TeD-Q：张量网络增强的分布式混合量子机器学习框架

Yaocheng Chen[1,2,3],Chung-Yun Kuo[1,2,3],Yuxuan Du[3],Dacheng Tao[3],和 Xingyao W u[3]

[1]物理系，国立台湾大学，台北，台湾[2]梁兆新宇宙学与粒子天体物理研究中心，国立台湾大学，台北，台湾[3]京东探索研究院，中国北京

arXiv:2301.05451v2 [quant-ph] 8 Dec 2024

TeD-Q 是一个用于量子机器学习（QML）、变分量子算法（VQA）和量子计算模拟的开源软件框架。它将经典机器学习库与量子模拟器无缝集成，使用户能够在训练量子机器学习模型的同时，充分利用经典机器学习的强大能力。TeD-Q 支持自动微分，可通过反向传播、参数移位和有限差分法获取梯度。借助张量收缩，可以模拟包含大量量子比特的量子电路。TeD-Q 还提供图形化模式，可实时显示量子线路和训练进度。

1 引言

量子计算是最有希望帮助我们摆脱摩尔定律束缚的方向之一。近期结果表明，在某些问题上，量子计算确实优于经典计算 [1, 2, 3]。随着量子硬件的发展，量子软件和算法研究也在不断跟进。量子软件在连接量子硬件与现实世界应用方面发挥着至关重要的作用。目前

Yaocheng Chen：共同贡献；本工作是在他于京东探索研究院担任研究实习生期间完成的。Yuxuan Du：duyuxuan123@gmail.com，通讯作者

Dacheng Tao：dacheng.tao@gmail.com，通讯作者
Xingyao Wu：wu.x.yao@gmail.com，共同贡献；通讯作者

近年来，量子软件平台领域已涌现出相当多的研究成果。有些平台已允许用户访问真正的量子硬件 [4, 5, 6], ，而另一些则更侧重于构建完整的软件环境，以更好地开发量子算法 [7]。

TeD-Q 是一个基于 Python、具备可微功能的量子编程框架。它尤其针对量子机器学习问题 [8, 9, 10, 11] 和变分量子算法 [12, 13, 14], 进行了优化，这些问题和算法构成了当代含噪中尺度量子（NISQ）算法 [15, 16]的大部分。TeD-Q 为不同后端提供了通用的编程框架，包括量子硬件、量子模拟器和分布式 GPU 加速器。在顶层应用方面，用户可以方便地使用 TeD-Q，无需担心如何处理不同的后端。用户只需像进行经典机器学习一样训练量子神经网络（QNN）。

在 TeD-Q 中，量子电路被视为一个 Python 函数。用户可以自由选择 PyTorch 或 JAX 作为与执行后端交互的接口。得益于这种系统化集成，还可以利用成熟的面向人工智能的库 [17] 所提供的丰富功能，更好地实现量子算法。具体优势包括批量执行、即时编译（JIT） [18], 、分布式并行图形处理器优化，以及用于反向传播的自动微分。

TeD-Q 同时支持全振幅量子模拟和经张量网络增强的多振幅模拟。TeD-Q 会估算这两种模式的复杂度，用户可以自行选择要执行代码的模式。

TeD-Q 还内置了张量网

络模块和收缩路径优化器 JDtensorPath，可在张量网络模拟模式下进一步提升性能。这是通过优化张量网络收缩顺序实现的。

此外，TeD-Q 还提供图形化电路编辑器，通过直观的可视化界面让初学者也能轻松上手。

表 1 展示了 TeD-Q 与三个主要的通用量子机器学习平台——PennyLane [4],Qiskit [5] 和 Paddle Quantum [19] 之间的比较。

PennyLane 是一个跨平台的 Python 软件包，用于编写量子算法和量子-经典混合算法。它基于自动微分编程理念，将主流经典机器学习库与量子算法相结合，提供易于使用的接口[4]。Qiskit 是 IBM 的开源量子软件开发框架。它集成了 PyTorch，可通过 IBMQ 在真实量子硬件上运行[5]。Paddle Quantum 由百度开发，基于 PaddlePaddle 深度学习平台，并可接入名为 Quantum Leaf 的真实量子计算机 [19]。

表 1：现有量子模拟框架的功能概述。

	TeD-Q	PennyLane	Qiskit	Paddle Quantum
反向传播	v	v	-	v
基于超图的TNC	v	-	-	-
切片并行化	v	-	-	-
PyTorch接口	v	v	v	-
噪声模型	未来版本	v	v	v
硬件兼容性	通过 Qiskit, Quafu	v	v	v
电路编排器	v	-	v	-

TeD-Q 的主要特色是基于超图的张量网络收缩（TNC）以及通过指标切片实现并行化。量子电路模拟通常可通过矩阵乘法和张量网络收缩来完成。目前的量子软件平台通常只支持矩阵乘法模式 [4, 5, 19], ，当量子比特数量约低于 38 时，这种模式已足够。然而，当量子比特数量超过 38 时，时间成本和存储成本会呈指数级飙升。TeD-Q 的 TNC 模式可以轻松处理量子比特数量大于 38 的量子电路模拟。基于超图的 TNC 方法可在计算和内存复杂度方面带来数个数量级的改进，而通过指标切片实现并行化，则有助于提升 GPU 设备通信效率。由于张量网络收缩顺序会显著影响时间成本，TeD-Q 还提供了一款专用的张量网络收缩优化器，名为 JDTensorPath。这些新特性旨在提升大规模量子电路模拟的效率。

与 Qiskit、PennyLane 和 Paddle Quantum 相比，TeD-Q 还具有可重复使用的机制。在大多数量子算法（包括 NISQ算法）中，参数化量子电路会在不改变线路结构的情况下被多次评估。在 TeD-Q 中，量子电路内部的量子态和量子门对象会在每次评估时重复使用。对于小型量子电路，这一特性可以显著减少类实例化所需的时间。在第 6 节中多量子比特旋转示例的性能比较，可以轻松看出这一优势。

2 TeD-Q 的架构

TeD-Q 旨在快速、高效地实现量子算法。为此，应用设计组件、计算后端以及两者之间的接口被封装为相对独立的模块。TeD-Q 由四个主要模块组成：QInterpreter、Backends、Interface 和 Optimizer，如图 1 所示。

任何量子算法最终都会以经典代码的形式传递并处理。QInterpreter 在此发挥作用，将量子算法转换为可执行代码。量子态类和量子门类是 QInterpreter 的基本构件。对量子态施加一系列量子门，即可构成量子电路。

TeD-Q 中的量子电路对象不依赖于模拟或硬件后端。它是用户定义的量子函数的一种抽象，用于连接高层量子算法与低层特定于后端的指令。TeD-Q 的 QInterpreter 提供了获取和操作量子算子及测量表示形式的工具。这些工具和表示形式有助于生成量子电路。用户需要提供

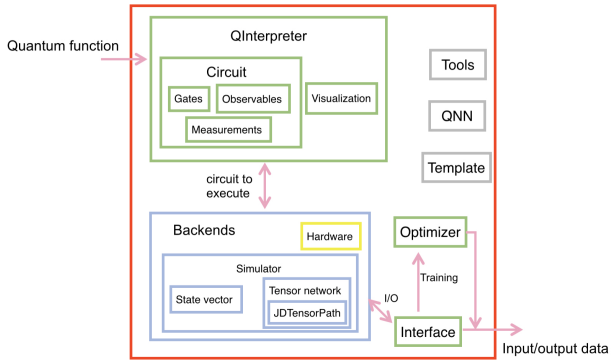


图 1：TeD-Q 架构设计。黄色部分为量子部分，其余部分为经典处理。

一个量子函数、量子比特数量以及参数或参数的形状（在这种情况下，将生成并使用随机参数），以构建量子电路。

量子函数定义为普通的 Python 函数。它接受经典输入参数，并使用其中的元素作为量子门参数。用户需要按照量子操作的应用顺序，将量子操作放入量子函数中。需要指定一个关键字参数 "qubit"，其值为一个整数列表，用于指明每个量子操作应用于哪个量子比特。在所有量子操作之后，必须在 return 语句中放置一个测量结果或测量结果列表。代码清单 1展示了一个定义量子函数并据此构建量子电路的简单示例。

```
import tedq

# Define quantum circuit
def circuitDef(params):
    tedq.RX(params[0], qubits=[0])
    tedq.RY(params[1], qubits=[1])
    tedq.CNOT(qubits=[0, 1])
    return [tedq.expval(tedq.PauliZ(qubits
    =[0]))
            tedq.expval(tedq.PauliZ(qubits
    =[1]))]

number_of_qubits = 2
parameter_shapes = [(2,)]

# Quantum circuit construction
circuit = tedq.Circuit(circuitDef,
    number_of_qubits, parameter_shapes =
    parameter_shapes)
```

代码清单 1：如何构建量子电路。第 4–9 行定义了一个量子电路函数，第 15 行则构造了一个 TeD-Q 量子电路对象。

该量子电路尚不可执行，因为计算模式和后端尚未选定。指定这些设置后，即可将其编译为可执行的编译后电路。编译后的电路可像标准 Python 函数一样用于计算结果。输入参数的顺序必须与相应量子门的顺序一致。按照代码清单 2. 所示，通过手动使用不同的规格编译量子电路，即可在不同后端上运行它。这就是它被称为可复用量子电路的原因。

```
# compile quantum circuit with "jax" backend

"j

ba
"pytorch")
```

代码清单 2：使用不同的后端编译量子电路。

3 模拟模式

3.1 态矢量传播模式

在量子理论中，波函数用于对系统的量子态进行数学描述。态矢量表示为一个复数 ket 向量 $|\psi\rangle$。在此模式下，量子门以一系列矩阵算子的形式作用于态矢量，因此每次操作后的状态为：

$$|\psi\rangle = U|\psi_0\rangle, \tag{1}$$

其中 U 是矩阵算子。态矢量传播模式是 TeD-Q 的默认模拟后端。如代码清单 3 所示，用户可以在量子电路定义函数的第一行添加 tedq.InitStateVector()，以使用自定义初始量子态。请注意，初始态矢量的长度必须与 2^n一致，其中 n 是量子比特数。如果未指定初始量子态，TeD-Q 默认将其设为基态 $|0\rangle^{\otimes n}$ 。使用 TeD-Q，此模式可让普通笔记本电脑处理多达 20 个量子比特的非平凡量子电路。

```
# Prepare quantum state vector

sq
initial quantum state
```

```
    # other circuit definition
```

代码清单 3：使用用户定义的初始量子态定义量子电路。

3.2 张量网络收缩模式

在态矢量传播模式下，振幅的数量会随着量子比特数量的增加而呈指数级增长， 2^n 描述一个含有 n 个量子比特的系统的量子态需要复数 [20]。因此，基于经典态矢量的模拟器极难处理含有 50 个以上量子比特的 NISQ设备。TeD-Q 集成了张量网络收缩方法以及主流深度学习框架 PyTorch 和 JAX，提供了高效的内置基于张量网络的量子模拟器。在此模式下，可在单个中央处理器或图形处理器上轻松模拟某些含有 $50 \sim 100$ 个量子比特的量子电路。

TeD-Q 中张量收缩模式的工作流程包括以下步骤：(a) 根据量子线路的输出类型对其进行处理，并通过内置的 TensorNetwork 模块将其转换为张量网络；(b) 对张量网络进行结构简化；(c) 为简化后的张量网络搜索最佳收缩序列；(d) 对收缩进行切片，使其满足内存限制；(e) 根据切片后的收缩序列，使用后端库执行实际收缩。步骤 (a) 和 (b) 的计算复杂度很低，可由单个中央处理器线程完成。步骤 (b) 是可选的，但通常能显著缩小张量网络的规模。除默认的简单路径搜索器外，步骤 (c) 还可以选用两种基于超图划分的先进软件包——CoTenGra[21]& JDtensorPath。JDtensorPath 最初是 TeD-Q 的内置模块，后来被拆分为独立软件，以便更好地管理代码结构。它与 TeD-Q 的设计范式完全契合，将在第3.2.2节介绍。步骤 (d) 也可以通过 CoTenGra 或 JDtensorPath 库完成。这同样是可选步骤，但会增加总收缩浮点运算次数（ FLOPs）的额外开销。在 JDtensorPath 中，步骤 (c) 和 (d) 由多个中央处理器并行完成。步骤 (c) 或 (d) 中找到的路径可以在步骤 (e) 中重复使用，以获得相同量子线路在不同输入下的结果参数。步骤 (e) 耗时最长，会在机器学习训练循环中反复执行。步骤 (e) 可以在单个中央处理器、单个图形处理器或 GPU 集群上运行。

3.2.1 张量网络与简化

量子电路可以通过多种方式进行模拟，其中张量网络方法近年来越来越受欢迎。TeD-Q 可以用相应的图形化张量网络表示量子电路。量子态和任意量子门都可以表示为张量，而张量网络保留了它们的拓扑结构。因此，得到的图 $G = (V, E)$ 包含与张量关联的顶点，边则由其指标标记。张量的秩由与其相连的边数决定。由于单个量子比特处于二维希尔伯特空间中，也就是说，每个指标的取值为 {0, 1}，因此秩为 k 的张量需要 $O(2^k)$ 存储空间。为了计算量子电路 $U|\phi_{\text{in}}$ 输出中得到特定最终量子态 $|\phi_{\text{out}}$ 的概率，即 $\phi_{\text{in}}|U|\phi_{\text{out}}$，需要对张量网络进行收缩，也就是说，对顶点之间共享的指标求和，同时保留开放边。图 2 展示了一个简单的双量子比特量子电路及其在 TeD-Q 中的图形化张量网络表示。张量网络中的每个顶点都关联一个张量，用以表示量子态或一个量子门。 A_a 和 B_b 是对应两个量子比特量子态 $|0$的张量，而C_{ca} 和 D_{db} 分别表示单量子比特 Pauli$X$ 门和旋转 R_y 量子门；双量子比特 CNOT门对应于 E_{efcd}，沿 z 方向的测量则由 F_{gf}表示。由于期望值可由 $I \otimes Z = 00|U\dagger(\theta)I \otimes ZU(\theta)|00$ 得出，要用张量网络计算该值，我们必须附加与原量子电路复共轭相对应的张量网络，即 $00|U^\dagger(\theta)$。随后， $R_y(\theta)$门可以表示为 D_{db} ，其张量形式为:

$$D_{db} = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}; \qquad (2)$$

而 CNOT门的张量形式为:

$$E_{00cd} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, E_{11cd} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \\ E_{01cd} = E_{10cd} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{3}$$

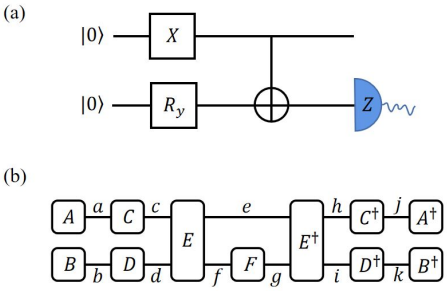


图 2：TeD-Q 中的量子电路及其图形化张量网络表示。(a) 一个双量子比特量子电路，在第二个量子比特上进行期望值测量。(b) 对应的图形化张量网络表示，每个顶点都关联一个由量子态或算子（量子门、量子态或可观测量）给出的张量。

因此，期望值可以通过以下求和计算:

$$T = \sum_{a,\ldots,k} A_a B_b C_{ca} D_{db} E_{efcd} F_{gf} E^*_{hieg} C^*_{jh} D^*_{ki} A^*_j B^*_k \\ = \sin^2(\theta/2) - \cos^2(\theta/2),$$

这可以通过传统的向量模式计算加以验证。结果表明，执行这些求和的实际顺序（例如，先对 a 还是 k 求和）影响很大，而寻找合适的收缩顺序构成了张量网络算法的大部分工作。

TeD-Q 支持在搜索收缩顺序之前，根据张量网络的结构及其张量的稀疏性，通过一系列化简对张量网络进行高效的局部处理。这些化简包括张量形状压缩、对角化简与反对角化简，以及秩化简，旨在降低张量网络的复杂度和张量的秩 [21]。经过局部预处理后，张量网络将从线图转换为超图（图的一种推广形式，允许一条边连接任意数量的顶点）。

张量网络及其超图表示如图 3 （a）所示。

请注意，红色超边（对应索引 c）与三个顶点（张量）相连。原始张量网络定义如下:

$$T_{de} := \sum_{a,c,b,f} A_{ac} B_{abf} C_{bce} D_{cd} E_{ef}, \tag{4}$$

其中，大写字母和小写字母分别表示张量及其指标。

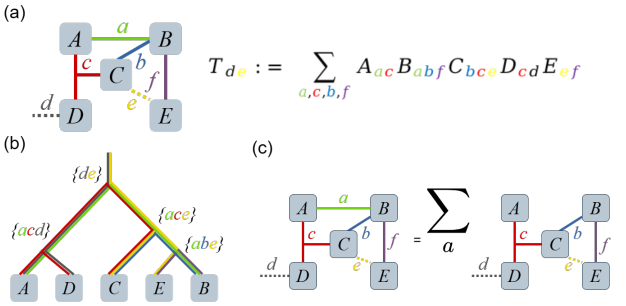


图 3：张量网络及其超图表示、二叉收缩树和指标切片示意图。(a) 张量网络及其超图表示示例，边 c 是一条连接三个顶点的超边。虚线表示开放边，实线表示闭合边。(b) 二叉收缩树，每个节点表示两个张量的成对收缩，以及其超图的二划分。向上的边表示中间张量的指标。(c) 对绿色边（指标 a）进行切片，生成结构相同的简单张量网络，并以 a 的所有可能取值标记。每次收缩都独立进行，将结果求和即可得到最终结果。

3.2.2 基于超图的张量收缩路径优化器

张量网络收缩的时间和内存复杂度取决于张量秩和收缩顺序。尽管对于一般情况，寻找最优收缩序列在理论上是一个非确定性多项式时间（NP）难题，但实践中存在高效的启发式算法 [22]，可获得“足够好”的序列。TeD-Q 支持多种来自第三方库的成熟算法，包括蛮力搜索、分支策略、贪心路径和随机贪心路径、动态规划方法、通过 QuickBB 或 FlowCutter 进行的线图树分解、社区检测、玻尔兹曼贪心凝聚收缩树，以及超图划分方法 [23, 21]。TeD-Q 默认的基础路径搜索器支持前四种方法。为了

获得更好的性能，我们采用了最初由 CoTenGra 提出的超图划分方法，并开发了自己的基于超图的路径搜索器库——JDtensorPath。

该算法的核心思想是递归地划分给定张量网络的超图，并按照图 3b 所示，以“自顶向下”的方式显式构建一棵不完整的二叉收缩树。树的根节点对应张量网络的初始完整图，以及最终结果的有效张量。对父节点的超图进行二划分，会生成两个表示有效中间张量的子节点。通过这种递归划分方式即可生成收缩树。收缩序列通过自底向上遍历该树获得；将两个子节点合并为一个父节点，对应于对两个中间张量进行一次成对收缩。

单次收缩（节点二划分）的浮点运算次数为：

$$\mathrm{FLOPs} = p * (2 * q - 1), \tag{5}$$

其中， p 是外部指标（未收缩的指标）维度的乘积， q 是收缩指标维度的乘积（若没有指标被收缩，则其值为 1）。对于每个节点的二划分，p 是固定的；因此，通过最小化划分切开的加权边（指标）之和，可以最小化相应成对收缩的开销。这种方法类似于一种自顶向下的贪心策略，因为它只考虑每次单独收缩的开销。此外，JDtensorPath 不仅要求当前节点的划分最优，还会考虑其子节点的划分，从而最小化整体树的大小和总收缩浮点运算次数。具体而言，要求父节点的外部指标尽可能均匀地分配到两个子节点上。此要求可以减少子节点划分中的 p ，进而影响其收缩所需的浮点运算次数。成对收缩的内存使用量等于所有相关张量大小之和。由于每次收缩的父节点张量都是固定的，将其指标均匀分配到子节点张量上也可以减少内存使用量。这是因为，假设 p 固定，并将 m、 n 分别定义为收缩过程中子节点张量剩余指标维度的乘积，

当给定$m * n = p$时，若 $m = n$，收缩 $m * q + n * q$的内存开销将达到最小。

采用先进的多层超图划分框架 KaHyPar [24, 25] 来构建收缩树，并使用现代并行分布式 Python 软件包 Ray 并行生成多个试验。JDtensorPath 可为 TeD-Q 提供高质量的收缩序列，并使张量网络收缩模式在各种量子电路模拟中实现数量级的加速。

3.2.3 切片

切片是一个可选步骤，旨在解决内存限制问题，并为并行计算提供一种高效的方法[26]。即使收缩路径十分出色，大型张量网络的收缩仍受其固有内存复杂度的限制。此外，张量之间错综复杂的数据依赖会成为使用多个计算单元进行收缩的瓶颈，因为这需要大量的处理器间通信。对指标进行切片，可以将原始张量网络拆分为多个结构相同（超图相同）的张量网络，每个网络的内存开销都足够小，可以装入单个处理单元。模拟器可以根据计算资源情况，并行执行这些子任务（任务之间互不依赖），也可以按顺序执行。原始张量网络的值等于对各个子任务进行收缩所得到的部分结果之和，其中选定切片指标的赋值是固定的[27]。以图3为例，切片后，原始张量网络可以进一步表示为：

$$T_{de} := \sum_{i=a} (\sum_{c,b,f} A_c^{(i)} B_{bf}^{(i)} C_{bce} D_{cd} E_{ef}). \tag{6}$$

找到要切片的最优指标子集同样是一项NP难任务，其复杂度为 $O(m^n)$，其中 m 和 n 分别表示所有指标和被切片指标的数量。这里采用的策略是使用如下所述的启发式贪心算法：

(a) 找出所有尺寸大于目标大小的中间张量；

(b) 随机选取一个由最大张量共享的索引，或在步骤 (a) 的张量中出现频率最高的索引；

(c) 重复上述步骤，直到收缩宽度足够低。

寻找切片指标与搜索最佳收缩路径密切交织在一起。切片基于已有的收缩路径，但原始张量网络的最优路径不一定适合切片。因此，JDtensorPath 提出了一种分为两个阶段、将指标切片纳入其中的收缩路径搜索启发式算法。第一阶段生成收缩树试验，随后进行 128 次伪切片试验；这些试验只计算浮点运算次数，不会实际对收缩树进行指标切片。第二阶段则在伪切片后浮点运算次数最低的收缩树上进行“真实”切片试验（默认进行 1024 次）。这样，模拟器便能在合理的搜索时间内，为子任务找到“足够好”的切片指标和收缩路径。切片启发式算法基于集合运算实现，且模拟器只会存储经过最佳切片的收缩树。因此，该算法效率很高且节省内存。此外，这个切片子程序还可以借助 Ray 框架，在中央处理器计算过程中并行执行。

3.2.4 用户 API

TeD-Q 支持用户自定义张量网络收缩过程的控制方式。代码清单4展示了一个使用 JDtensorPath 以张量网络收缩模式编译量子电路的示例。

```
def circuitDef(parameters):

#

par

con

slicing_opts = {'target_size':2**28,

)

(
```

```
                hyper_opt = hyper_opts,
                tn_simplify = False)
```

代码清单 4：如何使用张量网络收缩模式。

要启用张量收缩模式，用户需要指定切片和基于超图的优化器选项，分别如“$slicing_opts$”和“$hyper_opts$”所示，并将这些选项以及一个收缩路径优化器（如“$jdopttn$”所示）传入电路编译器。编译后的电路将以张量收缩模式构建。每个选项和参数的详细说明可在 GitHub 仓库以及 TeD-Q 文档中找到。

4 变分量子电路

优化是大多数 NISQ算法的重要共性因素，包括量子机器学习和量子化学模拟。它依赖于针对参数化量子电路（即所谓的变分量子电路（VQC））进行的量子—经典混合优化过程。例如，量子神经网络（QNN）可视为一种 VQC。在量子机器学习任务中，VQC 可作为一个组件，与特征提取和预测等预处理及后处理步骤协同工作。通常，任务的代价函数可取为 VQC 的期望值：

$$f(x;\theta) = \langle \hat{O} \rangle = \langle 0|U^{\dagger}(x;\theta)\hat{O}U(x;\theta)|0\rangle. \quad (7)$$

优化过程依赖于量子线路输出相对于量子门参数的梯度。参数移位法和有限差分法是获取量子电路梯度的常用方法，适用于量子硬件和模拟，也是 TeD-Q 内置的方法。然而，由于计算复杂度较高，采用这些方法进行模拟通常耗时很长。与此同时，随着大规模量子设备近期不断发展，快速模拟含有大量量子比特的线路已变得迫在眉睫。为此，TeD-Q 利用传统的反向传播技术和其新开发的张量网络引擎，提供了一种在模拟中高效计算电路梯度的方法。

4.1 反向传播方法

TeD-Q 中的反向传播方法基于成熟的自动微分库，例如 PyTorch[28] 和 JAX[29]，这些库已广泛应用于经典机器学习。这些库作为模拟后端集成到 TeD-Q 中。指定后端后，TeD-Q 会将量子电路的矩阵数据转换为相应格式。在评估量子电路（即“前向”计算）时，后端会存储所有中间结果。后端将利用链式法则，根据电路输出计算门参数的梯度，这一过程称为反向传播。该功能会存储每个中间结果的梯度，从而一次性计算出所有参数的梯度。尽管反向传播方法无法应用于真实量子设备，但它是在量子仿真中获取梯度最快的方法。因此，它是模拟后端的默认方法。

```
def circuitDef(parameters):
    # Circuit definition
# parse and compile the circuit
cir = tedq.circuit(circuitDef, n_qubits,
    parameters)
myCompileCir = cir.compilecircuit(
                    backend="pytorch")
```

代码清单 5：如何使用反向传播方法获取参数的梯度。

4.2 参数移位法与有限差分法

在真实的量子设备中，变量的梯度只能通过参数移位法或有限差分法获得。参数移位法可以提供解析导数，而有限差分法只能提供数值导数。

在 TeD-Q 中，我们基于理论论文 [30, 31] 以及 [4]的最新进展实现了参数移位法。该方法适用于形式为 $e^{i\mu H}$ 的量子门，其中 H 是具有两个或四个特征值的厄米算符， μ 是量子门的参数。对于单个门参数，可以通过将参数值分别设为 $\mu + s$ 和 $\mu - s$[30]，对量子线路的期望值进行两次计算，从而求得解析导数，如公式 8所示，其中 s 是偏移量， k 是偏移常数。TeD-Q 的后端会自动为每个量子门以适当的间隔偏移参数。对于 PyTorch 或 JAX 等仿真后端，用户可以在编译量子线路时于微分选项中指定参数移位法，以此计算梯度，如代码清单 6所示。对于硬件后端，参数移位法是计算梯度的默认且唯一方法。

$$\frac{\partial f}{\partial \mu} = \begin{cases} k[f(\mu+s) - f(\mu-s)] & \text{parameter shift} \\ \frac{f(\mu+\Delta\mu)-f(\mu)}{\Delta\mu} & \text{finite differential} \end{cases} \tag{8}$$

```
def circuitDef(parameters):
    # Circuit definition
# parse and compile the circuit
cir = tedq.circuit(circuitDef, n_qubits,
    parameters)
compileCir = cir.compilecircuit(
                    backend="pytorch",
                    diff_method = "
    param_shift")
```

代码清单 6：如何使用参数移位法获取参数的梯度。

4.3 Quantum 到经典计算接口

TeD-Q 的设计范式实现了量子处理与经典处理的无缝集成。在这种混合算法中，由可调变量参数化的量子电路被视为具有特定接口的普通 Python 函数。它可以插入经典优化函数中，也可以封装在机器学习框架模块的一层中。

TeD-Q 支持两种接口：JAX 数组和 PyTorch 张量。对于 JAX 仿真后端，JAX 数组是默认接口，但也可以在编译量子电路时通过 interface 关键字参数（interface="pytorch"）使用 PyTorch 张量接口。至于 PyTorch 仿真后端和硬件后端，本文撰写时 TeD-Q 仅支持 PyTorch 后端。这些接口能够在“前向”计算和“反向”梯度提取过程中，与经典部分无缝衔接。对于具有自有接口和默认反向传播梯度方法的仿真后端，集成十分简单，只需依赖 JAX 或 PyTorch 内置的数据结构和自动微分算法。对于其他情况（JAX 后端搭配

PyTorch 张量接口、硬件后端或参数移位方法），经典反向传播算法无法处理量子电路内部的量子信息。在这种情况下，量子电路被视为黑箱。TeD-Q 实现了自定义 Autograd 函数，可计算并提供量子电路相对于其经典输入和变量的梯度（单个输出）或雅可比矩阵（多个输入和多个输出）。代码清单 7 展示了一个示例，说明如何通过 PyTorch 张量接口将 JAX 后端量子电路连接到经典计算。更多详细示例请参见第 5.1.3 节。通过 JAX 或 PyTorch，用户几乎感受不到集成过程，同时可以灵活地在中央处理器、图形处理器和张量处理器上运行量子—经典混合算法代码。

```
# Quantum circuit construction
    # Neglect here

# Compile quantum circuit with JAX quantum
    simulation backend and PyTorch interface
compileCirJAX = circuit.compilecircuit(
                        backend="jax",
                        interface="pytorch")

# Connect quantum systems to classical cost
    function by PyTorch interface
def cost(weight, params):
    return weight * compileCirJAX(params)

# PyTorch tensor is used as input
weight = torch.tensor(0.5)
params = torch.tensor([0.12])
cost(weight, params)
```

代码清单 7：量子到经典计算接口。量子线路通过 JAX 仿真后端和 PyTorch 张量接口编译为可执行的量子对象。随后，可执行的量子电路与经典代价函数集成。

5 用例

用户界面的详细介绍和手册可在在线文档中查阅。文档包含循序渐进的示例，展示 TeD-Q 的功能。在接下来的部分中，我们将简要介绍其主要功能及用例。

5.1 基本量子线路

量子电路评估是 TeD-Q 的基本功能。它提供了一套通用的量子门，使各种量子电路都可以构建并模拟。借助第3.2节介绍的张量网络引擎，TeD-Q 的模拟速度比传统量子模拟器更快。此外，TeD-Q 配备了易于使用的图形化电路编辑器和电路绘制器。这些工具与 TeD-Q 的模拟器紧密集成。用户可以用它们快速构建量子电路原型。此外，TeD-Q 的硬件后端提供了连接 Qiskit 等真实量子设备的接口[5]。用户只需执行一条简单命令，即可将模拟切换为真实实验。以下各节将通过 TeD-Q API 展示这些功能。

5.1.1 快速模拟

考虑一个简单的双量子比特电路，其中量子比特 $Hadamard$ 0 上有一个量子门 0，量子比特 $CNOT$ 0 和 1上有一个量子门，并在量子比特 Rot1 上施加一个旋转门，作用于初始零态。我们希望根据输入的旋转角度得到其量子态。TeD-Q 可以通过代码块 8实现模拟。量子比特从 0 开始计数，多量子比特寄存器按第一个（第零个）量子比特在左侧的方式标记。只有仿真后端支持测量量子态，而仿真后端和硬件后端都支持期望值和概率测量。关键字参数“requires_grad”可以禁用梯度功能并提升模拟速度，因为后端无需存储用于反向传播的中间态。

```
import tedq import torch n_qubits=2
#
c
phi
py
```

```
quantum_state = compileCir()
```

代码清单 8：一个简单双量子比特电路示例。*Hadamard* 和 *CNOT* 量子门用于生成纠缠态，*Rot* 量子门用于添加旋转角度，最终的量子态会在末尾进行测量。

5.1.2 直观原型设计

TeD-Q 中的电路编排器是用于量子线路的所见即所得（所见即所得）编辑器。图形化编排器可通过一行命令初始化，如代码清单 9 所示。它由一个用于放置模板量子门的面板和若干条表示量子比特的线路组成，如图 4 所示。用户可以将量子门图标拖动到特定的量子比特线路上来构建量子线路，也可以将量子门从量子比特线路上拖走以将其移除。用户还可以将图标拖动到所需位置，以调整操作的顺序。

```
# Enable circuit composer
composer = tedq.circuit_composer(4, figsize
    =(11,5))
```

代码清单 9：电路编排器的用法

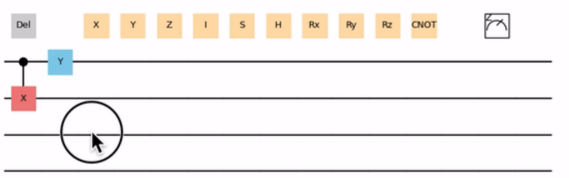


图 4：电路编排器

编排结果可以转换后输入 TeD-Q 的张量网络引擎。用户可以使用上一节中提到的相同方法获取模拟结果，如代码清单 10 所示。用户还可以从编排结果中获取电路定义，以便使用 TeD-Q API 进行进一步开发。

```
# Convert to circuit class
```

代码清单 10：将编排结果转换为 TeD-Q 量子线路并获取电路定义

除了电路编排器，脚本初始化的量子线路也可以通过电路绘制器进行可视化，如代码清单 11 所示。它采用与电路编排器相同的绘制风格。此外，该函数会自动将量子线路调整至绘图区域的合适大小。

```
# Init a circuit
circuit = tedq.circuit(circuitDef())
# Draw a circuit
tedq.draw_circuit(circuit)
```

代码清单 11：电路绘制器的用法

5.1.3 硬件兼容性

除了模拟之外，该量子线路还可以通过硬件后端应用于真实量子设备。截至撰写本文时，TeD-Q 仅支持 IBM-Q 量子设备，未来将支持更多量子硬件设备。内置的参数移位法用于获取量子电路的梯度，并提供 PyTorch 张量接口，以便与其他函数进行通信。代码清单 12 展示了一个示例：将量子电路编译为两个可执行的量子对象，分别使用模拟和硬件后端，并尝试通过内置的梯度下降优化器调整输入参数，以尽量减小二者之间的差异。

```
# Quantum circuit construction
backend="IBMQ hardware")
# Tuning variables with built-in optimizer
True))
```

```
new_variables = Optimizer.step(*params)
```

单 12：硬件兼容性示例。量子电路通过 JAX 仿真后端和硬件后端编译为两个可执行的量子对象。这两个可执行的量子电路通过 PyTorch接口连接到经典代价函数。

```
loss = torch.nn.MSELoss(reduction='mean')
```

代码清单 13：变分量子算法示例。

5.2 变分量子算法

变分量子算法由变分量子电路和经典优化器组成。第4节介绍了如何求得量子电路中门参数的梯度。通过递归更新量子门参数，可以根据特定的代价函数优化电路。该算法广泛应用于当前的 NISQ 设备和量子仿真中。借助 TeD-Q 的张量引擎，其仿真速度远高于其他仿真软件包。此外，借助硬件后端，变分算法也可以适配真实量子设备。下面的示例展示了一个基于变分量子算法的简单电路，说明如何使用 TeD-Q API 构建 VQA 电路。

```
import tedq
import torch
n_qubits=3
#
qu
to
my_compilecircuit = circuit.compilecircuit(
ba
True)
va
=0.
```

5.3 量子机器学习

机器学习研究如何从有限数据中实现高维希尔伯特空间中的泛化。它是量子计算一个非常引人注目的应用，因为二者在根本层面都涉及统计理论。用户可以通过编码方案（基态编码、振幅编码、角度编码等）将经典特征信息转换为量子态：

$$|\psi(x)\rangle = \begin{cases} |i_x\rangle & \text{basis encoding} \\ \sum_i^n x_i |i\rangle & \text{amplitude encoding} \\ \otimes_i^n R(x_i)|0^n\rangle & \text{angle encoding} \end{cases} \tag{9}$$

VQA 可以设计为一种线性模型，在希尔伯特空间中显式地对数据进行分类。得益于 TeD-Q 张量引擎的出色性能，可以构建并模拟更复杂的 VQA 电路，从而使用模拟器测试大规模量子及量子-经典混合机器学习算法。

5.3.1 模板

TeD-Q 提供了若干预定义的量子线路模板或试探态，这些模板或试探态在量子机器学习领域中广泛使用，可加快量子机器学习线路的开发。下面简要介绍 TeD-Q 中的三种模板：随机生成层、全连接层和硬件高效层。

随机生成层可以为量子线路提供单量子比特和双量子比特量子门，且它们的位置和参数都是随机的（图5）。它适用于模拟非线性

变换，如 TeD-Q 的 Quanvolu- tionNeural Network 示例所示。

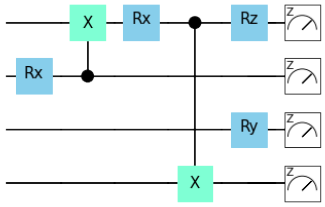


图5：随机生成的量子线路

全连接层是一种参数化量子线路，用于在各个量子比特之间提供最大程度的纠缠。层数称为深度，用户可以对其进行自定义。在每一层中，每个量子比特都通过 CNOT门与其他量子比特连接，随后接一个双参数旋转门（图6）。此模板通常用于在量子机器学习中模拟经典的全连接层。

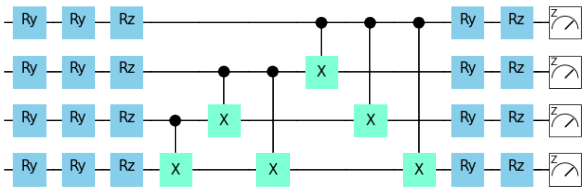


图6：全连接量子线路

硬件高效层是全连接量子线路的一种实用实现。在真实量子设备中，双量子比特操作只有在相邻量子比特之间执行时才高效。因此，此模板中的 CNOT门仅设置在选定量子比特与其相邻量子比特之间（图7）。

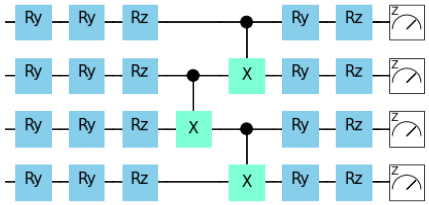


图7：硬件高效量子线路

5.3.2 量子层封装

用户可以通过将成熟的经典机器学习模型中的部分层替换为封装的量子层。代码清单 14 演示了如何在 TeD-Q 中构建流水线。

```
# Global variable
n_qubits = 8       # Number of qubits
q_depth = 12       # Number of variational
    layers
n_classes = 10     # Number of image classes

# Angle encoding
def features_encoding(nqubits, w):
    for idx in range(nqubits):
        tedq.Hadamard(qubits=[idx])
    for idx, element in enumerate(w):
        tedq.RX(element, qubits=[idx])

# Define a variational quantum circuit
def circuitDef(q_input_features,
    q_weights_flat):
    # Reshape weights
    q_weights = q_weights_flat.reshape((
    q_depth+1)*2, n_qubits)

    # Encode classical features into quantum
    state
    features_encoding(n_qubits,
    q_input_features)

    # Built-in template circuit
    tedq.templates.HardwareEfficient(n_qubits
      q_depth, q_weights)

    # Expectation values measurement
    exp_vals = [tedq.expval(tedq.PauliZ(
    qubits=[position])) for position in range
    (n_qubits)]
    return tuple(exp_vals)

# Quantum circuit construction
circuit = tedq.Circuit(circuitDef, n_qubits,
```

```
# Quantum layer with PyTorch NN interface
class DressedQuantumNet(nn.Module):
    def __init__(self):
        self.post_net = nn.Linear(2048,n_qubits)

    # Tensors moves through quantum circuit
    def forward(self, input_features ):

        return self.post_net(q_out)
```

```
# Replacing classical machine learning model
    ResNet-50 fully-connected layer with
    quantum layer
model_hybrid = torchvision.models.resnet50()
model_hybrid.fc = DressedQuantumNet()
```

代码清单 14：用于创建图像识别混合量子-经典机器学习模型的代码片段。量子电路封装在量子层中，用于替换经典深度学习模型 ResNet-50 的全连接层。本示例的主要结构参考了 PennyLane 关于量子迁移学习的教程[4]

上述示例展示了构建混合量子-经典机器学习模型的一种方法。TeD-Q 可以轻松地将量子算法与各种成熟的机器学习实践相结合。量子电路模型的表现就如同纯经典模型。代价函数或优化等后续流程都可以无缝采用。下一节将介绍一个纯量子机器学习示例，并将其用作基准，以比较 TeD-Q 与其他软件的性能。

6 性能

我们通过两个示例——多量子比特旋转（MQR）和多体局域化（MBL）[32]，将 TeD-Q 的性能与其他量子软件进行基准测试。MQR 是一种非常简单的算法，它使用硬件高效网络将量子态从全零态 $|0\rangle^{\otimes n}$翻转为全一态 $|1\rangle^{\otimes n}$。它适用于评估 TeD-Q 在个人计算机上运行小型、简单量子线路时的性能。MBL 是一个相对复杂的问题，在理论和实践方面都具有重要意义。该示例用于展示 TeD-Q 张量网络收缩模式的显著优势。比较所用的源代码可在 TeD-Q 的 GitHub 上找到。

6.1 多量子比特旋转

在此模拟中，初始量子态设为全零态 $|0\rangle^{\otimes n}$，随后采用变分硬件高效网络，并对每个量子比特在泡利 Z 基上进行期望值测量。通过调整参数并使用梯度下降法，最小化测量值之和来完成学习。平台性能通过完成 500 轮训练所需的时间来评估。由于存在用户迫切希望能在个人计算机上通过简单的量子线路开发和学习量子算法，因此评估采用单颗 AMD Ryzen^{TM} 5 3500U 移动处理器中央处理器。图8 展示了 TeD-Q 在此应用中的优势。

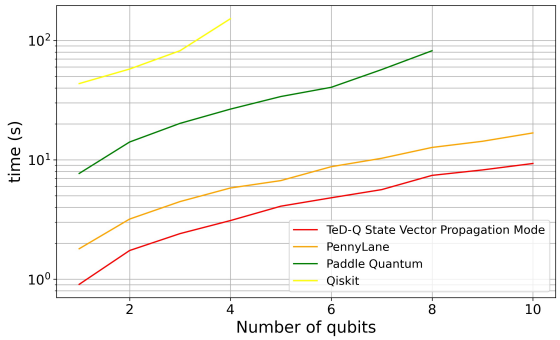


图 8：TeD-Q、Pen- nyLane、Paddle Quantum 和 Qiskit 在单台笔记本电脑中央处理器上运行小型简单量子线路的性能比较。TeD-Q 的结果使用 PyTorch 后端和反向传播梯度法。

6.2 氢分子的 VQE

变分量子本征求解器（VQE）是 NISQ 时代量子计算的一种热门应用 [15]。它是用于解决量子化学问题，尤其是本征态问题的变分量子算法（VQA）的一个具体实例。在 VQE 算法中，参数化量子电路会制备一个量子态，称为拟设态，该量子态应为分子的哈密顿量的基态。通过在不同测量设置下反复测量拟设态，可以计算出哈密顿量的期望值。随后可对量子电路的参数进行优化，以使期望值最小化。

这里我们通过氢分子的 VQE 来展示 TeD-Q 与其他平台的性能对比。在此实验中，共有 4 个量子比特，量子电路包含 12 个可自由变化的参数。（原则上，该问题可以简化为只使用一个量子比特，但我们没有这样做，因为我们只想在相同条件下比较速度。）

我们在配备 Intel Core^{TM} i5-1030NG7 中央处理器的 MacBook Air 上运行实验。如我们所见

从图 9可以看出，TeD-Q 收敛到误差为 10^{-6}哈特里的速度约为 PennyLane 的 3 倍、Qiskit 的 15 倍。在执行时间有限的情况下，例如图 10a 所示，在 5 秒内，TeD-Q 能够正确绘制不同氢氧距离下的基态能量曲线，而 PennyLane 几乎可以做到，Qiskit 则相去甚远。当时间限制设为 30 秒时，PennyLane 可以收敛到正确的能量曲线，但 Qiskit 仍然无法做到，如图 10b所示。通过这一比较可以看出，与所比较的平台相比，TeD-Q 在模拟传统 VQE 算法方面确实具有优势。

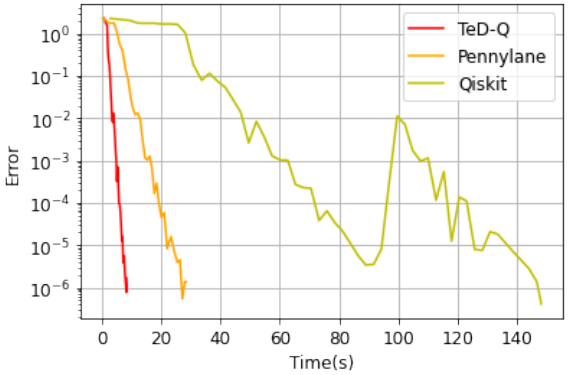
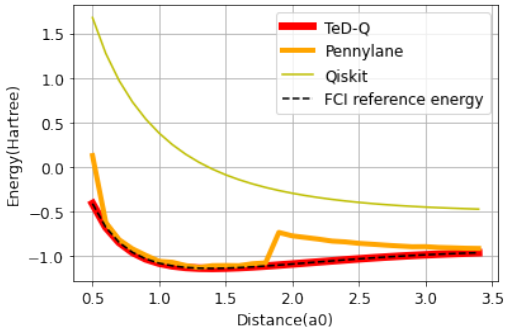
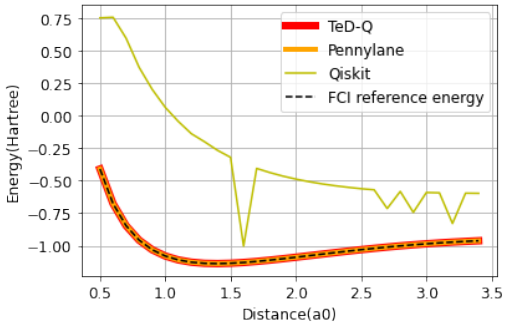


图9：VQE 结果与全组态相互作用参考能量之间的误差。



（a）执行时间限制为 5 秒。



（b）执行时间限制为 30 秒。

图 10：在有限执行时间下，采用 VQE 和全组态相互作用方法得到的氢原子能量曲线。

6.3 量子生成对抗网络

近年来，经典生成对抗网络（GAN）凭借迅速增长的应用，在人工智能领域占据了一席之地 [33]。其思路是利用生成网络创建与原始数据相似的新数据，同时由判别网络评估二者之间的距离。两个网络相互竞争，进行零和博弈，从而提升新生成数据的质量。

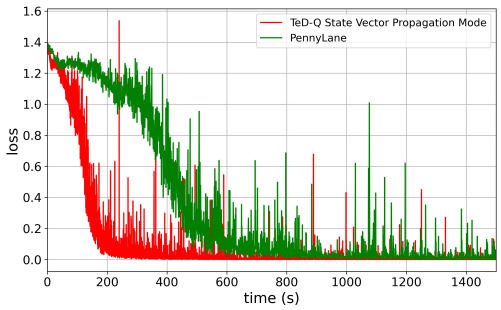
Quantum 生成对抗网络借鉴了这一思路，并希望借助量子计算提升性能 [34, 35]。虽然生成网络和判别网络都可以使用量子神经网络（QNN），但此处我们采用分块和非分块量子生成对抗网络方法，仅将生成部分量子化。

在此示例中，生成器尝试模仿一小组尺寸为 8×8 的灰度样本图像中的手写数字零。有关量子生成对抗网络示例的详细信息，请参阅 [35] 。类似于多量子比特旋转示例，我们使用 AMD Ryzen^{TM} 5 3500U 移动处理器中央处理器运行实验。从结果图11中可以看出，无论是在分块还是非分块场景下，TeD-Q 在量子生成对抗网络模型的训练中都优于 PennyLane。

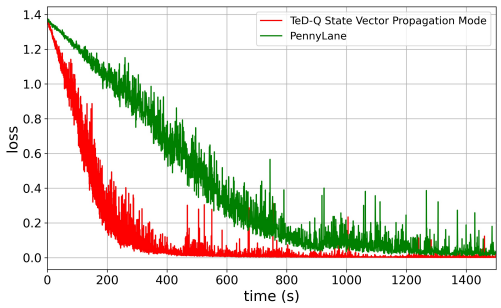
6.4 量子架构搜索

近期提出了量子架构搜索（QAS），用于针对给定的 VQA 任务生成更优的变分试探电路 [36, 37]。QAS 通过构建试探态池，在 VQA 优化的每一步迭代采样一个新的试探态。研究表明，这种做法能够提升 VQA 的性能，体现在电路输出和学习任务的可学习性两方面。

在量子架构搜索任务的训练过程中，每一步优化都会从候选池中选取新的样本来替换试探态量子线路，因此后端需要在每一步都编译新的试探态量子线路。因此，性能差异主要取决于执行每个量子线路的速度。我们在搭载 Intel Core ^{TM} i5-1030NG7 中央处理器的 MacBook Air 上进行了实验。图 12 展示了



(a) 量子生成对抗网络补丁



(b) 量子生成对抗网络非补丁

图 11：在单台笔记本电脑中央处理器上进行量子生成对抗网络训练时，TeD-Q 与 PennyLane 的性能对比。TeD-Q 的结果使用 PyTorch 后端和反向传播梯度方法。

训练所需的时间随迭代次数增加而变化。在执行量子架构搜索任务时，TeD-Q 的速度比 PennyLane 快两倍以上。

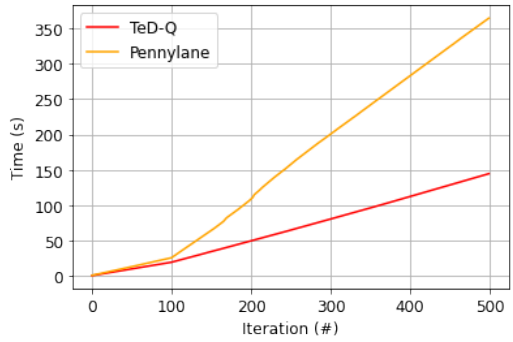


图 12：TeD-Q 与 Pennylane 的量子架构搜索性能对比

6.5 多体局域化

多体局域化（MBL）是孤立多体量子系统的一种相，可阻止系统发生热化；也就是说，处于这一相的系统能够在时间演化过程中保留初始信息。它提供了一种长期保持量子效应的可能性。近年来，随着量子技术的迅速发展，该领域的实验和理论研究取得了显著进展。本节将展示，TeD-Q 能以合理的速度处理一维多体局域化的模拟，以及包含 50 多个量子比特的量子神经网络；传统量子模拟器无法做到这一点。

对于一个孤立多体量子系统，它在其内禀哈密顿量的作用下演化。粒子之间的相互作用会交换信息和能量，使得整个封闭系统达到热平衡，因为整个系统是其子系统的热浴[38]。其初始量子态的信息最终会被抹去。然而，在存在足够强的无序势时，系统可以进入多体局域化相，从而避免落入热态，即使系统中的粒子仍然相互作用。

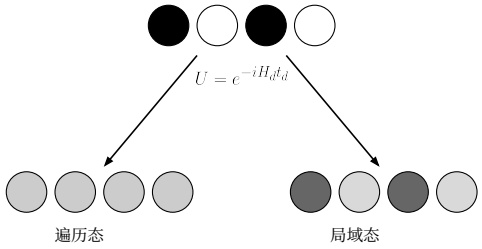


图 13：多体局域化。圆圈的阴影表示处于 $|1\rangle$ 量子态的概率。系统最初被制备在奈尔态。在内禀哈密顿量和可控无序势的作用下演化后，系统可能进入遍历态（如左下图所示），也可能进入局域态（如右下图所示）。

当无序势较小时，系统会发生热化，此时称为遍历态。另一方面，当无序强度足够大时，系统会进入 MBL 相并保持在初始态，此时称为局域态。然而，对于多体系统，需要进行多次测量才能区分局域态和遍历态。基于 [39], 所示的量子线路架构，我们设计并训练了一个量子神经网络，用于对一维多体系统的量子态进行分类。借助该量子神经网络电路，只需对第一个量子比特进行一次测量，我们就能判断系统处于遍历态还是局域态。

6.5.1 模拟设置

在我们的模拟中，量子比特沿链排列，并制备为奈尔态。随后施加模拟的哈密顿量 H_d，其中包含最近邻量子比特之间的相互作用，以及对每个量子比特可控的随机无序 d_i，其

$$H_d = \hbar \sum_{i,j} d_i \sigma_z^i + g_{i,j}(\sigma_x^i \sigma_x^j + \sigma_y^i \sigma_y^j)/2, \quad (10)$$

并且系统在此哈密顿量下演化一段时间 t_d。对于量子态分类，随后将该线路连接到一个由数字-模拟量子神经网络构建的分类器，该网络由 [39]提出。它由 $N_q + 1$ 个单量子比特操作 $R(\theta_i, \phi_i) = Z(\theta_i)X(\phi_i)Z(-\theta_i)$组成，这部分是数字部分；此外，每个量子比特上还施加一个幺正操作 H_0，作为模拟部分。

$$H_0 = g_{i,j}(\sigma_x^i \sigma_x^j + \sigma_y^i \sigma_y^j)/2, \quad (11)$$

最后，只在其中一个量子比特上进行测量。训练量子神经网络的参数，使其测量结果能够对系统的量子态进行分类。电路图如图 14所示。

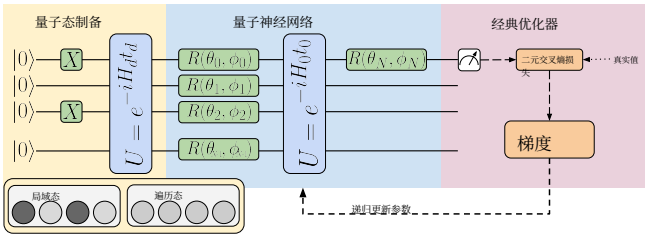
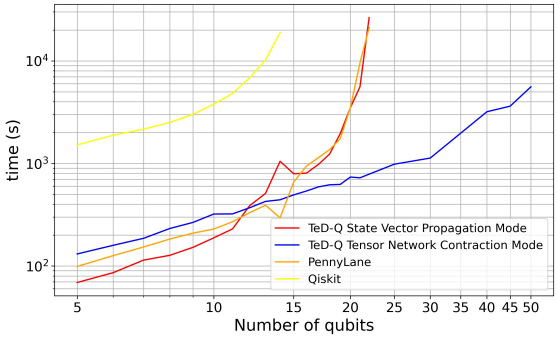
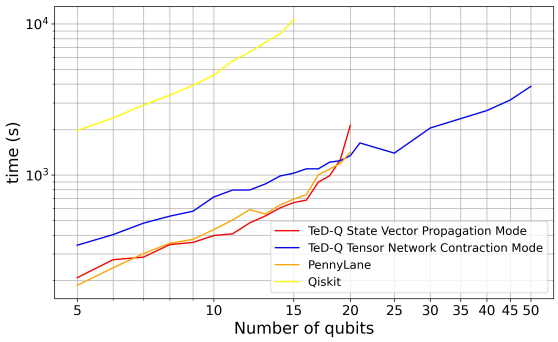


图 14：多体局域化模拟中使用的量子电路和经典优化器。该图可分为三个阶段——量子态制备、量子神经网络（QNN）和经典优化器。量子态制备阶段生成遍历态或局域态。QNN 阶段经过训练，用于区分遍历态和局域态。优化器阶段评估结果，并根据损失函数优化 QNN。

模拟参数，例如$g_{i,\ j}$和 t_d，遵循 [39]，中使用的设置，这些设置是64量子比特量子计算机的属性。随后使用二元交叉熵损失训练分类器中的参数，使系统处于遍历态时所选量子比特的期望值为1，而系统处于局域态时为0。



(a) 单中央处理器模式



(b) 单图形处理器模式

图 15：TeD-Q、PennyLane 和 Qiskit 在 MBL 任务上的性能比较。

6.5.2 速度比较

通过完成一维多体局域化训练50个周期所需的时间来评估TeD-Q的性能。电路深度约为89，每个周期包含40次量子线路训练迭代。为进行比较，我们还使用Qiskit和PennyLane搭建了相同的模拟。中央处理器模拟使用Intel(R) Xeon(R)Processor E5-2680 v4 @ 2.4GHz中央处理器，图形处理器模拟则使用Nvidia Tesla P40 24 GB图形处理器。

图15 比较了这三种量子模拟方法在中央处理器和图形处理器上的表现。TeD-Q 在使用态矢量传播模式时，表现与 PennyLane 相近。当模拟涉及超过 20 个量子比特时，这种方法在计算上变得不可行。不过，TeD-Q 使用张量网络收缩模式时，能够在相近的时间内处理多达 50 个量子比特的同一模拟。

6.6 讨论

总体而言，TeD-Q 和 PennyLane 的态矢量传播模式在中央处理器和图形处理器上的性能相近。这符合我们的预期，因为两者都会计算完整的振幅，并使用反向传播方法求得梯度。对于少于 11 个量子比特的情况，TeD-Q 在中央处理器上的表现略优于 PennyLane（20 ∼ 30%）；这可以用可复用机制来解释，该机制节省了量子态和量子门类实例化所需的时间。

张量网络收缩模式在大型量子比特系统中展现出强大优势。它可以大幅降低计算复杂度和内存复杂度。但目前，其性能受到图形处理器利用效率低下的限制。收缩路径中的大多数成对收缩都涉及小张量，它们在整体计算复杂度和内存复杂度中所占比例很小。但将这些小张量传输到图形处理器会产生相当大的开销。通过在中央处理器上预先计算小规模收缩，并仅在图形处理器上执行大规模收缩，可以缓解这一开销问题[26], 这一改进将在 TeD-Q 的未来版本中实现。

7 总结

在这项工作中，我们介绍了 TeD-Q，这是一个用于量子机器学习、变分量子算法和量子计算模拟的开源软件框架。借助张量收缩，可以模拟包含大量量子比特的量子线路。TeD-Q 将经典机器学习库与量子模拟器无缝集成，使用户能够在训练量子机器学习模型时发挥经典机器学习的优势。TeD-Q 还支持以分布式方式训练量子机器学习模型。鉴于大多数含噪声中等规模量子算法都需要获取同一量子线路多个实例的结果，这是一项重要功能。在 TeD-Q 中，量子线路被视为一个 Python 函数，因此易于使用。TeD-Q 与设备无关，用户可以在不同后端运行线路，包括软件模拟器（JAX 和 PyTorch）以及量子硬件。

为了更好地促进量子机器学习的实现，TeD-Q 支持自动微分，例如反向传播、参数移位和有限差分法，以获得梯度。它还提供了灵活的接口，可将量子电路连接到功能强大的机器学习库（如 PyTorch）。此外，TeD-Q 还支持可视化功能，用户可以实时查看量子电路和训练进度。

TeD-Q 未来的工作仍面临挑战。一个关键方面是如何利用分布式计算的能力来解决实际的量子机器学习问题。尽管 TeD-Q 支持分布式计算，但目前仍缺乏能够充分发挥其优势的算法和应用。TeD-Q 还可以进一步改进，以便能够与云端不同类型的量子硬件完全兼容。我们已在第 5.1.3 节展示了 TeD-Q 如何访问 IBM 云上的硬件资源，并将在后续版本中加入对其他平台（如 PennyLane 和 Amazon）的支持。原则上，TeD-Q 已准备好连接任何设备；然而，由于缺乏量子硬件资源，这一点尚未经过测试。未来若能看到 TeD-Q 与不同的硬件后端协同工作，并实现真正的分布式量子算法，尤其是量子机器学习算法，将令人振奋。

致谢

感谢刘菊华的宝贵讨论。

作者贡献。 X.-Y. W.、Y.-X. D. 和 D.-C. T. 提出了该项目。Y.-C. C. 和 X.- Y. W. 设计了软件架构。Y.-C. C.、X.-Y. W. 和 C.-Y. K. 负责编码并开展数值模拟。X.- Y. W.、Y.-X. D.、Y.-C. C.、C.-Y. K. 和 D.-C. T. 分析了结果。所有作者均参与了对结果的讨论以及稿件的编写。Y.-C. C.、C.-Y. K. 和 X.- Y. W. 在所有合著者的意见基础上撰写了稿件。X.-Y. W.、Y.-X. D. 和 D.-C. T. 负责监督整个项目。

参考文献

[1]Frank Arute、Kunal Arya、Ryan Babbush、Dave Bacon、Joseph C Bardin、Rami Barends,

Rupak Biswas、Sergio Boixo、Fernando GSL Brandao、David A Buell 等人。《使用可编程超导处理器实现量子霸权》。《自然》574，505–510（2019）。[2]Lars S Madsen、Fabian Laudenbach、Mohsen Falamarzi Askarani、Fabien Rortais、Trevor Vincent、Jacob FF Bulmer、Filippo M Miatto、Leonhard Neuhaus、Lukas G Helt、Matthew J Collins 等人。《使用可编程光子处理器实现量子计算优势》。《自然》606，75–81（2022）。[3] Yulin Wu、Wan-Su Bao、Sirui Cao、Fusheng Chen、Ming-Cheng Chen、Xiawei Chen、Tung-Hsun Chung、Hui Deng、Yajie Du、Dao-jin Fan 等人。《使用超导量子处理器实现强量子计算优势》。《物理评论快报》127，180501（2021）。[4]Ville Bergholm、Josh Izaac、Maria Schuld、Christian Gogolin、M Sohaib Alam、Shahnawaz Ahmed、Juan Miguel Arrazola、Carsten Blank、Alain Delgado、Soran Jahangiri 等人。《PennyLane：量子-经典混合计算的自动微分》（2018）。arXiv:2110.03402。[5]A tA v、MD SAJID ANIS、Abby-Mitchell、Héctor Abraham、AduOffei、Rochisha Agarwal、Gabriele Agliardi、Merav Aharoni、Vishnu Ajith、Ismail Yunus Akhalwaya、Gadi Aleksandrowicz、Thomas Alexander、Matthew Amy 等人。《Qiskit：用于量子计算的开源框架》。 https://github.com/Qiskit/qiskit （2021）。[6] Xiu-Zhe Luo、Jin-Guo Liu、Pan Zhang 和 Lei Wang。《Yao.jl：用于量子算法设计的可扩展、高效框架》。《Quantum》4，341（2020）。[7] 《Cirq：用于创建、编辑和调用含噪声中等规模量子（NISQ）量子线路的 Python 框架》。 https://github.com/quantumlib/Cirq （2022）。[8] Jacob Biamonte、Peter Wittek、Nicola Pancotti、Patrick Rebentrost、Nathan Wiebe 和 Seth Lloyd。《量子机器学习》。《自然》549，195–202（2017）。[9] Maria Schuld、Ilya Sinayskiy 和 Francesco Petruccione。《量子机器学习导论》。《当代物理学》56，172–185（2015）。[10] Maria Schuld 和 Nathan Killoran。《特征希尔伯特空间中的量子机器学习》。《物理评论快报》122，040504（2019）。[11] Hsin-Yuan Huang、Michael Broughton、Masoud Mohseni、Ryan Babbush、Sergio Boixo、Hartmut Neven 和 Jarrod R McClean。《数据在量子机器学习中的力量》。《自然·通讯》12，1–9（2021）。[12]Xiao Yuan、Suguru Endo、Qi Zhao、Ying Li 和 Simon C Benjamin。《变分量子模拟理论》。《Quantum》3，191（2019）。[13]Marco Cerezo、Andrew Arrasmith、Ryan Babbush、Simon C Benjamin、Suguru Endo、Keisuke Fujii、Jarrod R McClean、Kosuke Mitarai、Xiao Yuan、Lukasz Cincio 等。《变分量子算法》。《自然综述：物理学》3，625–644（2021）。[14] Lennart Bittel 和 Martin Kliesch。《训练变分量子算法是 NP 难问题》。《物理评论快报》127，120502（2021）。[15] John Preskill。《NISQ 时代及以后的量子计算》。《Quantum》2，79（2018）。[16] Kishor Bharti、Alba Cervera-Lierta、Thi Ha Kyaw、Tobias Haug、Sumner Alperin-Lea、Abhinav Anand、Matthias Degroote、Hermanni Heimonen、Jakob S Kottmann、Tim Menke 等。《含噪声中等规模量子算法》。《Reviews of Modern Physics》94，015004（2022）。[17] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga、Alban Desmaison、Andreas Kopf、Edward Yang、Zachary DeVito、Martin Raison、Alykhan Tejani、Sasank Chilamkurthy、Benoit Steiner、Lu Fang、Junjie Bai 和 Soumith Chintala。《PyTorch：一种命令式风格的高性能深度学习库》。载于《神经信息处理系统进展》第32卷，第8024–8035页。Curran Associates, Inc.（2019）。[18]John Aycock。《即时编译简史》。《《ACM计算调查》（CSUR）》35，97–113（2003）。[19] 《Paddle Quantum》。 https://github.com/PaddlePaddle/Quantum （2020）。[20]Michael A. Nielsen 和 Isaac L. Chuang。《量子计算与量子信息：十周年纪念版》。Cambridge University Press（2011）。

Johnnie Gray 和 Stefanos Kourtis。《超优化张量网络收缩》。《Quantum》5，410（2021）。[21] Vibhav Gogate 和 Rina Dechter。《一种求解树宽的完备随时算法》（2012）。arXiv:1207.4109。[22]G Daniel、Johnnie Gray 等。《Opt\[23]einsum：一种用于优化类似 einsum 的表达式之收缩顺序的 Python 软件包》。《Journal of Open Source Software》3，753（2018）。_ Sebastian Schlag。《高质量超图划分》。博士论文。卡尔斯鲁厄理工学院，德国。（2020）。[24] Yaroslav Akhremtsev、Tobias Heuer、Peter Sanders 和 Sebastian Schlag。《直接 k 路超图划分算法的工程化实现》。载于第19届算法工程与实验研讨会（ALENEX 2017），第28–42页。（2017）。[25] Cupjin Huang、Fang Zhang、Michael Newman、Xiaotong Ni、Dawei Ding、Junjie Cai、Xun Gao、Tenghui Wang、Feng Wu、Gengyan Zhang 等。《用于模拟量子计算的张量网络收缩高效并行化》。《Nature Computational Science》1，578–587（2021）。[26] Edwin Pednault、John A Gunnels、Giacomo Nannicini、Lior Horesh、Thomas Magerlein、Edgar Solomonik、Erik W Draeger、Eric T Holland 和 Robert Wisnieff。《通过张量缩并延迟实现帕累托高效的量子电路模拟》（2017）。arXiv:1710.05867。[27]Adam Paszke、Sam Gross、Soumith Chintala、Gregory Chanan、Edward Yang、Zachary DeVito、Zeming Lin、Alban Desmaison、Luca Antiga 和 Adam Lerer。《PyTorch 中的自动微分》。载于 NIPS 2017 自动微分研讨会。（2017）。网址：[28] https:。//openreview.net/forum?id=BJJsrmfCZ James Bradbury、Roy Frostig、Peter Hawkins、Matthew James Johnson、Chris Leary、Dougal Maclaurin、George Necula、Adam Paszke、Jake VanderPlas、Skye Wanderman-Milne 等。《JAX：Autograd 与 XLA》。Astrophysics Source Code LibraryPages ascl–2111（2021）。网址：https://ui.adsabs.harvard.edu/abs/2021ascl.soft11002B/abstract.[30]Maria Schuld、Ville Bergholm、Christian Gogolin、Josh Izaac 和 Nathan Killoran。《在量子硬件上评估解析梯度》。Phys. Rev. A 99, 032331 (2019)。[31]Gian-Luca R Anselmetti、David Wierichs、Christian Gogolin 和 Robert M Par- rish。《适用于费米子系统的局域、表达能力强且保持量子数的 VQE 变分形式》。New Journal of Physics 23, 113010 (2021)。[32] Rahul Nandkishore 和 David A Huse。《量子统计力学中的多体局域化与热化》。Annu. Rev. Condens. Matter Phys. 6, 15–38 (2015)。[33] Ian Goodfellow、Jean Pouget-Abadie、Mehdi Mirza、Bing Xu、David Warde-Farley、Sherjil Ozair、Aaron Courville 和 Yoshua Bengio。《生成对抗网络》。Communications of the ACM 63, 139–144 (2020)。[34] Jinkai Tian、Xiaoyu Sun、Yuxuan Du、Shan-shan Zhao、Qing Liu、Kaining Zhang、Wei Yi、Wanrong Huang、Chaoyue Wang、Xingyao Wu 等。《生成式学习中量子神经网络的最新进展》(2022)。arXiv:2206.03066。[35] He-Liang Huang、Yuxuan Du、Ming Gong、Youwei Zhao、Yulin Wu、Chaoyue Wang、Shaowei Li、Futian Liang、Jin Lin、Yu Xu 等。《用于图像生成的实验性量子生成对抗网络》。Physical Review Applied 16, 024051 (2021)。[36] Yuxuan Du、Tao Huang、Shan You、Min-Hsiu Hsieh 和 Dacheng Tao。《变分量子算法的量子电路架构搜索》。npj Quantum Information 8, 1–8 (2022)。[37] Kehuan Linghu、Yang Qian、Ruixia Wang、Meng-Jun Hu、Zhiyuan Li、Xuegang Li、Huikai Xu、Jingning Zhang、Teng Ma、Peng Zhao 等。《超导处理器上的量子电路架构搜索》(2022)。arXiv:2201.00934。[38]Dmitry A. Abanin、Ehud Altman、Immanuel Bloch 和 Maksym Serbyn。《专题综述：多体局域化、热化与纠缠》。Rev. Mod. Phys. 91, 021001 (2019)。[39] Ming Gong、He-Liang Huang、Shiyu Wang、Chu Guo、Shaowei Li、Yulin Wu、Qin-

gling Zhu、Youwei Zhao、Shaojun Guo、Hao-ran Qian 等。《在一台 61 量子比特可编程超导处理器上对量子多体态进行量子神经传感》(2022)。arXiv:2201.05957。

[40] He-Liang Huang、Yuxuan Du、Ming Gong、Youwei Zhao、Yulin Wu、Chaoyue Wang、Shaowei Li、Futian Liang、Jin Lin、Yu Xu、Rui Yang、Tongliang Liu、Min-Hsiu Hsieh、Hui Deng、Hao Rong、Cheng-Zhi Peng、Chao-Yang Lu、Yu-Ao Chen、Dacheng Tao、Xi- aobo Zhu 和 Jian-Wei Pan。《用于图像生成的实验性量子生成对抗网络》。 Physical Review Applied 16, 024051 （2021）。