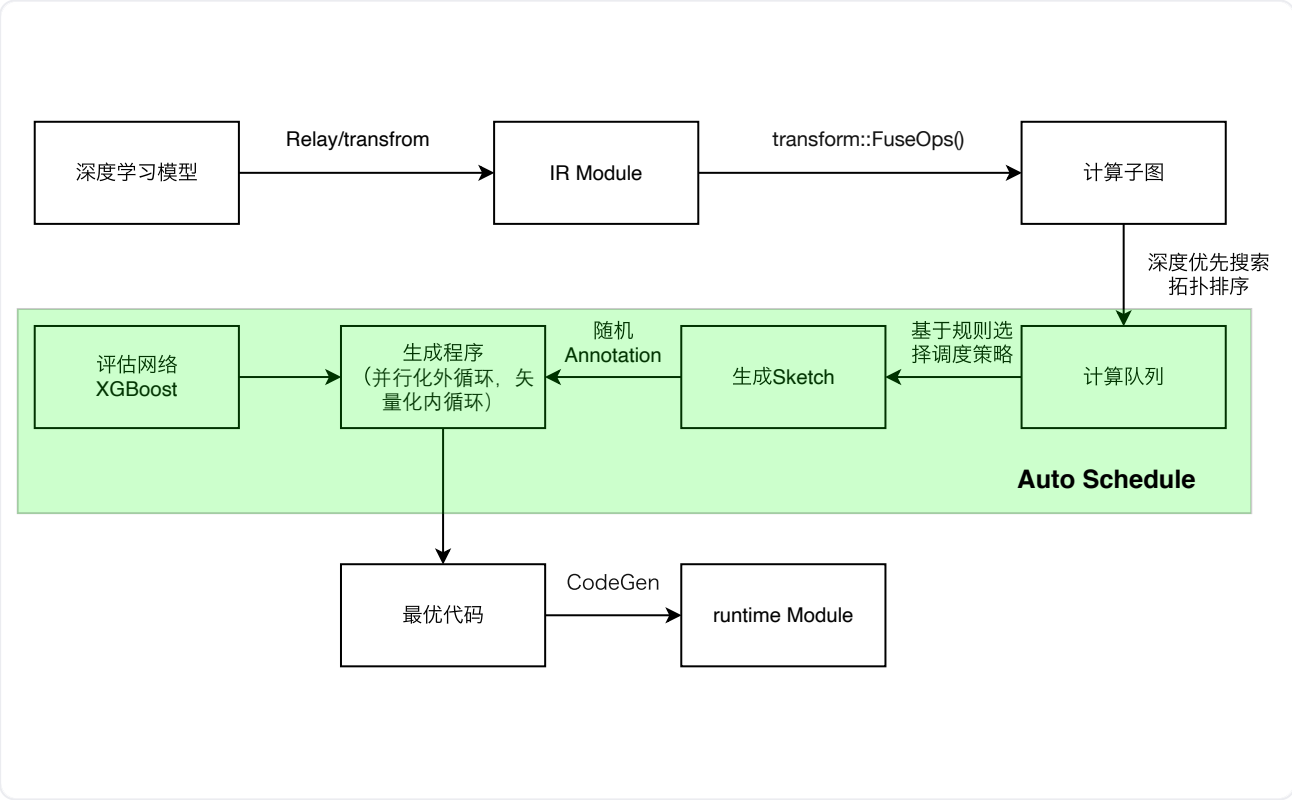
ANSOR中的costmodel主要用于加速对代码性能的评估。

ANSOR的流程：



Cost model模型用于预测每个程序的吞吐量。

ANSOR选用XGBoost作为代码性能评估的代价模型，用模型预估代替代码在实际硬件平台上跑节省更多时间和资源。

模型是通过代码中存在的一些内存复用、线程数量等特征来大致判断代码的快慢。这些特征可以**通过遍历TVM的tir**（tir包含底层程序的定义，还通过Op注册表定义了一组内置的函数及其属性）来获取：

指标类型	子类	特征集合
计算类	总操作数	float_mad, float_addsub, float_mul, float_mul, float_divmod, float_cmp, float_math_func, float_other_func int_mad, int_addsub, int_mul, int_divmod, int_cmp, int_math_func, int_other_func, bool_op, select_op,
	关键字相关	vec_num, vec_prod, vec_len, vec_type, unroll_num, unroll_prod, unroll_len, unrolll_type, parallel_num, parallel_prod, parallel_len, parallel_type,
	线程块相关	blockIdx_x_len, blockIdx_y_len, blockIdx_z_len, threadIdx_x_len, threadIdx_y_len, threadIdx_z_len, vthread_len;
访存相关	内存访问量	acc_type, bytes, unique_bytes, lines, unique_lines, stride
	内存复用	reuse_type, reuse_dis_iter, reuse_dis_bytes, reuse_ct
	二者比值	bytes_d_reuse_ct, unique_bytes_d_reuse_ct, lines_d_reuse_ct, unique_lines_d_reuse_ct
	步进	stride
计算密集度（单位访存下的计算量）	密集度	arith_intensity_curve
内存申请	申请空间大小	alloc_size, alloc_outer_prod, alloc_inner_prod, alloc_prod
循环相关	for 循环的特征	outer_prod，num_loops, float auto_unroll_max_step

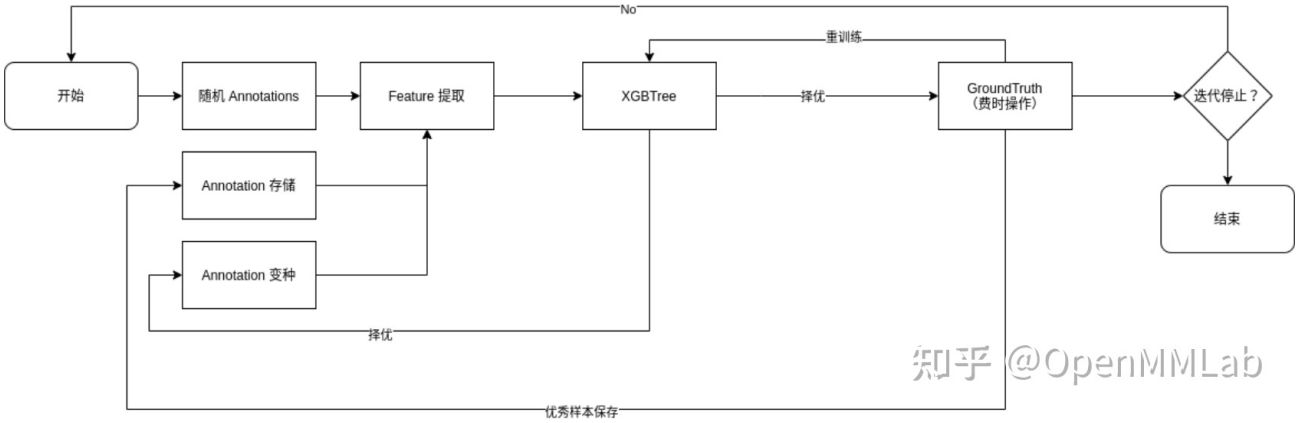
根据从细化的底层代码annotation中获取到的特征，输入网络得到较优代码，然后将较优代码通过

runtime得到ground truth，计算损失函数就可以训练出XGBoost模型的参数。

> 我们使用加权平方误差作为损失函数。因为我们主要关心从搜索空间中识别出性能良好的程序，所以我们更重视运行速度更快的程序。
>
> $$loss(f,P,y) = w_p(\sum_{s\in S(P)} f(s) - y)^2 = y(\sum_{s\in S(P)} f(s) - y)^2$$
>
> **f是模型，P是程序，y是程序吞吐量，S(P)是代码P中最内层的非循环语句，直接把y作为权重判断程序好坏**
>
> 所有程序都只需要用一个模型就能解决，并将同一DAG生成的程序的吞吐量归一化到[0-1]
>
> 每次都训练一个新的模型，而不是做增量更新

另外，为了提升训练数据量，将模型输出的优秀代码进行变异生成新代码，以及runtime结果的优秀代码都作为XGBoost模型的输入。即特征提取的输入有三个来源：

1. 随机的annotation，经过sketch之后的细化代码；
2. XGBoost模型输出的优秀代码，经过突变，增大了搜索空间，扩大样本集；
3. 之前迭代中得到的经过runtime的优秀代码。



迭代一定次数之后，评分高的annotation就可以作为最终代码，经过TVM后端生成硬件上运行的代码。

模型预测吞吐量用于评估代码好坏，代码在硬件上runtime出ground truth是真实运行时间。

进化操作重写和微调程序

Tile size mutation：这个操作随机选择一个平铺循环缩小一个因子，并将因子乘以另一个循环。

Parallel mutation：这个操作随机选择一个被标记为parallel的循环，通过融合相邻的循环级别或按因子分割它来改变并行粒度

Pragma mutation：程序中的一些优化是由compiler-specific pragma指定的（就是超参数）。这个操作随机选择一个pragma。对于这个pragma，这个操作将它随机地变异为另一个有效值。例如，我们的底层代码生成器通过提供auto_unroll_max_step=N pragma来支持使用最大步骤数的自动展开。我们可以随机调整数字N。

Computation location mutation：这个操作随机选择一个不是多层平铺的灵活节点(例如，卷积层中的填充节点)。对于这个节点，操作随机地将其计算位置更改为另一个有效的附加位置（这里应该指的是计算的缓存位置，首先针对非多层平铺，其缓存要求过大）

Node-based crossover：结合两个现有程序的重写步骤生成一个新的程序。在sketch generation和random annotation阶段记录重写步骤。在底层代码生成器中通过依赖分析合并之后程序的正确性。