arXiv:2602.06693v1 [cs.NI] 6 Feb 2026

分割学习中的最大完工时间最小化：从理论到实践

Robert Ganian*、Fionn Mc Inerney†和Dimitra Tsigkari†*算法与复杂性组，
维也纳工业大学，维也纳，奥地利，rganian@gmail.com
†Telef´onica 科学研究中心，巴塞罗那，西班牙，{fionn.mcinerney、dimitra.tsigkari}@telefonica.com

摘要—分割学习近来成为一种解决方案，可用于异构物联网设备上的分布式机器学习，客户端可以将部分训练任务卸载给计算能力强大的辅助节点。分割学习的核心挑战是，通过联合制定客户端—辅助节点分配方案以及辅助节点上的任务调度，最大限度地缩短训练时间。我们首先研究一种模型：每个辅助节点都有内存基数约束，限制其可分配的客户端数量，这代表同构任务的情形。通过复杂性理论，我们证明，即使对于该问题受到严格限制的实例，也不存在精确的多项式时间算法或近似方案。针对这些负面结果，我们进一步提出一个非平凡的多项式时间5-近似算法。在此基础上，我们接着研究Tirana等人 [INFOCOM 2024],所考虑的更一般的异构任务场景，其中辅助节点受内存容量约束，且客户端具有可变内存成本。对于这种情形，我们证明，除非P = NP，否则该问题不可能存在任何近似因子的多项式时间近似算法。不过，通过改进上述5-近似算法，我们为异构任务场景设计了一种新颖的启发式算法，并通过大量实验表明，其表现优于先前研究中的启发式算法。

I. 引言

分割学习（SL）是一种现代框架，允许具有异构计算资源的设备（客户端）协同训练深度神经网络（NN）模型 [1]。分割学习放宽了联邦学习（FL） [2]要求训练在客户端进行的限制，使客户端能够将部分训练任务卸载至一个辅助节点——通常是云端的服务器或无线通信中的基站。这使得缺乏在本地训练模型所需资源的异构客户端仍可参与分布式学习，并有可能利用未充分利用的网络资源，同时加速训练过程。基于这些原因，分割学习范式正日益受到在物联网（IoT）网络中具有重要意义，并在6G网络中有众多应用 [3], [4]。

所有作者对本项工作贡献相同。本文将发表于**IEEE INFOCOM 2026**论文集，并获得奥地利科学基金（FWF） [10.55776/Y1329 和 10.55776/COE12], 维也纳科学技术基金（项目 10.47379/ICT22029）、智能网络与服务联合事业（SNS JU）在欧盟地平线欧洲计划及创新计划下的资助（资助协议编号：101139067，ELASTIC），以及地平线玛丽·斯克沃多夫斯卡-居里博士后奖学金 OPALS（资助协议编号：101210495）的资助。本文所表达的观点和意见仅代表作者本人，并不一定反映欧盟的立场。欧盟及其资助机构均不对此承担责任。

Funded by the European Union

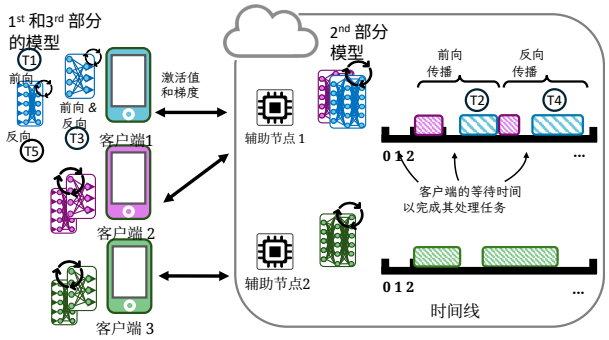

图1. 客户端-辅助方分配和调度决策示例。处理任务1至5（T1–T5）对应于客户端1的模型部分。

基于由层（即神经元块）组成的神经网络模型架构，在分割学习中，模型被拆分为 3 个部分。这些部分由连续的层构成，并由切分层分隔。第 1 部分和第 3 部分由客户端处理，而第 2 部分则由其中一个辅助节点处理。采用这种拆分方式时，客户端保留数据所有权；而采用两部分拆分时，客户端则需要向辅助节点共享其数据的标签 [1]。在本工作中，我们考虑切分层预先确定的分割学习训练框架：客户端并行训练各自的模型部分，辅助节点则为每个分配到的客户端维护一份独立的第 2 部分副本；每个训练轮次结束时，模型部分会在聚合器或承担该角色的辅助节点处进行聚合（例如通过联邦平均 [2]）。这种变体通常称为 SplitFedV1 [5]；重要的是，已有研究表明其准确率与联邦学习 [6], 相近，并且文献中已对其进行了广泛研究 [7]–[9]。

在前向传播过程中，三个模型部分需要依次处理，而在反向传播过程中则需要按相反顺序处理。因此，对于每个客户端，训练过程自然可分解为五项处理任务，如图 1 所示。这些任务分别是：对第 1 部分进行前向传播（T1），接着对第 2 部分进行前向传播（T2），对第 3 部分进行前向和反向传播（T3），对第 2 部分进行反向传播（T4），最后对第 1 部分进行反向传播（T5）。客户端与辅助节点之间通过交换激活值（前向传播期间）和梯度（反向传播期间）来完成交接。上述双向通信可能会严重影响总训练时间（完工时间），即所有客户端完成模型训练所需的最长时间。鉴于

在众多应用中（例如零接触网络 [10]），实现较短的训练时间至关重要，因此有必要优化分割学习操作，尤其是优化完工时间。从可扩展性的角度来看，这种优化也很重要，因为可扩展性是分割学习范式面临的主要挑战之一 [6], [11], [12]。

虽然在多个计算节点的场景下最小化完工时间是调度和资源分配中的经典优化问题，但分割学习带来了独特挑战。具体而言，辅助节点的容量有限、处理任务存在优先约束（如上所述），且任务的释放时间各不相同，因此我们无法直接应用一般调度文献中的已知算法结果。相关工作提出了我们关注的问题及其若干变体，并为最大完工时间最小化提供了启发式算法 [7], [13], [14]——参见第 I-B 节的讨论。除了 [13], 确立的该问题的弱NP-难性之外，我们对分割学习最大完工时间最小化的计算界和可近似性仍缺乏清晰认识。本文首次弥补了这一空白，详述如下。

A. 我们的贡献

我们首先考察如何计算使训练时间最小化的客户端—辅助节点分配及调度方案的复杂度，并将此问题记为 SL-MAKESPAN（所有所考虑问题的正式定义见第 II 节）。如上所述，即使只有两个辅助节点，SL-MAKESPAN 也是弱 NP-难的 [13]。我们调整已知归约，证明即使将问题限制为以下实例：每个客户端都与每个辅助节点相邻；处理时间非零的任务只有 T2任务；且每个辅助节点的处理能力相同，SL-MAKESPAN 仍然：(1) 即使每个辅助节点至多分配 3 个客户端，也具有强NP-难性；(2) 按辅助节点数量参数化时，具有 W[1]-难性[1] I（定理1）。此外，即使只有一个辅助节点，根据 [16] 的一个结果可以看出，仅通过调度任务来最小化完工时间也具有强NP-难性，即使 T1任务和 T5任务的处理时间为零，而 T2任务和 T4任务的处理时间均为单位时间（定理2）。

鉴于求解 SL-MAKESPAN 精确解本身难以处理，我们转而研究近似算法。遗憾的是，我们证明，除非 P = NP，否则 SL-MAKESPAN 不存在多项式时间近似方案（PTAS）。事实上，通过从 R || $C_{\max}$ 归约到 SL-MAKESPAN，我们证明了一个更强的结论：除非 P = NP，即使将问题限制为每个客户端都与每个辅助节点相邻，且只有 T2任务的处理时间非零的实例，SL-MAKESPAN 也不存在多项式时间 (3/2)-近似算法（定理3）。需要注意的是，R || $C_{\max}$ 的不可近似性无法进一步大幅提升，因为众所周知，它存在多项式时间 2-近似算法 [17], ，而缩小这一差距仍是一个长期悬而未决的问题 [18], [19]。我们证明，SL-MAKESPAN 也有相同的情况。也就是说，尽管越来越多的证据表明，即使对于限制条件非常严格的实例，SL-MAKESPAN 仍然难以处理，但作为我们的**首个主要理论贡献**，我们给出了 SL-MAKESPAN 的首个近似算法，具体而言，是一个多项式时间 5-近似算法（定理4）。

在此基础上，我们进一步考虑 [13], [14], 中提出的 SL-MAKESPAN 的一般化问题，并将其称为 GENSL-MAKESPAN。在该问题中，每个辅助节点都有一定的内存容量，并需要分配一部分内存来存储从客户端接收的激活值，而所需内存量因客户端而异。请注意，SL-MAKESPAN 可视为 GENSL-MAKESPAN 的一个特例，其中每个客户端所需的内存量相同，因此可将其视为单位成本。由于即使对于 SL-MAKESPAN，单个辅助节点上的调度问题也已是难解的（定理2），我们同样证明了，判定 GENSL-MAKESPAN 是否存在可行的客户端—辅助节点分配（记为 CH-ASSIGN）本身就是难解的。具体而言，我们给出了从 P || $C_{\max}$ 到 CH-ASSIGN 的多项式时间归约，从而得到了与 SL-MAKESPAN（定理 5）类似的难解性结果。特别是，由于判定是否存在可行的客户端-辅助节点分配是强NP-难的，作为我们的**第二项主要理论贡献**，我们证明，除非 P = NP，否则 CH-ASSIGN 不存在任何近似因子的多项式时间近似算法。

然而，作为我们的**主要实践贡献**，我们对用于 SL-MAKESPAN 的 5-近似算法进行了适当调整，从而得到一种用于 GENSL-MAKESPAN 的启发式算法，我们称之为**均衡分布**（简称**EquiD** ）。该算法通过求解整数规划（例如，使用求解器）来寻找可行的客户端-辅助节点分配，而调度部分则与我们的 5-近似算法相同。在第五节中，我们通过基于开源数据和合成数据的实验结果[2] 进一步表明，EquiD 的表现优于文献中的其他启发式算法，并且只需求解器找到最优解所需时间的一小部分，就能找到近似最优解。

B. 相关工作

分割学习（SL）最早于 [1]（同年还有 [20] ）作为一种分布式学习方法被提出；与联邦学习（FL）相比，该方法对参与设备的计算要求较低。此后，人们研究了该框架的多个变体，其中最突出的当属 SplitFed [5] 变体。除关注分割学习中完工时间最小化的问题（下文将对此展开讨论）的研究外，相关工作还研究了分割学习场景下与隐私和安全 [5], [21],[22],、异构数据下的收敛性 [8], [9], [23], [24],、切分层选择 [25], [26], 以及能效 [3], [27],[28]相关的问题。最后，部分文献关注一种“混合”场景，在该场景中，客户端可以通过联邦学习或分割学习参与训练，例如 [13], [29], [30]。

最小化分割学习的训练完工时间已被确认为分割学习的主要挑战之一 [12]。文献 [3]

[1]在广为接受的假设 FPT $\neq$ W[1] 成立的前提下（例如参见 [15]），对于输入规模为 n 的 SL-MAKESPAN，任何可计算函数 f 对应的 $f(I)\cdot n^{\mathcal{O}(1)}$ 时间内都无法求解。

[2]代码可在 https://github.com/ddd-ttt99/SL-makespan 获取。

提出了同时最小化完工时间和能耗的问题。不过，该研究假设每个辅助节点只服务一个客户端，这大大简化了客户端—辅助节点分配，也消除了调度的需要。文献 [7] 研究了只有一个辅助节点时的完工时间最小化问题，并提出了一种确定切分层和辅助节点调度的方法。另一方面，文献 [14] 的作者提出了存在多个辅助节点时的完工时间最小化问题，并提出了两种启发式算法。第一种将问题分解为多个子问题，第二种通过负载均衡算法确定客户端—辅助节点分配，并采用先到先服务调度策略。最后，还有一些研究通过考虑切分层选择、无线电频谱分配（在无线网络中）和设备聚类等决策来缩短训练时间，例如 [31]–[33]。

II. 问题表述

我们考虑一个系统，其中包含一组 $\mathcal{J} = \{1, \ldots, J\}$ 客户端（设备）和一组 $\mathcal{I} = \{1, \ldots, I\}$ 辅助节点（位于云端或网络边缘），它们通过链路 $\mathcal{E}$相连。这些链路可能反映云端的计算配置，或边缘的基于位置的策略。由此自然定义了一个二分图 $G = (\mathcal{J},\ \mathcal{I},\ \mathcal{E})$。相关研究通常假设辅助节点具有相同的计算/内存容量，例如 [3],；在本文第二部分中，我们遵循近期研究提出的框架，考虑更一般的问题，即辅助节点是能力各异的处理器（如 CPU 或 GPU）。这种推广能够刻画异构性较高的现实场景 [34]。具体而言，辅助节点 $i \in \mathcal{I}$ 的内存容量记为 M_i ，反映其物理内存（例如以兆字节或吉字节为单位）。最后，辅助节点的计算能力既可以用每秒周期数表示，也可以用计算任务的估算耗时表示（通过下文所述的性能分析获得）。

A. 分割学习训练的步骤

为了最小化训练完工时间，我们与以往工作 [7], [13], [14]一样，着重于最小化每个客户端处理单个数据批次所需的训练时间。值得注意的是，在实际应用中，训练过程自然会分解为多个连续的批次训练过程——批次数量取决于客户端数据的划分方式、训练轮次和训练回合的数量等。

与调度文献中的常见做法一样，我们采用时隙模型，将时间划分为区间 S_t，其中 $S_t = [t, t+1]$对所有整数 $t \geq 0$ 均成立；时隙以毫秒等单位计量。对于客户端$j \in \mathcal{J}$的每个等大小数据批次，都会执行以下任务：

T1： 客户端 j 对第 1 部分执行前向传播，并在 r_j 个时隙内将预先确定的切分层的激活值发送到云端。

T2任务： 分配的辅助节点 $i \in \mathcal{I}$ 在 p_{ij} 个时隙内对第 2 部分进行前向传播。请注意，这段时间取决于辅助节点 i的处理能力，以及客户端 j 的第 2 部分大小（各客户端的大小可能不同）。

T3任务： 客户端 j 从辅助节点 i接收第 2 部分的激活值，并对第 3 部分进行前向传播。随后，客户端 j计算损失函数，开始对第 3 部分进行反向传播（即计算梯度和模型权重），并将梯度传输至云端。这些操作持续 ℓ_j 个时隙。**T4任务：** 辅助节点 i 在 p'_{ij}个时隙内对第 2 部分进行反向传播。**T5任务：** 客户端 j 接收第 2 部分的梯度，并在 r'_j 个时隙内将其反向传播至第 1 部分。至此，一个批次的训练完成。

辅助节点会分配部分内存，用于存储从每个被分配的客户端接收的激活值/梯度，以及各客户端第 2 部分的模型权重。我们用 d_j表示客户端 j 在所分配辅助节点上的内存占用。在完工时间最小化问题中，我们区分两种对内存容量约束建模的方式。首先，将这些约束视为基数约束，即确定每个辅助节点最多能服务多少客户端。鉴于每个辅助节点的内存容量有限，或存在其他通信约束，辅助节点 i 在一次批次更新期间最多可服务 M_i 个客户端，对所有 $i \in \mathcal{I}$均成立；这等同于假设对所有$j \in \mathcal{J}$均有 $d_j = 1$ 。客户端到辅助节点的分配应满足基本服务约束，该约束刻画了处理任务同质的场景（即每个客户端的内存占用相同）。随后，在第四节中，我们还考虑综合服务约束，允许客户端具有不同的内存需求（即 d_j的值各不相同），这与以往工作 [13], [14]一致。我们假设前述内存容量 M_i 以及时间 r_j、 p_{ij}、 ℓ_j、p'_{ij}和 r'_j 均为非负整数。

设客户端—辅助节点分配 $Y : \mathcal{J} \to \mathcal{I}$ 是一种将客户端分配给辅助节点的映射，并且对所有 $i \in \mathcal{I}$， $Z_Y(i) := \{j \mid Y(j) = i\}$。若客户端—辅助节点分配 Y 满足以下条件，则称其为可行：
(a) 满足 G 的邻接约束，即对所有$j \in \mathcal{J}$，$(j, Y(j)) \in \mathcal{E}$；以及
(b) 满足辅助节点的服务约束，即对所有 $i \in \mathcal{I}$， $\sum_{j \in Z_Y(i)} d_j \leq M_i$。值得注意的是，在处理客户端的批次 [7], [14] 期间，客户端—辅助节点分配不能发生改变。

B. 批次的训练完工时间

在基本调度问题中，调度是将作业 $\mathcal{J}$分配到机器 $\mathcal{I}$ 的时间区间的一种方案，使每个作业都分配给一台机器，且同一台机器的同一时间区间内不会分配两个作业（即单线程处理）。对于分配给机器 $i \in \mathcal{I}$的每个作业 $j \in \mathcal{J}$，在相同并行机的情况下，它需要分配的时间区间数量等于其处理时间 p_j；在无关并行机的情况下，则等于 p_{ij}。如果允许抢占，则作业所分配的时间区间不必连续。如果没有指定图和机器容量约束（例如第 III 节中 $P \mid\mid C_{\max}$ 和 $R \mid\mid C_{\max}$ 的定义），则任何作业都可以分配给任何机器。调度的完工时间是最后一个作业完成的时间。

我们的设定与上述描述类似，但作业现在是客户端的T2任务和T4任务，机器是辅助节点，并且存在图约束以及辅助节点内存的基数／容量约束，同时允许抢占。我们强调，我们的任何结果都没有利用抢占，但为了与 [13], [14]的模型保持一致，我们允许抢占。只有客户端的T2任务和T4任务需要分配到辅助节点的时间区间；请注意，同一客户端的T2任务和T4任务必须分配给同一个辅助节点，而且只有在T2任务完成后才能调度T4任务。此外，还存在以下附加约束。给定客户端 j：(a) 其T2任务的释放日期等于 r_j，即完成其T1任务所需的时间（也就是说，T2任务不能在时间 r_j 之前开始处理）；(b) 其T2任务完成后，还需经过 ℓ_j 个时隙的延迟（即完成其T3任务所需的时间），其T4任务才能开始。客户端 j对应批次的完成时间记为 c_j，即其T5任务的完成时间；T5任务在T4任务完成 r'_j 个时隙后完成。这些要求源于SL工作流中各任务之间的优先关系。

我们强调，由于客户端只负责其本地模型的第 1 部分和第 3 部分（即任务 1、3 和 5），因此一旦这些任务可执行，它们就会立即处理：任务 1 在时间 0 开始；辅助节点完成任务 2 后开始任务 3；任务 4 完成后开始任务 5。原则上，分布式学习中的一个自然假设是，设备在训练期间处于空闲状态 [35]。因此，无需为客户端正在处理的任务定义调度。

一个批次的完工时间由所有客户端中最长的训练时间决定。因此，我们的目标是最小化 完工时间$\max_{j\in\mathcal{J}} c_j$。

现在，我们已有必要的定义和符号，可以正式定义所研究的问题。与证明计算复杂性结论时的常见做法一样，我们考虑相关问题的判定变体。不过需要指出，我们的所有算法也适用于优化版本[3]，并且都是构造性的，即能够输出一个可行的对应调度，以及计算得到的完工时间。对于辅助节点的内存基数约束，我们研究以下问题。

SL-MAKESPAN
输入： 一个二分图 $G=(\mathcal{J},\ \mathcal{I},\ \mathcal{E})$，辅助节点的内存容量 M_1, ..., M_I，客户端的内存需求$d_1=\cdots=d_J=1$，时间 r_j、p_{ij}、ℓ_j、p'_{ij}和 r'_j（对所有 $i\in\mathcal{I}$ 和 $j\in\mathcal{J}$），以及一个正整数 k。
问题： 是否存在可行的客户端-辅助节点分配和调度，使完工时间至多为 k?

在更一般的内存容量约束情形下——这反映了异构性较高的场景——我们考虑 [13], [14]中提出并研究的SL-MAKESPAN 的以下推广。

[3]我们略微滥用记号，对所考虑问题的判定版本和优化版本使用相同的名称。

广义分割学习完工时间问题（GENSL-MAKESPAN）
输入： 一个二分图 $G=(\mathcal{J},\ \mathcal{I},\ \mathcal{E})$，辅助节点的内存容量 M_1, ..., M_I，客户端的内存需求 d_1, ..., d_J，时间 r_j、p_{ij}、ℓ_j、p'_{ij}和 r'_j（对所有 $i\in\mathcal{I}$ 和 $j\in\mathcal{J}$），以及一个正整数 k。
问题： 是否存在可行的客户端-辅助节点分配和调度，使完工时间至多为 k?

SL-MAKESPAN 和 GENSL-MAKESPAN 的两个优化版本都采用上述相同的输入，但要求找到一个可行的客户端—辅助节点分配和调度，使完工时间最短。对于GENSL-MAKESPAN，为了证明仅仅判定是否存在可行的客户端—辅助节点分配就是强NP-难问题，我们定义GENSL-MAKESPAN 的以下限制版本。

客户端—辅助节点分配（CH-ASSIGN）
输入： 一个二分图 $G=(\mathcal{J},\ \mathcal{I},\ \mathcal{E})$、辅助节点的内存容量 M_1, ..., M_I 以及客户端的内存需求 d_1, ..., d_J。
问题： 是否存在可行的客户端—辅助节点分配?

III. SL-MAKESPAN：难解性与可近似性

本节研究精确求解和近似求解 SL-MAKESPAN 的计算复杂度。我们的一些正式计算复杂性结果，是通过从两个非常著名、以最小化完工时间为目标的调度问题进行多项式时间归约得到的：(a) 在相同并行机上的调度问题，记为 $P\ ||\ C_{\max}$；(b) 在无关并行机上的调度问题，记为$R\ ||\ C_{\max}$。这两个问题定义如下。

$P\ ||\ C_{\max}$
输入： 一组作业 $\mathcal{J}$、一组相同并行机$\mathcal{I}$、对所有$j\in\mathcal{J}$均为非负整数的作业处理时间 p_j，以及一个正整数 k。
问题： 是否存在一个完工时间至多为 k 的调度?

$R\ ||\ C_{\max}$
输入： 一组作业 $\mathcal{J}$、一组无关并行机 $\mathcal{I}$、对所有 $i\in\mathcal{I}$ 和 $j\in\mathcal{J}$均为非负整数的作业处理时间p_{ij}，以及一个正整数 k。
问题： 是否存在一个完工时间至多为 k的调度?

我们首先扩展两个辅助节点 [13] 情况下的弱 NP难解性结果，证明即使对 SL-MAKESPAN 的实例施加严格限制，该问题仍然是难解的。

定理 1. 即使将实例限制为满足 G是完全二分图，且对所有 i、$i'\in\mathcal{I}$和 $j\in\mathcal{J}$，$r_j=\ell_j=p'_{ij}=r'_j=0$和 $p_{ij}=p_{i'j}$，SL-MAKESPAN仍然是(1) 强 *NP* 困难的，即使 $M_1=\cdots=M_I=3$，并且(2)W*[1]*-困难（以 I 为参数）。

证明。 如果 G 是完全二分图，且对所有 i， $r_j = \ell_j = p'_{ij} = r'_j = 0$和 $p_{ij} = p_{i'j}, i' \in \mathcal{I}$和 $j \in \mathcal{J}$，那么 SL-MAKESPAN 等价于 $P \mid\mid C_{\max}$，其中 SL-MAKESPAN 中的客户端和辅助节点分别对应 $P \mid\mid C_{\max}$中的作业和机器。由于已知 $P \mid\mid C_{\max}$ 是 (1) 强 NP 困难的，即使必须恰好将 3 个作业分配给每台机器（ $P \mid\mid C_{\max}$ 的这一特例称为 3-划分问题） [36]；并且 (2) W[1]-困难（以机器数量为参数） [37], [38]，因此定理结论成立。 □

前一定理中的难解性源于确定客户端—辅助节点分配。然而，即使强制指定这种分配（或者只有一个辅助节点），调度问题在计算上仍然难以求解，即便是在高度受限的情况下也是如此。事实上，这是因为 SL-MAKESPAN 的某些受限实例等价于文献中一个已知的调度问题。

定理2. [16,定理. 24] 即使将实例限制为满足 $I = 1$, G 构成完全二分图，且对所有 $j \in \mathcal{J}$， $r_j = r'_j = 0$ 和 $p_{1j} = p'_{1j} = 1$ 的情形，SL-MAKESPAN 仍是强NP-难的。

由于精确求解 SL-MAKESPAN 本质上难以实现，接下来一个自然的思路是转而寻求近似解。遗憾的是，我们证明了即使对于非常受限的实例，近似求解 SL-MAKESPAN 也很困难。具体而言，除非P =NP，否则 SL-MAKESPAN 不存在多项式时间近似方案。事实上，还成立一个更强的结论：

定理3。 除非P = NP，即使将范围限制在满足以下条件的实例中：G 是完全二分图，且对所有 $i \in I$ 和 $j \in J$， $r_j = \ell_j = p'_{ij} = r'_j = 0$，SL-MAKESPAN 也无法存在多项式时间的*(3/2)*-近似算法。

证明。 如果 G是完全二分图，且对所有 $i \in I$ 和 $j \in J$， $r_j = \ell_j = p'_{ij} = r'_j = 0$，那么 SL-MAKESPAN 等价于 $R \mid\mid C_{\max}$，其中 SL-MAKESPAN 中的客户端和辅助节点分别对应于 $R \mid\mid C_{\max}$中的作业和机器。因此，定理的结论成立，因为除非P =NP [17]，否则 $R \mid\mid C_{\max}$ 无法存在多项式时间的$(3/2)$-近似算法。 □

令人惊讶的是，尽管这些负面结果即使在 SL-MAKESPAN 的实例受到严格限制时仍然成立，我们仍通过设计一种多项式时间 5-近似算法来克服这一难解性。作为我们 5-近似算法中的一个子程序（见算法1第1行），我们采用了 [39] 针对带容量约束且不带成本约束的广义指派问题（简称 GAPCC）提出的多项式时间 2-近似算法，其正式定义如下。值得注意的是， [39] 提出的多项式时间 2-近似算法考虑的是带额外成本约束的 GAPCC，但算法1并不需要这些约束，因此我们在问题定义中将其省略。

GAPCC

输入： 由作业 $\mathcal{J}$ 和无关并行机 $\mathcal{I}$ 构成的二分图 $G = (\mathcal{J}, \mathcal{I}, \mathcal{E})$，正整数 $M_1, \ldots, M_{|I|}$，以及对所有 $i \in \mathcal{I}$ 和 $j \in \mathcal{J}$ 均适用的非负整数作业处理时间 $p^\star_{ij}$。

目标： 在满足 G 的邻接约束的前提下，求完工时间最短的调度；其中每台机器 i 最多分配 M_i 个作业，对所有 $i \in \mathcal{I}$ 均成立。也就是说，当且仅当$(j, i) \in \mathcal{E}$时， $j \in \mathcal{J}$才能分配给 $i \in \mathcal{I}$。

值得强调的是，前述针对 GAPCC 的多项式时间 2-近似算法通过线性规划求解问题的分数松弛，然后通过迭代舍入获得整数解。现在，我们可以确立第一个主要理论贡献。

定理4. 存在一个针对 SL-MAKESPAN 的多项式时间 *5*-近似算法。

证明。 我们证明算法1是一个针对 SL-MAKESPAN 的多项式时间 5-近似算法，并首先证明其近似因子为 5。首先证明，当$r_j = \ell_j = r'_j = 0$ 对所有 $j \in \mathcal{J}$成立时，算法1生成的调度的完工时间不超过 [39]的 2-近似算法生成的调度的完工时间。对于一个适当的输入，考虑由 [39]的 2-近似算法给出的调度 S 。考虑机器 i：根据 S，它在时间 t 至时间 $t+x$ 期间空闲（其中 $t \geq 0$且 $x \geq 1$），并在时间 $t+x$ 至时间 $t+x+p_{ij}$处理作业 j，其中$p_{ij} \geq 1$。那么， S 的完工时间与调度 S'相同；该调度由 S 通过以下方式获得：改为让机器 i 在时间 t至时间 $t + p_{ij}$ 处理作业 j，并让机器 i 在时间 $t + p_{ij}$ 至时间 $t + x + p_{ij}$保持空闲。因此，任何调度 S' 的完工时间都不超过 S 的完工时间，只要它（1）将相同的作业分配给与 S相同的机器，（2）按照与 S相同的顺序处理这些作业，以及（3）确保每台机器在处理完分配给它的所有作业之前都不空闲。

现在，考虑机器 i，它按照调度 S'，在时间 t 至时间 $t + p_{ij}$处理作业 j，并在时间 $t + p_{ij}$ 至时间 $t + p_{ij} + p_{ij'}$处理作业 j'，其中 $t \geq 0$ 和$p_{ij}, p_{ij'} \geq 1$。那么， S' 的完工时间与调度S'' 相同；该调度由 S' 得到，具体做法是改为在机器 i上于时间 t 至时间 $t+p_{ij}$处理作业 j'，并于时间 $t+p_{ij'}$ 至时间 $t+p_{ij'}+p_{ij}$处理作业 j。因此，任何调度 S'' 的完工时间都至多与 S 相同，只要它（1）将相同的作业分配给与 S 相同的机器，并且（2）确保每台机器在处理完分配给它的所有作业之前都不空闲。注意，S'' 可以按任意顺序在这些机器上处理作业。

令 $\iota^\star$ 为一个 SL-MAKESPAN 实例。考虑另一个 SL-MAKESPAN 实例 ι，它与 $\iota^\star$ 相同，只是 $r_j = \ell_j = r'_j = 0$ 对所有 $j \in \mathcal{J}$。令 OPT 表示 ι的最优完工时间值。根据上述内容，算法1为 ι 给出的调度的完工时间为 k，因此 $k \leq 2 \cdot \mathrm{OPT}$，因为它采用了 [39]的 2-近似算法提供的可行客户端—辅助节点分配 $Y : \mathcal{J} \to \mathcal{I}$。事实上，上一段中的相同论证也证明

算法1：5-近似算法，用于 SL-MAKESPAN

输入： $G=(\mathcal{J},\ \mathcal{I},\ \mathcal{E})$、$M_1$、.....、$M_I$、$r_j$、$p_{ij}$、$\ell_j$、$p'_{ij}$和 r'_j，对所有 $i\in\mathcal{I}$和 $j\in\mathcal{J}$。

使用 G、M_i和 $p^\star_{ij}$，运行 [39, 定理2.3] 中针对 GAPCC 的多项式时间 2-近似算法$=p_{ij}+p'_{ij}$ 以所有 $i\in\mathcal{I}$ 和 $j\in\mathcal{J}$ 作为输入。该算法返回一个可行的客户端-辅助节点分配 $Y:\ \mathcal{J}\to\mathcal{I}$。

对于 $j=1:\ J$，**执行** // 初始化每个 $T4$ 可被处理的时间

  $w_j:=\infty$

结束

对于 $i=1:\ I$ **执行** // 为每个辅助节点 i 调度 $Z_Y(i)$ 中的客户端

  将 $Z_Y(i)$ 中的客户端按 ℓ_j 降序排序。令 Q 为该有序集合。

  将 $Z_Y(i)$ 中的客户端按 r'_j 降序排序。令 Q' 为该有序集合。

  $t:=0$ // 初始化时间

  当 $|Q|\neq 0$ 或 $|Q'|\neq 0$ **时执行** // 在 $Z_Y(i)$ 中调度客户端的 $T2$s 和 $T4$s

    $t=$最大值$\{t,$最小值$_{j\in Q,\ j'\in Q'}\{r_j,w_{j'}\}\}$ // 如果无法处理 $T2$ 或 $T4$，则调整 t

    如果 $|Q|\neq 0$ 且 $t\geq$最小值$_{j\in Q}\, r_j$，**则**

      将 $T2$ 分配给 Q 中索引最小且满足 $r_j\leq t$ 的客户端 j，对应时间区间为 S_t 至 $S_{t+p_{ij}}$ 在辅助节点 i上。

      $Q=Q\setminus\{j\}$ // 更新 Q，移除客户端 j的 $T2$

      $t=t+p_{ij}$ // 将 t 更新为客户端 j 的 $T2$ 完成的时间

      $w_j=t+\ell_j$ // 设置客户端 j 的 $T4$ 可被处理的时间

    end

    else

      将索引最小的客户端 j 中的 $T4$ 分配到时间区间 S_t，使得 $w_j\leq t$ 满足于 Q' $S_{t+p'_{ij}}$ 在辅助节点 i 上。

      $Q'=Q'\setminus\{j\}$ // 更新 Q'，移除客户端 j的 $T4$

      $t=t+p'_{ij}$ // 将 t 更新为客户端 j 的 $T4$ 完成的时间

      $c_j:=t+r'_j$ // 设置客户端 j的批次完成时间

    end

  end

结束

返回 最大值$_{j\in\mathcal{J}}\, c_j$

任务可以按照任何满足优先关系的顺序处理，而不会增加完工时间。

令 OPT* 为 $\iota^\star$的完工时间最优值。则，OPT* ≥OPT。令 $k^\star$ 为算法1针对 $\iota^\star$给出的完工时间。则，$k^\star\leq k+\max_{j\in\mathcal{J}} r_j+\max_{j\in\mathcal{J}}\ell_j+\max_{j\in\mathcal{J}} r'_j$，其中 $\max_{j\in\mathcal{J}} r_j$、$\max_{j\in\mathcal{J}}\ell_j$和 $\max_{j\in\mathcal{J}} r'_j$的值均相对于 $\iota^\star$而言。事实上，在算法1中，当仍有待处理的 $T2$，且需在 i 上处理时，辅助节点 i 可能处于空闲状态的最长累计时间至多为 $\max_{j\in\mathcal{J}} r_j$。此外，当仍有待处理的 $T4$，且需在 i 上处理，但没有待处理的 $T2$ 时，辅助节点 i可能处于空闲状态的最长累计时间至多为 $\max_{j\in\mathcal{J}}\ell_j$。最后，在辅助节点 i上处理完最后一个 $T4$ 后，完成分配给 i 的所有客户端的 $T5$所需的最长时间至多为 $\max_{j\in\mathcal{J}} r'_j$。

注意到以下三个不等式对 $\iota^\star$ 显然成立：(a) OPT* ≥ $\max_{j\in\mathcal{J}} r_j$，(b) OPT* ≥ $\max_{j\in\mathcal{J}}\ell_j$，以及 (c) OPT* ≥$\max_{j\in\mathcal{J}} r'_j$，其中 $\max_{j\in\mathcal{J}} r_j$、$\max_{j\in\mathcal{J}}\ell_j$和 $\max_{j\in\mathcal{J}} r'_j$ 的取值同样是相对于 $\iota^\star$ 而言的。因此，$k^\star\leq k+3\cdot$OPT$^\star\leq 2\cdot$OPT$+3\cdot$OPT$^\star\leq 5\cdot$OPT*，从而算法 1 是 SL-MAKESPAN 的 5-近似算法。

最后，我们确立算法1所需的运行时间界。首先请注意，2-近似算法 [39] 的运行时间为多项式时间，但由于将线性规划用作子程序 [40]，使用当前方法至少需要 $\mathcal{O}(n^{2+\varepsilon})$ 时间（其中 $\varepsilon>0$ 为常数，输入规模为 n）。对于算法 1 中的其余步骤，请注意，每个 FOR循环和 WHILE循环的迭代次数至多为 I 或 J。因此，这些循环的总体运行时间渐近上界为执行第 6–7 行所述排序步骤所需的时间。由于这些步骤位于前述 FOR循环中，第 1 行之后所有步骤所需的时间可上界为$\mathcal{O}(n^2\cdot\log n)$。因此，所提出算法的总体渐近运行时间界与第 1 [39]行调用已知调度算法所需的时间相匹配。

□

IV. GENSL-MAKESPAN：难解性与启发式算法

本节研究更一般的 GENSL-MAKESPAN 的计算复杂度，并为其设计一种启发式算法。与即使对于高度受限的实例（定理2），仅在单个辅助节点上调度任务也具有强NP-难这一事实类似，我们的第二项主要理论贡献证明：即使对于

限制极为严格的实例，仅为 GENSL-MAKESPAN 找到一个可行的客户端-辅助节点分配（形式化为 CH-ASSIGN 问题）也是强NP-难的。因此，除非P =NP，GENSL-MAKESPAN 不可能存在适用于任意近似因子的多项式时间近似算法。

定理 5. 即使将实例限制为 G是完全二分图，CH-ASSIGN 仍具有以下性质：(1) 即使 $I = 2$，仍是NP-困难的；(2) 以 I 为参数时，是W[1]-困难的；(3)即使对所有$i \in \mathcal{I}$和互不相同的 j、j'、j''、$j''' \in \mathcal{J}$，$M_i < d_j + d_{j'} + d_{j''} + d_{j'''}$，仍是强NP-困难的（即每个辅助节点最多可分配 *3* 个客户端）。

证明。我们通过从 $P \,||\, C_{\max}$ 到 CH-ASSIGN 的多项式时间归约来证明该定理。给定 $P \,||\, C_{\max}$的一个实例 $(\mathcal{J}^\star, \mathcal{I}^\star, k)$，我们按如下方式构造 CH-ASSIGN 的一个实例 $G = (\mathcal{J}, \mathcal{I}, \mathcal{E})$。对于所有 $i^\star \in \mathcal{I}^\star$，都有一个满足 $M_i = k$的辅助节点 $i \in \mathcal{I}$。对于所有 $j^\star \in \mathcal{J}^\star$，都有一个满足$d_j = p_{j^\star}$的客户端 $j \in \mathcal{J}$。最后，$\mathcal{J}$ 中的每个客户端都与 $\mathcal{I}$中的每个辅助节点相邻，即 G是完全二分图。至此，归约构造完成，且显然可在多项式时间内完成。

首先，假设对于实例（$\mathcal{J}^\star$，$\mathcal{I}^\star$，k），存在一个完工时间至多为 k 的调度 S。对于所有$j^\star \in \mathcal{J}^\star$，如果作业 $j^\star$ 在 S中被分配到机器 $i^\star \in \mathcal{I}^\star$，则将客户端 j 分配给辅助节点 i。令 $Y\colon \mathcal{J} \to \mathcal{I}$ 为相应的客户端—辅助节点分配。由于 G 是一个完全二分图，因此 Y 满足 G 的邻接约束。由于 S 的完工时间至多为 k，且客户端的内存需求与作业的处理时间相对应（即，对于所有 $j \in \mathcal{J}$，$d_j = p_{j^\star}$），因此，对于所有$i \in \mathcal{I}$，$\sum_{j \in Z_Y(i)} d_j \le k = M_i$。由此，$Y$ 也满足内存约束，因此，Y 是一个可行的客户端-辅助节点分配。

反过来，假设对于 CH-ASSIGN 的实例 $G =$（$\mathcal{J}$，$\mathcal{I}$，$\mathcal{E}$），存在一个可行的客户端-辅助节点分配 $Y\colon \mathcal{J} \to \mathcal{I}$。对于所有 $i \in \mathcal{I}$ 和 $j \in Z_Y(i)$，将作业 $j^\star$ 分配给机器 $i^\star$，并令 $Y^\star\colon \mathcal{J}^\star \to \mathcal{I}^\star$ 为相应的分配。对于所有 $i^\star \in \mathcal{I}^\star$，按照任意顺序将 $Z_{Y^\star}(i^\star)$ 中的作业安排在机器 $i^\star$ 上，并确保机器 $i^\star$ 不

t

直到处理完 $Z_{Y^\star}(i^\star)$ 中的每个作业后才空闲。令 S 表示相应的调度。由于 Y 是一个可行的客户端辅助

r

分配，因此对于所有 $i \in \mathcal{I}$，均有$\sum_{j \in Z_Y(i)} d_j \le M_i = k$。客户端的内存开销对应于作业的处理时间（即，对于所有 $j \in \mathcal{J}$，均有 $d_j = p_{j^\star}$），且 S 确保机器在处理完分配给它们的所有作业之前不会空闲，因此对于所有 $i^\star \in \mathcal{I}^\star$，均有$\sum_{j^\star \in Z_{Y^\star}(i^\star)} p_{j^\star} \le M_i = k$。由此，$S$ 是一个完工时间至多为 k 的调度。因此，定理的结论成立，因为 $P \,||\, C_{\max}$ （1）即使在 $\mathcal{I}^\star = 2$ [36] 时仍是弱 NP-难的；（2）以机器数量 [37], [38], 为参数时是 W[1]-难的；（3）即使对于所有 $j^\star \in \mathcal{J}^\star$ 都有 $k/4 < p_{j^\star} < k/2$，仍是强 NP-难的（ $P \,||\, C_{\max}$ 的这一特殊情形称为 3-划分问题） [36]。 □

鉴于这一负面结果，所能期待的最好结果就是一种启发式算法。在我们针对 SL-MAKESPAN 的近似算法基础上，我们为 GENSL-MAKESPAN 设计了一种启发式算法；这两种算法唯一的区别在于计算客户端—辅助节点分配的第一步（即算法1中的第1行）。这一步必须进行调整，因为即使只是为 GENSL-MAKESPAN 计算一个可行的客户端—辅助节点分配也是强 NP-难的（定理 5），也就是说，除非 P = NP，否则无法在多项式时间内完成。因此，对于该启发式算法，我们改用求解器来寻找一个可行的客户端—辅助节点分配，使所有辅助节点中分配给某一辅助节点的 T2任务和 T4任务之和的最大值最小；也就是说，求得一个客户端—辅助节点分配 Y，使其成为问题 $\min_Y \max^{i \in \mathcal{I}} \sum_{j \in Z_Y(i)} p_{ij} + p'_{ij}$ subject to Ybeing feasible 的解。换言之，我们的启发式算法将分配给辅助节点的客户端的 T2任务和 T4任务之和均衡分配，因此我们自然将其称为**均衡分布**（简称 **EquiD**）。下一节对 EquiD 的实验评估表明，这种均衡分配是获得高效且表现良好的启发式算法的关键。

V. 数值评估

本节评估 EquiDistributed（EquiD）在 GENSL-MAKESPAN 上的性能，并将其与基线方法及文献中的方法进行比较。实验表明，EquiD 通常能产生接近最优的输出，且其运行时间仅为求解器给出最优解所需时间的一小部分。我们首先介绍实验设置，随后阐述从实证分析中获得的见解。

A. 实验设置

数据集与异质性级别。在评估中，我们使用来自 [41] 的开源数据，其中包含每层、每批次的时间测量与内存需求。这些数据涉及使用两个不同的数据集（CIFAR-10 [44] 和 MNIST [45]），由 6 种不同的异构设备（笔记本电脑、虚拟机、RPi3、RPi4、Jetson GPU 和 Jetson CPU）对 ResNet101 [42] 和 VGG19 [43] 进行训练（对于 MNIST，仅前 4 种设备提供测量数据）。前两种设备作为辅助节点，其余为客户端。 [41] 中的数据在各层处理时间的差异以及前向/反向传播方面，与文献中其他分割学习测试平台的时间测量具有相似特征，例如 [7], [46], 。

我们根据上述数据创建问题实例，同时也创建合成数据。这些实例具有四种不同程度的异构性。第 1 级仅考虑具有相同切分层的 2 种不同类型的客户端设备，以及 2 种类型的辅助节点设备。由于切分层和设备类型决定了 T1任务到 T5任务的长度，因此这是一个相当同质的场景。第 2 级考虑具有相同切分层的所有不同客户端设备，以及 2 种类型的辅助节点设备。第 3 级与第 2 级相同，但切分层是随机选择的（第一切分层位于神经网络靠前的几层中，第二切分层位于靠后的几层中）。第 4 级与第 3 级相似，但测量数据采用合成数据：这些数据是随机选择的，但处于实际数据的取值范围内。在第 1 至 3 级中，内存需求和容量基于实际数据；而在第 4 级中，它们是在现有范围内均匀随机选取的。除非另有说明，设备与云端（辅助节点）之间的连接速度——它与内存需求共同决定 r_j，ℓ_j 和r'_j 的值——遵循 Akamai《互联网报告》 [47] 中的统计数据。

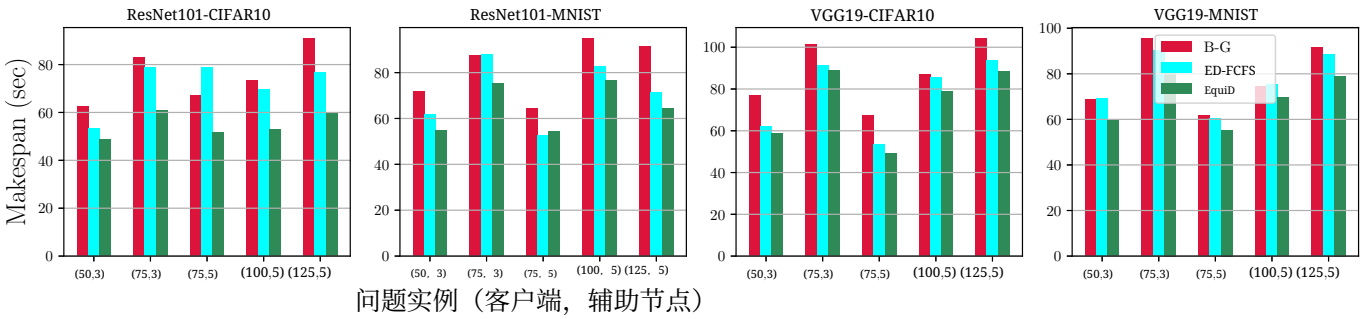

图2. 我们的算法EquiD、基线ED-FCFS以及来自 [14]的算法B-G在不同问题实例上取得的批次最大完工时间（单位：秒）。请注意，除一种情形外（ResNet101-MNIST，75个客户端和5个辅助节点），我们的算法EquiD在每种场景下计算出的完工时间都小于另外两种方法；在该例外情形中，仅ED-FCFS计算出略小的完工时间。

指标及与其他方法的比较。 我们主要关注训练完工时间指标（针对一个批次），同时也分析算法 EquiD 的执行时间。[4] 具体而言，我们将 EquiD 的性能（按计算得到的完工时间衡量）与以下方法进行比较：

- **B-G（均衡贪心）**：这是 [14] 中提出的方法。对于每个客户端，该方法会找到已分配客户端数量最少且有足够可用内存的辅助节点，并将该客户端分配给此辅助节点。随后，按照先到先服务（FCFS）策略进行调度。
- **ED-FCFS**：这是一个连接 EquiD 与 B-G 的基线方法。它以与 EquiD 相同的方式求解分配问题，但随后按照先到先服务的顺序调度客户端。

最后，由于文献中的其他方法（例如 I-B 小节中提到的方法）会联合决定通用分割学习框架的其他方面（如无线电频谱分配或设备聚类），因此无法与之进行比较。

B. 见解

首先，我们将 EquiD 与 GENSL-MAKESPAN 的最优解进行比较。表 I 报告了 ResNet101 和 CIFAR10 在不同实例规模（以 J 和 I 衡量）及异构性级别下的次优度。由于求解器寻找最优解耗时较长，只有规模较小的问题实例能够得到最优求解。[5] 因此，文中也给出了执行时间。最优解对应的执行时间是求解器 Gurobi [48]，所达到的时间，而我们观察到 SCIP 求解器 [49] 的执行时间大体相近。两种求解器均在一台普通笔记本电脑上运行。随着异构性增加，求解器寻找最优解最多需要 683 秒。总体而言，可以从表 I 的数据中得出以下观察结果。

我们的算法 *EquiD* 通常能将次优度控制在7.79% 以内（最坏情况下不超过19.77% ），且运行时间仅为求解器求得 GENSL-MAKESPAN 最优解所需时间的一小部分。

[4]我们在此强调，准确率等指标不在本研究的范围内，因为任务分配和调度决策不会影响模型的收敛。未来的工作可以探讨客户端掉线或客户端选择策略如何影响完工时间或模型收敛。

[5]我们采用 [14] 中的数学模型，时隙长度为 300 毫秒。

表 I 所提策略 EquiD 与最优解在次优度和执行时间方面的比较。

异构性层级	J	I	次优度 (%)	最优解执行时间 时间（秒）	EquiD 示例 时间（秒）
2	8	2	6.15	63.52	0.02
	10	2	0.00	42.82	0.02
	10	5	1.67	140.07	0.03
	12	2	7.79	94.07	0.02
	15	2	19.77	146.65	0.02
	15	5	2.78	449.84	0.02
3	8	2	2.10	71.21	0.01
	10	2	0.56	60.11	0.01
	10	5	2.86	136.54	0.03
	12	2	1.94	156.77	0.02
	15	2	0.00	241.97	0.01
	15	5	2.50	683.34	0.02

接下来，我们重点关注 EquiD 相较于其他方法的表现。图 2 展示了 EquiD、B-G 和 ED-FCFS 在不同问题实例下的完工时间。

EquiD 的表现始终优于 *B-G* 和 *ED-FCFS* 方法，完工时间最多可缩短34.6% （在 *ResNet101* 和 *CIFAR10* 的情况下）。

在大多数情况下，ED-FCFS 的表现优于 B-G，但不如 EquiD。这表明不同的客户端—辅助节点分配可能会影响完工时间，也说明在确定这些分配时考虑任务 $T2$ 和 $T4$的处理时间至关重要。此外，ED-FCFS 与 EquiD 之间的性能差距源于调度策略不同。忽略 T3任务和 T5任务长度的先到先服务调度可能不会优先处理较慢的客户端（拖尾客户端），从而导致完工时间更长。

在图 2 中，我们可以看到，与 ResNet101 相比，VGG19 的性能差距更小。这是因为训练 VGG19 时需要交换的激活值/梯度大小平均而言大于 ResNet101。因此，根据切分层的不同，通信延迟可能会很大，某些客户端可能会

非常缓慢，从而使完工时间在很大程度上与优化决策无关。因此，对于图 2 中针对 VGG19 开展的实验，客户端的连接性被选择在最快的连接范围内。

接下来，我们进一步比较 EquiD 与 B-G，重点考察任务异构性以及客户端和辅助节点数量的影响。图3展示了在不同异质性级别下，ResNet101 和 CIFAR10 场景中 B-G 得出的完工时间相较于 EquiD 得出的完工时间高出多少。随着客户端和辅助节点数量增加，相对差距也随之扩大（即比较粉色柱和灰色柱时）。这是因为客户端数量较多时，分配和调度决策需要优先考虑速度最慢的设备。EquiD 正是通过不仅依据 T2任务和 T4任务的长度，还依据 T3任务和 T5任务的长度进行调度来实现这一点的（算法1第6–7行）。

再看图3中的异质性级别：在低异质性级别下，相对差距最低可达25%，而在高度异构的场景中，这一差距则会大幅增加。

在高度异构的场景中，*B-G* 给出的完工时间比 *EquiD* 给出的大 70.4% 到117.8%。

为完成 EquiD 与 B-G 的比较，必须注意：只要存在解，EquiD 就保证能够给出一个解，即一个遵守 GENSL-MAKESPAN 约束的调度。然而，B-G 并非如此，因为它可能无法找到可行的客户端-辅助节点分配。一个简单的例子是：两个辅助节点的内存容量分别为 1 和 2，两个客户端的内存需求分别为 1 和 2。虽然在这种情况下显然存在可行的客户端-辅助节点分配，但 B-G 策略可能会将内存需求为 1 的客户端分配给内存容量为 2 的辅助节点，此时另一个客户端将无法被分配给任何辅助节点。

最后，在图 4 中，我们研究了在高度异构（即第 4 级）条件下，EquiD 在 ResNet101 和 MNIST 上的性能如何随客户端和辅助节点数量的变化而变化。正如预期，完工时间随客户端数量的增加而增加。当辅助节点数量减少时，情况也是如此。当客户端数量为 50 和 75 时，出现了这一观察结果的例外。在这些情况下，完工时间由某些慢速设备决定；尽管它们被分配给了快速辅助节点，但其训练时间并不受增加辅助节点的影响。这也是为什么，对于 5

辅助节点数量至少为 75 个时，完工时间保持稳定。作为最后一项观察，我们指出如下。随着辅助节点数量增加，完工时间最多可减少 21.8%。当客户端数量较多时，这一现象更加明显。

VI. 结论与未来工作

在这项工作中，我们研究了异质任务场景和同质任务场景下分割学习中最小化完工时间的问题，分别通过 SL-MAKESPAN 和 GENSL-MAKESPAN 中的内存约束来刻画。即使对于限制非常严格的实例，我们也证明了 SL-MAKESPAN 是强 NP难问题，且除非 P = NP，否则不存在多项式时间的（3/2）近似算法。尽管如此，我们

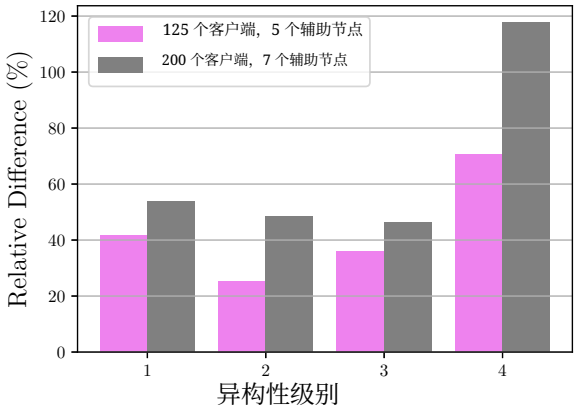

图 3. 不同异构性水平下，B-G 给出的完工时间比 EquiD 给出的完工时间大多少的相对差异。此处使用了 ResNet101 和 CIFAR10 的时间测量数据。

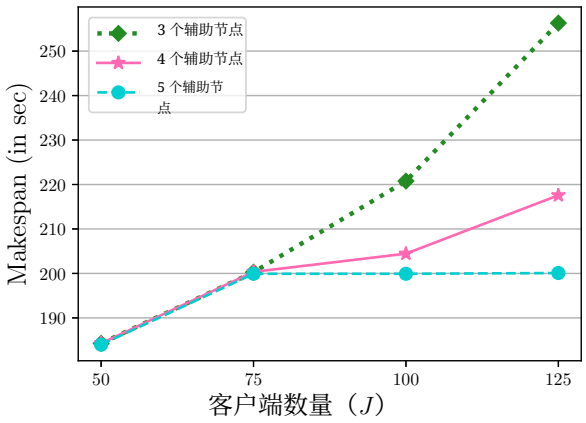

图 4. 客户端和辅助节点数量变化时 EquiD 达到的完工时间（使用 ResNet101 和 MNIST）。

通过为 SL-MAKESPAN 提供一个多项式时间的 5-近似算法，克服了这一难解性。对于 GENSL-MAKESPAN，我们证明，仅仅判定是否存在可行的客户端—辅助节点分配就是强 NP难问题，这意味着对于任何近似因子，都不存在多项式时间近似算法。尽管如此，我们基于自己的近似算法设计了一种名为 EquiD 的启发式算法，并通过大量实验表明，它优于文献中的其他启发式算法，而且只需求解器寻找最优解所需时间的一小部分，就能提供近乎最优的解。

未来的工作自然可以考虑在分割学习过程中纳入更多决策，例如确定切分层。然而，正如我们所证明的，即使问题的实例受到严格限制，也已经是困难问题，因此在这种情况下，很可能只能设计启发式算法。最有趣的未来研究方向之一，是为 SL-MAKESPAN 寻找多项式时间近似算法所能达到的最佳近似因子。由于我们的下界对其在 $R \mid\mid C_{\max}$ 上的限制情形也成立，因此，SL-MAKESPAN 的额外特性有望用于提高这一界限。

参考文献

[1] P. Vepakomma、O. Gupta、T. Swedish 和 R. Raskar，“医疗健康领域的分割学习：无需共享原始患者数据的分布式深度学习”，arXiv 预印本 arXiv:1812.00564，2018 年。[2] B. McMahan 、 E. Moore 、 D. Ramage、S. Hampson 和 B. A. y Arcas，“基于去中心化数据的深度网络通信高效学习”，载于第20届国际人工智能与统计会议（AISTATS）论文集，第1273–1282页，PMLR，2017年。[3] E. Samikwa、A. Di Maio 和 T. Braun，“ARES：面向物联网的自适应资源感知分割学习”，《计算机网络》，第218卷，109380页，2022年。[4] Z. Lin、G. Qu、X. Chen 和 K. Huang，“6G边缘网络中的分割学习”，IEEE无线通信，第31卷，第4期，第170–176页，2024年。[5] C. Thapa、P. C. M. Arachchige、S. Camtepe 和 L. Sun，“SplitFed：联邦学习与分割学习相遇”，载于第36届AAAI人工智能会议论文集，第36卷，第8485–8493页，2022年。[6] Y. Gao、M. Kim、C. Thapa、A. Abuadbba、Z. Zhang、S. Camtepe、H. Kim 和 S. Nepal，“物联网分布式机器学习技术的评估与优化”，《IEEE计算机汇刊》，第71卷，第10期，第2538–2552页，2021年。[7] Z. Wang、H. Xu、Y. Xu、Z. Jiang 和 J. Liu，“CoopFL：在异构边缘计算中通过深度神经网络划分与卸载加速联邦学习”，《计算机网络》，第220卷，109490页，2023年。[8] P. Han、C. Huang、G. Tian、M. Tang 和 X. Liu，“异构数据上的分割联邦学习收敛性分析”，《神经信息处理系统进展》（NeurIPS），第37卷，103544，2024年103476页。–。[9] Z. Xia、M. Hu、D. Yan、R. Liu、A. Li、X. Xie 和 M. Chen，“MultiSFL：通过多模型聚合与知识回放实现精准的分割联邦学习”，载于第39届AAAI人工智能会议论文集，第39卷，第914–922页，2025年。[10] C. Benzaid 和 T. Taleb，“面向5G及后续网络的人工智能驱动零接触网络与服务管理：挑战与研究方向”，《IEEE网络》，第34卷，第2期，第186–194页，2020年。[11] X. Lyu、S. Liu、J. Liu 和 C. Ren，“面向物联网数据驱动边缘智能的可扩展聚合分割学习”，《物联网杂志》，第6卷，第4期，第124–129页，2023年。[12] H. Hafi、B. Brik、P. A. Frangoudis、A. Ksentini 和 M. Bagaa，“面向6G赋能网络的分割联邦学习：需求、挑战与未来方向”，IEEE Access，第12卷，第9890–9930页，2024年。[13] J. Tirana、D. Tsigkari、G. Iosifidis 和 D. Chatzopoulos，“混合联邦分割学习中的训练完工时间最小化”，《IEEE移动计算汇刊》，第24卷，第6期，第5400–5417页，2025年。[14] J. Tirana、D. Tsigkari、G. Iosifidis 和 D. Chatzopoulos，“并行分割学习的工作流优化”，载于 INFOCOM 2024——IEEE计算机通信会议论文集，第1331–1340页，2024年。[15] M. Cygan、F. V. Fomin、L. Kowalik、D. Lokshtanov、D. Marx、M. Pilipczuk、M. Pilipczuk 和 S. Saurabh，《参数化算法》。施普林格，2015年。[16] W. Yu、H. Hoogeveen 和 J. K. Lenstra，“含延迟和单位时间操作的双机流水车间完工时间最小化问题是NP难的”，《调度杂志》，第7卷，第5期，第333–348页，2004年。[17] J. K. Lenstra、D. B. Shmoys 和 ´E. Tardos，“无关并行机调度的近似算法”，《数学规划》，第46卷，第259–271页，1990年。[18] ´Etienne Bamas、A. Lindermayr、N. Megow、L. Rohwedder 和 J. Schl¨oter，“圣诞老人问题、完工时间与拟阵：算法与归约”，载于2024年ACM-SIAM离散算法年度研讨会（SODA）论文集，第2829–2860页，2024年。[19] O. Svensson，“圣诞老人将作业调度到无关机器上”，《SIAM计算杂志》，第41卷，第5期，第1318–1341页，2012年。[20] O. Gupta 和 R. Raskar，“多个智能体上的深度神经网络分布式学习”，《网络与计算机应用杂志》，第116卷，第1–8页，2018年。[21] D. Pasquini、G. Ateniese 和 M. Bernaschi，“释放猛虎：针对分割学习的推理攻击”，载于ACM计算机与通信安全会议论文集，第2113–2129页，2021年。[22] Z. Zhang、A. Pinto、V. Turina、F. Esposito 和 I. Matta，“分割联邦学习中的隐私与通信效率”，《IEEE大数据汇刊》，第9卷，第5期，第1380–1391页，2023年。[23] Y. Liao、Y. Xu、H. Xu、L. Wang、Z. Yao 和 C. Qiao，“MergeSFL：结合特征合并与批量大小调节的分割联邦学习 ”，载于2024年IEEE第40届数据工程国际会议（ICDE）论文集，第2054–2067页，2024年。[24] J. Tirana、D. Tsigkari、D. S. Noguero 和 N. Kourtellis，《分割联邦学习中的数据异质性与遗忘标签》，即将发表于第40届AAAI人工智能会议论文集，2026年。[25] S. Wang、X. Zhang、H. Uchiyama 和 H. Matsuda，《HiveMind：迈向蜂窝原生机器学习模型拆分》，IEEE通信领域精选期刊，第40卷，第2期，第626–640页，2021年。[26] Z. Lin、W. Wei、Z. Chen、C.-T. Lam、X. Chen、Y. Gao 和 J. Luo，《分层拆分联邦学习：收敛性分析与系统优化》，IEEE移动计算汇刊，2025年。[27] X. Liu、Y. Deng 和 T. Mahmoodi，《无线无人机网络中混合拆分与联邦学习的节能用户调度》，载于 ICC 2022-IEEE 国际通信会议论文集，第1–6页，2022年。[28] M. Kim、A. DeRieux 和 W. Saad，《无线网络中个性化节能分割学习的讨价还价博弈》，载于2023年IEEE无线通信与网络会议（WCNC）论文集，第1–6页，2023年。[29] X. Liu、Y. Deng 和 T. Mahmoodi，《无线分布式学习：一种新的混合拆分与联邦学习方法》，IEEE无线通信汇刊，第22卷，第4期，第2650–2665页，2022年。[30] X. Liu、Y. Deng 和 T. Mahmoodi，《无线无人机网络中的新型混合拆分与联邦学习架构》，载于 ICC 2022- IEEE 国际通信会议论文集，第1–6页，2022年。[31] W. Wu、M. Li、K. Qu、C. Zhou、X. Shen、W. Zhuang、X. Li 和 W. Shi，《无线网络上的分割学习：并行设计与资源管理》，IEEE通信领域精选期刊，第41卷，第4期，第1051–1066页，2023年。[32] Z. Lin、G. Zhu、Y. Deng、X. Chen、Y. Gao、K. Huang 和 Y. Fang，《资源受限无线边缘网络上的高效并行分割学习》，IEEE移动计算汇刊，第23卷，第10期，第9224–9239页，2024年。[33] B. Huang、H. Zhao、L. Wang、W. Qian、Y. Yin 和 S. Deng，《面向分割与联邦学习的去中心化主动模型卸载与资源分配》，IEEE物联网期刊，2024年。[34] Q. Weng、W. Xiao、Y. Yu、W. Wang、C. Wang、J. He、Y. Li、L. Zhang、W. Lin 和 Y. Ding，《真实环境中的机器学习即服务：大规模异构GPU集群的工作负载分析与调度》，载于第19届USENIX网络系统设计与实现研讨会（NSDI）论文集，第945–960页，2022年。[35] Google AI，《联邦学习》，2025年。https://federated.withgoogle.com。[36] M. R. Garey 和 D. S. Johnson，《计算机与难解性：NP完全性理论指南》。W. H. Freeman，1979年。[37] K. Jansen、S. Kratsch、D. Marx 和 I. Schlotter，《重新探讨固定箱数的装箱问题》，《计算机与系统科学杂志》，第79卷，第1期，第39–49页，2013年。[38] D. Knop 和 M. Kouteck´y，《调度与n重整数规划相遇》，《调度杂志》，第21卷，第5期，第493–503页，2018年。[39] B. Saha 和 A. Srinivasan，《资源分配问题的一种新型近似技术》，《随机结构与算法》，第52卷，第4期，第680–715页，2018年。[40] S. Jiang、Z. Song、O. Weinstein 和 H. Zhang，《一种求解一般线性规划的更快算法》，载于第53届ACM SIGACT计算理论研讨会（STOC）论文集，第823–832页，2021年。[41] Github，2024年。https://github.com/jtirana98/SFL-workflow-optimization。[42] K. He、X. Zhang、S. Ren 和 J. Sun，《用于图像识别的深度残差学习》，载于2016年IEEE计算机视觉与模式识别会议（CVPR）论文集，第770–778页，2016年。[43] K. Simonyan 和 A. Zisserman，《用于大规模图像识别的超深卷积网络》，arXiv预印本 arXiv:1409.1556，2014年。[44] A. Krizhevsky 和 G. Hinton，《从微型图像中学习多层特征》，技术报告，多伦多大学，2009年。[45] Y. LeCun、L. Bottou、Y. Bengio 和 P. Haffner，《应用于文档识别的基于梯度的学习》，《IEEE学报》，第86卷，第11期，第2278–2324页，2002年。[46] H. Hafi、B. Brik、M. Bagaa 和 A. Ksentini，《神经网络深度对低资源无人机网络中拆分联邦学习性能的影响》，载于国际无线通信与移动计算会议（IWCMC）论文集，第1290–1295页，IEEE，2024年。[47] D. Belson，《2016年第四季度互联网状况报告》，Akamai Technologies，第9卷，第4期，2017年。[48] Gurobi Optimization LLC，《Gurobi优化器参考手册》，2024年。[49] S. Bolusani、M. Besanc.on、K. Bestuzheva、A. Chmiela、J. Dion´ısio、T. Donkiewicz、J. van Doornmalen、L. Eifler、M. Ghannam 等，《SCIP优化套件9.0》，arXiv预印本 arXiv:2402.17702，2024年。