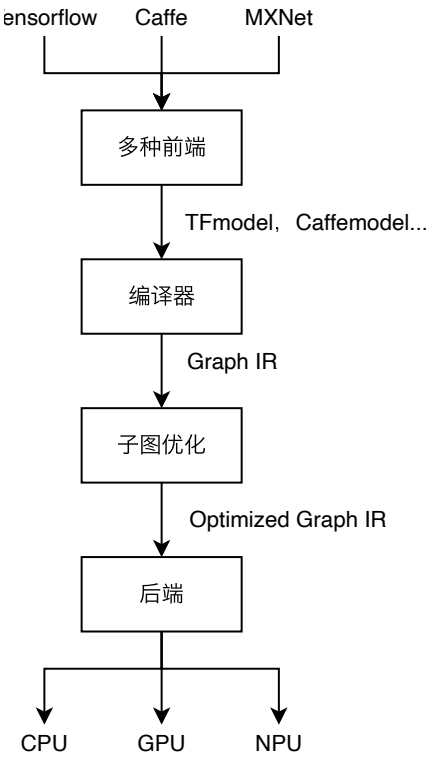
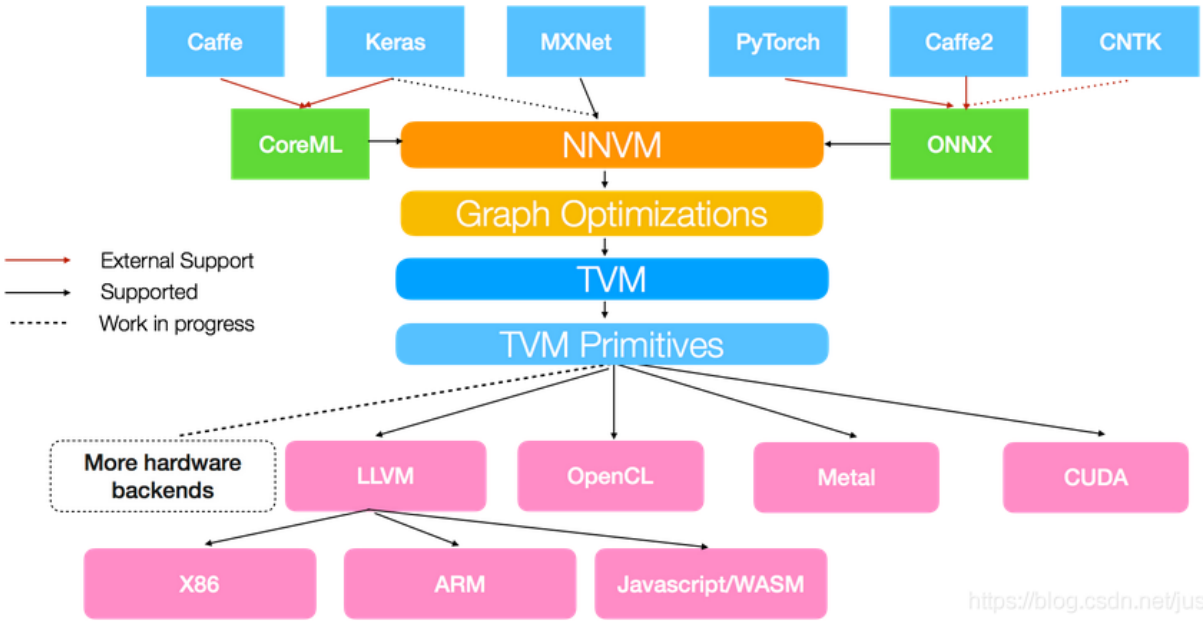
一、深度学习编译器需要完成的任务



深度学习编译器将各个训练框架训练出来的模型作为输入，然后将这些模型传入深度学习编译器之后吐出IR，由于深度学习的IR其实就是计算图，所以可以直接叫作Graph IR。针对这些Graph IR可以做一些计算图优化再吐出IR，最后在后端根据不同硬件的底层和数据分布格式生成可以运行的代码分发给各种硬件使用。

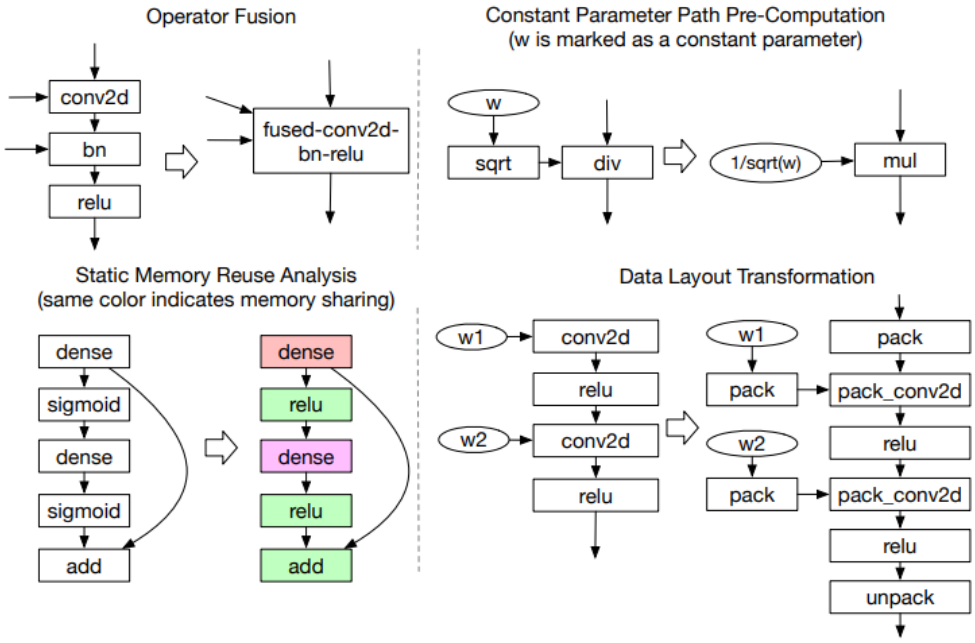
二、autoTVM和autoSchedule的整体架构和区别

TVM架构图



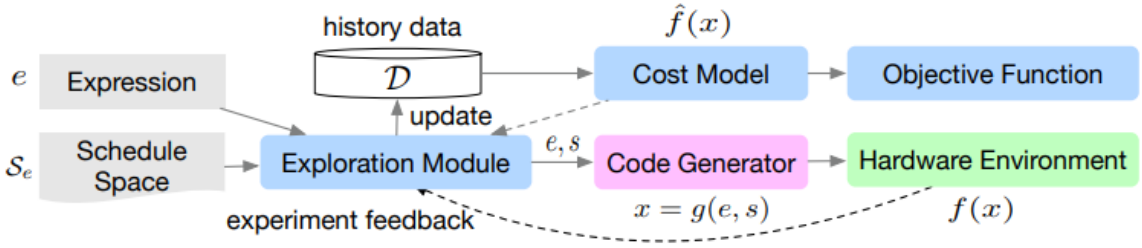
1.TVM架构的核心部分就是NNVM编译器（现在的TVM已经将NNVM升级为了Realy）。NNVM编译器支持直接接收深度学习框架的模型，如TensorFlow/Pytorch/Caffe/MxNet等，同时也支持一些模型的中间格式如ONNX、CoreML。这些模型被NNVM直接编译成Graph IR，然后这些Graph IR被再次优化，吐出优化后的Graph IR，最后对于不同的后端这些Graph IR都会被编译为特定后端可以识别的机器码完成模型推理。比如对于CPU，NNVM就吐出LLVM可以识别的IR，再通过LLVM编译器编译为机器码到CPU上执行。

2.TVM通过计算图优化，融合op内存访问的开销更低；数据布局转换，为每个算子提供定制的数据布局，即调整数据内存布局，在访问时速度更快，尤其是神经网络进行卷积等操作；



3.在代码生成过程中，autoTVM提供**算子人工优化模板**，通过机器学习的方式学习模板下对应

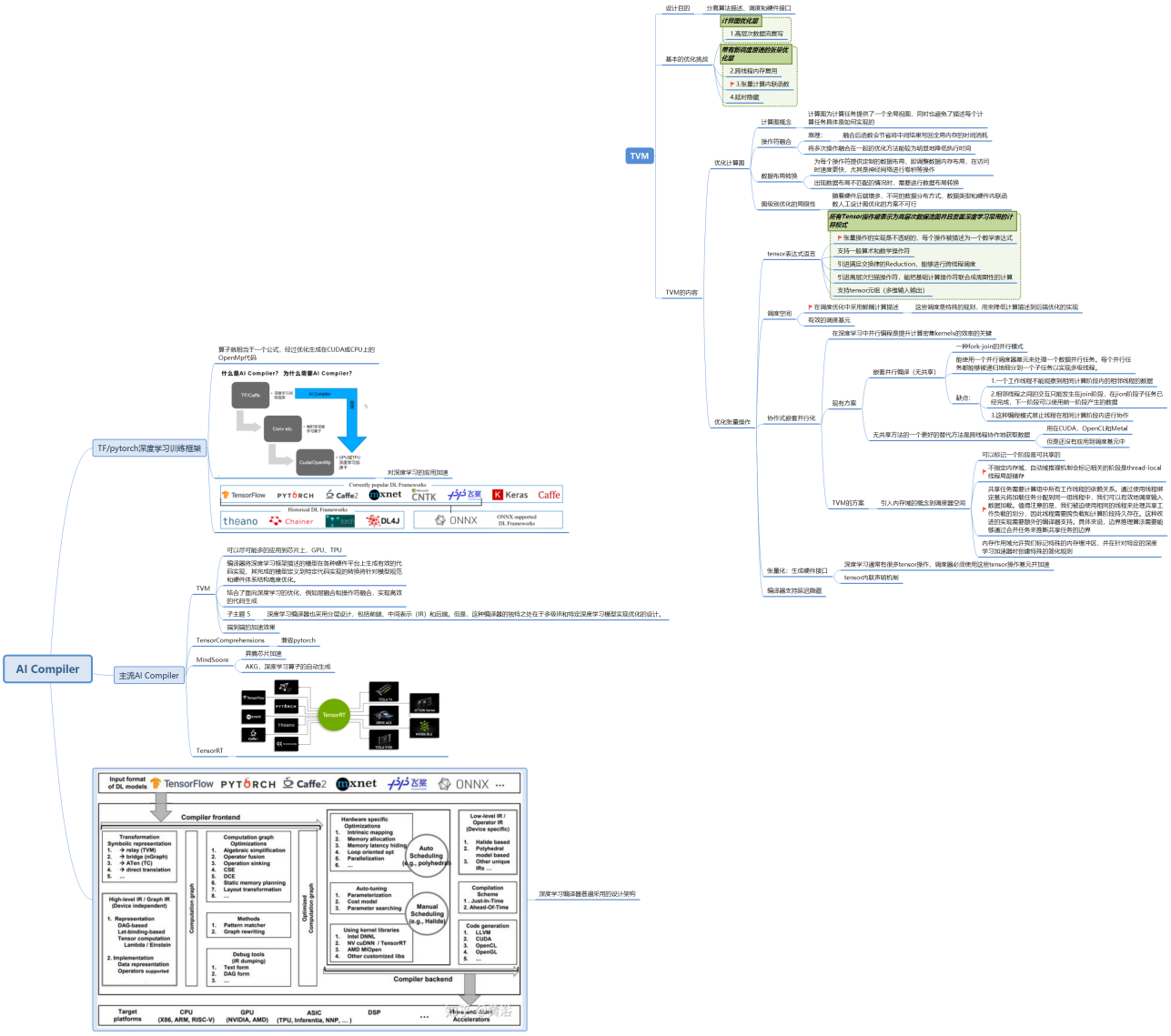
的参数值，最终返回最优参数配置。**用机器学习去代替人做调度参数的设计**



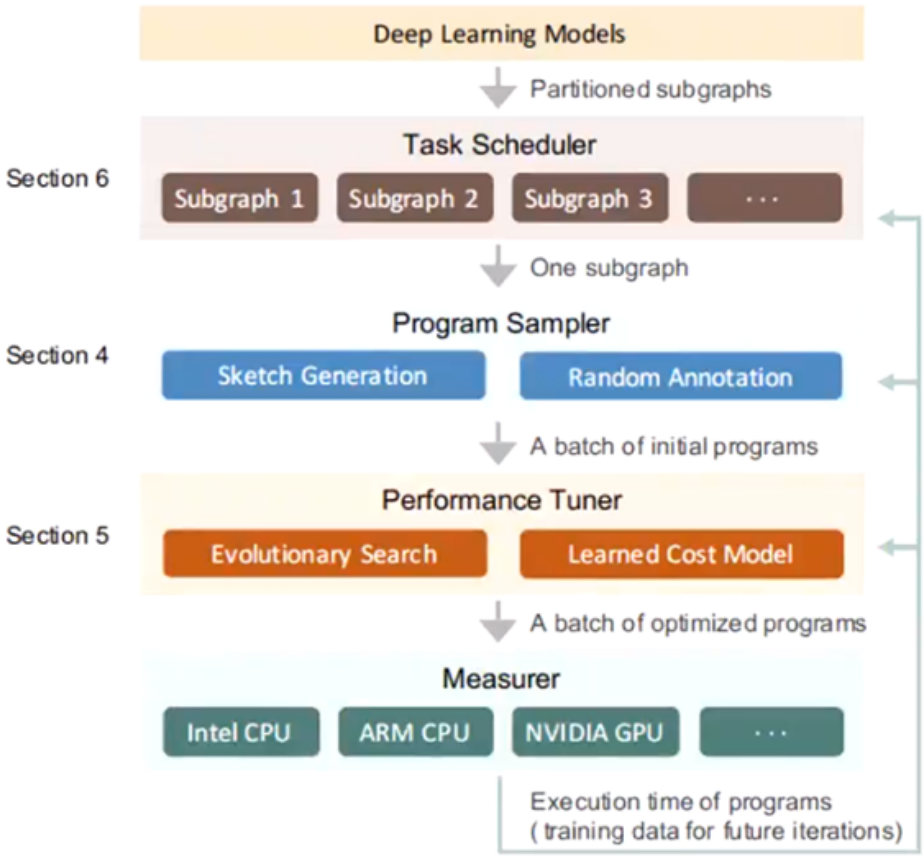
Cost Model　输入：代码生成器输出，硬件设备；输出：估计时间消耗。

使用cost model的目的是加速性能预测，给定一个algorithm, 可能有大量的candidate schedules需要评估性能，直接在硬件上评估性能耗时长，希望可以通过cost model模型快速评估性能。而采用神经网络/机器学习模型的原因和好处是，计算机硬件模型是相当复杂的（多层级存储结构，乱序执行等），使用一个简单的手工设计模型，比较难覆盖计算机硬件的复杂性从而比较难达到比较理想的性能预估。

TVM论文总结



autoSchedule流程图



autoSchedule主要由三部分组成：

（1）任务调度器：为优化神经网络中的多个子图分配时间资源；

（2）**程序采样器：**构造一个大的搜索空间并从中采样不同的程序；

（3）性能微调器：用于微调采样到的程序的性能。

autoSchedule流程：深度学习模型作为输入，使用Relay 的算子融合等操作将大模型划分为小子图。任务调度器用于分配时间资源以优化多个子图。在每次迭代中，它都会选择一个最有可能提升端到端性能的子图。对于选中的子图，autoSchedule分析它的张量表达式并为它生成几个草图模板，然后我们引入机器学习成本模型进行搜索以获得一批优化的程序。优化的程序被发送到实际硬件进行测量。测量完成后，分析结果并反馈以更新系统的所有组件。这个过程反复重复，直到优化收敛或我们用完时间预算。

autoSchedule相较于autoTVM的改进主要在于算子的程序模板template的人工设计到自动生成：

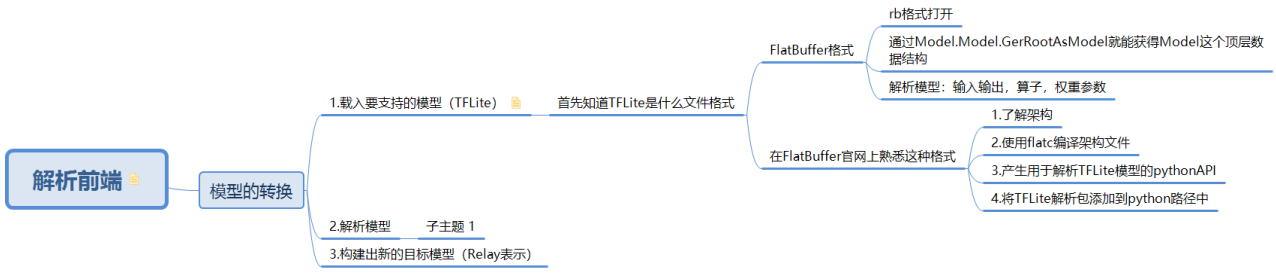
Template的缺点	Template-free的优点
该模板的设计需要目标硬件架构和算子语义的领域专业知识，数量较多，代码工作量大	消除大量手动工作，不必不停增加template代码

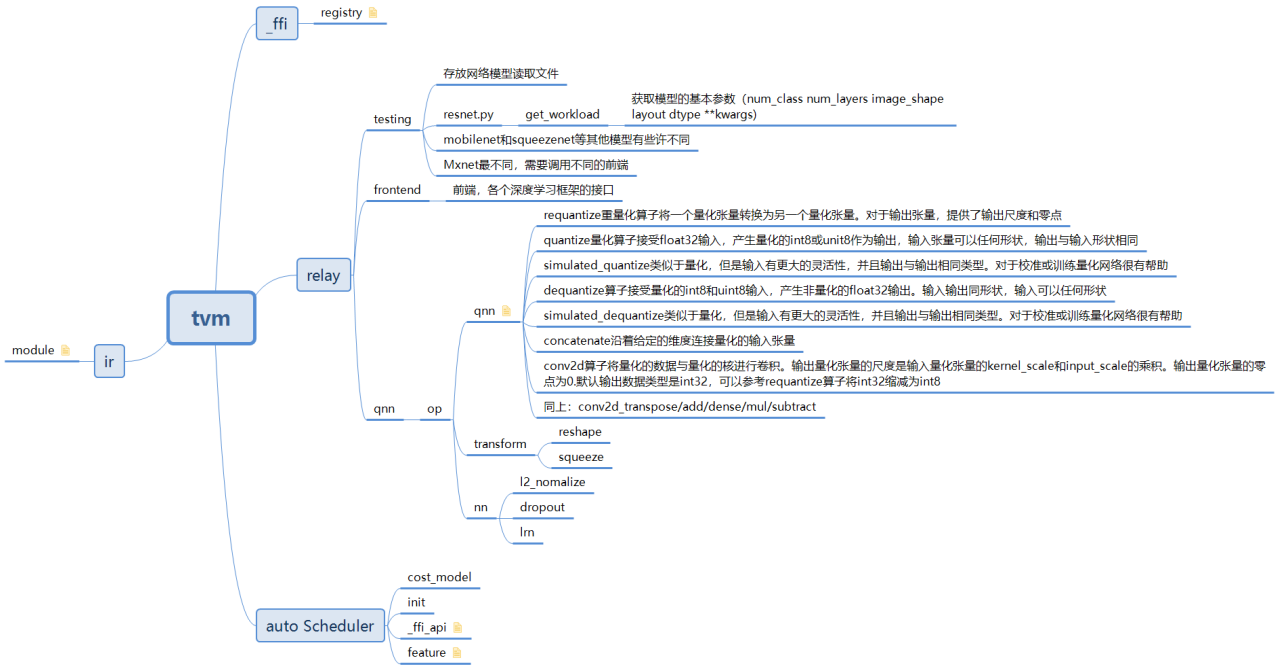
对于新的硬件平台以及更新新的算子时需要修改模板	更全局的优化，不是单独优化某一算子，而是组合优化，对应搜索空间也更大，更容易找到最优方案
优化每个具体算子，而算子之间具有相关性，所有算子最优也非全局优化	搜索速度更快，推理速度更快

关键点： 如何自动生成高质量的搜索空间是提升优化结果的关键，而costmodel的作用只是更快地逼近优化的上限。**所以，对于自动调优的问题，要想自动调优的结果有所突破，一个是数据（搜索空间），一个是特征工程（从原始数据中的找出/构建出一些具有性能优化意义的特征）。**

三、最近看代码的收获

近期开始阅读TVM的源代码，由于刚开始没有什么思路，就从前端开始看起的，了解了一下前端的搭建思路。欣哥指点了一下我阅读代码的整体思路，后面会注意不纠结于代码的细节而注重理解。





前端相关的代码在relay/frontend中，阅读的时候也是深感TVM源代码的框架之大，希望后面能

够拿捏。

下周从程序采样和后端开始看，后端是同公司自己的硬件条件紧密结合的，也是真正将TVM这一优秀算法应用到奥比自己产品的最重要一环。