AI for Compiler 介绍及最新进展

AI for Compiler 优化介绍及最新进展

Compiler SIG committer 刘飞扬

开放原子开源基金会 OPENATOM FOUNDATION | OpenEuler

目录

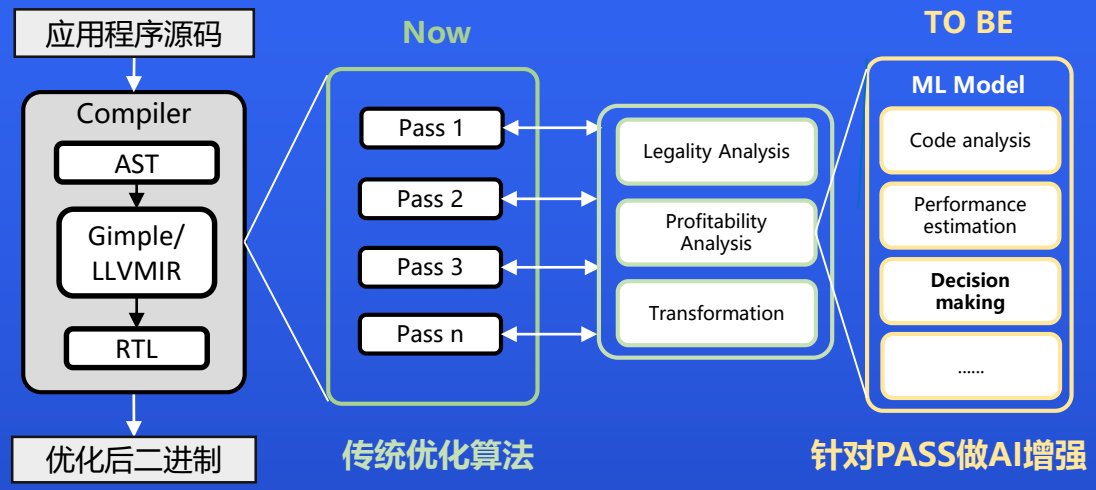
AI for Compiler 优化介绍及未来规划

AI for Compiler背景

由于编译优化算法的不断迭代，逐渐复杂化的启发式算法阻碍了编译器开发者的维护和算法进一步演进。在开发阶段，这种高复杂度的启发式算法容易对开发的软硬件环境产生依赖，导致在生产环境发生变更时，启发式算法的普适性较弱。同时，在维护阶段，编译器开发人员时常需要重新调优成本模型参数或者编译选项组合以适应新环境下的工作负载。最近的学术界和业界的研究表明，编译器可以通过用ML（机器学习）策略替换复杂的启发式算法或者用ML模型进行编译调优，来拓展编译器优化中的更多机会，提高编译优化算法的泛化性，并改善维护时的调优效率。

AI for Compiler业界技术探索

AI辅助编译优化



- **Google**
  - (Inline/RegAlloc) MLGO (2021)：使用**强化学习**训练神经网络来作决策，**替代inline启发式算法**，编译器自动收集数据改进决策策略。训练后的策略在迁移到其他软件编译时，减少3% ~ 7% 的代码体积。
- **Meta**
  - (LLM) LLM Compiler (2024)：基于CODE LLAMA，使用5460 亿个LLVM-IR和汇编代码标记的庞大语料库上训练模型，模型在**代码大小优化**上的潜力达到了自动调整搜索的77%，可显著缩短编译时间。
  - (Framework) CompilerGym (2023)：提供**编译器优化任务的强化学习环境的工具集**，帮助降低学术界编译优化研究门槛与业界编译优化任务的工程投入。
  - (Representation) ProGraML (2021)：提出一个**基于程序 IR 信息的图表示**，可以包含 IPA阶段的控制流、数据流、调用信息，并应用于各种深度学习模型训练。

编译器自动调优

编译器调优层级

语言级

函数级

编译选项级

Pass序列级

优化算法级

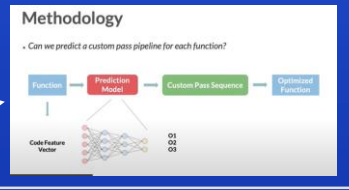
- **Mojo：Language Integrated Auto-tuning** (2023)

```
FILE_NAME.🔥
def exp_buffer[dt: DType](data: ArraySlice[dt]):

  # Search for the best vector length
  alias vector_len = autotune(1, 4, 8, 16, 32)

  # Use it as the vectorization length
  vectorize[exp[dt, vector_len]](data)
```

  - Framework级Auto-tuning，自动结合配置，找到最合适目标硬件优化方式。
- **Switch per function in LLVM** (2023)
  - 基于 LLVM 编译器，对 clang 开关作为函数属性的支持实现函数级调优。
- **Machine Learning Guided Ordering of Compiler Optimization Passes** (2021)



  - 每个函数选择自定义的优化序列。
- **Iterative Compilation - Give the compiler a second chance** (2023)
  - 针对启发式算法最更深层次的迭代调优

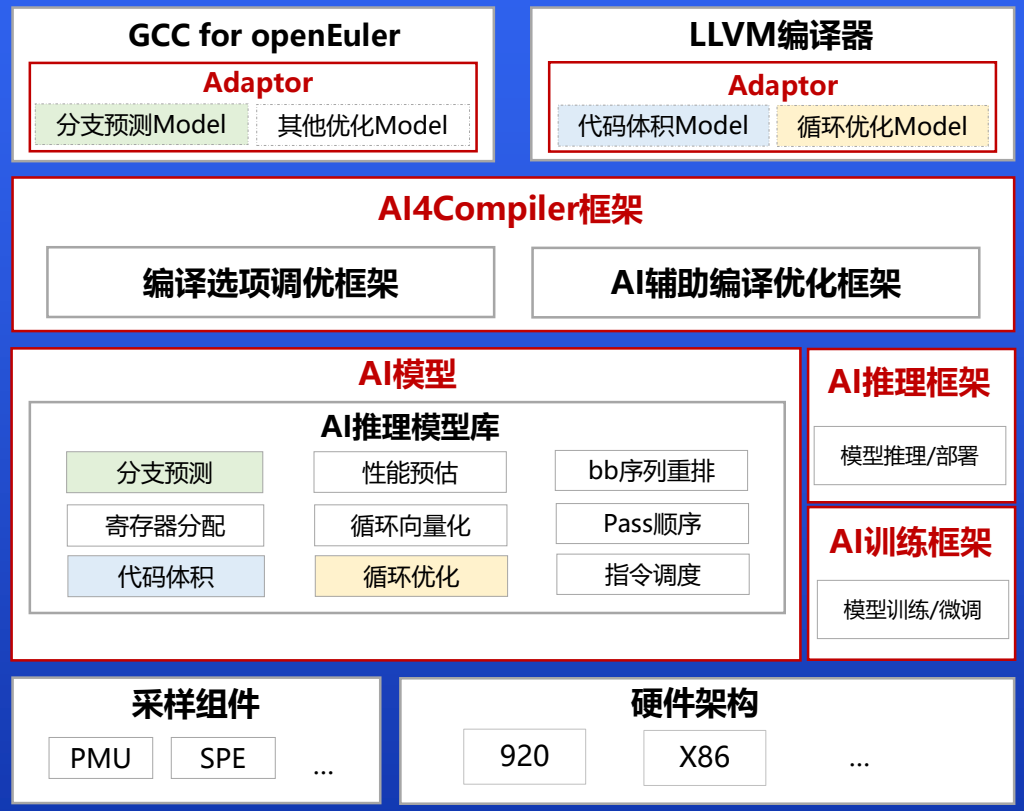
AI forCompiler业务挑战

用户使用流程

业务挑战

- 软硬件环境发生变更时，旧场景下开发的启发式编译优化算法普适性较弱，无法适用于新环境
- 应用在新负载下，调优人员需要重新调整应用的构建编译选项，所需调优时间较长

AI for Compiler技术点规划

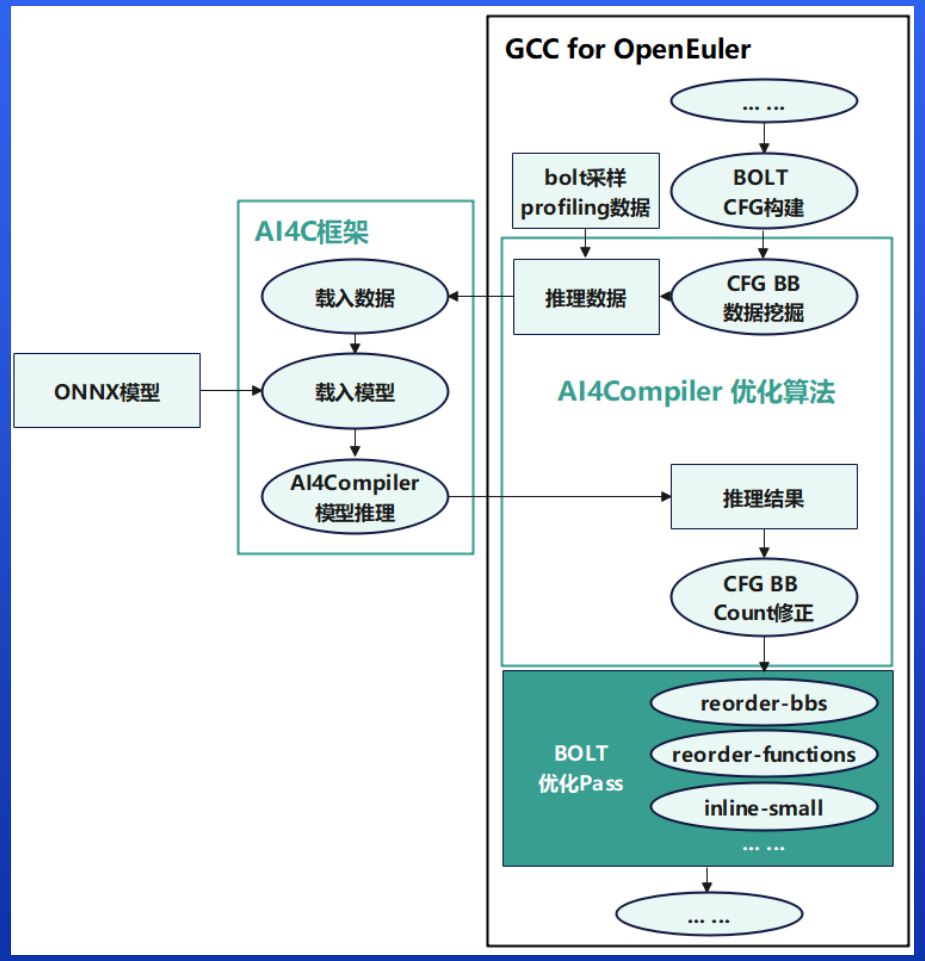


规划技术点

- 白盒特征提取：基于应用IR、程序CFG图、硬件平台信息、动态采集信息等内容，利用GNN或Transformer训练有效反映软硬件信息的特征，作为目标任务的输入。
- 选项调优：结合白盒特征，通过缩小或裁剪参数空间或者优化空间搜索算法等方式，加速旧应用新环境/新应用的调优效率。
- 性能评估：根据特征提取的程序表征以及编译选项组合，直接预测应用性能。
- 成本模型替换：结合白盒特征，辅助典型的编译优化，例如，数据预取等，以实验数据驱动的ML模型替换启发式成本模型，提升应用场景覆盖率及优化效果。

GCC上AI for Compiler的优化实践

MySQL：BOLT的基本块精度修正模型加速TPCC测试项性能



业务挑战

- **PGO易用性较差**： PGO插桩在易用、通用、性能损耗等方面存在缺陷
- **BOLT优化后前端瓶颈仍较高**： MySQL的TPCC测试项在使能编译器的CFGO反馈优化（采样）后，距离传统GCC PGO插桩模式仍有7.5%左右的差距，同时TPCC测试项的Frontend Bound还在32.6%左右

解决方案

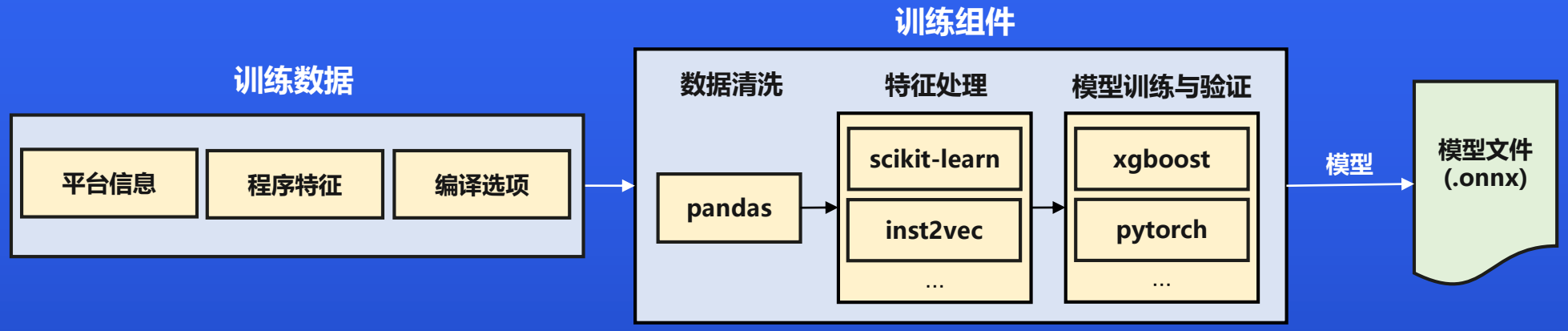
- **特征提取**：在speccpu2017、PostgreSQL等应用中，提取BOLT CFG中BB的内部指令、所处循环、函数、兄弟节点等相关静态特征，共220w训练样本
- **XGBoost 模型**：使用XGBoost模型来预测CFG的BB Count插桩值，使采样BOLT达到接近插桩BOLT的优化效果
- **BOLT CFG BB Count修正**：根据模型预测的置信度阈值以及父子节点的BB Count采样值，修正BOLT CFG内预测插桩值非零而采样为零的BB

优化收益

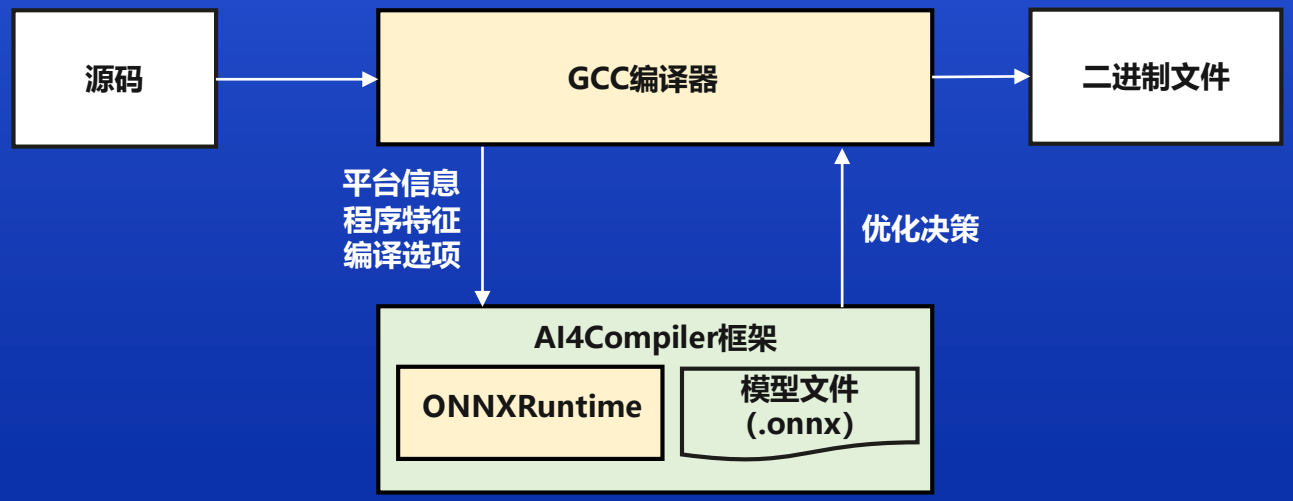
- **优化结果**：通过BOLT的基本块精度修正模型和编译选项调优等技术，MySQL基于CFGO特性（AFOT-BOLT采样）基线，TPCC测试项提升5%， BOLT的基本块精度修正模型逼近BOLT插桩模式上限

主要特性介绍——AI4C推理框架

编译器开发阶段（模型训练）：

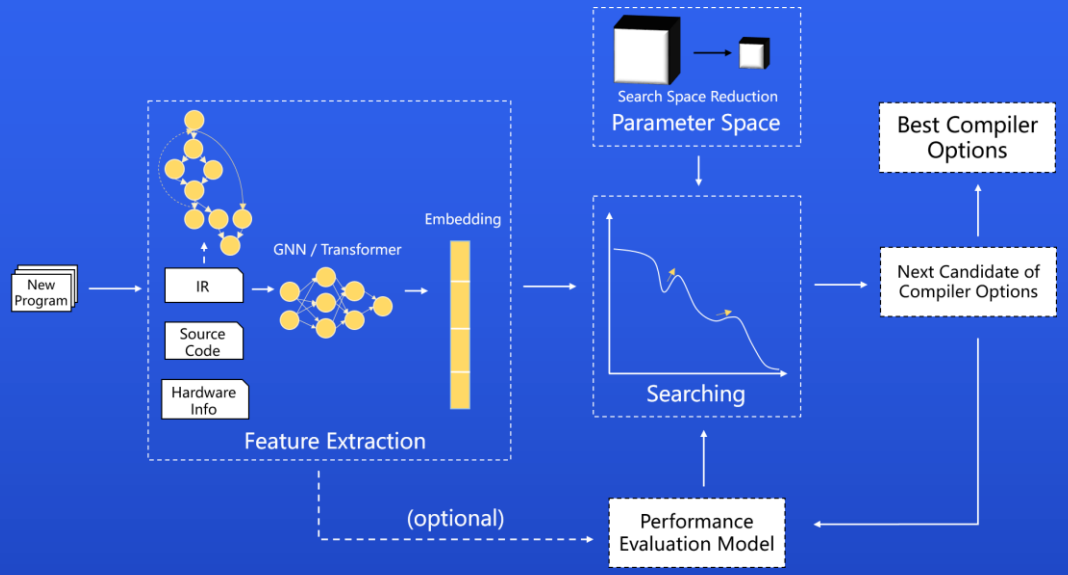


编译器使用阶段（模型推理）：



未来规划

编译自动调优工具（Autotuner）支持GCC



背景：

- 编译自动调优工具（Autotuner）将在9月30日开源至openEuler社区

未来规划：

- 在Autotuner中增加对GCC的支持
- 优化Autotuner目前支持的搜索算法与参数搜索空间
- 结合白盒信息分析代码特征，缩短应用调优时长，加速迁移对不同环境和硬件底座的调优

GCC优化适配LLVM预训练模型

1. **多编译器支持：**插件基于**MLIR**进行优化开发，基于MLIR处理不同编译器IR的转换，插件开发者对不同编译器可**避免重复研发插件**，一次开发可多编译器使用。
2. **支持独立插件：**插件服务与编译进程隔离运行，插件逻辑可与编译器解耦，**独立研发和部署**。

背景：

- 当前业界/学术界的预训练模型基本基于LLVMIR，GCC侧需要重新训练表征

插件框架方案：

- 通过插件框架将gimple转换为LLVMIR，能够保留映射信息，不存在对应问题
- 直接通过MLIR上学习软件白盒信息，让GCC和LLVM的AI辅助编译优化共享同一个IR学习的基底

结语

目前，学术界和业界都在AI for Compiler技术上有过积极探索， 例如，Google的MLGO和Meta的LLVM Compiler等技术。Compiler SIG近期在社区引入AI for Compiler技术的开发和构建，部分优化已显示出在代码体积、运行时间、吞吐量等性能目标提升，同时优化具备一定的泛化性。

Compiler SIG未来希望打通GCC编译器和LLVM编译器的技术壁垒，共建共享AI for Compiler的技术底座。

THANKS

Compiler SIG技术交流

- Compiler SIG 专注于编译器领域技术交流探讨和分享，包括GCC/LLVM/OpenJDK 以及其他的程序优化技术，聚集编译技术领域的学者、专家、学术等同行，共同推进编译相关技术的发展。

Compiler SIG 主页