关于 HETEROFL 一种用于异构设备联邦学习架构的复现与改进

摘要

本文是关于会议 HETEROFL: COMPUTATION AND COMMUNICATION EFFICIENT FEDERATED LEARNING FOR HETEROGENEOUS CLIENTS 的复现与改进，原论文所设计的 HeteroFL 结构，主要用于联邦学习中，针对异构设备的通信与计算能力不同，提出的一种可将模型进行等比例缩放的架构，使得通信和计算机能力较弱的设备，训练缩放后较小的模型，来减轻节点的压力，不至于拖累全局。本文在此基础上，引入物联网节点的功耗限制，每个节点的功耗使得该节点只能进行一定次数的训练与通信。且模型越大，提供的信息越多，消耗的能量越多，相对应的可参与训练的轮次越少，每轮训练时，都需要根据目前节点的功耗，可提供的数据等方面进行抉择，节点能量耗尽会强制下线，探讨在该情况下，HeteroFL 的情况以及能否进行改进。

关键词：联邦学习；异构设备；功耗限制

1 引言

随着 5G 和机器学习等技术发展，物联网也迎来新的发展，5G 等通信技术中的 eMMC 标准使得物联网节点得以大批量的布置，这些物联网节点可以用于智慧城市、智慧农业等生活中的方方面面。大量的物联网节点产生了海量的数据，这些数据正好可以应用于机器学习，目前机器学习对于少量的数据仍旧没有特别好的处理方法。但是这些物联网节点收集到的信息往往涉及隐私问题，直接进行数据传输对于功耗和隐私问题都有压力，故采用联邦学习的方式，即避免了隐私问题，又减轻了通信压力。[1]

对于一个场景而言，许多物联网节点互为异构设备，他们的计算、通信、存储等能力各不相同，这些异构性质对于联邦学习提出了挑战。HeteroFL 是一种可以应用异构设备的联邦学习架构，将训练模型的隐藏层进行了宽度缩放，以此来缩减模型，适配能力不同的设备，提升训练的效果。 [2]

HeteroFL 虽然对于异构设备的联邦学习进行了一定的处理，但是并未针对物联网节点，在功耗和无线通信方面进行改进。本文在功耗方面进行限制，所有节点都有自身的能量限制，每次训练通信都会消耗一定的能量，能量耗尽就会自动下线，无法再次参与通信。所有节点初始的能量一致，模型越大，每轮训练提供的信息越多，消耗的能量就越多。在该限制下，就需要一个算法，来确定每轮训练中该选择什么节点参与训练，来达到最好的效果，本文提出的算法参考背包问题的思想，按权重比例来分配，保证每轮的训练都有一定量的数据，不出现前后训练量差距过大的情况。命名为 HeteroFL-IoT。该算法在 HeteroFL 架构中，可以取得比默认更好的效果。

2 相关工作

2.1 基于分布式机器学习的联邦学习

联邦学习（Federated Learning, FL）是一种分布式机器学习方法，旨在通过在多个分散的数据源上共同训练模型，而无需将数据集中到中央服务器。与传统的集中式机器学习方法不同，联邦学习将数据保留在本地设备（如智能手机、物联网设备等）上，通过局部模型的训练和参数的共享进行协作。参与者仅上传本地模型的更新（例如梯度或权重），而不是数据本身，从而有效地保护了用户隐私并减少了对数据传输带宽的依赖。每次迭代后，中央服务器将收集到的各方模型参数进行聚合，生成一个更新后的全局模型，然后再将该全局模型分发回各个设备进行下一轮训练。[3]

目前最常用的 FL 方法是 FedAvg，由 McMahan 等人提出，它允许本地设备（或客户端）在上传模型更新之前，进行多轮本地训练，从而显著减少了通信开销 [4]。尽管 FedAvg 被广泛采用，FL 仍然面临一些主要挑战，包括通信效率、系统异构性、统计异构性和隐私问题 [5]。为了降低通信成本，一些研究提出使用数据压缩技术，如量化和草图 [6]，同时也有提出采用分割学习 [7]。在应对系统异构性（即设备计算能力差异）方面，异步通信和主动客户端采样等方法得到了发展 [8]。然而，FL 中的主要挑战仍然是统计异构性，尤其是在数据分布非独立同分布（non-IID）时。为了解决这一问题，研究人员提出了通过结合元学习、多任务学习、迁移学习和知识蒸馏等个性化学习方法来调整全局模型，使其更适应局部数据的变化 [9]。HeteroFL 主要就是针对设备异构方面进行改进，使得联邦学习在设备异构的情况下，依旧能取得一个较为良好的效果。

2.2 联邦学习中的异构设备

在联邦学习中，异构设备指的是参与训练的设备在硬件、计算能力、存储容量和网络带宽等方面存在差异。由于这些设备的性能各不相同，它们在训练过程中所能承载的计算负载、数据处理能力和通信效率也会有所不同。这种异构性使得在进行联邦学习时，如何平衡各设备之间的负载、协调其参与方式成为一项关键挑战 [10]。例如，智能手机、物联网设备和边缘计算节点等设备的处理能力可能相差巨大，某些设备可能只具备较低的计算能力和有限的内存，导致其在进行本地模型训练时无法处理复杂的计算任务，甚至可能出现设备长时间不参与训练的情况。

面对这种设备异构的情况，HeteroFL 架构主要通过对模型进行等比例的缩放，来减少训练和通信所需要的数据量，以此来适配设备的异构型，同时在模型聚合的时候，采用了一种类 FevAvg 的算法来进行模型的聚合，具体的模型缩放和聚合方法在第三节进行介绍。 [2]

2.3 物联网设备的功耗限制

物联网设备（IoT）是通过网络连接的智能设备，它们广泛应用于家庭自动化、工业监控、智能交通等领域。这些设备通常由传感器、处理器、通信模块和电源系统组成，而其功耗一直是设计中的关键因素。与传统的计算设备相比，物联网设备通常要求在有限的电池或能源来源下长期运行，因此其功耗需要尽可能低以延长设备的使用寿命。设备的功耗受多个因素影响，包括处理能力、传感器的工作频率、无线通信模块的使用频率等。对于传感器设备而

言，功耗通常在待机模式和活动模式之间波动，设备在进行数据采集和传输时消耗较多电力，而在待机模式下则尽量保持低功耗状态。为了实现低功耗，许多物联网设备采用低功耗硬件和优化的工作协议，如低功耗蓝牙（BLE）和 Zigbee 等短距离通信协议。[11]

为将 HeteroFL 架构使用于物联网设备的联邦学习中，需要对其进行改进，加入对于物联网功耗的限制。这也正是本文的工作。

3 本文方法

3.1 本文方法概述

在以往的工作中，我们总是假设全局模型和本地的局部模型都采用完全相同的模型。为了减少本地的训练量和通信量，也有类似于模型压缩等方法，但为了保持全局模型的完整性，那么只降低能力弱的节点的模型就很容易想到。我们使得本地的局部模型属于全局模型的一个子集，即 $W_i^{t+1} \subseteq W_g^t$，其中 i 表示客户端的 id，g 表示 $global$，t 为当前训练的轮次。

我们可以通过改变一个神经网络的深度还有宽度来改变神经网络的大小 [12]。考虑到我们的目标是降低局部模型的计算复杂度，所以我们选择改变隐藏通道的宽度。这样，我们可以显着减少局部模型参数的数量，同时局部和全局模型架构也在同一模型类内，从而稳定了全局模型聚合。

3.2 模型缩放方法

为单个隐藏层改变参数的方法如下所示。首先隐藏层模型参数通过 $W_g \subset R^{d_g \times k_g}$ 表示，d_g 和 k_g 表示该层的输入和输出，即 W_g 为一个由 $d_g \times k_g$ 的矩阵。当我们的优化参数选择模型的复杂度 p 分为 3 个等级，参与通信的客户端数量 m 为 6 时，其示意图如图 1所示。

在图 1中我们可以很明显的看出 $W_l^p \subset W_l^{p-1} \subset \cdots \subset W_l^1$，$W_l^1$ 表示当前复杂程度最高的模型。在该复杂度定义的基础上，我们进一步定义隐藏通道的收缩比例 r。

$$d_l^p = r^{p-1} d_g, k_l^p = r^{p-1} k_g \tag{1}$$

根据隐藏通道收缩率，可以进一步推导出全局模型的收缩率，用 R 来表示。

$$|W_l^p| = r^{2(p-1)} |W_g|, R = \frac{|W_l^p|}{|W_g|} = r^{2(p-1)} \tag{2}$$

在其中，r^{p-1} 表示的是复杂度为 p 的隐藏层和全局模型该层的比例，在输入输出都按该比例减少的情况下，整个模型的比例收缩 R 就为该复杂度的隐藏层收缩比例的平方。在有了模型缩放的方法和定义后，就可以考虑聚合的方法，具体的细节，在下一小节介绍。

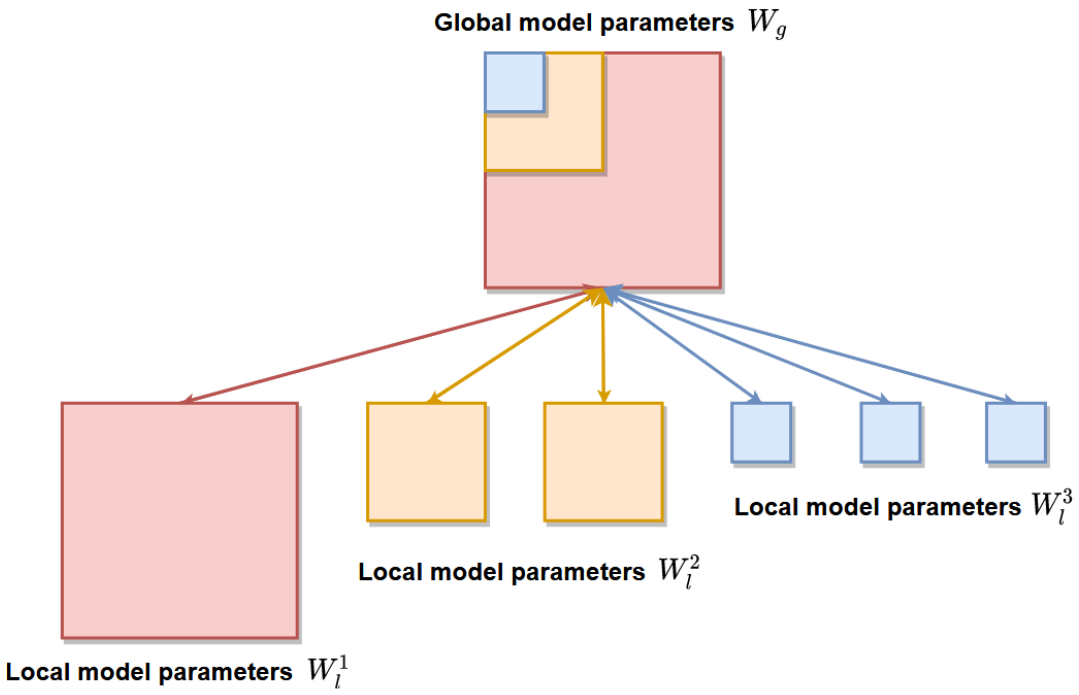

图 1. 模型的复杂度为 $p = 3$，参与通信的客户端数量 $m = 6$

3.3 模型聚合方法

根据每个客户端的能力，进行模型复杂度的分配，假设其每种模型复杂度的节点数量为 $\{m_1, m_2, \ldots, m_p\}$，模型的聚合方式为：

$$W_l^p = \frac{1}{m} \sum_{i=1}^{m} W_i^p \tag{3}$$

$$W_l^{p-1} \setminus W_l^p = \frac{1}{m - m_p} \sum_{i=1}^{m-m_p} W_i^{p-1} \setminus W_i^p, \ldots, W_l^1 \setminus W_l^2 = \frac{1}{m - m_{2:p}} \sum_{i=1}^{m-m_{2:p}} W_i^1 \setminus W_i^2 \tag{4}$$

$$W_g = W_l^1 = W_l^p \cup (W_{l-1}^p \setminus W_l^p) \cup \cdots \cup (W_l^1 \setminus W_l^2) \tag{5}$$

从公式（3）中可以看到，在模型聚合的时候，首先将复杂度相同为 p 的节点进行聚合，由于复杂度相同，其可视为简化的 FedAvg，也无需按照权重去取其比例。

公式（4）主要显示差异部分，例如 $W_l^{p-1} \setminus W_l^p$ 表示的是 p-1 模型复杂度的模型和 p 复杂度的模型在第 l 层之间的差异。

有了模型之间的差异之后，将其取一个并集，就可以获得全局模型，正如公式（5）所示。

3.4 功耗限制方法

对于功耗方面的限制，首先应该根据复杂度确定客户端的功耗，原文将复杂度分为 5 个等级，功耗分级也分为对应的等级。

假设：复杂度最高也就是功耗最高的设备，无法支持其每一轮都被选中，其可进行训练和通信的轮次一定小于全局通信轮次。

根据复杂度的分配程度，决定每轮的客户端的复杂度占比情况，保证每个通信训练轮次，其数据量不会有波动。

实际选择客户端时，根据客服端复杂度占比进行选择。

4 复现细节

4.1 与已有开源代码对比

本文复现了 HeteroFL 架构，这一架构已经由原论文进行了开源并公开。基于原 HeteroFL 架构的设计思路，本文特别关注了 HeteroFL 在物联网设备场景中的具体应用，结合物联网设备的特殊需求进行了针对性的改进。由于物联网设备通常受限于功耗的严格限制，因此本文在算法设计中限制了所有节点的训练参与能力，以降低设备的能耗开销。在此基础上，本文设计了一种改进算法，命名为 HeteroFL-IoT，该算法能够智能选择参与训练的物联网节点，从而在训练过程中有效分配资源。通过这种方式，HeteroFL-IoT 算法保证了每一轮训练中所需的计算量，同时避免了各轮之间训练量的过大波动，进一步提升了训练的稳定性和效率。

4.2 实验环境搭建

遵照原文的实验设置，在复杂度的等级区分，数据集和模型等，与原文保持不变，具体的情况如下表1和表2所示。

表 1. 实验环境信息

项目	详细信息
操作系统	Ubuntu 20.04 LTS
CPU	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
GPU	NVIDIA GeForce GTX 3090 × 8
内存	252GB
依赖项	Python 3.7, Pytorch 1.7, Numpy 1.19.2, etc.

表 2. 实验内容信息

项目	详细信息
数据集	MINST, CIFAR10, WikiText2
模型	CNN, ResNet18 Transformer
模型复杂度	1, 0.5, 0.25, 0.125, 0.0625
训练轮次	400，800，1600

4.3 具体算法

HeteroFL-IoT 的具体算法如下：

Algorithm 1 HeteroFL-IoT

Input：

数据 X_i 分布在 M 个本地客户端上，激活率 C 决定了每一轮通信的客户端的数量， 本地训练轮次 E，本地最小 batchsize B，学习率 η，全局模型参数 W_g，通道收缩率 r，复杂度的等级为 P。

System executes：

初始化全局模型参数 W_g^0 和本地的各项信息 $L_{1:K}$

for 每轮循环中，$t = 0, 1, 2, \ldots$ **do：**

$W_t \leftarrow max(C \times M, 1)$

$S_t \leftarrow$ 通过选择 M_t 个客户端，客户端的选择见下方的 **ClientSelect**

for 每一个客户端 $m \in S_t$ **in parallel do：**

根据基于本地信息 L_m 的复杂度 P 开始计算

$r_m \leftarrow r^{p-1}, d_m \leftarrow r_m d_g, k_m \leftarrow r_m k_g$

$W_m^t \leftarrow W_g^t[: d_m, : k_m]$

$W_m^{t+1} \leftarrow$ 客户端更新 $(m, r_m, W_,^t)$

for 每一个复杂度相同为 p 的客户端先进行聚合：

$W_g^{p-1,t+1} \setminus W_g^{p,t+1} \leftarrow \frac{1}{M_t - M_{M_{p:P,t}}} \sum_{i=1}^{M_t - M_{p:P,t}} W_i^{p-1,t+1} \setminus W_i^{p,t+1}$

$W_g^{t+1} \leftarrow \cup_{p=1}^{P} W_g^{p-1,t+1} \setminus W_g^{p,t+1}$

ClientSelect：

已知客户端的复杂度的比例分别为 a, b, c, d, e 这五种中的一种，并且得知其比例总和为 10，即 $a + b + c + d + e = 10$。

for 每次选择客户端时：

根据本地客户端的信息 $L_{1:K}$，确定其复杂度 p 的比例 L_p。

使用比例算出本轮该选择的数量 L_t，即 $L_t \leftarrow max(L_p \times 10, 1)$

选择客户端 $C_t \leftarrow L_t$

ClientUpdate： (m, r_m, W_m)

$B_m \leftarrow$ 根据 B 将本地数据 X_m 进行划分

for 每轮本地训练，共 E 轮：

for batch $b_m \in B_m$ **do：**

$W_m \leftarrow W_m - \eta \bigtriangledown \ell(W_m, r_m; b_m)$

将 W_m 返回给 server

4.4 创新点

针对原文 HeteroFL 架构未考虑到的物联网设备功耗问题，引入了节点的功耗限制，并且在此基础上，设计了 HeteroFL-IoT 算法用于保证在功耗限制的情况下依旧可以保证一定的训练效果。

5 实验结果分析

5.1 原文实验现象

```
目前是第2轮，该论参与训练的节点都有：[80, 55, 15, 42, 97, 60, 29, 24, 74, 65]
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(0%)  Local-Loss: 0.1638  Local-Accuracy: 96.6000  ID: 80(1/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:18.561140  Experiment Finished Time: 1:08:21.561140
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(10%)  Local-Loss: 0.1810  Local-Accuracy: 96.1000  ID: 55(2/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:16.362799  Experiment Finished Time: 1:07:46.362799
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(20%)  Local-Loss: 0.1567  Local-Accuracy: 96.5222  ID: 15(3/10)  Learning rate: 0.01  Rate: 1.0  Epoch Finished Time: 0:00:14.365140  Experiment Finished Time: 1:07:57.365140
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(30%)  Local-Loss: 0.1605  Local-Accuracy: 96.6500  ID: 42(4/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:12.276909  Experiment Finished Time: 1:07:43.276909
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(40%)  Local-Loss: 0.1588  Local-Accuracy: 96.7400  ID: 97(5/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:10.416567  Experiment Finished Time: 1:08:55.416567
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(50%)  Local-Loss: 0.1642  Local-Accuracy: 96.6611  ID: 60(6/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:08.293106  Experiment Finished Time: 1:08:33.293106
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(60%)  Local-Loss: 0.1680  Local-Accuracy: 96.6190  ID: 29(7/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:06.215069  Experiment Finished Time: 1:08:28.215069
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(70%)  Local-Loss: 0.1692  Local-Accuracy: 96.6208  ID: 24(8/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:04.148519  Experiment Finished Time: 1:08:31.148519
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(80%)  Local-Loss: 0.1681  Local-Accuracy: 96.6630  ID: 74(9/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:02.070536  Experiment Finished Time: 1:08:22.070536
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 2(90%)  Local-Loss: 0.1692  Local-Accuracy: 96.6300  ID: 65(10/10)  Learning rate: 0.01  Rate: 0.5  Epoch Finished Time: 0:00:00  Experiment Finished Time: 1:08:14
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Test Epoch: 2(100%)  Local-Loss: 0.1051  Local-Accuracy: 96.7300  Global-Loss: 0.1051  Global-Accuracy: 96.7300
```

图 2. 原始实验中，可见在 $a2b8$ 的复杂度分配下，实际选择节点时，占比为 $1:9$

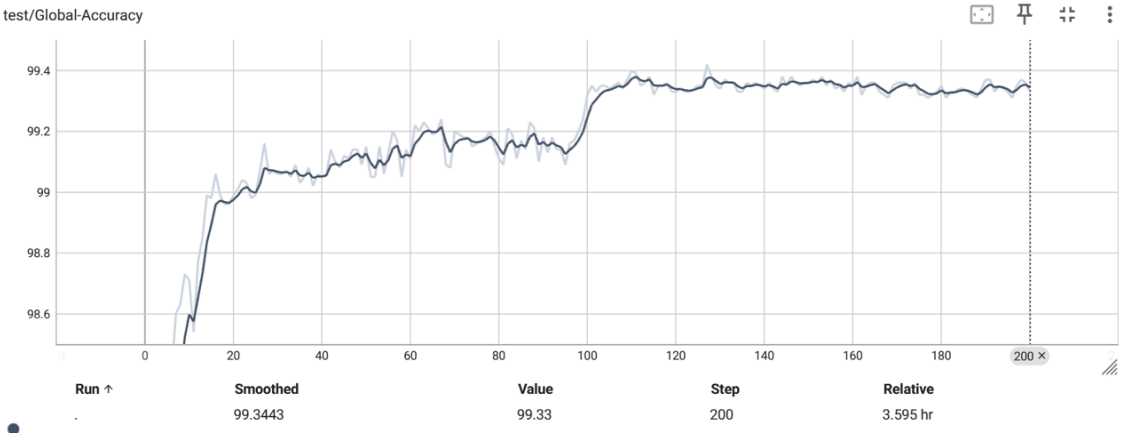

图 3. MNIST CNN client:100 C:0.1 iid fix a2-b8，最终全局精度

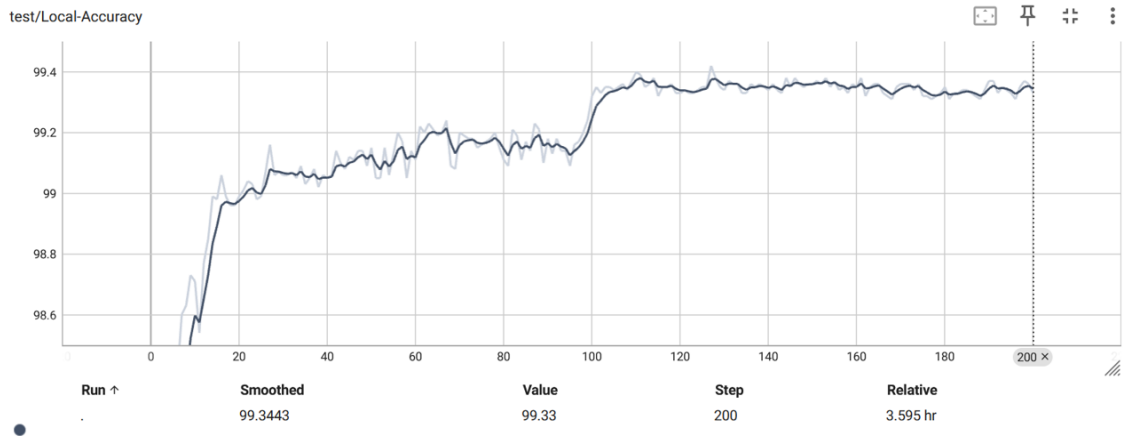

图 4. MNIST CNN client:100 C:0.1 iid fix a2-b8，最终本地精度

5.2 改进实验现象

```
(test2) lihongwei@labserver:~/data/review/src$ python train_classifier_fed.py --data_name MNIST --model_name conv --control_name 1_100_0.1_iid_fix_a2-b8_bn_1_1
第{i}个模型标识符: {cfg['model_tag']}
Experiment: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1
当前种子:{seed}
fetching data MNIST...
/home/lihongwei/data/review/src/utils.py:41: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module
implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#unt
rusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We rec
ommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this e
xperimental feature.
  return torch.load(path, map_location=lambda storage, loc: storage)
data ready
目前是第1轮，该轮参与训练的节点都有: [16, 18, 46, 91, 93, 67, 99, 58, 43, 34]
/home/lihongwei/miniconda3/envs/test2/lib/python3.9/site-packages/torch/functional.py:534: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the ind
exing argument. (Triggered internally at ../aten/src/ATen/native/TensorShape.cpp:3595.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(0%)  Local-Loss: 0.6377  Local-Accuracy: 83.5667  ID: 16(1/10)  Learning rate: 0.01  Rate: 1.0  Epoch Fini
shed Time: 0:00:25.010679  Experiment Finished Time: 1:32:35.010679
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(10%)  Local-Loss: 0.6374  Local-Accuracy: 84.1167  ID: 18(2/10)  Learning rate: 0.01  Rate: 1.0  Epoch Fin
ished Time: 0:00:19.631538  Experiment Finished Time: 1:21:42.631538
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(20%)  Local-Loss: 0.6733  Local-Accuracy: 83.6111  ID: 46(3/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fin
ished Time: 0:00:16.483934  Experiment Finished Time: 1:18:22.483934
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(30%)  Local-Loss: 0.6960  Local-Accuracy: 83.0083  ID: 91(4/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fin
ished Time: 0:00:13.738733  Experiment Finished Time: 1:16:10.738733
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(40%)  Local-Loss: 0.7077  Local-Accuracy: 82.8667  ID: 93(5/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fin
ished Time: 0:00:11.247747  Experiment Finished Time: 1:14:48.247747
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(50%)  Local-Loss: 0.7097  Local-Accuracy: 83.0167  ID: 67(6/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fin
ished Time: 0:00:08.893311  Experiment Finished Time: 1:13:52.893311
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(60%)  Local-Loss: 0.7139  Local-Accuracy: 82.8619  ID: 99(7/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fin
ished Time: 0:00:06.607531  Experiment Finished Time: 1:13:09.607531
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(70%)  Local-Loss: 0.7174  Local-Accuracy: 82.8250  ID: 58(8/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fin
ished Time: 0:00:04.355174  Experiment Finished Time: 1:12:17.355174
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(80%)  Local-Loss: 0.7218  Local-Accuracy: 82.8111  ID: 43(9/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fin
ished Time: 0:00:02.153197  Experiment Finished Time: 1:11:27.153197
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b8_bn_1_1  Train Epoch: 1(90%)  Local-Loss: 0.7240  Local-Accuracy: 82.7700  ID: 34(10/10)  Learning rate: 0.01  Rate: 0.5  Epoch Fi
nished Time: 0:00:00  Experiment Finished Time: 1:10:50
```

图 5. 改进实验中，可见在 $a2b8$ 的复杂度分配下，实际选择节点时，占比为 2 : 8

```
(test2) lihongwei@labserver:~/data/review/src$ python train_classifier_fed.py --data_name MNIST --model_name conv --control_name 1_100_0.1_iid_fix_a2-b4-c4_bn_1_1
第{i}个模型标识符: {cfg['model_tag']}
Experiment: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1
当前种子:{seed}
fetching data MNIST...
/home/lihongwei/data/review/src/utils.py:41: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module
implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#unt
rusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpic
kling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We rec
ommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this e
xperimental feature.
  return torch.load(path, map_location=lambda storage, loc: storage)
data ready
目前是第1轮，该轮参与训练的节点都有: [17, 11, 24, 28, 44, 45, 77, 66, 92, 89]
/home/lihongwei/miniconda3/envs/test2/lib/python3.9/site-packages/torch/functional.py:534: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the ind
exing argument. (Triggered internally at ../aten/src/ATen/native/TensorShape.cpp:3595.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(0%)  Local-Loss: 0.6298  Local-Accuracy: 85.4667  ID: 17(1/10)  Learning rate: 0.01  Rate: 1.0  Epoch F
inished Time: 0:00:24.673179  Experiment Finished Time: 1:31:20.673179
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(10%)  Local-Loss: 0.6393  Local-Accuracy: 85.0000  ID: 11(2/10)  Learning rate: 0.01  Rate: 1.0  Epoch
Finished Time: 0:00:19.426968  Experiment Finished Time: 1:20:51.426968
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(20%)  Local-Loss: 0.6742  Local-Accuracy: 84.5556  ID: 24(3/10)  Learning rate: 0.01  Rate: 0.5  Epoch
Finished Time: 0:00:16.437987  Experiment Finished Time: 1:18:09.437987
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(30%)  Local-Loss: 0.6878  Local-Accuracy: 83.9417  ID: 28(4/10)  Learning rate: 0.01  Rate: 0.5  Epoch
Finished Time: 0:00:13.557209  Experiment Finished Time: 1:15:09.557209
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(40%)  Local-Loss: 0.6983  Local-Accuracy: 83.6333  ID: 44(5/10)  Learning rate: 0.01  Rate: 0.5  Epoch
Finished Time: 0:00:11.018160  Experiment Finished Time: 1:13:16.018160
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(50%)  Local-Loss: 0.7130  Local-Accuracy: 83.0889  ID: 45(6/10)  Learning rate: 0.01  Rate: 0.5  Epoch
Finished Time: 0:00:08.676316  Experiment Finished Time: 1:12:04.676316
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(60%)  Local-Loss: 0.7389  Local-Accuracy: 82.4000  ID: 77(7/10)  Learning rate: 0.01  Rate: 0.25  Epoch
 Finished Time: 0:00:06.463333  Experiment Finished Time: 1:11:33.463333
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(70%)  Local-Loss: 0.7598  Local-Accuracy: 81.9042  ID: 66(8/10)  Learning rate: 0.01  Rate: 0.25  Epoch
 Finished Time: 0:00:04.279193  Experiment Finished Time: 1:11:02.279193
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(80%)  Local-Loss: 0.7784  Local-Accuracy: 81.1852  ID: 92(9/10)  Learning rate: 0.01  Rate: 0.25  Epoch
 Finished Time: 0:00:02.125185  Experiment Finished Time: 1:10:31.125185
Model: 0_MNIST_label_conv_1_100_0.1_iid_fix_a2-b4-c4_bn_1_1  Train Epoch: 1(90%)  Local-Loss: 0.7924  Local-Accuracy: 80.7733  ID: 89(10/10)  Learning rate: 0.01  Rate: 0.25  Epoc
h Finished Time: 0:00:00  Experiment Finished Time: 1:10:05
```

图 6. 在 $a2b4c4$ 的复杂度分配下，实际选择节点时，占比为 2 : 4 : 4

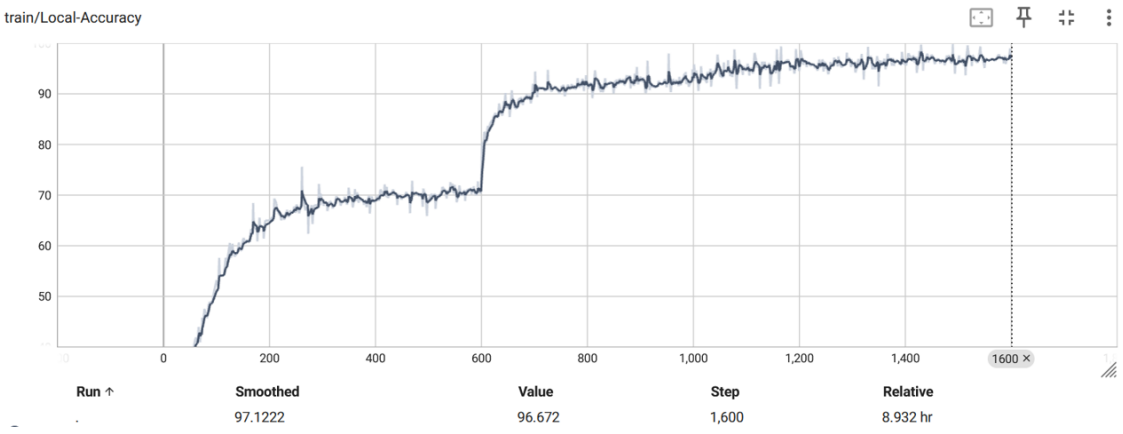

图 7. CIFAR10 resnet18 client:100 C:0.1 iid fix a2-b8，最终全局精度

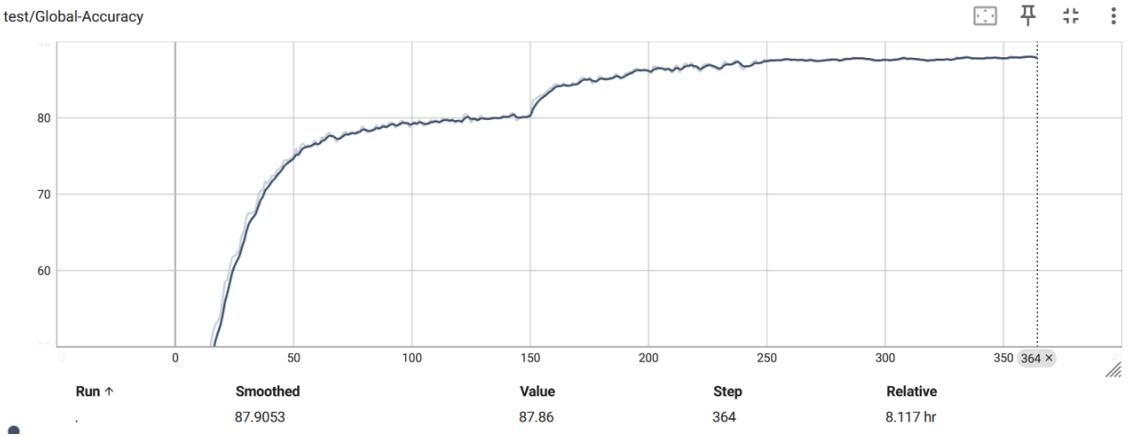

图 8. CIFAR10 resnet18 client:100 C:0.1 iid fix a2-b8，最终本地精度

6 总结与展望

6.1 目前存在的不足

本文虽然对于 HeteroFL 在物联网节点上的功耗情况做了限制，但是其限制情况较为简单粗暴，现实中物联网节点的功耗情况较为复杂，存在多种变量。同时物联网节点多为无线通信，无线通信对于通信量以及通信成功的概率都需要进行考虑。另外物联网节点所采集的数据信息多以传感器采集的数值数据为主，例如温湿度等，而文中所采用的数据集和模型，主要针对分类等任务。

6.2 未来研究方向

针对上面提到的不足，未来可以对物联网节点的功耗进行更全面，更复杂的限制，单独的为每一个节点确定功耗情况，同时考虑无线通信的通信失败的情况，限制每一轮通信发送的数据量。还需要针对线性回归等任务进行实验测试，更贴合实际情况。

参考文献

[1] Latif U. Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Commun. Surv. Tutorials*, 23(3):1759–1799, 2021.

[2] Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[3] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning: Concepts, advances, and challenges. *IEEE Trans. Knowl. Data Eng.*, 36(7):3615–3634, 2024.

[4] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[5] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.*, 37(3):50–60, 2020.

[6] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed SGD with sketching. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13144–13154, 2019.

[7] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 8485–8493. AAAI Press, 2022.

[8] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In Ameet Talwalkar, Virginia Smith,

and Matei Zaharia, editors, *Proceedings of the Second Conference on Machine Learning and Systems, SysML 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org, 2019.

[9] Paul Pu Liang, Terrance Liu, Ziyin Liu, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *CoRR*, abs/2001.01523, 2020.

[10] Mang Ye, Xiuwen Fang, Bo Du, Pong C. Yuen, and Dacheng Tao. Heterogeneous federated learning: State-of-the-art and research challenges. *ACM Comput. Surv.*, 56(3):79:1–79:44, 2024.

[11] Chenning Li and Zhichao Cao. Lora networking techniques for large-scale and long-term iot: A down-to-top survey. *ACM Comput. Surv.*, 55(3):52:1–52:36, 2023.

[12] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.