计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220738
61(4): 840−855, 2024

RR-SC: 边缘设备中基于随机计算神经网络的运行时可重配置框架

宋玉红　沙行勉　诸葛晴凤　许　瑞　王　寒
（华东师范大学计算机科学与技术学院　上海　200062）
（yhsong@stu.ecnu.edu.cn）

RR-SC: Run-Time Reconfigurable Framework for Stochastic Computing-Based Neural Networks on Edge Devices

Song Yuhong, Edwin Hsing-Mean Sha, Zhuge Qingfeng, Xu Rui, and Wang Han
（*School of Computer Science and Technology*, *East China Normal University*, *Shanghai* 200062）

Abstract　With the development of AI democratization, deep neural networks (DNNs) have been widely applied to edge devices, such as smart phones and automated driving, etc. Stochastic computing (SC) as a promising technique performs fundamental machine learning (ML) tasks using simple logic gates instead of complicated binary arithmetic circuits. SC has advantages of low-power and low-cost DNNs execution on edge devices with constrained resources (e.g., energy, computation and memory units, etc.). However, previous SC work only designs one group of setting for fixed hardware implementation, ignoring the dynamic hardware resources (e.g., battery), which leads to low hardware efficiency and short battery life. In order to save energy for battery-powered edge devices, dynamic voltage and frequency scaling (DVFS) technique is widely used for hardware reconfiguration to prolong battery life. In this paper, we creatively propose a run-time reconfigurable framework, namely RR-SC, for SC-based DNNs and first attempt to combine hardware and software reconfigurations to satisfy the time constraint of inference and maximally save energy. RR-SC using reinforcement learning (RL) can generate multiple groups of model settings at one time, which can satisfy the accuracy constraints under different hardware settings (i.e., different voltage/frequency levels). The solution has the best accuracy and hardware efficiency trade-off. Meanwhile, the model settings are switched on a backbone model at run-time, which enables lightweight software reconfiguration. Experimental results show that RR-SC can switch the lightweight settings within 110 ms to guarantee the required real-time constraint at different hardware levels. Meanwhile, it can achieve up to 7.6 times improvement for the number of model inference with only 1% accuracy loss.

Key words　edge devices; stochastic computing; run-time reconfiguration; dynamic hardware conditions; lightweight software reconfiguration; reinforcement learning

摘　要　随着人工智能民主化的发展,深度神经网络已经被广泛地应用于移动嵌入式设备上,例如智能手机和自动驾驶领域等.随机计算作为一种新兴的、有前途的技术在执行机器学习任务时使用简单的逻辑门而不是复杂的二进制算术电路.它具有在资源（如能源、计算单元和存储单元等）受限的边缘设备上执行深度神经网络低能耗、低开销的优势.然而,之前的关于随机计算的工作都仅仅设计一组模型配置并

收稿日期: 2022-08-22; **修回日期**: 2023-05-19
基金项目: 国家自然科学基金项目(61972154); 上海市科委项目(20511101600)
This work was supported by the National Natural Science Foundation of China (61972154) and Shanghai Science and Technology Commission Project (20511101600).
通信作者: 诸葛晴凤(qfzhuge@cs.ecnu.edu.cn)

在固定的硬件配置上实现，忽略了实际应用场景中硬件资源（如电池电量）的动态改变，这导致了低硬件效率和短电池使用时间. 为了节省电池供电的边缘设备的能源，动态电压和频率调节技术被广泛用于硬件重配置以延长电池的使用时间. 针对基于随机计算的深度神经网络，创新性地提出了一个运行时可重配置框架，即 RR-SC，这个框架首次尝试将硬件和软件的重配置相结合以满足任务的时间约束并最大限度节省能源. RR-SC 利用强化学习技术可以一次性生成多组模型配置，同时满足不同硬件配置（即不同的电压/频率等级）下的准确率要求. RR-SC 得到的解具有最好的准确率和硬件效率权衡. 同时，多个模型配置运行时在同一个主干网络上进行切换，从而实现轻量级的软件重配置. 实验结果表明，RR-SC 可以在 110 ms 内进行模型配置的轻量级切换，以保证在不同硬件级别上所需的实时约束. 同时，它最高可以实现 7.6 倍的模型推理次数提升，仅造成 1% 的准确率损失.

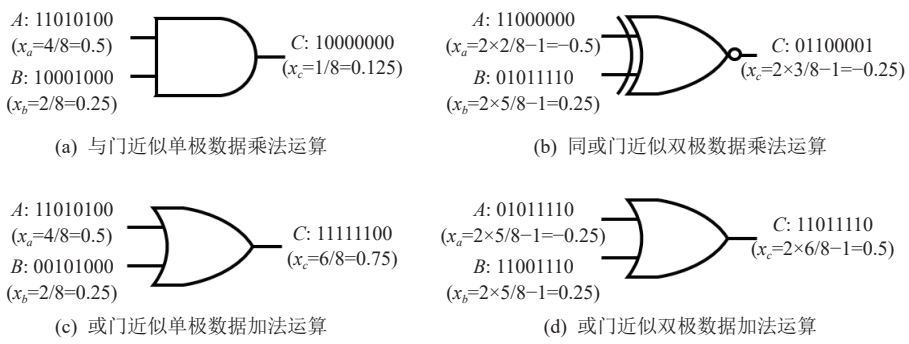
关键词 边缘设备；随机计算；运行时重配置；动态硬件环境；轻量级软件重配置；强化学习

中图法分类号 TP302

随着人工智能的发展，深度神经网络（deep neural network, DNN）作为典型的机器学习模型已经被广泛地应用于边缘设备上[1-2]，例如自动驾驶、智能家居、智慧医疗等应用领域. 然而，边缘设备拥有的资源（如能源、计算单元和存储单元等）是十分有限的，这限制了 DNN 在边缘设备上的高效部署. 尽管许多压缩技术[3-9]通过减少神经网络模型参数在一定程度上能够节省资源的使用，但随着压缩率的持续升高，模型推理准确率会产生明显的下降. 随机计算（stochastic computing, SC）[10-12]作为一种新兴的计算技术由于其特殊的数据表示和计算模式，使得其对于神经网络在硬件上的部署具有低能耗、低开销的优势. SC 通过使用简单的逻辑门近似运算以替代传统浮点数计算中复杂的算术单元，大大降低了计算单元单位时间内的能耗，以及计算单元和存储单元的使用量.

SC 使用一个随机的 0/1 比特串来表示数据，它不同于传统的二进制比特串——每个位置具有不同的权重值. 对于 SC 数据来说，每个位置的权重都是一样的. 因此，SC 数据在噪声环境中具有很高的数据容错性，1 个比特位发生翻转，对整体数值的影响并不大. SC 数据具有 2 种数据类型：单极（unipolar）数据和双极（bipolar）数据. 单极数据表示的是 [0, 1] 范围内的实数，双极数据表示 [−1, 1] 范围内的实数. 对单极数据来说，对任意的实数 $x(0 \leqslant x \leqslant 1)$，0/1 比特串中的每个比特都有 x 的概率为 1，都有 $1-x$ 的概率为 0，即 $x = P(\text{串中出现}1)$. 图 1（a）（c）中的比特串 A, B, C 均为单极数据表示. 例如，比特串 A：11010100 的长度为 8，其中出现 4 个 1，因此该比特串表示的数据 $x_a = 4/8 = 0.5$. SC 也可以表示 −1～1 的数据 x，即双极数据. 这种数据表示方式只需要对比特串中 1 出现的概率进行简单地变换即可，即 $x = 2 \times P(\text{串中出现}1) - 1$. 因为概率 P 的范围一定是 [0, 1]，因此双极数据 x 通过线性变换被调整到 [−1, 1] 范围内. 图 1（b）（d）中的比特串 A, B, C 均为双极数据表示. 例如，比特串 A：11000000 的长度为 8，其中出现 2 个 1，则 $P = 2/8 = 0.25$，那么数据 $x_a = 2 \times 0.25 - 1 = -0.5$. 对于 SC 数据来说，数据精度由比特串长度决定，数据比特串的位数越长数据表示越精确.

基于这样的数据表示方式，传统计算中的算术运算就可以利用概率知识来近似计算. 利用 SC 技术，传统浮点数运算中复杂的加法器、乘法器等算术单元可以用简单的逻辑门代替. 例如，对于单极数据来



(a) 与门近似单极数据乘法运算　(b) 同或门近似双极数据乘法运算

(c) 或门近似单极数据加法运算　(d) 或门近似双极数据加法运算

Fig. 1 Examples of SC based arithmetic units

图 1 基于随机计算的算术单元示例

说，可以利用一个简单的与门(AND)来近似乘法运算(见图 1(a))．它是利用概率计算中，2 个独立事件 A，B，$P(A\cap B)=P(A)\times P(B)$来近似计算，即$P(A\wedge B)=x_a\times x_b$．对于双极数据的乘法来说(见图 1(b))，可以用同或门(XNOR)来近似．在概率计算中，$P(A\odot B)=P(A)P(\overline{B})+P(\overline{A})P(B)$，即$P(A\odot B)=\frac{x_a+1}{2}\times\left(1-\frac{x_b+1}{2}\right)+\left(1-\frac{x_a+1}{2}\right)\times\frac{x_b+1}{2}=\frac{x_ax_b+1}{2}$，$P(A\odot B)$的结果正好就是$x_ax_b$结果的双极数据表示．因此，双极数据的乘法可以用同或门来近似见．类似地，单极数据和双极数据的加法都可以用或门来近似，见图 1(c)(d)．利用概率计算，SC 算术单元的复杂度被极大地简化，这减少了算术单元的运行能耗和硬件使用面积．但是，这种近似方式会产生较大的计算误差．因为要满足上述的概率计算公式，比特流A和流B的生成需要是独立事件，且在用或门近似加法运算时需要满足A，B是互斥事件．因此，对于 SC 来说，设计高准确率的算术单元至关重要．

现有的基于 SC 的工作[13-16]大多集中于设计高准确率的算术电路，接着利用这些算术电路将 DNN 模型部署到硬件设备上进行推理．这种部署往往通过启发式的方法确定一组模型配置(如模型结构、数据比特串长度等)用于硬件实施．并且，这种部署不会考虑动态改变的硬件环境(例如，电池电量会随着消耗而减少)．这些设计往往能够获得较高的推理准确率，却忽略了硬件效率和电池使用时间．另外一些工作[17-18]考虑设计数据精度缩放方法以灵活地改变数据比特串长度，但是神经网络依然被运行在固定的硬件配置上，使得硬件的使用效率没有被充分发挥．

因此，面对有限且动态变化的硬件资源，为了满足模型推理的时间约束和准确率要求，本文针对基于 SC 的神经网络创新性地提出了一个运行时可重配置的框架 RR-SC.对于电池搭载的边缘设备来说，面对动态变化的电池电量，本文提出利用动态电压和频率调节(dynamic voltage and frequency scaling, DVFS)技术来进行硬件层面重配置以节省能源使用．DVFS 根据剩余的电池电量情况，对硬件的电压和频率进行相应的调节．在电池电量充足时，使用较高的电压/频率(voltage/frequency, V/F)等级使模型保持在一个较快的速度上进行推理；在电池电量降低到一定程度时，切换至较低的硬件 V/F 等级以降低单位时间内的推理能耗，使得模型能够在有限的能源内运行更多次数，以延长电池的使用时间．除了硬件层面使用 DVFS 进行重配置外，RR-SC 还结合了软件层面的重配置以满足模型推理的实时性约束．如果在模型推理时仅仅在不同的电量对硬件的 V/F 等级进行切换，会使得一组模型配置在高 V/F 等级时能够满足时间约束，而切换到低 V/F 等级时推理时间变长，预先设定的时间约束无法被满足．因此，我们设计了 RR-SC 框架自动地为不同的 V/F 等级选择出最佳的软件配置．RR-SC 利用强化学习(reinforcement learning, RL)[19-20]技术一次性为不同的 V/F 等级生成对应的模型配置，即模型结构、SC 算术单元、数据比特串长度等，以满足模型推理的时间约束和不同 V/F 等级下的准确率要求，同时尽可能地延长电池的使用时间．为了实现运行时的轻量级切换，RR-SC 只生成 1 组模型结构作为主干网络，从而能够在不同的硬件配置下快速切换．

本文的主要贡献包括 3 个方面：

1) 为了满足基于 SC 的神经网络推理的实时性要求，并尽可能地延长电池搭载的边缘设备的使用时间，本文创新性地提出将硬件重配置和软件重配置相结合．

2) 基于自动机器学习技术提出了一个运行时可重配置的框架 RR-SC，以自动地生成硬件配置对应的模型配置(即软件重配置)．选择的软件配置拥有最好的准确率和运行效率的权衡．除此之外，模型运行时在一个搜索出来的主干网络上进行软件配置切换，以实现轻量级软件重配置．

3) 实验结果表明，RR-SC 能够保证在所有硬件级别下满足实时约束，并且可以在准确率损失仅为 1% 的情况下将模型推理次数增加 7.6 倍．同时，它还能在 110 ms 内满足不同软件配置的轻量级切换．

1 相关工作

SC 近似计算产生的高计算误差使得基于 SC 的 DNN 实现的相关研究[13-16]主要集中于设计高准确率的硬件电路．由于乘积累加(multiply-accumulate, MAC)运算是 DNN 实现中的基本运算，因此我们主要讨论乘法器和加法器的设计与实现．文献 [13] 提出使用同或门作为乘法器，并且提出了一个基于多选器树(MUX-tree)的加法器，我们称这个算术电路为 XNOR-MUX.XNOR-MUX 受制于比特流间的相关性问题(即概率近似失效)，使得计算误差较大．文献 [14] 将正负数据分开计算，并利用与门作为乘法器．同时，它提出一种新的加法器，这种加法器分别计算正数数据的和(*POS*)与负数数据的和(*NEG*)，最后利用

$\frac{POS-NEG+1}{2}$对结果进行巧妙地转换，这种转换只需要一个 MUX 门，使得单极数据计算的结果正好为双极数据表示下正数部分和负数部分的差. 我们称这个电路为 AND-SEP. 但当输入数据数量较多时，该加法器在没有数据缩放时会产生计算溢出问题使得计算误差较大. 文献 [15] 为双极数据设计了新的乘法器 uMUL 和加法器 uNSADD，并利用并行计数器来缓解输入的相关性问题. 但在某些特殊情况下会出现冗余 1 或缺失 1 的问题从而产生计算误差，这种电路被称为 uGEMM. 文献 [16] 进一步提出了一种基于累加器的块内加法器以减少计算错误，并设计了一种输出修正方案来解决块之间的冗余/缺失 1 的问题. 块内的算术电路被称为 AND-ACC，带有输出修正方案的电路被称为 AND-LOOP.这些设计虽然相比传统的二进制电路具有更低的运行能耗，但它们都没有考虑电池搭载的边缘设备电量动态改变的情况，因此没有涉及任何软件和硬件的重配置. 当神经网络被部署到硬件上时，模型推理始终运行在相同的 V/F 等级下，模型的推理会快速消耗电池电量，使得模型最终推理总次数较少.

另一方面，文献 [17−18] 考虑了运行时比特串长度的动态改变. 文献 [17] 提出一个新的计算提前终止方法，使得模型推理时可以只利用较少位数的比特串就能获得较好的推理准确率. 文献 [18] 扩展了文献 [21] 的方法并设计了新的数据表示方式，使得输入、输出数据的精度可以在运行时任意更改. 这些工作虽然实现了运行时软件层面的重配置，一定程度上延长了电池的使用时间，但它们都运行在固定的硬件配置下，使得硬件资源没有被充分利用.

因此本文提出一种软/硬件协同重配置的方式，利用硬件重配置以尽可能地延长电池使用时间，增加模型的推理总次数. 同时搭配软件重配置以满足机器学习任务的实时性要求. 本文提出的方法充分利用硬件资源，实现高效实时的模型推理. 同时，保证了模型推理的准确率.

2 动机和问题定义

2.1 动　机

动机 1: 电池搭载的边缘设备的能源是有限且动态变化的. 大多数移动设备都是电池供电的，例如手机、无人机、机器人等[22]. 延长能源有限的电池使用时间是很重要的. 同时，任务执行需要符合实时性要求. 面对有限的能源资源，许多模型压缩方法[3-5]通过减小模型大小和参数数量在一定程度上提高模型推理速度以降低能源消耗. 但是这些压缩方法依然依赖于复杂的浮点数算术电路运算，在能源消耗方面仍有很大的优化空间. SC 利用新的数据表达方式和简化的算术电路进一步降低模型推理能耗，但由于数据表示精度、电路计算误差等问题，使得许多研究者都致力于研究高准确率的算术电路，忽略了硬件的效率和动态变化的硬件资源. 因此，面对逐渐减少的电池电量情况，DVFS[23]技术常被用于硬件重配置，以节省能源使用. 表 1 展示了 Odroid-XU3 平台中 Cortex A7 内核的可用 V/F 等级. 本文针对电池搭载的边缘设备，在电池电量较高时配置较高的 V/F 等级；当电池电量降低到一定程度时，切换到更低的 V/F 等级. 在本文中，我们称这种动态电压和频率等级调节为硬件重配置.

Table 1　V/F Levels Supported by ARM Cortex A7 Core in Odroid-XU3 Mobile Platform

表 1　Odroid-XU3 移动平台中 ARM Cortex A7 内核支持的电压/频率等级

等级	频率/MHz	电压/mV
l_1	400	916.25
l_2	600	917.5
l_3	800	992.5
l_4	1 000	1 066.25
l_5	1 200	1 141.25
l_6	1 400	1 240

动机 2: 仅仅使用硬件重配置无法满足实时性要求. 我们通过对比实验验证了硬件重配置能够延长电池使用时间，实验结果展示在表 2 中. 为了度量电池的使用时间，本文用模型在一定能源总量下的推理总次数来表示，表 2 中所有的方法都具有相同的能源总量. 实验设置 S1 和 S2 使用了相同的多层感知机(multi-layer perceptrons, MLP)模型. 我们设定 MLP 模型的结构为 784-100-200-10，表示 MLP 的 3 层全连接层分别有 100, 200, 10 个神经元，784 表示输入数据的大小. 模型推理时使用 AND-ACC 算术电路. 在本实验中的模型配置均使用默认设置，但对于 RR-SC 方法，模型的配置通过优化方法确定. 实验设置 S1 仅使用一种模型配置，且模型推理运行在固定的硬件配置下，这里的 SC 数据使用 32 b 的 0/1 串表示. 在实验设置 S2 中使用了硬件重配置(即 DVFS)，S2 一共设置了 3 个 V/F 等级 F, N, E. F 模式表示使用高

Table 2 Running Results Comparison of Three Experimental Settings with the Time Constraint of 45 ms

表 2 时间约束 45 ms 下的 3 种实验设置运行结果对比

设置	模型	数据精度	准确率/%	DVFS	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升
S1	M1	32	95.4	F	32.7	√	298.1	2.0×10^5	-
S2	M1	32	95.4	F	32.7	√	298.1	1.0×10^5	1.4 倍
				N	45.8	×	230.2	7.8×10^4	
				E	57.2	×	124.1	9.7×10^4	
S3	M1	32	95.4	F	32.7	√	298.1	1.0×10^5	2.3 倍
	M2	16	88.1	N	28.9	√	142.1	1.3×10^5	
	M3	8	62.2	E	23.8	√	51.8	2.3×10^5	

注：实验设置 S1 表示没有任何运行时重配置，S2 表示仅具有硬件重配置，S3 表示同时具有软件和硬件重配置；“-”表示性能比较基准.

V/F 等级下的快速执行模式；N 模式表示使用中间 V/F 等级下的普通速度执行模式；E 模式表示使用低 V/F 等级下的节省能耗的执行模式. 相应地，我们选择表 1 中的 l_6，l_4，l_3 分别作为 F 模式、N 模式和 E 模式下的 V/F 设置. 从表 2 中我们可以看出，在 S2 实验设置下，由于配置了 DVFS 技术，模型的总的推理次数相比于 S1 实验设置提升了 1.4 倍，这表明 DVFS 能够在一定程度上延长电池使用时间. 然而，当 V/F 等级降低时，如 S2 实验设置下的 N 模式和 E 模式的模型推理时间无法满足时间约束. 因为当频率降低时，计算速度也相应地变慢. 在本实验中，我们设置的时间约束为 45 ms.

为了解决仅使用硬件重配置无法满足实时性要求的问题，我们探索了软件重配置以搭配不同的硬件配置. 在不同的硬件 V/F 等级下搭配不同的数据精度，以降低运行的时间和能耗、满足实时性要求和增加模型推理总次数. 我们同样用实验来证明，如表 2 中的 S3 实验设置：结合硬件重配置和软件重配置. 实验结果表明，S3 设置下模型推理总次数是 S1 模式下的 2.3 倍，并且相比于 S2 模式提升了近 40%. 与此对应的是，模型推理准确率产生了一定程度的损失. 通过搭配 DVFS，模型推理总次数显著增加，电池一次放电的使用时间延长. 接着搭配软件重配置，也就是不同的模型配置，模型推理的时间约束能够被满足.

2.2 问题定义

本文为了解决动机 1 和动机 2 对应的问题，提出了一个完整的基于自动机器学习的框架，能够为不同的硬件配置一次性选择多种软件配置. 本文的问题定义为：

定义 1. 给定边缘设备能源总预算 TE，推理时间约束 TC，一组运行时用于切换的 V/F 等级 $L=\{l_1, l_2,\cdots,l_i,\cdots\}$，各等级下的准确率约束 $AC=\{Ac_1,Ac_2,\cdots,Ac_i,\cdots\}$，本文框架主要用于决定：

1）一个主干网络模型结构 BM；

2）一个随机计算算术电路 C；

3）多组用于软件重配置的比特串长度 $BL=\{bl_1,bl_2,\cdots,bl_i,\cdots\}$.

运行时，硬件根据剩余的能源（即电池电量）切换相应的 V/F 等级，被选择出来的主干模型 BM 使用算术电路 C 进行推理运算，根据不同的 V/F 等级切换相应的数据精度 BL. 通过 RR-SC 选择出来的软件配置可以满足时间约束 TC 和各级准确率约束 AC，同时最大化模型 BM 在不同 V/F 等级下的推理次数总和以及模型推理准确率. 本文使用模型在总能源预算 TE 下的推理次数来表示硬件的效率. 因此本文的优化问题可以表示为：

$$\text{Maximum } Runs + Acc$$
$$\text{s.t. } \forall lat_i \leqslant TC \text{and} \forall A_i \leqslant Ac_i,$$

其中 $Runs$ 表示模型在 V/F 等级 L 下的总运行次数，Acc 表示多个模型推理的平均准确率，lat_i 和 A_i 分别表示各 V/F 等级下的模型推理时间和准确率.

3 RR-SC 框架设计

3.1 总体设计

图 2 展示了 RR-SC 框架的整体设计. 我们建立了一个基于循环神经网络（recurrent neural network，RNN）的 RL 控制器（见 3.2 节），用来从预先定义好的离散的搜索空间选择出相应的软件配置. RL 控制器利用深度机器学习的方法，可以通过训练优化以预测出更好的结果. 离散搜索空间由模型结构（包括层数、每层神经元个数、卷积核大小等）、SC 算术电路库、数据精度（即数据比特串长度）组成. RL 控制器

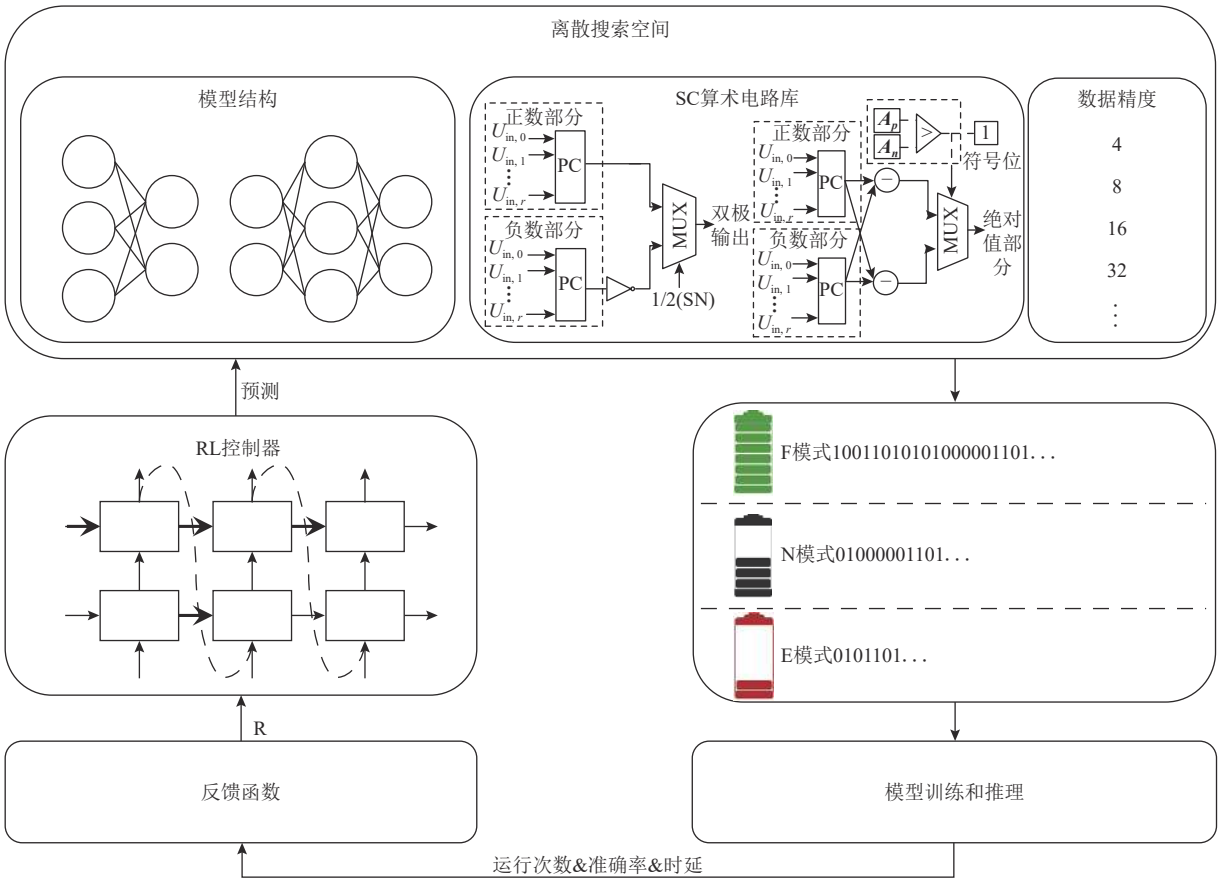


Fig. 2 RR-SC framework

图 2 RR-SC 框架

选择出来一组模型结构作为主干网络 *BM*, 在不同的 V/F 等级下均使用相同的 SC 算术单元 *C* 推理以节省能耗和切换开销. 同时 RL 控制器生成多种数据精度 *BL*, 以在不同的 V/F 等级下进行轻量级切换. 当模型配置确定后, 主干网络接着被训练和推理以获得各等级的能耗、时延和准确率(见 3.3 节). 这些指标被输入设计好的反馈函数(见 3.4 节)得到奖励值 *R*, *R* 值被用于更新 RL 控制器参数使其能够快速收敛. RL 控制器的选择过程被执行 *EP* 次, 每次通过从环境中获得的奖励值 *R* 来更新 RNN 网络的参数以指导下一次的参数选择.

3.2 强化学习控制器

为了一次性地获得能够满足时间约束 *TC* 和准确率约束 *AC* 的软件配置, 本文利用 RL 技术来为基于 SC 的 DNN 智能、高效地选择最好的参数, 以在固定能源总量下获得最多的模型推理次数和最好的推理准确率. RL 控制器是 RR-SC 框架的核心部件, 它的设计基于深度学习网络模型. DNN 的应用使得 RL 控制器在收敛到全局最优解方面具有很大的优势. 智能体(agent)每次自动地观察环境状态并通过奖励值进行参数的自我更新, 以在未来的执行过程中得到更好的动作预测结果.

1)状态空间(state space). 本文使用基于 SC 的 DNN 在相应参数配置下的推理准确率*Acc*, 及在给定能源总量和 V/F 设置下的推理总次数*Runs*来表征当前的环境状态, 如表 3 所示. RL 控制器利用以往动作及相应的反馈值作为输入以学习以往的经验, 用以训练自身网络, 使其未来能够预测出拥有更高反馈值的参数组合. 反馈函数中的性能值能够全面反映模型在嵌入式设备上推理时的不同方面, 即推理总次数反映了运行时的硬件效率, 平均推理准确率反映了模型的分类水平. 它们能够使智能体更好地理解复

Table 3 State Space of RL Controller

表 3 RL 控制器的状态空间

状态	描述
Acc	基于 SC 的 DNN 在相应动作下的推理准确率
Runs	基于 SC 的 DNN 在给定能源总量及相应 V/F 设置下的推理总次数

杂的系统，并能及时捕捉环境的动态变化.

2)动作空间(action space). 本文的强化学习的动作空间由3类参数的组合空间组成：模型结构、SC算术电路以及数据精度. 其中，模型结构决定了推理模型的大小，它包括模型的层数、每层的神经元个数、卷积核大小、卷积步长、输入通道数等要素，包含哪些要素由具体的模型决定. 接着，本文根据现有的SC算术电路设计工作构建SC算术电路库，包括XNOR-MUX, AND-SEP, uGEMM, AND-ACC, AND-LOOP这5种电路. 数据精度指不同的SC数据表示位数，越长的数据表示越高的数据精度. 值得注意的是，RR-SC是一个可重新配置的框架，它允许将其他因素包含在内，动作空间的值可以根据不同的需求和场景进行调整.

3)智能体. 在RR- SC的设计中，强化学习中的智能体被称为RL控制器. RL控制器的实现基于一个RNN网络，这和文献[24]中的实现类似. RNN相关的网络能够很好地处理序列相关的输入，提取和保留序列中的上下文关系及信息，因此RL控制器有助于通过学习以往动作中的经验，不断训练出更好的RNN网络以做出更好的选择. 首先，RL控制器从动作空间内预测出一组模型结构，一个SC算术单元和多组数据精度用于在不同的V/F等级下进行切换，这种预测是基于一个Softmax分类器. 选择出来的参数用于构建主干模型. 接着进行模型的预训练和硬件推理，通过训练和推理，模型的能耗、时延和准确率会被获得以用于构建反馈函数. 接着，基于反馈函数计算得到奖励值R，一个蒙特卡罗策略梯度算法[25]被应用于更新RL控制器中的参数θ. 更新θ的计算公式为：

$$\nabla J(\theta)=\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\gamma^{T-t}\nabla_{\theta}\mathrm{lb}\pi_{\theta}(a_t|a_{(t-1):1},\theta)(R_k-b),$$

其中，m表示RNN输入的批次大小，T表示每次选择的步数. 奖励值R在每一步都按指数因子减少，基线b是R的平均指数移动. θ的更新是为了让RL控制器在下一轮的搜索中选择拥有更高R值的配置，使其可以更快地收敛到全局最优解.

为了获得更稳定的结果[26]，我们对RR-SC执行多次，并以拥有最高R值的配置作为我们最终的结果. RR-SC的搜索时间开销以多次运行的平均时间计算得到.

3.3 模型训练和推理

当RL控制器选择出来一组模型配置后，模型训练和推理过程被启动. RL控制器选择出来的模型配置包括模型结构，即模型层数、每层神经元个数、卷积层核数、步长等，SC算术电路和多种数据精度. 当选择出来的模型配置通过性能预测器(见3.4节)计算得到的推理时间满足设定的时间约束TC时，开始相应的训练器和模型推理过程. 如果任意V/F等级下的时间约束和准确率约束没有被满足，那么模型的训练和推理过程被提前终止，以节省RR-SC的搜索开销. 图3展示了模型训练和推理模块的主要流程. 首先，对于选择出来的参数，模型结构被解析，对应结构的浮点数模型被构建. 接着，浮点数模型被预训练多次，以获得拥有高准确率的模型参数，即权重和偏置. 为了获得模型在硬件上的推理准确率，基于SC的模型被建立. 模型的结构与选择出来的主干网络模型相同，只是数据的表示由浮点数据转换为SC数据，即单极数据和双极数据. 所有浮点数输入数据和模型参数均通过随机数生成器(stochastic number generator, SNG)转换成SC的数据比特串. 随后，选择出来的算术电路C被用于实现基于SC的DNN模型. 对于不同的硬件配置，即V/F等级来说，被选择出来的多个数据精度分别对应多个V/F等级，应用不同数据精度的模型通过算术电路C推理得到各自的准确率. 多个模型推理的平均准确率用于计算奖励值



Fig. 3 Model training and inference stages used for obtaining energy, latency and accuracy

图3 模型训练和推理阶段用以获得能耗、时延和推理准确率

R. 理论上讲，当硬件配备高 V/F 等级时，相应的高数据精度被选择出来，也就是更长的比特串长度. 这样，在硬件能源（即电池电量）相对充足时，模型推理能够尽可能的准确. 当硬件能源减少到一定程度后，数据精度也相应地被调低，这样在损失一定准确率的情况下，能够尽可能降低模型推理能耗，以延长电池的使用时间.

3.4 性能预测器和反馈函数

为了获得基于 SC 的模型推理时间，我们设计了一个性能预测器. 我们在高级综合（high-level synthesis, HLS）工具上模拟基于 SC 的加法器和乘法器，并获得当前数据精度下每个操作数计算所需的时间. 接着，我们根据选择出来的模型结构计算所有操作的数量. 这样，就可以获得模型在当前 V/F 等级下的推理时间. 那么当模型被切换至其他 V/F 等级时，推理时间可以通过频率的线性变换计算得到. 频率越高，推理的速度越快，推理时间越短. 另外，为了获得当前硬件设置下的模型推理次数，我们通过总能源 TE 和模型一次推理所需要花费的能耗 E_i 计算得到. 模型一次推理所花费的能耗 E_i 通过功率乘以时间得到，即 $E_i = Power_i \times lat_i$. 其中，当前 V/F 等级下的功率和时延通过模型在 HLS 中模拟得到. 那么模型的推理总次数 $Runs$ 即为当前等级的总能源 TE_i 与一次推理所需的能耗 E_i 的比值，即

$$Runs = \sum_i TE_i / E_i.$$

本文假定在相同的工作负载和不变的物理环境下设计，因此忽略工作负载和环境变化带来的功率的动态变化. 除此之外，温度对能耗也有一定的影响，但本文所使用的随机计算技术是一种显著降低模型推理能耗的技术，低能耗技术对工作温度的影响很小甚至没有[27]. 并且，本文在相同的起始温度下对不同的软硬件配置进行能耗预估，温度对能耗的影响对于最后的配置的选择几乎没有影响. 因此本文不考虑温度对能耗的影响.

经过模型训练和推理过程，以及模型预测器的高级综合之后，可以得到模型在各个 V/F 等级下的推理次数和准确率. 接下来，当前轮的奖励值 R 可以通过反馈函数计算用以更新 RL 控制器. 反馈函数同时也用来表示我们的优化目标，即在时间约束 TC 及准确率约束 AC 下，最大化模型推理次数和准确率. 反馈函数的计算公式为：

$$R = \begin{cases} -1 + R_{\mathrm{rs}}, \exists lat_i > TC \text{ 或 } \exists A_i < Ac_i, \\ Avg\left(\dfrac{A_i - A_{\mathrm{l}}}{A_{\mathrm{u}} - A_{\mathrm{l}}}\right) + R_{\mathrm{rs}}, \forall lat_i \leqslant TC \text{ 且 } cond = \mathrm{True}, \\ Avg\left(\dfrac{A_i - A_{\mathrm{l}}}{A_{\mathrm{u}} - A_{\mathrm{l}}}\right) - pen + R_{\mathrm{rs}}, \text{其他}. \end{cases}$$

在介绍具体的函数之前，我们先介绍 5 个符号：1）R_{rs}指的是多个模型推理总次数的归一化值，它的范围在 [0,1]；2）lat_i表示每一种 V/F 等级下模型的推理时间；3）A_i表示各种 V/F 等级下的模型推理准确率，A_{u}和A_{l}是我们预先设置的最高准确率和最低准确率，用以对准确率进行归一化；4）Ac_i表示各 V/F 等级下的准确率约束；5）$cond$是一个二值条件，$cond$ = True 表示对于$\forall i < j$，我们有$A_i > A_j$，意味着运行在高 V/F 等级的模型拥有更高的推理准确率，否则$cond$ = False，在这种情况下我们会给奖励值 R 一个惩罚 pen.

那么，对于计算奖励值 R 有 3 种情况：

1）如果$\exists lat_i > TC$或者$\exists A_i < Ac_i$意味着时间约束 TC 和准确率约束 AC 并不是在任意的 V/F 等级下都被满足，这里我们直接设置$R = -1 + R_{\mathrm{rs}}$. 这意味着，当通过性能预测器得到的模型在选择的动作参数下无法满足边缘设备要求的时间约束 TC 时，就不需要进行真正的模型训练和推理以获得准确率，因此直接将准确率反馈值设置成−1. 这样，RR-SC 的搜索开销在一定程度上就可以减少. 同样地，如果基于 SC 的神经网络模型在某些 V/F 等级下的推理准确率无法满足预期的要求，那么其他 V/F 等级下的模型就不需要再耗费时间和资源进行推理.

2）如果任意 V/F 等级下的推理时间约束和准确率约束都能够被满足，并且$cond$ = True，也就是期望拥有高 V/F 等级的硬件配置能够获得更高的准确率，那么我们使用归一化后的平均准确率和模型推理总次数作为奖励值 R. 这种情况对应着本文优化问题的目标函数，RR-SC 期望在满足时间约束和准确率约束的情况下，最大化运行总次数和推理准确率.

3）否则，我们在第 2 种情况下的奖励值 R 增加一个惩罚值 pen. 因为 RR-SC 更希望能够在高 V/F 等级下拥有更高的推理准确率，给定一个惩罚值相当于告诉智能体这是一种不被期望的结果.

4 实验结果及分析

4.1 实验设置

1）实验方法对比. 首先，我们将 RR-SC 框架与之前的基于 SC 的工作[13-16] 进行对比. 本文方法结合了

软件和硬件的重配置,在满足实时性约束的前提下最大限度地延长电池使用时间.通过利用 RL 技术一次性选择多种软件配置对应于多种硬件配置.硬件配置根据剩余能源电量进行 V/F 等级的调整,选择出来的软件配置对应各个硬件配置.而现有的前沿工作主要针对算术电路进行设计,缺少硬件和软件重配置.同时,RR-SC 框架选择出来的模型被可视化以便于展示.接着,我们利用 RR-SC 选择出来的模型结构和配置,进一步探究软件重配置和硬件重配置分别对推理时间和能源消耗的影响.

2)实验指标及平台设置.我们使用 MNIST 数据集在 MLP 和 LeNet 模型上评估本文方法,使用 Cifar10 数据集在 AlexNet 模型上验证.搜索空间的设置包括模型结构、SC 算术单元、数据精度 3 个部分.通过探究,我们将搜索空间内所有因素的值都收缩到一个合理的范围内,本文实验设置的搜索空间被总结在表 4 中.对于硬件指标,算术电路由 Bluespec SystemVerilog 实现,然后编译为 Verilog 以在 Vivado v2020.2 上评估能耗、时间等硬件指标.对于准确率指标,我们在 2× NVIDIA Tesla P100 GPU 服务器(16 GB GPU 内存)上进行模型推理和 RL 的搜索过程.实验在 Python 3.6.0, GCC 9.3.0, PyTorch 1.5.1 和 CUDA 10.1 环境上进行.对于边缘设备,我们使用移动平台 Odroid-XU3 [28].在实现 SC 模型中,Sobol[29-30] 随机数生成器被用于从浮点数到 SC 数据的转换.

Table 4 Specific Search Space Settings of MLP, LeNet and AlexNet

表 4 MLP,LeNet,AlexNet 的具体搜索空间设置

模型	搜索空间	设置
MLP	层数	1, 2, 3
	每层神经元数	32, 64, 100, 128, 200, 256, 400, 512, 1 024
	SC 算术电路库	XNOR-MUX, AND-SEP, uGEMM, AND-ACC, AND-LOOP
	数据精度	4, 8, 16, 32, 64, 128
	配置组合数	241 920
LeNet	卷积层、池化层步长	1, 2
	卷积核大小	3, 5, 7
	池化层核大小	1, 2
	卷积层输出通道数	6, 16, 32, 64
	全连接层神经元数	32, 64, 100, 128, 200, 256, 400, 512, 1 024
	SC 算术电路库	XNOR-MUX, AND-SEP, uGEMM, AND-ACC, AND-LOOP
	数据精度	4, 8, 16, 32, 64, 128
	配置组合数	940 584 960
AlexNet	卷积层、池化层步长	1, 2
	卷积核大小	3, 5, 7
	池化层核大小	1, 2
	卷积层输出通道数	6, 16, 32, 64, 256
	全连接层神经元数	32, 64, 100, 128, 200, 256, 400, 512, 1 024
	SC 算术电路库	XNOR-MUX, AND-SEP, uGEMM, AND-ACC, AND-LOOP
	数据精度	4, 8, 16, 32, 64, 128
	配置组合数	30 233 088 000 000

本文使用 LSTM 网络作为 RL 控制器,LSTM 输入的批次大小 $m=35$,每次选择的步数 $T=50$,学习率 $lr=0.99$.本文实验探究 2 种时间约束下的 RR-SC 搜索结果:对于 MLP 来说,本文选择 95 ms 和 120 ms 这 2 种时间约束来进行实验探究;对于 LeNet 来说,时间约束分别为 540 ms 和 2 700 ms;对于 AlexNet 来说,时间约束分别设置为 21 000 ms 和 125 000 ms.本文共设置 3 种 V/F 等级,从高到低分别为表 1 中的 l_6, l_4, l_3 这 3 个等级.相应地,对于在 MNIST 数据集上进行训练的 MLP 和 LeNet 这 2 种模型来说,3 种 V/F 等级下的模型推理准确率约束分别设置为 95%, 90%, 85%;对于在 Cifar10 数据集上进行训练的 AlexNet 模型来说,3 个 V/F 等级下的模型推理准确率约束分别设置为 90%, 85%, 80%.同时,对于反馈函数中的 A_u 和 A_l 分别设置为 98% 和 60%.

4.2 RR-SC 框架评估

首先对于 RR-SC 框架的评估,我们与现有的 SC 相关工作[13-16] 进行对比.这些工作主要针对 SC 算术电路进行设计,没有充分考虑硬件环境的动态改变.为了对比公平,将 SC 相关的工作与通过 RR-SC 搜索到的模型结构设置成一样的,并且数据精度取 RR-SC 方法中搜索出来的最高精度,然后对它们的模型推理准确率、时延及模型推理次数进行比较.

MLP 模型在不同时间约束下的结果与 SC 相关工作的对比展示在表 5 和表 6 中.表 5 展示了时间约束设置为 95 ms 时的结果,表 6 展示时间约束为 120 ms 时的结果.表 5 实验得到的 MLP 的模型结构为 784-200-32-10,表示模型具有 3 层全连接层,并且全连接层的神经元的个数分别为 200, 32, 10.表 6 实验得到的模型结构为 784-128-100-10 这 2 组实验选择的 SC 算术电路均为 AND-LOOP.我们将 RR-SC 选择出来模型结构及相应的算术电路进行可视化展示,如图 4 所示.RR-SC 实验中的硬件配置均设置成表 1 中的 l_6, l_4, l_3.对于其他对比工作来说,V/F 等级设置为 l_6.从实验结果中可以看出,AND-LOOP 方法[16] 拥有最高的推理准确率,XNOR-MUX 方法[13] 拥有最低的推

Table 5 Comparative Results of SC Relative Work and RR-SC with the Time Constraint 95 ms on MLP

表 5 MLP 上 SC 相关工作和 RR-SC 在时间约束为 95 ms 时的比较结果

方法	算术电路	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [13]	XNOR-MUX	64	12.8	627.1	×	2 527.3	23 741	3.9 倍
文献 [14]	AND-SEP	64	25.5	94.7	√	762.6	78 678	1.2 倍
文献 [15]	uGEMM	64	64.8	430.5	×	3 378.3	17 760	5.2 倍
文献 [16]	AND-ACC	64	94.6	93.3	√	818.9	73 269	1.3 倍
文献 [16]	AND-LOOP	64	97.8	90.7	√	1 306.7	45 917	2.0 倍
RR-SC（本文）	AND-LOOP	64	97.8	90.7	√	1 306.7	22 959	
		32	96.3	73.22	√	582.3	30 912	
		32	96.3	91.5	√	313.8	38 241	

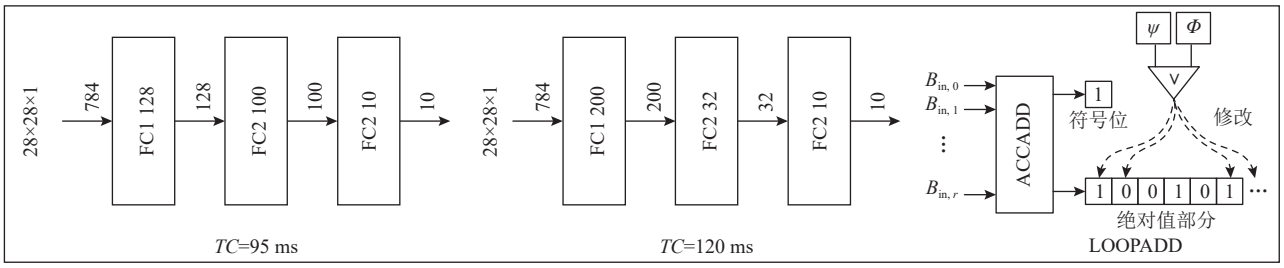
Table 6 Comparative Results of SC Relative Work and RR-SC with the Time Constraint 120 ms on MLP

表 6 MLP 上 SC 相关工作和 RR-SC 在时间约束为 120 ms 时的比较结果

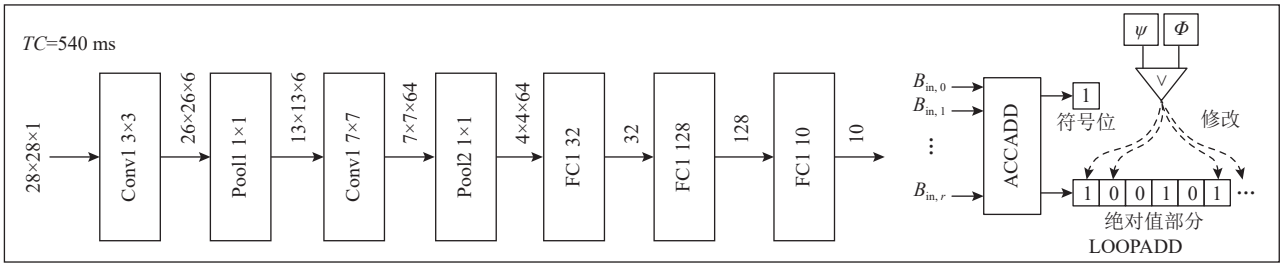
方法	算术电路	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [13]	XNOR-MUX	128	20.8	649.3	×	2 606.1	23 023	4.3 倍
文献 [14]	AND-SEP	128	59.2	123.9	×	954.2	62 880	1.6 倍
文献 [15]	uGEMM	128	84.3	589.8	×	4 623.1	12 978	7.6 倍
文献 [16]	AND-ACC	128	94.3	121.9	×	1 072.3	55 954	1.8 倍
文献 [16]	AND-LOOP	128	98.3	118.2	√	1 682.1	35 670	2.8 倍
RR-SC（本文）	AND-LOOP	128	98.3	118.2	√	1 682.1	17 835	
		64	96.4	88.6	√	695.3	25 888	
		32	95.6	63.9	√	216.3	55 478	

理准确率. RR-SC 方法的平均准确率仅比 AND-LOOP 分别低 1 个百分点和 1.5 个百分点. 然而, RR-SC 的推理次数是 AND-LOOP 推理次数的 2.0 倍和 2.8 倍. 在众多参数中, RR-SC 找到了最好的配置. 对于 XNOR-MUX 和 uGEMM 来说, 它们的模型推理准确率和时间约束都无法满足我们的要求的. 同时, RR-SC 在推理次数方面实现了高达 7.6 倍的提升. 对于 AND-SEP 和 AND-ACC 来说, 它们在 MLP 784-200-32-10 模型能够满足我们的时间约束要求, 但是在 MLP 784-128-100-10 这个模型上的时间约束要求无法被满足. 从准确率和推理时间来说, AND-ACC 是最具竞争力的一个方法. 对于推理次数来说, 由于结合了硬件重配置和软件重配置, RR-SC 比 AND-ACC 实现了约 1.8 倍的性能提升. RR-SC 方法可以在满足一定的时间约束和准确率约束的情况下, 最大化模型推理次数和平均准确率, 拥有最好的性能权衡.

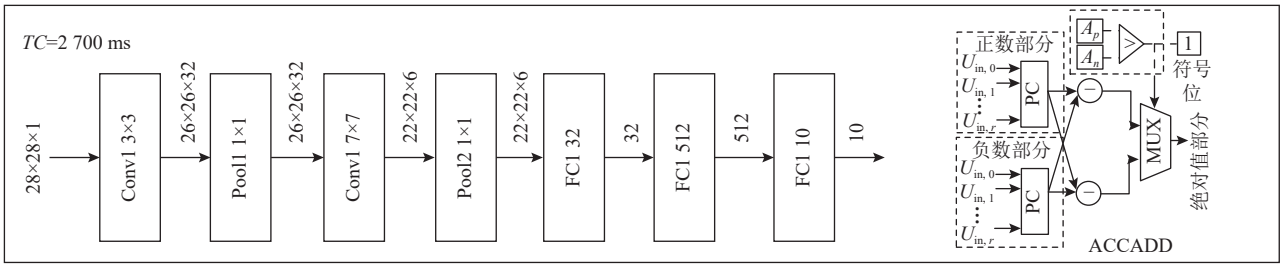
对于 LeNet 来说, 同样地, 一个较紧的时间约束实验结果和一个较松的时间约束实验结果分别被展示在表 7 和表 8 中. 表 7 中的实验设置时间约束为 540 ms, 表 8 实验设置时间约束为 2 700 ms. AlexNet 的实验结果被展示在表 9 和表 10 中, 设置时间约束分别为 21 000 ms 和 125 000 ms. 因为 LeNet 和 AlexNet 相较于 MLP 更大、也更复杂, 因此我们设置了更长的时间约束. 对于表 7 中的实验, 一个结构为 Conv1(k=3, s=1, c=6)-Pooling1(k=1, s=2)-Conv2(k=7, s=1, c=64)-Pooling2(k=1, s=2)-FC1(32)-FC2(128)-FC3(10)的主干网络被选择出来. 对于表 8 中的实验来说, 主干网络结构为 Conv1(k=3, s=1, c=32)-Pooling1(k=1, s=1)-Conv2(k=5, s=1, c=6)-Pooling2(k=1, s=1)-FC1(32)-FC2(512)-FC3(10). 其中, k表示卷积层或池化层的核大小, s表示卷积层和池化层的滑动步长, c表示卷积层的输出通道数. 在卷积层和池化层之后紧接 2 个全连接层. 对于 AlexNet 网络来说, 表 9 实验下选择出来的主干网络模型为 Conv1(k=5, s=1, c=256)-Pooling1(k=1, s=1)-Conv2(k=3, s=2, c=32)-Pooling2(k=1, s=1)-Conv3(k=7, s=2, c=16)-Conv4(k=5, s=2, c=16)-Conv5(k=3, s=1, c=256) -Pooling3(k=1, s=1)-FC1(512)-FC2(1 024)-FC3(10), 表 10 实验下选择出来的主干网络模型为 Conv1(k= 7, s=2, c=256)-Pooling1(k=1, s=1)-Conv2(k=7, s=1, c=64)-Pooling2(k=1, s=1)-Conv3(k=3, s=1, c=256)-Conv4(k=7, s=1, c=32)-Conv5(k=7, s=1, c=256)-FC1(1 024)-FC2(512)-FC3(10). 同样地, 对于 LeNet 和 AlexNet 的实验中选择出



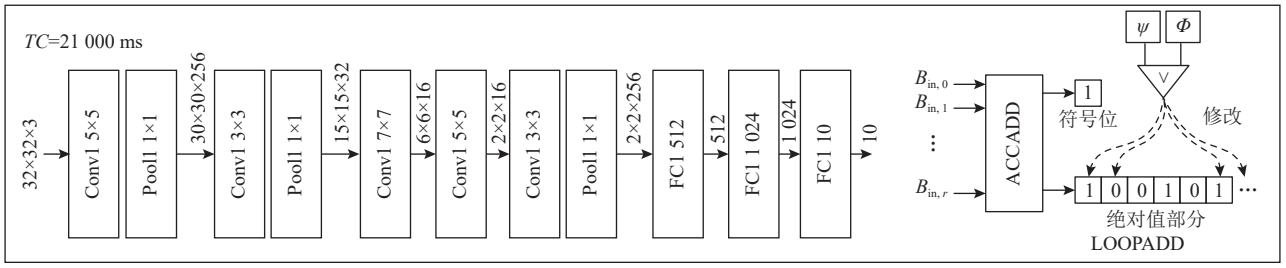
(a) 95 ms和120 ms时间约束下搜索到的MLP模型结果



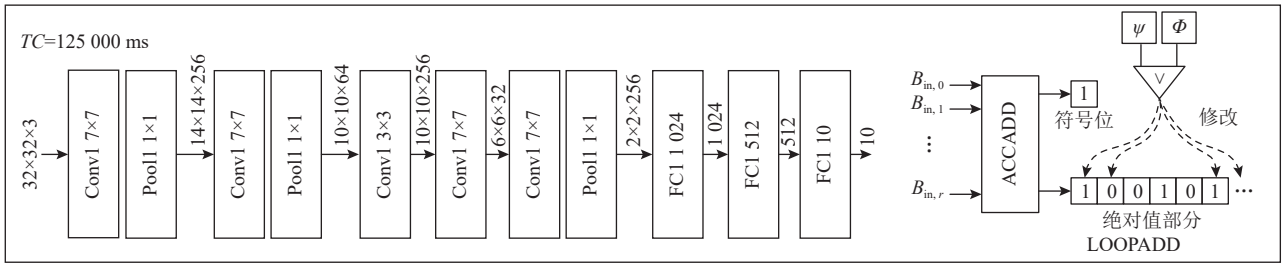
(b) 540 ms时间约束下搜索到的LeNet模型结果



(c) 27 00 ms时间约束下搜索到的LeNet模型结果



(d) 21 000 ms时间约束下搜索到的AlexNet模型结果



(e) 125 000 ms时间约束下搜索到的AlexNet模型结果

Fig. 4 Visualization of the model architecture and SC arithmetic circuit searched by RR-SC

图 4 RR-SC 搜索到的模型架构和 SC 算术电路可视化

来的模型参数也被展示在图 4 中. 对于 LeNet 来说，当 *TC* = 540 ms 时，RR-SC 选择出来的 SC 算术电路是 AND-LOOP，对于 *TC* = 2 700 ms 时，RR-SC 在 AND-ACC 上有更好的表现. 从准确率的角度来说，AND-LOOP 依然能够获得最高的推理准确率，但 RR-SC 在产生高达 3.1 倍的硬件效率的同时，只产生了 1.2% 的准确率丢失. 从推理时间的角度来说，有些 SC 工作在推理时能够满足我们约定的时间约束，但由于缺乏软件和硬件的重配置，使得硬件效率没有被充分发挥. 对比所有的方法，RR-SC 能够产生最高 4.4

Table 7 Comparative Results of SC Relative Work and RR-SC with the Time Constraint 540 ms on LeNet

表 7 LeNet 上 SC 相关工作和 RR-SC 在时间约束为 540 ms 时的比较结果

方法	算术电路	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [13]	XNOR-MUX	64	10.8	3 675.9	×	1 845.5	32 512	1.4 倍
文献 [14]	AND-SEP	64	56.3	555.2	×	1 538.6	38 996	1.2 倍
文献 [15]	uGEMM	64	66.3	2 523.2	×	2 720.9	22 051	2.1 倍
文献 [16]	AND-ACC	64	95.1	547.0	×	1 637.9	36 632	1.2 倍
文献 [16]	AND-LOOP	64	97.6	531.5	√	2 129.5	28 175	1.6 倍
RR-SC（本文）	AND-LOOP	64	97.6	531.5	√	2 129.5	14 088	
		32	95.8	429.5	√	1 275.3	14 114	
		32	95.8	536.9	√	687.4	17 457	

Table 8 Comparative Results of SC Relative Work and RR-SC with the Time Constraint 2 700 ms on LeNet

表 8 LeNet 上 SC 相关工作和 RR-SC 在时间约束为 2 700 ms 时的比较结果

方法	算术电路	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [13]	XNOR-MUX	128	19.5	14 320.7	×	2 961.9	20 257	2.1 倍
文献 [14]	AND-SEP	128	62.5	2 621.5	√	2 550.4	23 525	1.8 倍
文献 [15]	uGEMM	128	74.3	13 008.4	×	6 244.0	9 609	4.4 倍
文献 [16]	AND-ACC	128	96.8	2 687.9	√	2 635.8	22 763	1.8 倍
文献 [16]	AND-LOOP	128	97.2	2 607.6	√	4 432.9	13 535	3.1 倍
RR-SC（本文）	AND-ACC	128	96.8	2 687.9	√	2 635.8	11 381	
		32	95.7	1 147.4	√	1 321.8	13 617	
		32	95.7	1 434.3	√	712.4	16 844	

Table 9 Comparative Results of SC Relative Work and RR-SC with the Time Constraint 21 000 ms on AlexNet

表 9 AlexNet 上 SC 相关工作和 RR-SC 在时间约束为 21 000 ms 时的比较结果

方法	算术电路	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [13]	XNOR-MUX	64	9.8	138 038.7	×	899 459.9	66	3.6 倍
文献 [14]	AND-SEP	64	48.2	20 850.7	√	285 989.9	209	1.1 倍
文献 [15]	uGEMM	64	60.2	94 754.1	×	1 237 486.5	48	4.9 倍
文献 [16]	AND-ACC	64	89.8	20 542.9	√	310 091.0	193	1.2 倍
文献 [16]	AND-LOOP	64	90.1	19 960.4	√	500 412.8	119	2.0 倍
RR-SC（本文）	AND-LOOP	64	90.1	19 960.4	√	500 412.8	59	
		32	87.6	16 131.8	√	223 012.7	80	
		32	87.6	20 164.7	√	120 201.5	99	

Table 10 Comparative Results of SC Relative Work and RR-SC with the Time Constraint 125 000 ms on AlexNet

表 10 AlexNet 上 SC 相关工作和 RR-SC 在时间约束为 125 000 ms 时的比较结果

方法	算术电路	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [13]	XNOR-MUX	128	13.6	682 697.7	×	6 059 624.1	9	4.0 倍
文献 [14]	AND-SEP	128	57.4	124 975.3	√	2 317 931.8	25	1.5 倍
文献 [15]	uGEMM	128	70.3	620 135.8	×	11 001 805.0	5	7.3 倍
文献 [16]	AND-ACC	128	90.0	128 138.5	×	2 631 405.5	22	1.7 倍
文献 [16]	AND-LOOP	128	91.3	124 310.2	√	4 202 999.0	14	2.8 倍
RR-SC（本文）	AND-LOOP	128	91.3	124 310.2	√	4 202 999.0	7	
		64	91.1	93 190.2	√	1 737 447.2	10	
		32	88.7	67 245.5	√	540 599.9	22	

倍的性能提升. 对于 AlexNet 来说，当 TC = 21 000 ms 时，RR-SC 能够产生高达 4.9 倍的硬件效率提升，但比拥有最高准确率的 AND-LOOP 电路只产生了 1.7% 的准确率损失. 类似地，当 TC = 125 000 ms 时，RR-SC 比其他方法最高能达到 7.3 倍的运行数提升，相比 AND-LOOP 电路准确率损失只有 0.9%. 同时，RR-SC 也能够满足设定的时间约束和准确率约束.

除此之外，RR-SC 产生的模型在不同配置中进行切换时产生的开销也非常小. 对于 MLP 来说，模型切换需要约 9 ms；对于 LeNet 来说，模型切换需要约 110 ms；对于 AlexNet 来说，模型切换需要约 4 030 ms. RR-SC 支持实时轻量级的模型切换. MLP，LeNet，AlexNet 每次因 V/F 等级切换而切换数据精度时产生的能耗开销分别为 18 μJ，220 μJ，8.1 mJ.

4.3 硬件和软件重配置方法评估

在本节中，我们对软硬件重配置方式进行比较. MLP，LeNet 和 AlexNet 的配置使用 RR-SC 选择出来的模型结构、算术电路和数据精度进行实验对比. 我们分别对比了无任何重配置、只有硬件重配置(HW)、只有软件重配置(SW)和结合软硬件重配置(RR-SC)4 种方法. 对于所有的方法，MLP 结构设置为 784-200-32-10，LeNet 结构设置为 Conv1(k=3，s=1，c=6)-Pooling1(k=1,s=2)-Conv2(k=7,s=1,c=64)-Pooling2(k=1，s=2)-FC1(32)-FC2(128)-FC3(10)，AlexNet 模型结构设置为 Conv1(k=5，s=1，c=256)-Pooling1(k=1，s=1)-Conv2(k=3,s=2,c=32)-Pooling2(k=1,s=1)-Conv3(k=7，s=2，c=16)-Conv4(k=5，s=2，c=16)- Conv5(k=3，s=1，c=256)-Pooling3(k=1，s=1)-FC1(512)-FC2(1 024)-FC3(10)，均使用在相对低的时间约束下 RR-SC 选择出来的模型结构. 为了公平比较，这 4 种方法均使用相同的 SC 算术电路，即 AND-LOOP. 对于硬件重配置来说，3 个 V/F 等级依然设置为 l_6，l_4，l_3. 类似地，只有软件重配置的设置和只有硬件重配置的其他配置被展示在表 11、表 12 和表 13 中.

Table 11 Experimental Results of Methods with/Without Hardware and Software Reconfiguration on MLP

表 11 MLP 模型上搭配/不搭配软硬件重配置方法的实验结果

方法	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [16]	64	97.8	90.7	√	1 306.7	45 917	2.0 倍
HW	64	97.8	90.7	√	1 306.7	22 959	1.5 倍
			127.0	×	1 008.8	17 842	
			158.7	×	543.7	22 070	
SW	64	97.8	90.7	√	1 306.7	22 959	1.5 倍
	32	96.3	52.3	√	754.3	23 863	
	32	96.3	52.3	√	754.3	15 908	
RR-SC（本文）	64	97.8	90.7	√	1 306.7	22 959	
	32	96.3	73.22	√	582.3	30 912	
	32	96.3	91.5	√	313.8	38 241	

Table 12 Experimental Results of Methods with/Without Hardware and Software Reconfiguration on LeNet

表 12 LeNet 上搭配/不搭配软硬件重配置方法的实验结果

方法	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [16]	64	97.6	531.5	√	2 129.5	28 175	1.6 倍
HW	64	97.6	531.5	√	2 129.5	14 088	1.2 倍
			744.1	×	1 644.0	10 948	
			930.1	×	886.1	13 542	
SW	64	97.6	531.5	√	2 129.5	14 088	1.4 倍
	32	95.8	306.8	√	1 651.9	10 896	
	32	95.8	306.8	√	1 651.9	7 264	
RR-SC（本文）	64	97.6	531.5	√	2 129.5	14 088	
	32	95.8	429.5	√	1 275.3	14 114	
	32	95.8	536.9	√	687.4	17 457	

Table 13 Experimental Results of Methods with/Without Hardware and Software Reconfiguration on AlexNet

表 13 AlexNet 上搭配/不搭配软硬件重配置方法的实验结果

设置	数据精度	准确率/%	推理时间/ms	TC Satisfy	能耗/mJ	推理次数	提升倍数（本文方法/该方法）
文献 [16]	64	90.1	19 960.4	√	500 412.8	119	2.0 倍
HW	64	90.1	19 960.4	√	500 412.8	59	1.5 倍
			27 944.6	×	386 318.7	46	
			34 930.7	×	208 221.8	57	
SW	64	90.1	19 960.4	√	500 412.8	59	1.5 倍
	32	87.6	19 960.4	√	288 876.6	62	
	32	87.6	19 960.4	√	288 876.6	41	
RR-SC（本文）	64	90.1	19 960.4	√	500 412.8	59	
	32	87.6	16 131.8	√	223 012.7	80	
	32	87.6	20 164.7	√	120 102.5	99	

表 11~13 实验结果表明，当模型不搭配任何的软硬件重配置时，虽然其拥有最高的推理准确率，且时间约束能够被满足，但是它能进行的模型推理次数是最少的. 当模型只搭配硬件重配置时，模型推理的时间约束无法被满足，但其对延长电池使用时间有一定的优化. 对于只有软件重配置的实验来说，模型的时间约束都能被满足，但推理准确率有一定的下降，且推理次数也比只搭配硬件和 RR-SC 方法更少. 对于 RR-SC 方法来说，能够在满足时间约束的条件下，最大限度延长电池的使用时间，相比于无任何软硬件配置的实验有接近 2.0 倍的性能提升；对比只有硬件重配置或只有软件重配置的方法来说，能够达到 1.5 倍的性能提升；而且，RR-SC 相比于最高的准确率来说，在 MLP, LeNet 和 AlexNet 中分别只产生了 1.0 个百分点，1.2 个百分点，1.7 个百分点的下降.

5 结　　论

对于电池搭载的边缘设备来说，能源（即电池电量）是有限且动态改变的. 面对这种硬件环境，本文针对基于 SC 的神经网络创新性地提出了硬件和软件重配置相结合的方法以最大限度地延长电池使用时间. 对于不断改变的电池电量来说，本文设置电量阈值以便于在电量降低到一定程度之后进行硬件配置的切换（即切换 V/F 等级）. 同时，为了选择最好的软件配置，本文提出一个运行时可重配置框架 RR-SC，在满足模型推理时间约束和准确率约束的前提下，最大化模型推理准确率和推理总次数. RR-SC 利用强化学习技术能够一次性选择多组软件配置以对应不同的硬件配置. 每组配置能够同时满足规定的时间约束和准确率约束. 同时，多个模型配置在同一个主干网络上进行切换，从而在运行时实现轻量级的软件重配置. 实验结果表明，RR-SC 在给定能源总量的情况下最高可以将模型推理次数增加 7.6 倍，且精度损失相较于最高准确率方法小于 1.7 个百分点. 同时，可以在 110 ms 内进行 LeNet 模型配置的轻量级切换，并且可以满足不同硬件配置下的实时性要求.

作者贡献声明：宋玉红提出论文方案，完成实验和撰写论文；沙行勉提出指导意见和分析方案；诸葛晴凤提出指导意见和讨论方案；许瑞和王寒参与方案讨论和修改论文.

参　考　文　献

[1] Garvey C. A framework for evaluating barriers to the democratization of artificial intelligence [C] //Proc of the 32nd AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2018: 8079−8080

[2] Wang Hanrui, Wu Zhanghao, Liu Zhijian, et al. Hat: Hardware-aware transformers for efficient natural language processing [C] //Proc of the 58th Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: ACL, 2020: 7675−7688

[3] Song Yuhong, Jiang Weiwen, Li Bingbing, et al. Dancing along battery: Enabling transformer with run-time reconfigurability on mobile devices [C] // Proc of the 58th Design Automation Conf. Piscataway, NJ: IEEE, 2021: 1003−1008

[4] Jiang Weiwen, Yang Lei, Dasgupta S. , et al. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start[J]. IEEE Transactions on Computer-Aided Design of

Integrated Circuits and Systems, 2020, 39(11): 4154−4165

[5] Peng Hongwu, Huang Shaoyi, Geng Tong, et al. Accelerating transformer-based deep learning models on FPGAs using column balanced block pruning [C] //Proc of the 22nd Int Symp on Quality Electronic Design (ISQED). Piscataway, NJ: IEEE, 2021: 142−148

[6] Ji Rongrong, Lin Shaohui, Chao Fei, et al. Deep neural network compression and acceleration: A review[J]. Journal of Computer Research and Development, 2018, 55(9): 1871−1888 (in Chinese)
(纪荣嵘, 林绍辉, 晁飞, 等. 深度神经网络压缩与加速综述 [J]. 计算机研究与发展, 2018, 55(9): 1871−1888)

[7] Gong Cheng, Lu Zhi, Dai Surong, et al. Ultra-low loss quantization method for deep neural network compression[J]. Journal of Software, 2021, 32(8): 2391−2407 (in Chinese)
(龚成, 卢冶, 代素蓉, 等. 一种超低损失的深度神经网络量化压缩方法 [J]. 软件学报, 2021, 32(8): 2391−2407)

[8] Meng Ziyao, Gu Xue, Liang Yanchun, et al. Deep neural architecture search: A survey[J]. Journal of Computer Research and Development, 2021, 58(1): 22−33 (in Chinese)
(孟子尧, 谷雪, 梁艳春, 等. 深度神经架构搜索综述 [J]. 计算机研究与发展, 2021, 58(1): 22−33)

[9] Li Hangyu, Wang Nannan, Zhu Mingrui, et al. Recent advances in neural architecture search: A survey[J]. Journal of Software, 2022, 33(1): 129−149 (in Chinese)
(李航宇, 王楠楠, 朱明瑞, 等. 神经结构搜索的研究进展综述 [J]. 软件学报, 2022, 33(1): 129−149)

[10] Gaines B R. Stochastic computing systems [G/OL] // Advances in Information Systems Science. Berlin: Springer, 1969 [2022-11-24]. https://link.springer.com/chapter/10.1007/978-1-4899-5841-9_2

[11] Jeavons P, Cohen D A, Shawe-Taylor J. Generating binary sequences for stochastic computing[J]. IEEE Transactions on Information Theory, 1994, 40(3): 716−720

[12] Qian Weikang, Li Xin, Riedel M D, et al. An architecture for fault-tolerant computation with stochastic logic[J]. IEEE Transactions on Computers, 2010, 60(1): 93−105

[13] Li Peng, Lilja D J, Qian Weikang, et al. Computation on stochastic bit streams digital image processing case studies[J]. IEEE Transactions on Very Large Scale Integration Systems, 2013, 22(3): 449−462

[14] Li Bingzhe, Qin Yaobin, Yuan Bo, et al. Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function [C] //Proc of the 35th IEEE Int Conf on Computer Design (ICCD). Los Alamitos, CA: IEEE Computer Society, 2017: 97−104

[15] Wu Di, Li Jingjie, Yin Ruokai, et al. Ugemm: Unary computing architecture for gemm applications [C] //Proc of the 47th ACM/IEEE Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2020: 377−390

[16] Song Yuhong, Sha E H, Zhuge Qingfeng, et al. Bsc: Block-based stochastic computing to enable accurate and efficient TinyML [C] // Proc of the 27th Asia and South Pacific Design Automation Conf (ASP-DAC). Piscataway, NJ: IEEE, 2022: 314−319

[17] Kim K, Kim J, Yu J, et al. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks [C] // Proc of the 53rd Annual Design Automation Conf (DAC). New York: ACM, 2016: 124: 1−124: 6

[18] Sim H, Kenzhegulov S, Lee J. DPS: Dynamic precision scaling for stochastic computing-based deep neural networks [C] //Proc of the 55th Annual Design Automation Conf (DAC). New York: ACM, 2018: 13: 1−13: 6

[19] Liu Quan, Zhai Jianwei, Zhang Zongzhang, et al. A survey on deep reinforcement learning[J]. Chinese Journal of Computers, 2018, 41(1): 1−27 (in Chinese)
(刘全, 翟建伟, 章宗长, 等. 深度强化学习综述 [J]. 计算机学报, 2018, 41(1): 1−27)

[20] Yu Xian, Li Zhenyu, Sun Sheng, et al. Adaptive virtual machine consolidation method based on deep reinforcement learning[J]. Journal of Computer Research and Development, 2021, 58(12): 2783−2797 (in Chinese)
(余显, 李振宇, 孙胜, 等. 基于深度强化学习的自适应虚拟机整合方法 [J]. 计算机研究与发展, 2021, 58(12): 2783−2797)

[21] Sim H, Lee J. A new stochastic computing multiplier with application to deep convolutional neural networks [C] //Proc of the 54th Annual Design Automation Conf (DAC). New York: ACM, 2017: 29: 1−29: 6

[22] Tomic T, Schmid K, Lutz P, et al. Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue[J]. IEEE Robotics & Automation Magazine, 2012, 19(3): 46−56

[23] Horowitz M, Indermaur T, Gonzalez R. Low-power digital design [C] // Proc of 1994 IEEE Symp on Low Power Electronics. Piscataway, NJ: IEEE, 1994: 8−11

[24] Jiang Weiwen, Zhang Xinyi, Sha E H, et al. Accuracy vs. Efficiency: Achieving both through FPGA-implementation aware neural architecture search [C/OL] // Proc of the 56th Annual Design Automation Conf (DAC). New York: ACM, 2019[2022-11-24]. https://dl.acm.org/doi/abs/10.1145/3316781.3317757

[25] Williams R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning[J]. Machine Learning, 1992, 8(3): 229−256

[26] Dong Xuanyi, Yang Yi. Nas-bench-201: Extending the scope of re-producible neural architecture search [C/OL] // Proc of the 8th Int Conf on Learning Representations (ICLR). 2020[2022-11-24]. https://openreview.net/forum?id=HJxyZkBKDr

[27] Skadron K, Stan M, Huang Wei, et al. Temperature-aware microarchitecture [C] // Proc of the 30th Int Symp on Computer Architecture (ISCA). Los Alamitos, CA: IEEE Computer Society, 2003: 2−13

[28] Google. Odroid-XU3 [EB/OL]. 2020[2022-11-24].https://www.hardkernel.com/shop/odroid-xu3/, 2020.

[29] Liu Siting, Han jie. Energy efficient stochastic computing with sobol sequences [C] // Proc of the 20th Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2017: 650−653

[30] Najafi M H, Lilja D J, Riedel M. Deterministic methods for stochastic

computing using low-discrepancy sequences［C/OL］// Proc of the 37th IEEE/ACM Int Conf on Computer-Aided Design (ICCAD). New York: ACM, 2018［2022-11-24］.https://dl.acm.org/doi/abs/10.1145/3240765.3240797



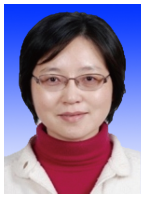
Song Yuhong, born in 1997. PhD candidate. Student member of CCF. Her main research interests include embedded system, software-hardware co-design, automated machine learning, stochastic computing, and quantum computing.

宋玉红, 1997 年生. 博士研究生. CCF 学生会员. 主要研究方向为嵌入式系统、软硬件协同设计、自动机器学习、随机计算、量子计算.



Edwin Hsing-Mean Sha, born in 1964. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include big data systems, high-performance intelligent computing, advanced storage, parallel/distributed systems and computing, embedded systems, scheduling and optimization, resource allocation, and quantum computing.

沙行勉, 1964 年生. 博士, 教授, 博士生导师. CCF 高级会员. 主要研究方向为大数据系统、高性能智能计算、先进存储、并行/分布式系统和计算、嵌入式系统、调度及优化、资源分配、量子计算.



Zhuge Qingfeng, born in 1970. PhD, professor, PhD supervisor. Member of CCF. Her main research interests include parallel architectures, embedded systems, supply-chain management, real-time systems, optimization algorithms, compilers, and scheduling.

诸葛晴凤, 1970 年生. 博士, 教授, 博士生导师. CCF 会员. 主要研究方向为并行架构、嵌入式系统、供应链管理、实时系统、优化算法、编译器、调度.



Xu Rui, born in 1996. PhD candidate. Student member of CCF. Her main research interests include non-volatile memory, optimization algorithms, and computer architecture.

许　瑞, 1996 年生. 博士研究生. CCF 学生会员. 主要研究方向为非易失性存储器、优化算法、计算机架构.



Wang Han, born in 1997. PhD candidate. Her main research interests include storage systems, embedded systems, optimization algorithms, machine learning, and adaptive learning systems.

王　寒, 1997 年生. 博士研究生. 主要研究方向为存储系统、嵌入式系统、优化算法、机器学习、自适应学习系统.