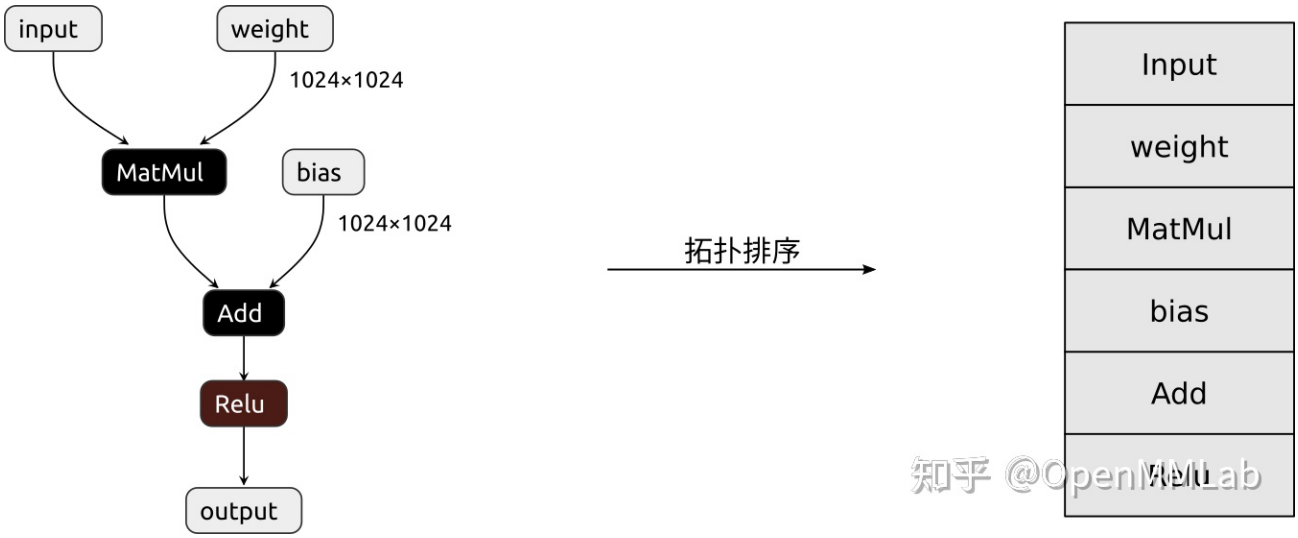
Sketch 对每个计算子图进行优化，可以分为以下三个步骤：

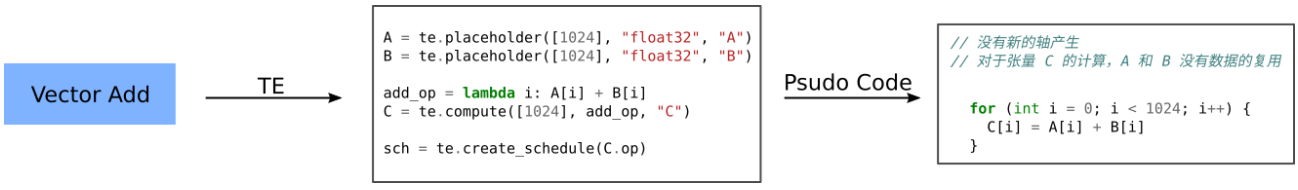
1. 拓扑排序
2. 静态分析
3. 调度策略的选择



ANSOR 采用**深度优先搜索**的方式将计算图转化为计算队列，方便后续的静态分析和调度策略的选择。

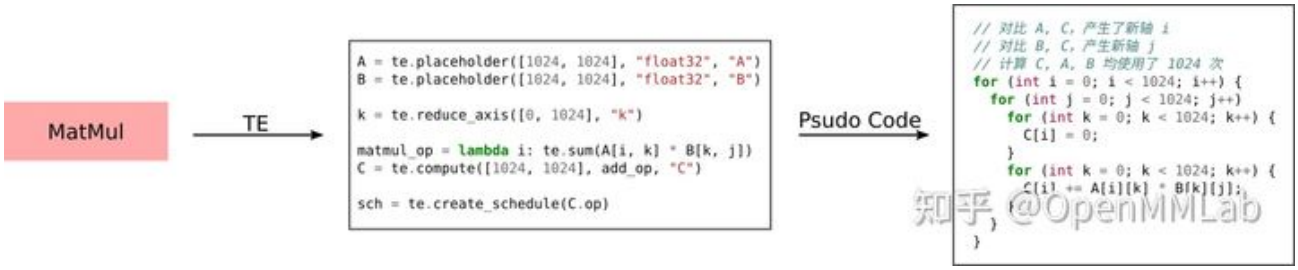
接着，**静态分析**用来提取算子的特征，这一步非常关键，也是 ANSOR 的一个创新点。ANSOR 不会重点对某一个算子类型做处理，而是提取算子的特征。

一个典型的特征就是这个算子是否对输入数据做了复用。例如 Vector Add 算子，其 Tensor Expression 的表达和伪代码如下所示：



它输出的 axis 和输入的 axis 相同，均为 i，因此没有数据的复用。

而 Matmul 算子，其 Tensor Expression 的表达和伪代码如下所示：

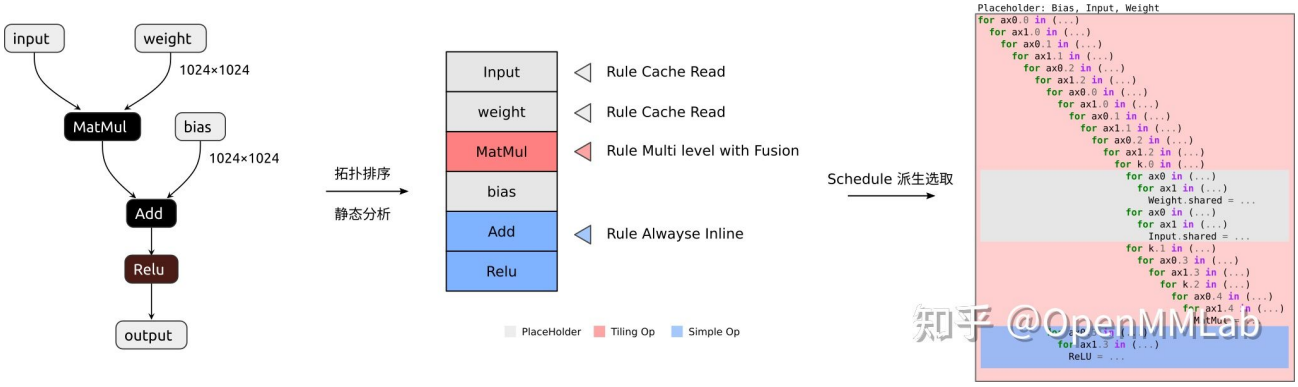


输入 A 和输入 B 产生输出 C，C 的 axis 为 i, j；A 的 axis 为 i，k；B 的axis 为 k，j。说明在计算 C[i, *] 的过程中，A[i, k] 被重复使用了 Range(j) = 1024 次，B[k, j] 也被重复使用了 Range(i) = 1024 次。如果我们**在实际代码生成的过程中，将 A 和 B 的一部分数据进行缓存**，可以显著降低带宽，从而减少运行时间。

ANSOR 根据静态分析提取了如下算子特征：

特征	说明
IsStrictInlinable绝对内联	输入和输出的维度是否是一一对应的，且语句中不含有 if-then-else 等特殊的表达式
HasDataReuse数据复用	在计算输出数据时，输入的数据是否被重复使用
Has Fusible Consumer可融合	后面的算子是否可以和该算子融合
Has More Reduction Parallel有更多并行化	该算子主要为 reduce 操作，这种情况是否需要用 rfactor 调度原语

在静态分析获取算子特征后，下一步为派生策略的选择，**从后向前遍历拓扑排序生成的计算队列（策略从后往前选）**，根据算子的特征为算子选择调度策略。 ANSOR 提供了如下调度策略：



2.3 Annotation

Sketch 虽然生成了代码的结构，但是一些细节还没有确定，如：

1. GPU thread bind
1. for 循环的 unroll、vectorize、parallize
1. Split 的 factor

Annotation 会随机确定上述内容，生成完整的代码。Annotation 只负责随机的生成代码，并不会考虑性能，性能由 Evolutionary Search 来保证。

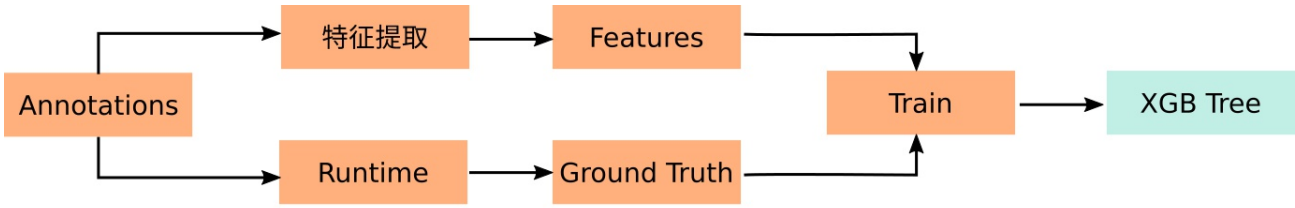
2.4 Evolutionary Search

待优化的计算有可能十分复杂，遍历所有的 Annotation 来获取最优实现是不现实的。ANSOR 训练了一个 XGBoost Tree 来对代码的性能进行评估，相比直接跑 Runtime 获取 ground truth，训练获得的 XGBoost Tree 可以大致确定代码的好坏,并且花费的时间更少。为了评定 Annotation 生成的代码的性能，ANSOR 对每个算子的实现提取了如下特征来训练模型，这些特征可以通过遍历 TVM 的 tir 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 Annotation 获取的样本，提取上述特征，再通过 Runtime 获取样本的 ground truth，就可以训练一棵 XGBoost Tree：

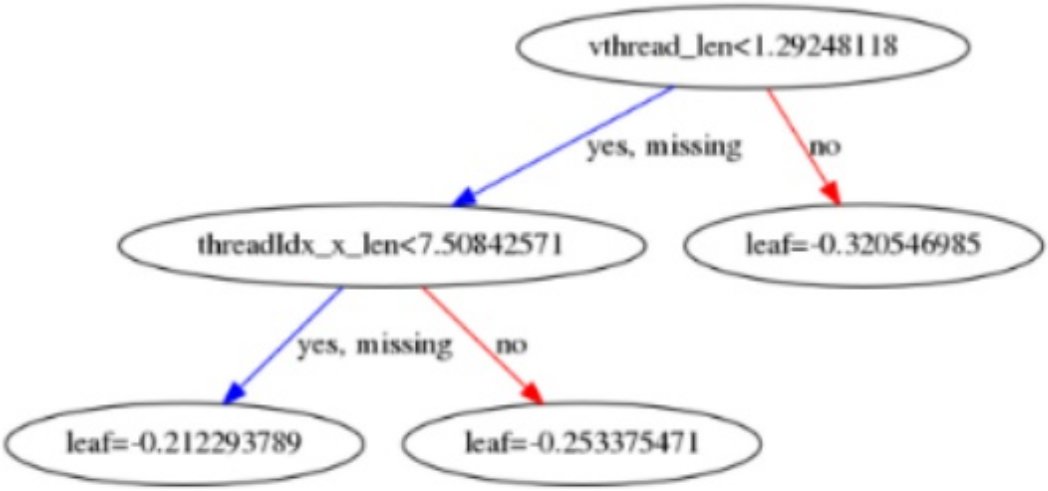
（根据这些特征，跟ground Truth进行训练，得到权重等参数，得到XGBTree模型，用于预测样本需要的运行时间，彻底替代groundtruth）



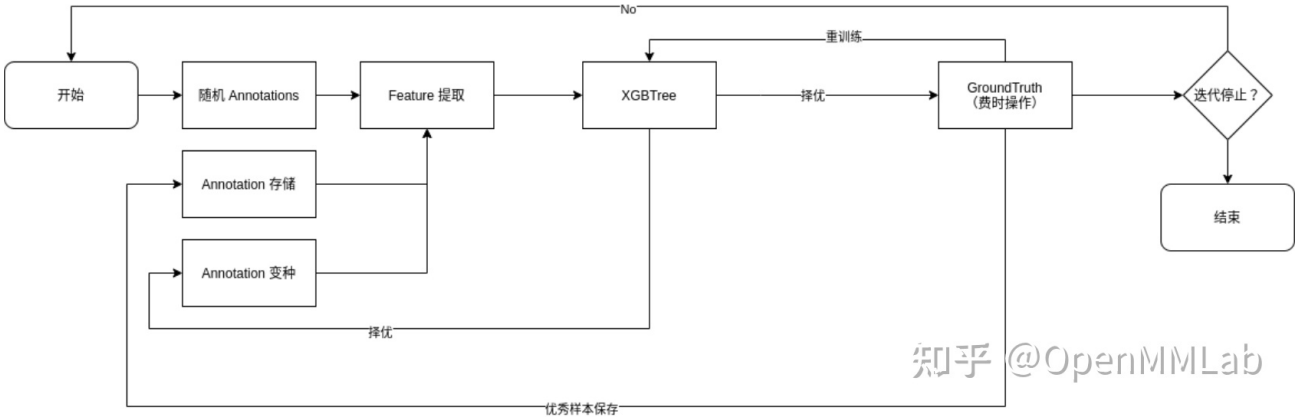

```
for ax1 in (Random)
  for ax2 in (Random)
    ...
  for ax3 in (Random)
    ...
for ax4 in (Random)
  for ax5 in (Random)
    ...
  for ax6 in (Random)
    ...
...
```

$$\longrightarrow \quad x_i^0, x_i^1, x_i^2, ..., x_i^n, y_i \quad \longrightarrow \quad \hat{y} = f(x^0, x^1, ..., x^n)$$

通过 XGBoost 库提供的接口可以对 XGBoost Tree 可视化，一个 Matmul + ReLU + Add 的 Annotation 样本训练出的 XGBoost Tree 如下图所示，对于矩阵乘法来说，XGBoost Tree 的特征向量是比较稀疏的，起决定性的特征主要有 vthread_len、threadIdx_x_len、blockIdx_x_len 等。



ANSOR 应用 Evolutionary Search，使得优化能更快的收敛，减少遍历的时间。整个 Evolutionary Search 的流程如下图所示：



可以看到特征提取的输入有三个来源：

1. 随机的 Annotation，保证了样本的丰富性。
2. 之前迭代中获取的 Annotation 优秀样本，有助于训练得到更准确的 XGBTree 。
3. 优秀样本的变种，根据进化算法，好的样本的 "突变" 往往会更容易获得好的样本，因此可以采用对好的 Annotation 的参数进行微调的方式扩充样本集。

提取的特征会用来训练 XGBoost Tree，XGBoost Tree 又能对 Annotation 样本进行一次初筛选，将评分高的 Annotation 通过 TVM Runtime 获得 Ground Truth。根据 Ground Truth 就可以选择出优秀的代码实现，也就是 ANSOR Pipeline 的输出，算法至此就运行完毕了。

ANSOR 的输出可以无缝的转化成 Schedule，省去了手写 Schedule 的烦恼，直接使用 TVM

CodeGen Pipeline 即可进行代码的生成。

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。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。

进化搜索利用变异和交叉重复生成一组新的候选程序，并输出一组得分最高的小程序。这些程序将在目标硬件上编译和测量，以获得真实的运行时间成本。收集到的测量数据被用于模型的更新
这样，学习到的代价模型的准确性逐渐提高，以匹配目标硬件。因此，进化搜索逐渐为目标硬件平台生成更高质量的程序
不同于TVM和FlexTensor中的搜索算法，它们只能在固定的类网格参数空间中工作，Ansor中的进化操作是专门为张量程序设计的。它们可以应用于一般的张量程序，并且可以处理具有复杂依赖关系的搜索空间。

与Halide自动调度器中的展开规则不同，这些操作可以对程序执行无序修改，解决顺序限制