计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202110943
60(5): 1121−1135, 2023

基于图神经网络的 OpenCL 程序自动优化启发式方法

叶贵鑫　张宇翔　张　成　赵佳棋　王焕廷
（西北大学信息科学与技术学院　西安　710127）
（陕西省无源物联网国际联合研究中心（西北大学）　西安　710127）
（gxye@nwu.edu.cn）

Automatic Optimization Heuristics Method for OpenCL Program Based on Graph Neural Network

Ye Guixin, Zhang Yuxiang, Zhang Cheng, Zhao Jiaqi, and Wang Huanting
（*School of Information Science and Technology*, *Northwest University*, *Xi'an* 710127）
（*Shaanxi International Joint Research Centre for the Battery-free Internet of Things*(*Northwest University*), *Xi'an* 710127）

Abstract　The last decade years witnessed the rapid development of heterogeneous computer architecture due to the popularization of the Internet of things. As the first cross-platform heterogeneous parallel computing framework, OpenCL(open computing language)has the advantages of standardization and portability. However, OpenCL has certain defects in performance portability because of the complexity and diversity of software and hardware platforms. To address this problem, prior methods leverage deep learning to build an optimization model. But they suffer from an insignificant code optimization effect because existing deep learning-based methods only capture the order dependencies of the program while ignoring the syntactic and semantic information. To this end, we propose ACCEPT, an automated heuristic optimization on OpenCL programs by building a multi-relational graph neural network. Differ from existing methods, ACCEPT first extracts the deep structure and semantic features of the OpenCL program by constructing a multi-relational code graph, then applies an improved graph neural network to extract the high-dimensional feature representation of the constructed code graph. Finally, a decision neural network is used to yield the optimization parameters. We evaluate ACCEPT with heterogeneous device mapping and thread coarsening factor prediction tasks. The experimental results show that ACCEPT outperforms state-of-the-art (SOTA) methods. On the heterogeneous device mapping task, the accuracy can reach 88.7%, and the speedup can be increased by 7.6% compared with the SOTA methods. On the thread coarsening task, the speedup is 5.2% higher than SOTA methods.

Key words　heuristic optimization; graph network; OpenCL; deep learning; heterogeneous device

摘　要　物联网的发展与普及促使计算机异构架构迅速发展，开放运算语言（open computing language, OpenCL）作为首个跨平台异构并行计算框架，具有标准化、可移植性等优点，但因不同平台下软硬件的复杂性和多样性，使 OpenCL 在性能上的移植性存在一定的缺陷. 现有的方法通过深度学习构建优化模型来提高程序运行效率，但所构建的预测模型仅考虑代码的顺序依赖关系，忽略了语法语义信息，导致代码优化效果不明显. 为解决上述问题，提出了一种基于多关系图神经网络的 OpenCL 程序自动优化启发式方法. 该方法首先把 OpenCL 代码转换成多关系代码图，能够提取代码的深度结构与语法语义特征；然后利用改进后的图神经网络模型，将构建的代码图编码为高维的特征向量；最后使用决策网络完成任务预测.

收稿日期：2021-09-15；**修回日期**：2022-03-18
基金项目：国家自然科学基金项目（62102315, 61972314）；陕西省国际合作项目（2021KW-04, 2020KWZ-013）
This work was supported by the National Natural Science Foundation of China (62102315, 61972314)and the International Cooperation Project of Shaanxi Province (2021KW-04, 2020KWZ-013).

为验证方法的有效性,分别在异构设备映射和线程粗化因子预测2个任务上进行实验评估.结果表明,在异构设备映射任务中,最优设备预测准确率能够达到88.7%,相较于现有最先进的方法,加速比可提高7.6%;在线程粗化任务中,加速比相较于现有最优的方法可提高5.2%.

关键词 启发式优化;图网络;OpenCL;深度学习;异构设备

中图法分类号 TP391

近年来,随着大数据、人工智能技术的应用与普及,计算机体系结构朝着异构化、并行化的方向发展,计算能力强、执行效率高的异构计算平台是当前环境所迫切需要的[1].为实现异构并行计算,开放运算语言[2](open computing language, OpenCL)框架应运而生.因OpenCL在C99—ISO/IEC 9899:1999标准上进行了优化扩展,因此它能够在CPU、GPU、数字信号处理器等处理器上执行.然而,由于异构平台软硬件环境的差异,OpenCL性能的可移植性[3]存在较大的差异,即同一个OpenCL程序在不同异构平台下的执行效率有较大差异.

现有的针对OpenCL的优化工作通常围绕异构并行映射及线程粗化因子预测任务展开.异构并行映射任务旨在将目标程序映射到合适的硬件设备上(如GPU或CPU),以实现高效的执行.线程粗化因子预测旨在为即将执行的程序分配适当的线程数,以实现系统开销与程序并行执行效率的最大收益,故需要判断线程是否会在粗化过程中收益,从而获得最佳的粗化因子.

针对上述2种代码优化任务,现有的解决思路是由专家知识定义优化特征,并利用机器学习技术构建专家预测模型,进而推断出执行效率高的异构平台(CPU或GPU)或最优粗化因子[4-5],但该类方法需要根据专家知识和经验人工提取特征,不仅需要较高的专家成本,也难以实现端到端的自动优化.随着深度学习技术的发展,深度学习被广泛应用到代码建模相关任务中,如克隆检测[6]、代码自动生成[7]、漏洞检测[8]等.近年来,研究者提出利用深度学习技术构建代码自动优化预测模型[9-10].而现有的基于深度学习技术的优化方法[6],大多利用自然语言处理模型如长短期记忆(long short-term memory, LSTM)神经网络[11]、循环神经网络(recurrent neural network, RNN)等,将代码按照自然语言的格式进行编码,并输入到黑盒深度学习模型中构建代码优化预测模型.这使得预测模型仅能够捕获代码的顺序依赖关系,而难以提取代码所特有的深度结构与语义信息.因此与传统基于机器学习方法相比,预测效果并无明显提升.

针对现有方法在上述2种代码优化任务上存在的问题,本文提出了ACCEPT(automatic OpenCL code optimization heuristics),一种基于多关系图神经网络的OpenCL程序自动优化启发式方法.该方法首先将OpenCL代码表示成图数据结构,然后利用图神经网络[12]学习代码图中的深层结构与语义信息,将图中节点信息映射到高维的向量空间中以提取代码的深度特征信息,最后利用全连接神经网络层构建异构映射和线程粗化因子预测模型.本文一方面基于程序的抽象语法树(abstract syntax tree, AST),结合数据流图和控制流图以及程序中变量的传递、调用关系,在源代码上构建了一种具有多种边关系的程序图结构,以增强变量间的依赖关系;另一方面,为了丰富图中节点的特征信息,最大程度地提取代码的特征信息,本文基于编译器框架LLVM[13](low level virtual machine)的中间表示形式(intermediate representation, IR),添加了顺序流、控制流和数据流,在IR代码上构建了一种多关系程序图.同时,通过改进现有的图神经网络[11],使其能够处理上述2种类型的程序图结构.此外,为了解决数据不足的问题,本文在线程粗化因子任务中使用了迁移学习技术[14],用于从异构映射任务中迁移标注数据以改进线程粗化任务的学习效果.

为评估方法的有效性,本文在异构映射任务与线程粗化因子预测任务上进行实验.在异构映射任务中,使用7种标准数据集,在3种平台下与现有的6种经典方法进行对比实验.在线程粗化因子预测任务中,采用17个OpenCL内核程序,在4种平台下与现有的5种方法进行对比.实验结果表明,在异构映射任务中,本文方法的预测准确率能够达到88.7%,且加速比与现有最优方法相比提高7.6%.在线程粗化因子的预测任务中,加速比与现有最优方法相比提高5.2%.

本文的主要贡献有3个方面:

1)提出了一种程序图构建方法.该方法既能够表征代码的深度结构特征,又能够捕获代码的语法语义信息.

2）提出了一种基于多关系图神经网络的 OpenCL 程序自动优化启发式方法 ACCEPT. 与现有方法不同，该模型能够自动提取代码的深层次结构和语义信息，并结合迁移学习技术进行建模，能够有效解决训练数据不足的问题.

3）进行了充分的对比分析实验. 结果表明，ACCEPT 在异构映射与线程粗化任务上均优于当前最优的方法，加速比相较最优方法分别提高 7.6% 和 5.2%.

1 相关工作

目前，主流的启发式优化方法均采用了机器学习和深度学习技术，机器学习的优势是不需要获取硬件平台的配置信息，仅需获得程序的静态及动态特征信息[4,15-16]，通过对特征自动化建模来完成编译运行时优化任务. 而现有工作大多数采用人工的方式筛选特征，由于编译过程中涉及大量的解析优化过程，使得人工挑选出的特征表征能力不强，并且挑选特征的过程耗时耗力，无法实现端到端的自动优化. 随着深度学习技术的发展，研究者提出利用深度学习模型来构建代码优化预测模型，即将源代码类比成自然语言进行处理，使用深度神经网络自动提取特征，虽然实现了端到端的优化，但该方法更多关注于提取代码的顺序依赖关系，而忽略了代码所特有的语法语义与结构信息. 本节接下来分别介绍现有的基于传统机器学习技术与基于深度学习技术的代码优化方法.

1.1 基于传统机器学习技术的代码优化

Grewe 等人[4] 针对异构设备并行任务，利用决策树算法构建预测模型. 为此，其通过获取代码的静态信息如计算操作、全局内存和局部内存信息，以及程序执行的动态信息如数据传输大小、内核工作项大小作为代码特征信息. 而上述特征的设计需要特定领域的专家知识，并且仅适用于支持 LLVM 的异构设备的代码优化任务.

Magni 等人[15] 针对线程粗化因子任务研发了一套预测模型，该模型利用二元决策过程，在 6 个粗化因子（1, 2, 4, 8, 16, 32）中预测 1 个最佳的值. 模型将 1 个 6 分类问题转换成递归的二元判断任务. 例如，第 1 次判断能否从粗化中受益，若不能受益，则最佳粗化因子为 1；否则进行粗化，之后再判断是否从中受益，若不能受益，则最佳粗化因子为 2；否则进行粗化并进入下一轮的递归判断，直至 5 次决策完成为止. 文献 [15] 的作者手动挑选出 17 个代码相关的特征，如程序分支语句个数、指令个数等. 分别计算每个相关特征的粗化收益率，然后使用主成分分析（principal components analysis, PCA）算法，在保存信息量 95% 以上的前提下获得前 7 个主成分.

1.2 基于深度学习技术的代码优化

Cummins 等人[9] 提出了一种基于文本的预测模型，该模型可直接输入源代码，并可以应用于异构映射和线程粗化 2 种代码优化任务. 首先使用自定义的编码方式为 OpenCL 程序构建字符表，并通过字符表将每个 OpenCL 程序编码成等长的数字序列；然后使用词嵌入技术和 LSTM 神经网络提取代码特征；最后使用 2 层全连接神经网络预测结果. 总之，Cummins 等人[9] 使用基于深度学习技术的语言模型，将 OpenCL 程序当作文本，从文本的顺序关系中提取特征完成自动优化任务.

Cummins 等人[17] 提出了一种基于文本上下文的嵌入模型. 该模型的改进方法首先使用 LLVM 前端将源代码编译成 IR，从 IR 中抽取数据流和控制流，为文本嵌入提供上下文信息；然后对 LLVM 的 IR 进行预处理，如过滤注释和元数据，并使用特定的字符代替其中的变量及标识符；最后对预处理的 IR 指令构建词汇表，将每条指令映射到嵌入空间以生成指令级的嵌入向量，利用该向量完成程序优化任务.

近年来针对 OpenCL 程序启发式优化[6,18] 的相关工作主要从依赖人工定义特征的方法过渡到依靠深度学习技术自动提取特征的方法. 针对现有研究方法的不足，为了解决使用深度学习技术的代码优化方法不能够提取到准确的代码所特有的特征信息的问题，本文提出了一种基于多关系图神经网络的 OpenCL 程序自动优化启发式方法，将代码转换成多关系代码图[19]，不仅能够获取程序语法语义信息，还能够捕获程序语言的结构信息，然后使用深度学习图网络[20] 对图数据不同关系建模，学习程序图的特征表示并进行分类，完成相关的代码优化任务.

2 系统设计与实现

本节主要介绍基于图网络的 OpenCL 程序自动优化启发式方法 ACCEPT 及其具体实现，该方法的整体框架如图 1 所示.

ACCEPT 以待优化代码的程序图作为输入，输出待优化代码的优化参数，如异构映射平台或线程粗化因子. 为构建 ACCEPT 模型，需要收集大量的训练数据[21]，为此本文借助现有的生成模型 CLgen[22]，合

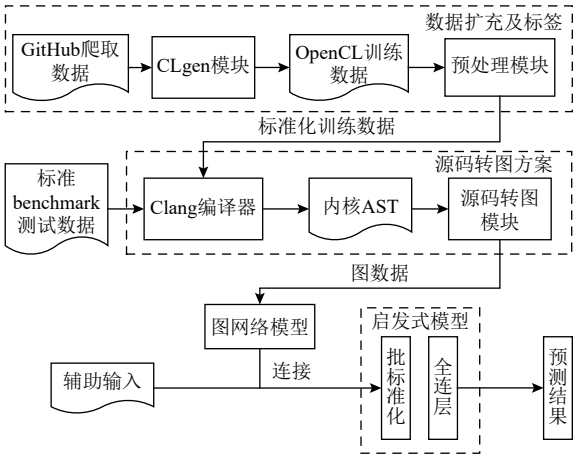

Fig. 1 ACCEPT framwork

图 1 ACCEPT 框架

成训练所需的 OpenCL 内核程序. 然后，对生成的 OpenCL 内核程序进行归一化处理，以消除 OpenCL 内核程序间编程风格差异对训练任务的影响. 接下来，使用 Clang 编译器对预处理后的 OpenCL 内核程序进行构图，分别在源代码和 IR 代码层次上构建对应的 AST 增强图和多关系 IR 图. 最后，利用改进的图网络模型，分别提取 AST 增强图和多关系 IR 图的高维特征表示，并将 2 种特征向量进行拼接，利用决策网络对其进行分类，完成代码优化相关的预测任务. 接下来介绍 ACCEPT 的具体实现细节.

2.1 数据生成与标记

深度学习模型需要大量的训练数据，但对于代码优化相关的任务，当前并没有足够多的公开可用的数据集. 为此，本文利用现有的方法 CLgen[22] 来自动生成 OpenCL 数据. 此外，OpenCL 代码的运行[23] 还需要有对应的主机端代码(host code)来驱动，用于获取平台信息和资源. 为了构建主机端代码，需要对 OpenCL 源代码进行解析，以获取主机端代码所需要的参数信息，进而驱动内核程序的运行，最后统计运行信息完成对训练数据的标记.

1)OpenCL 代码生成. 为生成模型所需要的训练数据，本文从 GitHub 中爬取了 12 351 个 OpenCL 文件，过滤掉无法静态编译与没有实际功能的代码文件后，最终得到 5 200 个 OpenCL 文件作为训练集，用于构建代码生成模型 CLgen.

CLgen 主要由编码、LSTM 神经网络、OpenCL 生成和过滤器 4 个模块组成. 其使用混合编码方案[24]，对于 OpenCL 语言的关键字等使用基于字(token) 的编码方案；对于用户自定义的标识符、函数名等使用基于字符的编码方案，其目的是为了平衡词汇表规模、保留代码语义. LSTM 神经网络模块的作用是提取代码的高维抽象特征，LSTM 循环单元结构能够有效学习代码 token 之间的顺序依赖关系，CLgen 使用 2 层 LSTM 网络，每层网络设置 256 个神经元，训练过程中循环迭代 20 次，学习率设为 0.01. OpenCL 合成模块通过迭代采样的方式生成 OpenCL 内核程序，主要将 LSTM 层输出的向量解码成对应的代码. 本文将生成 OpenCL 内核程序的最大字符长度设置为 5 000. 最后再利用过滤器模块移除静态解析失败、编译失败以及运行超时等无效的 OpenCL 内核程序，最终得到 11 081 条可用的数据.

2)驱动代码生成. 为了构造主机端代码，本文首先对 OpenCL 源代码进行解析，获取驱动相关信息如函数名、参数、工作组及传输值，将其写入固定的主机端代码模板.然后获取设备信息组成上下文信息，设置消息队列存放需要运行的内核代码，因此驱动代码定义了内核程序执行前的通用流程，仅需对运行的内核程序提取必要信息写入驱动代码即可.

3)训练数据标记. 为了获得训练数据对应的标签，本文数据程序分别在 CPU 和 GPU 上运行，以计算出在 OpenCL 内核程序上的实际运行时间与置信区间. 为此，首先在主机端代码模板中设置宏参数，以获取程序执行的时间戳、开始执行时间与结束执行时间，进而获取内核程序的实际执行时间. 然后，将 OpenCL 内核程序分别在 CPU 和 GPU 上重复执行，统计出程序在 CPU 和 GPU 上执行时间的置信区间，最终选择执行时间最短的平台作为内核程序的标签.

2.2 代码预处理

为了降低不同编码风格对深度学习模型的影响，需要对 OpenCL 代码进行归一化处理. 为此，首先应用 CLgen 对 OpenCL 内核程序进行标准化处理，去除与优化任务无关的冗余信息，如删除宏定义指令、删除条件编译与注释、解析 AST 以及统一变量命名风格等. 标准化简化了多关系图网络对源代码的建模任务的复杂性，消除了程序中无关的语义信息对模型的干扰，进而降低模型复杂度. 图 2 展示了预处理前后的对比图. 可以看出，注释、变量或函数的命名与程序员的编码习惯相关，这可能导致对模型学习的干扰.删除注释并不会影响代码的语法语义信息，同时对变量及函数名进行标准化处理，以减小微小语义差异对模型造成的影响.

具体地，首先解析出 OpenCL 内核程序的 AST，遍历 AST 树，根据树中节点的类型删除条件编译、源代码注释. 对编译预处理的宏指令譬如#include 开

```
#define TYPE uint4
__kernel void function(__global uint4 * FirInput, __global uint4 * SecInput) {
//get local memory id
  TYPE id = get_local_id(0);
  TYPE y = 10;
  if (id > y) {
    id = id + 1;
    SecInput[id] = FirInput[id];
  } else {
    y = y × 2;
  }
}
```

预处理前

```
__kernel void A(__global uint4 * a,
__global uint4 * b) {
  uint4 c = get_local_id(0);
  uint4 d = 10;
  if (c > d) {
    c = c + 1;
    b[c] = a[c];
  } else {
    d = d × 2;
  }
}
```

预处理后

Fig. 2 Comparison before and after preprocessing

图 2 预处理前后对比

头的标识符进行逐一替换. 对于变量(VarDecl)及函数声明(FunDecl)的节点, 顺序替换为统一的字符, 如用户自定义的方法名与变量名分别按照先后顺序替换为{*A*, *B*, ⋯, *Z*}与{*a*, *b*, ⋯, *z*}.

2.3 多关系代码图构建方法

为准确地表示程序深度语法、语义及结构特征信息, 本文在源码 AST 的基础上设计了 5 种类型的边关系, 构造了 AST 增强图. 同时, 基于 LLVM 的 IR 设计了 3 种边关系, 构造了多关系 IR 图.

2.3.1 代码图关系说明

在 AST 层面, 本文在 AST 树的基础上构建源代码的语法语义关系, 首先使用 Clang 编译器对 OpenCL 代码进行语法分析, 得到 OpenCL 源码的 AST, 程序图的主体是程序的抽象语法树, 树中的节点由语法结点和语法令牌(Token)组成. 其中语法节点是非叶子节点, 语法 Token 是树中的终端节点, 节点的属性字段(字符串)标识了节点的类型. 然后由根节点开始深度搜索 AST 树, 在遍历过程中, 根据节点的类型构建了 5 种边关系: ASTChild 边、ComputedFrom 边、NextToken 边、GuardedBy 边、Jump 边. 同样地, 在 IR 层面, 通过 LLVM 内置函数可以直接获得数据流和控制流信息, 同时为了记录 IR 节点的顺序, 构建了 IR 顺序流, 如图 3 所示.

表 1 列举了不同的边类型, 接下来将详细介绍 2 种图中各种类型边的含义.

2.3.2 AST 增强图边类型

图 4 展示了 AST 增强图中边的类型, 接下来详细介绍各种边的构建.

1)ASTChild 边. ASTChild 边连接了父节点和子节点, 用于表征代码的抽象结构化信息, 它代表一个程序图的基本结构. 通过 Clang 编译器获得 AST 树的根节点, 基于根节点深度遍历获得 AST 的其他节点, 本文使用 ASTChild 边构建 OpenCL 代码的基本图结构.

```
define kernel void @A(i32, i32) {
  %3 = tail call i64 @get_local_id(i32 0)
  %4 = trunc i64 %3 to i32
  %5 = icmp ugt i32 %4, 10
  br i1 %5, label %6, label %12
  ; <label>:6:
  %7 = add i64 %3, 1
  %8 = and i64 %7, 4294967295
  %9 = getelementptr inbounds %0, i64 %8
  %10 = load i32, %9
  %11 = getelementptr inbounds %1, i64 %8
  store %10, %11
  br label %12
  ; <label>:12:
  ret void
}
```

Fig. 3 LLVM IR instruction

图 3 LLVM IR 指令

Table 1 Program Diagram Connection Relationship

表 1 程序图连接关系

源程序	关系类型
抽象语法树	ASTChild 边、ComputedFrom 边、Jump 边, NextToken 边、GuardedBy 边
中间指令	IR 顺序流、数据流、控制流

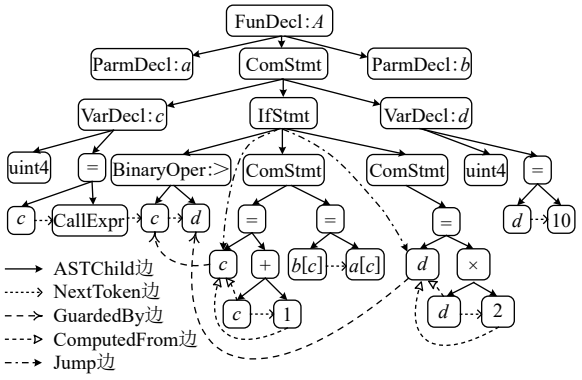

Fig. 4 AST augment graph

图 4 AST 增强图

2)ComputedFrom 边. ComputedFrom 边连接了等

式左右两边变量的赋值关系，用于记录代码的数据流，在内核程序中存在着较多的赋值语句，变量的赋值使用 ComputedFrom 边进行存储.

3）Jump 边. Jump 边记录了跳转语句的跳转方向，用于表征代码的控制流，程序语句的跳转可以控制程序运行的结果，例如常见的循环语句、判断语句等，Jump 边控制着程序执行的路径，是程序特有的重要特征.

4）NextToken 边. NextToken 边连接了 AST 树中同一层的叶子节点，其记录了节点之间的顺序关系，进而丰富了代码的上下文依赖信息.

5）GuardedBy 边. GuardedBy 边将条件语句中的变量与其作用域相同的变量相连，用于表征程序的控制流.对于条件语句而言，条件表达式会影响程序的执行流程，同时，表达式中变量的值也会影响程序控制流的跳转方向.因此，为了捕获复杂的控制依赖关系，本文使用 GuardedBy 边记录条件语句中变量的关系. 例如，对于条件语句 if($x>y$) {…x…} else {…y…}，使用 GuardedBy 边将 if 域中的 x 与表达式 $x>y$ 相连，将 else 域中的 y 与表达式 $x>y$ 相连.

为了进一步捕获控制依赖信息，Jump 边用来连接所有的跳转依赖关系，包含 if, switch…case, while, for, do…while 所形成的跳转关系. 在上述跳转指令中，本文将判断表达式与所有跳转域中的第一个变量相连以捕获跳转关系.

2.3.3 IR 多关系图边类型

1）IR 顺序流边. IR 顺序流边连接了当前指令与下一条指令，用于记录 LLVM IR 指令的顺序关系，同样地，这种顺序关系对于图网络而言也是不可或缺的. 如图 3 为内核程序对应的 IR 指令，IR 顺序流边记录图中每条指令的执行顺序.

2）IR 数据流边. 同样，在 IR 层面中，IR 数据流边连接变量的调用关系，记录了寄存器中变量的赋值过程与走向，是 IR 指令的重要特征.

3）IR 控制流边. IR 控制流连接了跳转指令与下一条指令. 在遇到 br 等跳转指令时会记录指令的跳转方向，与 AST 层面的 Jump 边类似，IR 控制流边记录指令的跳转路径.

同样地，在 IR 层面通过 LLVM 内置函数获得 IR 的数据流和控制流信息并进行向量存储，为了记录 IR 的执行顺序，本文增加了顺序流边，该边有效记录了指令的执行顺序. 图 5 展示了所构建的 IR 多关系图.

2.3.4 多关系代码图的构建

多关系代码图的构建如算法 1 所示.该算法以

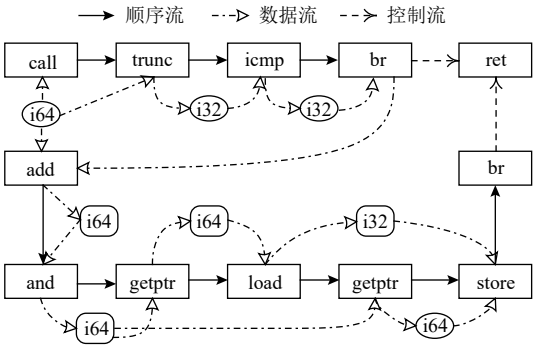

Fig. 5 IR multi-relationship graph

图 5 IR 多关系图

OpenCL 源码为输入，通过分析源码的 AST 与 IR 中的数据控制依赖关系，输出二元组(G_{AST}, G_{IR})，分别存储对应的 AST 增强图 G_{AST} 与多关系 IR 图 G_{IR}.

算法 1. 多关系代码图生成算法.

输入：OpenCL 源代码 *cl*;

输出：源代码对应的 AST 增强图与多关系 IR 图(G_{AST}, G_{IR}).

① *Entry* ← *Clang.parse*(*cl*);
② (G_{AST}, G_{IR}) ← *initialize*();
③ *Cursor* ← *deepSearch*(*Entry*);
④ if *Cursor.kind* = “FunDecl” then
⑤ G_{AST}.*add*(*createASTChild*());
⑥ if *Cursor.hasToken*() then
⑦ G_{AST}.*add*(*createNextToken*());
⑧ end if
⑨ end if
⑩ if *Cursor.kind* = “BinaryOper” then
⑪ G_{AST}.*add*(*createComputedFrom*());
⑫ end if
⑬ if *Cursor.kind* = “ForStmt” or “IfStmt” then
⑭ G_{AST}.*add*(*createJump*());
⑮ end if
⑯ if *Cursor.kind* = “IfStmt” then
⑰ G_{AST}.*add*(*createGuardedBy*());
⑱ end if
⑲ *ir* ← Clang.*compile*(*cl*);
⑳ *firstBlock* ← *findFirstBlock*(*ir*);
㉑ *curInst* ← *seqSearch*(*firstBlock*);
㉒ if *curInst.Name*() ≠ “br” and “indirectbr” and “switch” then
㉓ G_{IR}.*add*(*createIRSeqFlow*(*curInst*, *firstBlock*));
㉔ end if

㉕ if *exitReference*(*curInst.Name*()) then
㉖ G_{IR}.*add*(*createIRDataFlow*(*curInst*, *firstBlock*));
㉗ end if
㉘ if *curInst.Name*() == “br” or “indirectbr” or “switch” then
㉙ G_{IR}.*add*(*createIRControlFlow*(*curInst*, *firstBlock*));
㉚ end if
㉛ return (G_{AST}, G_{IR}).

具体地，为构建 G_{AST}，利用 Clang 编译器提取代码的 AST，从根节点逐层遍历 AST，在遍历 AST 的过程中，利用游标（算法 1 中变量 *Cursor*）的类型来判断 AST 边的类型，并在对应的 AST 节点位置分别添加 ASTChild 边、NextToken 边、ComputedFrom 边、GuardedBy 边以及 Jump 边（如算法 1 中行④~⑱），进而完成 G_{AST} 的构建. 另一方面，为构建 IR 多关系图，使用 Clang 编译器将 OpenCL 内核程序转换成 LLVM 的 IR，利用程序分析技术解析 IR 并定位到第 1 个基本块，然后顺序遍历基本块中的指令（如算法 1 中行⑲~㉑），在遍历指令过程中，向 G_{IR} 中依次添加 IR 顺序流、数据流以及控制流（如算法 1 中行㉒~㉚）.最终，算法返回二元组（G_{AST}, G_{IR}）（如算法 1 中行㉛）.

为将所构建的代码图表示成深度学习模型能够处理的数据形式，ACCEPT 使用邻接矩阵表示图的节点间的连接关系，即当 2 个节点之间存在边时，则将邻接矩阵中对应位置置为 1，否则置为 0. 由于 G_{AST} 存在 5 种类型的边，G_{IR} 存在 3 种类型的边，每种类型的边都需要使用 1 个邻接矩阵表示，因此共需要 8 个邻接矩阵表示（G_{AST}, G_{IR}）.同时，为了丰富图中节点之间的连接信息，通过转置邻接矩阵，为图添加反向边，增强图神经网络对节点间关系的学习能力.

2.3.5 多关系代码图节点特征提取

ACCEPT 使用 Word2Vec[25] 对代码图中每个节点进行编码，其能够将单个词映射到连续向量空间，这有助于学习代码的语法与语义关系.例如，声明类的节点更容易被映射到相近的向量空间中，模型可以学习 if-else 语句的顺序关系等. 为了保存节点间丰富的信息，本文将每个 Token 和单词映射到 100 维固定长度的嵌入向量.

为对代码图中的节点进行编码，ACCEPT 利用深度优先遍历，首先从根节点遍历代码图，定位节点类型，如 AST 增强图中的节点类型包括根节点（TranslationUnit）、参数说明（ParmDecl）等，多关系 IR 图中节点类型主要包括 call, icmp, load 等. 然后利用 Skip-Gram 嵌入算法学习节点类型和节点上下文之间的关系，该嵌入方法根据节点类型出现的频率构建词汇表，然后根据词汇表中节点类型抽取对应的特征向量，以构建图节点的特征矩阵. 此外，为尽可能减小词汇表大小，本文对变量、函数名以及常数采用 Token 级别的编码方式.

2.4 多关系图神经网络建模

为了能够准确地对多关系代码图建模，并提取 OpenCL 代码的深度结构和语义特征，本文提出了一种多关系图深度神经网络模型，该网络在关系图卷积网络（relational graph convolutional network, RGCN）的基础上，通过改进其输入结构、邻域聚合与节点信息传播策略，使其能够处理具有多种边类型的多关系代码图，同时可以针对不同类型的边进行特征提取，以便学习更深层的代码特征，提取 OpenCL 代码的高维抽象特征向量. 为此，多关系图网络模型以邻接矩阵与节点特征矩阵为输入，利用邻域聚合算法，按照边类型对图中节点信息进行更新，最后使用图嵌入策略完成一轮特征提取. 为实现多跳节点之间的特征信息交互，需要在图网络中进行多轮迭代特征提取，以完成对多关系中所有的节点特征的更新，输出 OpenCL 代码的高维抽象特征. 接下来，将详细介绍邻域聚合与图嵌入的具体实现.

2.4.1 邻域聚合

为了学习程序的高维抽象表示，在图网络中需要递归更新图中每个节点的特征表示，这通过对邻域节点的特征进行聚合操作来实现，以学习节点邻域信息，从而找到每个节点更加准确的抽象表示. 邻域聚合表达式为

$$\boldsymbol{l} := \sum_{u \in N_{(v)}} (\boldsymbol{\mu}_u^t), \tag{1}$$

其中，$\boldsymbol{l}$为邻域聚合的结果，节点u是当前节点v的邻域，$N_{(v)}$是节点v的邻域集，$\boldsymbol{\mu}_u^t$是节点u第t层的更新特征. 邻域信息的聚合能够有效地传递节点的信息，为后续的图嵌入过程提供信息交互的支持.

由 2.3.5 节可知，节点特征的初始化任务由 Word2-Vec 模型得到. 每一次迭代都将更新节点当前状态，并将其作为消息发送给图中的所有邻域节点来交换信息. 当每个顶点完成消息聚合后，当前节点状态将被用于下一次迭代的邻域聚合过程. 重复此迭代更新步骤直至达到固定的迭代次数. 当完成固定迭代次数的更新后，网络将输出学习到的节点特征矩阵. 本文将节点特征矩阵按行展开聚合，组成新的一维

特征向量, 即为 OpenCL 程序的高维抽象特征向量.

2.4.2 图嵌入

图嵌入旨在更新节点的特征信息, 完成节点间的信息传递. 接下来以 AST 增强图为例, 详细说明图嵌入具体过程.

假设 OpenCL 程序对应的 AST 增强图表示为 $G=\langle V,E\rangle$, 其中 V 为节点的集合, E 为边的集合, $E=\{\boldsymbol{E}_{\mathrm{AST}},\boldsymbol{E}_{\mathrm{NT}},\boldsymbol{E}_{\mathrm{GB}},\boldsymbol{E}_{\mathrm{Jump}},\boldsymbol{E}_{\mathrm{CF}}\}$, 集合中的边分别对应 ASTChild 边、NextToken 边、GuardedBy 边、Jump 边以及 ComputedFrom 边. 首先根据边类型进行邻域聚合, 假设 Word2Vec 初始化的节点特征矩阵为 $\boldsymbol{X}$, 更新特征为 $\boldsymbol{\mu}$, 则 $\boldsymbol{\mu}$ 的计算公式为

$$\boldsymbol{\mu}=(\boldsymbol{E}_{\mathrm{AST}}+\boldsymbol{E}_{\mathrm{NT}}+\boldsymbol{E}_{\mathrm{GB}}+\boldsymbol{E}_{\mathrm{Jump}}+\boldsymbol{E}_{\mathrm{CF}})\boldsymbol{X}. \tag{2}$$

特征矩阵 $\boldsymbol{X}$ 与每种边构建的图进行消息传递, 生成每种边独有的新的节点特征, 最终将 5 种特征矩阵进行聚合, 完成边类型的特征提取. 类似地, IR 多关系图的嵌入同样经过 AST 增强图嵌入流程完成 3 种边类型的特征提取. 具体的嵌入过程如图 6 所示.

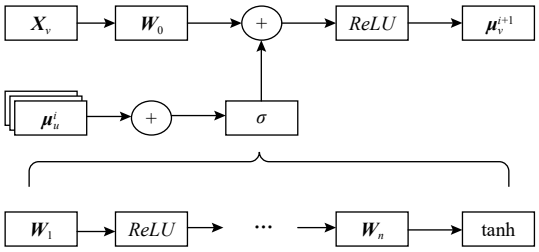

Fig. 6 Graph embedding process

图 6 图嵌入过程图

首先将节点特征向量 $\boldsymbol{\mu}$ 以及二元组 $(G_{\mathrm{AST}},G_{\mathrm{IR}})$ 输入嵌入层, 假设当前节点 v 的特征向量为 $\boldsymbol{X}_v$, 节点 u 是当前节点 v 的邻域, 为了学习每个节点的嵌入向量, 过程为: 设置当前节点特征向量 $\boldsymbol{X}_v$ 的初始化权重矩阵 $\boldsymbol{W}_0$, 第 i 轮更新的邻域节点集聚合成 $\boldsymbol{\mu}_u^i$, 之后进入 n 轮嵌入过程, 每一轮迭代嵌入使用修正线性单元(rectified linear unit)激活函数, 记为 $ReLU$, 用于过滤噪声, 经过 n 轮嵌入后使用 tanh 激活函数, 与当前节点特征叠加再使用 $ReLU$ 激活后获得当前节点第 i+1 轮的节点更新, 完成更新后获得 OpenCL 代码的特征矩阵, 最后将特征矩阵逐行展开并聚合为一维的特征向量, 该特征向量即为代码的高维特征表示. 上述迭代更新嵌入过程表示为:

$$\boldsymbol{\mu}_v^{t+1}:=\tanh\left(\boldsymbol{W}_0\boldsymbol{X}_v+\sigma\left(\sum_{u\in N_{(v)}}(\boldsymbol{\mu}_u^t)\right)\right), \tag{3}$$

$$\sigma(\boldsymbol{l}):=\boldsymbol{W}_1 ReLU(\boldsymbol{W}_2 ReLU(\cdots(\boldsymbol{W}_n\boldsymbol{l})). \tag{4}$$

式(3)中 $\boldsymbol{\mu}_v^{t+1}$ 是节点 v 在第 t+1 层的更新状态, 节点 u 是当前节点 v 的邻域, $N_{(v)}$ 是节点 v 的邻域集, 其中初始顶点状态 $\boldsymbol{\mu}_v^0$ 使用 Word2Vec 嵌入方法构建. 随着迭代次数的增加, 节点特征能够传递到图网络中距离当前节点多跳的其他节点. 总体而言, 本文根据边类型进行邻域聚合, 通过多轮嵌入学习邻域特征, 以完成整个图的节点更新过程.

2.5 决策网络

决策网络的目的是预测代码优化参数, 如异构设备映射或线程粗化因子, 其输入是多关系图神经网络. 图 7 展示了决策网络的架构, 可以看出, 决策网络由向量拼接层、批归一化层以及全连接层组成.

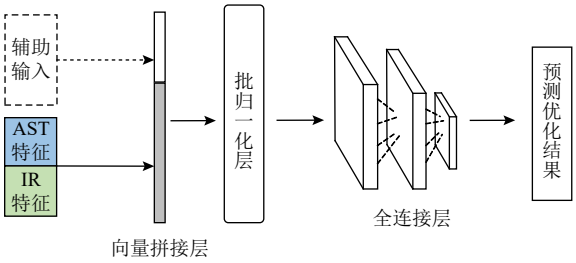

Fig. 7 The structure of decision network

图 7 决策网络结构

向量拼接层将多关系图神经网络层学习到的特征向量(AST 增强图特征向量与 IR 多关系图特征向量)以及辅助输入聚合成固定长度的一维特征向量. 辅助输入是程序的动态运行特征, 如程序运行内存大小、栈内存大小等. 由于辅助输入的值并不在 [0, 1] 范围内, 使得辅助输入与嵌入向量具有不同的量纲, 将导致训练时间过长、模型难以收敛. 因此使用归一化层, 将拼接好的特征向量的值标准化在 [0, 1] 范围内. 标准化的特征向量能够捕获代码的诸多特征, 例如语义相似性、控制流、依赖流等. 最后标准化的特征向量将被送入全连接层神经网络进行分类, 完成优化预测任务.

2.6 模型训练

ACCEPT 使用反向传播机制协同训练图网络和决策网络. 与大多数机器学习训练方法不同, ACCEPT 不必手动从程序中构建和提取特征, 只需输入源代码, 就能实现端到端的优化预测. 训练过程中, 本文使用随机梯度下降(stochastic gradient descent, SGD)算法, 并结合 Adam 优化器[26]加速模型的收敛速度, 使用标准交叉熵损失函数作为目标函数, 使用常用于分类任务的 Sigmoid 和 Softmax 激活函数. 目标函数定义为

$$\varsigma_G = -\sum_{i=1}^{N} y_{i,c}\,\mathrm{lb}(p_i, c). \quad (5)$$

其中N是分类的数目，在异构设备映射任务中N=2，在线程粗化任务中N=6；$y_{i,c}$的取值为 0 或 1，表示标签c是否是训练样本i的正确分类；p_i是样本i属于标签c的预测概率. 随机梯度下降将寻找适合的参数使得最小化损失函数，即减小预测值与期望值的对数差异进而完成模型的训练.

3 实验设置

3.1 实验平台

本文采用 TensorFlow 深度学习框架构建多关系图神经网络模型.所有实验均运行在 Ubuntu 16.04 操作系统上，其内存为 12GB, CPU 处理器为 Intel Xeon E5-2 667, 编程语言为 Python 3.6.

3.2 数据集构成

异构设备映射任务采用了 DeepTune[9] 提供的数据集，该数据集从 7 个 benchmark 集（Parboil[27], NVIDIA CUDA SDK[28], AMD SDK[29], SHOC[30], NPB[31], Rodinia[32], PolyBench[33]）中抽取 256 个 OpenCL 内核程序，通过改变辅助输入将其扩充至 680 条数据.

线程粗化任务采用了 DeepTune[9] 给定的标记数据，该数据集从 3 个 benchmark 集（Parboil[27], NVIDIA CUDA SDK[28], AMD SDK[29]）中抽取了 17 个 OpenCL 内核程序. 其中 Parboil 4 个，NVIDIA CUDA SDK 3 个，AMD SDK 10 个.

3.3 评估指标

为评估 ACCEPT 对异构平台映射与线程粗化因子预测的性能，选取准确率（*Accuracy*）和加速比（*Speedup*）作为评估指标. 准确率指模型预测正确的个数与测试数据总数的比值. 加速比指同一程序在目标设备上的运行时间与在静态映射上的执行时间的比值，本文中静态映射为 AMD 平台.准确率计算为

$$Accuracy = \frac{1}{m}\sum_{i=1}^{m} true\_label_i, \quad (6)$$

其中 m 为测试集的个数，$true_label_i$为第 i 个测试数据的预测结果. 加速比计算为

$$Speedup = \frac{1}{m}\sum_{i=1}^{m} (static\_time_i / predict\_time_i), \quad (7)$$

其中 m 为测试集的个数，$static_time_i$为第 i 个测试数据对应的静态映射时间，$predict_time_i$为模型第 i 个测试数据预测最优设备上的执行时间. 加速比的结果由对所有测试数据的静态映射时间（在静态映射上的执行时间）与预测时间的比值求和再取平均得到，实验中加速比的值为每种 benchmark 下的几何平均值.

为验证 ACCEPT 的有效性，在所有实验中本文采用 10 折交叉验证，实验的准确率与加速比为 10 次实验的平均值，从而减小噪声数据对实验结果的影响.

4 实验结果与分析

4.1 实验概况

本文分别在异构设备映射与线程粗化因子预测 2 个代码优化任务上评估 ACCPET 的有效性.同时，分别与现有的经典代码优化方法进行对比实验，充分验证 ACCEPT 的有效性，如表 2 所示.

Table 2 Comparison of Machine Learning Methods

表 2 机器学习方法对比

方法	代码表示	模型
Grewe 等人[4]	手工特征	决策树
DeepTune[9]	源码 Token 序列	LSTM
DeepTune IR[17]	LLVM IR Token	LSTM
Inst2Vec[34]	LLVM IR Token	LSTM
GNN-AST[35]	AST	GNN
GNN-CDFG[35]	CDFG	GNN
Magni 等人[15]	手工特征	神经网络

为保证对比实验的公平性，本文使用自动化模型调优工具 AutoML（automated machine learning）对所有未公布训练模型参数（如学习率、Batch Size、Epoch 等参数）的待比较代码优化方法进行训练调优，使其均能够达到最优的实验结果.在对比实验模型训练过程中，均统一采用 10 折交叉验证，在每一轮交叉验证中，若损失函数在连续 20 个训练轮次内无下降，或者在验证集上达到 99% 以上的准确率，则终止训练.

4.2 异构设备映射任务

OpenCL 是一个面向异构并行程序编写代码的框架，使用 OpenCL 编写的程序可以透明地在多个设备上运行，例如 GPU 或 CPU 等异构设备. 异构设备映射任务的目的是预测 OpenCL 程序最佳的执行设备，以提高程序执行的加速比. 故模型预测结果的准确性是提高程序加速比的前提条件，也是异构设备映射任务评估的主要指标.

4.2.1 实验方法

该实验所用的异构平台分别为 AMD Tahiti 7970,

NVIDIA GTX 970. 因模型训练需要足够的训练数据, 而现有公开可用的数据并不充足, 因此在准确率实验评估中, 本文使用 OpenCL 程序生成器 CLgen 生成了 11 081 个有效且可编译的 OpenCL 内核程序并在 3.2 GHz 6 核 Xeon E5-2667CPU 和 NVIDIA Titan XP GPU 的平台上进行标记, 将标记的数据作为训练集, 测试集使用 DeepTune[7] 方法中所公开的 680 个标准的 OpenCL 内核程序.

4.2.2 对比实验

表 2 前 6 种代码优化方法为该实验的对比对象, 包括: Grewe 等人[4], 其使用手工特征和决策树算法构建代码优化模型; DeepTune[9] 与 DeepTune IR[17] 分别使用源代码的 Token 序列和 LLVM IR 序列做代码表征, 并都使用了 LSTM 模型; Inst2Vec[34] 使用 LLVM IR 生成的图作为代码表征和 LSTM 模型; GNN-AST 与 GNN-CDFG 分别使用 AST 图以及 CDFG 图提取代码特征, 这 2 种方法使用了基于图网络变体模型. 此外, 对于异构设备映射任务, 该实验保留了原有方法的设置, 使用程序执行的动态信息(如工作组大小和数据大小)作为模型的辅助输入.

4.2.3 实验结果与分析

图 8 展示了在不同平台下 ACCEPT 与其他方法的加速比对比结果. 由图 8 可知, 无论是在早期的 GPU 平台如 NVIDIA GTX 970, 还是在最新的 NVIDIA Titan XP 上, ACCEPT 优化后的加速比均优于其他经典的代码优化方法. 在 AMD Tahiti 7970 平台上, ACCEPT 取得了平均 3.18×的加速比; 在 NVIDIA Titan XP 平台上, ACCEPT 的加速比能够达到 2.6×, 比其他方法平均提高至少 25%. 虽然 ACCEPT 在 NVIDIA GTX 970 平台上的性能提升相对较小, 为 1.31×, 但其总体加速比相比其他方法最高.

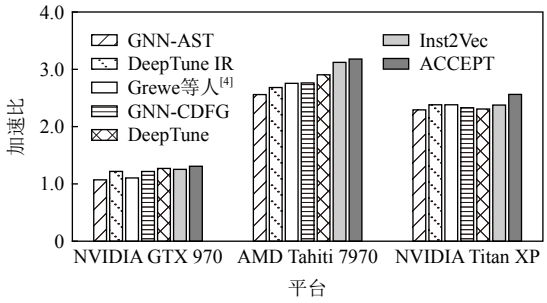

Fig. 8 Comparison results of speedup

图 8 加速比对比结果

由实验结果推断, ACCEPT 能够利用构建的多关系图和改进的多关系图神经网络, 有效地提取代码的深度结构和语义特征. 具体表现在: 仅通过构建 OpenCL 代码的抽象语法特征图(如 GNN-AST 方法), 或者仅构建代码数据依赖图(如 GNN-CDFG 方法), 所取得的加速比都没有 ACCEPT 高. 这说明, 本文所构建的 AST 增强图和 IR 多关系图更有助于提取代码的高维抽象特征.

进一步地, 为了深入分析 ACCEPT 的性能, 本文统计了异构映射的准确率. 图 9 展示了在不同平台下准确率对比评估结果, 与 DeepTune 和 Inst2Vec 相比, ACCEPT 能够将准确率从 82% 提高到 88.7%, 并且 ACCEPT 能够以最高的概率预测出最佳的异构映射平台. 直观上, 预测准确率越高, 其相对加速比就越高, 因此由图 9 可以看出, 准确率总体趋势和加速比一致, 这也是 ACCEPT 能够取得最高加速比的原因.

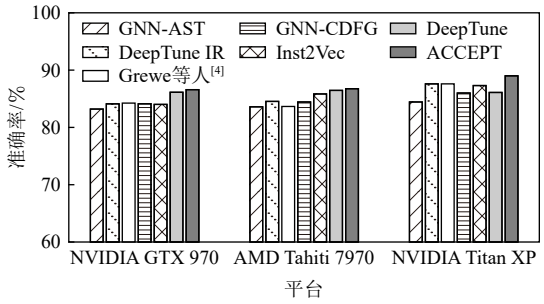

Fig. 9 Comparison results of accuracy

图 9 准确率对比结果

4.3 线程粗化任务

线程粗化任务的对象是并行执行的程序, 其目的是选择待执行程序的最佳线程粗化因子, 即需要并行执行的线程数量, 从而达到系统开销与程序并行执行效率的最佳平衡点. 该实验具体的量化指标是加速比, 程序在某个线程粗化因子下的加速比越高, 说明其执行效率越高. 与现有工作相同, 该评估实验中分别设置 6 种粗化因子, 分别为 1, 2, 4, 8, 16, 32, 其中粗化因子为 1 表示不进行线程粗化, 其对应的加速比值为基准值.

4.3.1 实验方法

为验证 ACCEPT 在线程粗化任务上的有效性, 该实验分别在 AMD Radeon HD 5900, AMD Tahiti 7970, NVIDIA GTX 480, NVIDIA Tesla K20c 这 4 个平台上进行. 由于线程粗化任务中仅有公开可用的 17 个 OpenCL 内核程序, 为了训练 ACCEPT, 该实验使用 DeepTune[7]公开的 680 条 OpenCL 内核程序数据做训练, 然后再利用迁移学习技术, 使用留一法评估模型. 具体地, 将 17 条数据分成 17 份, 并将每份数据按照训练与测试 16∶1 的比例进行分割, 每次迁移时, 使用 16 条数据作为训练集, 剩余 1 条作测试, 因此在

每个平台上的评估实验最终会训练出 17 个模型，将 17 个模型加速比的几何平均值作为该平台上的评估结果.

4.3.2 对比实验

为充分评估 ACCEPT 的有效性，该实验分别与 DeepTune, Inst2Vec, GNN-CDFG, GNN-AST 以及文献 [15] 中的方法共计 5 种方法进行比较. 其中文献 [15] 使用人工特征提取的方法，将 OpenCL 内核程序所占用内存及指令数等静态信息作为特征，在特征中使用主成分分析法获取高维特征表示，然后使用基于二元决策树的方法预测线程粗化因子. 此外，该实验还比对了在不同平台下线程粗化任务的最优加速比，即如图 10 中 Oracle.

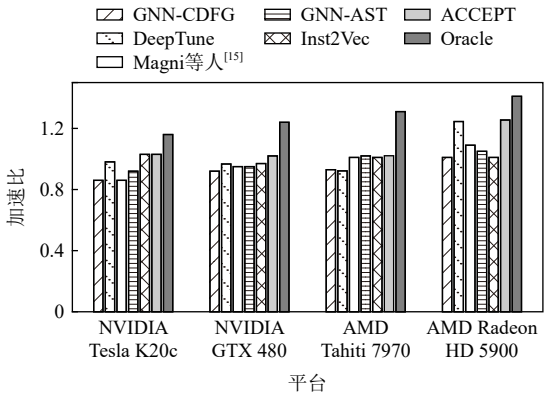

Fig. 10 Comparison of speedup

图 10 加速比对比

4.3.3 实验结果与分析

图 10 展示了在不同平台下各种方法的对比实验结果. 由图 10 可以看出，ACCEPT 整体上的性能表现优于现有的代码优化方法. 例如，在 AMD Radeon HD 5900 平台上，ACCEPT 能将加速比提高到 1.26×，其提升性能略高于 DeepTune 的加速比，并且远高于其他方法. 在 NVIDIA GTX 480 平台上，ACCEPT 是唯一高于基准加速比(即基准为 1)的方法，其加速比为 1.02×，而其他方法均低于基准加速比. 在其他方法中，DeepTune 与 GNN-CDFG 在 NVIDIA Tesla K20c, NVIDIA GTX 480, AMD Tahiti 7970 这 3 个平台上的加速比均低于基准值，这说明使用人工定义的特征或使用控制数据依赖特征并不能够提取到代码的深度特征信息. 虽然 DeepTune 的性能与 ACCEPT 接近，但其微小的差距也说明了利用 LSTM 并不能够有效地对代码进行特征建模，而 ACCEPT 使用多关系图网络，能够有效地提取代码图中的结构与语义信息.

总的来说，与其他方法相比，ACCEPT 在 4 个异构平台上均有性能的提升，尤其在 AMD Radeon HD 5900 上的性能提升最显著. 同时从实验结果来看，ACCEPT 可以在训练语料库较小时有效地支持迁移学习. 由此可以看出，ACCEPT 不仅在异构映射任务上取得了最好的结果，而且在线程粗化任务上也取得了较好的结果，充分说明了 ACCEPT 对于代码特征的提取能力，是一种有效的代码优化启发式方法.

4.4 数据生成质量分析

在异构映射任务中，本文使用生成的 OpenCL 程序内核数据训练 ACCEPT 模型. 程序生成的质量将直接影响模型的预测准确率，为评估程序生成的质量，本实验分别对所生成的 OpenCL 程序的可用性及编码质量进行评估.

4.4.1 OpenCL 程序可用性评估

为验证所生成 OpenCL 内核程序的可用性，该评估实验分别在 Intel Xeon E5-2667 和 NVIDIA Titan XP 这 2 个平台上进行，实验平台的具体参数如表 3 所示. 对于生成数据，通过统计其在编译过程中出现编译失败、编译异常、编译超时以及运行过程中出现运行错误和超时的内核程序的数量来评估生成的 OpenCL 内核程序的可用性.该实验的数据为 20 000 个生成的 OpenCL 内核程序.

Table 3 Platform Arguments for Evaluation of Program Generation Quality

表 3 程序生成质量评估平台具体配置信息

平台	频率/GHz	内存/GB	驱动型号
Intel Xeon E5-2667	3.2	102	Intel 18.1.0.0920
NVIDIA Titan XP	1.6	12	NVIDIA 410.72

实验结果如表 4 所示，在 Intel Xeon E5-2667 设备上，编译阶段失败、异常和超时的内核程序数量分别为 234, 78, 44, 运行阶段失败与超时的数量分别为 744 和 127, 共计有 1 227 个低质量的 OpenCL 内核程序，即大约有 93.9% 的内核程序能够通过编译及运行；在 NVIDIA Titan XP 平台上，大约有 1 381 个内核程序没有通过编译及运行. 这 2 个平台下最终得到的 OpenCL 内核程序数量不一致，这是因为不同平台下

Table 4 Results of Program Quality Evaluation

表 4 程序质量评估结果

平台	编译阶段内核程序数量			运行阶段内核程序数量		通过的内核程序数量
	失败	异常	超时	失败	超时	
Intel Xeon E5-2667	234	78	44	744	127	18 773
NVIDIA Titan XP	231	158	33	786	173	18 619
总计	465	236	77	1 530	300	37 392

所使用的硬件驱动不同，并且不同平台的编译及运行情况也不同.总之，在2个平台上，平均能够生成93%以上的可用的OpenCL内核程序.

4.4.2 OpenCL程序双盲实验评估

生成数据与真实数据是否符合同一个概率分布，关系到训练出的模型是否能够实用.即若生成数据与真实数据极为相似，那么使用生成数据训练出的模型能够应用于真实数据.为验证生成与真实OpenCL[①]内核程序的相似性，本文设计了双盲验证实验.该实验中，选择了具有OpenCL编码经验的30位程序员作为志愿者，共28位硕士和2位博士，其中13位为女性，17位为男性.实验过程中，分别随机抽取真实与生成的内核程序各15个，并进行随机打乱，让每一位志愿者进行人工标记.为保证实验的公平性，实验前对测试的内核程序进行标准化操作，其目的是为了减少注释以及函数名所带来的标记干扰；同时，在测试之前，分别给每位志愿者展示30个标准化后的生成与真实的OpenCL内核程序.实验过程中不限标记时间，直至志愿者将所有的30个内核程序标记完成.该实验分别统计了标记正确的生成与真实OpenCL内核程序的志愿者人数，实验结果如表5所示.

Table 5 Results of Double-blind Experiment

表5 双盲实验结果

正确标记OpenCL的数量范围	正确标记生成代码的人数	正确标记真实代码的人数
0~4	6	3
5~8	21	21
9~15	3	6

仅有少数志愿者能够正确标记出大部分生成与真实的OpenCL内核程序，而大部分志愿者能够正确标记生成与真实OpenCL内核程序的数量不超过8个.由此可以看出，标记生成代码的准确率为46.6%，而标记真实代码的准确率为53.4%，平均准确率接近50%，这意味着人工标记过程中，其准确率与随机猜测的概率基本相同，说明了生成与真实的OpenCL内核程序相似度比较高，这也是使用生成数据训练模型能够以较高的准确率预测出真实OpenCL代码适合的异构平台的原因.

4.5 模型分析评估实验

为进一步验证本文所提出的模型与构图方法的有效性，构建最优的模型与构图方法，本文分别评估了神经网络层数、不同构图方法以及不同图神经网络对实验结果的影响.该评估实验以异构映射任务为例，使用DeepTune[9]提供的数据集[②]，在AMD Tahiti 7970平台上进行.

4.5.1 网络层数的影响

神经网络的层数是网络模型的重要参数，设置合理数量的中间层，能够增强模型的表征与特征抽取能力.相反，若设置的中间层数太少，将使网络难以模拟出具有复杂分类空间的模型；而中间层过多，将使模型太过复杂导致难以收敛.

为选取合适的网络层数，实验中除构建不同层数的多关系图神经网络外，其他实验设置均与4.2节中保持一致.实验结果如表6所示，可以看出，该实验结果符合深度学习模型的一般规律，即随着网络层数的增加，模型的表征能力与特征提取能力不断增强，预测准确率由79.4%增长到82.0%，而随着层数的继续增加，模型变得越来越复杂，极易导致过拟合，准确率逐渐降低.该实验表明，在异构映射任务中，当网络层数设置为5时，模型表现最优.同样地，因不同的任务具有不同的复杂度与分类空间，本文使用上述方法确定线程粗化因子预测任务的最佳网络层数.

Table 6 Comparison of Accuracy with Different Numbers of Network Layers

表6 不同网络层数的准确率对比

网络层数	准确率/%
1	76.4
3	79.1
5	82.0
7	77.9
9	76.4

4.5.2 构图方式的影响

ACCEPT不仅依靠表征能力强的多关系图网络模型，还需要正确的、能够表征代码深度结构与语义特征的构图方式将OpenCL代码转换成多关系程序图.为验证构图方式对ACCEPT的影响，该实验保持多关系图网络结构不变，通过改变构图方式进行评估.为此，本文分别在源码与IR代码的基础上，向AST中加入不同类型的边，构建出4种程序图，分别为：ACCEPT-vanilla-AST，表示仅构建源代码的AST图；

① 真实OpenCL代码是从真实项目中收集的完整代码片段.

② 需要说明的是，该实验仅在680条数据集上进行，因此ACCEPT的准确率与4.2节中的数值不同.

ACCEPT-AST 与 ACCEPT-CDFG, 分别表示仅构建 AST 增强图与 IR 多关系图; ACCEPT-8, 表示将 AST 增强图与 IR 多关系图中的每种边类型单独构图, 然后将所构建的 8 种图的特征向量进行拼接. 图 11 展示了不同构图方式之间的对比关系图.

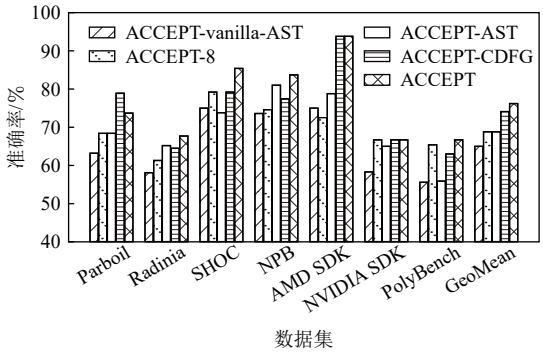

Fig. 11 Accuracy of different graph construction

图 11 不同构图方法的准确率

由图 11 可知, 在除 SHOC 外的 7 个数据集上, ACCEPT-AST 的平均准确率比 ACCEPT-vanilla-AST 准确率高, 这说明给 AST 增加额外的 5 种数据控制依赖关系边, 能够更加精准地提取代码的语法语义信息.ACCEPT-8 虽然包含了所有的 8 种边类型, 但其在每个 benchmark 上的准确率均不高于 75%, 这说明仅仅将 8 种图的特征进行简单地拼接, 而不考虑不同边类型所携带的特征信息, 并不能将节点的信息传递给其他的邻居节点, 导致模型表征能力并不强. 此外, 从 ACCEPT-AST 与 ACCEPT-CDFG 比较结果来看, ACCEPT-CDFG在大部分数据集上实现的准确率高于 ACCEPT-AST, 这很大程度上是因为 CDFG 在 IR 代码上进行构图, IR指令更加简洁, 并且能够突出底层的执行逻辑. 需要说明的是, 在 Parboil[29] 数据集上, ACCEPT-CDFG 的准确率接近 80%, 大于 ACCEPT 的 74% 的准确率, 分析发现, Parboil 数据集中的算法具有丰富的控制流信息, 使得 ACCEPT-CDFG 更能聚焦代码特征信息.

从图 11 的实验结果来看, 与其他 5 种不同的构图方法比, ACCEPT 取得了最高的准确率, 这表明通过将源代码的增强 AST 与 IR 代码的多关系控制依赖图相结合, 能够更加精准地提取到代码的深度结构和语义特征.

4.5.3 图网络模型的影响

为验证本文所构建的多关系图神经网络的有效性, 该实验中将 ACCEPT 分别与现有的图神经网络模型进行比较.本文分别选择利用门控信息来传递信息的 GGNN[36](gated graph sequence neural networks)、基于线性特征调制的 GNN_FILM[37](graph neural networks with feature-wise linear modulation)、基于多边关系建模的图卷积神经网络 RGCN[38](relational graph convolutional networks), 及其 RGCN 的变体网络 GNN_Edge_MLP[39]. 实验过程中, 由于不同的网络模型需要输入不同的图结构, 因此本文将 AST 增强图与 IR 多关系图转换成每个图网络模型适合处理的格式. 对比实验结果如图 12 所示, 其中 GeoMean 为 5 种模型在 7 种数据集的几何平均值.

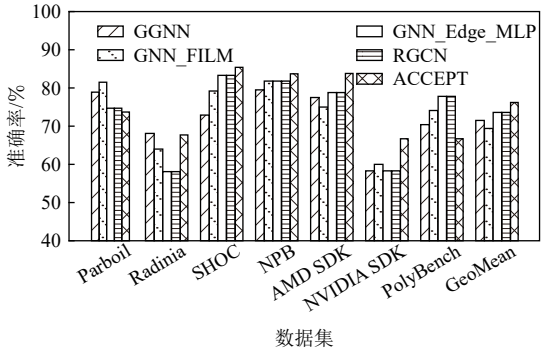

Fig. 12 Accuracy of different graph neural networks

图 12 不同图神经网络的准确率

总体上, 与其他 4 种经典的图网络模型相比, ACCEPT 取得了最好的结果, 其平均准确率达到了 77.5%, 其次为 RGCN 与 GNN_Edge_MLP, 平均准确率均为 74%. 尽管 RGCN 及 GNN_Edge_MLP 支持对多边建模, 但并不适合对异构代码图进行建模, 而 AST 和 CDFG 是 2 种具有不同特征信息的异构图, 这将在很大程度上降低 RGCN 与 GNN_Edge_MLP 这 2 种模型的特征提取与建模能力. 由于 GGNN 与 GNN_FILM 更适合使用门控信息或线性调制来提取特征, 因此对关系程序图的建模能力较弱, 所取得的准确率最低.

5 总 结

当前针对 OpenCL 程序的优化方法需要过多的人为参与, 难以有效提取程序的深度结构与语法语义特征信息, 进而无法对代码进行有效地建模, 使得程序优化效果并不明显. 针对上述问题, 本文提出了 ACCEPT, 一种基于多关系图神经网络的 OpenCL 程序自动优化启发式方法, 旨在无需手工提取特征的情况下, 实现端到端的自动化优化, 提高程序的执行效率. 与传统的基于深度学习的优化方法相比, ACCEPT 将源代码转换为代码特有的多关系程序图, 以表征代码的深度结构与语义特征信息. 为此,

ACCEPT 分别在源代码和 IR 代码上对目标程序进行构图, 通过添加 8 种类型的数据控制依赖边, 保留了代码的语法及语义信息. 同时, 为了处理所构建的多关系程序图, ACCEPT 改进了现有的图形神经网络模型, 通过融合程序的控制流、数据流以及抽象语法树所表征的代码的结构与语义特征信息, 进而生成高维特征向量, 最终输入到决策网络中进行分类, 完成预测优化因子任务.

为验证模型性能, 本文分别在异构映射与线程粗化因子预测 2 个代码优化任务上, 进行了详细的对比分析实验, 与现有的经典代码优化方法相比, ACCEPT 在准确率、加速比上均提升明显, 在异构设备映射任务中, 加速比可提高 7.6%. 在线程粗化任务中, 加速比可提高 5.2%. 为充分地说明本文所提多关系程序图与多关系图神经网络的有效性, 分别在神经网络层数、构图方式以及网络模型上进行了详细的模型分析及对比实验, 实验结果进一步表明了 ACCEPT 的有效性, 说明 ACCEPT 在代码优化领域具有一定的学术价值和应用前景.

作者贡献声明: 叶贵鑫提出了文章思路和实现方案并负责撰写与修改论文; 张宇翔负责系统设计与实现; 张成负责数据收集与模型调优、数据分析; 赵佳棋负责数据预处理; 王焕廷负责代码图多种边关系的构建. 叶贵鑫和张宇翔为共同第一作者.

参 考 文 献

[1] Dietze R, Rünger G. The search-based scheduling algorithm HP* for parallel tasks on heterogeneous platforms[J]. Concurrency and Computation: Practice and Experience, 2020, 32(21): e5898

[2] Nvidia.OpenCL[EB/OL].[2021-09-10].https://developer.nvidia.com/opencl

[3] Pennycook S J, Hammond S D, Wright S A, et al. An investigation of the performance portability of OpenCL[J]. Journal of Parallel and Distributed Computing, 2013, 73(11): 1439−1450

[4] Grewe D, Wang Zheng, O'Boyle M F P. Portable mapping of data parallel programs to OpenCL for heterogeneous systems[C/OL]//Proc of the 11th IEEE/ACM Int Symp on Code Generation and Optimization (CGO). Piscataway, NJ: IEEE, 2013 [2021-09-10]. https://ieeexplore.ieee.org/document/6494993

[5] Balasalle J, Lopez M A, Rutherford M J. Optimizing Memory Access Patterns for Cellular Automata on GPUs[M]//GPU Computing Gems Jade Edition. San Francisco, CA : Morgan Kaufmann, 2012: 67 − 75

[6] Shen Yuan, Yan Hanbing, Xia Chunhe, et al. A novel method for malware clone detection based on deep learning[J/OL]. Journal of Beijing University of Aeronautics and Astronautics, 2021[2021-09-10].https://doi.org/10.13700/j.bh.1001-5965.2020.0400(in Chinese)
(沈元, 严寒冰, 夏春和, 等. 一种基于深度学习的恶意代码克隆检测技术[J/OL]. 北京航空航天大学学报, 2021[2021-09-10].https://doi.org/10.13700/j.bh.1001-5965.2020.0400)

[7] Cummins C, Petoumenos P, Murray A, et al. Compiler fuzzing through deep learning[C]//Proc of the 27th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 2018: 95−105

[8] Lin Ruoqin, Luo Qiong. Software vulnerability detection algorithm based on deformable convolutional neural network[J]. Computer Integrated Manufacturing Systems, 2021, 38(3): 405−409 (in Chinese)
(林若钦, 罗琼. 基于可变形卷积神经网络的软件漏洞检测算法[J]. 计算机仿真, 2021, 38(3): 405−409)

[9] Cummins C, Petoumenos P, Wang Zheng, et al. End-to-end deep learning of optimization heuristics[C]// Proc of the 26th Int Conf on Parallel Architectures and Compilation Techniques (PACT). Piscataway, NJ: IEEE, 2017: 219−232

[10] Chen Tianqi, Moreau T, Jiang Ziheng, et al. TVM: An automated end-to-end optimizing compiler for deep learning[C]// Proc of the 13th USENIX Symp on Operating Systems Design and Implementation (OSDI'18). Berkeley, CA: USENIX Association, 2018: 578−594

[11] Malhotra P, Vig L, Shroff G, et al. Long short term memory networks for anomaly detection in time series[C]//Proc of the 23rd European Symp on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges, Belgium: Louvain-la-Neuve Ciaco, 2015: 89−94

[12] Zhou Jie, Cui Ganqu, Hu Shengding, et al. Graph neural networks: A review of methods and applications[J]. AI Open, 2020, 1: 57−81

[13] LLVM. The LLVM compiler infrastructure[EB/OL]. [2021-09-10].https://llvm.org/

[14] Ganin Y, Ustinova E, Ajakan H, et al. Domain-adversarial training of neural networks[J]. The Journal of Machine Learning Research, 2016, 17(1): 1−35

[15] Magni A, Dubach C, O'Boyle M. Automatic optimization of thread-coarsening for graphics processors[C]//Proc of the 23rd Int Conf on Parallel Architectures and Compilation. New York: ACM, 2014: 455−466

[16] Cooper K D, Schielke P J, Subramanian D. Optimizing for reduced code space using genetic algorithms[C]//Proc of the 2nd ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems. New York: ACM, 1999: 1−9

[17] Cummins C, Fisches Z V, Ben-Nun T, et al. Programl: Graph-based deep learning for program optimization and analysis[J]. arXiv preprint, arXiv: 2003.10536, 2020

[18] Nugteren C, Codreanu V. CLTune: A generic auto-tuner for OpenCL kernels[C]// Proc of the 9th IEEE Int Symp on Embedded Multicore/Many-core Systems-on-Chip. Piscataway, NJ: IEEE, 2015: 195−202

[19] Hamilton W L, Ying R, Leskovec J. Inductive representation learning on large graphs[C]//Proc of the 31st Int Conf on Neural Information Processing Systems. New York: Curran Associates, 2017: 1025−1035

[20] Battaglia P W, Hamrick J B, Bapst V, et al. Relational inductive biases, deep learning, and graph networks[J]. arXiv preprint, arXiv:

1806.01261, 2018

[21] Tzeng E, Hoffman J, Saenko K, et al. Adversarial discriminative domain adaptation[C]//Proc of the 30th IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2017: 7167−7176

[22] Cummins C, Petoumenos P, Wang Zheng, et al. Synthesizing benchmarks for predictive modeling[C]//Proc of the 15th IEEE/ACM Int Symp on Code Generation and Optimization (CGO). Piscataway, NJ: IEEE, 2017: 86−99

[23] Munshi A, Gaster B, Mattson T G, et al. OpenCL Programming Guide[M]. Cambridge, MA: Addison-Wesley, 2011

[24] Allamanis M, Sutton C. Mining source code repositories at massive scale using language modeling[C]//Proc of the 10th Working Conf on Mining Software Repositories (MSR). Piscataway, NJ: IEEE, 2013: 207−216

[25] Mikolov T, Sutskever I, Chen Kai, et al. Distributed representations of words and phrases and their compositionality[C]//Proc of the 26th Advances in Neural Information Processing Systems. New York: Curran Associates, 2013: 3111−3119

[26] Balles L, Hennig P. Dissecting Adam: The sign, magnitude and variance of stochastic gradients[C]//Proc of the 35th Int Conf on Machine Learning. Cambridge, MA: JMLR , 2018: 404−413

[27] Stratton J A, Rodrigues C, Sung I J, et al. Parboil: A revised benchmark suite for scientific and commercial throughput computing[J/OL]. Center for Reliable and High-Performance Computing, 2012[2021-09-10]. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.228.8896&rep=rep1&type=pdf

[28] NVIDIA. NVIDIA CUDA SDK [EB/OL]. 2012 [2021-09-10]. http://developer.nvidia.com/ object/cuda.html

[29] AMD. AMD/ATI stream SDK[EB/OL]. 2010 [2021-09-10]. http://www.amd.com/stream/

[30] Danalis A, Marin G, McCurdy C, et al. The scalable heterogeneous computing (SHOC) benchmark suite[C]//Proc of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units. New York: ACM, 2010: 63−74

[31] Seo S, Jo G, Lee J. Performance characterization of the NAS parallel benchmarks in OpenCL[C]//Proc of the 12th IEEE Int Symp on Workload Characterization (IISWC). Piscataway, NJ: IEEE, 2011: 137−148

[32] Che Shuai, Boyer M, Meng Jiayuan, et al. Rodinia: A benchmark suite for heterogeneous computing[C]//Proc of the 10th IEEE Int Symp on Workload Characterization (IISWC). Piscataway, NJ: IEEE, 2009: 44−54

[33] Grauer-Gray S, Xu Lifan, Searles R, et al. Auto-tuning a high-level language targeted to GPU codes[C/OL]//Proc of the 1st Innovative Parallel Computing (InPar). Piscataway, NJ: IEEE, 2012[2021-09-10].https://ieeexplore.ieee.org/document/6339595

[34] Ben-Nun T, Jakobovits A S, Hoefler T. Neural code comprehension: A learnable representation of code semantics[J]. arXiv preprint, arXiv: 1806.07336, 2018

[35] Brauckmann A, Goens A, Ertel S, et al. Compiler-based graph representations for deep learning models of code[C]//Proc of the 29th Int Conf on Compiler Construction. New York: ACM, 2020: 201−211

[36] Li Yujia, Tarlow D, Brockschmidt M, et al. Gated graph sequence neural networks[J]. arXiv preprint, arXiv: 1511.05493, 2015

[37] Brockschmidt M. GNN-FILM: Graph neural networks with feature-wise linear modulation[C]//Proc of the 37th Int Conf on Machine Learning. Cambridge, MA: JMLR, 2020: 1144−1152

[38] Schlichtkrull M, Kipf T N, Bloem P, et al. Modeling relational data with graph convolutional networks[C]//Proc of the 15th European Semantic Web Conf. Berlin: Springer, 2018: 593−607

[39] Microsoft. Graph neural network with edge MLPs[EB/OL]. 2017 [2021-09-10].https://github.com/microsoft/tf2-gnn#schlichtkrull-et-al-2017

Ye Guixin, born in 1990. PhD, associate professor. Member of CCF. His main research interests include authentication, software security, and software testing.

叶贵鑫, 1990 年生. 博士, 副教授. CCF 会员. 主要研究方向为身份认证、软件安全、软件测试等.

Zhang Yuxiang, born in 1996. Master candidate. His main research interest is program optimization and analysis.

张宇翔, 1996 年. 硕士研究生. 主要研究方向为程序优化和分析.

Zhang Cheng, born in 1998. Master candidate. Her main research interest is machine learning.

张　成, 1998 年生. 硕士研究生. 主要研究方向为机器学习.

Zhao Jiaqi, born in 1995. Master candidate. His main research interests include software security and program optimization.

赵佳棋, 1995 年生. 硕士研究生. 主要研究方向为软件安全和程序优化.

Wang Huanting, born in 1996. Master candidate. His main research interests include software security and program optimization.

王焕廷, 1996 年生. 硕士研究生. 主要研究方向为软件安全和程序优化.