2025 华为智联杯· 无线程序设计大赛

AI 硬件功耗和最短推理时延—任务书

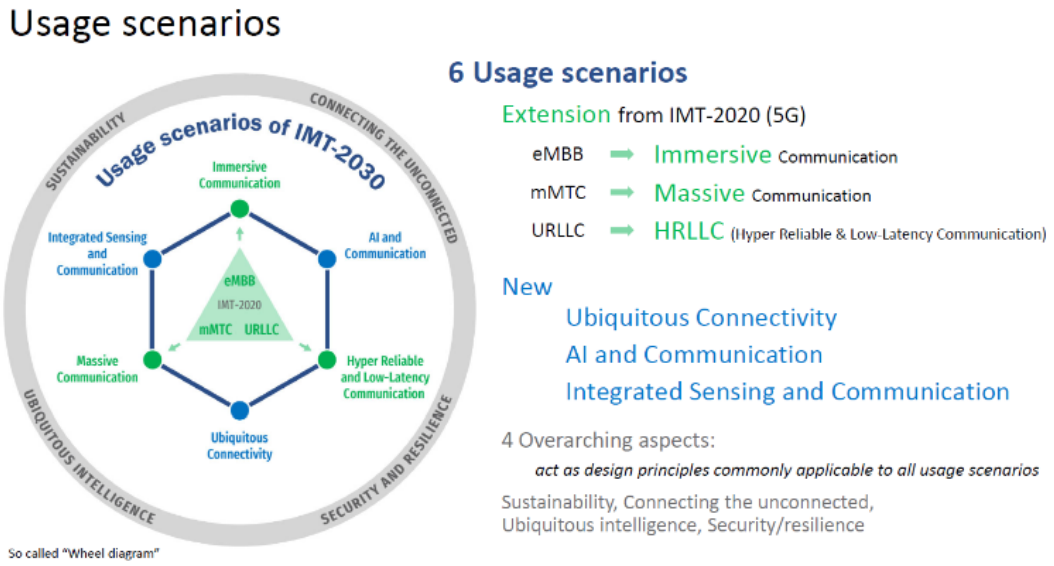
HUAWEI

目 录

1. 背景信息 1
2. 题目 2
2.1 术语 2
2.2 AI硬件平台功耗模型 2
2.2.1 问题抽象 2
2.2.2 赛题任务 3
2.3 AI最短推理时延 3
2.3.1 问题抽象 3
2.3.2 赛题任务 5
3. 输入说明 7
3.1 AI硬件平台功耗模型输入 7
3.1.1 训练集输入 7
3.1.2 测试集输入 7
3.2 AI最短推理时延输入 8
4. 输出说明 11
4.1 AI硬件平台功耗模型输出 11
4.2 AI最短推理时延输出 11
5. 评分规则 12
6. 附录 13
6.1 最短推理时延用例一 13
6.2 最短推理时延用例二 13

1.背景信息

近年来，AI技术发展迅速，AI与通信融合得到了业界的广泛关注和持续投入。例如AI信道估计、AI收发机、AI波束管理等， AI与无线通信融合有助于运营商实现网络自动化，提高无线通信网络谱效、能效、体验、性能等，在IMT-2030对无线通信发展的下一跳关键技术规划中已占据重要位置（AI and Communication）。



在无线通信领域，涉及AI相关的硬件平台的功耗和推理时延都十分的关键。根据指定的特征数据进行功耗的评估，针对给定AI推理任务在对应硬件平台的执行给出一个最短的推理时延任务调度策略，这些都是非常有挑战和价值的难题。

期待您的精彩解决方案！

2.题目

2.1 术语

名词	解释
RMSE	均方根偏差（root-mean-square error，RMSE），用于衡量模型预测值或估计量（样本值或总体值）与观测值之间差异的一种指标。
计算图	计算图就是一个表示运算的有向无环图(Directed Acyclic Graph，DAG)，由数据张量和算子组成；
算子	算子(Operator) 是计算图中的节点，接收输入张量并输出新张量；
子图	子图是算子内的计算图片段，在本题中子图内的子图节点是最小的任务执行单元；
Shape	算子输入张量的维度信息，可以简单理解为多维数组维度；
Tiling	通过将大的张量分割成更小的块以便更有效的利用硬件资源的优化技术；不同的shape会有不同的tiling策略，不同Tiling策略会有不同的子图；
核	硬件平台的核包括多种类型的核，如NPU、GPU、CPU等；
内存	硬件平台的内存包括多种类型的内存，如内存、显存、缓存等；

2.2 AI 硬件平台功耗模型

2.2.1 问题抽象

对于AI硬件平台，可以用特定的采样数据来进行其相关功耗的评估。

- **线下训练任务：**在本赛题中，将提供2778条AI硬件平台的任务指标向量$\boldsymbol{U}_{\mathbf{train}}$和对应功耗$\boldsymbol{P}_{\mathbf{train}}$(带标签，其中每条数据提供三次采集结果，三次结果中的特征和标签可以任意对应) 。你需要使用这2778条数据作为训练集训练自己的 AI 模型 f 表示如下:

$$P = f(U)$$

  请利用好三次采样来排除噪点，并使用合适的模型尽可能提高模型预测的准确度。
- **线上推理任务：**将给出若干条 AI 硬件平台在特定场景下的任务指标 U_{test}(无标签)，你需要利用你的 AI 模型推理得到这些任务指标 U_{test} 对应的功耗功率 $\widehat{\boldsymbol{P}}$，并在规定时间内完成相关文件上传。模型准确度的判断标准使用验证集的 RMSE 进行：

$$\text{RMSE} = \sqrt{(\frac{1}{\text{N}}\sum_{i=1}^{N}(P_i - \hat{P})^2)}$$

其中 $\hat{P}_i$ 和 P_i 分别为测试集中样本 i 的预测值和实际值。

2.2.2 赛题任务

请根据给定的训练集中带标签的数据，构建 AI 模型。你需要考虑：数据分析、预处理、模型搭建和调优、在验证集的泛化能力等。本小题在赛事平台提交时，需要同时上传：预测结果+训练/推理源代码。训练如有使用随机数，请在代码注释中标注出随机数或种子的取值以便进行代码复现与审查。

2.3 AI 最短推理时延

2.3.1 问题抽象

(1) 硬件平台信息

给定AI推理平台信息如下：

- 执行单元由多种异构核组成，表示如下：

$$C = \{(c_1, n_1), ..., (c_p, n_p)\}$$，c表示类型，n表示数目

- 存储单元由层次化内存架构，表示如下：

$$M = \{(m_1, s_1), ..., (m_q, s_q)\}$$，m表示类型，s表示大小

(2) 计算图信息

计算图是由一系列算子构成的一个或多个DAG图，详细如下：

- 算子ID共有N个，表示如下：

$$O = \{o1, o2, ..., o_N\}$$

- 算子之间有依赖，依赖可用O的笛卡尔积子集表示，如下：

$$E_O \subseteq O \times O, \quad （O_i, O_j)表示O_i的后继是O_j$$

- 算子ID会对应唯一的算子类型Type；
- 算子输入shape，这里都是用1维，会给出这1维的大小；
- (算子类型Type + 算子输入shape) 会确定1个或多个tiling策略；
- (算子类型Type + 算子输入shape + tiling策略索引) 会确定算子的子图信息；

针对这一系列的子图信息的说明，详细如下：

- 子图由多个子图节点ID组成，子图节点ID在算子内唯一，算子ID和子图节点ID全局唯一标识一个执行节点。算子O_i所包含的子图节点ID集合表示如下：

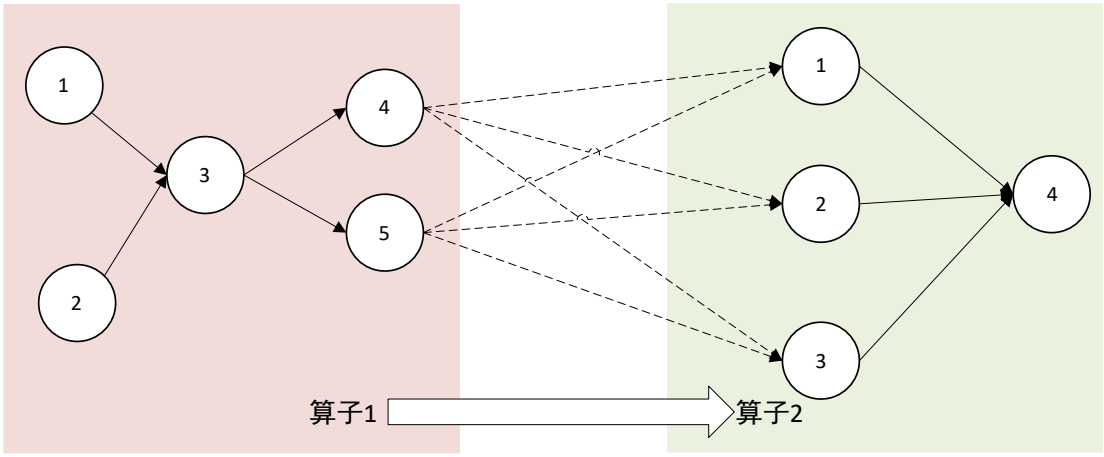
$$V_{oi} = \{v_{i1}, v_{i2,}...\}$$

所有算子的子图节点ID集合就是全部的子图节点集合V；

- 子图节点ID也有前后执行依赖，表示如下：

$$E_V \subseteq V \times V \qquad (v_p, v_q) \text{表示} v_q \text{依赖} v_p$$

注意：一个算子内所有的子图节点ID执行完毕后，表示该算子执行完毕，才能执行其后续算子；并且算子内最后执行的所有子图节点的后续子图节点就是该算子的后续算子内的所有最先执行的子图节点。如下图：



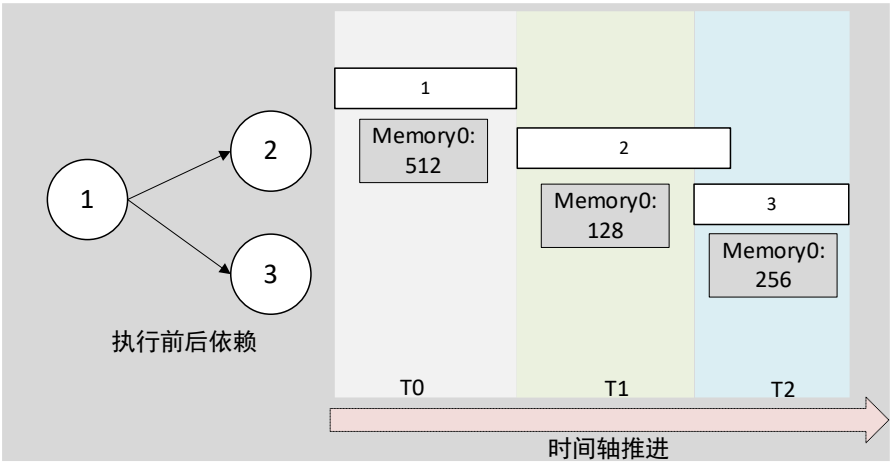
节点依赖举例说明：如上图所示算子1后继节点是算子2，那么算子1的所有子图节点执行完毕才可以执行算子2；同时有个隐含的依赖是算子1中最后执行的子图节点4和5，它们的后继是算子2的最开始的子图节点1、2和3，如上图中的虚线所示。这个隐含的前后关系会影响到后面的同类型内存释放约束，参见后续内存释放依赖约束部分。

- 子图节点执行的时间和资源占用表示如下：
  - 执行时间；
  - 核的类型，子图节点执行只会占用一个核；
  - 内存的类型和大小，子图节点执行会占用一种或多种类型内存；

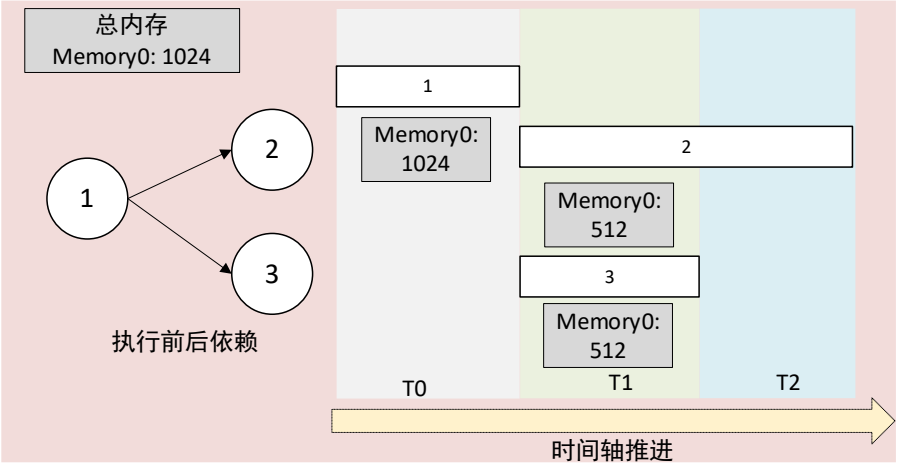
(3) 相关约束

给定的约束如下：

- 同一时刻，同1个核上只能执行1个子图节点；
- 子图节点作为最小执行单元，一旦开始执行，就必须执行完毕；
- 同一时刻，所有正在运行的子图节点，不能超过系统的核和内存总资源；
- 算子ID、子图节点ID的执行有前后依赖，如上面的有序对集合E_o,E_V所示；
- 对于核的使用：节点执行完毕，则立即释放使用的核资源；
- 对于内存的使用：节点执行完毕，内存能否立即释放取决于后续节点，该节点和后续节点之间（包括算子内和算子间）如果存在相同类型的内存占用，需要等后续使用相同类型内存的节点均开始执行时才能释放对应类型内存。一个节点释放同一类型的内存时要么不释放，要么全部释放。内存释放约束举例说明：



1) 如上图所示，时间推进先后顺序为T0, T1, T2，节点1后继节点是2,3, 都使用了Memory0类型的内存(不同类型的内存使用没有影响)，所以只有到T2时刻，2和3都处于开始执行或者执行完毕状态，1才可释放自己执行使用的Memory0的512大小的内存。



2) 在内存能否释放的场景判断中，同一个时刻，系统会先判断当前时刻启动的节点，然后再去判断这个时刻结束的节点。上图中，Memory0类型总内存1024，节点1使用1024，节点2和3都使用512，在1结束的时刻，2和3同时开始，那么1可以在结束时刻合法释放，2和3可以正常启动，正好使用了系统内存1024，这个调度就合法。

2.3.2 赛题任务

在满足上面约束条件的情况下，针对给定的AI硬件平台信息和算子信息，给出最优的一个计算图调度执行策略，尽可能缩短最终的推理时延。详细如下：

(1) 题目输入

- 硬件平台信息：核类型和数目，内存种类和大小；规格：
  - 核类型：<= 10, 每种类型的核数目 $< 10^3$；
  - 内存种类：<= 10, 每种类型的内存大小 $<10^{16}$；
- 算子信息： (算子类型，算子输入shape)会确定1个或者多个tiling策略，(算子

类型，算子shape，tiling策略ID) 会确定唯一子图信息。算子内的子图信息：子图节点ID，子图节点执行的核类型、运行时间、内存类型和大小；规格：

- 算子类型个数：< 10^2
- 同一算子类型对于相同shape的输入其tiling个数：<10^2
- 单个算子内的子图节点个数：<10^4

注意：选择tiling的时候可以先判断是否可行，因为有的tiling策略可能没有可行的调度结果。

- 计算图信息：包括算子ID，算子ID的前后依赖关系，算子ID对应的算子类型，算子输入shape。单图和多图输入形式没有变化，只是DAG之间是否有依赖关系的区别。规格：
  - 全局算子节点数：单图节点数<10^5，图个数<10^3
  - 全部节点个数：<10^7

详细输入格式见后续输入说明章节。

(2) 输出结果

输出对应的调度策略，包括如下信息：

算子ID和对应选择的tiling策略索引、子图节点ID和分配的起始时间，分配的核ID。上面的调度策略就是这个计算图的最终执行调度策略。这个调度策略中最晚执行结束的子图节点的结束时间就是最终的推理时延。

详细输出格式见后续输出说明章节。

(3) 赛题提交

本小题在赛事平台提交时，需要：按照格式上传对应源代码。

3.输入说明

3.1 AI 硬件平台功耗模型输入

3.1.1 训练集输入

训练集共给出2778条有标签的采样数据。

- 命名规范：训练集给定数据命名规则为 data_train.jsonl
- 文件格式：jsonl 格式的纯文本文件，每行为一条数据
- 数据格式：float32

对于每一行数据：

```
{
   "Features":
      [
         [67000, 25526, 89656, 155756, 555152, …],
         [67025, 25530, 89629, 155795, 556941, …],
         [67069, 25589, 89654, 155718, 555465, …]
      ],
   "Labels":
      [
         200.56,
         201.59,
         203.69
      ]
}
```

其中"Features"字段为一个 3*1209 的矩阵，表示AI硬件3次采样下的1209维特征值 $\boldsymbol{U}_{\mathbf{train}}$，而 "Labels" 字段为3个标量，表示3次采样下的标签值$\boldsymbol{P}_{\mathbf{train}}$。同一样本行数据的"Features"与"Labels"字段的值对应，但其中3次采样的特征和标签不必按序一一对应。这是由于对特征和标签的采样是分时进行的，采样3次仅为了确保采样结果具有代表性，同一行数据3次采样的特征和标签可以任意对应。选手应利用好3次采样排除噪点，提高模型准确度。

3.1.2 测试集输入

对于测试集，线上比赛提供313条无标签的采样数据，线下比赛提供新的442条测试样本。

- 命名规范：测试集数据命名data_test.jsonl
- 文件格式：jsonl格式的纯文本文件，每行表示一条数据的特征
- 数据格式：float32

对于每一行数据：

```
{"Feature": [67000, 25526, 89656, 155756, 555152, …]}
```

其中将只提供"Feature"字段，"Feature"字段为一个 1*1209 的矩阵，表示AI硬件1次采样下的1209维特征值$\boldsymbol{U}_{\mathbf{test}}$。

3.2 AI 最短推理时延输入

线上评测时，输入数据从标准输入读入。每个用例输入有N行，格式如下：

- 第1行：表示添加的AI计算平台信息；
- 第2到N-1行：表示添加的算子、tiling、子图等信息；
- 第N行：计算图信息，这个给出后输入完毕，后续请计算输出结果；

数据格式：数据格式可以都用uint64格式，内存大小和执行时间这两个字段较大，其余不会超过u32的最大值。

举例说明：

下面是一个用例的完整输入，共6行：

```
SetSocInfo([[0,2],[1,4],[2,4]],[[0,10240],[1,10240],[2,40960]])
AddOpInfo(1,1024,0,[[1,2],[2,3]],[[1,0,5],[2,1,10],[3,2,5]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024]])
AddOpInfo(1,1024,1,[[1,2],[2,3],[1,4],[4,3]],[[1,0,5],[2,1,5],[3,2,5],[4,2,5]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024],[4,2,1024]])
AddOpInfo(1,256,0,[[1,2],[2,3]],[[1,0,4],[2,1,2],[3,2,3]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024]])
AddOpInfo(2,512,0,[[1,2],[2,3],[1,4],[4,5]],[[1,0,4],[2,1,6],[3,2,4],[4,2,6],[5,1,4]],[[1,0,1024],[1,2,1024],[2,0,1024],[2,1,1024],[3,1,1024],[4,0,1024],[4,1,1024],[5,1,1024]])
GetInferenceScheResult([[1,2],[2,3]],[[1,1,1024],[2,2,512],[3,1,256]])
```

针对例子的详细说明如下：

第1行的说明如下：

SetSocInfo([[0,2],[1,4],[2,4]],[[0,10240],[1,10240],[2,40960]])

上面的含义类似下面的伪代码：

```
SetSocInfo(
  vector<array<u32, 2>>& coreInfo, // [[CoreType, CoreNum],...]
  vector<array<u64, 2>>& memInfo   // [[MemType, MemSize],...]  MemSize需要u64
)
```

(1) 第一个参数表示核类型和该类型的核个数，示例中核类型0的核有4个，核类型1的核有4个，类型2的核有4个；同类型的核的ID从0开始递增编号，例如0类型的核有4个，则核ID的编码为0~3，依此类推，核ID取值为[0, N-1]，N为该类型核个数；

(2) 第二个参数表示层次化内存类型以及大小信息，示例中内存类型0的大小为10240，内存类型1大小为10240，内存类型2的大小为40960；

第2行的说明如下：

AddOpInfo(1,1024,0,[[1,2],[2,3]],[[1,0,5],[2,1,10],[3,2,5]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024]])

含义类似下面的伪代码：

```
AddOpInfo(
  u32 OpType,
  u32 shape,
  u32 tiling,
  vector<array<u32, 2>>& opSubGraph,  // [[NodeID(prev), NodeID(next)],...]
  vector<array<u64, 3>>& opRunDetail, // [[NodeID, CoreType, ExecTime],...]
  vector<array<u64, 3>>& opMemDetail  // [[NodeID, MemType, MemSize],...]
) // ExecTime和MemSize需要u64，其余数据都不会超过u32最大值
```

(1) 第一个参数表示算子类型OpType (每个算子ID会对应一个算子类型OpType。算子类型OpType与算子输入shape一起，会匹配到一到多个tiling策略，选手需要在所有匹配的策略中搜索最优的tiling策略)；

(2) 第二个参数表示算子输入shape；

(3) 第三个参数表示tiling策略索引；

(4) 第四个参数表示算子内的子图信息，包括子图节点ID和执行顺序；(子图节点ID在算子内唯一，算子ID和子图节点ID唯一标识一个执行节点，这行的示例表示子图节点ID 1执行完了执行子图节点ID 2，子图节点ID 2执行完了执行子图节点ID 3)；

(5) 第五个参数表示子图节点运行的需要的核类型和时间，即子图节点ID 1运行在核类型为0的核上，执行时间为5，子图节点ID 2运行在核类型为1的核上，执行时间为10，子图节点ID 3运行在核类型为2的核上，执行时间为5(这里的核类型就是SetSocInfo函数中给出的核类型)；

(6) 第六个参数表示子图内每个节点占用的内存信息，即子图节点ID 1占用内存类型0的大小为1024，子图节点ID 1占用内存类型2的大小为1024；子图节点ID 2占用了内存类型0大小为1024，依此类推；

本行的算子数据就如下图所示：

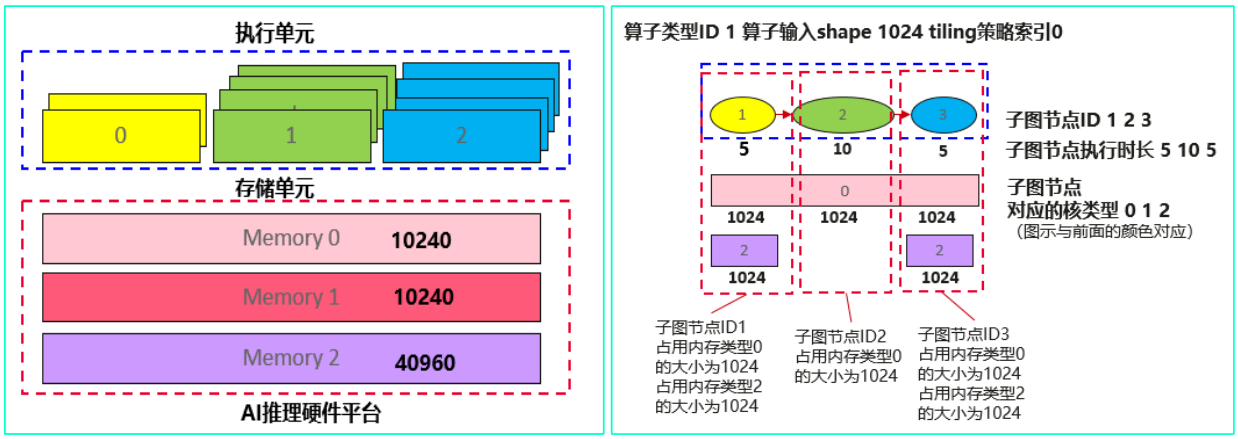


图 AddOpInfo信息解释

上图中左侧表示了硬件平台，颜色表示的核和内存的类型，右边是对应算子的子图相关信息，包含了算子的执行顺序依赖，需要的核和内存资源以及对应的执行时长等。

第3、4、5行和第2行类似，这里不再一一展开描述。

第6行的说明如下：

GetInferenceScheResult([[1,2],[2,3]],[[1,1,1024],[2,2,512],[3,1,256]])

含义类似下面的伪代码：

```
GetInferenceScheResult(
  vector<array<u32, 2>>& opGraph, // [[OpID(prev), OpID(next)],...]
  vector<array<u32, 3>>& opNode   // [[OpID, OpType, shape],...]
)
```

(1) 第一个参数表示用于待执行的计算图信息，用算子ID和其执行关系来表示。即算子ID 1的算子执行完了才能执行算子ID为2的算子，算子ID为2的算子执行完了才能执行算子ID为3的算子。

(2) 第二个参数表示计算图中每个算子ID所对应的算子类型OpType和算子输入shape。

以上就是本题用例的输入说明。

4.输出说明

4.1 AI 硬件平台功耗模型输出

请将测试集数据的测试的输出结果输出到results.jsonl文件中，输出样本的顺序与data_test.jsonl数据顺序一一对应。

对于每一行数据：

```
{
  "PredictResult": 102.89
}
```

其中 "PredictResult" 字段为一个标量，数据格式为 float32，表示你需要提供的推理结果 $\hat{P}$。

除了预测结果results.jsonl以外，你需要同时上传训练/推理源代码。训练过程中如有使用随机数，请在代码注释中标注出随机数或种子的取值以便进行代码复现与审查。

4.2 AI 最短推理时延输出

输出结果为1行，包含了所有子图节点的调度信息，用数组表示。每个数组元素也是一个数组，表示单个子图节点的详细调度信息，整体是个二维数组，格式如下：

[[算子ID，tiling策略，子图节点ID，执行起始时间，分配核ID],…]

注：调度结果中每个子图节点的输出顺序不作要求。

举例说明：

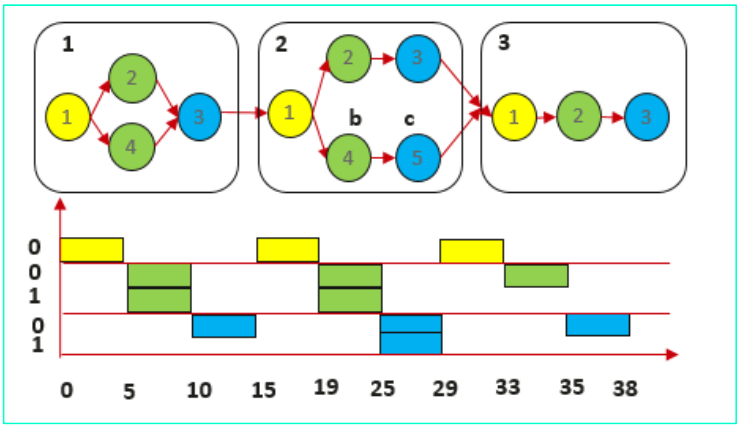
下面是一个完整的输入举例，对应的输入就是前面输入章节的输入：

```
[[1,1,1,0,0],[1,1,2,5,0],[1,1,3,10,0],[1,1,4,5,1],[2,0,1,15,0],[2,0,2,19,0],[2,0,3,25,0],[2,0,4,19,1],[2,0,5,25,1],[3,0,1,29,0],[3,0,2,33,0],[3,0,3,35,0]]
```

详细解释如下：

以首个数组元素 [1,1,1,0,0]为例，表示：
算子ID 1，选择tiling策略1，算子内子图节点ID为1的节点，执行开始时间为0，分配核ID为

调度结果含义如下图，颜色对应核和内存的类型，横轴表示时间，纵轴表示同类型的核ID。



以上就是本题的输出说明，调度时延就是38

5.评分规则

选手的排名取决于两道题目的**总得分**。如果不同选手的总分相同，则先提交代码者获胜。每道题的评分规则如下：

(1) AI硬件平台功耗

本题作为AI题目，判题程序会根据选手提交的模型预测结果按照*RMSE*计算预测误差，然后根据理论上下界计算该题得分。本题的分数上界：20000. 得分的计算公式为：

$$Score = \max(100,\quad 20000 \times (1 - \frac{RMSE}{RMSE_{\max}}))$$

预测结果数据格式合法就会有基础分100，*RMSE*越小越接近分数上界，$RMSE_{max}$是50，超过这个最大误差阈值，就只能获得基础分。

(2) 最短推理时延调度

本题作为传统算法类题目，按照如下规格进行判分：

1. 判题程序会从选手程序标准输出读取分配方案，计算对应用例的调度结果的最终时延。
2. 判题采用多组用例数据，用例规模大小和复杂度不同，复杂度和规模越高的用例分值越高。每个用例的调度时延有其理论的最大时间T_{max}和最小时间T_{min}。如果选手提交的程序计算出的该用例的调度结果合法，那么选手的调度时间T一定满足：

   $$T_{\min} \le T \le T_{\max}$$

   用例的基础得分S_{base}，上界得分是S_{max}，这个用例得分计算规则如下：

   $$Score = S_{\text{base}} + (S_{\max} - S_{base}) \times \frac{T_{max} - T}{T_{max} - T_{min}}$$

   注意：选手程序调度结果非法，则该用例得分为0分。

   所有用例的得分之和就是这个题目的得分，本题目所有用例的总得分上界为80000分。
3. 对于单个用例，选手的程序所有计算步骤（包含读取输入、计算、输出方案）所用时间总和不超过60秒，内存不超过1GB。若程序运行超时、执行出错或输出不合法的解（包括调度分配方案不满足题目约束或解的格式不正确），则判定该用例无成绩，不会影响其他用例。
4. 禁止在代码中执行 shell 命令、使用多线程等影响判题机器运行与公平性的行为。此行为在赛后的最终测评阶段也将无法得分！
5. 比赛结束后将进行代码查验，如发现代码重复或违规等情况，将取消该团队参赛资格和现有成绩。
6. 比赛允许多次提交，评分后台会选取最高的得分的那次提交作为最终提交。

⚠ 注意

对于传统算法题目，因为进程调用存在一定的时间开销，用时统计在判题程序侧和选手程序侧可能存在细微差异。建议选手控制算法用时的时候要留有一定的冗余。

6.附录

6.1 最短推理时延用例一

该用例就是前面输入输出说明章节的用例。

输入：

```
SetSocInfo([[0,2],[1,4],[2,4]],[[0,10240],[1,10240],[2,40960]])
AddOpInfo(1,1024,0,[[1,2],[2,3]],[[1,0,5],[2,1,10],[3,2,5]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024]])
AddOpInfo(1,1024,1,[[1,2],[2,3],[1,4],[4,3]],[[1,0,5],[2,1,5],[3,2,5],[4,2,5]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024],[4,2,1024]])
AddOpInfo(1,256,0,[[1,2],[2,3]],[[1,0,4],[2,1,2],[3,2,3]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024]])
AddOpInfo(2,512,0,[[1,2],[2,3],[1,4],[4,5]],[[1,0,4],[2,1,6],[3,2,4],[4,2,6],[5,1,4]],[[1,0,1024],[1,2,1024],[2,0,1024],[2,1,1024],[3,1,1024],[4,0,1024],[4,1,1024],[5,1,1024]])
GetInferenceScheResult([[1,2],[2,3]],[[1,1,1024],[2,2,512],[3,1,256]])
```

输出：

```
[[1,1,1,0,0],[1,1,2,5,0],[1,1,3,10,0],[1,1,4,5,1],[2,0,1,15,0],[2,0,2,19,0],[2,0,3,25,0],[2,0,4,19,1],[2,0,5,25,1],[3,0,1,29,0],[3,0,2,33,0],[3,0,3,35,0]]
```

上面是一个合法调度结果，调度时延是38

6.2 最短推理时延用例二

输入：

```
SetSocInfo([[0,2],[1,4],[2,4]],[[0,10240],[1,10240],[2,40960]])
AddOpInfo(1,1024,0,[[1,2],[2,3]],[[1,0,5],[2,1,10],[3,2,5]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024]])
AddOpInfo(1,1024,1,[[1,2],[2,3],[1,4],[4,3]],[[1,0,5],[2,1,5],[3,2,5],[4,2,5]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024],[4,2,1024]])
AddOpInfo(1,256,0,[[1,2],[2,3]],[[1,0,4],[2,1,2],[3,2,3]],[[1,0,1024],[1,2,1024],[2,0,1024],[3,0,1024],[3,2,1024]])
AddOpInfo(2,512,0,[[1,2],[2,3],[1,4],[4,5]],[[1,0,4],[2,1,6],[3,2,4],[4,2,6],[5,1,4]],[[1,0,1024],[1,2,1024],[2,0,1024],[2,1,1024],[3,1,1024],[4,0,1024],[4,1,1024],[5,1,1024]])
GetInferenceScheResult([[1,2],[2,3]],[[1,1,1024],[2,2,512],[3,1,256]])
```

输出：

```
[[1,0,1,0,0],[1,0,2,5,0],[1,0,3,13,0],[2,0,1,0,1],[2,0,2,5,1],[2,0,3,13,1],[3,0,1,18,1],[3,0,2,22,0],[3,0,4,22,1],[3,0,3,28,0],[3,0,5,28,1],[4,0,1,32,1],[4,0,2,36,0],[4,0,3,38,0]]
```

上面是一个合法调度结果，调度时延是为41