arXiv:2601.02062v1 [quant-ph] 5 Jan 2026

将量子软件工具集成至 MLIR

Patrick Hopf[†‡]、Erick Ochoa Lopez[§*]、Yannick Stade[†]、Damian Rovara[†]、Nils Quetschlich[†]、Ioan Albert Florea[†]、Josh Izaac[¶]、Robert Wille[†‡‖]和 Lukas Burgholzer[†‡]

[†]德国巴伐利亚州慕尼黑慕尼黑工业大学
[‡]德国巴伐利亚州慕尼黑附近的加兴慕尼黑量子软件公司 (MQSC)[‖]奥地利上奥地利州哈根贝格哈根贝格软件能力中心
[¶]加拿大安大略省多伦多 Xanadu 量子技术公司[§]加拿大安大略省万锦市 AMD

{patrick.hopf, yannick.stade, damian.rovara, nils.quetschlich, ioan.florea, robert.wille, lukas.burgholzer}@tum.de eochoalo@amd.com, josh@xanadu.ai

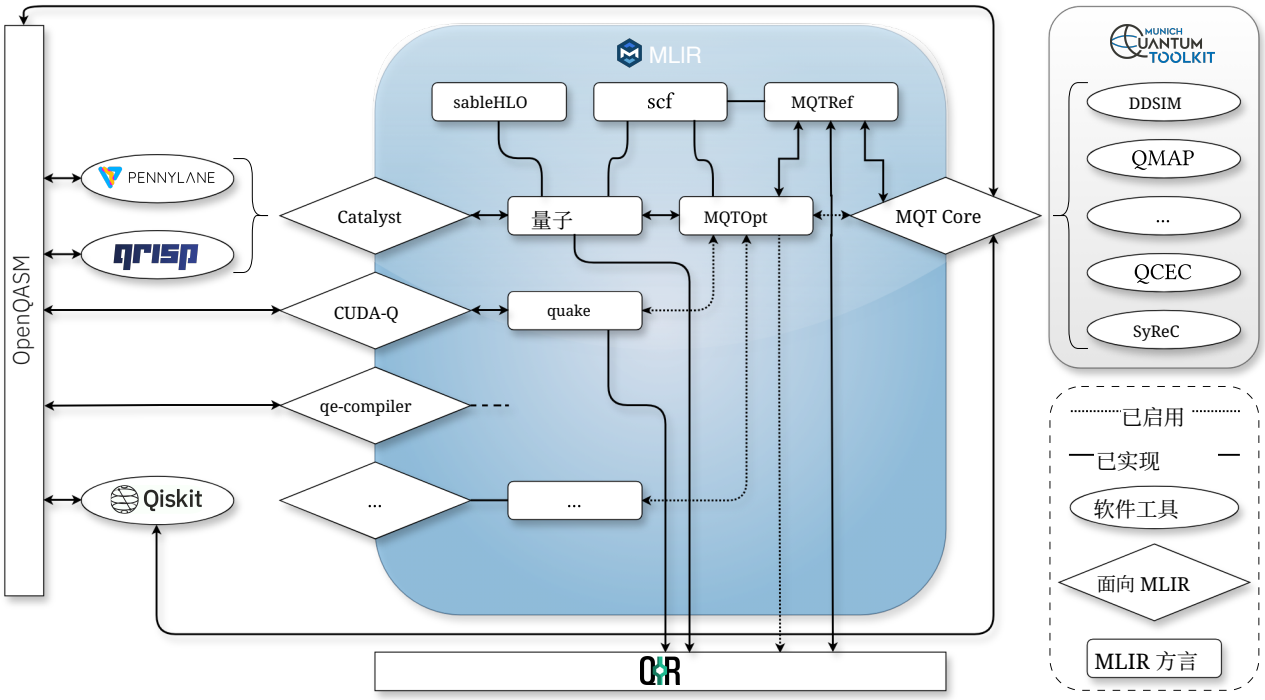


图 1：MLIR 生态系统内外的量子计算软件工具。

摘要

编译器将代码转化为行动：它们将高级程序转换为可执行的硬件指令，这是实现可靠且可扩展的量子计算的关键一步。然而，量子编译仍处于起步阶段，许多现有解决方案都较为临时，往往是独立从头开发的。这种互操作性的缺失造成了巨大的潜力浪费，因为量子软件工具仍彼此孤立，无法无缝集成到统一的工具链中。

多层中间表示（*MLIR*）解决了经典计算领域中类似的挑战。它是在 LLVM 项目中开发的。长期以来，该项目一直为可靠的软件栈提供支持，并实现了跨多种软件和硬件组件的编译，在高性能计算环境中尤为重要。然而，MLIR 陡峭的学习曲线构成了显著的入门障碍，在量子计算领域尤其如此：量子软件栈的大部分内容仍主要由实验研究人员出于实际需要构建，而非由经验丰富的软件工程师开发。

*在 Xanadu 量子技术公司（加拿大多伦多）任职期间完成的工作。

允许免费为个人或课堂用途制作本作品全部或部分内容的数字或纸质副本，前提是这些副本不得为营利或商业利益而制作或分发，且每份副本首页均须包含本声明及完整引文。必须尊重由作者以外的其他人持有的本作品各组成部分的版权。注明出处后可以进行摘要。若要以其他方式复制、重新发表、发布到服务器或分发至列表，则须事先获得明确许可和/或支付费用。请联系 permissions@acm.org 申请许可。*SCA/HPCAsia 2026*，日本大阪 © 2026 版权归所有者/作者所有。出版权已授权给 ACM。ACM ISBN 979-8-4007-2067-3/2026/01 https://doi.org/10.1145/3773656.3773658

本文为量子（软件）工程师提供了一份实用的动手指南，帮助他们克服这条陡峭的学习曲线。我们通过一个将 Xanadu 的 PennyLane 框架与慕尼黑量子工具包（MQT）相结合的具体案例研究，概述可操作的集成步骤，重点介绍最佳实践，并分享从实际开发中积累的宝贵经验。本文旨在帮助量子工具开发者应对 MLIR 的复杂性，并推动其成为连接快速发展的量子软件工具生态系统的统一桥梁，最终引导构建更加模块化、可互操作且集成度更高的量子软件栈。

CCS 概念

• **硬件** → 量子计算； • **计算机系统组织** → **量子计算**； • **软件及其工程** → **编译器**；**可重定向编译器**；源代码生成。

关键词

量子软件开发、量子编译、中间表示、软件栈、MLIR。

ACM 引用格式：

Erick Ochoa Lopez§ * Yannick Stade†，Damian Rovara†，Nils Quetschlich†，Ioan Albert Florea†，Josh Izaac¶，Robert Wille†‡‖，以及 Lukas Burgholzer†‡。2026。《将量子软件工具集成至 MLIR》。载于 *SCA/HPCAsia 2026*：亚洲超级计算与亚太地区高性能计算国际会议（*SCA/HPCAsia 2026*），*2026*年*1*月*26*日至*29*日，日本大阪。ACM，纽约州纽约，美国，13页。https://doi.org/10.1145/3773656.3773658

Patrick Hopf

1 引言

随着量子软件工具的迅速发展以及量子硬件能力的不断提升，量子编译器变得日益重要。量子编译器充当高级编程语言与低级硬件指令之间的转换器，在连接这两个快速演进的领域中发挥着关键作用。一方面，工业界和学术界团队正针对各种需求开发专用的量子软件工具；另一方面，硬件提供商提供了种类繁多的量子计算平台，包括超导、囚禁离子、中性原子和光子系统，每种平台都有各自的原生门集，以及特定于硬件的能力和限制。为了避免每当出现新的软件工具或硬件平台时都重复造轮子，显然需要一种可扩展且稳健的编译基础设施，以协调各种高级量子编程语言与异构量子后端——正如经典计算和高性能计算系统中理所当然的那样。

然而，与成熟的经典编译框架相比，专用于量子编译的工具仍在缓慢发展。许多工具依赖针对狭窄用例量身定制的临时方案，彼此孤立地开发，而且经常从头重新实现核心组件。由此造成的互操作性缺失导致潜力大量流失，因为如果这些工具能够实现无缝集成，它们及其用户本可以互相受益。

幸运的是，仍有希望：对于经典编译流程，多层中间表示（*MLIR*）[36] 框架已证明，它能够有效应对量子编译目前面临的许多挑战。将这一成熟的基础设施用于量子编译器，是一个顺理成章的选择，原因有以下几点：

- 作为 LLVM [34] 项目的一部分，MLIR 已得到广泛采用——它不仅被 JAX [8], 、TensorFlow [1], 和 PyTorch [43], 等主要软件框架采用，从而与经典工作流深度集成；也被 Intel 和 NVIDIA 等硬件供应商采用，建立了与经典硬件和高性能计算硬件的关键联系，而这正是量子-经典混合程序的一项关键需求。
- 此外，MLIR 允许在共享基础架构中无缝组合多个编译通道，从而促进模块化、复用和互操作性。
- 归根结底，对量子软件开发者来说，扩展成熟且广泛应用的经典编译器基础架构，为其添加量子专用功能，要比从头构建完全独立的工具链，再逐步添加硬件接口、线性代数或控制流等经典功能更为务实。

然而，MLIR 是一个极其复杂的项目，其陡峭的学习曲线严重阻碍了它的普及——尤其是在这样一个群体中：成员主要来自计算机科学或编译器设计以外的不同学科，后来转入这一领域。因此，我们需要开创性的跨学科项目，将两个领域联系起来：一方是量子计算，另一方是编译器设计。

本文旨在帮助开发者应对 MLIR 的复杂性，并推动其作为连接碎片化量子软件生态系统的统一桥梁得到广泛采用。为此，我们 प्रस्तुत了一份面向量子软件工程师的实用指南，帮助他们克服在量子编译背景下使用 MLIR 所面临的陡峭学习曲线。我们展示了 MLIR 如何凭借其轻量且模块化的插件基础设施，充当量子程序表示与变换工作流的统一基础。全文始终强调实践方法，并重点介绍一项为期一年的集成工作中总结的最佳实践，该工作将两个主要量子软件框架连接起来：Xanadu 的 PennyLane [7] 和慕尼黑量子工具包（MQT）[60]。

本文其余部分安排如下：第 2 节回顾量子编译和经典编译中的相关概念，并概述相关工作。第 3 节阐述量子软件工具之间互操作性有限的问题，并说明 MLIR 是一种潜在的解决方案；目前，框架较高的学习门槛限制了其应用。为降低入门门槛，第 4 节介绍最佳实践，并通过一个详细的案例研究展示如何在 MLIR 生态系统中集成两个具有代表性的量子软件框架 —— PennyLane 和 MQT。第 6 节随后讨论此类集成带来的重要见解和未来前景。最后，第 7 节总结本文的贡献。

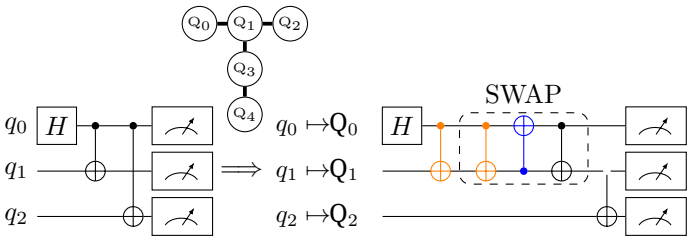


图 2：面向支持原生门集 {H门、T、受控非门} 的 T形五量子比特架构，编译用于制备 GHZ态的量子程序。

2 背景

为使本文内容完整自洽，本节简要回顾量子编译和经典编译的基础知识，并以中间表示相关工作的概述作为结尾。

2.1 量子编译

要在选定的量子设备上执行量子程序，必须先将该程序转换为可执行文件。此过程由编译器完成，编译器会将量子程序转换为一系列硬件专用指令（大致类似于经典的指令集架构，即 ISA）。这些指令必须遵循所选量子计算机施加的约束。

为此，编译通常会划分为作用于量子电路的遍，量子电路由量子比特线和量子门操作组成。这些遍可以按顺序执行，也可以多次调用，从而形成复杂的编译流程。所有遍大致可分为以下三类：

- 布局与路由：这些遍会将量子程序中的每个量子比特分配给目标硬件上的物理量子比特。它们确保根据设备的拓扑结构，所有量子门都可以执行。为了解决拓扑约束，可能会引入额外操作——例如超导系统中的 SWAP门，或中性原子和囚禁离子平台上的移动/穿梭操作——如 [5, 33, 51, 52, 61] 所示。
- 综合：这些遍会将目标设备原生不支持的量子操作分解为由其原生门集中的门组成的序列，采用的策略包括 [3, 18, 31, 45, 66] 中提出的方法。
- 优化：这些遍旨在提高量子程序的效率，从而缓解当前含噪量子计算机固有的一些局限性，采用的技术包括 [24–26, 41, 42] 中提出的方法。

示例 1。 假设左侧图 *2* 所示的、用于制备 *GHZ*态 的三量子比特量子程序需要映射到一个 *T*形五量子比特架构（如顶部所示），并综合为其原生门集 {H门、T、受控非门}。通过在 *Circuit* 与设备量子比特之间选择一一映射，并插入一个 *SWAP*门——将其综合为三个受控非门——即可推导出与设备拓扑相匹配的、可完全执行的 *Circuit*。后续优化过程可能会识别出两个橙色的受控非门相互抵消，并且在初始量子比特状态为 $|0\rangle$ 的假设下，蓝色受控非门 同样可以移除——最终只留下末尾的两个受控非门 操作。

为构建此类编译流程，所有组成编译流程的遍通常都会以一种通用格式——所谓的中间表示 （中间表示）——作为量子程序的输入和输出。为此，许多量子软件工具都在高层和低层抽象层级上引入了各自的中间表示；例如，Qiskit 的 DAGCircuit [29]、TKET 的 Circuit [55]、TKET2 的 HUGR [32]，以及 MQT 的 QuantumComputation [12]和 PennyLane 的 QuantumTape [7]。虽然这种方法在单个工具内部效果良好，但正如示例 2 所示，它使不同工具难以集成到统一的编译流程中。此外，目前尚未建立共享框架或标准化基础设施，以便在编译过程中促进不同量子软件工具之间高效的中间表示转换。

示例 2. *MQT Predictor [49]* 支持组合 *Qiskit* 和 *TKET* 的编译器遍，从而构建超越单个工具能力的优化编译流程。尽管基于强化学习的多种遍交错策略能够生成高度优化的量子程序，但底层编译例程以及随之而来的模型训练过程，必须不断在 *Qiskit* 和 *TKET* 使用的不同电路表示之间进行转换，因此带来了显著的运行时开销。

2.2 经典编译

与仍在发展中的量子软件生态系统相比，经典编译已受益于数十年的发展。其成功的关键因素之一是采用共享框架，其中最具代表性的是 LLVM [34]：一种广泛采用的编译器基础设施，提供可复用的库和工具。LLVM 使用一种称为 LLVM 中间表示的统一中间表示，将高级语言（如 C/C++、Rust 或 Java）转换为硬件专用的低级指令（例如汇编语言）。

除了语言转换之外，LLVM 已成为现代高性能计算软件栈的基石。它为 Clang 等广泛使用的生产级编译器提供基础，支持对高性能计算工作负载至关重要的向量化和并行化技术，并作为领域专用语言和加速器工具链的后端。凭借可复用的优化过程、特定目标后端和定义明确的中间表示，LLVM 能够在各种高性能计算架构上实现高度的可移植性和性能调优，涵盖从多核 CPU 到 GPU 及其他专用加速器；量子处理器也正开始崭露头角 [16]。

虽然 LLVM 中间表示有助于模块化编译器设计，但依赖单一的静态中间表示，可能难以表达高层抽象或底层硬件细节。为弥合这一差距，多层中间表示（*MLIR*） [35], 应运而生；它是一种更灵活的基础设施，支持不同抽象层级，旨在服务于领域专用编译器工程学。

MLIR 允许开发者通过方言定义自定义中间表示，并为特定领域组合定制化遍。它已被 JAX [8], Tensor- Flow [1], 和 PyTorch [43]等经典软件工具广泛采用。该框架原生支持将自定义方言和编译器遍打包并作为插件分发。通过串联多个遍，开发者可以构建灵活且模块化的编译流水线。

在内部，MLIR 遵循静态单赋值（SSA）语义：这意味着每个值 都只定义一次。所有值都显式关联到定义它们的操作以及使用它们的所有操作。这种所谓的定义-使用 链使得在编译通道中能够高效遍历程序。MLIR 类似图的程序表示由操作节点和值边组成，与量子电路天然契合；量子电路通常被建模为有向无环图。

总而言之，这些特性使 MLIR 成为通过自定义方言和遍次逻辑来表示和转换量子程序的理想基础，最终能够部署独立插件。

2.3 相关工作

鉴于如今有大量量子软件工具可供使用，出现同样丰富多样的量子程序表示也就不足为奇；每种表示都针对特定需求和用例进行了设计。人们提出了多种数据结构来描述量子电路的功能，其中最为显著的是张量网络 [47] 和决策图 [62, 63], , 它们旨在表示量子操作所作用的指数级庞大状态空间，因此适用于经典量子电路模拟 [20, 22, 67]或验证 [50]。其他表示则针对特定的编译任务而设计，例如 ZX 演算 [30]及其衍生形式 [2, 40], 、用于 Clifford 电路的稳定子形式体系 [19], , 以及基于图的模型 [6, 58, 59]。然而，这些表示并非旨在充当量子程序的通用交换格式，因为它们通常优先考虑紧凑性或特定特征的可处理性，而非广泛的表达能力和互操作性。

IBM 的 OpenQASM 3 [14] 引入了构建统一量子计算与经典计算的通用且可扩展中间表示的早期尝试。它在其前身 OpenQASM 2 [15] 成功的基础上，已在量子计算社区中得到广泛采用。然而，这也需要重新实现许多早已在经典编译器基础设施中建立并不断完善的概念 [56]。

为解决这一局限，量子中间表示（QIR） [48]——最初由 Microsoft 提出——开始逐渐出现 [56]。QIR 构建于经典领域已经确立的 LLVM 框架之上，并将其扩展为量子编程语言与各种量子硬件后端之间的通用接口。为此，它定义了一组用于表达量子程序的、嵌入 LLVM 中间表示中的硬件和语言无关格式，并已被采用到诸如 [21, 64] 等编译工具中。由于基于 LLVM，QIR 面临着经典编译中遇到的同样挑战。

因此，MLIR 最近在量子计算社区中日益流行也就不足为奇了——它提供了可定制的抽象层级和可扩展的中间表示框架。Q-MLIR [39] 探索了将量子构造集成到 MLIR 生态系统中的方式，同时保持与 QIR接口标准的兼容性。同样，QIRO [28] 扩展了 MLIR，以支持量子 - 经典协同优化。NVIDIA 的 CUDA-Q [57] 和 Xanadu 的 Catalyst [27] 等行业框架也构建于 MLIR 之上，为第三方工具开发者提供可扩展的编译器后端和插件访问能力。

关于早期量子中间表示（例如 QSSA [44], XACC[38], 或 QBIR [37]）以及较新的基于 MLIR 的方法（例如 Q-MLIR [39] 和 QIRO [28]）的全面综述与比较研究，可参见 [13]。

归根结底，尽管早期已尝试开发基于 MLIR 的量子编译中间表示，但面向种类繁多的量子计算工具，统一的集成途径仍然缺失。编译器社区与量子社区仍大多彼此隔绝，主要原因在于这两个领域都需要深厚的专业知识。

要充分释放这些学科交叉领域的潜力，必须让 MLIR 更易于量子计算社区使用和掌握。

3 动机

尽管 MLIR 提供了强大的基础设施，可用于构建能够连接多样化量子计算工具生态系统的复杂量子编译器，但它在量子社区中的应用仍然有限。本节将探讨阻碍 MLIR 在量子社区中得到更广泛应用的主要障碍，并概述本研究旨在如何应对这些障碍。

3.1 所考虑的问题

在经典计算和高性能计算领域，编译已日益标准化和统一，LLVM 中间表示在其中发挥着核心作用。MLIR 的引入及其对自定义方言的支持，放宽了以受控且一致的方式使用单一中间表示的限制。相比之下，量子计算所用的各种中间表示，缺乏适用于编译的统一共享基础设施。其主要原因在于，量子计算——更具体地说，量子软件工程——仍是一个新兴研究领域，尚无既定标准。许多研究人员和硬件供应商往往主要关注量子设备本身，而将软件视为次要事项。这造就了高度异构的量子计算软件生态，其中量子软件工具种类繁多。这些工具不仅在底层数据结构、编译策略和开发理念上各不相同，也因缺少共享中间表示而互操作性有限，实际上阻碍了不同工具的组合使用，除非投入大量精力。年轻的量子（软件）生态发展迅速，这进一步加剧了这一问题



图 3：映射一个　　使用 PennyLane [7] 且未进行 MLIR 集成的 MQT QMAP [61] 所定义的量子程序　　.

而其高度易变所带来的维护和持续开发需求，也使情况更加复杂。

尽管早期的标准化工作（例如 OpenQASM）提供了一些“变通方法”，但这些格式更适合作为人类可读的静态程序描述，而非适用于编译器基础设施的动态内存表示。因此，在现代量子软件栈中，它们通常不会被用作内部表示，这会导致显著的解析开销，并且在转换过程中经常丢失结构和语义信息。

示例 3. 假设示例 *1* 中描述的量子程序使用 *Xanadu* 的 *PennyLane* *[7]* 定义，并且需要使用 *MQT* 的 *QMAP* *[61]*进行映射。该程序在图 *2* 左侧展示，其功能是制备 *GHZ*态。由于这些工具没有共享中间表示，因此无法轻易实现联合编译流程。因此，必须采用一种变通方法，如图 *3* 所示。

将程序定义为 *PennyLane* `QuantumTape` 后，必须先将其导出为 *OpenQASM*字符串。随后可将该字符串转换为 *MQT Core* 的中间表示 `QuantumComputation`，再使用 *QMAP* 进行映射。映射完成后，必须将编译后的电路沿所有中间步骤反向转换，才能再次在 *PennyLane* 中使用。最终，这一流程需要安装 *Qiskit*，以便 `pennylane-qiskit` 导入 *OpenQASM*——这又引入了一个依赖项（及其自身的中间表示和转换例程）。

此外，映射过程产生的量子比特分配并未在 *OpenQASM* 中间表示中得到明确保留。因此，无法推断原始电路中的某个量子比特被映射到了哪个设备量子比特，而这对于推理编译后的电路的正确性可能至关重要 *[11]*。

与其为每一种新的量子软件工具重新开发一种中间表示，并依赖繁琐的工作流在不同工具之间进行通信，不如借鉴经典编译器和软件基础设施领域数十年的经验。MLIR 框架已在经典领域展现出显著优势，并为可扩展且可复用的量子编译器设计提供了坚实基础。然而，要理解并有效使用这样一个复杂而全面的框架——该框架采用 C++ 开发——绝非易事，并且会带来两大困难。

第一个是技术层面的：量子计算软件工程师通常具有物理学或工程学背景，往往更熟悉 Python；这也反映在以 Python 为主的量子计算软件格局中。因此，入门会遇到一道天然的门槛，因为学习 MLIR 通常还需要熟练掌握 C++ 。MLIR 大量使用模板编程模式，使情况更加复杂。即使只是设置庞大的 LLVM 项目（MLIR 是其子组件），本身也可能是一项挑战。这类项目包含数百万行代码，依赖项众多，并依靠 CMake 进行配置和构建 ——CMake 虽然功能强大，却也十分复杂，连计算机科学家都可能觉得难以掌握。

第二个障碍是概念性的：即使技术门槛降低，MLIR 仍引入了编译器相关概念——例如 SSA 语义、IR 转换和 pass 流水线——这些概念对物理学家而言可能并不熟悉，因为他们更习惯于酉矩阵表示、模式匹配和物理设计方面。尽管 xDSL [17]等项目旨在通过 Python 提供类似 MLIR 的抽象来降低这种障碍，但对于缺乏编译器构造背景的人来说，学习曲线依然陡峭。

PENNYLANE

```
导入 pennylane 作为 qml
从 catalyst 导入 测量
从 mqt_插件 导入 QMAP, 插件

@qml.qjit(pass_插件={plugin}, 方言_插件={plugin})
@QMAP({"cMap": [(0,1),(1,0),(1,2),(2,1)]})
@qml.qnode(qml.device("lightning.qubit", 导线=3))
定义 测试_ghz_电路():
    qml.Hadamard(wires=[0])
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[0, 2])
    返回 [测量(i) 对于 i 在 范围(3)]
```

图4：映射一个 qu 使用 MLIR 插件系统，对 PennyLane [7] 中定义的量子程序进行 antum [61] MQT QMAP m.

我们认为，这些因素在很大程度上导致 MLIR 尚未在量子计算领域得到广泛采用。MLIR 尚未充分发挥其潜力，无法将碎片化的量子软件工具生态连接起来，这凸显了我们需要一份清晰易懂的指南，帮助量子工程师将这一成熟的经典框架集成到他们的工具中。

3.2 贡献

本文旨在为希望利用 MLIR 作为编译框架的量子软件开发者提供一份实用指南。本文展示了如何有效利用 MLIR，通过共享的编译骨干连接异构量子软件工具。

为此，我们通过一个案例研究展示两种主要量子软件框架的集成：Xanadu 的 PennyLane 和慕尼黑量子工具包（MQT）。本文提供了在 MLIR 生态系统中成功集成所需的基础组件；演示仅聚焦于最关键的概念，同时也指明了有助于深入探索的宝贵资源，帮助读者轻松入门，而不必被 MLIR 的复杂细节所困扰。我们利用 MLIR 专用的插件基础设施，展示了一种轻量且模块化的方法，从而最大限度地降低复杂性和外部依赖——这些都是阻碍广泛采用的常见因素。全文强调动手实践，并突出最佳实践，分享集成过程中的关键见解。为进一步降低入门门槛并促进社区更广泛地采用，完整代码已作为 MQT 代码库的一部分公开发布 [60]。

下一节将展示如何借助 MLIR 可扩展的插件架构，实现无缝互操作性。

示例4。 示例*3*中描述的工作流程显然并非最优。不过，如图*4*所示，借助 *MLIR* 插件系统，此类工作流程的复杂度得以大幅降低。在幕后，程序通过专用方言和遍在 *MLIR* 中高效转换。只需在 *PennyLane* 中提供自定义插件和流水线，即可触发编译，无需解析、转储和加载 *OpenQASM*，也完全摆脱了对 *Qiskit* 的依赖。

4 集成

本节介绍一项实际集成工作，借助 MLIR 实现两个主要量子软件框架之间的互操作性。前两部分分别介绍这两个框架，第三部分则概述在 MLIR 生态系统中实现集成所需的关键组件。以上内容共同为希望在自己的工具链中采用 MLIR 的量子软件工程师提供了一份实用蓝图。

4.1 Xanadu 的 PennyLane

PennyLane [7] 是一个跨平台的量子编程 Python 库。与其他量子软件框架相比，它的主要特色在于内置支持通过量子指令和经典指令进行端到端自动微分。近期，PennyLane 新增了通过 Catalyst 编译器 [27]支持量子即时（QJIT）编译的功能。受 Python 生态系统中类似方法（包括 Numba 和 JAX 等框架）的启发，量子即时编译允许使用 Python 编写动态的量子-经典混合程序，但这些程序并非由 Python 解释器执行。相反，程序从 Python 中执行时，会被捕获（包括所有经典处理、量子指令和控制流），并通过 MLIR 和 LLVM 编译为经过优化的机器二进制文件。之后再次调用该 Python 程序时，将直接在指定的模拟器或硬件设备上执行这个预编译的二进制文件，无需重新编译或重新捕获程序。

在内部，Catalyst 编译器由三个主要组件构成：

(1) **Python 前端**。用户执行一个将通过 QJIT 编译的函数时，该函数会使用表示抽象函数参数的追踪器 执行，从而捕获函数的行为。此 Python 前端扩展了 JAX 的 [8]追踪基础设施，可捕获包含经典指令、PennyLane 量子操作和原生 Python 控制流的 Python 函数。随后，前端会将程序表示降级为 MLIR：量子操作使用专门的量子方言，经典指令则使用 OpenXLA 提供的 StableHLO 方言。

Catalyst 提供的专用 MLIR 量子方言允许用户定义自己的自定义门，这些门会被降级为运行时函数调用，并将门的名称作为参数传递。量子操作以 SSA 形式表示，其中量子操作接收输入量子比特并返回输出量子比特。

(2) **MLIR 编译器**。在编译期间，Catalyst 会将程序表示优化为 LLVM，并将程序编译为二进制文件。在此过程中，量子优化直接应用于结构化程序（即，不移除或展开任何经典控制流）。

(3) **运行时**。执行编译后的二进制文件时，Catalyst 会在用户指定的量子硬件或模拟器设备上执行量子指令，例如 PennyLane 的 Lightning 模拟器套件 [4]。

4.2 慕尼黑量子工具包（MQT）

慕尼黑量子工具包（*MQT*）[60] 是一套用于量子计算的开源软件工具，由慕尼黑工业大学设计自动化教席和慕尼黑量子软件公司（MQSC）共同开发。它还是慕尼黑量子软件栈（MQSS）[9]生态系统的一部分；该生态系统作为慕尼黑量子谷（MQV）计划的一部分进行开发。其总体目标是为整个量子软件栈中的设计任务提供解决方案。这包括为终端用户实现应用提供高级支持、用于量子电路经典模拟、编译和验证的高效方法、量子纠错工具、物理设计支持等。这些方法由相应的数据结构（如决策图或 ZX 演算）和核心方法（如 SAT 编码/求解器）提供支持。

鉴于整个工具包都需要有效地表示和操作量子电路，而且该项目起源于 OpenQASM 2 刚刚提出、此前讨论的工具尚未问世之时，MQT 采用了基于 C++ 的自有 QuantumComputation 中间表示，它是 MQT Core [12] 库的一部分，以下称为 *MQT Core IR*。

该中间表示用于 MQT 中的各种顶层库。其中之一是 MQT QMAP [61]。它的名称源于其最初主要用于将量子电路映射到连通性有限的超导架构（即解决布局布线问题）。多年来，它已发展为一套编译工具，可将量子电路映射到各种量子比特技术，如今还支持中性原子量子计算机 [51]。

截至本文撰写时，MQT Core 还提供了专用的 MLIR 方言，从 3.3.3 版本起即可使用。

https://github.com/munich-quantum-toolkit/core/tree/v3.3.3
https://mqt.readthedocs.io/projects/core/en/v3.3.3

此类方言对插件至关重要，下面将重点讨论其中一个特定方言，即 MQTOpt。

4.3 实现

将 PennyLane 和 MQT 集成到 MLIR 中需要若干基础组件。为了帮助读者清晰地理解这些构建模块，我们介绍一种专用的 MLIR 方言、相关的 变换以及嵌入式 插件 基础设施。我们鼓励读者探索 MQT GitHub 仓库中提供的配套开源实现：

https://github.com/munich-quantum-toolkit/core-plugins-catalyst
https://mqt.readthedocs.io/projects/core-plugins-catalyst

方言： Xanadu 的 Catalyst 提供的量子 MLIR 方言可直接访问量子程序的结构和语义。例如，图 4 中实现的 GHZ 电路的部分 MLIR 表示如图 5a 所示，采用 Catalyst 的 SSA 格式。

然而，正如第 2 节所述，许多量子软件工具并未开箱即支持 MLIR。由于 MQT 同样缺乏原生 MLIR 支持，因此需要在 MQT Core IR 与 MLIR 之间进行转换。

为简化这一过渡并尽量减少此类变更造成的影响，我们创建了一种专用量子方言（称为 MQTOpt），其语义与现有的 MQT Core IR 非常接近。除了采用 SSA 语义外，它还强制执行单次使用约束，即每个值只能被使用一次。这种线性类型属性自然地保证了不可克隆定理 [65],，因为每个量子比特都保证恰好被定义和使用一次。这也进一步简化了量子比特提取或插入期间对寄存器状态变化的跟踪，从而便于后续转换为明确建模硬件上物理量子比特分配的程序表示，或从这类表示进行转换。例如，从量子寄存器中提取一个量子比特会同时得到一个量子比特和一个更新后的寄存器；如图 5b 所示，这在自定义的 MQTOpt 方言中有所体现。

此外，图 5b 突出了方言特有的操作数和结果类型（例如，量子比特寄存器）以及操作（例如，X 门），并展示了与之关联的修饰符（此处为 ctrl）。与 Quantum 方言（参见图 5a）的另一项关键区别在于，Catalyst 将门表示为由 gate_name 属性区分的通用操作，而 MQTOpt 方言则将每个门定义为专用的方言特有操作。请注意，所示代码仅为程序的文本表示（由 MLIR 自动生成），不一定与 C++ 实现中的类、类型和属性名称一致。

通过定义特征，可以进一步简化方言的实现；特征封装了不同类型或操作共有的属性。在 MQTOpt 方言中，结构不变量（例如，特定操作所需的参数数量，如旋转门）通过在相应的特征定义中实现 verifyTrait() 函数来强制执行。

变换： 接下来，为了实现这两种方言之间的无缝转换，MLIR 转换模式提供了一种有效且便捷的机制，可用于定义不同抽象层次上的转换。通常，MLIR 变换可分为三种类型：与非 MLIR 表示之间的导入/导出、不同 MLIR 方言之间的转换，以及单一方言内部的变换。具体而言，转换

```
%r0 = qntm 分配(0)       : !qntm 寄存器
%q0 = qntm 提取 %r0[0]: !qntm 寄存器->!qntm 位
%q1 = qntm 提取 %r0[1]: !qntm 寄存器->!qntm 位
%q2 = qntm 提取 %r0[2]: !qntm 寄存器->!qntm 位
//...
%q3   = qntm 自定义操作"H"() %q0          : !qntm 位
%q4:2 = qntm 自定义操作"受控非门"() %q3, %q1  : !qntm 位,!qntm 位
%q5:2 = qntm 自定义操作"受控非门"() %q4#0, %q2: !qntm 位,!qntm 位
//...
%r2 = qntm 插入 %r0[0], %q5#0: !qntm 寄存器,!qntm 位->!qntm 寄存器
%r3 = qntm 插入 %r6[1], %q4#1: !qntm 寄存器,!qntm 位->!qntm 寄存器
%r4 = qntm 插入 %r7[2], %q5#1: !qntm 寄存器,!qntm 位->!qntm 寄存器
qntm 释放 %r4 : !qntm 寄存器
```

(a) Catalyst Quantum MLIR 方言

```
//...
%r1, %q0 = mqtopt.extractQubit(%r0, %idx0) :
      (!mqtopt.量子比特寄存器, i64) ->
      (!mqtopt.量子比特寄存器, !mqtopt.量子比特)
%r2, %q1 = mqtopt.extractQubit(%r1, %idx1) :
      (! mqtopt.量子比特寄存器, i64) ->
      (! mqtopt.量子比特寄存器, ! mqtopt.量子比特)
//...
%q3 = mqtopt.H门() %q0 : !mqtopt.量子比特
%q4:2 = mqtopt.X门() %q1 ctrl %q3: !mqtopt.量子比特, !mqtopt.量子比特
//...
```

（b）MQTOpt MLIR 方言

图 5：表示同一量子程序（部分内容）的两个 MLIR 方言。

从更高层级的方言或中间表示转换为更低层级的方言，被称为降级转换。

在集成过程中，通过提供一个类型转换器以及若干转换模式，实现了（部分）转换，用于在源量子方言和目标量子方言之间进行转换（同时保持非量子逻辑不变）。与主要由样板代码构成的方言定义相比，此类模式通常需要手动实现，因为它们往往涉及非平凡的重写逻辑——例如，在 MQTOpt 方言中强制执行单次使用约束。

此时需要指出，实现一个方言及其相关转换，已经具备了在 MLIR 框架内支持量子程序编译所需的所有组件。有了这些组件，未来的开发可以（但并非必须）基于 MLIR 的 pass 管理器基础设施，支持高级编译器分析与变换。

示例 5. 要实现量子比特映射例程，可以先定义分析 *pass*，以提取量子比特交互模式，随后使用变换 *pass*，尽可能减少所需的交换 操作数量。*MLIR* 的 *pass* 管理器基础设施支持通过`PassOptions`在运行时进行配置，例如，可以通过键值对指定耦合映射，如图 *4* 所示。

插件：最终，我们的目标是让用户无需编译整个库，甚至整个 MLIR 项目，即可使用上述 MLIR 遍。幸运的是，MLIR 的 pass 管理器基础设施支持注册遍，并动态加载基于插件的方言和遍。要实现这种动态方式，必须将 MLIR DialectPlugin 和 PassPlugin 编译为共享库。编译后的文件随后可以通过多种方式分发，例如，打包成预编译的二进制 wheel 包，类似于传统的编译扩展。

同样，工具开发者无需编译整个 Catalyst 项目即可实现专用插件。在其与 MQT 集成的背景下，该实现只需要 Quantum 方言的头文件，而不必构建整个库。

最后，通过利用共享库插件，可以执行遍流水线——其中指定了完整往返流程所需的所有相关遍。

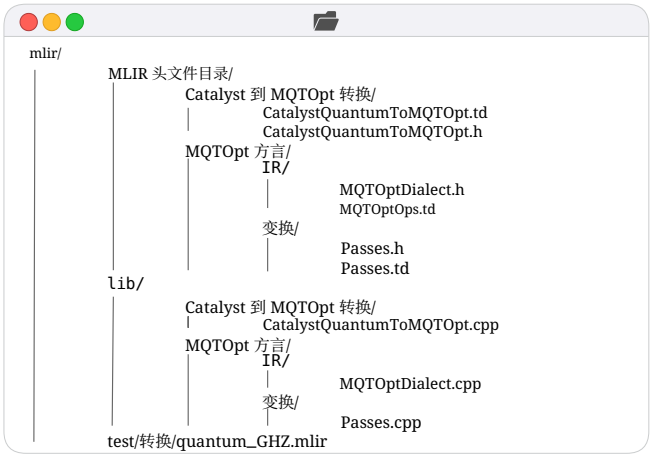
自定义方言的定义、与该方言之间的变换，以及配套的插件基础设施，为将量子软件工具集成到 MLIR 生态系统中提供了所有必要的构建模块。

示例 6。 再次考虑使用 *PennyLane* 定义量子程序并通过 *QMAP* 进行映射的任务——即在 *Catalyst* 与 *MQT* 之间执行完整的往返转换。完整往返流水线的调用示例见图 *6b*。从 `Quantum` 方言中定义的 *GHZ* 电路（如图 *5a* 所示，并在 `quantumGHZ.mlir` 中提供）开始，使用 `catalystquantum-to-mqtopt` 和 `mqtopt-to-catalystquantum` 遍分别完成向 `MQTOpt` 的转换及转换返回。通过 *QMAP* 进行映射，以及导出到 *MQT Core IR* 并从中重新导入，均使用 `mqt-qmap` 遍完成。请注意，耦合映射通过 `cmap` 遍选项指定。整个往返过程可以使用 `mlir-opt` 命令行工具执行；该工具支持通过共享库文件加载方言和通道插件。

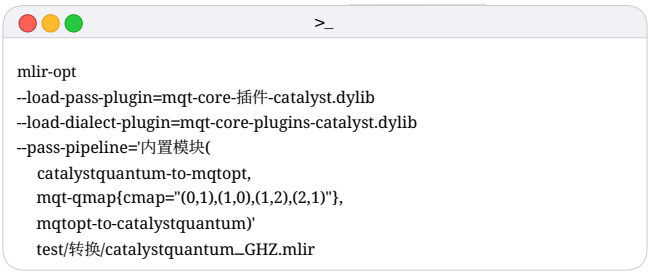
5 项最佳实践

熟悉一个典型 MLIR 集成的基本构建模块后，最初较高的入门门槛已经降低。为了进一步简化采用过程，我们提出了一系列最佳实践。这些实践已被证明对成功实现集成至关重要，也能大幅减少类似项目的开发工作量。我们建议希望在自己的工具中采用 MLIR 的量子软件开发人员将这些实践纳入考量。

首先，不要重复造轮子。MLIR 是一个快速发展的生态系统，汇聚了来自工业界和学术界的贡献，并包含众多维护良好的开源项目。为避免重复实现已有解决方案，建议积极参与更广泛的 MLIR 社区，尤其是通过 Discord 或 GitHub 等平台，并研究现有实现。许多 MLIR 方言，例如算术方言 arith 和结构化控制流方言 scf，提供了成熟且可复用的组件，可直接用于新项目或在其中扩展。为鼓励复用，我们已将本文所述 MLIR 方言和插件的实现作为开源项目发布在 MQT GitHub 仓库中。



(a) MQT 的 MLIR 项目结构。



(b) 使用 MLIR 插件执行一次映射往返。

图 6：基于 MLIR 的项目的典型文件结构及其 mlir-opt 用法概览。

项目结构如图 6a 所示，遵循基于 MLIR 的项目中常见的目录布局。熟悉现有项目有助于避免重复开发，并能为方言设计与实现提供宝贵参考。

其次，应优先考虑模块化、轻量化的设计。如果设计时不够谨慎，量子软件工程师——他们往往并不熟悉 LLVM 或 MLIR 的复杂细节——很容易被其规模和复杂性所压倒（这正是撰写本文的原因）。模块化是现代编译器开发的一项基本原则，它通过提升可维护性、可测试性和可扩展性，帮助减轻软件生态系统日益复杂所带来的负担。实际操作中，这意味着隔离依赖项，避免编译 LLVM 项目等庞大代码库中的全部代码。例如，随着 PennyLane 等全栈框架不断发展，针对大型单体工具链进行编译或链接会变得越来越不切实际。幸运的是，MLIR 的 pass 和插件基础设施使开发者能够将自定义功能整洁地封装起来，而无需深入了解核心系统或对其进行修改。因此，在上述集成工作中，我们将 MQTOpt 方言及其相关遍作为预编译的 MLIR 插件提供。采用模块化、轻量化且基于插件的架构，能够提升可维护性，并降低新贡献者的入门门槛。尽管在某些情况下（例如调试和测试）仍可能需要从源代码构建 MLIR （或其他大型依赖项），但良好的隔离式设计可确保这种情况只是例外，而非常态。

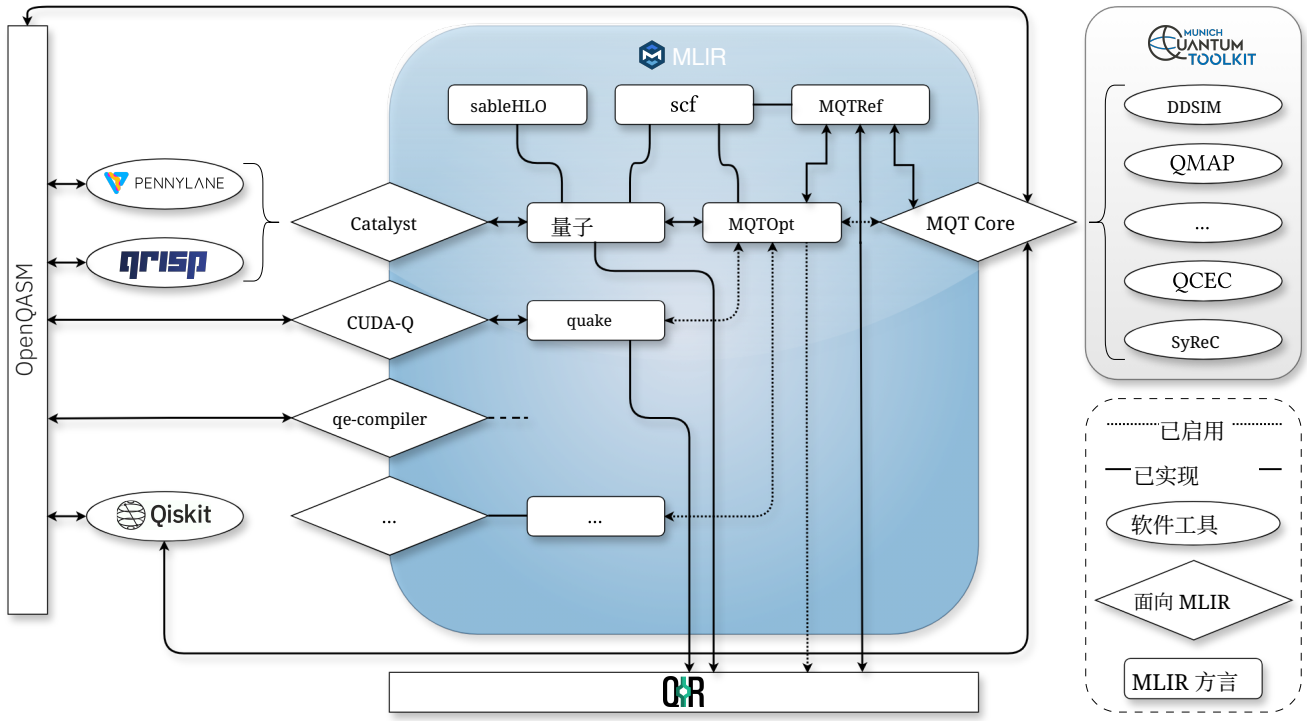
最后，使用专用工具。 LLVM 和 MLIR 提供了种类繁多的实用工具，能够大幅简化量子软件工具的开发。其中有两种工具尤为实用，并贯穿于本文配套的开源代码库中。

这里的目的并非提供完整教程，而是着重介绍这两种工具的用途及其在开发流程中的作用。

首先，借助所谓的 TableGen 工具及其配套定义文件，可以显著简化 MLIR 核心组件的实现。这些文件能够自动生成 MLIR 所需的大量样板代码，包括头文件声明和 getter/setter 方法，从而支持从高度通用到完全自定义（即手动定义）的各种实现。如图 6a 所示，MQTOpt 方言中的操作定义在相应的 MQTOptOps.td 文件中（遵循 MLIR 生态系统中的命名约定）。该定义文件的代码不到 700 行，而 TableGen 生成的对应文件 MQTOptOps.cpp.inc 则包含近 20,000 行复杂的模板代码，这充分体现了 TableGen 所抽象掉的大量实现工作。

其次，mlir-opt（已在示例 6 中简要提及）提供了一个集中式驱动程序，用于连接并协调 MLIR 遍的执行。它的命令行入口点在开发和调试期间尤其有用，能够在不同编译阶段对中间表示进行细粒度检查。结合 LLVM 工具 lit 和 FileCheck，它还可以助力系统化地测试 MLIR 遍和变换。在 PennyLane–MQT MLIR 集成中，专用测试文件（例如 quantumGHZ.mlir）会附带 lit 命令和 FileCheck 指令，用于运行 mlir-opt，并验证应用特定变换后的输出是否正确。总而言之，TableGen 和 mlir-opt 等专用工具不仅减少了手动工作和潜在的实现错误，还能提升代码库的一致性，并支持采用自动化测试工作流。

遵循这些最佳实践，有助于实现高效且可持续的 MLIR 集成，其短期和长期优势将在下文讨论。



Fig图7：MLIR 生态系统内外量子计算软件工具的分布图。

6 讨论

前文各节概述了如何高效地将量子软件工具集成到 MLIR 框架中，从而显著降低量子软件开发者的入门门槛。接下来将讨论集成过程中获得的关键见解，以及未来工作的方向。尽管以下要点源自 MQT 集成，但这些见解普遍适用于任何希望采用 MLIR 的量子软件工具。

6.1 见解

在确立成功实现 MLIR 集成所需的关键组件和最佳实践后，我们现在退一步，评估这项工作的直接影响。为此，请看图 7，它展示了 Catalyst 与 MQT Core 之间新近实现的交互，而这一切完全是在 MLIR 框架内实现的。

集成后的工具如今已成为一个主要由社区开发的生态系统的一部分，加入了不断壮大的 MLIR 采用者群体。因此，它受益于 MLIR 生态系统持续提供的支持与进步。具体而言，MLIR 社区专家带来的性能改进和更新，会直接提升使用 MLIR 基础设施的工具的性能。

另一项优势是提升了双方用户的易用性——无论是使用 PennyLane 的用户，还是使用 MQT 的用户。例如，量子算法专家现在可以专注于实现自己的算法，使用简洁直观的 Catalyst 装饰器与 QMAP 进行映射，如图 4 所示，而不必再经历图 3 所示的繁琐工作流程。

借助新的集成，Catalyst 与 MQT Core 之间的交互实现了第三方量子计算工具之间的互操作性；此前，这些工具只能通过 OpenQASM 等基于文本的交换格式连接。一方面，基于 Catalyst 编译器构建的工具（例如 Qrisp [53]，）现在可以直接与 MQT 编译工具交互。尽管截至本文撰写时尚未对此进行测试，但预计所需工作量很小。另一方面，任何基于 MQT Core 库构建的工具——例如 DDSIM [10] 或 QCEC [46]——现在都可以处理 MLIR 输入。这种连接直接促进了这些软件工具在编译层面的互操作性，显著拓展了可组合量子编译工作流的范围。

6.2 展望

从更宏观的视角来看图7，基于 MLIR 的集成还显现出其他长期优势。具体而言，这种集成能够轻松扩展以支持各种量子软件工具，并与之实现无缝互操作，还为更广泛地集成到成熟的软件栈中铺平了道路。例如，现在只需付出极少的额外努力，即可与其他基于 MLIR 的框架集成，例如 NVIDIA 的 CUDA-Q。在这种情形下，只需实现 MQTOpt 方言与 CUDA-Q 的 quake 方言之间的转换通道即可；现有的模块化代码库使这项扩展能够以极低的成本完成。

此外，现在也可以轻松集成 scf 等经典方言。例如，Catalyst 的追踪功能已能从 scf 提供的循环等结构化控制流结构中显著受益。扩展 MQTOpt 方言以支持结构化控制流，将能够实现更紧凑的程序表示，并提高编译遍的适用性。例如，由重复模式组成的量子电路可以用固定大小的程序表示，这一优势对于 Grover算法等算法的复杂实例尤为重要。此外，在这些情况下，可能只需映射单次循环迭代中的固定量子比特相互作用，而不必展开整个电路并多次冗余映射相同的片段；这种重复映射在量子机器学习电路中很常见。

更广泛地说，对于许多考虑未来集成 MLIR 的量子软件工具，这一观察同样成立。电路分析及后续变换等任务通常计算密集，且多以 Python 实现，因此能够从 MLIR 提供的高效、结构化程序表示中受益匪浅。将此类功能转移至 MLIR 层，可以显著提升性能和可扩展性。

从图7还可以明显看出，迄今为止，Qiskit 与 MLIR 生态系统的联系仍相对薄弱。相关的 qe-compiler [23], 可将 OpenQASM 转换为多个专用 MLIR 方言，但尚未得到广泛采用，且基本处于无人维护的状态。与此同时，MQT Core 已支持在 Python 层面与 Qiskit 直接集成。随着 Qiskit 持续转向基于 Rust 的后端，如何直接连接 MLIR 与 Qiskit 生态系统，成为一个颇具挑战性的问题。近期为 Qiskit 编译组件开发的 C API 或许能为此开辟一条可行路径，使 MQT Core 成为推动两个生态系统集成的理想候选。

最后，本文展示的 MLIR 插件为使用 QIR 作为交换格式实现统一交互铺平了道路。由于 QIR 构建于 LLVM 中间表示之上，它已被用于在高性能计算软件栈中部署量子加速器 [54]。出于同样的原因，QIR 与 MLIR 兼容，可以轻松集成到基于 MLIR 的编译流程中。因此，MLIR 集成工具（如 MQT）可以直接应用于 QIR文件。反过来，任何已支持 QIR 的工具也都可以轻松纳入该流程，从而构建模块化且可扩展的量子软件栈。

总而言之，这些即时和长期收益表明，降低寻求采用 MLIR 框架的量子软件工程师的入门门槛，是一项值得开展的工作。他们的集成工作将显著提升互操作性，并为构建更加互联的量子计算生态系统奠定基础，最终推动其融入经典软件栈和高性能计算软件栈。

7 结论

本工作展示了如何利用 MLIR 的方言和 Pass 插件系统，开发一种模块化且轻量级的编译骨干，并以数十年来积累的经典编译器专业知识为基础。为说明这一点，我们给出了一个集成两个主要量子计算框架的实际示例，用简洁的基于 MLIR 的解决方案取代了以往繁琐且临时性的解析机制。鉴于 MLIR 为量子软件开发者带来了陡峭的学习曲线，我们提供了源自此次集成工作的最佳实践和见解。最终实现以开源软件形式提供，旨在为量子计算社区更广泛范围内的类似集成工作提供基础和启发。归根结底，本工作为不熟悉 MLIR、希望将其工具与该框架集成，从而改善与其他软件工具以及经典和高性能计算基础设施之间互操作性的量子软件工程师，提供了必要的指导。这样做显著降低了量子编译采用 MLIR 传统上所需的入门门槛，并为其集成到成熟的高性能计算软件栈中铺平了道路。

致谢

隶属于慕尼黑工业大学的作者感谢欧洲研究委员会（ERC）通过欧盟地平线2020研究与创新计划提供的资助（DA QC，资助协议编号 101001318；MILLENION，资助协议编号 101114305），以及慕尼黑量子谷的支持。慕尼黑量子谷由巴伐利亚州政府通过巴伐利亚高科技议程 Plus 提供资金支持。本研究获得了 BMK、BMDW、上奥地利州在 COMET 计划框架下提供的资助，也获得了 BMFTR 在 FullStaQD 项目下的资助；此外，本研究还基于德国联邦议院的决议，通过 SYNQ 项目获得支持，并获得德国研究联合会（DFG，德国研究基金会——563436708）的资助。

Xanadu 的作者感谢 Xanadu 的 Catalyst 和 PennyLane 团队就 Catalyst 的 MLIR 插件功能进行了有益的讨论和审阅，使得本实现得以完成；参与者包括但不限于 David Ittah、Mehrdad Malek、Paul HaochenWang、Joey Carter 和 Raul Torres。所有作者都感谢 MLIR 和 LLVM 的贡献者付出的辛勤努力。

参考文献

[1] Martín Abadi、Ashish Agarwal、Paul Barham、Eugene Brevdo、Zhifeng Chen、Craig Citro 等。2015。TensorFlow：异构系统上的大规模机器学习。https://www.tensorflow.org/ 软件。[2] Matthew Amy、Dmitri Maslov 和 Michele Mosca。2014。通过拟阵划分优化 Clifford+T 电路的多项式时间 T 深度。IEEE 集成电路与系统计算机辅助设计汇刊 33，10（2014），1476–1489。https://doi.org/10.1109/TCAD.2014.2341953[3] Juan Miguel Arrazola、Thomas R Bromley、Josh Izaac、Casey R Myers、Kamil Brádler 和 Nathan Killoran。2019。光子量子计算机上的状态制备和门综合机器学习方法。《量子科学与技术》4，2（2019），024004。https://doi.org/10.1088/2058-9565/aaf59e [4] Ali Asadi、Amintor Dusko、Chae-Yeun Park、Vincent Michaud-Rioux、Isidor Schoch、Shuli Shu、Trevor Vincent 和 Lee James O'Riordan。2024。利用 PennyLane Lightning 在高性能计算平台上进行混合量子编程。arXiv: 2403 02512 [ ] 2403 02512 . quant-ph https://arxiv.org/abs/ 。[5] Bao Bach、Ilya Safro 和 Ed Younis。2025。. 通过位置图架构抽象，为穿梭式离子阱机器实现高效编译。arXiv:2501.12470 [quant-ph] https://arxiv.org/abs/2501.12470 arXiv:2501.12470。[6] Medina Bandic、Carmen G Almudever 和 Sebastian Feld。2023。基于相互作用图的量子基准测试表征，以改进量子电路映射技术。《量子机器智能》5，2（2023），40。[7] Ville Bergholm、Josh Izaac、Maria Schuld、Christian Gogolin、Shahnawaz Ahmed、Vishnu Ajith 等。2022。PennyLane：量子-经典混合计算的自动微分。arXiv:1811.04968 [quant-ph] https://arxiv.org/abs/1811.04968 arXiv:1811.04968。[8] James Bradbury、Roy Frostig、Peter Hawkins、Matthew James Johnson、Chris Leary、Dougal Maclaurin、George Necula 等。2018。JAX：Python+NumPy 程序的可组合变换。http://github.com/jax-ml/jax 软件。[9] L. Burgholzer、J. Echavarria、P. Hopf、Y. Stade、D. Rovara、L. Schmid、E. Kaya、B. Mete、M. N. Farooqi、M. Chung、M. De Pascale、L. Schulz、M. Schulz 和 R. Wille。2026。慕尼黑量子软件栈：连接终端用户、整合多种量子技术、加速高性能计算。发表于 SCA/HPCAsia：亚洲超级计算及亚太地区高性能计算国际会议。https://doi.org/ 10 1145 3773656 3773669 . / 。Lukas Burgholzer、[10] 2022 Alexander Ploier 和 Robert Wille。. 利用任意路径和决策图模拟量子电路。发表于《2022 年欧洲设计、自动化与测试会议论文集》（比利时安特卫普）（DATE '22）。欧洲设计与自动化协会，64–67。[11] Lukas Burgholzer、Rudy Raymond 和 Robert Wille。2020。验证 IBM Qiskit 量子电路编译流程的结果。发表于量子计算与工程国际会议。https://doi.org/ 10 1109 49297 2020 00051 . / QCE 。Lukas Burgholzer、Yannick Stade、Tom Peham 和 Robert Wille。. MQT Core：[12] 2025 慕尼黑量子工具包（MQT）的核心。《开放源代码软件期刊》 10, 108 2025 7478 10 21105 07478 ( ), 。[13] https://doi.org/ 。/joss。F. Javier Cardama、Jorge Vázquez-Pérez、César Piñeiro、Juan C. Pichel、Tomás F. Pena 和 Andrés Gómez。2025。量子计算中间表示综述。《超级计算期刊》81，2（2025），418。https://doi.org/ 10 1007 11227 024 06892 2 . /s - - - Andrew Cross、Ali Javadi-Abhari、[14] Thomas Alexander、Niel De Beaudrap、Lev S. Bishop、Steven Heidel 等。2022。OpenQASM3：更广泛、更深入的量子汇编语言。《ACM量子计算汇刊》3，3，文章12（ 2022 50 10 1145 3505636 ），页。https://doi.org/ 。/ Andrew [15] W Cross、Lev S Bishop、John A Smolin 和 Jay M Gambetta。2017。开放量子汇编语言。arXiv 预印本 arXiv:1707.03429（2017）。[16] Amr Elsharkawy、Xiao-Ting Michelle To、Philipp Seitz、Yanbin Chen、Yannick Stade、Manuel Geiger、Qunsheng Huang、Xiaorang Guo、Muhammad Arslan Ansari、Christian B. Mendl、Dieter Kranzlmüller 和 Martin Schulz。2025。高性能计算与量子加速器的集成——量子编程工具综述。《ACM量子计算汇刊》6，3，文章 24 2025 46 10 1145 3743149 （7月），页。[17] https://doi.org/ 。/ Mathieu Fehr、Michel Weber、Christian Ulmann、Alexandre Lopoukhine、Martin Paul Lücke、Théo Degioanni 等。2025。xDSL：面向基于 SSA 的编译器的 Sidekick 编译。发表于第23届 ACM/IEEE代码生成与优化国际研讨会论文集（美国内华达州拉斯维加斯）（CGO '25）。计算机协会， 179 192 10 1145 3696443 3708945 – 。[18] https://doi.org/ 。/ 。Brett Giles 和 Peter Selinger。. 多量子比特 Clifford+T 电路的精确综合。《物理评论A》87，3（2013），032332。https://doi.org/10.1103/PhysRevA.87.032332[19] Daniel Gottesman。1997。稳定子码与量子纠错。加州理工学院。[20] Thomas Grurl、Jürgen Fuß 和 Robert Wille。2025。利用决策图模拟量子电路。Springer Nature 瑞士，21–35。https://doi.org/10. 1007/978-3-031-71036-0_3 [21] Harshit Gupta、Rohan Jain、Samuel Kushnir、Priyansh Parakh 和 Ryan James Hill。2025。qBraid-QIR：用于 QIR 转换、集成和实用工具的 Python 软件包。https://github.com/qBraid/qbraid-qir 软件。[22] Johannes Hauschild 和 Frank Pollmann。2018。使用张量网络进行高效数值模拟：Tensor Network Python (TeNPy)。《SciPost物理学讲义》（ 2018 10 21468 5 ）。https://doi.org/ 。/scipostphyslectnotes。[23] Michael B. Healy、Reza Jokar、Soolu Thomas、Vincent R. Pascuzzi、Kit Barton、Thomas A. Alexander、Roy Elkabetz、Brian C. Donovan、Hiroshi Horii 和 Marius Hillenbrand。2024。IBM量子引擎编译器的设计与架构。arXiv预印本 arXiv:2408.06469（2024）。arXiv:2408.06469 [quant-ph] https://arxiv.org/abs/ 2408 06469 。Patrick Hopf、Nils Quetschlich、Laura Schulz 和 Robert Wille。[24] 2025。改进量子电路编译的评价指标。载于：2025年欧洲设计、自动化与测试会议（DATE）。1–7。https://doi.org/10.23919/DATE64628.2025.10992761[25] Thomas Häner、Torsten Hoefler 和 Matthias Troyer。2020。量子程序的基于断言的优化。《ACM编程语言会议录》 4, 2020 1 20 10 1145 3428201 OOPSLA（），–。[26] https://doi.org/。/ Toshinari Itoko、Rudy Raymond、Takashi Imamichi 和 Atsushi Matsuo。2020。使用门变换和交换优化量子电路映射。《集成VLSI期刊》70，C（2020），43–50。https://doi.org/10.1016/j.vlsi.2019.10.004[27] David Ittah、Ali Asadi、Erick Ochoa Lopez、Sergei Mironov、Samuel Banning、Romain Moyard、Mai Jacob Peng 和 Josh Izaac。2024。Catalyst：用于自动微分混合量子程序的 Python 即时编译器。《开放源代码软件期刊》 9, 99 2024 6720 10 21105 06720 （），。https://doi.org/。/joss。[28] David Ittah、Thomas Häner、Vadym Kliuchnikov 和 Torsten Hoefler。2022。QIRO：一种基于静态单赋值、用于优化的量子程序表示。《ACM量子计算汇刊》3，3（2022），1–32。https://doi.org/ 10 1145 3491247 。/ Ali [29] Javadi-Abhari、Matthew Treinish、Kevin Krsulich、Christopher J. Wood、Jake Lishman、Julien Gacon 等。2024。使用 Qiskit 进行量子计算。arXiv:2405.08810 [quant-ph] https://arxiv.org/abs/2405.08810 arXiv:2405.08810。[30] Aleks Kissinger 和 John van de Wetering。2020。PyZX：大规模自动图式推理。《电子理论计算机科学会议录》 318 2020 229 241 10 4204 318 14 （），–。https://doi.org/。/ eptcs.。[31] A. Yu Kitaev。1997。。量子计算：算法与纠错。《俄罗斯数学调查》52，6（1997），1191–1249。https://doi.org/10.1070/RM1997v052n06ABEH002155[32] Mark Koch、Agustín Borgna、Seyon Sivarajah、Alan Lawrence、Alec Edgington、Douglas Wilson 等。[n. d.]。HUGR：量子-经典中间表示。软件。[33] Fabian Kreppel、Christian Melzer、Diego Olvera Millán、Janis Wagner、Janine Hilder、Ulrich Poschinger 等。2023。用于基于穿梭操作的囚禁离子量子计算机的量子电路编译器。《量子》7（2023），1176。https://doi.org/ 10 22331 2023 11 08 1176 。/q- - - - Chris Lattner 和 Vikram Adve。2004。[34] LLVM：用于长期程序分析与变换的编译框架。载于：国际代码生成与优化研讨会论文集：反馈驱动与运行时优化（美国加利福尼亚州帕洛阿尔托）（CGO '04）。IEEE计算机学会，75。[35] Chris Lattner、Mehdi Amini、Uday Bondhugula、Albert Cohen、Andy Davis、Jacques Pienaar 等。2020。MLIR：面向摩尔定律终结的编译器基础设施。arXiv:2002.11054 [cs.PL] https://arxiv.org/abs/2002.11054 arXiv:2002.11054。[36] Chris Lattner、Mehdi Amini、Uday Bondhugula、Albert Cohen、Andy Davis、Jacques Pienaar 等。2021。MLIR：扩展面向领域特定计算的编译器基础设施。载于：2021年IEEE/ACM国际代码生成与优化研讨会（CGO）。2–14。https://doi.org/10.1109/CGO51591.2021.9370308[37] Xiu-Zhe Luo、Jin-Guo Liu、Pan Zhang 和 Lei Wang。2020。Yao.jl：用于量子算法设计的可扩展、高效框架。《量子》4（2020），341。https://doi.org/ 10 22331 2020 10 11 341 。/q- - - - Alexander J. McCaskey、Dmitry I. Lyakh、Eugene F. Dumitrescu、Sarah S. Powers 和 Travis S. Humble。2020。XACC：面向异构量子-经典计算的系统级软件基础设施。《量子科学与技术》 5, 2 2020 10 1088 2058 9565 6 6 （）。https://doi.org/。/ - /ab bf Alexander McCaskey 和 Thien Nguyen。。用于量子汇编语言的 MLIR 方言。载于：[39] 2021 2021年IEEE量子计算与工程国际会议（QCE）。IEEE计算机学会，255–264。https://doi.org/ 10 1109 52317 2021 00043 。/QCE . . Arianne Meijer van de Griend。。[40] 2025 使用基于DAG或基于相位多项式的中间表示的量子编译器比较。《系统与软件期刊》 221, 2025 14 10 1016 2024 112224 C （），页码。https://doi.org/。/j.jss。[41] Siyuan Niu、Akel Hashim、Costin Iancu、Wibe Albert De Jong 和 Ed Younis。2024。通过 BQSKit 中的电路缩减实现有效的量子资源优化。载于：第61届ACM/IEEE设计自动化会议论文集（美国加利福尼亚州旧金山）（DAC '24）。计算机协会，纽约州纽约市，美国，

[42] Siyuan Niu、Akel Hashim、Costin Iancu、Wibe Albert De Jong 和 Ed Younis。2024。通过 BQSKit 中的电路缩减实现有效的量子资源优化。载于：第*61*届*ACM/IEEE*设计自动化会议论文集（美国加利福尼亚州旧金山）（*DAC '24*）。计算机协会，第321条，6页。https://doi.org/10.1145/3649329.3656534[43] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan 等。2019。PyTorch：一种命令式风格、高性能的深度学习库。载于：第*33*届神经信息处理系统国际会议论文集。Curran Associates Inc.，第721条，12页。[44] Anurudh Peduri、Siddharth Bhat 和 Tobias Grosser。2022。QSSA：一种用于量子计算的、基于 SSA 的中间表示。载于：第*31*届*ACM SIGPLAN*国际编译器构造会议论文集（*CC '22*）。ACM，2–14。https://doi.org/ 10 1145 3497776 3517772 。/。[45] Tom Peham、Nina Brandl、Richard Kueng、Robert Wille 和 Lukas Burgholzer。2023。使用 SAT 求解器综合深度最优的 Clifford 电路。载于：*2023*年*IEEE*量子计算与工程国际会议（*QCE*）。https://doi.org/ 10 1109 57702 2023 00095 2305 01674 。/QCE . . arXiv: . [46] Tom Peham、Lukas Burgholzer 和 Robert Wille。。参数化量子电路的等价性检查：验证变分量子算法的编译。载于：第*28*届亚洲及南太平洋设计自动化会议论文集（*ASPDAC '23*）。ACM，702–708。https://doi.org/10.1145/3566097.3567932[47] B Pirvu、V Murg、J I Cirac 和 F Verstraete。2010。矩阵乘积算符表示。《新物理学杂志》12，2（2010），025012。https://doi.org/10.1088/1367-2630/12/2/025012[48] QIR联盟：https://qir-alliance.org [n. d.]。*QIR*规范。QIR联盟：https://qir-alliance.org。https://github.com/qir-alliance/qir-spec [49] N. Quetschlich、L. Burgholzer 和 R. Wille。2025。MQT Predictor：通过设备专用电路编译自动选择量子计算设备。《*ACM*量子计算汇刊》（*TQC*）（2025）。https://doi.org/ 10 1145 3673241 2310 06889 。/ arXiv: . [50] Aaron Sander、Lukas Burgholzer 和 Robert Wille。2024。通过中间矩阵乘积算符检查量子电路的等价性。*arXiv*预印本 *arXiv:2410.10946*（2024）。[51] Ludwig Schmid、Sunghye Park 和 Robert Wille。2024。混合电路映射：充分利用中性原子量子计算机的全部计算能力。载于：第*61*届*ACM/IEEE*设计自动化会议论文集。1–6。[52] Daniel Schoenberger、Stefan Hillmich、Matthias Brandl 和 Robert Wille。2024。面向可扩展囚禁离子量子计算机的穿梭操作。*IEEE* 集成电路与系统计算机辅助设计汇刊（2024），1–1。https://doi.org/10.1109/TCAD.2024.3513262[53] Raphael Seidel、Sebastian Bock、René Zander、Matic Petrič、Niklas Steinmann、Nikolay Tcholtchev 和 Manfred Hauswirth。2024。Qrisp：用于基于量子门的量子计算机可编译高级编程的框架。arXiv: 2406 14792 [ ] 2406 14792 。quant-ph [54] https://arxiv.org/abs/。Amir Shehata、Peter Groszkowski、Thomas Naughton、Muralikrishnan Gopalakrishnan Meena、Elaine Wong、Daniel Claudino、Rafael Ferreira da Silva 和 Thomas Beck。2026。弥合范式：面向高性能计算与量子计算融合的设计。《未来一代计算机系统》174（2026年1月），107980。https://doi.org/10.1016/j.future.2025.107980[55] Seyon Sivirajah、Silas Dilkes、Alexander Cowtan、Will Simmons、Alec Edgington 和 Ross Duncan。2020。TKET：面向 NISQ设备的可重定向编译器。《量子科学与技术》6，1（2020），014003。https://doi.org/10.1088/2058-9565/ab8e92[56] Yannick Stade、Lukas Burgholzer 和 Robert Wille。2024。迈向 QIR 支持：采用量子中间表示的思考。arXiv: 2411 18682 [ ] 2411 18682 。quant-ph [57] https://arxiv.org/abs/。CUDA-Q开发团队。n. d.。CUDA-Q。https://github.com/NVIDIA/cuda-quantum[58] Davide Venturelli、Minh Do、Bryan O'Gorman、Jeremy Frank、Eleanor Rieffel、Kyle EC Booth、Thanh Nguyen、Parvathi Narayan 和 Sasha Nanda。2019。量子电路编译：自动推理的新兴应用。载于：调度与规划应用研讨会。[59] Hanrui Wang、Pengyu Liu、Jinglei Cheng、Zhiding Liang、Jiaqi Gu、Zirui Li、Yongshan Ding、Weiwen Jiang、Yiyu Shi、Xuehai Qian 等。2022。Quest：用于量子电路可靠性估计的图变换器。*arXiv*预印本 *arXiv:2210.16724*（2022）。[60] Robert Wille、Lucas Berent、Tobias Forster、Jagatheesan Kunasaikaran、Kevin Mato、Tom Peham 等。2024。MQT手册：量子计算设计自动化工具与软件综述。载于：*IEEE*量子软件国际会议（*QSW*）。https://doi.org/10.1109/QSW62656.2024.00013 arXiv:2405.17543[61] Robert Wille 和 Lukas Burgholzer。2023。MQT QMAP：高效量子电路映射。载于：*2023*年国际物理设计研讨会论文集（*ISPD '23*）。ACM，198–204。https://doi.org/10.1145/3569052.3578928

[62] Robert Wille、Stefan Hillmich 和 Lukas Burgholzer。2022。基于决策图的量子计算工具。《*ACM*量子计算汇刊》 3, 3, 13 2022 17 10 1145 3491246 文章（），[63] 2023 页码。https://doi.org/。/ Robert Wille、Stefan Hillmich 和 Lukas Burgholzer。《量子计算的决策图》。Springer International Publishing，1–23。https://doi.org/10.1007/978-3-031-15699-1_1[64] Elaine Wong、Vicente Leyton Ortega、Daniel Claudino、Seth Johnson、Sharmin Afrose、Meenambika Gowrishankar 等。2024。面向量子中间表示的跨平台执行引擎。*arXiv* 预印本 *arXiv:2404.14299*（2024）。arXiv:2404.14299 [量子物理] https://arxiv.org/abs/2404. 14299[65] W. K. Wootters 和 W. H. Zurek。1982。单个量子不可克隆。《自然》 299, 5886 1982 802 803 10 1038 299802 0 （），–。[66] 2021 https://doi.org/。/ a Ed Younis、Koushik Sen、Katherine Yelick 和 Costin Iancu。Qfast：融合搜索与数值优化，实现可扩展的量子电路综合。载于*2021 IEEE*量子计算与工程国际会议（*QCE*）。[67] Alwin Zulehner、Stefan Hillmich 和 Robert Wille。2019。如何高效处理复数值？为量子计算实现决策图。载于*2019 IEEE/ACM*计算机辅助设计国际会议（*ICCAD*）。IEEE，1–7。https://doi.org/10.1109/iccad45719.2019.8942057