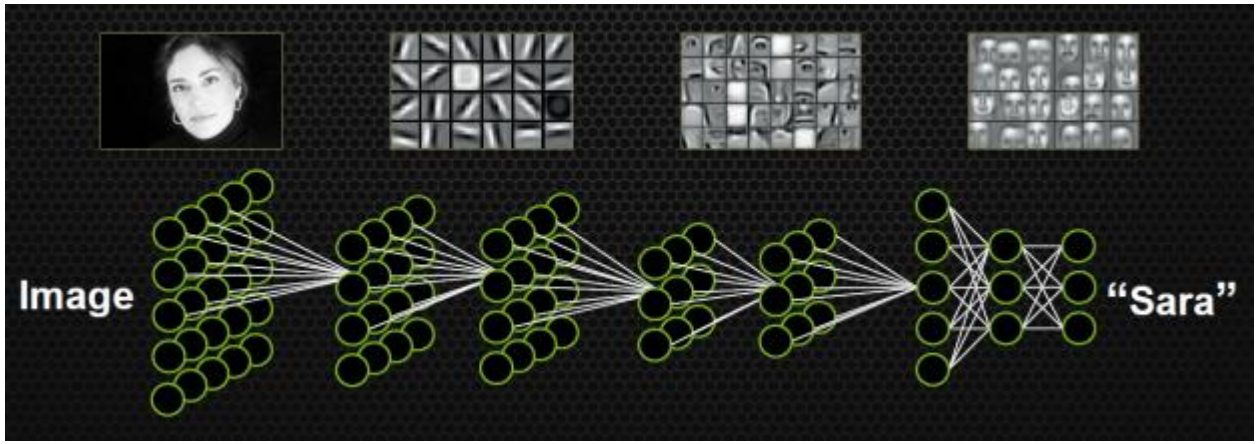
- ***背景***
- *系统设计*
- *实验评估*
- *总结*

背景

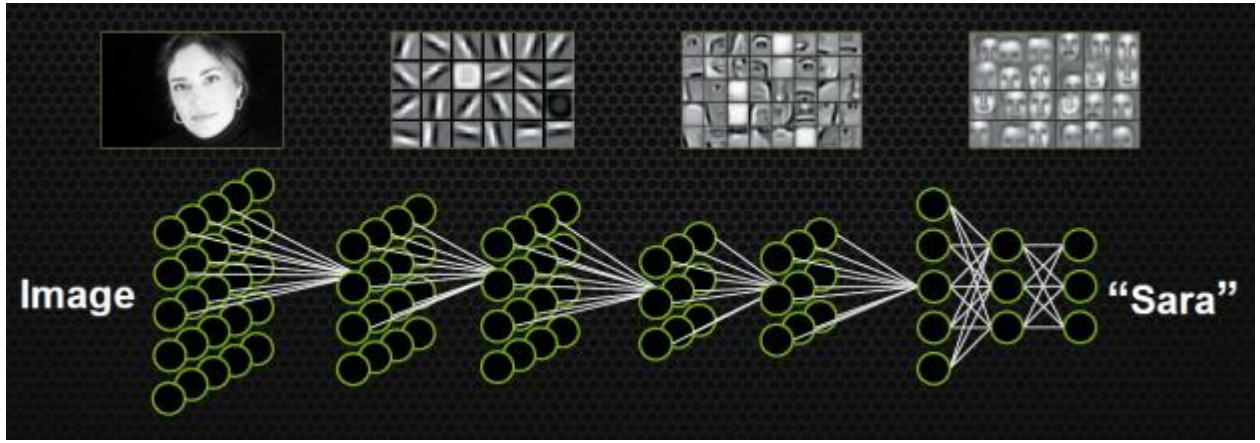
深度学习任务

机器视觉、NLP、LLM…





背景



高维的深度学习被分解下来，其最核心、最本质的计算单元就是一次次的张量计算。

在深度学习的python代码中：

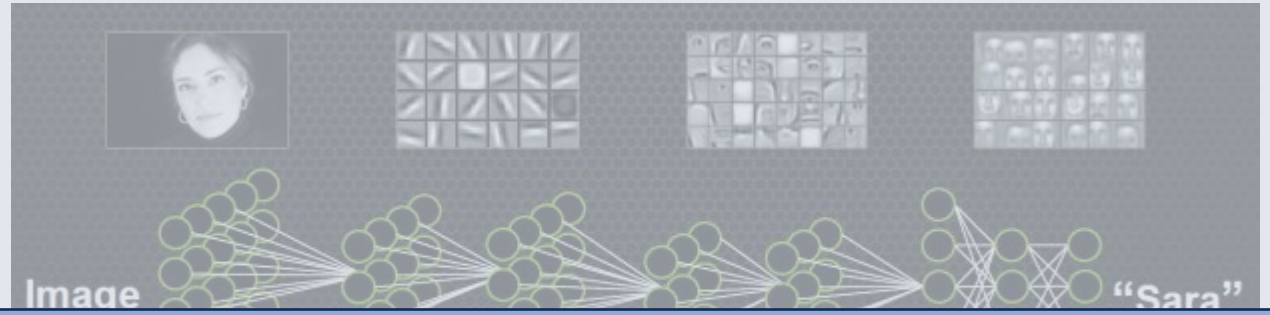
```
conv_layer = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3)
input_image = torch.randn(1, 3, 32, 32)

output_feature_map = conv_layer(input_image)
```

在底层：

```
output_manual= （）
for b in range(batch):
for c_out in range(out_channels):
        for y in range(h - kh + 1):
            for x in range(w - kw + 1):
                # 1. 切片：从输入图片中提取当前窗口的区域
                image_patch = image_np[b, :, y:y+kh, x:x+kw]
                # 2. 乘加：将窗口区域与对应的卷积核进行元素相乘再求和
                convolution_sum = np.sum()
                # 3. 加偏置：加上偏置项
                output_manual[b, c_out, y, x] = convolution_sum + biases[c_out]
```

背景



在深度学习的python代码中：

```
conv_layer = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3)
input_image = torch.randn(1, 3, 32, 32)

output_feature_map = conv_layer(input_image)
```

在底层：

```
output_manual= （）
for b in range(batch):
for c_out in range(out_channels):
        for y in range(h - kh + 1):
biases[c_out]
```

张量计算：深度学习任务中最本质的计算单元

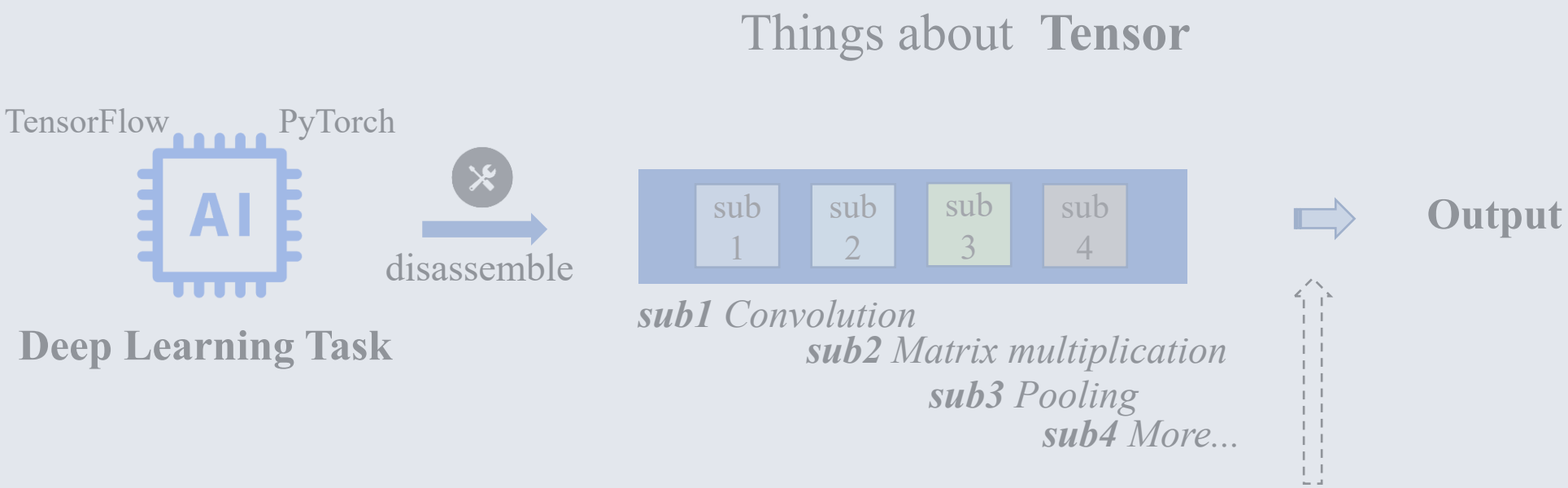
背景

张量计算优化任务

子图1 卷积 Convolution

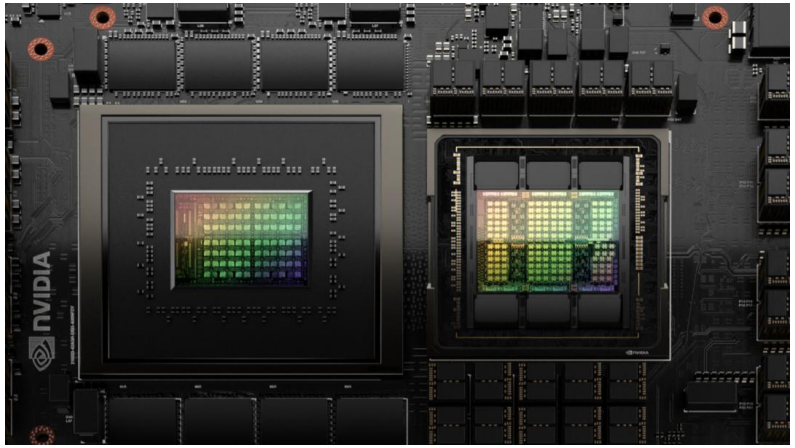
- **循环分块 (Tiling)**：把大图片拆成**多大**的小块处理，以适应CPU或GPU的缓存大小
- **循环顺序**：是先算完图片的第一行再算第二行，还是先算完第一列再算第二列？不同的访问顺序，数据在内存中的效率差别极大。
- **并行化**：有这么多CPU核心和GPU线程，这个计算任务到底该怎么分配下去才能完美协作、不互相等待？
- **算法选择**：是用im2col+GEMM还是Winograd算法…

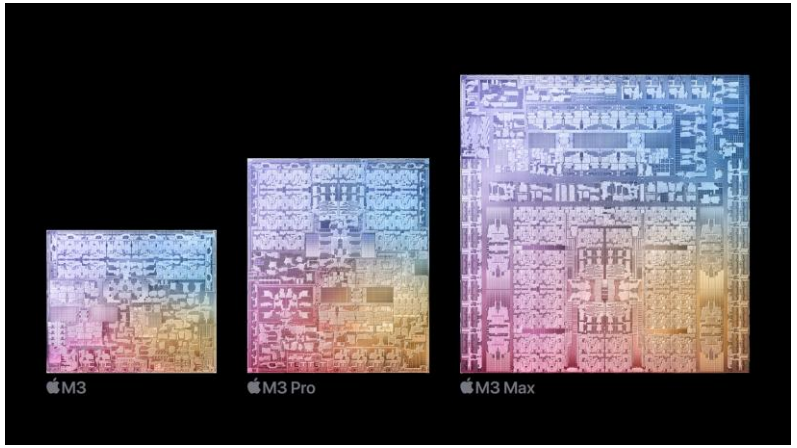
背景



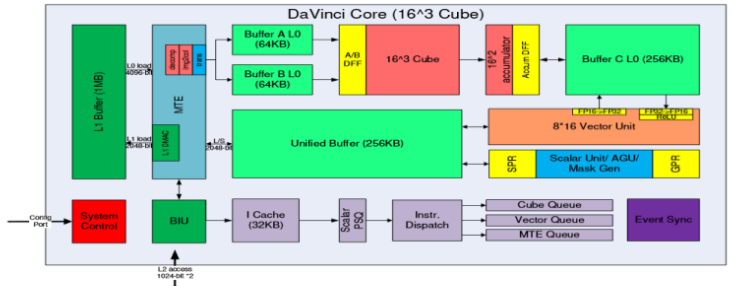
张量计算优化任务：决定一次次的张量计算如何更高效运行

背景





DaVinci Core



- **Cube**: 4096(16^3) FP16 MACs + 8192 INT8 MACs
- **Vector**: 2048bit INT8/FP16/FP32 vector with special functions (activation functions, NMS- Non Minimum Suppression, ROI, SORT)
- Explicit memory hierarchy design, managed by MTE

- **获得高性能（即低延迟）且高效能（即编译时间短）的张量程序对深度学习至关重要**
- **计算硬件架构日益复杂**
- **为充分利用硬件性能，张量程序需做出诸多“决策”**
  - 决定循环轴的切分尺寸
  - 设置展开步数(嵌套循环中，将循环体内的代码复制多次)
  - 选择算子的计算位置
  - 制定并行化和向量化策略
  - 决定线程绑定
  - …

背景

针对神经网络，生成高性能的张量程序十分重要

cuDNN ***oneDNN***

[OSDI' 18] ***TVM****: An Automated End-to-End Optimizing Compiler for Deep Learning*

背景

[OSDI' 18] *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*

[OSDI' 20] *Ansor: Generating High-Performance Tensor Programs for Deep Learning*

- 决定循环轴的切分尺寸
- 设置展开步数
- 选择算子的计算位置
- 制定并行化和向量化策略
- 决定线程绑定
- …

→ 不同的可选参数 → 庞大的候选组合

预定义规则 ⇢ **模版** ⇢ **搜索空间**

背景

现有方法

1. 通过启发式约束对决策空间进行修剪，或者套用模版

- 对于不同硬件和子图，难以选择合适的性能指标；更优的指标值未必带来更佳的性能表现。
- 有时，启发式约束会强行缩减搜索空间，从而剔除高性能候选方案，导致性能不佳。

启发式

- 一套硬编码的“如果...就...”规则，用来快速做决策。
- 例如：“因为NVIDIA GPU的基本并行单位（warp）是32个线程，所以循环分块时，矩阵大小必须是32的倍数，这样能最大化利用硬件。”
- 但对于某个特定大小的卷积核，可能48x48的分块大小计算起来效率更高。

模板

- 专家为优化任务设计的一个带有一些“空格”的“代码框架”。
- 例如：专家为卷积编写了一个模板，只允许在[16, 32, 64]这几个值里选择分块大小。自动调优工具会把这些组合都试一遍。
- 但如果最优的分块大小其实是48，那么无论怎么搜索，都永远找不到这个最优解。

背景

现有方法

2.自动搜索一组定义程序代码（sketches）的参数，通过成本模型评估效率

Sketch：机器给出的更自由、多样化的“模板”，只定义了宏观的策略和步骤，不涉及精确的细节

- 决定循环轴的切分尺寸
- 设置展开步数
- 选择算子的计算位置
- 制定并行化和向量化策略
- 决定线程绑定
- …

对于一个真实的矩阵乘法的Sketch，其构成可能是：
[规则1: 循环分块 (?,?)-> 规则2: 算子融合(?,?) -> 规则3: 并行化(?)]
或者是：
[规则1: 算子融合 (?,?)-> 规则2: 矩阵展开(?,?) -> 规则3: 并行化(?)]

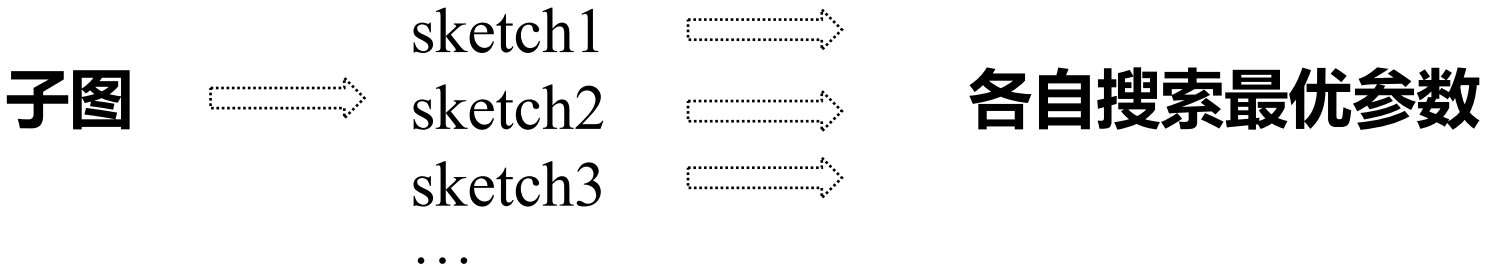
优化策略已经被确定
但具体数值待定

根据计算特性不同，
Ansor系统性地给出多个
可能的、合理的优化方案

背景

现有方法

2.自动搜索一组定义程序代码（sketches）的参数，通过成本模型评估效率



- 尽管这类自动调优技术能够找到执行延迟更低的深度学习程序，但在扩展的搜索空间中进行广泛的程序探索通常会导致程序优化时间的大幅增加。
- 每次搜索，如果在真实硬件上测量，成本太高，所以引入一个模型来模拟评估。
- 但性能成本模型仍然依赖于可靠的特征工程。

从现有方法中看到的机会

1. 每个子图都被分配了一个独立的搜索空间和一组独特的优化方案

- 尽管子图之间可能存在重叠（计算流程十分相似），但每个子图都被分配了独立的搜索空间和独特的优化集（sketch合集），这可能造成冗余。

2.现有的自动调优方法 执行大量的随机搜索来识别最优程序代码

- 不仅耗时，还会导致程序优化效率低下。
- 实际上，搜索的这个过程存在一个“从更优初始点开始” 的可能。

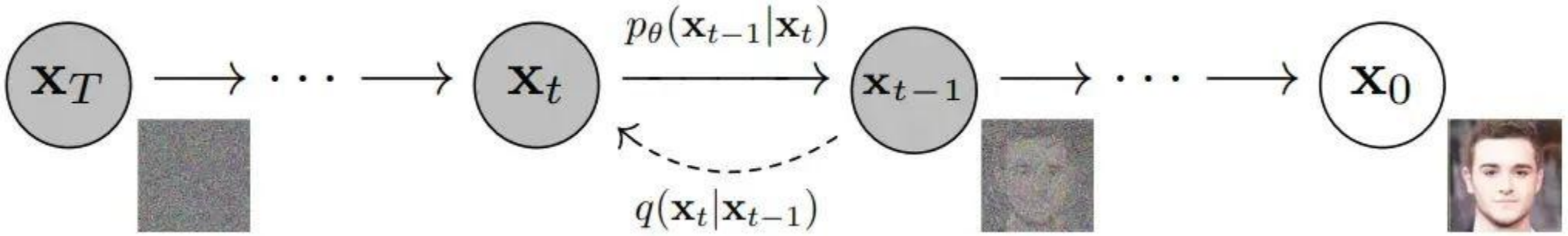
提纲

- 背景
- **系统设计**
- 实验评估
- 总结

系统设计

扩散模型（Diffusion Model）

生成一位欧美男性的肖像照



纯噪声图片 ⟶ 模糊的轮廓 (继续去噪) ⟶ 清晰的欧美男性肖像照

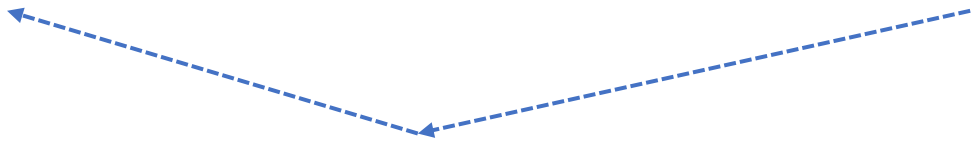
贝叶斯优化（ Bayesian Optimization ）

目标：用**最少的尝试次数**，找到一个“黑盒函数”的最优解

- 贝叶斯优化提出了三个概念：先验 (Prior)；证据 (Evidence)；后验 (Posterior) 。
- 先验——优化的起点：黑盒中的初始位置，自身对于自身位置的一个猜测。
- 证据——优化的转折点：黑盒中观察到的一个新数据（可能会**支持**先验，也可能会**挑战**甚至**推翻**它）。
- 后验——优化的下一步：在结合了“**先验**”和“**新证据**”之后，你得出的、经过修正的、对自己所处位置的**新判断**。

一个动态的公式：

先验 (初始猜测) + 新证据 (观测数据) → 后验 (修正后的结论)



系统设计

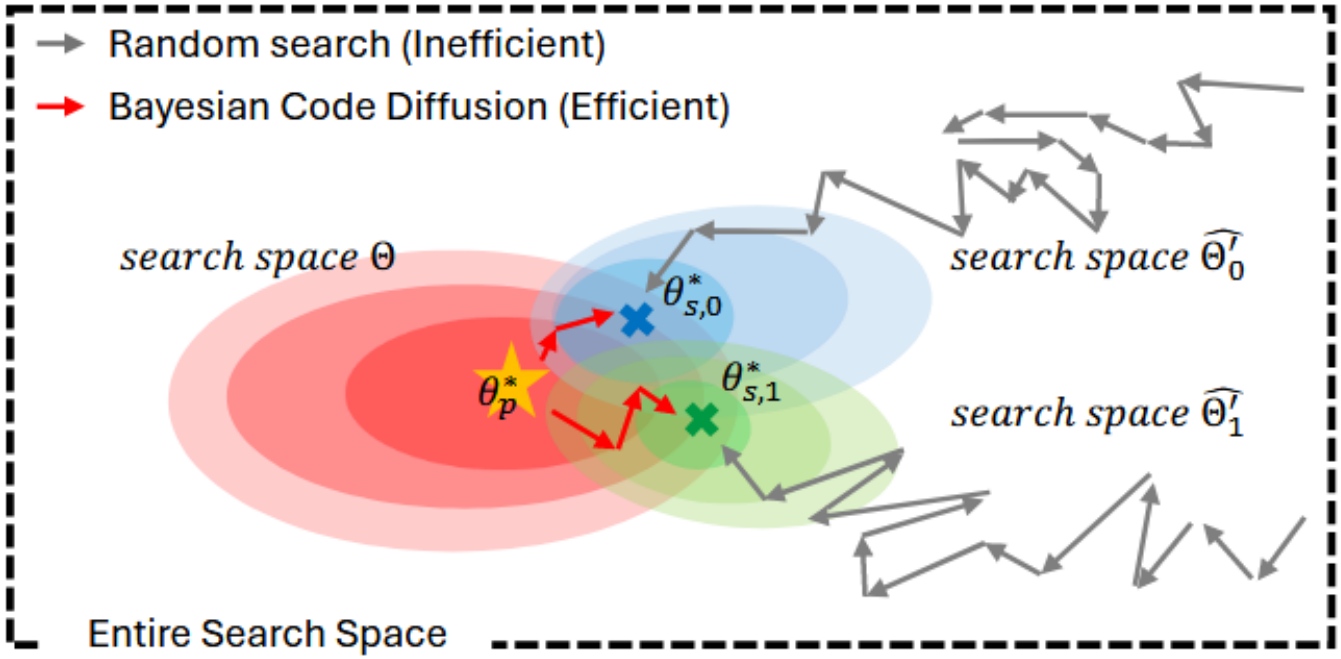


Figure 2: The concept of Bayesian code diffusion: a sufficiently optimized *prior* parameter of one subgraph (θ^*_p) is propagated to similar subgraphs (*posteriors*). Then, *posterior* parameters ($\theta^*_{s,\{0,1\}}$) are derived and refined from the *prior* for each subgraph via code diffusion using a Bayesian formulation, enabling efficient deep learning program optimization.

机会：

1. 尽管子图之间可能存在重叠，但每个子图都被分配了独立的搜索空间和独特的优化集。

2. 现有的张量程序搜索过程，存在一个“从更优初始点开始” 的可能。

先验信念 (Prior)：红色区域和 θp∗
证据 (Evidence) :搜索与评估的过程
后验信念 (Posterior):红色箭头指向的最终结果

观察1：草图（sketch）的共性

BERT

Subgraph	Subgraph Description	Sketches
$\mathcal{G}_0$	batch_matmul_3	S_0
$\mathcal{G}_1$	variance	S_1
$\mathcal{G}_2$	batch_matmul_1	S_0
$\mathcal{G}_3$	softmax_broadcast_to	S_2
$\mathcal{G}_4$	batch_matmul	S_0
$\mathcal{G}_5$	batch_matmul_4	S_0
$\mathcal{G}_6$	batch_matmul_2	S_0
$\mathcal{G}_7$	mean	S_3

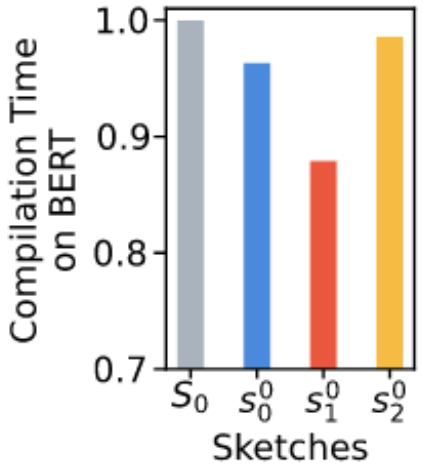


Table 2: (Left) The subgraphs in BERT [11] and their sketches obtained by applying Ansor [47], where subgraphs $\mathcal{G}_{0,2,4,5,6}$ produce the same sketches $S_0 = \{s_0^0, s_1^0, s_2^0\}$.

结论：八个子图中有五个共享相同的sketch，证明其操作存在相似性
但因为当前方法的不完善，却依然被分配了独立的程序优化任务和不同的搜索空间

观察1：草图（sketch）的共性

BERT

Subgraph	Subgraph Description	Sketches
$\mathcal{G}_0$	batch_matmul_3	S_0
$\mathcal{G}_1$	variance	S_1
$\mathcal{G}_2$	batch_matmul_1	S_0
$\mathcal{G}_3$	softmax_broadcast_to	S_2
$\mathcal{G}_4$	batch_matmul	S_0
$\mathcal{G}_5$	batch_matmul_4	S_0
$\mathcal{G}_6$	batch_matmul_2	S_0
$\mathcal{G}_7$	mean	S_3

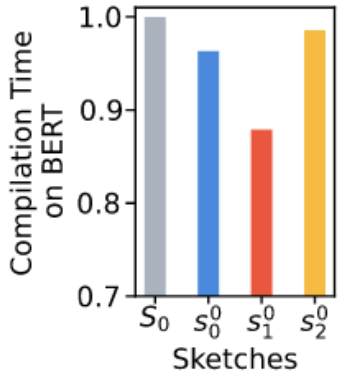
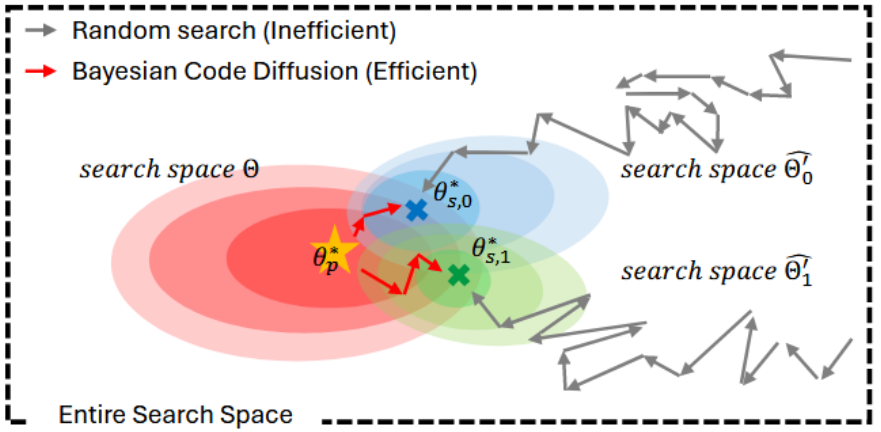


Table 2: (Left) The subgraphs in BERT [11] and their sketches obtained by applying Ansor [47], where subgraphs $\mathcal{G}_{0,2,4,5,6}$ produce the same sketches $S_0 = \{s_0^0, s_1^0, s_2^0\}$.



结论：八个子图中有五个共享相同的sketch，证明其操作存在相似性
但因为当前方法的不完善，却依然被分配了独立的程序优化任务和不同的搜索空间

观察2：基于单个sketch的优化

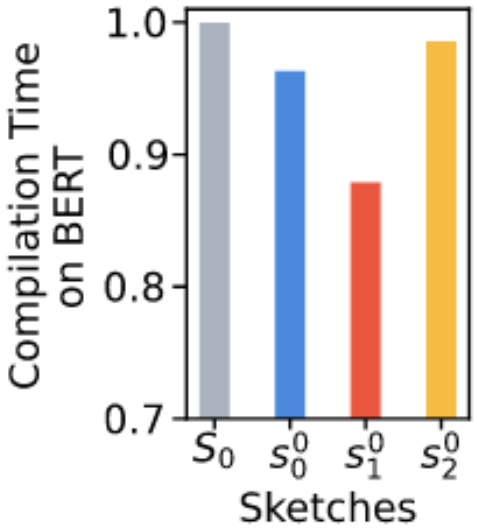


Figure 3: (Right) The compilation time of the subgraphs $\mathcal{G}_{0,2,4,5,6}$ taken by Ansor [47] on the operation `batch_matmul` in BERT [11] using the sketches S_0 and each sketch $s_i^0 \in S_0$ on a GPU. Here, '1.0 (gray)' denotes the time required when using all sketches $S_0 = \{s_0^0, s_1^0, s_2^0\}$.

- 对于BERT的**batch_matmul**操作，得到一系列TVM定义的sketch（ S_0^0 S_1^0 S_2^0 ）

  - 四种不同的优化策略（S_0, s^0_0, s^1_0, s^2_0），
  - 在分别独立进行优化时，各自花费了多长时间，才首次达到了一个预先设定的、同样的高性能目标。
  - 纵轴展示的是将各自的'达标时间'与最慢策略（S_0）的时间进行比较后得出的相对比例。"

结论：专注于单个sketch进行优化，反而会更有效率

颜色的深浅，代表**最优解**的相似程度

观察3：相似子图的参数距离很相近

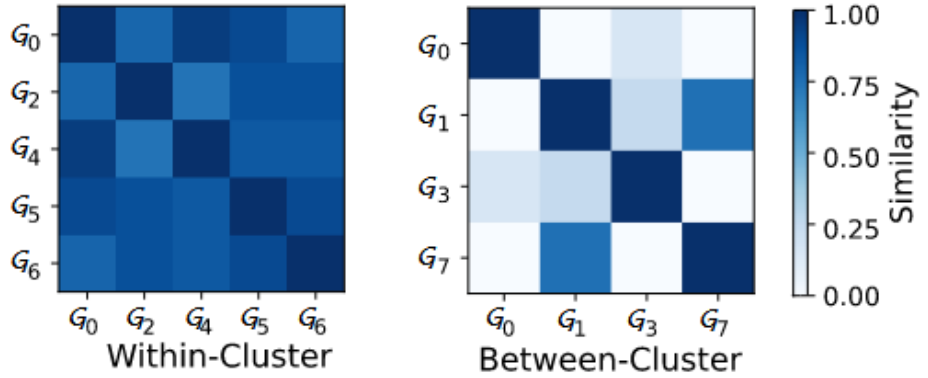


Figure 4: The cosine similarities between configurations optimized by Ansor [47]. The subgraphs $\{\mathcal{G}_0, \mathcal{G}_2, \mathcal{G}_4, \mathcal{G}_5, \mathcal{G}_6\}$ are optimized from the same sketch, while $\{\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_3, \mathcal{G}_7\}$ have different sketches. The darker, the higher similarity. The details of distance measurement is in Fig. 10.

左图：$\{G_0\, G_2\, G_4\, G_5\, G_6\}$这组子图在经历了完整的自动调优过程后，最终都共同选择了**同一个sketch方案**作为它们各自的最佳性能实现路径。”

右图：$\{G_0\, G_1\, G_3\, G_7\}$这组子图各自的最佳优化方案来自**完全不同的sketch**。因此，它们的最终参数配置彼此之间**差异巨大**。

结论：相似的子图，其最优解也相似

实验实施

相同sketch的子图，被分配了独立的程序优化任务和不同的搜索空间

对于拥有相同草图的子图
如何分组?

专注于单个sketch会更有效率

相似的子图组内如何选出率先被优化的**先验**“榜样”。

相似的子图，其最优解也相似

如何优化“榜样”，并将优化经验在组内扩散。

对于拥有相同草图的子图**如何分组**?

Subgraph	Subgraph Description	Sketches
$\mathcal{G}_0$	batch_matmul_3	S_0
$\mathcal{G}_1$	variance	S_1
$\mathcal{G}_2$	batch_matmul_1	S_0
$\mathcal{G}_3$	softmax_broadcast_to	S_2
$\mathcal{G}_4$	batch_matmul	S_0
$\mathcal{G}_5$	batch_matmul_4	S_0
$\mathcal{G}_6$	batch_matmul_2	S_0
$\mathcal{G}_7$	mean	S_3

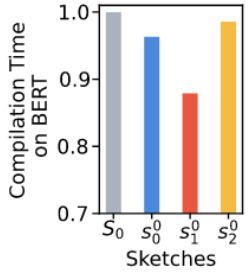
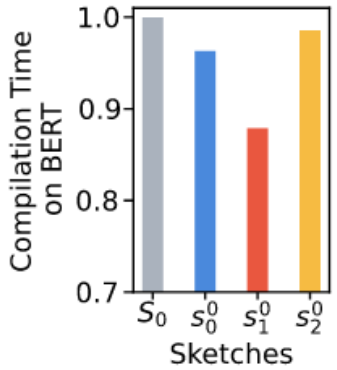


Table 2: (Left) The subgraphs in BERT [11] and their sketches obtained by applying Ansor [47], where subgraphs $\mathcal{G}_{0,2,4,5,6}$ produce the same sketches $S_0 = \{s_0^0, s_1^0, s_2^0\}$.



- 基于观察得到，结构类似的子图会生成完全相同的一套sketch。又发现，当一个任务拥有一套（多个）可选的sketch方案时，将优化资源专注于其中最高效的那一个，能大大缩短编译时间。

- 因此，我们首先根据子图生成的sketch集合是否相同，来对它们进行分组（聚类）。
- 然后，在每个组内选出一个“先验”子图，集中资源为它找到那个唯一的“最优sketch方案”。
- 最后，让组内其他的“后验”子图直接继承这个“最优方案”，从而自动“筛选”掉其他所有低效的sketch。

对于拥有相同草图的子图**如何分组**?

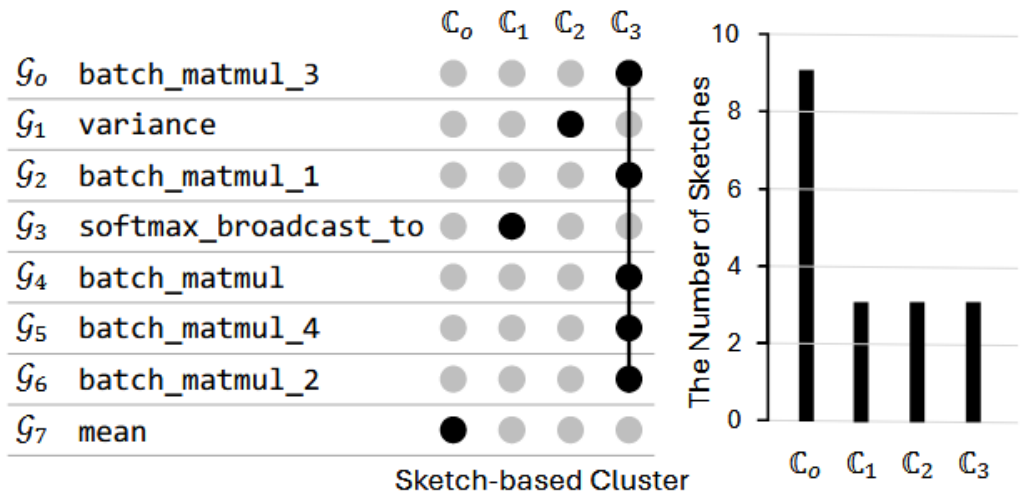
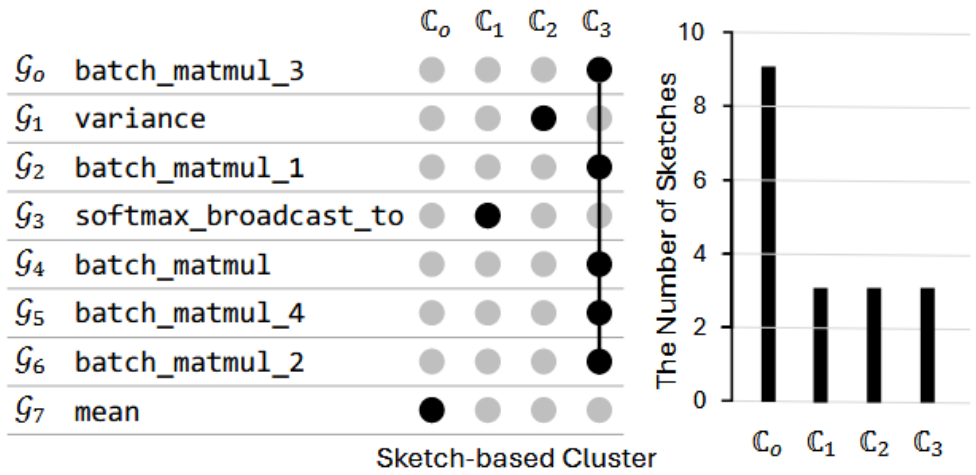


Figure 5: An example of sketch clustering applied to BERT [11], comprising four clusters: $\mathbb{C}_0$, $\mathbb{C}_1$, $\mathbb{C}_2$, and $\mathbb{C}_3$. Subgraphs within the same cluster are linked by vertical lines. As shown in Fig. 3, subgraphs with identical *sketches* are grouped into the same cluster. The figure on the right shows the number of *sketches* for each cluster. For example, $\mathbb{C}_3$ consists of subgraphs that have *sketches* $S_0 = \{s_0^0, s_0^1, s_0^2\}$.

- 根据子图的sketch是否相似来进行聚类
- 在一组相似的计算任务中，我们选出来一个**先验子图**进行重点优化，其他图就划为**后验子图**
- 在先验子图找到最优解（sketch+具体数值）后，后延子图的所有sketch都被指定为最优解的sketch，并开始在这个基础上寻找具体数值。

系统设计

相似的子图组内如何选出率先被优化的**先验**“榜样”。



- 五张子图，因为具有相同的3种sketch而分到一组。五张子图都来自batch_matmul操作，只是张量维度有所不同
- 先验子图 G_p 的选择标准是：**其张量维度与集群中所有其他子图的维度最接近**，同时与不同集群中的子图保持区别。
- 这个选择基于一个假设：拥有与所有其他子图最相似的张量维度的子图，预计将能为后验子图的代码扩散带来最好的效果同时也能提高代码的兼容性。
- 通过计算余弦相似度，构建矩阵计算。

如何优化“榜样”，并将优化经验在组内扩散。

为每个子图选择不同的**数值**

Table 3: The examples of parameter initialization rules for program sampling used in Ansor [47] for both CPU and GPU.

CPU	GPU
InitFillTileSize	InitFillTileSize
InitUnroll	InitUnroll
InitChangeComputeLocation	
InitVectorization	
InitParallel	

- **优化的“工具箱”**
- **InitFillTileSize**：将一个大循环分割成两个或更多个嵌套循环
- …

Original Code	Transformed Code via SP

`for i in 0…1024:`
`  …`
	`for i_outer in 0…16:`
	`  for i_inner in 0…64:`
	`  …`

`extent: 1024,`
`length: 16, 64`

Figure 7: An example code transformation using the split-step (SP) with `extent=1024` and `length=[16,64]`.

- **CPU、GPU的最高速的计算缓存很小**
- 将计算任务划分成小数据块，能高效利用**高速的计算缓存**

如何优化“榜样”并在组内扩散

为每个子图选择不同的**数值**

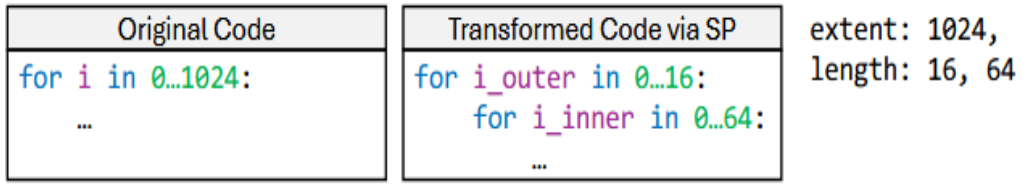


Figure 7: An example code transformation using the split-step (SP) with `extent=1024` and `length=[16,64]`.

三种方法，使得后验子图智能地借鉴“先验” 的成功经验：

1. **“像素级”模仿 (最近因子法)**
   做法：如果新旧任务的循环长度差不多，我们就直接照搬或选择一个**最接近**的拆分方案。

2. **“等比例”缩放 (比例缩放法)**
   做法：如果新任务的循环长度是旧任务的两倍，我们就把旧的拆分方案也**等比例地“放大”两倍**。

3. **“打破常规”探索 (随机选择法)**
   做法：偶尔完全**随机**地尝试一种全新的拆分方案。

系统设计

原始循环的长度

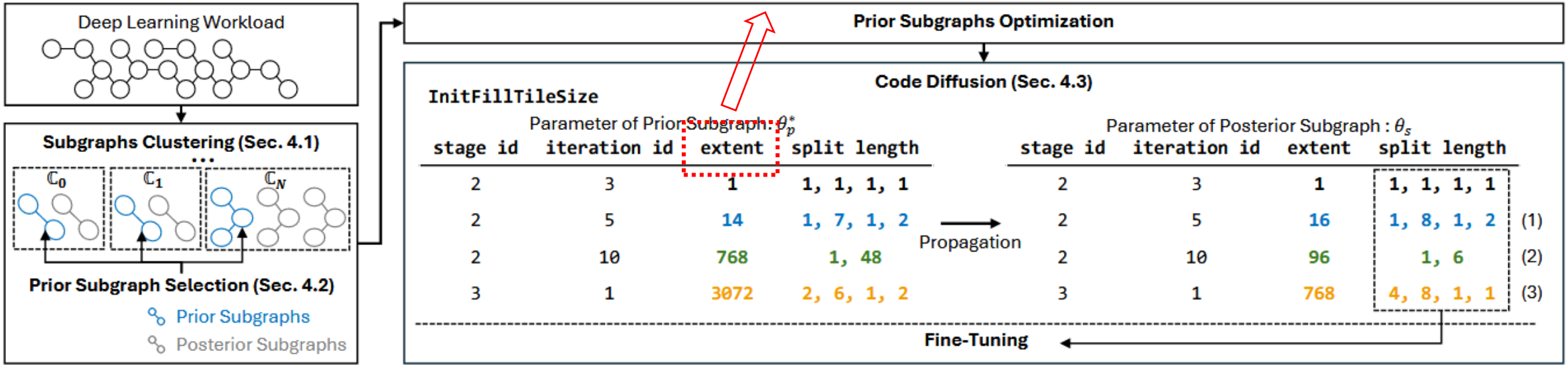


Figure 6: Given a set of subgraphs from a deep learning model, they are first clustered according to their sketches derived by Ansor [47]. Within each subgraph cluster, a prior subgraph is selected and optimized. Subsequently, the remaining subgraphs are optimized through prior parameter propagation, followed by iterative code diffusion using the Bayesian implementation.

提纲

- 背景
- 系统设计
- **实验评估**
- 总结

Evaluation

实验设置

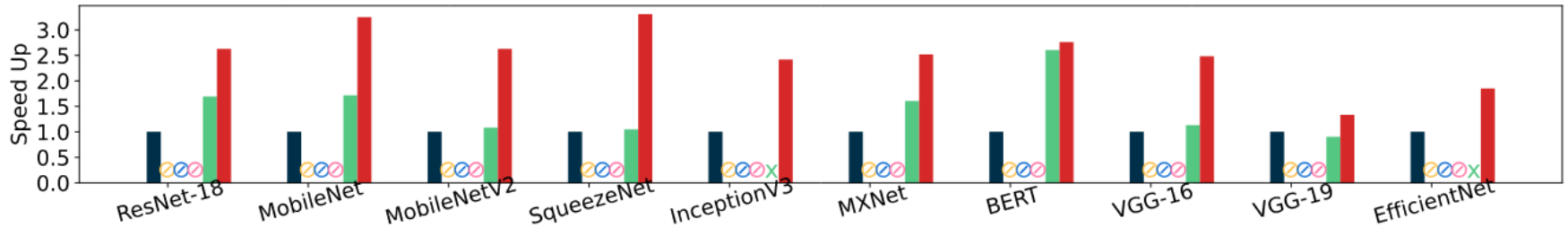
Baseline： Ansor

CPU: Intel Core i9-11900K@3.50GHz
GPU： Nvidia A6000

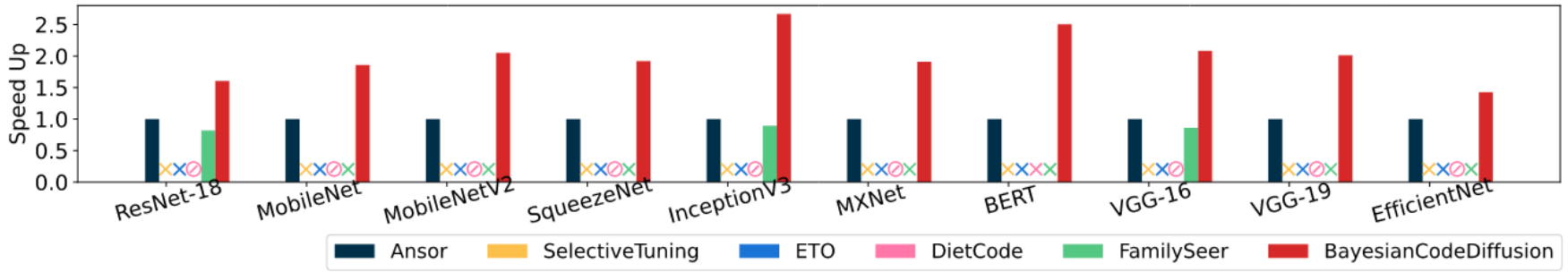
Workload： ResNet-18 VGG-{16,19}， BERT， MobileNet， MobileNet-V2， SqueezeNet-V1.1， Inception-V3， MXNet， EfficientNet

实验评估

模型**编译**端到端优化测试



(a) Compilation (optimization) speedup on CPU (Intel Core i9-11900K)



(b) Compilation (optimization) speedup on GPU (Nvidia A6000)

Figure 13: The compilation (optimization) time speedup in generating deep learning programs with execution latencies equivalent to the best latencies achieved by Ansor [47], normalized to a baseline of 1.0 (the blue bars). The symbol '×' indicates that the corresponding method fails to generate a program with latency equivalent to Ansor's best, while '⊘' denotes that the corresponding method is not applicable to the target deep learning model or hardware (CPU/GPU).

⊘：表示该方法不适用于当前的硬件或模型

×：在规定的时间内，未能找到一个性能能比肩Ansor的程序

实验评估

模型编译后，生成程序的**执行延迟**变化

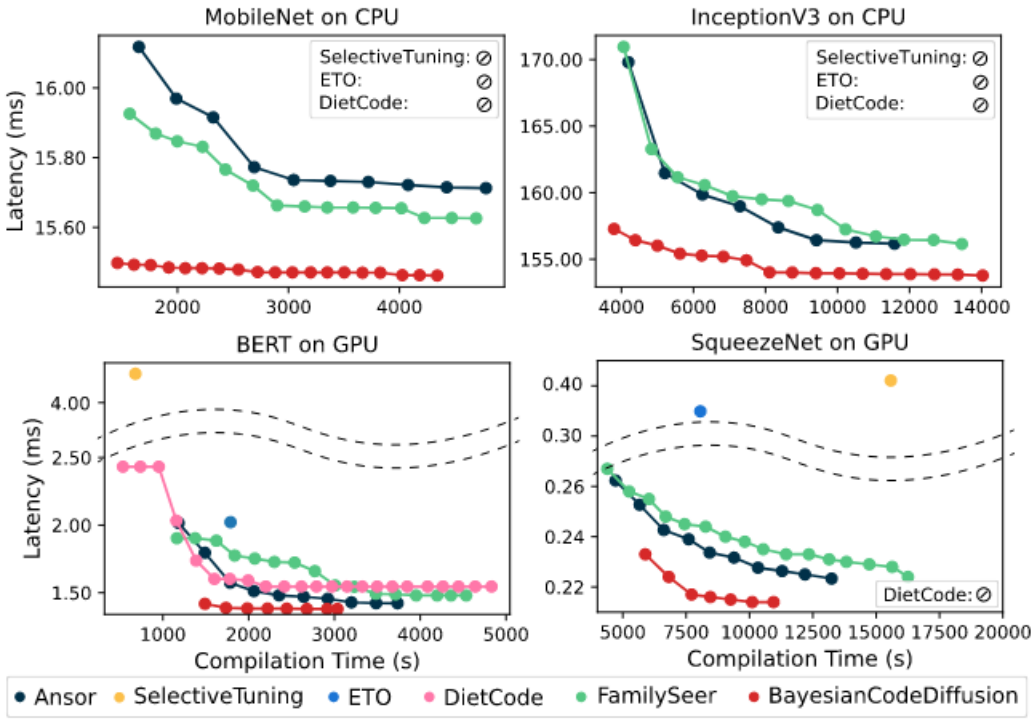


Figure 14: The execution latencies (ms) of the generated programs over compilation (optimization) time (s). The methods that are not applicable either to the target deep learning models or hardware (CPU/GPU) are indicated by the symbol '⊘'.

实验评估

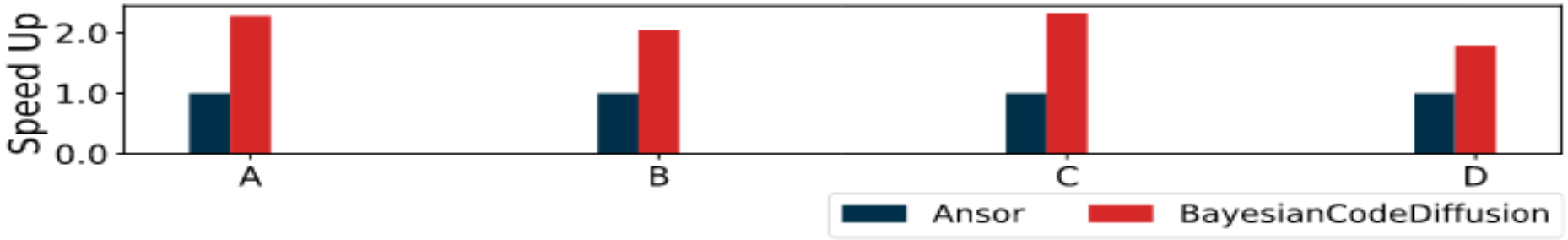


Figure 15: The optimization speedups on subgraph clusters in Tab. 5, compared against Ansor [47].

在一个**特定的“子图集群”**（即一组非常相似的任务）上，本文方法（红色）和传统方法Ansor（黑色）的性能对比。图15是优化速度的对比

实验评估

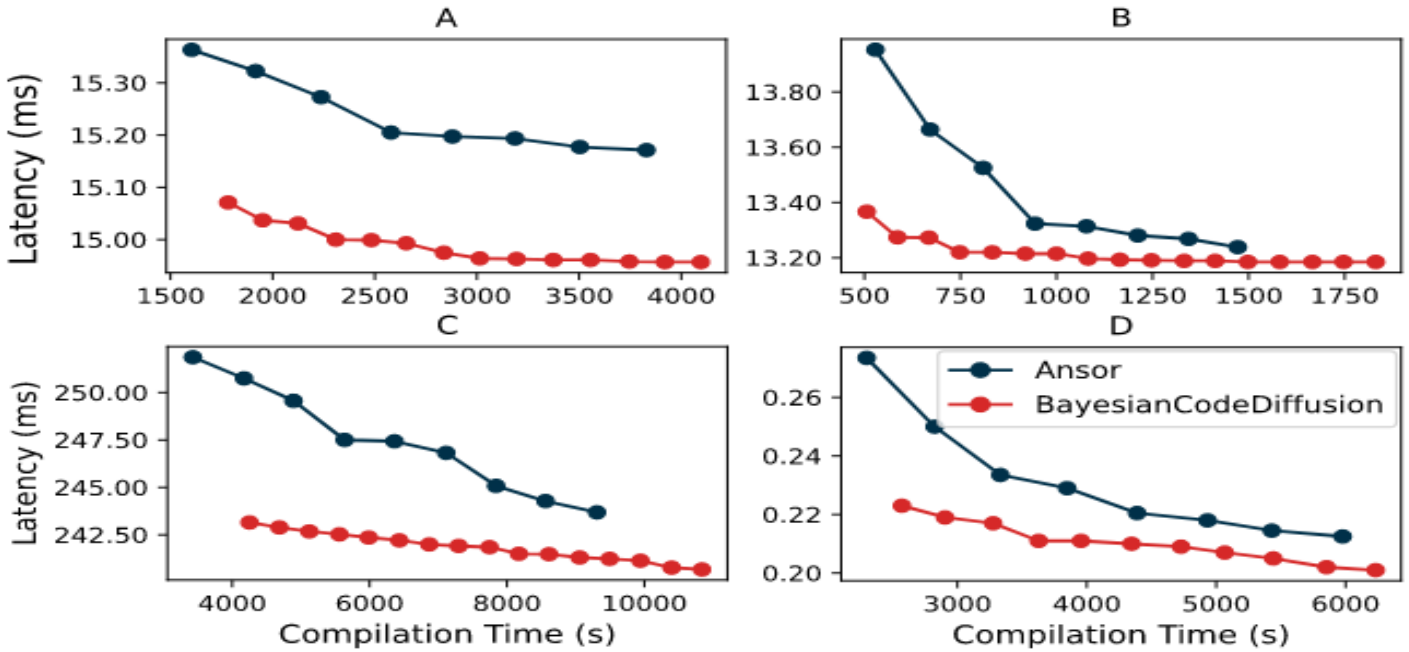


Figure 16: The latencies of the generated programs (ms) over the subgraph cluster compilation (optimization) time (s).

在一个特定的“子图集群”（即一组非常相似的任务）上，优化过程中程序性能（延迟）随时间的变化。

实验评估

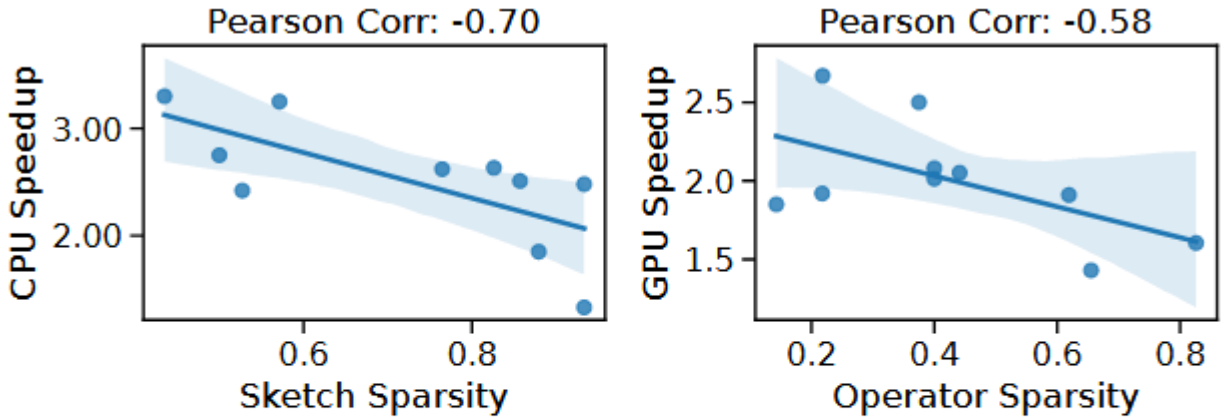


Figure 18: The Pearson correlation [13] between subgraph sparsity and speedup on CPU and GPU. On CPUs, speedup is correlated more strongly with sketch sparsity, whereas on GPUs, it is more closely associated with operator sparsity.

什么因素对贝叶斯扩散的加速效果影响最大

结论：

- 模型的稀疏度越低（即内部越相似、越重复），贝叶斯优化方法获得的加速比就越高。
- CPU上，能不能找到相似的sketch是加速的关键；
- GPU上，能不能找到相似的算子更重要；

实验评估

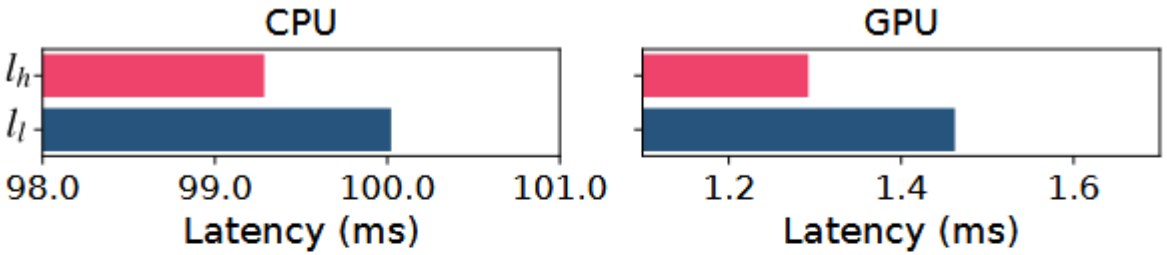


Figure 19: A comparison of l_h and l_l on CPU and GPU, where l_h and l_l denote the program execution latency of the first diffused code optimized using high- and low-similarity *prior* subgraphs, respectively. The bar lengths indicate the mean value of execution latencies of various deep learning models.

验证论文中提出的**选择最相似的成员作为先验**这一策略是否正确

结论：

- 高相似度先验能带来更好的初始性能

提纲

- 背景
- 系统设计
- 实验评估
- **总结**

总结

- **核心思想**：将贝叶斯理论中的“先验知识指导后验探索”思想，创造性地应用于解决相似计算任务间的冗余优化问题。
- **搜索优化**：核心贡献在于优化了搜索过程：通过高质量的“先验”设定了更优的搜索起点，并通过“代码扩散”实现了更高效的搜索方式，摆脱了传统随机搜索的低效。
- **充足的成果与验证**：实验证明，该方法在CPU和GPU上均取得显著的编译加速（最高3.31x）和性能提升。并通过充分的对比实验和消融研究，系统性地验证了框架各模块的有效性。

总结

➢ 这个paper有什么问题，基于这个paper还能做什么？

- **对比实验不充足**：完全没有和Roller(OSDI'21)、Tenset(OSDI'21)、TLP(OSDI'23)、TLM (OSDI'24)做对比，不知道性能是否有差别。

- **“先验”不一定是真正的最优**：论文方法的核心是找到一个高质量的“先验”($\theta p*$)，但当前是通过分配大量搜索时间来**经验性地**估计这个解，**无法保证**它就是理论上的真正最优解 。一个不够好的“先验”可能会影响后续所有“后验”的优化效果。

- **与TLM结合**：由TLM**辅助**，直接为一组相似的子图**生成一个质量极高的“初始优化方案”**（即高质量的先验Sketch和参数），然后再利用本文提出的在线“代码扩散”方法，对LLM生成的方案进行针对性的、面向特定硬件的精细微调。

总结

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

- 的确可以考虑将子图分类优化这一方法应用在TLM的拓展应用上，然后或许可以考虑加入硬件记忆（sketch记忆），在遇到相似的子图或者sketch时，让TLM直接从记忆库中找到类似记录，然后再进行微调优化。

➢ 这个paper能不能泛化？

- 扩散的方法，或许可以用于确定代码生成的边界问题。针对LLM的代码生成任务，我们可以让LLM给出类似于张量程序的一个架构，而具体的每个张量操作的边界问题则交给贝叶斯泛化。

Bayesian Code Diffusion for Efficient Automatic Deep Learning Program Optimization

OSDI'25

Thank you !

Reporter: Hangshuai He