TinyReptile：基于联邦元学习的 TinyML

1st Haoyu Ren西门子股份公司，慕尼黑工业大学德国慕尼黑
haoyu.ren@siemens.com

2nd Darko Anicic西门子股份公司德国慕尼黑
darko.anicic@siemens.com

3rd Thomas A. Runkler西门子股份公司，慕尼黑工业大学德国慕尼黑
thomas.runkler@siemens.com

arXiv:2304.05201v1 [cs.LG] 11 Apr 2023

摘要——**微型机器学习（TinyML）是一个快速发展的领域，旨在让资源受限的微控制器（MCU）也能普及机器学习（ML）。鉴于这类微型设备无处不在，自然会让人思考：TinyML 应用能否通过汇聚各自的知识而获益。联邦学习（FL）使分布式智能体能够在不共享敏感本地数据的情况下，共同学习一个全局模型。然而，由于实际部署环境复杂，且各设备可用的数据存在异质性，通用的全局模型未必适用于所有设备。此外，TinyML 硬件的部署受到显著的计算和通信限制，而传统机器学习无法应对这些限制。针对这些挑战，我们提出了 TinyReptile，这是一种受元学习和在线学习启发的简单而高效的算法，可在微型设备之间协同学习神经网络（NN）的稳健初始化，使其能够根据新设备的数据快速适配。我们在 Raspberry Pi 4 和仅配备 256 KB RAM 的 Cortex-M4 微控制器上展示了 TinyReptile。针对多种 TinyML 应用场景的评估表明，与性能相当的基线算法相比，该方法至少可将资源消耗和训练时间分别降低至原来的二分之一。**

索引词—**微型机器学习、元学习、联邦学习、在线学习、神经网络、物联网、微控制器。**

I. 引言

过去十年间，机器学习（机器学习）应用的发展一直受到大数据的出现和处理能力提升的推动。因此，为了获得更好的性能，人们开发了大规模人工智能（AI）模型，这些模型需要海量的计算资源并消耗大量电力。例如，微软和 NVIDIA 推出了一种名为“Megatron-TuringNLG”的机器学习模型 [1]，该模型拥有5300亿个神经元，训练时约需消耗5GWh 的电力。业界日益意识到，部署先进的机器学习模型成本高昂且难以持续。

鉴于这些问题，微型机器学习（TinyML）在机器学习与嵌入式系统的交汇处日益受到关注，推动数据处理从数据中心转向越来越小的物联网单元。这些低成本设备通常由电池供电，旨在利用有限的资源（例如 256 KB RAM 和 64 MHz CPU）长时间运行。TinyML 可在靠近事件发生地点的物联网设备上直接实现近乎实时的边缘人工智能。通过避免向云端传输数据，TinyML 在数据隐私、延迟和能耗方面带来优势。效率。据估计，目前有超过 2500 亿台嵌入式设备正在使用中，且需求持续增长，尤其是在工业领域 [1]。鉴于生产环境中部署了大量物联网设备，一个重要的研究方向是共享并整合从各个微型设备中学到的经验，以提升 TinyML 应用的性能和鲁棒性。

联邦学习（联邦学习）提供了一种在分布式代理/设备上学习全局模型的方法。联邦学习在设计上能够保护本地数据隐私，因为只有模型更新会被合并到全局模型中，原始数据从不会被复制到云端。然而，物联网设备分布在各个场域中，现实世界环境复杂且不断变化，使得本地数据通常不会在各设备之间保持一致。即使是相邻的两台设备，其环境也可能存在显著差异。此外，每台设备可能对机器学习任务有自己的需求，例如关注不同的输出类别。因此，通过联邦学习训练得到的通用共享模型应用于某台本地设备时，性能可能会差到任意程度。此外，大多数联邦学习算法都设计为运行在功能强大的机器上，并未考虑计算资源限制。它们的通信方案会按照精心安排且一致的计划，定期向中央服务器传输权重更新。所有这些对于许多 TinyML 应用来说都不可行，因为物联网设备在通信、内存和计算方面都受到高度的资源限制。不稳定的网络连接可能会使通信长时间中断，而设备端资源不足也可能妨碍这些算法的运行。

为应对这些挑战，我们提出了 TinyReptile。这是对著名的 Reptile算法 [2],的一种改进形式，通过利用在线学习，在微型设备之间的联邦环境中执行模型无关元学习。元学习旨在找到一个初始共享模型，使当前或新加入的智能体只需利用少量本地数据执行几步梯度下降，就能快速适应其环境。换句话说，元学习就是学习如何学习。我们的 TinyReptile算法基于在线学习的理念；我们设计该算法，以便在考虑资源限制的同时，以分布式方式实现增量式设备端学习。借助在线学习，我们会在数据到达时将模型应用于这些数据。

[1]https://venturebeat.com/ai/why-tinyml-is-a-giant-opportunity/

以流式方式处理而不存储历史数据，这与许多工业应用中的实际情况相近。TinyReptile 还保留了串行架构的优势：设备可以依次与中央服务器通信，无需依赖持续且并发的连接。此外，它有助于为每台设备定制更适合的模型。

通过将 TinyReptile 与 FedAVG 和 FedSGD 进行比较，我们展示了当设备的本地数据存在异质性时，传统联邦学习架构可能会失效。我们在两个已有的元学习数据集和一个构建的数据集上研究 TinyReptile，分别是用于回归问题的“正弦波示例”、用于图像分类任务的“Omniglot”，以及我们构建的用于音频分类任务的“关键词识别”。由于大多数元学习基准数据集都是为图像分类而设计的，我们贡献了“关键词识别”数据集，用于评估元学习在音频分类问题上的表现；该数据集基于广受欢迎的 TinyML 数据集“Speech Commands”构建。我们在树莓派和 Arduino 微控制器（微控制器）上，使用这些数据集将 TinyReptile 与基线算法 Reptile 进行了比较。评估结果表明，与基线 Reptile 算法相比，TinyReptile 在实现相当性能的同时，能够将计算资源和训练时间至少减少一半。为指导 TinyReptile 在微型硬件上的部署，我们在各种设置下对 TinyReptile 中的超参数进行了表征，并在附录 A 中展示了结果。

本文其余部分将在第二节讨论有关 TinyML、在线学习、联邦学习和元学习的相关工作。第三节介绍 TinyReptile 及其方法。第四节描述基准数据集和实验设置，并分析结果。最后，第五节总结全文并讨论未来工作。

II. 相关工作

TinyML： 近年来，我们见证了 TinyML 在研究和工业领域取得的显著进展。综述 [6] [7]直观地概述了 TinyML 的不同发展方向，以及面临的挑战和机遇。具体而言，相关研究可分为硬件、软件和应用三个主要方面。在硬件方面，人们提出了模拟存内计算等新技术 [8]，以优化资源受限设备的处理能力。深度学习加速器 [9] 和智能传感器 [10] 专为高效执行人工智能工作负载而设计。软件方面的进展主要体现在压缩方法 [11], 、管理系统 [12], 和高效算法 [13]的开发上。然而，大多数现有解决方案（如面向微控制器的 TensorFlow Lite [14], ）只支持部署静态机器学习模型以进行推理，因此设备无法从实际环境中学习最新知识。只有少数研究 [15] [16]探讨了在微型设备上训练模型的可能性。尽管如此，TinyML 的发展已开始造福社会，例如应用于预测性维护 [17] 和医疗保健 [18]。

联邦学习： 联邦学习聚焦于在中央服务器的协调下联合训练全局模型，同时保持训练数据分散存储。联邦学习研究始于2016 [3], ，当时提出了两种黄金标准基线算法“FedAVG”和“FedSGD”。此后，相关研究不断在多个领域推进：优化算法 [19], 、模型更新压缩 [20], 、差分隐私 [21], 和鲁棒性 [22]。一些联邦学习研究已在树莓派等能力较低的边缘设备上开展 [23]。然而，在 TinyML 背景下，将联邦学习应用于资源高度受限设备的研究仍然很少。目前最先进的研究仅限于两项相关工作 [24] [25]。这些工作的实验要么通过仿真实现，要么在小规模受控环境中开展，无法反映真实世界的情况。例如，联邦学习会受到客户端异质性的影响，进而导致学习进展缓慢且不尽如人意。

元学习： 元学习旨在利用小型数据集实现快速本地适应，因此成为TinyML场景的理想选择。元学习与迁移学习相似。在迁移学习中，可以先对全局模型进行预训练，之后再针对特定的小型数据集进行微调，但无法保证能够学习到有利于泛化的良好初始化 [26]。相反，元学习训练一个通用模型，其明确目标是使其易于微调。元学习领域最早的进展被称为“模型无关元学习”（MAML） [27], ，它通过基于梯度的优化使这一概念得到发展。随后，出现了多篇研究其收敛行为和实证性质的论文 [28] [29]。然而，MAML 需要计算高阶导数，使其实现复杂且计算成本高昂。为缓解这一问题，研究者提出了一些采用更简单操作的轻量级变体，包括一阶MAML 和 Reptile [2]。此外，元学习也已在联邦 [30] [31] 和在线学习场景 [32] [33]中得到研究。然而，据我们所知，尚无相关工作专注于将元学习应用于受限设备。

在线学习： 机器学习系统有两个任务：推理和训练。在线机器学习是指在线执行这些任务，也就是按顺序处理数据，而不重新访问之前的样本，从而节省内存资源，并尽可能不牺牲模型性能。然而，与批量/离线学习相比，在线学习 [34] [35] 受到的关注较少，因为大多数机器学习工程师往往认为，设备及其数据始终可以作为一个批次获取。River 是一个广受欢迎的在线机器学习 Python 库 [36]。我们认为，在线学习非常适用于 TinyML 应用。这一理念有助于我们开发内存占用更低的算法，并使模型在不断变化的环境中保持最新。已有不少有趣的研究致力于这一领域 [37] [38]。在本研究中，我们将在线学习融入元学习，从而在联邦场景下实现受限设备上的元学习。

III. 方法

元学习的目标是教会神经网络（NN）如何仅凭少量训练样本，就能快速适应或外推到新任务（环境）。我们希望通过在异构且资源受限的物联网设备之间利用联邦学习来实现这一目标；这些设备在训练期间接触到的带标签样本都很有限。本节将定义问题设置，并介绍TinyReptile 的形式化表述。

A. 元学习

首先，我们描述元学习的优化问题。这里我们考虑一个独立任务的分布T，这些任务具有相似性，例如，所有任务都使用相同结构的神经网络进行图像分类。然而，每个任务都有不同的目标输出，例如，一个任务是对“狗与猫”进行分类，另一个任务是对“苹果与梨”进行分类。我们的目标是为神经网络找到良好的初始权重 ϕ，使其在获得从分布 T 中采样且在训练期间从未遇到过的新任务 t 时能够快速学习，例如对“自行车与摩托车”进行分类。

我们将任务划分为训练任务 $T_{training}$ 和测试任务 $T_{testing}$。元学习算法 F 在 $T_{training}$ 上训练神经网络，并使用 $T_{testing}$ 评估训练后模型的质量。每个任务 t 都有一个数据集 D，该数据集被划分为两部分：用于任务内训练的支持集 S，以及用于任务内测试的查询集 Q。$D = \langle S, Q \rangle$ 通常包含数量有限的样本。为了评估经过训练的初始化，我们首先需要在所有 $T_{testing}$上使用支持集 S 对模型权重 ϕ 进行微调，每个任务分别执行 k 步。然后，$\hat{}$ 微调后的新权重 ϕ 将在每个测试任务对应的查询集 Q 上进行测试，最后对所有 $T_{testing}$ 的结果求平均。由于元学习旨在寻找能够快速学习的模型，因此训练步骤 k 通常定义为一个较小的数值。元学习算法 F 的目标是最小化各任务上的损失函数：

$$L(\phi) = \sum_{n=1}^{N} l_n(\hat{\phi}_n^k), \tag{1}$$

其中，$\hat{\phi}_n^k$ 表示在任务 n的支持集上基于初始权重 ϕ 微调 k 步后的模型权重，l_n n N ($\hat{\phi}_n^k$) 表示任务 n查询集上的损失，l_n n N 表示任务数量。

元学习的目标与迁移学习有所不同：元学习旨在找到 ϕ，使其在每个新任务上经过快速微调后都能取得良好表现。换言之，元学习追求的是模型的潜在性能。另一方面，迁移学习的损失函数定义如下：

$$L(\phi) = \sum_{n=1}^{N} l_n(\phi_n), \tag{2}$$

这表明，迁移学习旨在找到一个无需微调即可适用于所有任务的良好初始化。也就是说，迁移学习寻求模型的当前性能。

B. TinyReptile

我们提出了一种用于 TinyML 的联邦元学习算法，称为TinyReptile。TinyReptile 利用跨异构物联网设备的联邦学习，为神经网络学习一种初始化。当我们仅基于少量带标签的本地示例针对新的机器学习问题优化初始化时，本地微调应当能够快速完成。TinyReptile是专为 TinyML 定制的Reptile 在线学习版本。借助在线学习的特性，TinyReptile 会按照数据到达的顺序对其进行处理，从而节省计算资源，并使推理和训练能够直接在受限的物联网设备上进行。

值得注意的是，在元学习中，存在包含各自数据集的任务；而在 TinyReptile 中，我们拥有具备各自数据集或用于收集数据的传感器的设备。基于联邦学习的理念，TinyReptile 通过将每个设备上的机器学习问题表示为元学习中的训练任务，最小化损失函数 1 的一个版本。TinyReptile 提升了初始模型在分布式设备中所有任务上的适应能力T：

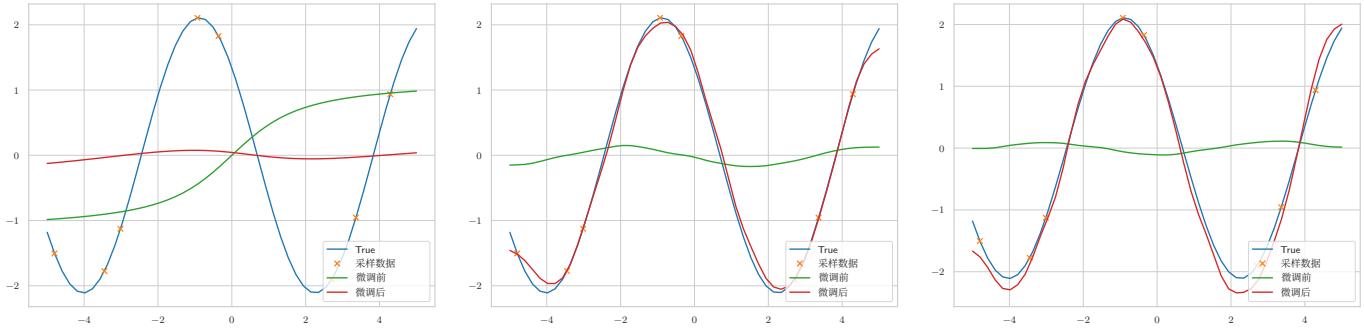
$$\underset{\phi}{minimize}\, \mathbb{E}_t[\frac{1}{2} L(\hat{\phi}_t^k, \phi_t^*)]^2, \tag{3}$$

其中，ϕ 表示初始化，L 表示设备 t的微调后的权重$\hat{}$ ϕ_t^k 与最优权重 ϕ_t^* 之间的距离。TinyReptile算法定义如下（我们以橙色突出显示TinyReptile算法与Reptile算法之间的主要差异）：

算法 1: TinyReptile

1: 中心模型权重 ϕ;
2: 中央服务器学习率 α;
3: 客户端 $T = \langle T_{training}, T_{testing} \rangle$，每个客户端都拥有本地流式数据 D。对于评估，D 在 $T_{testing}$ 是划分为支持集 (S) 和查询集 (Q)，$D = \langle S, Q \rangle$;
4: 客户端学习率 β;
5: **对**轮 = 1 , 2, … **执行**
6: 采样 一个 客户端 t 来自 $T_{training}$ 关于损失 L_t;
7: 发送 ϕ 给客户端 t;
8: **对于**每个 x 在本地流式支持集 S_t **执行**
9: 计算 $\hat{\phi}_t^{k+1} = SGD(\hat{\phi}_t^k, x, \beta, L_t)$，记作一客户端上的一步 SGD;
10: **结束循环**
11: Send $\hat{\phi}_t$ 返回服务器;
12: 更新中心模型权重：
$$\hat{\phi} \leftarrow \phi + \alpha(\phi_t - \phi)$$
13: （可选）在 $T_{testing}$ 上评估 ϕ ;
14: **结束循环**

通过在线学习，TinyReptile能够进行增量学习。如上所述，我们的算法是Reptile算法 [5]的一种改进形式。我们的方法与Reptile之间的关键区别在于，Reptile在每个客户端上进行批量训练，其中神经网络在整个支持集上进行训练



（a）迁移学习　（b）Reptile。　（c）TinyReptile（我们的方法）。

图1：迁移学习、Reptile和 TinyReptile在正弦波回归示例中的演示，其中我们在测试客户端的支持集（八个采样点）上对训练好的神经网络进行微调，执行八次本地 SGD 更新。该模型由四个全连接层组成： $1 \to 32 \to 32 \to 1$。这表明，使用 Reptile和 TinyReptile训练的神经网络可以快速收敛到采样的正弦波，并推导出采样点之外的值，而使用迁移学习则很难做到这一点。

以批处理方式进行，训练数据会被堆叠并重复使用，因此需要消耗大量资源。我们的方法以流式处理方式每次处理一个数据点，并且每次更新后都可以丢弃该数据，因此具有很高的资源效率。在每一轮中，TinyReptile 会针对采样训练客户端支持集 S 中的流数据依次执行随机梯度下降（SGD），例如物联网设备的实时传感器数据。随后，将初始中心权重朝着 SGD 更新后得到的权重方向更新。接下来，如果需要评估，测试客户端可以评估训练后的初始化。通过这种方式，TinyReptile 尝试将初始权重移向一个最优点，使其尽可能接近所有任务/设备。因此，TinyReptile 学习到一种神经网络初始化，可在新任务/设备上快速微调。换言之，TinyReptile 优化的是泛化能力。感兴趣的读者可以在 Reptile 论文 [2]中找到更多理论分析。TinyReptile 的主要优势包括：

- 稳健性：TinyReptile 采用串行通信方案。许多联邦学习算法采用并行通信方案，要求所有客户端在训练期间都与中央服务器保持稳定连接，并且所有设备的处理能力必须相近，这在许多现实世界的生产环境中并不现实。网络中的干扰或客户端中的一些“掉队者”可能会影响甚至中断运行。相比之下，TinyReptile 在每个轮次评估一个设备，从而有助于学习过程的稳定性，尤其是在TinyML 环境中，因为许多物联网设备部署在网络覆盖有限的偏远地区。

- 资源高效：设备端学习在资源受限的设备上很困难，因为它们通常没有足够的资源来存储和处理大量数据。通过在线学习，数据以流的形式到达，并在 TinyReptile中逐个进行处理。每次学习迭代后，都可以有效地丢弃数据。在任何时刻，内存中只保存一个数据样本，而且我们不需要存储历史数据。与其他离线/批量学习算法相比，TinyReptile 所需的资源显著更少。

- 可扩展：TinyReptile 的资源效率使其核心能够兼容各种设备，从高性能设备到微型设备。由于其串行通信架构，任何运行TinyReptile的设备都可以随时加入或退出学习过程，使该算法在生产环境中具备可扩展性和可靠性。

表I：实验中使用的模型概览。

	模型类型	Size	参数
正弦波示例	全连接	19.4 KB	1153
关键词识别（4 类别）	卷积神经网络	95.7 KB	19812
Omniglot（5 个类别）	卷积神经网络	485.1 KB	113733

IV. 实验与评估

本节首先简要介绍两个热门的元学习数据集：正弦波回归问题和 Omniglot图像分类问题。接着，我们介绍用于元学习的音频分类数据集“关键词识别”，并说明使用该数据集的动机。随后，我们以正弦波示例为案例研究，直观地概述元学习，并展示传统联邦学习算法在元学习场景中会失效。之后，我们通过分析 TinyReptile 和 Reptile 在 Raspberry Pi 4 Model 4 [2] 和 Arduino Nano BLE 33 微控制器 [3]上的实际表现，对比二者在收敛速度、内存需求和时间消耗方面的差异。实验中，我们使用 MLPerf Tiny基准测试 [39] 中介绍的神经网络，以确保结果一致且具有可比性。这些神经网络概述见表 I。最后，为了便于在 TinyML设备上部署，我们在附录 A 中研究了 TinyReptile 中若干超参数的影响。

[2]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ [3] https://docs.arduino.cc/hardware/nano-33-ble-sense

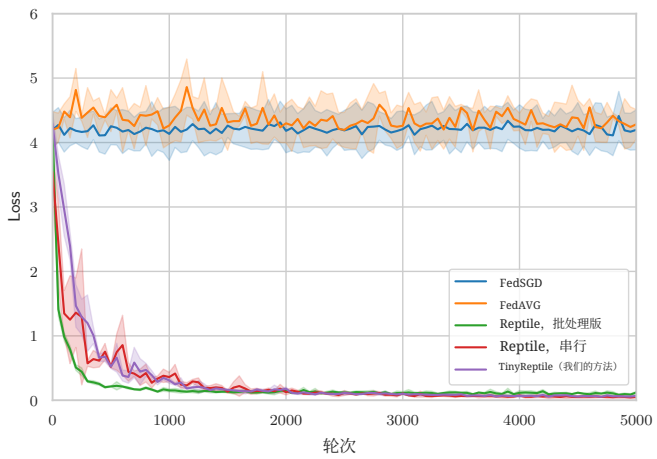


图2：FedSGD、FedAVG、Reptile（批量和串行）以及 TinyReptile 在正弦波回归示例上的训练收敛情况。这表明，我们的 TinyReptile 能够达到与 Reptile 相当的性能。然而，在元学习场景中，传统联邦学习算法（如 FedSGD 和 FedAVG）难以学习到有意义的知识。

我们对每个实验进行三次，并将结果绘制为测量值的均值 ± 标准差。

A. 元学习数据集

正弦波：正弦波回归问题来自 [27]，，定义如下：每个客户端/设备都有一个拟合正弦函数 $f(x) = a\sin(b\,x + c)$ 的底层任务，其中参数（a, b, c）是随机选取的。客户端的目标是协作学习一组良好的神经网络初始权重，使其仅凭少量采样的（x_t, y_t）对，就能快速泛化到新的正弦函数$f_t(x) = a_t \sin(b_t\, x + c_t)$。

Omniglot：Omniglot 图像数据集包含来自 50 种不同字母表的 $C = 1623$ 个字符，每个字符有 20 个样本。元学习问题定义如下：每个客户端 t 都有一个分类任务，该任务包含从 C中采样的 M 个字符（类别）。虽然所有客户端的类别数量相同，但每个客户端分类任务中的字符都是随机选取的。例如，如果 $M = 5$，所有客户端都会使用标签 $0{-}4$ 训练五个类别，但每个客户端的分类字符各不相同。训练目标与上文的正弦波问题类似：学习良好的初始权重，使其仅凭少量数据示例，就能快速适应新的分类问题。

关键词识别：“关键词识别”音频数据集是对“语音命令”数据集的一种改编 [5]，该数据集包含 35 个独立单词，每个单词的样本数超过 1,000。由于该数据集的元学习设置与上文所述的“Omniglot”分类问题相似，只是分类目标不同，因此我们不再赘述。相反，我们将详细阐述提出该数据集的理由。TinyML 是一个新兴领域，它将机器学习的强大能力引入嵌入式系统，其潜在应用十分广泛，从手势检测到语音识别皆有涵盖。不过，我们发现，大多数成熟的元学习数据集都归类为图像分类，无法充分覆盖 TinyML 的应用需求。因此，我们希望提供一个新数据集，用于评估元学习在其他 TinyML 应用场景中的表现。尽管“语音命令”数据集最初是为另一个类别数量有限的机器学习问题构建的，但我们相信，经过改编的“关键词识别”数据集仍能让社区受益。

B. 正弦波示例——案例研究

我们首先通过正弦波回归问题展示元学习，并比较迁移学习、Reptile 和我们的 TinyReptile 的表现，如图1所示。结果表明，在元学习环境中，迁移学习无法学习出有意义的初始化。这是因为迁移学习旨在训练一个能够同时处理所有客户端上全部正弦函数的神经网络，这意味着理想情况下，训练好的模型将返回所有任务的 $f(x)$ 平均值。在正弦波示例中，由于每个任务中的随机参数（a、 b、 c），正弦函数 $\mathbb{E}_t[f_t(x)]$ 对所有 x 值的平均值都近似为零$f(x) = a\sin(b\,x + c)$。

接下来，我们比较 FedSGD、FedAVG、Reptile（串行版本和批处理版本）以及 TinyReptile 的训练进展。Reptile 的批处理版本与串行版本之间的区别在于，批处理版本要求服务器同时与多个客户端建立连接，以便并行训练模型。相比之下，在串行版本中，任意时刻只有一个客户端连接到服务器。如图2所示，结果符合我们的预期：当本地数据表现出异质性时，FedSGD 和 FedAVG 无法学习出合适的初始化，因为它们的学习目标与迁移学习相似。然而，我们的 TinyReptile 可以取得与 Reptile 算法相当的结果。尽管 Reptile 的批处理版本在训练早期阶段收敛更快，但它需要更多的计算资源，并且需要与许多客户端建立并发网络连接，这不适用于大多数 TinyML 场景。

C. 评估结果

我们评估了 Reptile 和 TinyReptile 在 Raspberry Pi 4 和 Arduino Nano BLE 33 微控制器上的性能。我们不考虑其他联邦学习算法，因为它们不适用于元学习问题。如前所述，我们在实验中考虑的神经网络见表I。我们尝试了以下不同超参数的取值范围，并找到了适用于这些问题的可行组合：客户端学习率 β (0.001–0.02)、训练步数 k (1–32) 和支持集大小 S (1–32)。不过，我们没有对超参数进行微调以优化最终结果。

内存需求：为了在硬件上运行这些算法，我们需要测量所考虑的神经网络的内存使用量。显著结果见表 II。由于

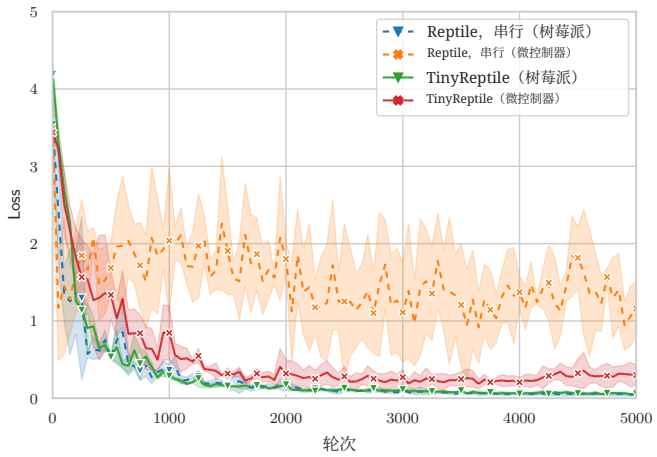


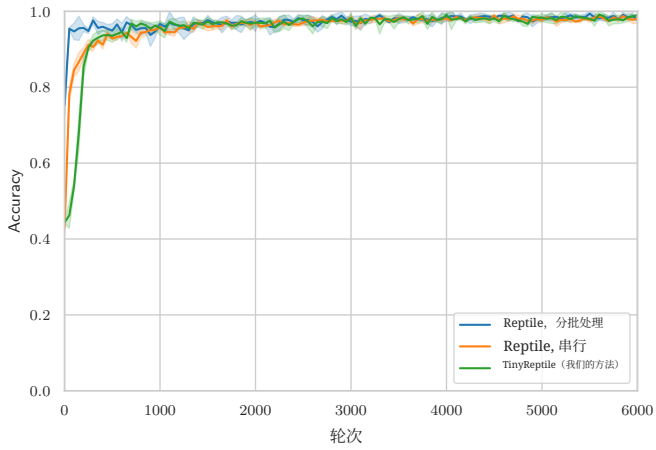
图 3：Reptile（串行）和 TinyReptile在 Raspberry Pi 4和 Arduino Nano33 BLE Sense 微控制器上的正弦波回归示例中的训练收敛情况。这表明TinyReptile可以达到与 Reptile 相当的结果。微控制器上的整体性能略差，尤其是使用 Reptile 时，这可能是由于微控制器的数值精度有限所致。

表II：内存需求对比（结果是在支持集大小为 $S = 32$ 的条件下测量的）。

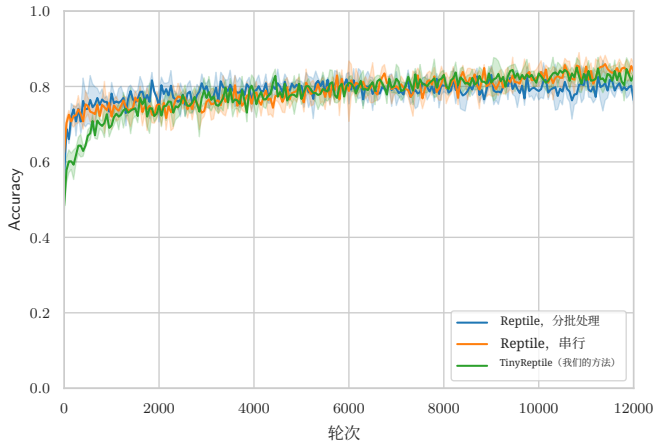
	Reptile（批处理和串行）	TinyReptile（本文方法）
正弦波示例	10.7 KB	4.8 KB
关键词识别（4 类别）	5816.4 千字节	437.2 KB
Omniglot（5 个类别）	3778.1 KB	667.2 KB

在线学习方面，与 Reptile 相比，TinyReptile 在所有任务中的内存消耗至少减少了一半。结果还表明，由于这款微控制器仅配备 256 KB 内存，因此只能使用正弦波示例在 Arduino Nano 33 BLE Sense 开发板上进行实验。

训练收敛：在图 3 中，我们展示了 Reptile（串行）和 TinyReptile 在正弦波示例上的收敛性能，并展示了在 Arduino 微控制器上运行 TinyReptile 的可行性。Arduino Nano 33 BLE Sense 微控制器配备了 64 MHz Cortex-M4 CPU、256 KB 内存和 1 MB 闪存，其规格处于TinyML设备的典型范围内。实验中，我们选择了以下超参数：$S_{training} = 32$、$\beta = 0.01$，以及本地训练轮数 $E = 8$（用于使用 Reptile 进行训练）。我们观察到，与 Raspberry Pi 4 相比，Arduino 微控制器上的收敛速度更慢、波动更多，结果也略差，这可能是由于硬件的数值精度有限所致。尽管如此，只要训练轮次足够多，模型仍能在微控制器上取得合理的性能。尤其是，Reptile（串行）在微控制器上的表现不如 TinyReptile。我们的解释是，由于数值精度有限



(a) Omniglot，五个输出类别。



(b) 关键词识别，四个输出类别。

图 4：Reptile（批处理和串行）与 TinyReptile 在 Raspberry Pi 4 上使用 Omniglot 和关键词识别数据集进行训练时的收敛情况。结果表明，只要训练轮次足够，TinyReptile 就能达到与 Reptile 相当的性能。

由于微控制器的精度有限，对批量数据计算出的梯度可能会相互抵消，使本就不够精确的梯度变得更加不精确。

由于 Arduino 开发板受限，无法用于 Omniglot 和关键词识别任务，因此我们仅在 Raspberry Pi 4 上比较 Reptile 和 TinyReptile 在这两个数据集上的训练进度，如图 4 所示。在实验中，我们选择以下超参数：$S_{training} = 16$、$\beta = 0.002$、本地轮次 $E = 8$（使用 Reptile 训练时），以及每轮采样的客户端数量 $T = 32$（使用 Reptile 的批处理版本训练时）。结果证实，TinyReptile 需要稍多一些训练轮次才能达到与 Reptile 相同的准确率，这是合理的，因为 TinyReptile 一次只基于一个数据点进行更新，而 Reptile 则以批次形式在整个支持集上训练。

时间消耗：我们研究了 Reptile 和 TinyReptile 的训练时间消耗，以展示其效

表III：Arduino Nano 33 BLE Sense 上正弦波示例中单次训练轮次的时间消耗对比（结果是在支持集大小为 $S = 32$的条件下测得）。

	发送	本地训练	接收	总计
Reptile	1.58 秒	8.32 秒	1.65 秒	11.55 秒
TinyReptile（我们的方法）	1.58 秒	0.44 秒	1.65 秒	3.67 s

表 IV：在 Raspberry Pi 4 上进行一轮训练的耗时比较（结果是在支持集大小为 $S = 32$ 的条件下测得）。

	Reptile（批处理与串行）	TinyReptile（我们的方法）
正弦波示例	0.56 秒	0.24 秒
关键词识别（4 类）	11.96 秒	3.45 秒
Omniglot (5 类别)	22.53 秒	10.11 秒

率。表 III 和表 IV 的结果表明，在所有任务中，TinyReptile 的训练速度都优于 Reptile。尤其是在资源受限的 Arduino 开发板上，本地训练时间缩短了 16 倍，这表明 TinyReptile 是 TinyML 的理想选择。

最后，我们在附录 A中为感兴趣的读者提供了对 TinyReptile 中不同超参数的分析，以支持 TinyReptile 的轻松部署。

V. 结论与未来工作

Reddi 教授和 Pete Warden 认为，机器学习的未来前景光明，而且会越来越小巧 [4]。元学习对 TinyML 应用很有吸引力，因为由于部署环境各不相同，我们往往需要为每台 TinyML 设备定制机器学习解决方案，而且微控制器的数量还在不断增加。然而，这些受限的物联网设备通常是没有操作系统的裸机设备，因此在设备上进行学习仍是一项尚待解决的挑战。大多数 TinyML 解决方案仅支持在这些设备上进行推理。本研究提出了 TinyReptile，这是一种基于 Reptile算法修改、专为 TinyML 场景设计的轻量级元学习算法。借鉴联邦学习和在线学习的概念，TinyReptile 能够利用资源有限的分布式物联网设备协同学习稳健的神经网络初始化，之后可快速适配到新的设备或环境。我们的贡献如下：

- 据我们所知，我们提出的 TinyReptile 是首次尝试在资源受限设备上实现元学习。
- 我们介绍了音频分类数据集“关键词识别”，该数据集源自“语音命令”数据集，为评估 TinyML 场景下的元学习提供了一个新的分类问题。
- 我们评估 TinyReptile在各种元学习任务中的表现，将其与其他方法进行比较，并测量其实际-在 Raspberry Pi 4 和 Arduino Nano BLE 33 Sense 微控制器上的性能。结果证实，TinyReptile 在至少节省 50% 资源消耗和训练时间的同时，能够达到与基准算法相当的性能。
- 我们研究了 TinyReptile 中若干超参数的影响，如附录 A 所示，以指导其在现实世界中的部署。

目前已经开展的未来工作包括进一步改进算法，例如应用学习率退火技术，通过微调超参数为现实世界部署提供最佳实践，在其他元学习任务上进行基准测试，将该算法与其他最先进的方法进行比较，将评估扩展到其他微型设备，将该算法应用于大规模工业应用场景，以及研究微控制器降低数值精度所产生的影响。

[4]https://pll.harvard.edu/course/future-ml-tiny-and-bright?delta=0

致谢

本项工作部分得到了 NEPHELE 项目（ID: 101070487）的支持，该项目在欧洲地平线计划“面向边缘的未来欧洲平台：元操作系统”主题下获得了资助。

参考文献

[1] S. Smith、M. Patwary、B. Norick、P. LeGresley、S. Rajbhandari、J. Casper、Z. Liu、S. Prabhumoye、G. Zerveas、V. Korthikanti、E. Zhang、R. Child、R. Y. Aminabadi、J. Bernauer、X. Song、M. Shoeybi、Y. He、M. Houston、S. Tiwary 和 B. Catanzaro，《使用 deepspeed 和 megatron 训练 megatron-turing nlg 530b：一种大规模生成式语言模型》，2022年。 [在线]。可访问：https://arxiv.org/abs/2201 11990 。A. Nichol、J. Achiam 和 J. Schulman，《关于一阶元学习算法》，2018 [ ] 1803 02999 。在线。可访问：https://arxiv.org/abs/。H. B. McMahan、E. Moore、D. Ramage、S. Hampson 和 B. A. y. Arcas，《从去中心化数据中高效通信地学习深度网络》，*arXiv* 预印本 *arXiv:1602.05629*，2016年。 [在线]。可访问：https://arxiv.org/abs/1602 05629 。B. M. Lake、R. Salakhutdinov 和 J. B. Tenenbaum，《Omniglot 挑战：三年进展报告》，2019年。 [在线]。可访问：https://arxiv.org/abs/1902 03477 。P. Warden，《Speech Commands：一个用于有限词汇量语音识别的数据集》，2018 [ ] 1804 03209 。在线。可访问：https://arxiv.org/abs/。L. Dutta 和 S. Bharali，《TinyML 与物联网相结合：全面综述》，物联网，第16卷，文章编号100461，2021年。 [在线]。可访问：https://www.sciencedirect.com/science/article/pii/S2542660521001025

[7] P. P. Ray，《TinyML 综述：最新进展与前景》，沙特国王大学学报——计算机与信息科学，第34卷，第4期，第1595–1623页，2022年。 [在线]。可访问：https://www.sciencedirect.com/science/article/pii/S1319157821003335

[8] G. Datta、S. Kundu、Z. Yin、R. T. Lakkireddy、J. Mathai、A. Jacob、P. A. Beerel 和 A. R. Jaiswal，《P2M：面向资源受限微型机器学习应用的像素内存内处理范式》，2022年。 [在线]。可访问：https://arxiv.org/abs/2203 04737 。M. Scherer、A. Di Mauro、G. Rutishauser、T. Fischer 和 L. Benini，《用于 TinyML 应用的 22 nm FDX 技术全数字 TNN 加速器：1036 TOP/s/W、12.2 mW、每次推理 2.72 µJ》，载于*2022*年 *IEEE* 低功耗与高速芯片研讨会（*COOL CHIPS*），2022年，第1–3页。[10] Q. Lu 和 B. Murmann，《使用对数梯度输入图像提高 TinyML 计算机视觉的能效和鲁棒性》，2022 [ ] 2203 02571 。在线。可访问：https://arxiv.org/abs/。S. Zhuo、H. Chen、R. K. Ramakrishnan、T. Chen、C. Feng、Y. Lin、P. Zhang 和 L. Shen，《针对 TinyML 的低精度量化实证研究》，2022年。 [在线]。可访问：https://arxiv.org/abs/2203.05492

[12] H. Ren、D. Anicic 和 T. Runkler，《如何大规模管理微型机器学习：行业视角》，2022年。 [在线]。可访问：https://arxiv.org/abs/2202 09113 。T. Szydlo、P. P. Jayaraman、Y. Li、G. [13] Morgan 和 R. Ranjan，《TinyRL：迈向在微型嵌入式设备上实现强化学习》，载于《第31届 ACM 国际信息与知识管理会议论文集》，系列：CIKM '22。美国纽约州纽约：计算机协会，2022年，第4985–4988页。 [在线]。可访问：https://doi.org/10 1145 3511808 3557206 。/。R. [14] David、J. Duke、A. Jain、V. J. Reddi、N. Jeffries、J. Li、N. Kreeger、I. Nappier、M. Natraj、S. Regev、R. Rhodes、T. Wang 和 P. Warden，《TensorFlow Lite Micro：TinyML 系统上的嵌入式机器学习》，2020 [ ] 2010 08678 。在线。可访问：[15] https://arxiv.org/abs/。L. Ravaglia、M. Rusci、D. Nadalini、A. Capotondi、F. Conti 和 L. Benini，《一种支持设备端持续学习与量化潜在回放的 TinyML 平台》，IEEE新兴与精选电路和系统主题期刊，第11卷，第4期，第789–802页，2021年。[16] J. Lin、W.-M. Chen、Y. Lin、j. cohn、C. Gan 和 S. Han，《MCUNet：物联网设备上的微型深度学习》，载于《神经信息处理系统进展》，H. Larochelle、M. Ranzato、R. Hadsell、M. Balcan 和 H. Lin 编，第33卷。Curran Associates, Inc.，2020年，第11 711–11 722页。 [在线]。可访问：https://proceedings.neurips.cc/paper/2020/file/86c51678350f656dcc7f490a43946ee5-Paper.pdf [17] E. Njor、J. Madsen 和 X. Fafoutis，《TinyML 预测性维护入门：输入与模型优化》，载于IFIP 人工智能应用与创新国际会议。Springer，2022 年，第 67–78 页。[18] V. Tsoukas、E. Boumpa、G. Giannakas 和 A. Kakarountas，《医疗保健领域的机器学习与 TinyML 综述》，载于第25届泛希腊信息学会议，2021年，第69–73页。[19] T. Li、A. K. Sahu、M. Zaheer、M. Sanjabi、A. Talwalkar 和 V. Smith，《异构网络中的联邦优化》，《机器学习与系统论文集》，第2卷，第429–450页，2020年。[20] F. Sattler、S. Wiedemann、K.-R. M¨uller 和 W. Samek，《基于非独立同分布数据的鲁棒且通信高效的联邦学习》，《IEEE 神经网络与学习系统汇刊》，第31卷，第9期，第3400–3413页，2020年。[21] B. Jiang、J. Li、H. Wang 和 H. Song，《通过混合差分隐私和自适应压缩实现工业边缘计算中的隐私保护联邦学习》，《IEEE 工业信息学汇刊》，第1–1页，2021年。[22] O. Aramoon、P.-Y. Chen、G. Qu 和 Y. Tian，《元联邦学习》，2021 [ ] 2102 05561 。在线。[23] 可访问：https://arxiv.org/abs/。Y. Gao、M. Kim、S. Abuadbba、Y. Kim、C. Thapa、K. Kim、S. A. Camtepe、H. Kim 和 S. Nepal，《面向物联网的联邦学习与分割学习端到端评估》，arXiv 预印本 arXiv:2003.13376，2020年。[24] K. Kopparapu 和 E. Lin，《TinyFedTL：微型设备上的联邦迁移学习》，2021 [ ] 2110 01107 。在线。可 [25] 访问：https://arxiv.org/abs/。M. M. Grau、R. P. Centelles 和 F. Freitag，《微控制器上的设备端机器学习模型训练，并探讨联邦学习》，载于《社会公益信息技术会议论文集》，系列：GoodIT '21。美国纽约州纽约：计算机协会，2021年，第198–203页。 [在线]。可访问：https://doi.org/10 1145 3462203 3475896 。/。T. Hospedales、A. Antoniou、[26] P. Micaelli 和 A. Storkey，《神经网络中的元学习：综述》，《IEEE 模式分析与机器智能汇刊》，第44卷，第9期，第5149–5169页，2021年。[27] C. Finn、P. Abbeel 和 S. Levine，《用于快速适配深度网络的模型无关元学习》，载于《第34届国际机器学习会议论文集》，系列：ICML'17。澳大利亚新南威尔士州悉尼：JMLR.org，2017年，第1126–1135页。[28] A. Raghu、M. Raghu、S. Bengio 和 O. Vinyals，《快速学习还是特征复用？理解 MAML 的有效性》，2019年。 [ ] 1909 09157 在线。可访问：[29] https://arxiv.org/abs/。Y. Jiang、J. Kone˘cny、K. Rush 和 S. Kannan，《通过模型无关元学习提升联邦学习的个性化能力》，2019年。 [ ] 1909 12488 在线。可访问：[30] https://arxiv.org/abs/。A. Fallah、A. Mokhtari 和 A. Ozdaglar，《个性化联邦学习：一种元学习方法》，2020年。 [在线]。可访问：https://arxiv.org/abs/2002 07948 。S. Lin、G. Yang 和 [31] J. Zhang，《基于联邦元学习的协作学习框架》，载于2020年 IEEE 第40届分布式计算系统国际会议（ICDCS）。IEEE，2020年，第289–299页。

[32] C. Finn,A. Rajeswaran, S.Kakade, and S. Levine, "在线元学习"，载于国际机器学习大会。PMLR，2019年，第1920–1930页。[33] D. A.E.Acar、R.Zhu 和 V. Saligrama，"高效内存在线元学习"，载于第*38*届国际机器学习大会论文集，系列：机器学习研究论文集，M. Meila 和 T. Zhang 编，卷139。PMLR，2021年7月18–24日，第32–42页。 [在线]。可访问：https://proceedings.mlr.press/v139/acar21b.html [34] H. M. Gomes、J. Read、A.Bifet、J. P.Barddal 和 J. Gama，"流数据的机器学习：最新进展、挑战与机遇"，*ACMSIGKDD*探索通讯，卷21、22，2019年2月，第6–页。[35] A. Gepperth 和 B. Hammer，"增量学习算法及其应用"，载于欧洲人工神经网络研讨会（*ESANN*），2016年。[36] J. Montiel、M. Halford、S. M.Mastelini、G. Bolmier、R. Sourty、R.Vaysse、A. Zouitine、H. M.Gomes、J. Read、T. Abdessalem和 A. Bifet，"River：用于Python 流数据的机器学习"，2020年。 [ ] 2012 04740 在线。可访问：[37] https://arxiv.org/abs/。L.Ravaglia、M.Rusci、D. Nadalini、A. Capotondi、F. Conti 和 L. Benini，"一种用于通过量化潜在回放进行设备端持续学习的TinyML 平台"，*IEEE*新兴与精选电路和系统主题期刊，卷11，第4期，第789–802页，2021年。[38] H. Ren、D.Anicic 和 T. A. Runkler，"TinyOL：微控制器上的 TinyML在线学习"，载于*2021*年国际神经网络联合会议（*IJCNN*）。电气与电子工程师学会，2021年，第1–8页。[39] C.Banbury、V.J. Reddi、P.Torelli、J.Holleman、N.Jeffries、C. Kiraly、P. Montino、D. Kanter、S. Ahmed、D. Pau、U. Thakker、A. Torrini、P. Warden、J. Cordaro、G. Di Guglielmo、J.Duarte、S. Gibellini、V. Parekh、H. Tran、N. Tran、N.Wenxu 和 X.Xuesong，"MLPerf Tiny基准测试"，2021年。 [在线]。可访问：https://arxiv.org/abs/2106.07597

附录

A. 现实世界部署的实用指南

在这里，我们研究各项要素（即超参数）如何影响 TinyReptile 的性能，并为 TinyReptile 在现实世界中的部署提供指导。

首先，我们研究了 β 和 $S_{training}$ 对 Arduino 开发板上正弦波示例的影响。图 5 中的结果表明，较大的训练客户端 $S_{training}$ 支持集大小能够提升性能。此外，较高的客户端学习率 β 可以加快训练早期阶段的收敛速度，但不一定能改善最终性能。

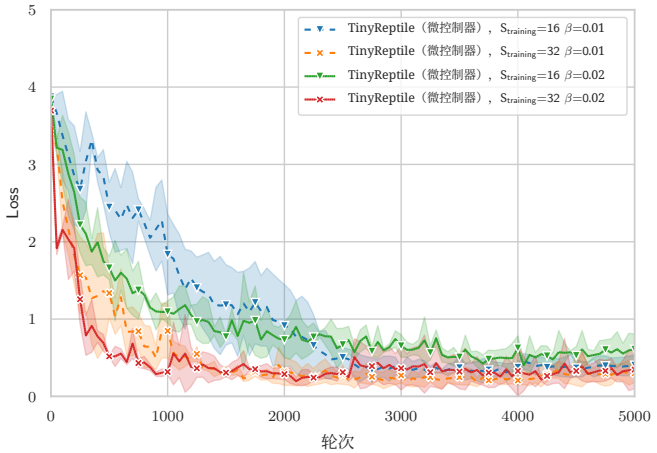
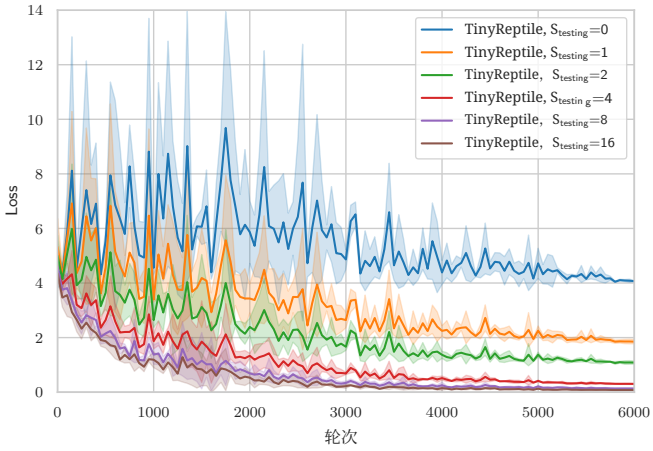


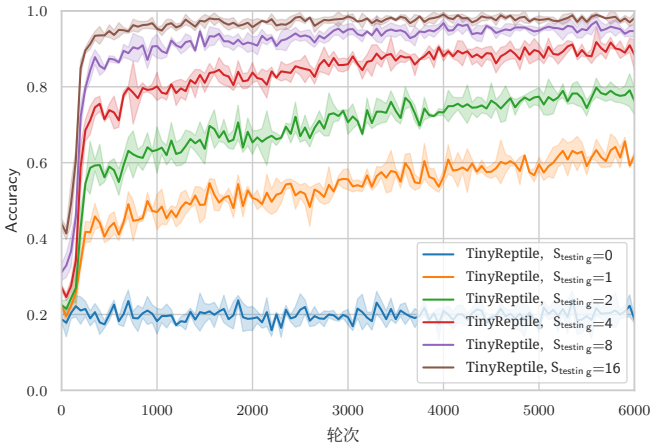
图 5：TinyReptile 在 Arduino Nano 33 BLE Sense 微控制器上的正弦波回归示例中的训练收敛情况。

这表明，学习率退火技术在实践中可能会有所帮助。

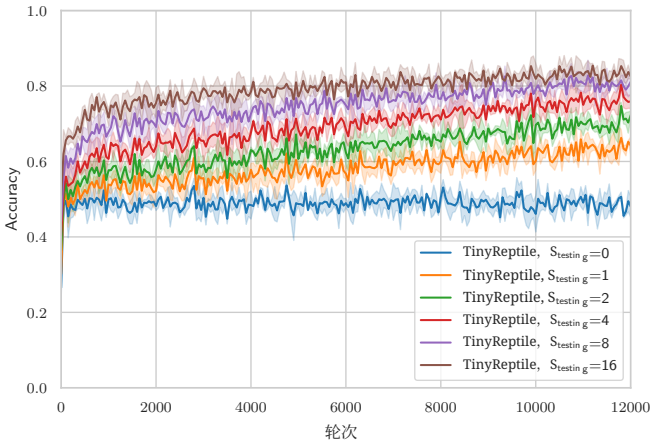
接下来，我们讨论测试客户端 $S_{testing}$ 的支持集大小所产生的影响，因为 TinyML 客户端在进行本地微调时，通常只有极少甚至没有标注数据。具体而言，我们希望研究本地微调需要多少数据才能取得良好性能。如图 6 所示，如果没有可用的支持数据，使用 TinyReptile 训练得到的全局初始化泛化效果并不好。然而，即使只提供一对数据用于适配，也能显著提升性能。此外，$S_{testing}$ 越大，性能越好。



(a) 正弦波示例。



(b) 具有五个输出类别的 Omniglot。



(c) 关键词识别，包含四个输出类别。

图 6：TinyReptile 在正弦波示例、Omniglot 和关键词识别数据集上的测试准确率随测试客户端支持集大小 $S_{testing}$ 的变化。