

武汉大学

本科毕业论文（设计）

面向智能装备检测的
RISC-V 专用处理器研究

姓 名：李 禾 越

学 号：2022302111308

专 业：计算机科学与技术

学 院：计 算 机 学 院

指导教师：龚 奕 利

二〇二六年四月

原创性声明

本人郑重声明：所呈交的论文（设计），是本人在指导教师的指导下，严格按照学校和学院有关规定完成的。除文中已经标明引用的内容外，本论文（设计）不包含任何其他个人或集体已发表及撰写的研究成果。对本论文（设计）做出贡献的个人和集体，均已在文中以明确方式标明。本人承诺在论文（设计）工作过程中没有伪造数据等行为。若在本论文（设计）中有侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名：李朱然 指导教师签名：樊爽利
日 期： 2026 年 5 月 14 日

版权使用授权书

本人完全了解武汉大学有权保留并向有关部门或机构送交本论文（设计）的复印件和电子版，允许本论文（设计）被查阅和借阅。本人授权武汉大学将本论文的全部或部分内容编入有关数据进行检索和传播，可以采用影印、缩印或扫描等复制手段保存和汇编本论文（设计）。

作者签名：李朱然 指导教师签名：樊爽利
日 期： 2026 年 5 月 14 日

摘 要

随着工业系统向智能化与边缘化发展，智能装备检测任务对计算平台的实时性与能效提出了更高要求。该类任务面向振动、温度、声学等多源传感信号的实时采集、处理与特征分析，用于实现设备运行状态评估与故障识别。由于其由多种异构算子构成，计算与访存开销在不同阶段交替成为主要性能瓶颈，传统通用处理器与固定功能加速器难以同时兼顾性能与灵活性。因此，基于可扩展指令集架构的领域专用处理器设计成为一种有效途径。

本文围绕面向智能装备检测的 RISC-V 专用处理器研究，提出一种由应用负载驱动的定制指令自动选择方法。首先，通过将程序建模为数据流图，结合静态分析与动态分析，构建综合软件代价、硬件代价与动态热度的收益模型。在此基础上，提出一种基于图同构网络（GIN）与近端策略优化（PPO）的指令选择方法，实现复杂约束下的高效决策。进一步，基于 OpenASIP 平台完成定制指令映射与处理器集成，构建端到端设计流程。

实验结果表明，在给定硬件资源约束下，所提方法能够有效筛选具有实现价值的定制指令，并降低程序执行周期，在基准任务上运行时最高加速比可达 1.3679，验证了方法在智能装备检测场景下的可行性与有效性。

关键词：智能装备检测；RISC-V；专用处理器；定制指令

ABSTRACT

With the advancement of industrial systems toward intelligence and edge computing, intelligent equipment condition monitoring tasks impose stringent requirements on real-time performance and energy efficiency. These tasks involve the acquisition, processing, and feature extraction of multi-source sensor signals, such as vibration, temperature, and acoustic data, for equipment condition assessment and fault diagnosis. Due to heterogeneous operators, computational and memory access overheads alternately become dominant bottlenecks across different stages, making it difficult for general-purpose processors and fixed-function accelerators to achieve both performance and flexibility. Therefore, domain-specific processors based on extensible instruction set architectures provide an effective solution.

This thesis presents a RISC-V domain-specific processor for intelligent equipment condition monitoring and proposes an application-driven method for automatic custom instruction selection. The program is modeled as a data flow graph, and a benefit model is constructed by integrating software cost, hardware cost, and execution hotness through static and dynamic analysis. Based on this, a method leveraging Graph Isomorphism Networks (GIN) and Proximal Policy Optimization (PPO) is proposed for efficient decision-making under constraints. The selected instructions are then mapped and integrated using the OpenASIP platform, forming an end-to-end design flow.

Experimental results show that, under hardware resource constraints, the proposed method effectively identifies valuable custom instructions and reduces program execution cycles. Among the benchmark tasks, the maximum speedup achieved is 1.3679, demonstrating the feasibility and effectiveness of the approach in intelligent equipment condition monitoring scenarios.

Keywords: intelligent equipment condition monitoring; RISC-V; application-specific processor; custom instruction

目 录

摘要	I
ABSTRACT	II
1 绪论	1
1.1 研究背景与意义	1
1.1.1 智能装备检测的发展背景与应用需求	1
1.1.2 检测任务中的计算瓶颈	2
1.1.3 RISC-V 可扩展架构在专用处理器设计中的优势	3
1.2 国内外研究现状	4
1.2.1 面向特定领域应用的专用处理器架构	4
1.2.2 指令扩展与自动化设计方法	6
1.2.3 面向复杂约束与大规模设计空间的智能化方法研究	7
1.2.4 现有方法的局限性与本文切入点	9
1.3 本文主要研究内容	9
1.4 本文组织结构	10
2 面向 RISC-V 定制指令扩展的建模与搜索方法	12
2.1 相关技术基础	12
2.1.1 程序数据流图与定制指令扩展	12
2.1.2 RISC-V 指令集与指令扩展机制	14
2.1.3 图神经网络	15
2.1.4 强化学习	16
2.2 问题形式化描述	17
2.2.1 程序表示	17
2.2.2 候选定制指令表示	18
2.2.3 约束条件	19
2.3 收益建模与综合评价	20
2.4 定制指令识别问题建模	21
2.5 算法总体框架	22
2.6 状态表示与输入构建	24
2.6.1 输入图表示	24

2.6.2	节点属性表示	25
2.6.3	动态活跃度构建	25
2.6.4	节点状态编码	26
2.7	基于 GIN 的图状态编码	26
2.7.1	初始节点表示	26
2.7.2	GIN 消息传递	27
2.7.3	全局上下文表示	27
2.8	基于 PPO 的节点选择策略	28
2.8.1	动作空间与合法动作约束	28
2.8.2	策略网络与价值网络	29
2.8.3	奖励设计	29
2.8.4	PPO 优化目标	29
2.8.5	候选集合维护与 Top-K 保留机制	30
2.9	算法流程总结	31
2.10	本章小结	32
3	面向 RISC-V 定制指令扩展的自动化实现系统	34
3.1	系统总体框架	34
3.2	程序分析与数据流图构建	36
3.3	候选子图到扩展操作的生成与实现	38
3.4	编译映射与程序镜像生成	40
3.5	处理器生成与仿真验证流程	42
3.6	本章小结	43
4	实验结果与性能评估	44
4.1	基准任务说明	44
4.2	实验环境	44
4.3	对比实验	45
4.3.1	指令集架构约束对指令选择的影响	45
4.3.2	定制指令扩展对处理器性能的影响	46
4.4	消融实验	49
4.5	本章小结	50

5 结论与展望	52
5.1 结论	52
5.2 展望	53
参考文献	55
致谢	63

1 绪论

随着智能制造与工业智能化的持续发展,面向复杂运行环境的智能装备检测任务对计算平台提出了更高要求。这类任务通常需要在资源受限条件下完成多源数据处理、状态识别与实时决策,其计算过程兼具较强的数据处理需求、复杂控制特征以及持续演进的算法适应需求。传统通用处理器、固定功能加速器等方案在性能、能效与灵活性之间往往难以同时取得理想平衡,使得面向特定应用场景开展指令集扩展与专用处理器设计成为值得关注的研究方向。

基于此,本文围绕面向智能装备检测任务的 RISC-V 专用处理器设计展开研究,重点关注真实应用负载驱动下的定制指令识别、选择与落地实现问题。本章首先介绍智能装备检测任务的发展背景、计算瓶颈以及 RISC-V 可扩展架构在专用处理器设计中的应用优势;随后综述国内外相关研究现状,分析现有方法的主要不足,并据此明确本文的研究内容、技术路线与全文组织结构。

1.1 研究背景与意义

1.1.1 智能装备检测的发展背景与应用需求

随着工业系统持续向自动化、网络化与智能化方向演进,智能装备检测正在由传统依赖人工巡检和离线分析的模式,逐步转向基于多源数据的在线感知、状态识别与预测性维护。与此同时,检测任务的部署位置也从中心服务器逐渐下沉至嵌入式与边缘计算平台,以降低系统时延、提升响应速度并增强现场适应能力^[1-3]。在这一应用演进过程中,智能装备检测不仅需要处理振动、图像、电流、温度等多类型数据,还需要在资源受限条件下完成连续、实时且可靠的分析,因此对底层计算平台的性能、能效与灵活性提出了更高要求。

从计算特性看,智能装备检测通常涉及滤波、快速傅里叶变换(Fast Fourier Transform, FFT)、卷积以及轻量神经网络推理等典型操作,其运行过程同时呈现出较强的计算密集性与访存密集性。一方面,卷积、频域变换等算子具有较高的数据复用潜力,适合通过专门的数据通路并行机制提升执行效率;另一方面,故障诊断、特征提取与多阶段推理过程中又常伴随复杂控制流、非线性运算以及频繁的数据交换,使系统性能容易受到存储访问开销的制约,呈现明显的存储受限(memory-bound)特征^[4-5]。

此外,由于相关任务通常运行于嵌入式或边缘环境,系统设计还必须在实时

性、功耗和可部署性之间取得平衡^[6-7]。然而, 现有通用处理器虽然具备较好的编程灵活性, 却往往难以同时满足检测任务对吞吐率与能效的要求; 专用加速器虽然能够针对特定算子实现高效计算, 但在面对检测流程多样化和算法持续演进时又缺乏足够的适应能力。因而, 如何在性能、能效与灵活性之间建立更合理的折中, 已成为智能装备检测计算平台发展的关键问题。同时, 对于兼具规则数值计算、非规则控制和多阶段数据交换的检测流程, 单纯依赖面向高并行吞吐优化的通用加速器并不一定能够获得最佳的系统级折中。面向这一需求, 通过指令集扩展构建领域专用处理器, 为在保持一定通用性的同时提升关键检测任务的执行效率提供了一条可行路径。基于上述需求, 兼顾性能、能效与可编程性的专用指令集处理器成为值得关注的技术路径, 相关研究现状将在第1.2.1节进一步讨论。

1.1.2 检测任务中的计算瓶颈

智能装备检测通常包含预处理、特征提取与状态判别等多个处理阶段, 其计算过程既涉及卷积、FFT 等高密度数值运算, 也包含状态控制、流程切换与结果管理等控制相关操作, 因此整体上呈现出规则计算与非规则控制相互交织的特征。在边缘部署场景下, 这类任务的性能瓶颈并不单纯来源于算术计算规模本身, 而更多体现为算法执行特征与底层体系结构之间的失配; 尤其当处理流程同时具有复杂数据访问和多阶段控制特征时, 系统性能更容易受到处理器指令组织方式与存储层次效率的共同限制^[4,8]。

从具体计算热点来看, FFT 和卷积是检测任务中具有代表性的核心算子, 其高频出现使其执行效率对整体系统性能具有重要影响^[8-9]。然而, 在通用处理器上, 这类算子通常需要被拆分为多条细粒度基础指令完成, 导致原本具有较强结构性的计算过程被离散化执行, 从而引入额外的指令调度、循环控制与数据搬运开销。与此同时, 相关算子的性能不仅取决于算术操作数量, 还显著受到数据访问方式的影响; 当访存模式表现出跨度变化、重排频繁或局部性不足等特征时, 系统吞吐率往往进一步受限。已有研究表明, 在既定微结构约束下, 采用面向应用特征的定制指令扩展, 有助于缩短热点操作的执行路径, 更好适配嵌入式应用中的典型计算模式, 从而提升整体执行效率^[10-11]。

除计算路径本身外, 访存开销也是检测任务中不可忽视的关键瓶颈。存储墙(memory wall)现象表明, 处理器性能的提升往往难以同步转化为系统实际执行

效率,其重要原因在于存储访问延迟和带宽约束日益成为主导因素^[4]。在检测任务中,频繁的数据装载、中间结果交换以及寄存器与主存之间的数据移动,会使处理器在大量周期内处于等待存储响应的状态;对于 FFT 这类典型算子,相关研究也指出,数据访问模式对实际性能的影响往往不亚于甚至超过算术复杂度本身^[8]。进一步地,在实时系统或边缘系统环境中,Cache 和 TLB 失效还可能引入额外的执行时间波动,从而削弱系统行为的可预测性与稳定性。由此可见,智能装备检测任务中的性能瓶颈并非单一的计算问题或单一的访存问题,而是计算过程与存储访问深度耦合后形成的系统性约束。

1.1.3 RISC-V 可扩展架构在专用处理器设计中的优势

针对智能装备检测任务中计算密集与访存密集并存、同时又要求一定可编程性与算法适应能力的特点,单纯依赖通用处理器或固定功能加速器都难以取得理想平衡。前者具备较好的通用性,但在热点算子执行效率与能效方面往往受限;后者虽然能够针对特定计算过程进行深度优化,却通常缺乏对算法迭代和任务变化的灵活支持。因此,能够在通用可编程基础上面向特定应用进行定制优化的专用指令集处理器(Application-Specific Instruction-set Processor, ASIP),为缓解上述矛盾提供了一种更具折中性的实现路径。在这一设计思路下,处理器既保留了软件可编程性,又可通过指令集与微结构协同定制提升关键操作的执行效率,而 RISC-V 的开放可扩展架构则为该类设计提供了良好的实现基础^[12]。

RISC-V 采用“基础指令集 + 标准扩展 + 自定义扩展”的模块化组织方式,在保证体系结构兼容性的同时,为面向应用的扩展设计保留了充足空间^[12]。其优势不仅体现在开放性与较低的体系结构复杂度上,更体现在对自定义指令扩展的明确支持:RISC-V 规范对扩展命名规则、编码空间管理以及标准扩展与自定义扩展之间的兼容边界均给出了清晰说明,从而为应用驱动的定制指令设计提供了规范化依据^[13]。这使得设计者能够围绕检测任务中的热点操作构建专用指令,并进一步结合流水线组织、数据通路和存储访问机制开展协同优化,从而推动处理器设计由面向一般负载的通用适配,转向面向目标场景的定制优化。

从设计方法与实现生态来看,RISC-V 与 ASIP 方法具有较高契合性。一方面,LLVM 等工具链已具备对 RISC-V 扩展指令的支持,为定制指令的编译映射与软件集成提供了基础条件^[14];另一方面,OpenASIP 提供了覆盖应用建模、编译映射到处理器实现的 ASIP 设计流程,并在后续版本中进一步支持面向

RISC-V 的指令集与体系结构协同设计^[15-17]。因此,在智能装备检测这一既要求高效执行又要求一定灵活性的应用场景中,基于 RISC-V 架构开展指令集扩展与处理器定制设计,能够较好地体现 ASIP 在性能、能效与可编程性之间的折中优势。已有研究表明,在 RISC-V 处理器中引入面向特定领域的 ISA 扩展,并结合编译器与软件库协同优化,可以在边缘侧量化神经网络推理等任务中获得显著的性能与能效收益,这也从实践层面说明了开放 ISA 生态下开展定制指令设计的可行性^[18]。

1.2 国内外研究现状

1.2.1 面向特定领域应用的专用处理器架构

面向特定领域应用的专用处理器架构,主要是在处理器通用性与执行效率之间寻求平衡。传统通用处理器具有较强的可编程性和软件兼容性,但在特定应用场景下往往难以同时满足性能、功耗和实时性要求;而完全定制的专用硬件虽然效率较高,却通常存在开发周期长、灵活性不足等问题。在这一背景下,应用专用指令集处理器(ASIP)逐渐成为一种重要技术路线,其基本思想是在通用处理器基础上,通过扩展指令集或增加专用功能部件来加速特定应用中的热点计算,从而兼顾灵活性和能效优势^[11,19]。与此同时,部分研究开始从完整设计流程角度出发,将可定制数据通路、软核处理器生成以及编译器支持结合起来,以形成较完整的应用驱动 ASIP/可配置处理器协同设计框架^[20]。

从已有研究和工程实践来看,专用处理器架构主要包括数字信号处理器(Digital Signal Processor, DSP)、图形处理器(Graphics Processing Unit, GPU)以及面向人工智能任务的专用加速器等类型。DSP 以高效支持乘加、滤波和变换等规则性运算为特点,适用于信号处理类任务;GPU 强调数据级并行处理能力,适合大规模相似操作并发执行;AI 加速器则围绕矩阵计算、卷积等核心算子进行专门优化,在神经网络推理中表现出较高的吞吐率和能效^[21-22]。相比之下,ASIP 不追求对某一单一算子的极致固化,而是更强调在特定应用领域内通过指令扩展实现“面向场景的灵活优化”,因此更适合智能装备检测这类算法类型较多、任务需求不断演进的应用场景^[11]。如图1.1所示,不同类型处理器在通用性与硬件定制程度之间形成连续设计空间。通用处理器(General-Purpose Processor, GPP, 典型实现为中央处理器 Central Processing Unit, CPU)强调通用性与软件可编程能力,但在特定应用上的能效与性能受限;专用集成电路

(Application-Specific Integrated Circuit, ASIC) 或专用加速器则通过高度定制化设计实现高性能与高能效,但缺乏灵活性。应用专用指令集处理器(ASIP)位于两者之间,通过针对目标应用扩展指令集并定制关键数据通路,在灵活性与效率之间取得折中。

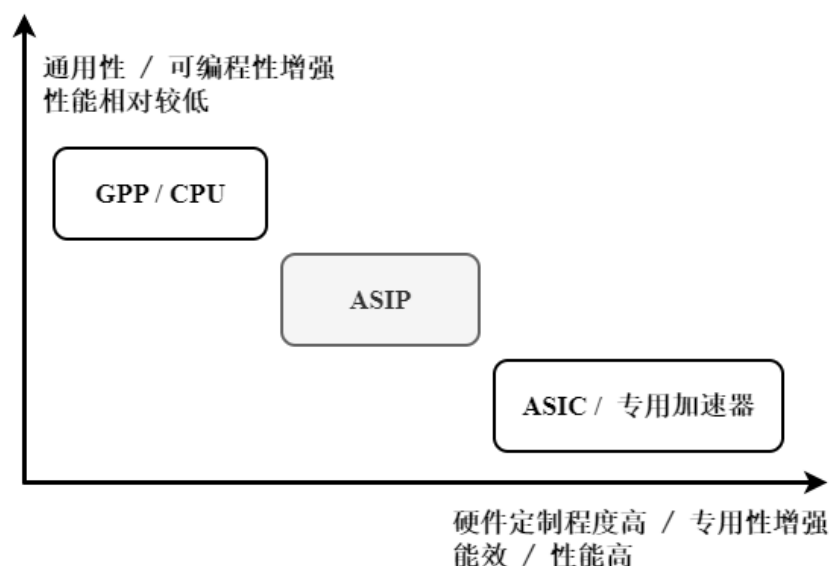


图 1.1 不同类型处理器在通用性与硬件定制程度维度下的设计空间分布及 ASIP 位置

在设计方法上,早期专用处理器设计较多依赖人工分析与经验调优,由设计者根据应用代码手工识别热点计算并定制硬件结构。这类方法针对性较强,但设计效率较低,且结果较依赖专家经验。随着研究不断发展,代码分析、启发式搜索和约束优化等方法逐步被引入专用处理器设计过程,使热点识别、候选生成和收益评估能够形成较系统的自动化流程^[10-11]。其中,代码分析不仅用于识别热点代码区域,还可作为连接静态程序结构分析与动态执行行为评估的辅助设计工具,从而提高候选定制指令收益评估对真实工作负载的贴近程度^[23]。不过,总体来看,面向特定领域的专用处理器设计仍然存在设计周期长、泛化能力有限以及对领域知识依赖较强等问题,这也推动了自动化指令扩展方法的进一步发展^[11,19]。

近年来,随着应用复杂度的不断提升以及设计空间规模的迅速扩大,传统依赖人工经验的基于规则和启发式的设计方法逐渐暴露出效率低、可扩展性差等局限性,难以在复杂约束下获得接近最优的设计方案。因此,相关研究逐渐转向更加自动化的设计流程,通过结合程序分析、组合优化以及机器学习等技术,实现对设计空间的高效探索与性能优化^[24]。

1.2.2 指令扩展与自动化设计方法

指令扩展的基本思想,是从目标应用程序中识别具有较高加速价值的计算子结构,并将其映射为可由硬件直接支持的定制指令,从而减少原有指令序列的执行开销,提升程序整体性能与能效^[10-11,25-28]。对于面向智能装备检测的应用而言,这类方法具有较强吸引力,因为相关算法通常同时包含滤波、变换、特征提取以及轻量神经网络推理等多类计算过程,程序中往往存在较为明显的热点计算模式,为定制指令设计提供了良好基础。

从自动化设计流程看,指令扩展问题通常可以分为两个核心子问题:候选指令枚举(custom instruction enumeration)与候选指令选择(custom instruction selection)^[11,29-31]。前者关注如何从程序表示中识别满足约束条件的候选计算子图,后者则关注如何在多个候选指令之间进行权衡,选出能够带来最大总体收益的指令扩展集合。这种划分方式相比按规则模板或模式类型分类,更能准确反映自动化指令扩展方法的求解流程与优化本质。

从问题本质上看,定制指令扩展涉及在满足多种微体系结构约束条件下,对程序中的候选计算模式进行全局选择与组合,其目标是在性能收益与硬件代价之间取得最优折中。因此,该问题通常可被建模为带约束的组合优化问题。随着程序规模和候选模式数量的增加,候选空间呈指数级增长,使得简单的局部贪心策略难以获得接近全局最优的解。这一特性也促使相关研究逐渐从基于规则的局部优化方法,转向能够进行全局搜索与决策的优化方法^[26,32]。

尽管现有方法在特定应用场景下已经能够获得较好的优化效果,但其仍然面临若干局限。首先,许多方法高度依赖静态程序结构分析,对复杂控制流、跨基本块模式以及动态行为特征的刻画能力有限。其次,随着程序规模增大和候选图结构复杂度提升,候选空间容易迅速膨胀,传统启发式方法在不同任务上的稳定性与泛化能力也受到限制^[11,33]。此外,在智能装备检测任务中,多源数据处理、异构算子组合以及负载变化等特点进一步提高了指令扩展自动化设计的难度。

基于上述问题,近年来研究开始尝试引入更智能的建模与搜索方法,以联合刻画程序结构特征、硬件约束以及性能收益,提高候选指令搜索与选择过程的自动化程度和适应能力。例如,将候选选择过程建模为序列决策或组合优化问题,并借助学习型方法提升复杂设计空间中的搜索效率,已成为值得关注的研究方

向^[34]。这也为本文面向智能装备检测应用的 RISC-V 定制指令扩展研究提供了方法上的参考。

1.2.3 面向复杂约束与大规模设计空间的智能化方法研究

定制指令选择问题通常可形式化为带约束的组合优化问题，在某些建模方式下还可进一步转化为集合覆盖问题或加权最大团问题等等价形式^[26,29,32,35-37]。给定程序的数据流图，需要在满足输入输出约束和硬件资源限制的条件下，从候选子图集合中选择若干模式，以最大化性能收益或最小化实现成本^[26,32]。其中，在满足凸性与 I/O 约束条件下进行候选子图枚举，是定制指令识别的重要基础问题，相关研究已对最大凸子图求解与约束处理进行了系统分析^[38]。由于候选模式之间存在重叠与资源竞争，该问题具有明显的全局优化特性。

现有方法主要包括精确方法与启发式方法两类。精确方法如整数线性规划 (ILP) 和动态规划能够在严格建模约束的前提下获得全局最优解，但计算复杂度较高，难以扩展到大规模问题^[26,32,39]。启发式方法如贪心算法、遗传算法和模拟退火能够较快获得可行解，但其结果依赖于搜索策略，稳定性和最优性难以保证^[40-42]。随着程序规模和候选模式数量增加，设计空间呈指数级膨胀，如何在保证求解质量的同时提高搜索效率，成为该领域的重要问题^[26]。

由于程序计算过程天然具有数据依赖结构，定制指令识别与选择问题适合被建模为图结构上的子结构发现与全局选择问题。通常，程序可表示为数据流图 (DFG)，其中节点表示操作，边表示数据依赖关系，从而为候选模式分析与自动化决策提供统一基础^[43]。在此基础上，图神经网络可用于学习节点及子图的结构表示，强化学习则可将候选指令选择过程建模为受约束的序列决策问题。二者结合后，能够在保留程序结构信息的同时引导搜索过程，为大规模定制指令选择提供更具扩展性的求解路径^[44-45]。

例如，S2V-DQN 将图嵌入与强化学习结合，在多类图优化问题中表现出较好的求解能力^[44]。近年来，相关研究进一步将图神经网络与 PPO 等策略优化方法引入定制指令识别与选择问题，用于在复杂约束和大规模程序图下学习更优的决策策略^[34,45-46]。相关结果表明，与传统方法相比，该类方法在图规模增大时通常能够在求解质量、运行开销与资源利用之间保持更稳定的综合折中^[46]。

如图1.2所示，随着节点规模增加，基于 PPO 的方法在平均加速比方面整体优于 SARSA 和 ILP，并在大规模场景下表现出较好的可扩展性；同时，其运行

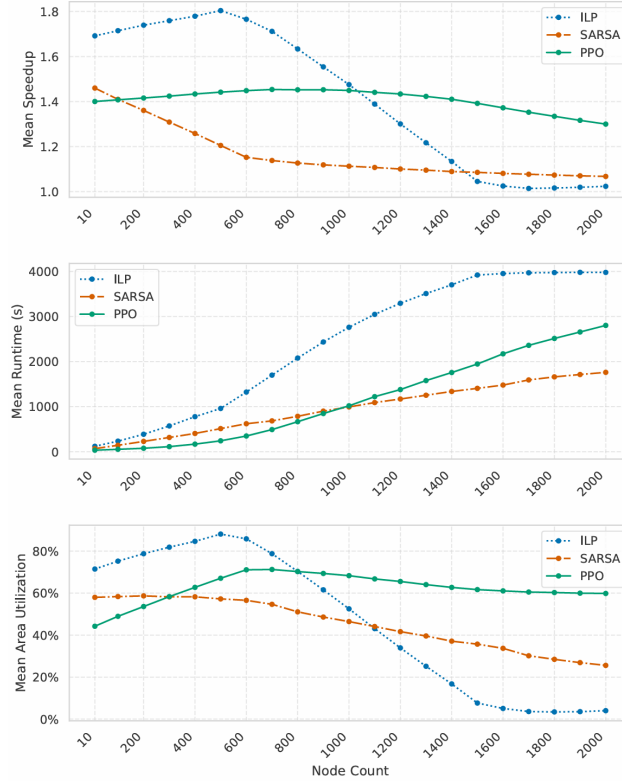


图 1.2 不同方法在不同节点规模下的平均加速比、运行时间与面积利用率对比^[46]

时间增长趋势相对平缓，面积利用率也保持在较高水平，说明学习驱动的方法在大规模数据流图优化问题中具有较好的应用潜力^[46]。

此外，部分最新研究开始考虑定制指令与底层微结构之间的协同优化，通过在指令识别阶段引入流水线冲突、资源竞争、结构冒险以及相关控制机制等约束，使设计结果更加贴近实际处理器实现^[47-50]。

需要指出的是，现有学习驱动方法虽然为大规模定制指令选择提供了新的思路，但多数工作仍主要关注程序图层面的模式识别与组合选择，对真实应用负载中的热点分布、动态执行行为以及具体处理器实现约束考虑仍相对有限。因此，如何将程序图结构、运行时特征与硬件资源约束统一纳入选择过程，仍是值得进一步研究的问题。

基于上述分析，本文面向智能装备检测任务中的复杂应用负载，构建 GIN + PPO 的联合框架，用于求解大规模、强约束条件下的定制指令自动选择问题。其中，GIN 用于增强候选子图结构差异的表达能力，PPO 用于提升复杂约束条件下策略学习的稳定性与泛化能力。本文进一步结合动态分析信息以及 RISC-V 专用处理器实现约束，将输入输出端口限制、面积预算和非法操作约束纳入候

选指令选择过程,使优化目标更加贴近真实应用驱动的处理器设计场景。

1.2.4 现有方法的局限性与本文切入点

综上,现有面向领域应用的处理器定制设计、定制指令自动识别与选择方法,以及基于图建模的学习方法,已形成较为丰富的研究基础。然而,将其直接应用于智能装备检测场景时仍存在不足:一是已有研究多面向通用基准程序或规则性较强的计算核,难以适应智能装备检测中混合型计算模式及其动态变化的热点特征;二是传统方法较多依赖人工经验、启发式规则或局部匹配,在复杂程序图和多重硬件约束下容易面临搜索空间膨胀与全局优化不足的问题;三是已有学习驱动方法对真实应用负载、动态分析信息以及后续 ISA 扩展落地实现之间的协同考虑仍不充分。

基于此,本文将研究切入点确定为面向智能装备检测任务的定制指令自动选择问题,重点探索一种由真实应用负载驱动、融合静态程序结构与动态执行行为、并服务于 RISC-V 专用处理器实现的系统化方法。本文结合图神经网络与强化学习,对候选子图的结构特征、执行收益与硬件约束进行联合建模,并进一步将优化结果映射到 RISC-V 可扩展指令集架构中,依托 OpenASIP 平台完成指令生成、处理器集成与性能验证。

1.3 本文主要研究内容

本文围绕“面向智能装备检测的 RISC-V 专用处理器”展开研究,重点关注真实应用负载驱动下的定制指令自动识别、选择与处理器落地实现,主要内容如下。

(1) 面向智能装备检测应用,结合静态程序结构分析与动态分析信息,构建同时反映节点操作类型、执行热度和硬件代价的程序图表示,为候选定制指令评估与选择提供更贴近真实工作负载的输入。

(2) 针对凸性、输入输出端口数、面积预算和指令合法性等约束,将定制指令选择过程建模为数据流图上的约束序列决策问题,为复杂应用场景下的自动化定制指令选择提供统一的问题表达。

(3) 设计了一种基于 GIN 与 PPO 的定制指令自动选择方法。利用 GIN 对程序图中的节点与子图结构进行表示学习,增强对复杂候选模式的区分能力;结合 PPO 学习约束条件下的节点选择策略,以优化定制指令构造过程中的长期累计收益。

(4) 将优化结果映射到 RISC-V 可扩展指令集架构中, 依托 OpenASIP 平台完成定制指令集成、处理器原型构建与实验验证, 形成从应用分析、候选指令选择到体系结构实现与性能评估的完整流程。

1.4 本文组织结构

本文围绕面向智能装备检测任务的 RISC-V 专用处理器设计, 按照“问题建模—算法设计—系统实现—实验评估”的主线展开, 构建从应用需求分析到处理器验证的完整技术链路, 本文组织结构如图1.3所示。

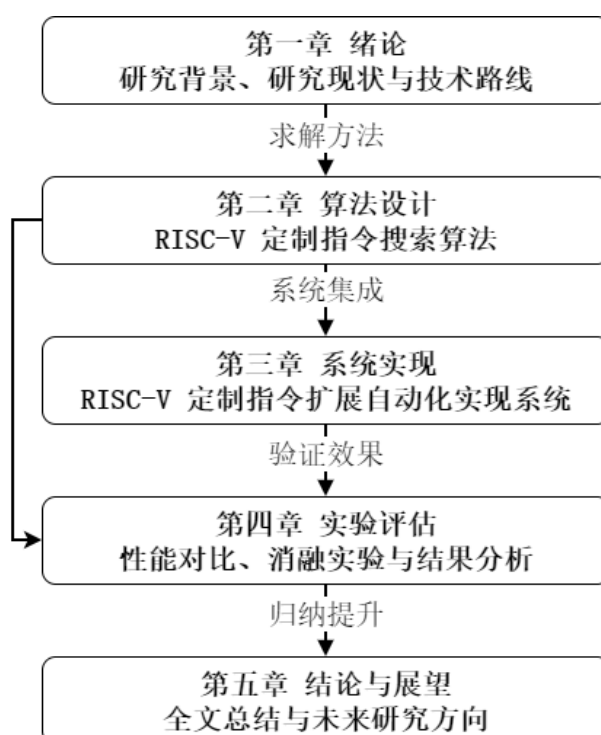


图 1.3 本文组织结构

第一章为绪论。介绍智能装备检测的背景与应用需求, 分析边缘场景下的计算与访存瓶颈, 以及 RISC-V 架构在专用处理器设计中的优势; 综述领域专用处理器、定制指令选择、设计空间探索及图学习与强化学习方法的研究现状, 总结现有工作的不足, 并明确本文的研究内容与技术路线。

第二章研究面向 RISC-V 定制指令扩展的建模与搜索方法。该章首先介绍相关技术基础, 而后以程序数据流图为统一表示形式, 将候选定制指令抽象为数据流图中的诱导子图, 并从合法操作、连通性、凸性、输入输出规模以及面积预算等方面建立硬件可实现性约束。在此基础上, 进一步构建软件执行代价、硬件执行代价、静态收益与动态活跃度相结合的综合评价模型, 将定制指令识别

问题形式化为受多重约束的 Top- K 候选子图识别问题。针对候选空间随程序规模增长而迅速扩大的问题,该章将候选子图的增量构造过程建模为带约束的序列决策过程,提出基于图同构网络(Graph Isomorphism Network, GIN)与近端策略优化(Proximal Policy Optimization, PPO)的搜索方法。该方法利用 GIN 编码程序数据依赖结构,利用 PPO 学习节点扩展策略,并结合合法动作掩码与候选集合维护机制,实现高价值候选定制指令的自动搜索与保留。

第三章为面向 RISC-V 定制指令扩展的自动化实现系统。基于算法输出,完成候选子图到扩展操作的映射,并实现 RISC-V 扩展指令;构建涵盖候选选择、描述生成、编译支持、OpenASIP 集成及处理器生成与验证的自动化流程。

第四章为实验结果与性能评估。在典型智能装备检测基准任务上,从对比实验与消融实验两方面验证方法有效性,分析指令扩展对性能(例如执行周期)的影响,并评估不同图编码器与强化学习策略的性能差异。

第五章为结论与展望。总结全文工作与主要结论,并从高收益指令扩展、多输入输出支持、微结构协同优化及系统稳定性等方面展望未来研究方向。

2 面向 RISC-V 定制指令扩展的建模与搜索方法

上一章分析了智能装备检测任务的计算特征与性能瓶颈，说明了面向特定应用扩展 RISC-V 指令集的必要性，并指出定制指令识别是专用处理器设计的关键环节。针对该类应用中存在的热点计算模式，若能自动发现高复用、高收益且硬件可实现的局部计算结构，并将其映射为定制指令，可在保持 RISC-V 开放性与可扩展性的同时提升执行效率。

定制指令识别本质上是受约束的子图搜索问题。候选指令通常对应程序数据流图中的子图，其可行性受操作类型、数据依赖、输入输出端口、凸性和硬件面积等因素限制；其收益还与动态执行频率密切相关。随着程序规模扩大，候选子图数量迅速增长，传统穷举方法难以满足实际需求。因此，需要对该问题进行形式化建模，并设计兼顾搜索效率、约束满足与候选质量的自动化方法。

本章围绕上述问题展开研究。首先，本章在2.1节介绍相关技术基础。随后，以程序数据流图为基础，将候选定制指令抽象为受约束的诱导子图，分别从合法操作、连通性、凸性、输入输出规模以及面积预算等方面建立硬件可实现性约束；而后，结合静态执行代价、硬件关键路径代价与动态活跃度信息，构造候选子图的综合评价函数，并将定制指令识别问题表述为受约束的 Top- K 候选子图识别问题。在此基础上，进一步将候选子图的增量构造过程建模为带约束的马尔可夫决策过程，提出一种基于图同构网络 (Graph Isomorphism Network, GIN)^[51] 与近端策略优化 (Proximal Policy Optimization, PPO)^[52] 的候选定制指令搜索方法。该方法利用 GIN 编码数据流图中的局部依赖结构，利用 PPO 学习节点扩展策略，并通过合法动作掩码与 Top- K 候选维护机制，在搜索过程中持续保留综合评价较高的候选定制指令，为后续指令编码、硬件实现与实验评估奠定基础。

2.1 相关技术基础

2.1.1 程序数据流图与定制指令扩展

在面向特定应用的定制指令扩展设计中，程序的数据流结构为识别可融合操作提供了核心依据。通常，研究对象以程序基本块或热点区域对应的数据流图 (Data Flow Graph, DFG) 为基础进行建模；在工程实现中，DFG 往往由静态单赋值形式 (Static Single Assignment, SSA) 或 LLVM IR 等中间表示中的数据依

赖关系抽取得到。编译器首先通过代码分析、基本块划分以及数据依赖分析提取热点代码区域,并在此基础上构建 DFG,其中节点表示算术或逻辑操作,边表示数据依赖关系^[11,33]。相比源代码级表示,DFG 能够更直接地反映指令级并行性以及操作之间的组合潜力,从而为定制指令的构造提供结构化基础^[11,24,33,41]。

在候选指令枚举阶段,核心任务是在 DFG 中搜索可映射为单条扩展指令的子图结构。这一过程通常需要满足若干微体系结构约束,例如子图的凸性约束、输入输出操作数个数限制、非法操作约束以及面积或时延等硬件代价约束^[10,25,30,38]。这些约束本质上反映了从“程序结构”到“可实现指令”之间的映射边界,即只有满足硬件可实现性的子图,才能进一步转化为实际的指令扩展。

需要指出的是,随着程序规模的增长,DFG 中潜在子图数量呈指数级增长,使得候选枚举问题本质上成为一个受约束的组合搜索问题。为控制搜索复杂度,已有研究提出了多种方法,包括基于剪枝的精确枚举、动态规划、凸子图枚举、分支定界方法,以及结合启发式规则的近似搜索策略等^[26,31-32,40,43]。这些方法在搜索空间覆盖度与计算开销之间进行权衡,为后续候选选择阶段提供高质量的候选集合。

在候选指令选择阶段,目标是在面积、时延、功耗、寄存器端口数以及候选之间依赖关系等约束条件下,从已枚举的候选子图中选出总体收益最大的指令集合^[10-11]。与简单的局部优化不同,该问题需要综合考虑候选子图之间的重叠关系、可组合性以及其对整体程序执行路径的影响,因此通常被建模为带约束的组合优化问题,在多数情况下具有较高的计算复杂度^[35-37,53]。针对这一问题,已有研究广泛采用整数线性规划、动态规划、约束规划、分支定界方法以及多种启发式与元启发式算法,以在求解质量与效率之间取得平衡^[32,35-37,53]。

从评价指标来看,定制指令的优劣不能仅由子图规模或操作数目决定,而需要结合收益与代价进行综合分析。常见收益指标包括候选指令带来的执行周期减少、程序整体加速比以及热点代码覆盖程度;常见代价指标则包括硬件面积开销、关键路径时延、能耗以及额外的数据传输成本等^[10-11]。因此,定制指令扩展设计本质上是一个多目标权衡过程,其结果不仅依赖于程序本身的数据流结构特征,也与目标处理器的微体系结构约束密切相关。

进一步地,从体系结构实现角度来看,上述子图到指令的映射过程与 RISC-V 提供的指令扩展机制形成了良好的契合关系。具体而言,在操作数个数、数据

位宽、访存副作用以及异常语义可控的前提下,满足约束的 DFG 子图可以被抽象为具有固定输入输出接口的复合操作,并通过 RISC-V 预留的自定义指令编码空间进行表达;同时,其操作语义可映射为扩展执行单元中的组合逻辑或流水化功能单元。由此,程序层的数据流结构、指令级表示以及硬件执行单元之间形成了一种统一的映射关系,这也为后续基于学习方法的自动化定制指令生成提供了建模基础。

2.1.2 RISC-V 指令集与指令扩展机制

RISC-V 指令集架构采用模块化设计思想,由基础整数指令集(如 RV32I / RV64I)与若干标准扩展(如 M、A、F、D 等)构成,同时明确支持用户定义的自定义扩展,从而在保证体系结构兼容性的前提下提供良好的可扩展性^[12-13]。这种“基础指令集 + 标准扩展 + 自定义扩展”的分层组织方式,使得处理器设计能够在通用性与专用性之间进行灵活权衡。

在指令编码层面,RISC-V 为自定义指令预留了若干操作码空间,这些编码不会与未来标准扩展发生冲突^[12-13]。设计者可以在这些预留空间内定义新的操作语义,从而实现面向特定应用的指令扩展。自定义指令通常基于已有指令格式(如 R 型、I 型、S 型等)进行扩展,其基本思路是复用原有的寄存器操作数编码字段(rs1、rs2、rd),并通过 funct3、funct7 等功能字段区分不同操作,从而在不改变指令基本译码结构的前提下引入新的功能。

从硬件实现角度来看,自定义指令扩展主要涉及译码逻辑与执行单元的协同扩展。一方面,在译码阶段需要识别新增的 opcode 与 funct 组合,并将其映射为特定控制信号;另一方面,在执行阶段需引入新的功能单元或扩展现有算术逻辑单元(ALU),以支持扩展指令对应的运算语义。此外,对于涉及复杂数据访问模式的指令,还可能需要对访存路径或寄存器堆访问机制进行相应调整。因此,自定义指令设计本质上是指令集与微体系结构的协同优化过程。

在软件支持方面,自定义指令的有效使用依赖于编译器工具链的支持。当前主流编译框架(如 LLVM)已支持 RISC-V 架构及其扩展机制,设计者可以通过引入 intrinsic 函数、后端指令模式或内联汇编的方式,将高层语言中的特定计算模式映射为自定义指令^[14,42]。进一步地,通过扩展编译器后端(如指令选择、模式匹配与调度优化),可以实现从程序数据流图到定制指令的自动映射,从而充分发挥扩展指令在性能与能效方面的优势。此外,OpenASIP 等 ASIP 设计工

具链也为应用专用处理器的软硬件协同设计、指令集定制以及编译器支持提供了工程化支撑^[15-17]。

综上, RISC-V 提供了一种规范化且工程可行的指令扩展机制, 使得设计者能够围绕应用特征定义专用操作, 并通过软硬件协同设计实现性能优化。这一机制为面向特定领域的专用处理器(如 ASIP)设计提供了重要基础, 也为后续基于程序分析与自动化方法的定制指令生成奠定了技术前提。

2.1.3 图神经网络

在面向程序结构的表示学习问题中, 传统向量或序列表示往往难以充分刻画操作之间的数据依赖关系、控制关系以及拓扑结构特征。相比之下, 图表示学习方法能够以节点和边的形式显式描述程序中的操作、依赖和控制结构, 因此更适合用于程序数据流图、控制流图以及中间表示层面的结构建模^[54-55]。图神经网络(Graph Neural Network, GNN)通过在图结构上进行邻域信息聚合与表示更新, 能够学习节点及其上下文的嵌入表示, 从而捕获局部依赖模式与较高层次的结构特征^[56]。在程序分析领域, 已有研究表明, 将程序表示为包含数据流、控制流或调用关系的图结构, 有助于神经网络模型学习程序语义和结构特征^[54-55]。

GNN 的核心思想是基于消息传递机制, 在每一层中对节点邻域的信息进行聚合, 并结合节点自身特征进行非线性变换, 从而逐层扩展感受野, 实现从局部到全局的结构建模。在程序数据流图(DFG)中, 节点通常对应具体操作, 边表示数据依赖关系, 因此通过多层消息传递, GNN 能够逐步融合操作之间的依赖信息, 学习到反映子图结构特征的表示。这种基于结构传播的表示方式, 使其特别适用于刻画候选子图之间的结构相似性与依赖关系。

在具体模型方面, 图卷积网络(Graph Convolutional Network, GCN)通过对谱域卷积的近似实现邻域特征聚合, 是推动现代图神经网络广泛应用的代表性模型之一^[57]。GCN 在结构较为平滑的图中能够有效提取局部特征, 但其表达能力在区分不同拓扑结构方面存在一定局限。相较之下, 图同构网络(Graph Isomorphism Network, GIN)通过引入更具判别性的聚合函数(如基于多层感知机的非线性变换), 在消息传递类图神经网络中具有与 Weisfeiler-Lehman (WL)图同构测试相当的判别能力, 从而能够更精确地区分结构相似但连接模式不同的子图^[51]。

针对定制指令扩展问题, 候选子图的优劣不仅取决于节点数量或操作类型, 还与其内部拓扑结构及数据依赖模式密切相关。例如, 具有相同节点规模的两个子图, 其关键路径长度或并行性特征可能存在显著差异, 从而导致不同的性能收益。因此, 需要一种能够敏感捕捉结构差异的表示方法。而在需要区分候选子图细粒度拓扑差异的场景下, GIN 相比简单邻域平均类方法具有更强的结构判别潜力。类似地, 在图结构组合优化问题中, 基于图表示学习的方法已被用于学习节点、边或子图级别的决策策略^[44-45]。

综上, GNN 能够为程序数据流图提供统一的结构化表示框架, 通过学习节点及其上下文的嵌入向量, 刻画候选子图的结构特征、相似性以及潜在依赖关系。这些表示可进一步作为强化学习策略网络的输入, 为后续定制指令选择决策提供具有判别性的特征基础。

2.1.4 强化学习

在定制指令选择问题中, 决策过程不仅需要考虑单个候选子图的局部收益, 还需综合其与其他候选之间的依赖关系、资源竞争以及对整体程序执行路径的影响。因此, 该问题具有显著的序列决策特性, 可形式化建模为一个马尔可夫决策过程 (Markov Decision Process, MDP)^[58]。在该建模框架下, 状态用于描述当前已选择的候选集合、剩余候选集合及其在程序数据流图中的结构信息, 动作对应对候选子图的选择、跳过或终止操作, 而奖励函数则用于刻画当前决策对整体性能收益与硬件代价的综合影响^[34,58]。在该建模中, 若状态能够充分包含当前候选集合及资源约束等信息, 则后续状态转移仅依赖于当前状态与动作, 从而满足马尔可夫性假设。

强化学习已被用于若干组合优化问题中, 通过学习构造式决策策略来近似求解复杂离散优化任务^[44-45,59]。与传统基于整数线性规划或启发式搜索的方法不同, 强化学习通过与环境的交互, 逐步学习从状态到动作的映射策略, 从而能够在决策过程中显式建模“当前选择对未来收益的影响”。在定制指令扩展场景中, 不同候选子图之间往往存在重叠或资源冲突关系, 早期决策会对后续可选空间产生约束, 因此仅基于局部最优的选择策略难以获得全局最优解。强化学习通过累积回报的优化目标, 可以在长时序决策过程中权衡即时收益与长期收益, 从而更好地适应该类具有全局依赖性的优化问题。

在状态表示方面, 候选子图及其依赖关系通常以程序数据流图的形式给出,

其结构复杂且维度较高。传统表格型强化学习方法难以直接处理此类高维、结构化状态空间，而深度强化学习通过引入神经网络进行函数逼近，能够有效学习复杂状态到策略或价值函数的映射^[58]。因此，将图神经网络与强化学习相结合，利用 GNN 对程序图结构进行编码，再将其作为强化学习策略网络的输入，是处理图结构组合优化问题的一种自然选择^[44-45]。

在具体算法选择上，策略梯度方法通过直接优化参数化策略，适用于动作空间较大且难以显式枚举全局最优决策的场景。近端策略优化（Proximal Policy Optimization, PPO）通过引入裁剪式代理目标函数，对新旧策略之间的更新幅度进行约束，从而在提高采样效率的同时增强训练稳定性^[52]。因此，在候选指令数量较大、候选之间存在重叠与资源冲突的情况下，采用基于 PPO 的强化学习方法，有助于在复杂设计空间中学习较优的指令选择策略。

综上，强化学习为定制指令选择提供了一种从“全局序列决策”角度出发的建模方法，通过结合图结构表示与策略学习机制，能够在复杂设计空间中探索高质量的指令组合方案，为后续自动化指令扩展方法的实现奠定基础。

2.2 问题形式化描述

定制指令识别的核心任务在于，从目标程序的计算结构中发现满足硬件实现约束且具有较高加速潜力的局部计算模式。为对该任务进行严格表述，本文以程序数据流图作为统一的程序表示形式，将候选定制指令抽象为数据流图上的子图，并从合法操作、结构完整性、接口规模及硬件资源等维度建立约束体系。以下依次给出程序表示、候选子图定义及各项约束条件的形式化描述。

2.2.1 程序表示

设目标应用经过程序分析后可表示为一个有向数据流图

$$G = (V, E), \quad (2.1)$$

其中， V 为节点集合，包括实际运算指令节点以及为刻画输入边界而引入的占位节点； $E \subseteq V \times V$ 为有向边集合，表示节点之间的数据依赖关系。若 $(u, v) \in E$ ，则表示节点 v 的某个输入依赖于节点 u 的输出。

对于任意节点 $v \in V$ ，记其操作类型为 $\text{op}(v)$ ，其对应的硬件延迟代价和面积代价分别为 $d(v)$ 与 $a(v)$ 。进一步地，记节点的动态执行权重为 $h(v)$ ，用于表征该节点在目标应用运行过程中的相对活跃程度；该权重可由动态分析统计值或归一化执行频次给出。

2.2.2 候选定制指令表示

设 $S \subseteq V$ 为从图 G 中选出的节点集合, 则由其诱导得到的子图记为

$$G[S]. \quad (2.2)$$

本文将一个候选定制指令抽象为 DFG 上的一个候选子图 $G[S]$ 。该候选只有在满足结构、接口和代价约束时, 才可被视为可行定制指令。下文中, “候选子图” 强调其在 DFG 上的图结构表示, “候选定制指令” 强调其潜在的硬件映射含义; 在问题建模层面, 二者可对应使用。

为便于说明, 图2.1给出了一个数据流图中候选定制指令子图的示例。图中每个椭圆节点表示一条中间指令或运算操作, 有向边表示数据依赖关系。虚线框内的节点集合构成一个候选子图, 其内部包含 `and`、`or` 和 `add` 等操作。若该子图满足合法操作、连通性、凸性、输入输出端口数量以及面积预算等约束, 则可被视为一个候选定制指令, 并进一步映射为硬件中的复合运算单元。

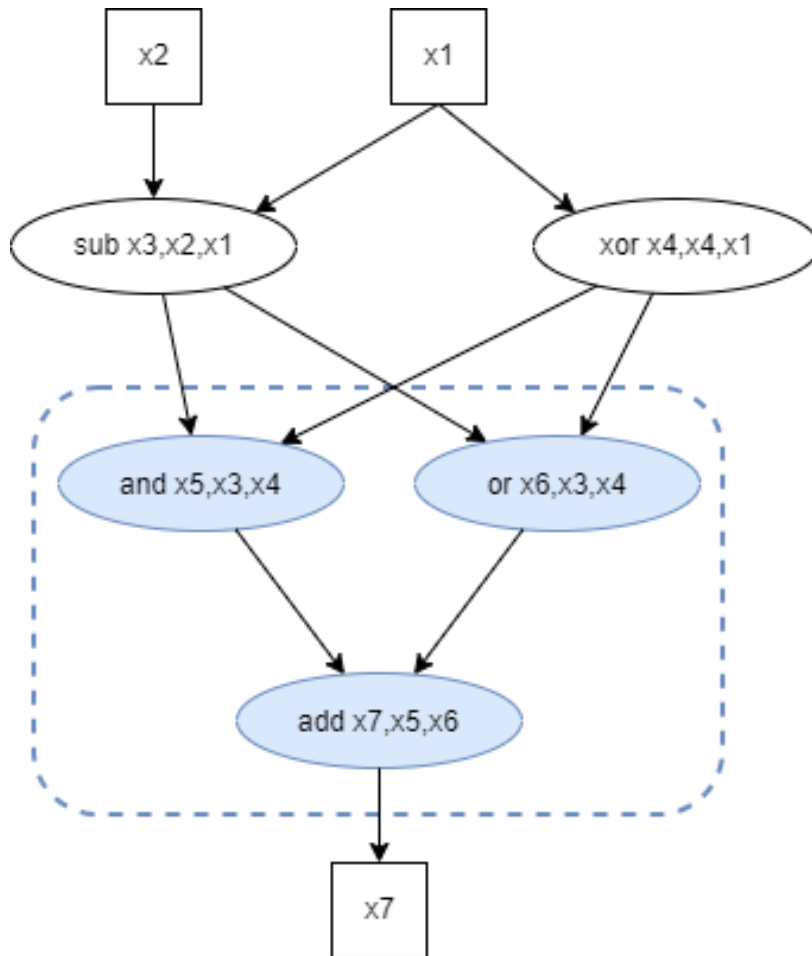


图 2.1 数据流图中的候选定制指令子图示例

可以看到, 候选子图并不是任意节点集合, 而必须在保持原始数据依赖关系正确性的前提下形成可封装的局部计算结构。因此, 后续需要从结构完整性、接口规模和硬件代价等方面对候选子图进行约束。

2.2.3 约束条件

为保证候选定制指令具有可实现性, 本文考虑如下约束。

其一, 合法操作约束。由于并非所有程序操作都适合被纳入定制指令, 故各操作均需满足合法性检测。一般而言, 算术运算、逻辑运算、移位运算及比较等规则性较强的操作更适合进行局部聚合; 而与控制流、复杂访存、函数调用或地址生成密切相关的操作通常不宜直接纳入候选子图。因此, 定义允许进入候选子图的操作集合为 $\mathcal{O}_{valid}$, 则应满足

$$\forall v \in S, \quad \text{op}(v) \in \mathcal{O}_{valid}. \quad (2.3)$$

其二, 连通性约束, 即候选子图应构成一个连通局部模式。记 $G[S]^{undir}$ 为 $G[S]$ 的无向投影, 则要求

$$G[S]^{undir} \text{ is connected.} \quad (2.4)$$

其三, 凸性约束。若存在某个未被选中的节点位于候选子图内部节点之间的依赖路径上, 则该候选模式将破坏原始依赖关系的闭合性。因而, 候选子图应满足凸性约束。形式上, 若存在 $x \notin S$ 与 $u, v \in S$, 使得

$$u \rightsquigarrow x, \quad x \rightsquigarrow v, \quad (2.5)$$

则认为该候选子图不满足凸性。

其四, 输入输出约束。定义候选子图的输入集合为

$$\text{IN}(S) = \{u \notin S \mid \exists v \in S, (u, v) \in E\}, \quad (2.6)$$

即所有由子图外部流入子图内部的数据源节点。定义候选子图的输出集合为

$$\text{OUT}(S) = \{u \in S \mid \exists v \notin S, (u, v) \in E\}, \quad (2.7)$$

即所有从子图内部输出到子图外部的结果节点。这里的 $\text{OUT}(S)$ 表示候选子图内部其结果被子图外部节点使用的边界输出节点, 而非子图外部接收节点本身。设系统允许的最大输入数与最大输出数分别为 $I_{\max}$ 和 $O_{\max}$, 则候选子图应满足

$$|\text{IN}(S)| \leq I_{\max}, \quad |\text{OUT}(S)| \leq O_{\max}. \quad (2.8)$$

其五, 面积预算约束。定制指令在获得性能收益的同时, 也会引入额外硬件开销, 因此候选子图的规模不能无限扩张。本文使用节点面积代价之和近似表征候选子图的硬件资源消耗。设节点 v 的面积代价为 $a(v)$, 则候选子图 S 的面积可表示为

$$A(S) = \sum_{v \in S} a(v). \quad (2.9)$$

与采用固定绝对面积上限的方式不同, 本文将面积预算定义为相对于当前数据流图总体面积的比例约束。设原始 DFG 的总面积为

$$A_{\text{total}} = \sum_{v \in V} a(v), \quad (2.10)$$

则候选定制指令所允许的面积预算写为

$$A_{\text{budget}} = \rho A_{\text{total}}, \quad (2.11)$$

其中, $\rho \in (0, 1]$ 为面积预算系数。于是, 候选子图需满足

$$A(S) \leq A_{\text{budget}}. \quad (2.12)$$

这种约束方式能够使面积限制随不同程序图的规模自适应变化, 从而避免在不同 DFG 之间使用统一绝对阈值所带来的不平衡问题。

2.3 收益建模与综合评价

在满足上述约束的前提下, 本文关注候选定制指令对目标应用执行效率的提升能力。为此, 引入软件执行代价与硬件执行代价两个量。

当候选子图仍由基础指令序列完成时, 其软件执行代价可近似表示为节点延迟之和, 即

$$D_{sw}(S) = \sum_{v \in S} d(v). \quad (2.13)$$

该定义忽略了处理器流水线重叠、指令调度及访存等待等微结构因素, 主要用于对候选模式进行统一的相对收益评估。

当候选子图被聚合为一条定制指令后, 其硬件执行代价可近似表示为候选子图关键路径的组合延迟。记 $G[S]$ 中所有有向路径集合为 $\mathcal{P}(S)$, 则有

$$D_{hw}(S) = \max_{\pi \in \mathcal{P}(S)} \sum_{v \in \pi} d(v). \quad (2.14)$$

该定义主要刻画候选模式映射为专用数据通路后的关键路径时延，未显式计入接口控制逻辑及寄存器开销。

由此，候选子图的静态收益定义为软件执行代价与硬件执行代价之差：

$$\text{Gain}(S) = D_{sw}(S) - D_{hw}(S). \quad (2.15)$$

当 $\text{Gain}(S)$ 越大时，说明该候选模式在理论上越可能通过硬件聚合的方式获得性能收益。

然而，仅考虑候选模式的静态结构收益，仍不足以全面衡量其实际价值。原因在于，不同候选模式在真实应用运行过程中被触发的频率并不相同：某些结构上具有较好加速潜力的模式，若动态执行次数较低，则其实际贡献可能有限。为此，本文在静态收益基础上进一步引入动态热度信息，对候选模式进行加权评价。

设 $h(v)$ 表示节点 v 在目标程序动态执行过程中被触发的归一化频次或执行次数权重，则候选子图 S 的平均热度定义为

$$H(S) = \frac{1}{|S|} \sum_{v \in S} h(v). \quad (2.16)$$

其中， $h(v)$ 反映的是节点 v 在目标程序动态执行过程中的相对活跃程度。于是，在静态收益 $\text{Gain}(S)$ 的基础上，可将候选模式的综合评价写为

$$W(S) = \text{Gain}(S) \cdot (1 + \beta H(S)), \quad (2.17)$$

其中， β 为热度权重系数，用于调节动态信息对总体评价的影响程度。

上述定义表明，本文并不单纯追求结构上可加速的候选模式，而是更关注那些同时具有较好静态收益与较高动态重要性的候选子图。换言之，动态加权收益体现了对“是否值得优先扩展为定制指令”的综合判断。

2.4 定制指令识别问题建模

综合前文对候选子图表示、可实现性约束及综合评价函数的定义，本文将定制指令识别问题抽象为一个受约束的 Top- K 候选识别问题。

对于任意候选节点集合 $S \subseteq V$ ，若其诱导子图 $G[S]$ 同时满足合法操作约束、连通性约束、凸性约束、输入输出规模约束以及面积预算约束，则称其为一个可行候选子图。进一步地，定义所有可行候选子图构成的集合为

$$\mathcal{F} = \{S \subseteq V \mid S \text{ 满足所有可实现性约束}\}. \quad (2.18)$$

对于任意 $S \in \mathcal{F}$, 本文采用式 (2.17) 定义的综合评价函数 $W(S)$ 对候选子图的扩展价值进行度量。其中, $\text{Gain}(S)$ 刻画候选模式在静态结构层面的潜在收益, $H(S)$ 则反映其在目标应用运行过程中的动态重要性, β 为热度调节系数。由此, $W(S)$ 越大, 表明该候选子图越值得优先扩展为定制指令。

需要指出的是, 本文关注的问题并非仅从可行候选集合中选取单个最优子图, 而是希望从中识别并维护综合评价最高的前 K 个候选子图, 以为后续指令扩展决策提供更充分的候选空间。因此, 本文将定制指令识别问题表述为如下 Top- K 候选识别问题: 在可行候选集合 $\mathcal{F}$ 中, 寻找一个候选集合 $\mathcal{C}_K$, 使其满足

$$\mathcal{C}_K \subseteq \mathcal{F}, \quad |\mathcal{C}_K| = K, \quad (2.19)$$

并且

$$\forall S_i \in \mathcal{C}_K, \forall S_j \in \mathcal{F} \setminus \mathcal{C}_K, \quad W(S_i) \geq W(S_j). \quad (2.20)$$

上述定义表明, 定制指令识别的核心任务在于: 在满足硬件可实现性约束的前提下, 对程序数据流图中的候选局部计算模式进行有效搜索、评价与排序, 并保留其中最具扩展价值的前 K 个候选子图。然而, 可行候选空间的规模随程序图节点数呈指数级增长, 穷举式搜索在实际规模的程序上不可行。为此, 本文在上述问题建模的基础上, 设计了一种基于图同构网络 (Graph Isomorphism Network, GIN)^[51] 和近端策略优化 (Proximal Policy Optimization, PPO)^[52] 的候选定制指令搜索算法, 下文将进行详细介绍。

2.5 算法总体框架

本文将候选定制指令识别问题建模为一个带约束的马尔可夫决策过程 (Markov Decision Process, MDP)。给定程序的数据流图 (DFG), 智能体从空节点集合出发, 逐步构建候选子图。

在该 MDP 中, 状态由当前已选择的子图以及原始 DFG 共同决定, 动作空间由当前可扩展的合法节点及停止动作构成。在每一步决策中, 策略网络基于当前状态对候选动作进行评分, 并在合法动作集合内选择一个动作执行, 从而实现对于子图的增量式构建。具体而言, 动作包括两类: (1) 从 DFG 中选择一个节点加入当前候选子图; (2) 执行停止动作以终止当前回合。

与传统穷举搜索或启发式合并方法相比, 本文方法通过将候选子图构造过程建模为序列决策问题: 一方面利用 GIN 对 DFG 的局部依赖结构进行编码, 使

模型能够感知节点的上下文拓扑关系；另一方面利用 PPO 对节点扩展过程进行策略优化，使候选子图的构造面向长期累计收益进行搜索，而非依赖固定规则。

算法整体框架如图2.2所示，可分为以下四个阶段：

- (1) 以 LLVM IR 为中间表示，构建程序的数据流图，并为每个节点提取静态特征；
- (2) 基于程序动态执行轨迹统计指令执行次数，并映射为节点级动态特征；
- (3) 采用图神经网络（GIN）对当前状态进行编码，得到节点表示及全局图表示；
- (4) 在约束条件下，基于近端策略优化算法（PPO）逐步选择节点，生成候选定制指令子图。

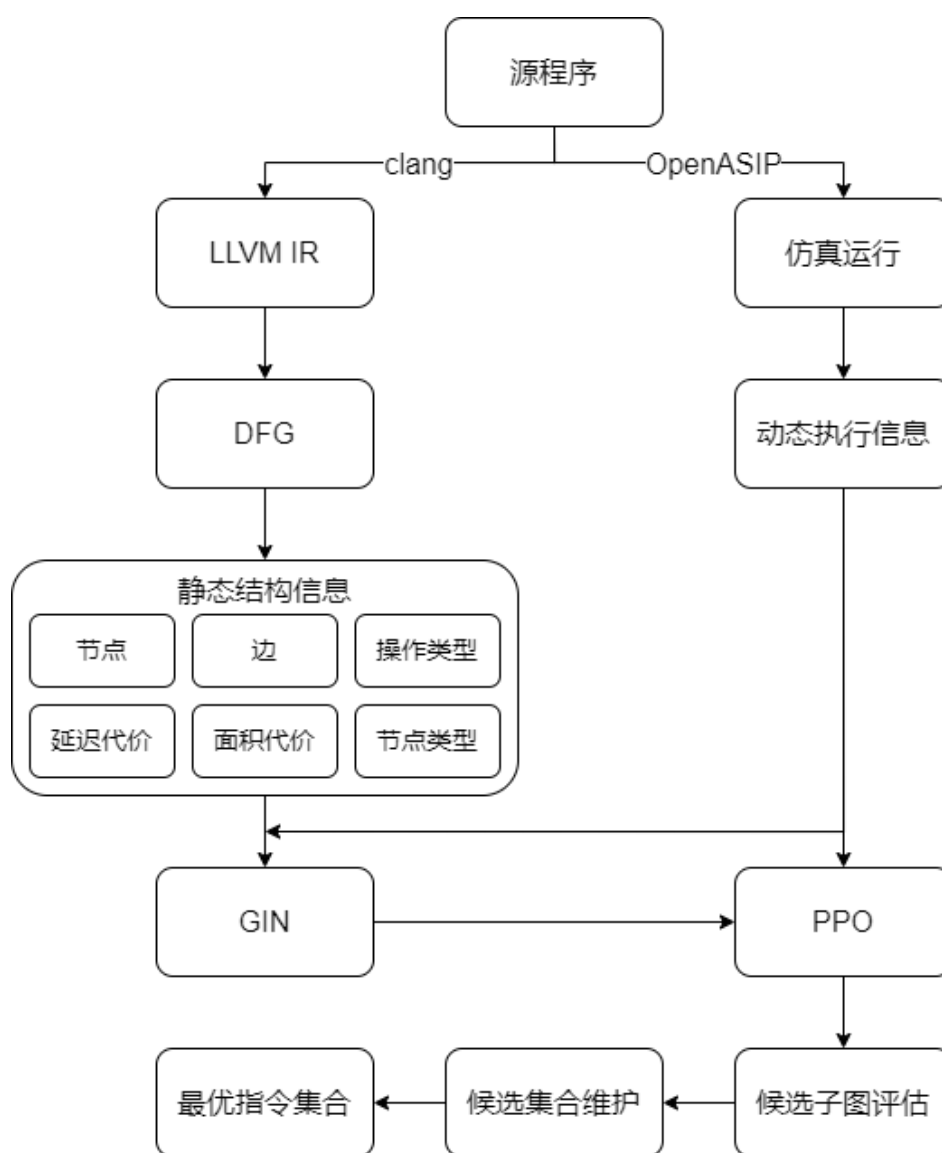


图 2.2 算法框架图

需要进一步说明的是,GIN 与 PPO 在本文方法中并非两个彼此独立的模块,而是分别承担“状态表征”和“决策优化”的功能。定制指令识别的动作对象是 DFG 中的节点,而节点是否适合被加入当前候选子图,取决于其操作属性、硬件代价、动态热度以及与已选节点之间的拓扑依赖关系。GIN 通过邻域聚合机制对这些结构化信息进行编码,将原始 DFG 状态映射为可供策略网络使用的节点表示和全局表示;PPO 则在该表示基础上学习节点选择策略,使智能体能够在合法动作约束下逐步扩展候选子图。

换言之,GIN 解决的是“如何理解当前图状态”的问题,PPO 解决的是“在当前状态下如何选择下一步动作”的问题。二者的结合使搜索过程不再依赖固定启发式规则,而能够利用图结构表征结果进行面向长期收益的策略优化,从而更适合处理定制指令识别中结构约束强、搜索空间大、候选质量依赖上下文的问题。以下各节依次介绍算法的总体框架、状态表示与特征构建、图编码方法、策略优化以及候选维护机制。同时,在单个决策回合内,原始 DFG 的拓扑结构保持不变,而状态的变化体现在节点选择掩码或子图标记的动态更新。因此,策略网络的输入不仅包含节点的静态与动态属性,还包含当前子图的结构信息,从而刻画决策过程的历史依赖性。

2.6 状态表示与输入构建

2.6.1 输入图表示

本文算法以程序数据流图 $G = (V, E)$ 作为输入,其中 V 为节点集合, E 为有向边集合。图中节点表示程序中的运算操作或边界输入源,有向边表示显式数据依赖关系。若 $(u, v) \in E$,则表示节点 v 的计算依赖于节点 u 的输出。

需要说明的是,本文使用的输入图并非直接调用编译器现成 DFG 分析结果得到,而是以 LLVM IR 为中间表示,通过自定义图构建脚本按规则生成。具体而言,首先将源程序编译为 LLVM IR;随后,根据需要可进一步执行 `mem2reg`、`instcombine`、`simplifycfg` 和 `instnamer` 等规范化处理;最后对 IR 文本进行解析,识别指令、操作数及其显式数据依赖关系,并生成供后续训练使用的 DFG JSON 文件。

为了与后续训练管线保持一致,图构建过程对节点类型和操作码进行了任务相关的规范化处理:一方面,仅保留训练阶段支持的操作类别,并将语义相近的操作统一映射到同一类别;另一方面,对常量、全局变量、函数参数以及跨基

本块传入的外部 SSA 值, 不直接作为可选运算节点处理, 而是构造为边界占位节点, 用于保留候选子图的输入边界信息。由此得到的图表示更加贴合本文候选定指令识别任务的需求。

2.6.2 节点属性表示

根据实现, 本文为每个节点构造一个 8 维属性向量:

$$f(v) = [\text{lat}(v), \text{area}(v), \mathbb{I}_{inst}, \mathbb{I}_{ext}, \mathbb{I}_{const}, \mathbb{I}_{global}, \mathbb{I}_{arg}, h(v)]. \quad (2.21)$$

其中, $\text{lat}(v)$ 和 $\text{area}(v)$ 分别对应问题建模中定义的节点延迟代价 $d(v)$ 与面积代价 $a(v)$; $\mathbb{I}_{inst}$ 和 $\mathbb{I}_{ext}$ 用于区分节点是实际指令节点还是边界输入节点; $\mathbb{I}_{const}$ 、 $\mathbb{I}_{global}$ 和 $\mathbb{I}_{arg}$ 分别表示该节点是否来源于常量、全局变量以及参数或外部 SSA 输入; $h(v)$ 表示节点 v 对应指令的动态执行次数经归一化后得到的相对执行频率。需要说明的是, 各类操作的延迟和面积开销均由查表方式获得, 具体数据来源于表 2.1^[36,53]。该表基于 $0.13 \mu\text{m}$ CMOS 工艺, 对 32 位 MAC 操作进行归一化 (设其延迟和面积为 1.00), 并给出了不同操作类型在相同数据宽度下的归一化延迟和面积参数。若节点不是实际运算指令, 则其延迟、面积和动态活跃度均置为 0。这种特征设计同时保留了节点的硬件代价信息、来源类型信息以及动态执行信息, 能够为后续图编码和动作决策提供更完整的输入。

表 2.1 $0.13 \mu\text{m}$ CMOS 工艺中基本操作的归一化延迟和面积

操作	数据宽度	归一化延时	归一化面积
乘加运算 (MAC)	$32 \times 32 + 64$	1.00	1.00
加法器 (Adder)	$32 + 32$	0.31	0.10
乘法器 (Multiplier)	32×32	0.96	0.98
除法器 (Divider)	$32/32$	7.45	2.48
桶形移位器 (Barrel Shifter)	32	0.23	0.12
按位与/或 (Bitwise AND/OR)	32	0.03	0.02

2.6.3 动态活跃度构建

为兼顾静态可实现性与动态运行价值, 本文引入节点动态活跃度 $h(v)$ 。与基于操作类型统计的方法不同, 本文直接基于程序动态执行轨迹统计每条指令实例的实际执行次数, 从而获得与 DFG 节点一一对应的动态信息。

具体而言,从程序运行过程中采集动态指令序列,并通过指令标识(如 PC 或 IR 映射关系)将每次执行映射回 DFG 中的节点。设节点 v 对应的指令在动态轨迹中的执行次数为 $c(v)$,则其归一化动态活跃度定义为

$$h(v) = \frac{c(v)}{\sum_{u \in V_{inst}} c(u)}, \quad (2.22)$$

其中 V_{inst} 表示所有实际指令节点集合。

对于常量、全局变量和外部输入等非指令节点,由于其不对应独立动态执行行为,本文令

$$h(v) = 0. \quad (2.23)$$

该方法本质上是一种基于动态轨迹统计的热点刻画方式。虽然其形式相对简单,但能够以较低代价为节点提供反映程序实际运行特征的附加信息,并与后续奖励计算保持一致。

2.6.4 节点状态编码

除静态属性外,策略网络还需要感知当前候选子图的构造进度。为此,本文为每个节点设置一个 4 维状态向量:

$$z_t(v) = [z_{unvisited}, z_{selected}, z_{invalid}, z_{masked}]. \quad (2.24)$$

其中, **unvisited** 表示节点尚未被选中且本身属于可参与选择的有效节点; **selected** 表示节点已被加入当前候选子图; **invalid** 表示节点本身不允许纳入候选定制指令,如访存、控制流、调用类节点以及边界占位节点; **masked** 表示节点并非永久非法,但在当前时刻因不满足约束而暂时不可选。

因此,当前环境状态可由图结构、节点属性和节点状态共同表示。与仅依赖静态程序图相比,这种表示方式能够更准确地反映候选子图当前的扩展情况及剩余可行动作空间。

2.7 基于 GIN 的图状态编码

2.7.1 初始节点表示

在每个决策时刻,本文首先将节点属性向量和节点状态向量映射到统一隐藏空间。对于节点 v ,其初始表示定义为

$$h_v^{(0)} = \alpha_z W_z z_t(v) + \alpha_f W_f f(v), \quad (2.25)$$

其中, W_z 和 W_f 为可学习参数矩阵, α_z 与 α_f 分别为状态特征和属性特征的融合系数。通过这种方式, 模型在图编码开始前即可同时获得节点的硬件代价、来源类型、动态活跃度以及当前选择状态。

2.7.2 GIN 消息传递

由于候选定制指令识别本质上是局部依赖子图的选择问题, 模型不仅需要区分节点操作类型, 还需要刻画节点所处的数据依赖结构。基于此, 本文采用图同构网络 (Graph Isomorphism Network, GIN) 作为图编码器对程序数据流图进行表示学习。

设第 k 层的节点表示为 $h_v^{(k)}$, 则 GIN 的邻域聚合过程可表示为

$$m_v^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in \mathcal{N}_{in}(v)} h_u^{(k-1)} \right), \quad (2.26)$$

其中, $\mathcal{N}_{in}(v) = \{u \mid (u, v) \in E\}$ 表示在数据流图中指向节点 v 的前驱依赖节点集合, 即所有为节点 v 提供输入数据的节点; $\epsilon^{(k)}$ 为可学习参数, 用于调节中心节点特征与邻域信息之间的权重关系; $\text{MLP}^{(k)}(\cdot)$ 表示第 k 层的多层感知机, 用于对聚合后的特征进行非线性变换。

为在多层消息传递过程中保留节点的初始属性信息, 并缓解表示过度平滑 (over-smoothing) 问题, 本文在每一层 GIN 聚合输出后引入初始表示的跳跃连接 (skip connection)。具体而言, 将当前层的聚合结果与节点的初始表示 $h_v^{(0)}$ 相加, 并通过双曲正切函数进行非线性变换, 得到第 k 层的最终节点表示:

$$h_v^{(k)} = \tanh(m_v^{(k)} + h_v^{(0)}). \quad (2.27)$$

这种设计使得节点表示在逐层融合邻域信息的同时, 始终保留其原始属性与状态特征, 从而在增强表达能力的同时提升训练稳定性。经过多层传播后, 每个节点的表示不仅包含自身特征, 还融合了多跳邻域的依赖上下文信息。

2.7.3 全局上下文表示

在节点选择过程中, 仅依据单个节点嵌入进行判断是不充分的, 因为一个节点是否值得加入还取决于当前已选子图的整体状态。为此, 本文进一步构造全局上下文表示。

首先, 对所有节点嵌入取平均, 得到全图表示:

$$h_G = \frac{1}{|V|} \sum_{v \in V} h_v. \quad (2.28)$$

其次,对当前已选节点集合 S_t 对应的嵌入求和,得到当前候选子图表示:

$$h_{CI} = \begin{cases} \sum_{v \in S_t} h_v, & S_t \neq \emptyset, \\ 0, & S_t = \emptyset. \end{cases} \quad (2.29)$$

然后,将二者加权融合为当前决策所需的上下文表示:

$$\hat{h}_G = \alpha_{ci} h_{CI} + (1 - \alpha_{ci}) h_G. \quad (2.30)$$

对于每个节点动作,将节点表示与全局上下文拼接,得到动作评分输入:

$$\bar{h}_v = [h_v \parallel \hat{h}_G]. \quad (2.31)$$

据此可知,节点动作依赖于节点局部特征与全局状态,而停止动作与价值估计则主要基于全局表示。对于停止动作,本文基于当前全局上下文表示构造停止动作特征 $\bar{h}_{stop}$,并与节点动作共同参与后续策略评分与价值估计。

2.8 基于 PPO 的节点选择策略

2.8.1 动作空间与合法动作约束

在任一时刻,智能体的动作包括两类:一类是从图中选择一个节点加入当前候选子图,另一类是执行停止动作结束当前回合。若图中共有 n 个节点,则动作空间可写为

$$\mathcal{A} = \{1, 2, \dots, n, \text{stop}\}. \quad (2.32)$$

但并非所有动作在任意时刻都合法,本文通过合法动作掩码对不可行动作进行屏蔽,主要约束包括:

- (1) **节点合法性约束**: 仅允许选择可融合的运算类指令节点,访存、控制流、调用类和外部节点均不可选;
- (2) **邻接性约束**: 当当前候选子图非空时,新加入节点必须与已选节点至少存在一条直接依赖边;
- (3) **凸性约束**: 加入新节点后,子图内部依赖路径不能穿越未选中的中间节点;
- (4) **输入输出约束**: 子图输入数与输出数不得超过预设上限;
- (5) **面积约束**: 子图总面积不能超过面积预算;
- (6) **连通性约束**: 候选子图必须保持连通。

在实现中,不合法动作对应的 logit 被置为极小值,从而使其采样概率趋近于 0。这种方式避免了对非法动作再额外设计惩罚项,也能够显式保证生成结果满足约束。

2.8.2 策略网络与价值网络

对每个节点动作, 策略网络根据其动作表示 $\bar{h}_v$ 输出对应评分:

$$\ell_v = f_{\theta}^{act}(\bar{h}_v), \quad (2.33)$$

其中, f_{θ}^{act} 为动作评分网络, ℓ_v 为节点 v 的动作 logit。

对于停止动作, 本文根据当前全局表示构造停止动作表示, 并计算其 logit:

$$\ell_{stop} = f_{\theta}^{act}(\bar{h}_{stop}). \quad (2.34)$$

对全部动作评分进行掩码处理后, 通过 Softmax 得到策略分布:

$$\pi_{\theta}(a_t|s_t) = \text{Softmax}(\ell_t^{masked}). \quad (2.35)$$

同时, 本文构建价值网络估计当前状态的状态价值:

$$V_{\phi}(s_t) = f_{\phi}^{value}(\bar{h}_{stop}), \quad (2.36)$$

其中, f_{ϕ}^{value} 为价值函数网络。价值网络为策略梯度提供基线, 用于降低训练方差。

2.8.3 奖励设计

根据本文第2.3节给出的综合评价函数 $W(\cdot)$, 本文在强化学习训练中采用增量奖励形式。设当前候选子图为 S_t , 执行动作后得到新子图 S_{t+1} , 则即时奖励定义为

$$r_t = W(S_{t+1}) - W(S_t), \quad (2.37)$$

其中

$$W(S) = \text{Gain}(S) \cdot (1 + \beta H(S)), \quad (2.38)$$

$\text{Gain}(S)$ 与 $H(S)$ 的定义见第2.3节, β 为动态权重系数。

当策略选择停止动作时, 若当前子图规模小于最小节点数阈值, 则给予负奖励; 否则停止奖励为 0。这种处理能够抑制过早终止, 促使策略生成具有实际意义的候选子图。

2.8.4 PPO 优化目标

本文采用 PPO 对策略网络进行更新。设旧策略参数为 θ_{old} , 当前策略参数为 θ , 则概率比定义为

$$\rho_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}. \quad (2.39)$$

在优势函数估计为 $\hat{A}_t$ 时, PPO 的裁剪目标为

$$L^{clip}(\theta) = \mathbb{E}_t \left[\min \left(\rho_t(\theta) \hat{A}_t, \text{clip}(\rho_t(\theta), 1 - \epsilon_{clip}, 1 + \epsilon_{clip}) \hat{A}_t \right) \right]. \quad (2.40)$$

同时, 价值网络损失为

$$L^{value}(\phi) = \mathbb{E}_t \left[(V_\phi(s_t) - \hat{R}_t)^2 \right], \quad (2.41)$$

策略熵为

$$\mathcal{H}(\pi_\theta) = - \sum_{a \in \mathcal{A}} \pi_\theta(a|s_t) \log \pi_\theta(a|s_t). \quad (2.42)$$

因此, 最终优化目标写为

$$L = -L^{clip} + c_v L^{value} - c_e \mathbb{E}_t[\mathcal{H}(\pi_\theta)], \quad (2.43)$$

其中, c_v 和 c_e 分别为价值损失系数和熵正则系数。

为了兼顾偏差与方差, 本文采用广义优势估计 (GAE) 计算优势函数。设折扣因子为 γ , GAE 衰减系数为 λ , 时间差分误差为

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \quad (2.44)$$

则优势函数估计为

$$\hat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}. \quad (2.45)$$

相应地, 目标回报定义为 $\hat{R}_t = \hat{A}_t + V_\phi(s_t)$ 。

2.8.5 候选集合维护与 Top-K 保留机制

需要指出的是, 强化学习训练过程中每个回合仅生成一个候选子图, 但不同回合可能得到多个满足约束且收益较高的候选定制指令。若仅保留单一最优结果, 易忽略其他高质量候选。基于此, 本文在训练过程中设计了候选集合维护机制, 在记录当前全局最优候选的同时, 额外维护一个按综合评分排序的 Top-K 候选集合。

具体而言, 设当前候选子图为 S , 以前文定义的综合评价 $W(S)$ 作为排序依据。当一个回合结束后, 首先对当前候选子图执行合法性检查; 若通过合法性检查, 则作为有效候选参与排序。

为避免重复记录相同候选, 本文使用候选子图对应的节点集合及其来源图标识作为唯一键进行去重。对于每个新产生的合法候选, 若其尚未出现, 则与当

前 Top-K 集合中的最低分候选进行比较：当候选集合大小不足 K 时直接插入；当集合已满且当前候选评分高于集合中最低评分项时，用其替换最低评分候选。由此可在训练全过程中持续维护一个固定规模的高质量候选集合。

该机制具有两点作用：其一，能够缓解强化学习采样带来的波动性，避免最终结果过度依赖单次最优回合；其二，能够为后续定制指令筛选、硬件实现评估及消融分析提供更丰富的候选基础。因此，本文最终输出不仅包括全局评分最高的候选定制指令，还包括一个按评分排序的 Top-K 候选集合。

2.9 算法流程总结

综合上述设计，本文方法从程序数据流图出发，通过强化学习逐步构造高质量的候选定制指令子图。

首先，对目标程序进行分析，提取其数据流图 $G = (V, E)$ ，并为每个节点构造静态属性 $f(v)$ ；同时结合动态执行轨迹，统计各节点对应指令的执行频率，得到节点的动态活跃度特征 $h(v)$ 。在此基础上，初始化强化学习环境，以空子图作为初始状态。

在每一个决策步中，算法根据当前候选子图状态以及节点的静态与动态属性构造状态表示，并利用图神经网络 (GIN) 对当前图结构进行编码，得到节点嵌入及全局图表示。随后，在满足结构约束与合法性条件的前提下生成动作掩码，用于限定当前可选的扩展节点集合。策略网络基于当前状态输出动作概率分布，并通过采样选择下一个扩展节点或终止操作。

若选择扩展节点，则将对对应节点加入当前子图，并依据综合评价函数 $W(S)$ 的增量计算即时奖励；若选择终止动作，则根据当前子图整体性能计算终止奖励并结束当前回合。完成一条轨迹后，利用广义优势估计 (GAE) 计算优势函数与目标回报，并基于 PPO 目标函数对策略网络和价值网络进行更新。

上述过程在多个训练回合中反复执行。对于每一轮生成的候选子图，算法进一步进行合法性检查与性能评估；若候选满足约束条件且规模合理，则用于更新全局最优解，并根据综合评分维护一个按优先级排序的 Top-K 候选集合 C_K 。该候选集合不仅能够避免算法仅依赖单一最优解，还可为后续定制指令的硬件实现和指令编码设计提供更多可选方案。最终，算法输出全局最优候选定制指令 C^* 以及高质量候选集合 C_K 。

该过程不仅利用数据流图刻画指令间的数据依赖关系，还通过动态活跃度

信息引导搜索过程更加关注程序中的高频执行区域，从而提高候选定制指令对实际性能提升的贡献。与仅依赖静态图结构的搜索方法相比，本文方法能够在保证合法性的前提下兼顾结构质量、执行收益与搜索效率。为便于说明，算法伪代码如算法1所示。

算法 1: 基于 GIN 和 PPO 的候选定制指令搜索算法

Input: 程序 DFG $G = (V, E)$, 动态轨迹统计结果, 训练轮数 N

Output: 最优候选 C^* 及 Top-K 集合 $\mathcal{C}_K$

```

1  构造节点属性  $f(v)$  与动态活跃度  $h(v)$ ;
2  初始化模型参数与候选集合  $\mathcal{C}_K$ ,  $C^* \leftarrow \emptyset$ ;
3  for  $iter \leftarrow 1$  to  $N$  do
4      初始化候选子图  $S \leftarrow \emptyset$ ;
5      for  $t \leftarrow 1$  to  $T_{\max}$  do
6          生成合法动作掩码  $M_t$ ;
7          if  $M_t$  为空 then
8              break
9          end
10         用 GIN 编码当前状态, 得到节点与全局表示;
11         根据策略网络采样动作  $a_t$ ;
12         if  $a_t = stop$  then
13             if  $|S| < S_{\min}$  then
14                 给予负奖励
15             end
16             break
17         else
18             将节点  $a_t$  加入  $S$ ;
19             按增量形式计算奖励  $r_t = W(S_{t+1}) - W(S_t)$ ;
20         end
21     end
22     基于轨迹计算优势函数并用 PPO 更新模型;
23     if  $S$  合法且  $|S| \geq S_{\min}$  then
24         更新最优候选  $C^*$  与 Top-K 集合  $\mathcal{C}_K$ ;
25     end
26 end
27 return  $C^*, \mathcal{C}_K$ ;
    
```

2.10 本章小结

本章围绕面向 RISC-V 定制指令扩展的候选识别问题，完成了从问题建模、收益评价到搜索算法设计的系统化描述。

在问题建模方面,本章以程序数据流图作为统一表示形式,将候选定制指令建模为数据流图上的诱导子图,并从硬件可实现性角度建立了多维约束体系。具体而言,候选子图需要同时满足合法操作约束、连通性约束、凸性约束、输入输出规模约束以及面积预算约束,从而保证其具备映射为定制指令的基本条件。在此基础上,本章进一步定义了候选子图的软件执行代价、硬件执行代价和静态收益,并引入节点动态活跃度对候选模式进行加权评价,形成兼顾结构收益与运行热点的综合评价函数。基于上述定义,本文将定制指令识别统一表述为受约束的 Top- K 候选子图识别问题,为后续算法设计提供了明确的优化目标。

在搜索方法方面,本章将候选子图的构造过程建模为带约束的马尔可夫决策过程,并设计了基于 GIN 和 PPO 的候选定制指令搜索算法。该算法以 LLVM IR 构建的数据流图为输入,结合节点静态属性、硬件代价、动态活跃度以及当前子图状态构造节点特征;随后利用 GIN 对程序数据依赖结构进行编码,获得节点级表示与全局上下文表示;在此基础上,策略网络根据合法动作掩码逐步选择扩展节点或执行停止动作,并通过 PPO 对节点选择策略进行优化。该设计使搜索过程能够在显式满足约束的前提下,面向长期累计收益生成高质量候选子图。

此外,本章还设计了候选集合维护与 Top- K 保留机制。通过对每轮生成的候选子图进行合法性检查、评分排序与去重维护,算法不仅输出单个全局最优候选,还能够持续保留多个综合评价较高的候选定制指令,从而缓解强化学习采样波动带来的不确定性,并为后续指令扩展筛选、硬件实现分析和实验对比提供更充分的候选基础。总体而言,本章建立了从程序数据流图到候选定制指令的形式化映射关系,并提出了相应的学习型搜索方法,为后续章节中的实现验证与性能评估奠定了方法基础。

3 面向 RISC-V 定制指令扩展的自动化实现系统

本章在第二章问题建模与算法设计的基础上,介绍本文方法的工程实现。围绕候选定制指令识别算法的输出结果,重点说明从程序分析、候选操作生成,到扩展操作实现、编译映射以及处理器生成与仿真验证的完整流程。通过本章所述实现,构建了一套支持定制指令自动识别与处理器扩展落地的实验平台,为后续性能评估提供实现基础。

3.1 系统总体框架

为验证本文提出的候选定制指令识别方法,并实现从程序分析、候选子图选择、扩展操作生成、编译映射到处理器生成与仿真评测的完整闭环,本文设计并实现了一套面向 RISC-V 专用处理器扩展的自动化实现系统。系统总体框架如图3.1所示。

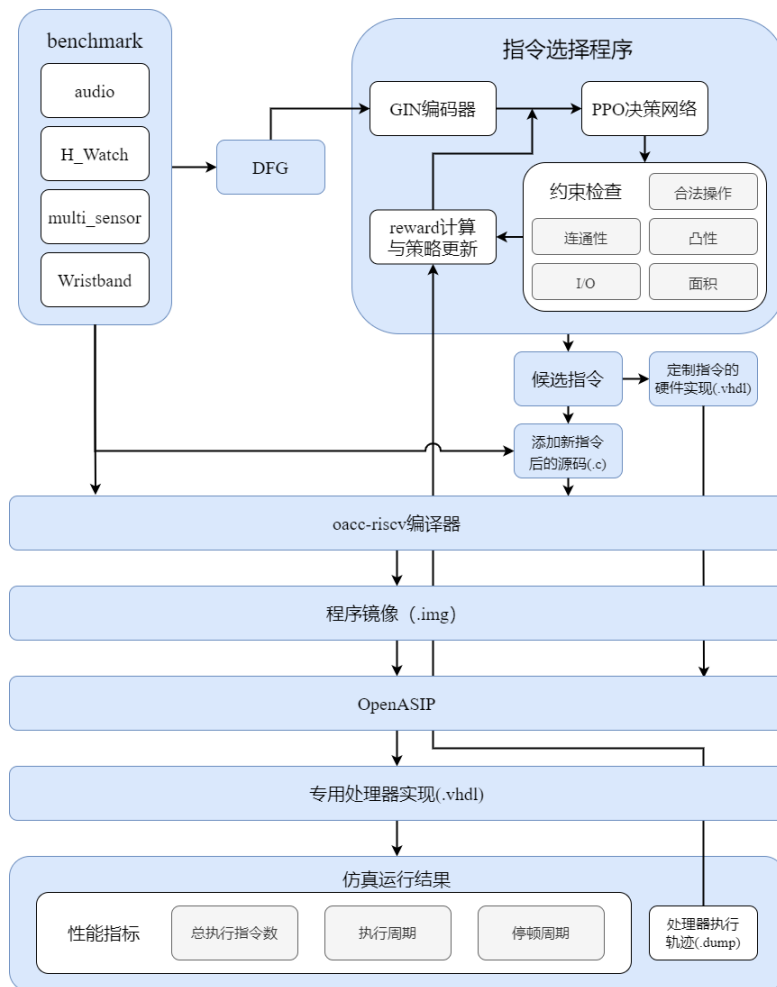


图 3.1 系统总体框架图

该系统以基准任务程序的 C 源代码为输入, 首先通过静态分析提取程序的数据流图 (Data Flow Graph, DFG), 并结合运行阶段采集的动态执行信息, 共同构成候选定制指令识别的输入。随后, 基于 DFG 执行候选子图搜索与筛选, 得到满足约束条件且具有潜在性能收益的候选操作集合。最后, 将候选操作转换为可供硬件实现的扩展操作描述, 并借助 OpenASIP 工具链完成处理器生成与仿真评测^[16]。

从实现流程上看, 系统整体可划分为五个层次: 程序分析层、候选操作生成层、扩展操作实现层、编译映射层以及处理器生成与验证层。各层之间通过统一的数据结构进行衔接, 形成从算法输出到处理器执行验证的完整闭环。为进一步明确系统各层之间的数据流关系, 表3.1给出了各功能层的主要输入、处理内容与输出结果。该表表明, 系统各层之间通过明确的中间产物进行衔接, 逐步将算法输出转化为处理器生成工具链可接受的操作描述、结构配置与程序镜像, 从而支撑后续仿真验证。

表 3.1 系统各层次输入输出与功能描述

系统层次	输入	功能描述	输出
程序分析层	C 源程序或 LLVM IR 文件	调用 clang 生成 LLVM IR; 解析函数、基本块和指令; 构建数据依赖关系	JSON 格式的 DFG 结构化文件
候选操作生成层	DFG 文件、动态执行信息、约束参数	基于第二、三章的定制指令识别算法, 执行候选子图搜索、约束筛选与收益评估	候选子图集合及其操作节点和收益信息等
扩展操作实现层	候选子图集合	提取子图输入输出接口并构建操作语义, 以生成和配置扩展操作的 OSAL 操作定义与 VHDL 实现	OSAL 操作定义文件、VHDL 硬件实现文件
编译映射层	已调用扩展的源程序、ADF 文件、OSAL 操作定义文件	编译源程序, 将扩展操作调用映射为目标处理器支持的操作, 生成程序镜像	目标程序镜像文件 (.img 格式)
处理器生成与验证层	ADF 文件、IDF 文件、HDB 文件、程序镜像	生成处理器 VHDL; 初始化指令和数据存储器; 执行仿真并采集运行结果	处理器 VHDL 工程、仿真输出文件、性能指标文件

3.2 程序分析与数据流图构建

候选定制指令识别算法以程序的数据流图 (Data Flow Graph, DFG) 作为输入。因此,在执行候选子图搜索之前,需要对输入程序进行静态分析,并将其转换为能够刻画计算操作及数据依赖关系的图结构表示。本节介绍从 C 源程序到 DFG 的构建过程。

图3.2以一段示例 C 程序为例,展示了从源程序到数据流图的完整转换流程及各阶段的表示形式。

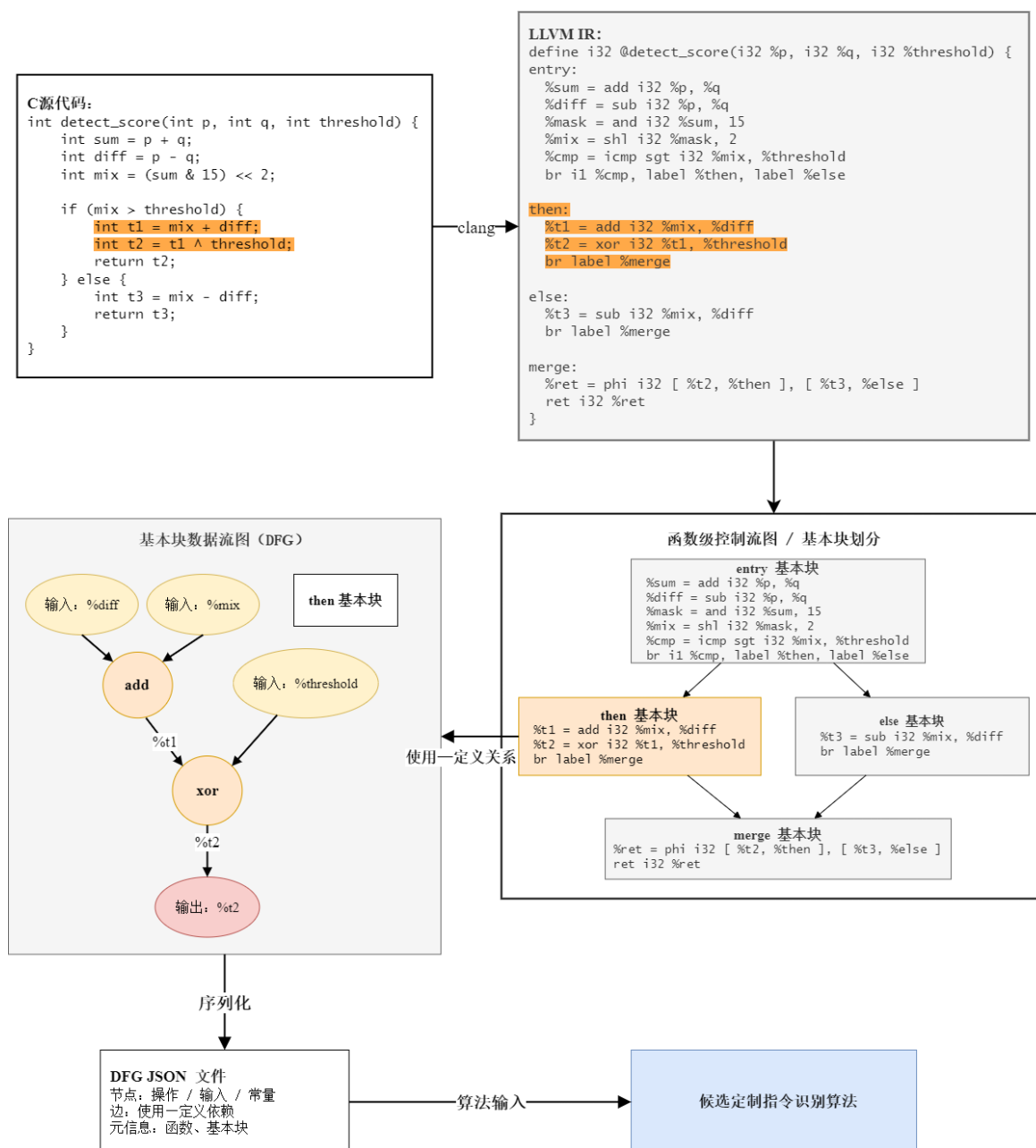


图 3.2 源程序到数据流图的表示映射

图中右上方为 C 源程序经 clang 编译后生成的 LLVM IR, 其中函数体被划分为多个基本块 (entry、then、else、merge); 右下方以控制流图形式展示了各基本块之间的跳转关系。系统选定目标基本块后, 对其中的指令序列进行数据依赖分析——图中以 then 基本块为例 (以橙色高亮标示), 系统基于 LLVM IR 中显式操作数引用关系近似提取使用一定义关系, 提取该基本块内各指令之间的数据传递关系, 构建出左侧所示的基本块级 DFG。图中 DFG 的计算节点 (add、xor) 对应 then 基本块中的算术与逻辑运算指令, 椭圆形节点表示外部输入的占位节点, 有向边表示操作数的数据依赖方向。最终, DFG 被序列化为 JSON 格式文件, 作为候选定制指令识别算法的输入。

本系统基于 LLVM 中间表示 (LLVM IR) 完成程序分析。输入可以是 C 源程序或已有的 LLVM IR 文件: 对于 C 程序, 系统通过 clang 自动生成对应的 LLVM IR; 对于 IR 文件, 则直接作为分析输入。采用 LLVM IR 作为统一中间表示, 有助于屏蔽源语言语法差异, 并为后续分析提供规范化的操作语义。在分析粒度上, 系统以函数为基本单位, 并支持在基本块级别构建局部 DFG。相较于全函数级图结构, 基本块级 DFG 能够在一定程度上规避复杂控制流对数据依赖分析的干扰, 使候选子图更集中于局部连续计算片段, 从而更符合定制指令融合的应用场景。

DFG 的节点用于表示程序中的有效计算操作。系统在解析 LLVM IR 指令时, 提取指令类型、操作数及其所在函数和基本块等信息, 并将受支持的指令转换为图节点。所关注的操作主要包括整数算术运算、移位运算、逻辑运算、比较运算、访存操作以及部分控制流相关指令。为减少操作类型差异对后续算法处理的影响, 系统对部分指令进行了统一化处理, 例如将比较类指令归一化为统一的比较操作, 将地址计算类指令归一化为地址生成操作。对于不直接影响计算语义的类型转换或别名类指令, 系统采用透明化处理方式, 将其结果重定向至原始操作数对应的节点, 从而避免在 DFG 中引入冗余结构。对于当前系统未支持的 LLVM IR 指令, 若其可视为简单值传递, 则同样进行别名折叠; 否则不纳入 DFG 构建过程, 以保证图结构与后续候选识别算法支持的操作集合一致。

DFG 的边用于刻画指令之间的数据依赖关系。系统基于 LLVM IR 中显式操作数引用关系近似提取使用一定义关系建立有向依赖边: 若某条指令的操作数由另一条指令产生, 则在对应节点之间添加一条依赖边。通过该方式, DFG

能够准确描述程序中计算结果的传递关系。对于立即数、函数参数以及全局变量等外部输入，系统不将其视为普通计算节点，而是作为外部输入进行处理。为保持图结构中输入边界的完整性，系统为这些外部值引入占位节点，并在节点属性中标记其来源类型，从而既保留依赖关系信息，又避免其被误识别为可融合的计算操作。

最终，系统将构建得到的 DFG 表示为 JSON 格式文件，包含节点集合、边集合以及所属函数和基本块等元信息。其中，节点记录操作类型、原始 LLVM IR 指令及操作数信息，边集合记录节点之间的数据依赖关系。该结构化 DFG 文件作为候选定制指令识别算法的输入，用于后续候选子图搜索、约束判断与收益评估。

通过上述过程，系统实现了从源程序到数据流图的自动化转换，为候选定制指令识别提供了统一且可解析的图结构表示。

3.3 候选子图到扩展操作的生成与实现

候选定制指令识别算法的输出为 DFG 上的一组候选子图。每个子图由若干运算节点及其内部数据依赖关系构成，用于描述一段具有潜在融合价值的局部计算模式。然而，该表示仍属于抽象图结构，尚不能直接用于处理器扩展与编译映射。因此，需要将候选子图进一步转换为可实现的扩展操作，并完成其在目标处理器中的集成。为此，系统将“候选子图到扩展操作”的过程划分为两个阶段：操作语义抽象与处理器实现集成。

在操作语义抽象阶段，系统将候选子图转换为统一的扩展操作描述。首先，根据子图与外部节点之间的数据依赖关系确定操作接口，即识别输入与输出边界。其中，来自子图外部且被内部节点使用的数据作为输入操作数，而子图内部生成并被外部节点使用的结果作为输出。其次，对子图内部节点执行拓扑排序，以恢复其计算执行顺序，从而保证操作语义的确定性。在此基础上，根据节点的操作类型构建扩展操作的功能语义描述，用于刻画子图内部的组合计算逻辑。最终，系统将操作接口、内部依赖关系及功能语义整合为统一的结构化表示，为后续处理器集成提供规范化输入。

需要指出的是，候选子图在生成阶段已满足连通性、凸性及资源约束等合法性条件，因此本阶段不再对其结构进行筛选，而是专注于完成从图结构到操作语义的转换。

在完成扩展操作描述后，系统基于 OpenASIP 工具链实现其在目标处理器中的集成。图3.3展示了候选子图到扩展操作的生成与集成流程。

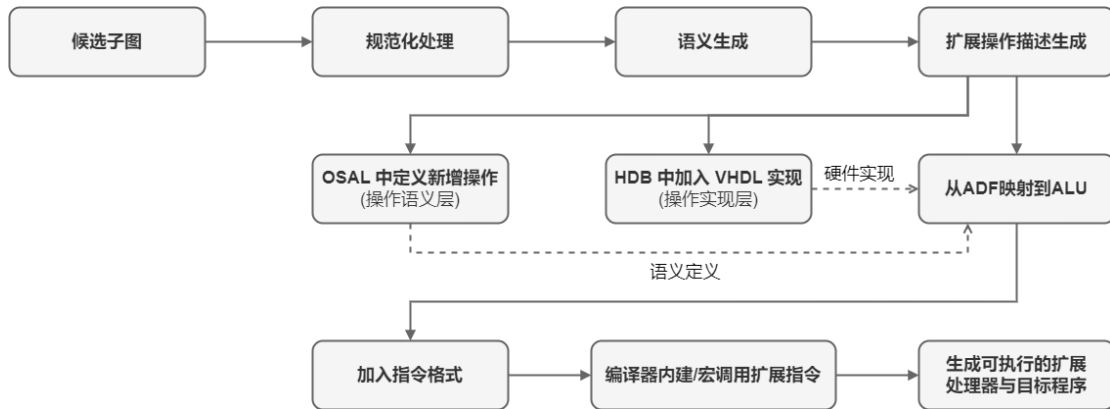


图 3.3 候选子图到扩展操作的生成流程

在具体实现中，系统首先根据扩展操作描述生成操作接口定义与功能语义，并在 OpenASIP 的操作抽象层（Operation Set Abstraction Layer, OSAL）中注册新增操作，使编译器与仿真器能够识别和调度这些操作。随后，为每个扩展操作提供对应的硬件实现描述（VHDL），并将其加入硬件数据库（Hardware Database, HDB），以支持操作的实际执行。

在处理器结构层面，通过修改结构描述文件（Architecture Definition File, ADF），将新增操作绑定到既有算术逻辑单元（ALU），并配置相应的操作端口及执行延迟属性。在不引入额外功能单元的前提下，该方法通过扩展 ALU 操作集合，使候选子图能够以组合操作的形式参与处理器执行。图3.4给出了扩展操作在处理器结构中的集成方式。

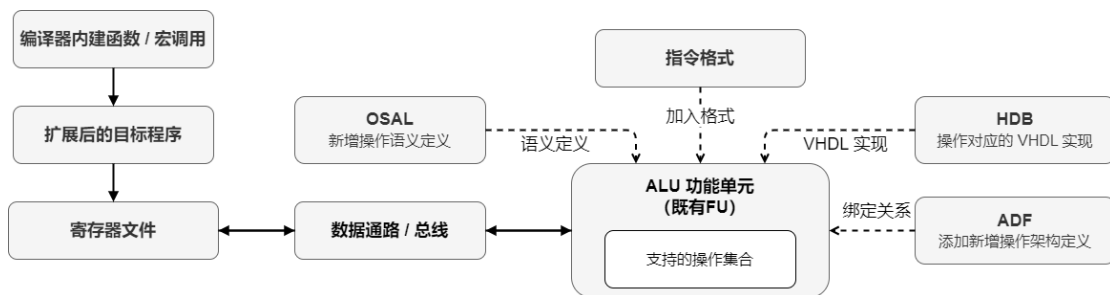


图 3.4 扩展操作在处理器结构中的集成方式

在上述结构集成的基础上，每个扩展操作均需提供对应的硬件实现描述。为说明扩展操作的具体硬件实现方式，以定点乘法扩展操作为例，其核心 VHDL 实现如表 3.2 所示。

表 3.2 定点乘法扩展操作的 VHDL 实现

```
op3 <= std_logic_vector(to_signed(2147483647, op3'length))
when shift_right(resize(signed(op1), 64) * resize(signed(op2),
64), 16) > to_signed(2147483647, 64) else
std_logic_vector(to_signed(-2147483648, op3'length))
when shift_right(resize(signed(op1), 64) * resize(signed(op2),
64), 16) < to_signed(-2147483648, 64) else
std_logic_vector(resize(shift_right(resize(signed(op1), 64)
* resize(signed(op2), 64), 16), op3'length));
```

该实现首先对两个输入操作数进行有符号扩展至 64 位并完成乘法运算，随后通过右移 16 位完成 Q16.16 定点格式下的缩放处理。为防止运算溢出，模块对缩放后的结果进行上下界判断以实现饱和截断，并将最终结果写入输出端口 op3。结合 OSAL 中的操作行为定义，该硬件模块能够在处理器执行阶段正确响应编译器生成的扩展操作调用。

通过上述过程，系统实现了从候选子图到扩展操作生成及其处理器集成的自动化转换，为后续编译映射与性能评估提供了基础。

3.4 编译映射与程序镜像生成

在完成扩展操作的处理器集成后，需要将源程序编译为目标处理器可执行的程序镜像。本文采用面向所构建目标处理器模型配置的 OpenASIP 编译工具链 (oacc) 完成应用程序编译。由于新增的扩展操作已通过操作抽象层 (OSAL) 注册到目标处理器的操作集合中，并在处理器结构描述文件 (ADF) 中完成了功能单元绑定，编译器能够在指令选择与调度阶段识别这些操作，并将其映射为目标处理器支持的操作序列。

为使扩展操作参与编译映射，需要在源程序中引入对应的操作调用。当前系统采用在源程序中显式插入扩展操作调用（如宏定义或内建接口）的方式，将候选子图对应的定制计算行为引入程序。该方式的主要目的是验证扩展操作在目标处理器上的功能正确性与性能表现。需要说明的是，由编译器自动识别源程序中的候选计算模式并完成替换的机制不在本文实现范围内，将作为后续工作进一步研究。

为直观说明扩展操作在源程序中的调用方式及其与原始计算之间的对应关系,本文以姿态解算中的偏航角(yaw)计算为例,给出操作替换前后的代码对比。该计算对应的表达式为 $\text{atan2}(2(qw \cdot qz + qx \cdot qy), 1 - 2(qy^2 + qz^2))$, 其内部包含多次乘法与加法运算,适合作为候选子图进行融合。

如表 3.3 所示,原始实现采用嵌套函数调用形式,通过多次调用 `q_mul` 和 `q_add` 逐步完成计算。如表 3.4 所示,在引入扩展操作后,原始代码中的定点乘法运算被逐一替换为 `_OA_QMUL` 扩展操作调用,从而将候选子图中识别出的乘法计算模式映射为处理器支持的专用操作。

表 3.3 原始偏航角计算代码

```
q16_16 num_yaw = q_mul(two, q_add(q_mul(qw,qz), q_mul(qx,qy)));
q16_16 den_yaw = q_sub(ONE, q_mul(two, q_add(q_mul(qy,qy),
q_mul(qz,qz))));
ypr[0] = q16_atan2(num_yaw, den_yaw);
```

表 3.4 插入扩展操作后的偏航角计算代码

```
_OA_QMUL(qw, qz, outputwz);
_OA_QMUL(qx, qy, outputxy);
_OA_QMUL(two, q_add(outputwz, outputxy), num_yaw);
_OA_QMUL(qy, qy, outputyy);
_OA_QMUL(qz, qz, outputzz);
_OA_QMUL(two, q_add(outputyy, outputzz), den_yaw1);
den_yaw = q_sub(ONE, den_yaw1);
ypr[0] = q16_atan2(num_yaw, den_yaw);
```

对比两段代码可以看出,原始代码中嵌套的 `q_mul` 函数调用被展开为独立的 `_OA_QMUL` 扩展操作调用。每个 `_OA_QMUL` 调用对应候选子图中的一个乘法节点:其输入参数对应子图的外部输入边界或前序节点的输出,计算结果作为中间值传递给后续运算。加法与减法运算(`q_add`、`q_sub`)以及最终的反正切运算(`q16_atan2`)仍保留原始的函数调用形式,体现了候选子图融合的粒度边界。编

在完成指令选择与调度后，编译器根据目标处理器的结构描述生成与之匹配的程序表示，并将结果输出为程序镜像文件（.img）。该镜像文件随后被用于初始化处理器的指令存储器与数据存储器，分别对应 `imem_init.img` 和 `dmem_data_init.img`，为后续的处理器的生成与仿真验证提供统一的程序输入。

在完成体系结构配置与程序编译后，系统进入处理器生成与验证阶段。本文基于 OpenASIP 提供的处理器生成工具 (Processor Generator, ProGe)，根据处理器结构描述 (Architecture Definition File, ADF) 及实现定义文件 (Implementation Definition File, IDF)，自动生成目标处理器的硬件描述 (VHDL) 以及对应的存储器初始化文件。处理器生成工具运行界面如图3.5所示。

图 3.5 处理器生成工具 (ProGe) 运行界面

表 3.5 处理器生成与仿真阶段主要输出文件

文件名称	作用说明
*.vhdl	处理器硬件描述文件, 即处理器的具体实现, 包含扩展操作的集成。
imem_init.img	指令存储器初始化文件, 用于加载目标程序指令。
dmem_data_init.img	数据存储器初始化文件, 用于初始化程序运行所需的数据段。
core0_imem_access_trace.dump	指令存储器访问轨迹文件, 用于分析程序执行过程中的取指行为。
riscv_instruction_trace.dump	指令执行轨迹文件, 用于统计指令执行情况和分析扩展操作是否被正确调用。
cycles.dump	总执行周期和停顿周期统计, 可用于分析处理器执行效率与流水线停顿情况。

通过上述流程, 系统实现了从扩展操作集成、处理器生成到程序执行与性能评估的完整闭环。

3.6 本章小结

本章围绕面向 RISC-V 定制指令扩展的自动化实现系统, 介绍了从候选子图生成到处理器验证的完整工程流程。首先, 给出了系统总体框架, 说明了各层之间的层次关系与数据流组织。其次, 介绍了基于 LLVM IR 的程序分析与数据流图构建过程, 为候选定制指令识别提供了统一的图结构输入。随后, 说明了候选子图向扩展操作的转换方法, 实现了从图结构模式到操作描述与硬件实现的生成。在此基础上, 进一步介绍了面向 OpenASIP 工具链的编译映射流程, 以及处理器生成与仿真验证过程。

通过上述设计, 本文构建了一个覆盖“候选识别—操作生成—操作集成—编译映射—处理器生成—仿真验证”的完整闭环系统, 使所提出的候选定制指令识别方法能够在实际处理器设计流程中得到验证。同时, 该系统也为后续开展更复杂场景下的定制指令自动化设计研究提供了可复现的工程基础。

4 实验结果与性能评估

4.1 基准任务说明

本章实验围绕智能装备检测场景展开,选取典型感知与计算任务构建评测基准(benchmark)。相较于通用计算基准,本工作更加关注嵌入式智能装备中的多模态数据处理特性,包括传感器数据解析、信号处理以及轻量级神经网络推理等计算模式。因此,所采用的基准程序均来源于实验室自研任务集,能够真实反映目标应用场景中的指令行为特征与计算热点。

具体而言,基准任务由四个典型应用程序构成,分别对应音频信号处理、健康监测、多传感器融合以及核心推理流程。首先,在音频处理任务中,实现了基于定点运算的FFT频谱分析及特征提取过程,其核心计算集中在蝶形运算与复数乘加操作上,涉及大量乘法、加减以及数据重排操作,是典型的信号处理密集型负载。其次,在手表传感任务中,涵盖心率估计、血氧计算以及步数检测等功能,算法中包含滤波、滑动平均及阈值判断等操作,其热点主要体现在整数滤波与状态更新逻辑,表现出明显的控制与算术混合特征。在多传感器融合任务中,主要处理姿态数据(四元数)并完成欧拉角转换,其计算过程涉及三角函数逼近、向量运算及坐标变换,热点算子集中在乘加组合及非线性函数近似上,具有较强的计算相关性和数据依赖性。此外,在腕带检测任务中,构建了面向腕带检测场景的轻量级卷积神经网络推理流程,主要包括特征输入、卷积层、激活函数、池化及全连接计算等模块,其中卷积操作(MAC运算)占据主要计算开销,是典型的数据并行密集型算子。

对应到本文实验程序中,音频处理(audio)程序实现基于定点运算的FFT频谱分析,手表传感(H_Watch)程序涵盖心率估计、血氧计算及步数检测功能,多传感器融合(multi_sensor)程序处理姿态数据并完成欧拉角转换,腕带检测(Wristband)程序构建轻量级卷积神经网络推理流程。这种组合不仅能够全面评估指令集扩展对不同类型计算负载的适应性,同时也为后续定制指令设计提供了具有代表性的热点算子支撑。

4.2 实验环境

本文实验流程分为两个阶段:候选指令选择与处理器体系结构评估。前者侧重程序分析与指令模式挖掘,后者基于OpenASIP平台完成指令集成与性能评

测^[16]。两阶段在工具链上相互独立，但在数据表示与指令语义上保持一致，从而保证实验流程的连贯性与结果的可复现性。

在候选指令选择阶段，算法运行于实验室服务器平台，采用 Python 实现，底层依赖 LLVM 工具链对基准程序进行分析。首先将 C 程序编译为 LLVM IR，并构建 DFG 以刻画算子间的数据依赖关系。在此基础上提取热点子图，并输入指令选择模型进行候选模式搜索。该阶段主要计算开销来自搜索过程，实验使用配备 NVIDIA GeForce RTX 5090 GPU 的服务器以提升候选搜索效率。

在体系结构建模与评估阶段，本文基于 OpenASIP 构建 RISC-V 处理器模型，并集成候选定制指令。通过修改架构描述文件 (ADF) 扩展功能单元与操作集合，并结合 OSAL 完成新增指令语义建模。应用程序通过面向该处理器模型配置的 OpenASIP 编译工具链 (oacc) 编译生成目标程序镜像，作为定制处理器输入。性能评估通过 OpenASIP 仿真工具完成，获取执行周期等关键指标。最终处理器模型由工具链自动生成硬件描述，并在统一配置下进行评测。

总体而言，本文构建了从程序分析、指令选择到体系结构验证的完整实验流程，使候选指令不仅在算法层面有效，同时具备实际处理器实现的可行性，从而保证实验结果的可信性。

4.3 对比实验

4.3.1 指令集架构约束对指令选择的影响

本文针对自建智能装备检测基准任务，进一步分析输入输出约束对指令选择过程的影响。实验在统一训练配置下，对不同输入输出约束策略进行联合训练，并统计前五个候选指令对应的奖励值（计算方式见本文2.8.3节），结果如图4.1所示。由于本文中奖励函数与性能指标通常呈正相关趋势，因此可以在一定程度上反映不同约束条件下处理器性能优化潜力的差异。

从图中可以观察到，当允许更灵活的多输入多输出约束时，模型能够探索更大的指令组合空间，从而更容易发现包含复杂算子融合的候选指令。这类指令通常能够覆盖更长的数据依赖链，减少中间结果的存储与访存开销，因此在性能上具有更大的优化潜力。相比之下，严格的输入输出限制虽然简化了指令格式，但会显著压缩搜索空间，使得模型倾向于选择简单算子组合，从而限制整体性能提升幅度。从体系结构角度分析，这一现象的本质在于输入输出约束直接决定了定制指令所能覆盖的数据流范围。较宽的输入输出接口允许在单条指令内完

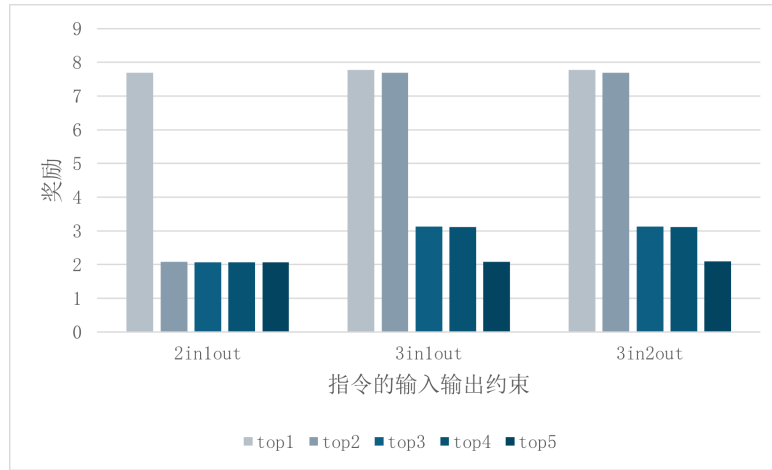


图 4.1 不同输入输出约束条件下的指令选择 reward 对比

成更多操作融合，从而降低指令间的数据传递与寄存器访问次数；而过于严格的约束则会将原本可以融合的操作拆分为多条指令执行，增加调度与访存开销。因此，在指令选择阶段引入合理的 ISA 约束，对于在“可实现性”与“性能收益”之间取得平衡至关重要。

此外，虽然已有工作^[46]同样采用 GIN 与 PPO 结合的方法进行指令选择，但其未考虑具体目标架构（如 RISC-V）的指令格式与编码约束，导致生成的大量候选指令难以直接应用于实际处理器设计。该问题在其原文中亦被指出，即后续需要结合完整的综合工具链以支持面向 RISC-V 等架构的实际部署。这一不足进一步说明，在指令选择过程中显式引入 ISA 约束，不仅能够提升搜索结果的可实现性，同时也是连接算法设计与硬件落地的关键环节。

4.3.2 定制指令扩展对处理器性能的影响

由于处理器扩展实现与仿真验证流程较为复杂，本文在自动识别得到的候选集合基础上，选取其中一条具有代表性的高收益候选指令进行原型集成与处理器验证，以验证所提方法的可落地性。该定制指令对应于基准程序中广泛调用的定点乘法函数 `q_mul`，其原始 C 语言实现如表4.1所示：

该函数实现了 Q16.16 定点格式下的带饱和截断乘法运算。其执行过程包含三个关键步骤：首先将两个 32 位定点操作数扩展为 64 位并完成乘法运算；随后对乘积结果进行 Q 位（16 位）右移以恢复定点小数对齐；最后执行饱和判断，将超出 32 位有符号整数表示范围的结果截断至 `INT32_MAX` 或 `INT32_MIN`。在缺乏浮点运算单元的嵌入式处理器上，定点乘法是卷积、全连接等神经网络核心算子的基础操作，在推理过程中被高频调用。然而，该函数在基础 RISC-V 指

表 4.1 定点乘法函数 q_mul 的原始实现

```
static inline q16_16 q_mul(q16_16 a, q16_16 b) {
    int64_t t = (int64_t)a * (int64_t)b;
    t >>= Q;
    if (t > INT32_MAX) t = INT32_MAX;
    if (t < INT32_MIN) t = INT32_MIN;
    return (q16_16)t;
}
```

令集上需要展开为乘法、移位、比较及条件赋值等多条指令序列，不仅增加了指令数量，还可能引入额外的中间寄存器传递、比较判断以及控制相关开销。因此，将其融合为一条单周期或少周期定制指令，能够有效缩短关键路径，减少指令开销，具有较高的硬件加速价值。

为全面评估定制指令的性能影响，本文在四个具有代表性的智能装备检测基准程序上分别进行了指令添加前后的仿真实验，包括音频处理 (audio)、手表传感 (H_Watch)、多传感器融合 (multi_sensor) 及腕带检测 (Wristband)。各基准程序添加指令前后的总执行周期数及对应加速比如表 4.2 所示，加速比对比结果如图 4.2 所示。

表 4.2 定制指令添加前后各基准程序总执行周期数对比

	audio	H_Watch	multi_sensor	Wristband
添加指令前	581 247	728 253	7 971	645 673
添加指令后	544 518	707 631	5 827	603 059
加速比	1.0675	1.0291	1.3679	1.0707

实验结果表明，引入定点乘法定制指令后，四个基准程序均获得了不同程度的性能提升。其中，multi_sensor 基准程序的加速效果最为显著，总执行周期从 7 971 下降至 5 827，加速比达到 1.3679，即执行速度约提升 36.79%。这主要是因为该程序规模较小且计算密集度高，定点乘法操作在其总执行时间中占比较大，因此定制指令对关键路径的缩短效果能够充分体现在整体性能上。Wristband 与 audio 基准程序分别获得了 1.0707 与 1.0675 的加速比，对应约 7.07% 与 6.75%

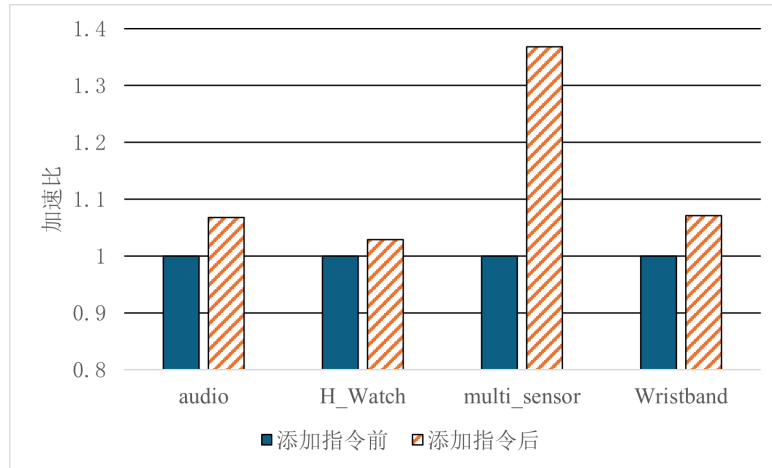


图 4.2 各基准程序引入定制指令后的加速比

的性能提升，表明定点乘法在这两个程序中同样属于高频调用的热点操作，定制指令有效减少了执行路径长度与指令调度开销。H_Watch 基准程序的加速比为 1.0291（约 2.91%），提升幅度相对较小，这可能与该程序中定点乘法调用在总计算量中占比较低、或程序执行瓶颈主要集中在访存与控制流等非计算密集型操作有关。

从整体趋势来看，定制指令的性能收益与目标操作在程序中的热点程度密切相关。当被优化的操作在程序关键路径上被高频调用时，通过硬件融合减少指令数量和中间结果传递开销，能够带来较为可观的加速效果；而当程序的执行瓶颈并非集中于该操作时，单一指令融合对整体性能的改善则相对有限。这一规律与 Amdahl 定律的基本原理相吻合：加速比的上限受限于被优化部分在总执行时间中所占的比例。

需要指出的是，`q_mul` 函数作为定点算术的核心原语，其加速效果具有良好的通用性和可扩展性。在实际的智能装备检测应用中，卷积层、全连接层以及激活函数等算子均大量依赖定点乘法运算，若模型规模扩大后定点乘法仍在总执行时间中保持较高占比，则该定制指令的整体收益有望进一步提升。同时，本实验结果也验证了前文所提出的自动指令识别与选择方法的有效性——通过系统化的候选指令提取与收益评估流程，能够准确定位程序中具有高加速潜力的操作模式，并指导硬件定制指令的设计与集成。

从体系结构设计角度分析，当前定制指令的实现采用了将乘法、移位与饱和截断逻辑融合至单一功能单元的方案，在数据通路组织上仍具有进一步优化的空间。后续工作可结合流水线结构对多周期操作进行更精细的调度，或面向更

大粒度的算子（如定点乘累加、定点向量内积等）开展指令融合设计，以进一步提升定制指令对整体处理器性能的贡献。

4.4 消融实验

为分析本文方法中各关键组件对指令选择效果的影响，本文从图编码器结构与强化学习策略两个维度进行消融实验。已有研究表明，PPO 在大规模子图搜索与复杂决策任务中具有较好的稳定性与收敛性能^[46]，本文在此基础上进一步验证不同模块组合的影响。

需要说明的是，在本文问题设置下通常存在多个具有相同最优奖励的候选子图，因此仅比较奖励难以区分不同方法的效果。本文采用“最优候选指令条数”作为评价指标，即在固定训练轮数（10000 episodes）后，统计各方法识别出的最优候选数量，并在 5 个随机种子（即图中配置 1 至配置 5 分别对应随机种子 35、42、45、48 和 50）下取结果进行分析。该指标反映模型对最优解集合的覆盖能力。在图表示学习部分，分别采用 GIN、MLP 以及不使用图编码器（none）进行对比；在强化学习策略部分，选取 PPO、REINFORCE 与 A2C 三种方法。在相同配置下完成训练。

图 4.3 给出了在 GIN 编码器下，不同强化学习策略的对比结果。可以看出，PPO 表现更为稳定，能够识别出更多最优候选指令；REINFORCE 波动较大，A2C 在部分情况下存在退化。这是因为该任务属于高维离散决策问题，奖励稀疏且延迟，PPO 通过限制策略更新幅度提升了训练稳定性，而 REINFORCE 与 A2C 在复杂状态空间下较难保持稳定收敛。

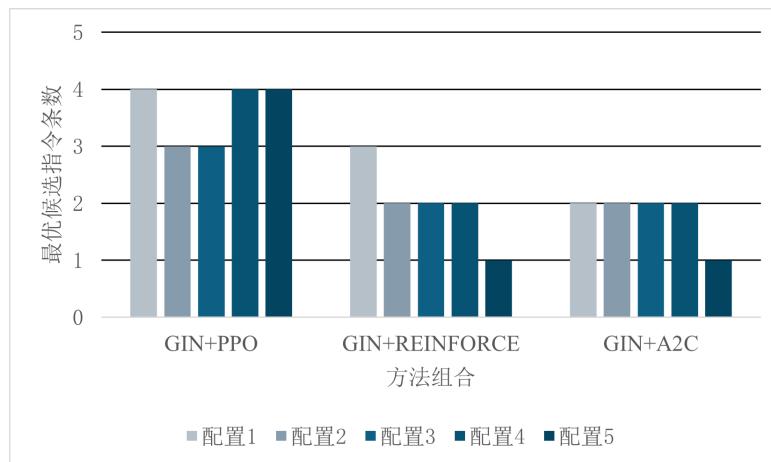


图 4.3 不同强化学习策略对比

图 4.4 展示了在 PPO 策略下，不同图编码器的对比结果。可以看到，GIN

在所有设置下均取得最优或接近最优结果，而 MLP 与不使用图编码器时性能明显下降，说明建模程序的图结构信息对于指令选择至关重要。

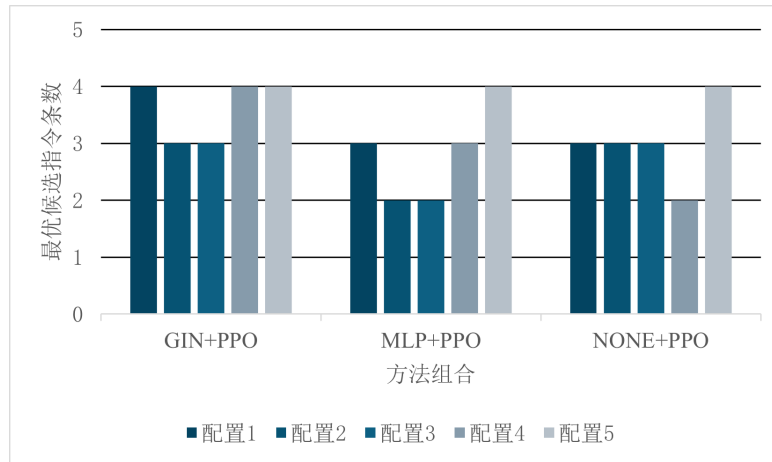


图 4.4 不同图编码器对比

此外，实验中观察到不同随机种子对训练结果存在较大影响，表现为最优候选指令条数在不同运行之间存在一定波动。但相较而言，GIN+PPO 组合在不同随机种子下表现出更一致的结果分布，稳定性明显优于其他策略组合，而其他方法在部分配置下波动更为显著。该现象主要源于强化学习在稀疏且延迟奖励场景下的高方差特性，不同初始化会导致探索路径与样本分布差异，从而影响策略收敛结果；同时，子图搜索空间具有较强的组合复杂性，策略易收敛至不同的局部最优解集合。未来可引入方差降低技术或结合课程学习（curriculum learning）提升训练过程的稳定性与结果的可复现性。

综合来看，GIN 提供了有效的结构表示能力，而 PPO 保证了搜索过程的稳定性，两者结合能够提升对最优候选集合的识别能力。需要指出的是，该指标主要用于评估方法的搜索能力与收敛特性，并不直接等价于处理器性能提升。

4.5 本章小结

本章围绕智能装备检测场景，构建了包含音频处理、健康监测、多传感器融合及核心推理在内的四个典型基准任务，并基于 OpenASIP 平台完成了从候选指令选择到处理器体系结构验证的完整实验流程。通过对比实验、消融实验与性能评估，系统验证了本文所提方法的有效性与可落地性。

在指令集架构约束对指令选择的影响分析中，实验表明输入输出约束直接决定了定制指令所能覆盖的数据流范围。较宽松的约束能够扩大搜索空间并发现更复杂的算子融合模式，从而获得更高的性能优化潜力；而过于严格的约束则

会限制指令组合空间,降低整体收益。本文在搜索过程中显式引入 RISC-V 架构约束,在保证候选指令具有较高奖励的同时确保其硬件可实现性,从而实现了”搜索质量”与”可落地性”的统一。

在定制指令扩展对处理器性能的影响评估中,本文选取定点乘法函数 `q_mul` 作为代表性候选指令进行原型集成与验证。该函数在基础 RISC-V 指令集上需要展开为多条指令序列,引入了额外的中间寄存器传递、比较判断与控制相关开销。实验结果表明,引入定制指令后,四个基准程序均获得了不同程度的性能提升,其中 `multi_sensor` 基准程序加速比达到 1.3679, `Wristband` 与 `audio` 分别获得约 7.07% 与 6.75% 的性能提升, `H_Watch` 获得约 2.91% 的提升。性能收益的差异主要取决于定点乘法操作在各程序中的热点程度与执行占比,符合 Amdahl 定律的基本原理。该结果验证了本文所提自动指令识别与选择方法能够准确定位程序中具有高加速潜力的操作模式,并指导硬件定制指令的设计与集成。

消融实验从图编码器结构与强化学习策略两个维度验证了各关键组件的必要性。实验采用”最优候选指令条数”作为评价指标,在固定训练轮数下统计各方法识别出的最优候选数量。结果表明,GIN 编码器能够有效建模 DFG 中的结构信息,显著优于 MLP 与不使用图编码器的方案;PPO 策略通过限制策略更新幅度提升了训练稳定性,在高维离散决策问题中表现优于 REINFORCE 与 A2C。GIN 与 PPO 的组合在不同随机种子下表现出更一致的结果分布,能够在复杂搜索空间中获得更优且更稳定的指令选择结果。

综合来看,本章实验从可实现性、性能收益与方法有效性三个方面验证了本文方法的有效性。在可实现性方面,通过引入 ISA 约束使候选指令能够直接应用于实际处理器设计;在性能收益方面,定制指令扩展能够有效减少指令条数并缩短执行路径,在基准程序上获得了最高 36.79% 的性能提升;在方法有效性方面,图结构建模与强化学习的结合能够在保证可实现性的前提下带来实际性能收益。实验结果表明,定制指令的优化效果高度依赖于目标算子的动态热点程度,未来通过针对高占比热点算子进行更大粒度融合,并结合微架构级优化,有望获得更显著的性能提升。

5 结论与展望

5.1 结论

本文围绕“面向智能装备检测的 RISC-V 专用处理器研究”这一主题,针对智能装备检测任务中计算密集与访存密集并存、算法模式多样以及边缘部署资源受限等特点,研究了面向真实应用负载的定制指令自动化选择与处理器实现方法。

首先,本文面向智能装备检测场景中的典型应用程序,对定制指令识别问题进行了形式化建模。通过将目标程序表示为程序数据依赖图,定义了候选定制指令的图表示方式,并系统刻画了合法操作、连通性、凸性、输入输出规模及面积预算等可实现性约束。同时,构建了结合软件执行代价与硬件执行代价的收益评估模型,引入动态执行信息对候选模式进行加权评价,使得候选指令的选择既能够反映结构层面的潜在加速能力,也能够体现其在真实运行负载下的重要性。该建模为后续自动化搜索与优化提供了统一基础。

其次,针对复杂程序图与多重硬件约束下搜索空间膨胀及全局优化困难的问题,本文提出了一种基于 GIN 与 PPO 的候选定制指令自动选择方法。该方法利用 GIN 对程序数据依赖图中的节点属性及结构关系进行编码,并通过 PPO 在约束条件下学习候选子图的构造策略,实现了在程序结构特征、动态执行热点及硬件实现约束共同作用下的有效搜索。实验结果表明,该方法在复杂离散搜索空间中能够获得更稳定且质量更高的候选指令选择结果。

再次,本文在算法研究基础上,构建了面向 RISC-V 和 OpenASIP 的定制指令自动扩展与处理器实现原型系统。系统以 C 源程序或 LLVM IR 为输入,基于 SSA 构建程序数据依赖图,并将候选子图转换为扩展操作描述;随后通过 OSAL 完成操作语义定义,通过 VHDL 实现扩展操作硬件逻辑,并结合 ADF、HDB 与 IDF 等描述文件完成处理器结构集成。在编译映射与处理器生成阶段,利用 OpenASIP 编译工具链(oacc)完成扩展操作的编译支持,并生成对应的处理器模型进行仿真验证,从而形成了覆盖“程序分析—候选识别—扩展生成—编译映射—处理器生成—仿真验证”的完整工程流程。

最后,本文通过指令集约束分析、性能评估及消融实验对所提出方法进行了验证。实验结果表明,输入输出约束会显著影响候选指令的搜索空间及可实现

性, 适度放宽约束有助于发现覆盖更长数据依赖链的高收益模式。在处理器原型验证中, 引入自动识别的定点乘法扩展指令 `q_mul` 后, 各基准程序执行周期均有所下降, 其中 `multi_sensor` 的加速比达到 1.3679, 其余程序亦取得不同程度的性能提升。消融实验进一步表明, GIN 对图结构信息的建模能力以及 PPO 在高维离散决策空间中的策略优化机制, 是提升候选指令搜索质量与稳定性的关键因素。

总体而言, 本文完成了面向智能装备检测任务的定制指令自动选择方法研究, 并在 OpenASIP 平台上实现了处理器扩展原型验证。研究表明, 将图结构建模、强化学习搜索与 ISA 约束感知相结合, 是实现领域专用处理器自动化指令扩展的一条可行技术路径。同时, 实验结果也表明定制指令的性能收益具有明显的负载相关性: 当目标操作在程序关键路径中占比较高时, 可获得较显著的加速效果; 而在访存或控制流瓶颈主导的场景下, 单一指令扩展的整体收益相对有限。该结论为后续面向高占比热点算子、复合指令扩展以及微结构协同优化的进一步研究提供了依据。

5.2 展望

尽管本文已完成候选定制指令识别及 RISC-V 专用处理器集成验证的基本流程, 仍存在若干有待深入研究的方向。

首先, 未来需进一步完善候选定制指令的识别与筛选机制, 重点挖掘高收益、广覆盖的热点指令模式。当前原型实现主要面向单一热点操作, 验证了定制指令扩展流程的可行性, 并在部分基准程序中取得加速效果。然而, 其对整体性能的提升仍受热点占比、瓶颈分布及编译器优化等因素限制。未来可面向智能装备检测中的高频复合算子(如定点乘累加、FFT 蝶形运算), 开展系统化的候选挖掘与收益评估, 从而筛选对关键执行路径影响更显著的定制指令。

其次, 未来需完善处理器硬件描述及实现框架, 以支持更灵活的多输入多输出扩展指令形式。本文实验表明, 较宽松的输入输出约束有助于扩展指令覆盖更长的数据依赖链, 并提升候选模式的潜在收益; 但在当前实现中, 扩展操作主要通过绑定既有 ALU 并扩展操作语义完成集成。受限于功能单元端口数量、寄存器接口及数据通路结构, 复杂候选模式仍难以落地。未来需进一步调整 ADF 中的功能单元与端口配置, 增强 OSAL 定义与 VHDL 实现的一致性, 并扩展寄存器文件与数据通路对多操作数、多结果写回的支持, 从而为复杂复合指令提

供硬件基础。

再次，后续研究可进一步加强定制指令选择与底层微结构设计之间的协同优化。本文目前主要从候选子图收益与基本实现约束出发开展指令选择，而对流水线调度、多周期功能单元、资源竞争、结构冒险及访存交互等微结构因素考虑仍然相对有限。未来可在指令识别阶段引入更细粒度的微结构约束模型，使候选指令的选择结果更加贴近真实处理器实现条件，进一步提升“算法最优”与“硬件可落地”之间的一致性。

最后，从实验平台与验证体系来看，后续还需要进一步提升仿真效率、工具链稳定性以及实验规模。尽管已初步验证方法可行性，但仿真环境复杂度与效率仍存在限制。未来可在更丰富的基准测试任务、更大规模的程序图以及更多处理器配置下开展系统实验，并通过完善自动化评测流程，提高实验的稳定性、充分性与说服力。

综上，本文工作可视为面向智能装备检测场景的 RISC-V 专用处理器指令扩展的一次初步探索。未来将围绕高占比热点算子识别、复合指令生成及其硬件支撑机制展开深入研究，并进一步加强微结构协同优化与工具链自动化能力，以持续提升专用处理器的性能、能效及工程实用价值。

参考文献

- [1] BOYES H, HALLAQ B, CUNNINGHAM J, et al. The industrial internet of things (iiot): An analysis framework[J/OL]. Computers in Industry, 2018, 101: 1-12. <https://www.sciencedirect.com/science/article/pii/S0166361517307285>. DOI: 10.1016/j.compind.2018.04.015.
- [2] SHI W, CAO J, ZHANG Q, et al. Edge computing: Vision and challenges [J/OL]. IEEE Internet of Things Journal, 2016, 3(5): 637-646. DOI: 10.1109/JIOT.2016.2579198.
- [3] XU H, YU W, GRIFFITH D, et al. A survey on industrial internet of things: A cyber-physical systems perspective[J/OL]. IEEE Access, 2018, 6: 78238-78259. DOI: 10.1109/ACCESS.2018.2884906.
- [4] WULF W A, MCKEE S A. Hitting the memory wall: implications of the obvious[J/OL]. SIGARCH Comput. Archit. News, 1995, 23(1): 20-24. 10.1145/216585.216588.
- [5] WILLIAMS S, WATERMAN A, PATTERSON D. Roofline: an insightful visual performance model for multicore architectures[J/OL]. Commun. ACM, 2009, 52(4): 65-76. <https://doi.org/10.1145/1498765.1498785>.
- [6] ESMAEILZADEH H, BLEM E, ST. AMANT R, et al. Dark silicon and the end of multicore scaling[J/OL]. SIGARCH Comput. Archit. News, 2011, 39(3): 365-376. <https://doi.org/10.1145/2024723.2000108>.
- [7] IENNE P, LEUPERS R. Customizable embedded processors: Design technologies and applications[M/OL]. Morgan Kaufmann Publishers, 2006. <https://api.semanticscholar.org/CorpusID:106531359>.
- [8] FRIGO M, JOHNSON S G. The design and implementation of fftw3[J/OL]. Proceedings of the IEEE, 2005, 93(2): 216-231. DOI: 10.1109/JPROC.2004.840301.
- [9] COOLEY J W, TUKEY J W. An algorithm for the machine calculation of complex fourier series[J/OL]. Mathematics of Computation, 1965, 19(90): 297-301. <https://doi.org/10.1090/S0025-5718-1965-0178586-1>.

- [10] ATASU K, POZZI L, IENNE P. Automatic application-specific instruction-set extensions under microarchitectural constraints[C/OL]//Proceedings of the 40th Design Automation Conference (DAC). 2003: 256-261. DOI: 10.1145/775832.775897.
- [11] GALUZZI C, BERTELS K. The instruction-set extension problem: A survey [J/OL]. ACM Trans. Reconfigurable Technol. Syst., 2011, 4(2). <https://doi.org/10.1145/1968502.1968509>.
- [12] RISC-V International. The risc-v instruction set manual, volume i: Unprivileged architecture[M/OL]. 2024. https://docs.riscv.org/reference/isa/v20240411/_attachments/riscv-unprivileged.pdf.
- [13] RISC-V International. The RISC-V instruction set manual, volume i: Unprivileged architecture, chapter 35: Extending RISC-V[EB/OL]. 2025. <https://docs.riscv.org/reference/isa/v20250508/unpriv/extending.html>.
- [14] LLVM Project. Risc-v target usage guide[EB/OL]. 2024. <https://releases.lvm.org/19.1.0/docs/RISCVUsage.html>.
- [15] JÄÄSKELÄINEN P, VIITANEN T, TAKALA J, et al. Hw/sw co-design toolset for customization of exposed datapath processors[M/OL]//HUSSAIN W, NURMI J, ISOAHO J, et al. Computing Platforms for Software-Defined Radio. Springer International Publishing, 2017: 147-164. https://doi.org/10.1007/978-3-319-49679-5_8.
- [16] OpenASIP Project. Openasip: Application-specific instruction-set processor design toolset[EB/OL]. <https://openasip.org/>.
- [17] HEPOLA K, MULTANEN J, JÄÄSKELÄINEN P. Openasip 2.0: Co-design toolset for risc-v application-specific instruction-set processors[C]//2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2022: 161-165.
- [18] GAROFALO A, TAGLIAVINI G, CONTI F, et al. XpulpNN: Accelerating quantized neural networks on RISC-V processors through ISA extensions [C/OL]//Proceedings of the Design, Automation and Test in Europe Conference (DATE). IEEE, 2020: 186-191. DOI: 10.23919/DATE48585.2020.91

16529.

- [19] KEUTZER K, MALIK S, NEWTON A R. From asic to asip: the next design discontinuity[C/OL]//Proceedings of the IEEE International Conference on Computer Design (ICCD). IEEE, 2002: 84-90. DOI: 10.1109/ICCD.2002.1106752.
- [20] ESKO O, JÄÄSKELAINEN P, HUERTA P, et al. Customized exposed datapath soft-core design flow with compiler support[C]//2010 International Conference on Field Programmable Logic and Applications. IEEE, 2010: 217-222.
- [21] Texas Instruments. Tms320c6000 technical brief[R/OL]. Texas Instruments, 2000. <https://www.ti.com/lit/pdf/SPRU197>.
- [22] JOUPPI N P, YOUNG C, PATIL N, et al. In-datacenter performance analysis of a tensor processing unit[C/OL]//ISCA '17: Proceedings of the 44th Annual International Symposium on Computer Architecture. New York, NY, USA: Association for Computing Machinery, 2017: 1–12. <https://doi.org/10.1145/3079856.3080246>.
- [23] KARURI K, AL FARUQUE M A, KRAEMER S, et al. Fine-grained application source code profiling for asip design[C/OL]//DAC '05: Proceedings of the 42nd Annual Design Automation Conference. New York, NY, USA: Association for Computing Machinery, 2005: 329–334. <https://doi.org/10.1145/1065579.1065666>.
- [24] HUSSEIN E, WASCHNECK B, MAYR C. Automating application-driven customization of asips: A survey[J/OL]. J. Syst. Archit., 2024, 148(C). <https://doi.org/10.1016/j.sysarc.2024.103080>.
- [25] POZZI L, VULETIC M, IENNE P. Automatic topology-based identification of instruction-set extensions for embedded processors[C]//Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition. IEEE, 2002: 1138.
- [26] YU P, MITRA T. Scalable custom instructions identification for instruction-set extensible processors[C/OL]//CASES '04: Proceedings of the 2004 Inter-

- national Conference on Compilers, Architecture, and Synthesis for Embedded Systems. New York, NY, USA: Association for Computing Machinery, 2004: 69–78. <https://doi.org/10.1145/1023833.1023844>.
- [27] CONG J, FAN Y, HAN G, et al. Application-specific instruction generation for configurable processor architectures[C/OL]//FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays. New York, NY, USA: Association for Computing Machinery, 2004: 183–189. <https://doi.org/10.1145/968280.968307>.
- [28] CLARK N T, ZHONG H, MAHLKE S A. Automated custom instruction generation for domain-specific processor acceleration[J/OL]. IEEE Transactions on Computers, 2005, 54(10): 1258-1270. DOI: 10.1109/TC.2005.156.
- [29] AHN J, CHOI K. Isomorphism-aware identification of custom instructions with i/o serialization[J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013, 32(1): 34-46. DOI: 10.1109/TCAD.2012.2214033.
- [30] POZZI L, ATASU K, IENNE P. Exact and approximate algorithms for the extension of embedded processor instruction sets[J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2006, 25(7): 1209-1229. DOI: 10.1109/TCAD.2005.855950.
- [31] ATASU K, LUK W, MENCER O, et al. Fish: Fast instruction synthesis for custom processors[J/OL]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2012, 20(1): 52-65. DOI: 10.1109/TVLSI.2010.2090543.
- [32] AHN J, LEE I, CHOI K. A polynomial-time custom instruction identification algorithm based on dynamic programming[C/OL]//ASPDAC '11: Proceedings of the 16th Asia and South Pacific Design Automation Conference. Yokohama, Japan: IEEE Press, 2011: 573–578. DOI: 10.5555/1950815.1950930.
- [33] LI T, SUN Z, JIGANG W, et al. Fast enumeration of maximal valid subgraphs for custom-instruction identification[C/OL]//CASES '09: Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. New York, NY, USA: Association for Com-

- puting Machinery, 2009: 29–36. <https://doi.org/10.1145/1629395.1629402>.
- [34] WANG S, XIAO C. Reinforcement learning for selecting custom instructions under area constraint[J/OL]. IEEE Transactions on Artificial Intelligence, 2024, 5(4): 1882-1894. DOI: 10.1109/TAI.2023.3308099.
- [35] ATASU K, DÜNDAR G, ÖZTURAN C. An integer linear programming approach for identifying instruction-set extensions[C/OL]//CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. New York, NY, USA: Association for Computing Machinery, 2005: 172–177. <https://doi.org/10.1145/1084834.1084880>.
- [36] LI T, JIGANG W, LAM S K, et al. Selecting profitable custom instructions for reconfigurable processors[J/OL]. Journal of Systems Architecture, 2010, 56(8): 340-351. <https://www.sciencedirect.com/science/article/pii/S138376211000024X>. DOI: 10.1016/j.sysarc.2010.04.004.
- [37] MARTIN K, WOLINSKI C, KUHCINSKI K, et al. Constraint programming approach to reconfigurable processor extension generation and application compilation[J/OL]. ACM Trans. Reconfigurable Technol. Syst., 2012, 5(2). <https://doi.org/10.1145/2209285.2209289>.
- [38] GIAQUINTA E, MISHRA A, POZZI L. Maximum convex subgraphs under i/o constraint for automatic identification of custom instructions[J/OL]. Trans. Comp.-Aided Des. Integ. Cir. Sys., 2015, 34(3): 483–494. <https://doi.org/10.1109/TCAD.2014.2387375>.
- [39] YU P, MITRA T. Satisfying real-time constraints with custom instructions [C/OL]//CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. New York, NY, USA: Association for Computing Machinery, 2005: 166–171. <https://doi.org/10.1145/1084834.1084879>.
- [40] XIAO C, CASSEAU E. Exact custom instruction enumeration for extensible processors[J/OL]. Integration, 2012, 45(3): 263-270. <https://www.sciencedirect.com/science/article/pii/S0167926011001003>. DOI: 10.1016/j.vlsi.2011.

11.011.

- [41] XIAO C, CASSEAU E, WANG S, et al. Automatic custom instruction identification for application-specific instruction set processors[J/OL]. *Microprocessors and Microsystems*, 2014, 38(8, Part B): 1012-1024. <http://www.sciencedirect.com/science/article/pii/S0141933114001288>. DOI: 10.1016/j.micpro.2014.09.001.
- [42] BLINDELL G H. Instruction selection: Principles, methods, and applications [M/OL]. Springer Cham, 2016. <https://link.springer.com/book/10.1007/978-3-319-34019-7>.
- [43] ATASU K, MENCER O, LUK W, et al. Fast custom instruction identification by convex subgraph enumeration[C/OL]//International Conference on Application-Specific Systems, Architectures and Processors. 2008: 1-6. DOI: 10.1109/ASAP.2008.4580145.
- [44] DAI H, KHALIL E B, ZHANG Y, et al. Learning combinatorial optimization algorithms over graphs[A/OL]. 2018. arXiv: 1704.01665. <https://arxiv.org/abs/1704.01665>.
- [45] DARVARIU V A, HAILES S, MUSOLESI M. Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective[A/OL]. 2024. arXiv: 2404.06492. <https://arxiv.org/abs/2404.06492>.
- [46] HUSSEIN E, WASCHNECK B, MAYR C. Scalable custom instruction identification using graph neural networks and reinforcement learning[J/OL]. *TechRxiv*, 2025, 2025(0512). <https://www.techrxiv.org/doi/abs/10.36227/techrxiv.174702136.65195692/v1>.
- [47] ZHOU X, LIU T, TANG T, et al. Efficient instruction fusion for asips: A low-cost approach to structural hazard control[J/OL]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2026, 45(5): 2418-2431. DOI: 10.1109/TCAD.2025.3611113.
- [48] ZHOU X, MAN Y, HAO P, et al. Asip performance enhancement by hazard control through scoreboard[J/OL]. *Micromachines*, 2024, 15(11). <https://www.mdpi.com/2072-666X/15/11/1287>. DOI: 10.3390/mi15111287.

- [49] REZUNOV E, ZURSTRÄßEN N, REIMANN L M, et al. Automatic microarchitecture-aware custom instruction design for risc-v processors [M/OL]//2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD). 2025. DOI: 10.1109/ICCAD66269.2025.11240781.
- [50] HAO X, ZHANG S, QIAO L, et al. Ciexplorer: Microarchitecture-aware exploration for tightly integrated custom instruction[C/OL]//ICS '25: Proceedings of the 39th ACM International Conference on Supercomputing. New York, NY, USA: Association for Computing Machinery, 2025: 975–990. <https://doi.org/10.1145/3721145.3730421>.
- [51] XU K, HU W, LESKOVEC J, et al. How powerful are graph neural networks? [A/OL]. 2019. arXiv: 1810.00826. <https://arxiv.org/abs/1810.00826>.
- [52] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms[A/OL]. 2017. arXiv: 1707.06347. <https://arxiv.org/abs/1707.06347>.
- [53] WANG S, XIAO C, LIU W, et al. A comparison of heuristic algorithms for custom instruction selection[J/OL]. Microprocess. Microsyst., 2016, 45(PA): 176–186. <https://doi.org/10.1016/j.micpro.2016.05.001>.
- [54] ALLAMANIS M, BROCKSCHMIDT M, KHADEMI M. Learning to represent programs with graphs[C/OL]//International Conference on Learning Representations. 2018. <https://openreview.net/forum?id=BJOFETxR->.
- [55] CUMMINS C, FISCHES Z V, BEN-NUN T, et al. Programl: A graph-based program representation for data flow analysis and compiler optimizations [C]//MEILA M, ZHANG T. Proceedings of Machine Learning Research: Vol. 139 Proceedings of the 38th International Conference on Machine Learning. PMLR, 2021: 2244-2253.
- [56] SCARSELLI F, GORI M, TSOI A C, et al. The graph neural network model [J/OL]. IEEE Transactions on Neural Networks, 2009, 20(1): 61–80. DOI: 10.1109/TNN.2008.2005605.
- [57] KIPF T N, WELLING M. Semi-supervised classification with graph convolutional networks[A/OL]. 2017. arXiv: 1609.02907. <https://arxiv.org/abs/>

1609.02907.

- [58] SUTTON R S, BARTO A G. Reinforcement learning: An introduction [M/OL]. 2nd ed. The MIT Press, 2018. <http://incompleteideas.net/book/the-book-2nd.html>.
- [59] BELLO I, PHAM H, LE Q V, et al. Neural combinatorial optimization with reinforcement learning[M/OL]//International Conference on Learning Representations. 2017. <https://openreview.net/forum?id=rJY3vK9eg>.

致 谢

壬寅仲秋，始涉泮水，志骛八极而游万仞。览缥緲以解肯綮，铿然击键；结郢质而泛觞咏，偃仰啸歌。而今丙午暮春，四载如隙，恐年岁之不吾与。惟见方塘水白，草木蔓发。江离采采被泽畔，杜若萋萋满芳洲。珞珈苍苍，落樱缤纷，林霭浮青，四时皆成气象；江汉汤汤，烟波浩渺，行吟其间，辄复心与物游。

四年求学，所感万端，然问学之道，非师者不能引其津涯。回首既往，尤感师恩之重。首先，谨向我的校外指导老师刘成老师致以最诚挚的感谢。毕设期间，承蒙老师悉心指点，从选题立意至论文撰写，屡加斧正。每于疑难未解之处，老师或剖析其理，或指陈其误，使我得以渐窥门径。其治学之严谨，思路之开阔，令我受益匪浅。

同时，谨向校内指导老师龚奕利老师致以衷心感谢。本科数载，蒙老师教诲与关照，不惟授以知识，尤重方法与取径。使我知所从入，亦知所从出，渐得为学之纲纪。往昔点滴，虽未必显于一时，却潜移默化，滋养至今。今得以完成此文，实赖昔日积累之功。

此外，复谢计算所诸位师兄师姐。研讨之间，或析其疑，或开其蔽，使我渐得门径。往来论辩，言近旨远，其益良多。又谢本科诸学长学姐及同窗故友。数载相与，或对卷夜深，或谈笑忘倦。风雨同舟，悲欢共度。回首往昔，虽时移境迁，而情谊犹在。

最后，谨以此文致谢我的父母。求学至今，承其抚育与支持，使我得以心无旁骛，专志于学。凡所抉择，皆蒙体谅，未尝有所拘牵。

凡此种种，未可尽言。子曰：志于道，据于德，依于仁，游于艺。虽学未成章，然志有所立，愿探赜索隐，以穷其理，由一隅而通其全，于榛莽中尽根荄。昔者知武辞之桑林，叔向贺其悬鹑。是知外累虽殷，不足夺其守；求索未已，犹当勉以行。零雨其濛，白露未晞，行而不辍，斯道未央。