https://crad.ict.ac.cn
x(x): xx–xx, xxxx

边缘智能计算系统中加速推荐模型训练的样本调度机制

李国鹏[1] 谈海生[2] 张 弛[1] 倪宏秋[1] 王子龙[1] 章馨月[1] 徐 洋[1] 田 晗[1] 陈国良[1]
[1]（中国科学技术大学计算机科学与技术学院 合肥 230027）
[2]（中国科学技术大学人工智能与数据科学学院 合肥 230027）
（guopengli@mail.ustc.edu.cn）

Samples Dispatching Mechanism for Accelerating Recommendation Model Training in Edge Intelligent Computing System

Li Guopeng[1], Tan Haisheng[2], Zhang Chi[1], Ni Hongqiu[1], Wang Zilong[1], Zhang Xinyue[1], Xu Yang[1], Tian Han[1], and Chen Guoliang[1]
[1]（*School of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei* 230027）
[2]（*School of Artificial Intelligence and Data Science*, *University of Science and Technology of China*, *Hefei* 230027）

Abstract Training deep learning recommendation models (DLRMs) using edge workers in edge intelligent computing system brings several benefits, particularly in terms of data privacy protection, low latency and personalization. However, due to the huge size of embedding tables, typical DLRM training frameworks adopt one or more parameter servers to maintain global embedding tables, while leveraging several edge workers to cache part of them. This incurs significant transmission cost for embedding transmissions between workers and parameter servers, which can dominate the training cycle. In this paper, we investigate how to dispatch input embedding samples to appropriate edge workers to minimize the total embedding transmission cost when facing edge-specific challenges such as heterogeneous networks and limited resources. We develop ESD, a novel mechanism that optimizes the dispatching of input embedding samples to edge workers based on expected embedding transmission cost. We propose HybridDis as the dispatch decision method within ESD, which combines a resource-intensive optimal algorithm and a heuristic algorithm to balance decision quality and resource consumption. We implement a prototype of ESD using C++ and Python and compare it with state-of-the-art mechanisms on real-world workloads. Extensive experimental results show that ESD reduces the embedding transmission cost by up to 36.76% and achieves up to 1.74x speedup in end-to-end DLRM training.

Key words distributed training; edge intelligence; deep learning; recommendation model; dispatch mechanism

摘 要 在边缘智能计算系统中使用边缘工作节点训练深度学习推荐模型（DLRMs）具有诸多优势，尤其是在数据隐私保护、低延迟和个性化推荐等方面. 然而，由于嵌入表的规模庞大，在训练 DLRM 时通常采用一个或多个参数服务器来维护全局嵌入表，同时利用多个边缘节点缓存嵌入表的一部分. 在此架构下，需要在边缘节点和参数服务器间传输嵌入以保证嵌入数据一致性，嵌入传输代价通常主导了训练周期. 本文旨在研究在边缘智能计算系统中，当面对异构网络和资源受限等挑战时，如何将嵌入样本调度到合适的边缘节点上进行训练，以最小化总嵌入传输代价. 为此，本文提出了一个基于预期嵌入传输代价的嵌入样本调度机制 ESD.在 ESD 中，本文设计了一个结合资源密集型最优解法和启发式解法的调度决策

收稿日期：2025-03-01；**修回日期**：2025-04-07
基金项目：国家自然科学基金重点项目（62132009）
This work is supported by the Key Program of the National Natural Science Foundation of China (62132009).
通信作者：谈海生（hstan@ustc.edu.cn）

方法 HybridDis, 以实现决策质量和资源消耗之间的平衡. 本文使用 C++和 Python 实现了 ESD 的原型系统, 并在真实工作负载下将其与现有最先进的机制进行比较. 大量实验结果表明, ESD 可将嵌入传输代价至多降低 36.76%, 并且在端到端 DLRM 训练速度上实现了最高 1.74 倍的加速[1].

关键词 分布式训练; 边缘智能; 深度学习; 推荐模型; 调度机制

中图法分类号 TP303;TP393

DOI: 10.7544/issn1000-1239.202550128 **CSTR:** 32373.14.issn1000-1239.202550128

推荐系统已成为日常生活中的重要组成部分, 其通过在海量数据上训练推荐模型来学习用户偏好和物品特征, 从而为电子商务[1-3]、在线娱乐[4-6]及社交媒体[7-9]等领域提供个性化推荐服务, 有效提升用户体验并驱动业务增长[10-11]. 近年来, 随着移动计算和互联网的快速发展, 将推荐系统集成到物联网应用中以提供个性化推荐服务已成为一种发展趋势, 如能源互联网中的能源管理推荐[12-13]、健康监测中的健康建议[14], 以及智慧旅游中的目的地智能推荐[15]. 然而, 传统基于云数据中心的推荐系统面临着双重挑战: 一方面, 集中式数据存储存在隐私泄露风险; 另一方面, 网络传输延迟可能影响服务质量[16-19]. 为此, 学术界和工业界近年来聚焦于将推荐模型的训练和部署从云数据中心迁移至边缘智能计算系统中, 以在保障隐私数据安全和低延迟的前提下为物联网应用提供个性化推荐服务[20-25]. 对于在边缘智能计算系统中部署推荐模型以提供推荐服务, 文献 [20] 对模型推理过程中的流水线进行调度以提高吞吐量, 为了降低推荐延迟, 文献 [21] 提出了边缘推荐系统 EdgeRec, 其将淘宝的商品详情页访问量提高了 9%. 对于在边缘智能计算系统中训练推荐模型, 文献 [23] 利用边缘节点进行分布式参数聚合, 文献 [24] 引入边缘节点存储隐私数据, 文献 [25] 利用边缘节点训练推荐模型, 并设计了边缘节点选择算法来优化模型准确率.

近年来, 基于深度学习的推荐模型(deep learning recommendation models, DLRMs), 如 WDL[26] 和 DFM[27], 已成为构建推荐系统的核心技术[28]. DLRM 主要由嵌入层和多层感知机(MLP)层组成. 嵌入层是一个大规模的嵌入表, 它将稀疏输入(例如用户 ID 和视频 ID)转换为嵌入(向量)[29-31]. 一般情况下, 嵌入表的存储空间需求可达数十 GB 到 TB, 而 MLP 层通常只需要几 MB 到几百 MB 的存储空间[32-33]. 因此, 在训练 DLRM 时, 一般采用参数服务器架构(PS)来处理占用大量内存的嵌入表. 具体来说, 在 PS 架构中, 使用一台或多台拥有大量内存的服务器(即参数服务器)维护全局嵌入表, 在配有 GPU 的工作节点中缓存部分嵌入表(即*嵌入缓存*). 在 DLRM 训练过程中, 工作节点会从参数服务器拉取嵌入以及向参数服务器推送嵌入梯度, 这一过程被称为*嵌入传输*. 在实际生产负载中, 用于 DLRM 训练的单个训练输入样本中可涉及数千个嵌入[34], 先前的研究和本文实验均表明, 嵌入传输代价是 DLRM 训练过程中的瓶颈[35]. 例如, 本文使用 Criteo Kaggle 数据集[36] 训练 WDL[26] 时, 高达 90% 的时间用于嵌入传输, 主导了整个训练周期.

本文目标是通过降低总嵌入传输代价来加速在边缘智能计算系统中训练 DLRM. 对从数据加载器加载到每个工作节点的嵌入样本进行管理是减少嵌入传输代价的有效方法[37-39]. 具体而言, 利用现今常用的数据加载器中的输入样本预取功能[40-41], 可以在当前训练迭代期间应用一个调度机制, 将下一个迭代的输入嵌入样本调度到合适的工作节点上, 以降低嵌入传输代价. 直觉上, 调度机制应专注于提高工作节点上的缓存命中率, 然而, 不同于传统的文件缓存, 在 DLRM 训练期间, 工作节点会与参数服务器间进行嵌入传输, 即存储在工作节点上的嵌入不是静态的, 这导致缓存命中率的高低无法直接对应到总嵌入传输代价的大小, 这给设计样本调度机制带来了挑战. 此外, 边缘智能计算系统中的异构网络和资源受限等特点也带来了挑战. 本文将问题挑战总结如下:

1)代价组成. 本文采用不牺牲模型精度的批量同步并行方法(bulk synchronous parallel, BSP)[42] 对 DLRM 进行训练. 在每个训练迭代开始之前, 每个工作节点的嵌入缓存中应确保存储有每个输入样本对应嵌入的最新版本. 如果嵌入的最新版本在缓存中不可用, 则必须从参数服务器拉取(未命中拉取, Miss Pull). 对于每次迭代, 工作节点和参数服务器之间会按需同步嵌入梯度(更新推送, Update Push). 此外, 当达到嵌入缓存容量时, 为了存储新的嵌入, 须将部分嵌入替换出去, 它们的梯度也需要被推送到参数服务器上(逐出推送, Evict Push). 因此, 嵌入传输代价来自 Miss Pull、Update Push 和 Evict Push 这 3 个操作. 所以, 样本调度机制的设计应考虑上述操作带来的

样本传输代价，而不是仅专注于优化命中率.

2)异构网络. 在边缘智能计算系统中训练DLRM时，工作节点与参数服务器之间的连接带宽一般显著低于云数据中心中所用网络的带宽(例如100 Gbps以太网或InfiniBand)，此外，多个边缘工作节点通常通过异构网络连接到参数服务器. 例如，一些工作节点使用1 Gbps网络，而其他工作节点则通过5 Gbps网络连接到参数服务器，这导致不同节点与参数服务器间传输相同大小的嵌入的代价不同[43-45]. 因此，对于嵌入传输代价，除了考虑Miss Pull、Update Push和Evict Push操作的数量外，还需要考虑工作节点与参数服务器之间的网络带宽.

3)资源受限. 在生产环境中，每次迭代通常涉及成千上万个嵌入样本，为这些样本做出调度决策会产生额外的资源消耗. 在边缘智能计算系统中，通常存在资源争用现象，即会有多个任务争用有限的资源[46-47]. 所以，本文应对样本调度机制带来的额外的资源消耗进行考虑. 此外，本文专注于能及时准确地捕捉用户兴趣漂移和新兴趋势的在线DLRM训练[33,48]，这要求调度决策必须在每次训练迭代内完成，以避免性能下降. 因此，在设计调度机制时，对调度机制的资源消耗和求解延迟的考虑是十分重要的.

先前的工作主要集中于降低在云数据中心中训练DLRM时的嵌入传输代价，未对边缘智能计算系统中的实际挑战加以考虑[49-50]. 为解决上述实际挑战，本文提出了嵌入样本调度机制ESD，旨在将嵌入样本调度到合适的边缘工作节点上进行训练以减少总嵌入传输代价. ESD首先为每个样本计算在不同边缘工作节点上进行训练的预期传输代价，再基于预期传输代价将样本调度到合适的节点上. 为避免决策时间超过每个迭代的训练时间，同时平衡决策质量和资源消耗，本文提出了HybridDis作为ESD中的调度决策方法，它结合了资源密集型最优解法和资源高效的启发式解法. 本文主要技术贡献有3点：

1)为在边缘智能计算系统中加速DLRM训练，本文首先形式化了DLRM训练过程中对嵌入样本进行调度以最小化总嵌入传输代价这一问题，本文是首个在边缘智能计算系统中优化嵌入传输代价的工作.

2)本文提出了ESD，一个通过计算将每个嵌入样本调度到不同工作节点上进行训练的预期代价来减少总嵌入传输代价的嵌入样本调度机制. 本文在ESD中提出了HybridDis，一个混合调度决策方法，旨在平衡决策质量和资源消耗，以应对资源受限挑战.

3)本文使用C++(包括CUDA)和Python语言实现了ESD原型系统，并在8个边缘工作节点上使用现实世界工作负载进行了广泛的实验评估. 评估结果显示，与LAIA[37]相比，ESD可减少高达36.76%的嵌入传输代价，在端到端DLRM训练速度上可实现最高1.74倍加速.

1 背景知识

针对本文的研究主题，本节将对边缘智能计算系统中的推荐模型、深度学习推荐模型系统架构和深度学习推荐模型训练过程中的嵌入传输代价的背景知识进行简要介绍.

1.1 边缘智能计算系统中的推荐模型

推荐系统现已成为电子商务和社交媒体等应用中不可或缺的一部分，其可提供个性化推荐以增强用户体验并推动业务增长[51-52]. 通常，企业在云数据中心中对推荐模型进行训练和部署，云服务器基于所有用户数据训练模型，并在接收到推理请求时将推荐结果推送到用户的设备上. 尽管云数据中心可以提供近乎无限的计算能力，但其面临包括高资源消耗[53]、公有网络访问依赖[54]以及用户隐私泄漏[55-58]等问题在内的挑战，这些问题降低了基于云数据中心训练和部署DLRM的可用度[59]. 随着边缘计算的兴起，在边缘智能计算系统中训练和部署推荐模型成为了新兴趋势. 该模式在确保隐私保护和低延迟的前提下可为物联网应用(如能源互联网[60]、健康监测[61]以及智慧旅游[15])提供推荐服务. 例如，在能源互联网中，推荐系统可以提出能源管理解决方案并协助资源分配；在健康监测中，根据医疗历史和实时健康信息提供个性化的健康建议. 为保障数据安全，本文所用边缘工作节点和参数服务器使用隔离外部互联网的局域网进行连接. 这种方法使隐私数据保持本地化，在防止泄露的同时能够提供低延迟的推荐服务. 此外，本文专注于在线DLRM训练，在系统的初始化阶段，DLRM模型是基于由物联网设备(如摄像头和传感器)生成的历史数据上进行训练的，为了在推荐服务期间持续提供高质量的推荐，DLRM模型使用来自物联网设备的流数据在线连续训练和优化[62-67]. 总的来说，本文聚焦于利用连接在同一局域网内的边缘工作节点和参数服务器进行在线DLRM训练，为物联网应用提供低延迟、隐私保护的个性化推荐服务.

1.2 深度学习推荐模型系统架构

深度学习推荐模型，如WDL[26]、DFM[27]、DCN[68]

和 FLEN[69], 将推荐决策建模为预测特定事件发生概率的问题, 例如预测网络观众观看推荐内容(视频或音乐)的可能性. DLRM 主要由嵌入层、特征交互层和多层感知机(MLP)层组成, 如图 1 所示. DLRM 的输入通常既包括稠密输入(如年龄和身高)也包括稀疏输入(如用户 ID 和音乐 ID), 这两种输入分别通过 MLP 层和嵌入层进行处理.

1)嵌入层. 嵌入层实际上是由嵌入表组成的, 每个嵌入表代表一个分类特征, 而表中的每一行代表一个特定的 ID(例如用户 ID 或音乐 ID). 嵌入表将每个 ID 转换为固定大小的向量, 即嵌入, 其中包含可训练的浮点数值.

2)多层感知机(MLP)层. MLP 层执行多个全连接操作以处理稠密输入并处理来自特征交互层的输出. 它为单个用户-项目对提供点击率的概率, 还包括了其他计算稠密型单元, 如批量归一化等.

3)特征交互层. 特征交互是组合稠密特征和稀疏特征以捕捉它们之间复杂且非线性关系的过程[30]. 特征交互的输出将由后续的 MLP 层进行处理. 特征交互的操作包括平均、求和等.

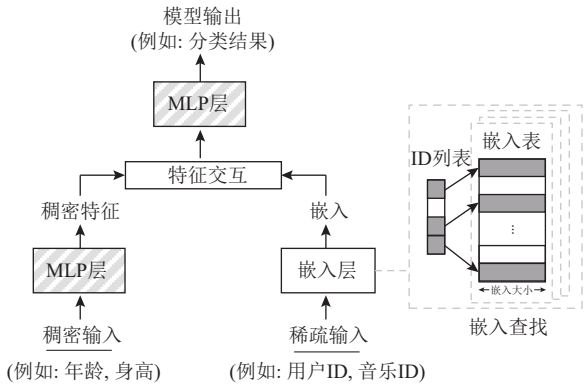


Fig. 1 Architectural of deep learning recommendation model system

图 1 深度学习推荐模型系统架构

1.3 深度学习推荐模型训练过程中的嵌入传输代价

在 DLRM 训练中, 可在嵌入层和 MLP 层使用不同的通信方式. MLP 层所占内存较少, 可以被单个工作节点所容纳, 因此, 可在每个工作节点上都保留一份 MLP 层的副本, 并使用全局规约(AllReduce)进行通信. 另一方面, 嵌入层占用内存较多, 可达 TB 级, 因此将全局嵌入表存储在一个单独的工作节点上是不可行的. 为此, 采用 PS 架构, 即在拥有较多内存参数服务器中维护全局共享的嵌入表, 而边缘工作节点则在其本地内存中缓存部分嵌入表, 即使用 PS 通信[70]. 然而, 在这种设计之上应用不牺牲模型精度的批量同步并行(BSP)时, 引入了高昂的嵌入传输代价[71]. BSP 训练是一种并行计算模型, 它将计算组织成同步步骤, 每个步骤包括局部计算、工作节点间的通信和全局同步屏障[42]. 在每次训练迭代中, 工作节点从参数服务器拉取嵌入, 并根据需要将嵌入梯度推送回参数服务器. 在按照第 5 节所示默认设置进行的实验中, 嵌入传输代价主导了训练周期, 可占端到端训练时间的 90%. 先前工作通过减少工作节点与参数服务器之间的嵌入传输次数[37,38,49,50] 或降低每次嵌入传输的代价[35,39,72,73] 来减少嵌入传输代价. 本文面向代价组成、异构网络和资源受限等挑战, 旨在最小化边缘智能计算系统中 DLRM 训练期间的总嵌入传输代价, 以加速 DLRM 训练. 本工作与前者类似, 并与后者研究正交.

2 系统模型和问题形式化

本节将介绍系统模型和问题的形式化描述. 表 1 列出了本文常用符号.

Table 1 List of symbols

表 1 符号列表

符号	描述
$\mathcal{W}$	边缘工作节点集合
$\mathcal{E}_i$	迭代I_i的输入嵌入样本, $\mathcal{E}_i=\{E_1,E_2,\cdots,E_{m\times n}\}$
I_i	第i个训练迭代
m	每个工作节点的批量大小
E_i	一个嵌入样本, $E_i=\{x_1,x_2,x_3,\cdots\}$
x_i	一个嵌入样本的 ID
$\boldsymbol{Emb}(x_i)$	嵌入样本 ID 为x_i对应的嵌入值(向量)
D_{tran}	一个嵌入的数据量
B_w^j	工作节点w_j和参数服务器间的网络带宽
T_{tran}^j	工作节点w_j和参数服务器间传输一个嵌入的代价, $T_{tran}^j=\frac{D_{tran}}{B_w^j}$

2.1 系统模型

1)边缘智能计算系统. 本文基于一个包含多个边缘 GPU 工作节点(简称“工作节点”)和一个参数服务器(PS)的边缘智能计算系统展开研究. 具体而言, 该系统由n个工作节点组成, 记为 $\mathcal{W}=\{w_1,w_2,\cdots,w_n\}$. 这些工作节点通过以太网连接到参数服务器, 带宽为 5 Gbps 或 0.5 Gbps.

2)训练输入样本. 在深度学习推荐模型训练中, 每个工作节点都关联一个数据加载器, 其是一个负责预处理训练样本的 CPU 进程. 数据加载器可从本地内存、磁盘或网络流中加载训练样本. 对于每个训

练迭代I_t，数据加载器将一批输入样本划分成多个微批次分配给每个工作节点. 为了保持各工作节点之间的工作负载平衡，本文使用m来表示每个工作节点的批量大小. 因此，对于n个工作节点，每次迭代的总训练输入样本数为$m\times n$. 在$m\times n$个输入嵌入样本$\mathcal{E}$中，$\forall E_i\in\mathcal{E}$都是一个嵌入样本，其包含多个嵌入 ID，即$E_i=\{x_1,x_2,\cdots,\}$. ID 类型的特征是对大规模类别信息的稀疏编码. 本文使用$\boldsymbol{Emb}(\boldsymbol{x}_i)$来表示$x_i$对应的嵌入值(向量).

3)嵌入缓存. 当在边缘工作节点上训练 DLRM 时，每个工作节点存有 MLP 的一个副本，并在训练过程中使用全局规约进行同步. 所有的嵌入存储在参数服务器上的全局嵌入表中，而每个工作节点则缓存一部分嵌入表. 工作节点通过与 PS 传输嵌入来管理本地缓存，以保证本地嵌入表和全局嵌入表之间的一致性. 本文使用r来表示缓存比例，即缓存中的嵌入数量与总嵌入数量的比例.

4)工作节点和参数服务器间的按需同步. 在 DLRM 的 BSP 训练中，在工作节点上，每次训练迭代都需要存有所需嵌入的最新版本，为此，工作节点与参数服务器间频繁传输嵌入值和梯度. 按需同步是一种减少传输代价的方法. 具体来说，在迭代I_t之后，不是将

所有嵌入梯度都推送到 PS，而是在I_{t+1}中使用按需推送. 例如，如果在迭代I_t期间，嵌入$\boldsymbol{Emb}(\boldsymbol{x}_i)$在工作节点$w_j$上进行了训练，并且在$I_{t+1}$中没有其他工作节点($w'\in\mathcal{W},w'\neq w_j$)接收到包含$x_i$的样本，则工作节点$w_j$不需要将嵌入$\boldsymbol{Emb}(\boldsymbol{x}_i)$梯度推送 PS. 相反，如果其他工作节点在$I_{t+1}$中需要$\boldsymbol{Emb}(\boldsymbol{x}_i)$，则工作节点$w_j$需要将$\boldsymbol{Emb}(\boldsymbol{x}_i)$推送到 PS，而其他工作节点将其从 PS 拉取到它们的缓存中. 因此，当使用按需同步时，每个工作节点在从数据加载器接收训练样本后的每次迭代的过程如下：按需嵌入梯度推送，拉取本地缓存中不存在的最新嵌入，前向传播，反向传播，稠密参数全局规约同步.

5)模型一致性分析. 本部分将通过理论分析阐述在 BSP 中使用按需同步和对样本进行调度不会影响模型的准确性. 相比于普通的分布式训练，对嵌入进行调度引入了两个主要变化：特定的输入嵌入样本调度机制(相对于随机调度机制)和按需同步(相对于训练迭代后将每个嵌入的梯度都推送到参数服务器上). 在 BSP 中，按需同步确保所有工作节点在即将到来的训练迭代中使用最新的嵌入，这与普通分布式训练的行为相同. 接下来证明 BSP 下对样本进行调度不会影响模型训练. 考虑到参数优化器遵循 SGD 算法[74]，对于给定批次的 $m\times n$个训练样本，模型权重y的梯度计算公式如下：

$$\nabla_y=\frac{1}{m\times n}\sum_{i=1}^{m\times n}\frac{\partial L(\mathcal{E}_i,y)}{\partial y}\tag{1}$$

其中，$\mathcal{E}_i$表示该批次中的第i个训练样本，L表示损失函数. 根据式 (1)，批次的梯度计算为每个训练样本的个体梯度之和. 由于个体梯度仅依赖于输入样本和当前模型权重，并且这些权重在本迭代开始前已在 BSP 下同步，因此将批次划分为n个微批次不会影响梯度结果.

$$\frac{1}{m\times n}\sum_{i=1}^{m\times n}\frac{\partial L(\mathcal{E}_i,y)}{\partial y}=\frac{1}{m\times n}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{\partial L\left(E_{ij},y\right)}{\partial y}\tag{2}$$

对于式(2)，其中，E_{ij}是第i个微批次中的第j个训练样本(对于工作节点w_i). 因此，由调度机制生成的调度结果在整个训练过程中都保留了与 BSP 相同的梯度，并收敛到相同的模型.

2.2 问题形式化

本文的目标是减少推荐模型训练中的嵌入传输代价. 如图 2 所示，在 DLRM 训练中，使用按需同步时，嵌入传输的代价来自未命中拉取、更新推送和逐出推送. 在图 2 中，为了简单起见，本文使用数字来表示嵌入 嵌入 ID 和嵌入值.

1)未命中拉取(miss pull). 当从数据加载器接收到训练样本时，工作节点应该从本地缓存或 PS 中检索必要的嵌入，称为嵌入查找. 如果本次迭代所需的嵌入已在本地缓存中并且是最新版本，则无需从 PS 中拉取嵌入. 在缓存的语境中，这是缓存命中. 相反，如果所需的嵌入不在本地缓存中或者已经过期(即不是最新版本)，则需要从 PS 中拉取，即导致缓存未命中. 缓存未命中会导致额外的嵌入传输代价，称为未命中拉取(Miss Pull)，例如，在图 2 中，迭代I_2中拉取 $\boldsymbol{Emb}(\boldsymbol{x}_8)$、$\boldsymbol{Emb}(\boldsymbol{x}_9)$、$\boldsymbol{Emb}(\boldsymbol{x}_{10})$ 和 $\boldsymbol{Emb}(\boldsymbol{x}_{11})$.

2)更新推送(update push). 在使用按需同步时，梯度推送发生在每次迭 代的开始而不是每次迭代的结束. 对于一个工作节点，当接收到训练样本时，如果在前一次迭代中某些嵌入在其他工作节点上进行了训练，其他工作节点应将嵌入梯度更新推送到参数服务器. 例如在图 2 中的迭代I_2，w_3应将 x_9的嵌入梯度推送到参数服务器，以便在工作节点 w_2上对其进行训练.

3)逐出推送(Evict Push). 除了缓存未命中导致的 Miss Pull 和按需同步引入的 Update Push 之外，当

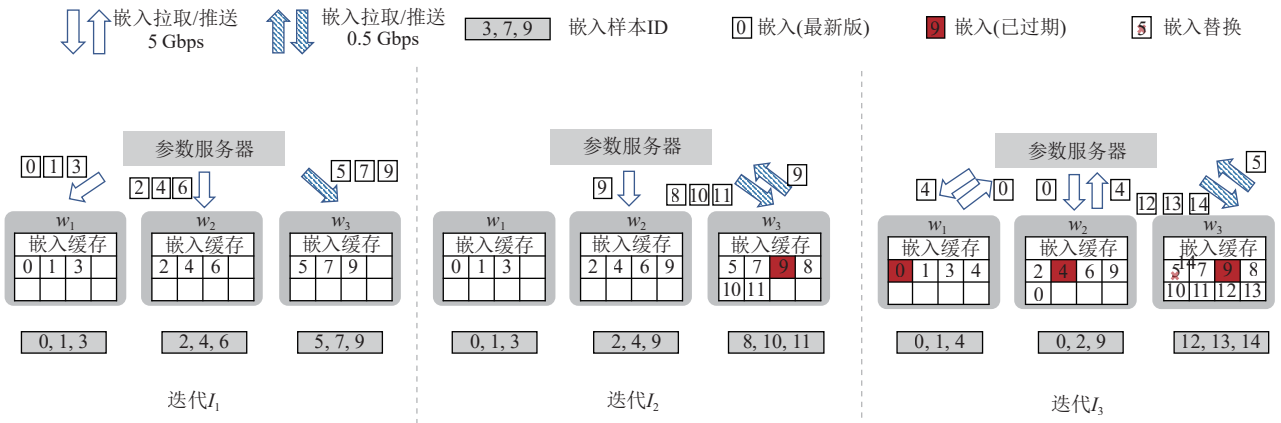


Fig. 2 Miss Pull, Update Push, and Evict Push transmission operations in DLRM training

图 2 深度学习推荐模型训练过程中的 Miss Pull、Update Push 和 Evict Push 操作

使用按需同步时，如果工作节点的缓存已满，则需要为新的嵌入留出存储空间. 在缓存替换（或逐出）期间，如果尚未同步的嵌入被替换出去，则需要将其梯度更新到 PS. 这被称为 Evict Push. 例如图 2 中迭代 I_3 将 $\boldsymbol{Emb}(\boldsymbol{x}_5)$ 替换出去，需要把 x_5 的梯度推送到参数服务器上.

本文的目标是最小化所有迭代的总嵌入传输代价. 对于每次嵌入传输，本文假设每次嵌入拉取和推送的数据大小相同，用 D_{tran} 来表示. 对于迭代 I_t，嵌入传输的数量为 T^t_{num}. 当网络带宽为 B_w 时，每个嵌入的传输代价 $T_{tran}=\dfrac{D_{tran}}{B_w}$.

问题 P:

$$Minimize \sum_{\text{所有迭代}} T^t_{num} \times T_{tran}$$

3 调度机制设计

本节将首先阐述 ESD 中输入嵌入样本调度的过程. 之后，在 3.2 节中介绍如何计算预期传输代价. 最后，鉴于边缘智能计算系统资源受限，介绍名为 HybridDis 的混合调度决策方法，该方法在资源消耗和求解质量之间进行平衡.

3.1 ESD 中样本调度机制流程

图 3 展示了 ESD 中嵌入样本调度的过程. 具体来说，在训练迭代 I_t 开始时，ESD 根据 I_{t+1} 的输入嵌入样本和每个工作节点上缓存嵌入的当前状态（是否缓存、是否为最新版本）为 I_{t+1} 做出调度决策. 此外，基于该调度决策及各工作节点上缓存嵌入的当前状态，ESD 可以生成每个工作节点在迭代 I_{t+1} 中的更新推送（Update Push）计划，这一部分内容在图 3 中未展示. 由于本文专注于在线 DLRM 训练，因此 I_{t+1} 的调度决策时间应小于等于 I_t 的训练时间，以确保嵌入样本调度能够减少训练时间. 上述调度过程是在每个工作节点本地执行的，而不是由一个集中式的调度器执行，以此避免分发调度决策带来的额外开销.

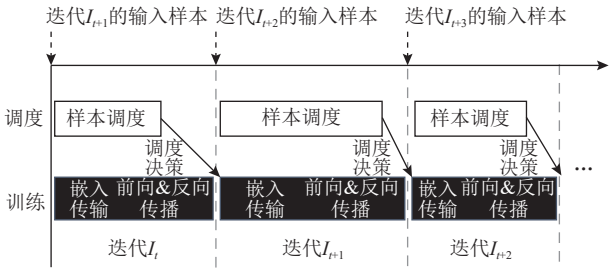


Fig. 3 Overview of embedding samples dispatching process in ESD

图 3 ESD 中的嵌入样本调度过程概览

当输入嵌入样本 E_i（$E_i \in \mathcal{E}_i$）在工作节点 w_j 上进行训练时，其预期传输代价为 c^j_i，本文将嵌入样本调度到合适的工作节点上以最小化总的嵌入传输代价. 当基于预期传输代价调度嵌入样本时，决策过程会消耗边缘工作节点的资源，而这些资源是有限的并且可能受到多个任务的竞争. ESD 引入了一种混合调度决策方法 HybridDis，它结合了资源密集型最优算法（Opt）和启发式算法（Heu），以平衡求解质量（即总嵌入传输代价）和资源消耗. 例如，ESD($\alpha=0.25$) 表示 25% 的输入嵌入样本使用 Opt 进行调度决策，而剩余 75% 使用 Heu. 本文分析了使用 Heu 时的调度误差，并使用 $\min_2-\min$ 对输入嵌入样本进行划分，其中 $\min_2$ 和 $\min$ 分别代表在不同工作节点上训练同一嵌入样本 E_i 的第二小和最小代价.

3.2 预期传输代价

本小节用于介绍如何计算预期传输代价. 由于每个工作节点上嵌入缓存的状态是可见的，并且当

今的数据加载器可以为下一个迭代预取嵌入样本，可以计算将每个嵌入样本调度到每个工作节点的预期传输代价(如算法 1 中的第 3 到第 9 行所示). 下面，通过一个例子来介绍预期传输代价的计算方法.

考虑在迭代I_t开始时，开始计算下一个迭代I_{t+1}的嵌入样本的预期传输代价. 如果输入嵌入样本$E_i=\{x_1,x_2,\cdots,\}$被调度到工作节点w_j进行训练，对于x_i的嵌入值$\boldsymbol{Emb}(\boldsymbol{x}_i)$，如果最新版本的$\boldsymbol{Emb}(\boldsymbol{x}_i)$不在$w_j$的缓存中，则需要执行一次 Miss Pull 操作，其代价为T^j_{tran}(第 7 行). 此外，如果在迭代I_t期间，其他工作节点上存有$\boldsymbol{Emb}(\boldsymbol{x}_i)$的最新版本，在按需同步的设置下，另一个工作节点($w_{j'}$)需要将$x_i$的梯度推送回参数服务器，因此$c^j_i+=T^{j'}_{tran}$，即将工作节点$w_{j'}$把梯度推送回参数服务器的传输代价累加到$c^j_i$上(第 9 行). 这样，对于迭代$I_{t+1}$，得到了将每个嵌入样本调度到不同工作节点的预期传输代价，并将这些预期传输代价存储在代价矩阵C中. 对于算法 1 中的第 6 行和第 8 行，in 表示$\boldsymbol{Emb}(\boldsymbol{x}_i)$的最新版本在边缘工作节点的嵌入缓存中可用，而 not in 则表示不可用.

此外，在边缘智能计算系统中训练 DLRM 面临边缘工作节点与 PS 之间异构的网络带宽，即不同的工作节点与参数服务器间的网络带宽不同，从而导致T^j_{tran}不同. 例如，在图 2 中的迭代I_2中，w_2与 PS 之间的带宽为 5 Gbps，而w_3与 PS 之间的带宽仅为 0.5 Gbps. 因此，从 PS 拉取$\boldsymbol{Emb}(\boldsymbol{x}_8)$到$w_3$的代价是从 PS 拉取$\boldsymbol{Emb}(\boldsymbol{x}_9)$到$w_2$的代价的十倍，即$T^3_{tran}=10\times T^2_{tran}$.

算法 1: ESD.

输入：一个迭代的输入嵌入样本$\mathcal{E}$，$|\mathcal{E}|=m\times n$，工作节点集合$\mathcal{W}$，代价矩阵C；

输出：调度决策(*Decision*).

① 初始化*Decision*；
② 初始化C中所有的元素为 0；
③ for each E_i in $\mathcal{E}$
④ for each w_j in $\mathcal{W}$
⑤ for each x_i in E_i
⑥ if x_i对应的嵌入$\boldsymbol{Emb}(\boldsymbol{x}_i)$ not in w_j
⑦ $c^j_i+=T^j_{tran}$；
⑧ if $\boldsymbol{Emb}(\boldsymbol{x}_i)$ in $w_{j'}$
⑨ $c^j_i+=T^{j'}_{tran}$；
⑩*Decision*=HybridDis(C)；
⑪ return 调度决策(*Decision*).

3.3 HybridDis

在获得代价矩阵C之后，本小节专注于设计一种调度决策方法来得到使总嵌入传输代价最小化的调度决策.

在资源受限的边缘智能计算系统中，确保得出调度决策所需的时间短于每次迭代的训练时间是一个挑战. 本节首先明确了在 CPU 上顺序执行匈牙利算法(Opt)会导致求解时间过长的问题. 然后简要分析了使用 CUDA 编程并行化匈牙利算法的可行性. 为了平衡解决方案质量(即总嵌入传输代价)和资源消耗，本节介绍了一种资源高效的启发式方法 Heu. Opt 是最优解法但资源消耗高，Heu 资源消耗低但求解质量差，因此，本文提出了一种结合 Opt 和 Heu 的混合调度决策方法 HybridDis. 在 HybridDis 中，将 Heu 的调度误差作为分割代价矩阵的标准.

对于代价矩阵C，可以使用匈牙利算法[75]求解得到能最小化总嵌入传输代价的调度方案. 当每个工作节点的批量大小为m，工作节点数量为n时，代价矩阵C的维度为$(m\times n)\times n$. 由于每个工作节点的批量大小为m，本文将C的每一列扩展为m列，生成一个阶数为k($k=m\times n$)的方阵C'，并将该方阵作为匈牙利算法的输入. 匈牙利算法的时间复杂度为$O\left(k^3\right)$[76]. 如表 2 所示，当在 CPU 上运行匈牙利算法时，当每个工作节点的批量大小从 32 增加到 1 024 时，执行时间从 9 毫秒增加到 134.986 秒，该延迟超过了每次迭代的训练时间，违反了每次求解时间不能超过训练时间的要求，即应在I_t的训练时间内计算出I_{t+1}的调度决策以避免增加端到端的训练时间. 经分析，匈牙利算法中，初始归约中的每行或每列减去最小值、初始匹配中的零元素覆盖以及矩阵调整期间的矩阵更新都可以并行化实现. 因此，本文使用 CUDA 并行实现了匈牙利算法，以确保调度决策的延迟保持在训练时间限制内[76]. 表 2 所示的执行时间证明了这种并行化的有效性. 3.4 节通过流程图的方式简要介绍了匈牙利算法，对于匈牙利算法的完整描述，请参阅 [77−78].

Table 2 Execution Time Comparison Between Serial and Parallel Implementations of Hungarian Algorithm for Different Batch Size Per Worker when Using 8 Workers

表 2 在使用 8 个工作节点的情况下，不同批量大小，串行与并行实现的匈牙利算法执行时间 ms

每节点批量大小	32	64	128	256	512	1 024
CPU 串行	9	62	528	3 360	50 976	134 986
GPU 并行	21	28	82	186	811	1 385

尽管匈牙利算法可以提供最优调度决策以最小化每次迭代的总嵌入传输代价,但使用 CUDA 实现匈牙利算法会消耗有限且争用的 GPU 资源. 为了平衡调度决策质量和资源消耗,如算法 2 所示,本文提出了 HybridDis,一种结合匈牙利算法(Opt)和启发式算法(Heu)的混合方法,用于将输入嵌入样本调度到边缘工作节点上. 算法 2 中的第 9 到第 18 行描述了启发式方法 Heu. 在代价矩阵C中,row_i 表示将E_i调度到n个工作节点的 n个预期嵌入代价,当该工作节点未达到其最大工作负载时,Heu 贪心地将E_i调度到预期代价最低的工作节点. 例如,如果 c_i^j是C中row_i的最小值,则E_i会被调度到节点w_j上. 在本文中,为了避免不同工作节点之间的不平衡,每个工作节点处理m个嵌入样本,即m是每个工作节点的最大工作负载. 显然,这种方法不是最优的,根据定理 1,最坏情况下的调度误差是 $\min_{\lfloor i/m\rfloor+1}-\min$,其中 $\min_{\lfloor i/m\rfloor+1}$和 min分别代表$row_i$中的第$\lfloor i/m\rfloor+1$小值和最小值.

为了平衡决策质量和计算效率,本文中 HybridDis 选取 $\min_2-\min$作为划分标准. 具体来说,本文将具有较高$\min_2-\min$值(潜在调度误差)的 $\alpha\times m\times n$ ($0\leqslant\alpha\leqslant1$)行分配给 Opt,而其余行则由 Heu 处理(算法 2 中的第 2 到第 5 行). 这确保了潜在调度误差较大的嵌入样本由最优求解方法处理. 值得注意的是,划分标准是可替换的,可以根据具体需求进行调整. 根据实际 DLRM 训练场景中观察到的数据分布模式,可以使用其他指标,如 $\min_3-\min$、$\min_3-\min_2$或行平均值.

算法 2: HybridDis.

输入: 一个迭代的输入嵌入样本$\mathcal{E}$, $|\mathcal{E}|=m\times n$, 工作节点集合$\mathcal{W}$, 最大负载$maxworkload=m$, 代价矩阵 C, α, 负载列表$workload[1\cdots n]$;

输出: $\mathcal{D}$. /*调度决策*/

① 使用 0 初始化负载列表$workload[1\cdots n]$;

② 计算C中每行的$\min_2-\min$;

③ 对C中每行基于$\min_2-\min$进行降序排序;

④ $C_{heu}\leftarrow C$中$\lceil|\mathcal{E}|\times\alpha\rceil$到$|\mathcal{E}|$行; /*保留原始行号*/

⑤ $C_{opt}\leftarrow C$中 0 到$\lfloor|\mathcal{E}|\times\alpha\rfloor$行

⑥ 将C_{opt}中每列复制为$\lfloor m\times\alpha\rfloor$列;

⑦ $\mathcal{D}=\mathrm{Opt}(C_{opt})$

⑧ $maxworkload=m-\lfloor m\times\alpha\rfloor$

⑨ for each row_i in C_{heu}

⑩ while True do

⑪ 选择row_i中的最小值c_i^j;

⑫ if $workload[j]<maxworkload$

⑬ 将E_i调度到w_j;

⑭ 将上述调度决策添加到$\mathcal{D}$;

⑮ $workload[j]+=1$;

⑯ break;

⑰ else

⑱ 将c_i^j从row_i的可选范围删除;

⑲ return $\mathcal{D}$.

定理 1 . 当使用 Heu 时,对于第i行row_i,最坏情况下的调度误差是$\min_{\lfloor i/m\rfloor+1}-\min$,其中 $\min_{\lfloor i/m\rfloor+1}$和 min分别代表$row_i$中的第$\lfloor i/m\rfloor+1$小值和最小值.

证明. 在代价矩阵C($m\times n$行, n列)中,每列代表一个节点,每节点最多可接受m个列(样本)被调度到自己这里来, m即 $maxworkload$. 对于 $row_i(0\leqslant i\leqslant m-1)$,最坏误差为 0,因为任意一个列都不会达到 $maxworkload$. 对于$row_i(m\leqslant i\leqslant 2m-1)$,最坏的情况如下, $row_i(0\leqslant i\leqslant m-1)$都被调度到同一个节点$w_{u^1}$,而在$row_i(m\leqslant i\leqslant 2m-1)$中,最小值都是 $c_i^{u^1}$,而代表节点w_{u^1}的列已经达到了$maxworkload$,那么对于$row_i(m\leqslant i\leqslant 2m-1)$,只能选择次小值$\min_2$. 对于$row_i(2m\leqslant i\leqslant 3m-1)$,类似地,当$row_i(0\leqslant i\leqslant m-1)$都被调度到同一个节点$w_{u^1}$且$row_i(m\leqslant i\leqslant 2m-1)$都被调度到同一个节点$w_{u^2}$,而对于$row_i(2m\leqslant i\leqslant 3m-1)$,其最小值和次小值均是 $c_i^{u^1}$或 $c_i^{u^2}$,但 w_{u^1}和 w_{u^2}均达到了 $maxworkload$,这导致$row_i(2m\leqslant i\leqslant 3m-1)$只能选择$\min_3$,以此类推,对于$row_i$(即$E_i$),在最坏情况下的调度误差是 $\min_{\lfloor i/m\rfloor+1}-\min$, $\min_{\lfloor i/m\rfloor+1}$和 min分别表示第$i$行$row_i$的第$\lfloor i/m\rfloor+1$小值和最小值. 证毕.

3.4 匈牙利算法求解流程

本文形式化的嵌入样本调度问题可被建模为二分图的最优匹配问题(总代价最小的匹配),即把k个嵌入样本分配给n个工作节点,并使分配的总代价最小. 由于每个工作节点的批量大小为m,本文将代价矩阵C的每一列扩展为m列,生成一个阶数为 k($k=m\times n$)的方阵C',则我们要做的是在这个矩阵C'中找到个数为k的一组独立元素(独立是指这组元素中任意 2 个元素不在同一行或同一列),并使这些元素的总代价最小,这组独立元素即为一个最优匹配. 文献 [78] 中陈述了 2 条定理: 1)如果矩阵C'中最大独立 0 元素个数为p,则最少p条线可覆盖C'中所有 0 元素. 2)矩阵C'的每一行(或列)同时减去一个常数并不会影响算法最后的解. 依据这两条定理我们可以对这个问题进行变形. 通过对矩阵中的行和列减去该行或该列的最小值,可以在代价矩阵中创造出一些 0 元素,并且确保矩阵中元素均为非负,由定理 2)可知此时问题

的解不变. 根据定理 1), 即最大匹配等价于最小顶点覆盖, 可通过该定理确定当前匹配是否为最大匹配.

因为我们需要求的是总代价最小的匹配, 而此时矩阵中最小的元素是 0, 因此算法的思路是: 通过在矩阵中不断创造新的 0 元素从而把最优匹配问题转化为完备匹配问题, 即只把 0 元素作为二分图中的边, 在这些边中找到完备匹配(让每个点都与一个边相连, 即为所有的工作节点和嵌入样本找到一一对应的边); 那么算法最终的解一定是最优匹配(因为所有边代价均为 0, 总和一定是最小的). 在本算法中, 该解的形式为最大独立 0 的集合(个数为k), 每个独立 0 即为一条匹配边.

算法流程图如图 4 所示. 初始阶段首先对矩阵进行行归约与列归约, 即每行、每列减去该行、该列的最小值, 形成一些 0 元素, 并找到一组独立 0(即一组匹配), 标记为 0*, 需要注意的是这组独立 0 不一定是当前该矩阵的最大独立 0 集合(初始匹配). 接下来尝试用经过 0*的横线或竖线(每条线覆盖一整行、列的元素)来覆盖所有的 0. 由定理 1)可知, 如果存在这样的线集合, 则说明当前的 0*集合即为当前矩阵的最大独立 0 集合, 也即为当前二分图的最大匹配; 如果不存在, 则当前 0*集合并非当前矩阵的最大独立 0 集合. 如果不能覆盖所有 0, 说明现在还不是最大匹配, 则通过寻找增广路径的办法来增加匹配(独立 0)

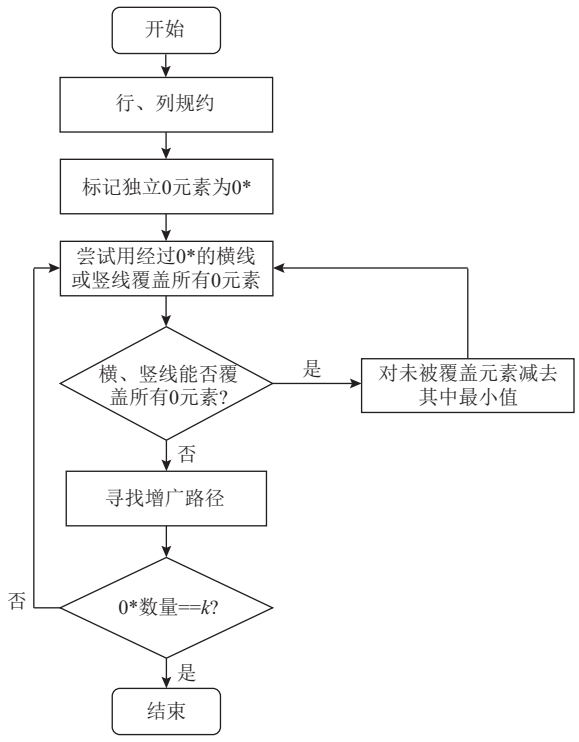


Fig. 4 Flow chart of Hungarian algorithm

图 4 匈牙利算法流程图

的数量, 该步骤一定会使 0*的数量加 1. 检查最新的 0*集合, 如果 0*数量等于k, 说明找到了完备匹配(即最优解), 此时 0*集合即为解集; 如果 0*数量 < k, 则继续尝试用经过 0*的横线或竖线来覆盖所有的 0. 若能覆盖所有 0, 说明现在已经是当前二分图的最大匹配, 但需要注意的是, 目前还不是完备匹配. 由于当前矩阵最大独立 0 个数 < k, 因此需要在矩阵中(未被覆盖的元素里)创造新的 0 来增加最大独立 0 的个数, 即对未被覆盖的元素减去未被覆盖元素中的最小值, 由定理 2), 可通过对未被覆盖的列减去该最小值, 对被覆盖的行加上该最小值来实现.

4 系统实现

本文在 HET[49] 的基础上使用 C++和 Python 实现了 ESD 的原型. 除了第 3 节中描述的设计外, ESD 维护所有工作节点的缓存快照, 以提供嵌入的信息. 这些缓存快照用于计算预期传输代价, 并在调度结束时根据调度结果进行更新. 本文替换了 HET 中的数据加载器实现, ESD 中的数据加载器返回嵌入样本和按需同步的嵌入 ID 作为稀疏输入. 为了支持不干扰训练过程的输入预取, ESD 由专用线程启动, 且 ESD 的调度决策通过共享内存传递给数据加载器 . 此外, 本研究采用 CUDA 并行策略加速匈牙利算法, 主要优化包括: 1)在行列归约阶段, 通过并行归约核函数并行计算每行/列的最小值, 并利用线程块级并行完成矩阵减法操作; 2)针对独立 0 元素的标记(初始匹配阶段), 通过原子操作解决多线程竞争条件下的 0 元素标记问题, 确保独立 0 元素标记的互斥性; 3)矩阵更新时, 利用并行归约定位未覆盖元素中的最小值, 通过动态并行与块级分块策略优化内存访问模式, 实现加减操作. 特别地, 采用共享内存缓存技术优化全局内存访问模式, 通过矩阵分块处理减少数据访存延迟.

5 实验评估

本文使用如表 **3** 所列的工作负载来评估 ESD 的性能. 与最先进的机制 LAIA 相比, ESD 实现了至多 1.74 倍的加速比, 并能将嵌入传输代价降低至多 36.76%. 通过对每个工作节点的批量大小、缓存比例和嵌入大小进行敏感性分析, ESD 始终优于基线方法. 所有实验均重复五次, 取平均值作为最终结果.

5.1 实验设置

1)实验环境. 工作节点和参数服务器(PS)均配

备了一颗28 核心、主频为2.0 GHz 的Intel Xeon Gold 6 330 CPU 以及 64 GB(PS 为 500GB)的内存. 每个工作节点配备一块 Nvidia 4 090 GPU. 工作节点和参数服务器通过 0.5 Gbps 或 5 Gbps 的以太网连接. 所有机器运行 Ubuntu 18.04, Python 3.8, CUDA 11.3, cuDNN 8.2.0, NCCL 2.8 和 OpenMPI 4.0.3. 实验环境中有 8 个边缘工作节点和 1 个参数服务器, 其中 4 个工作节点通过 5 Gbps 以太网与参数服务器相连, 另外 4 个则通过 0.5 Gbps 以太网连接与参数服务器相连. 默认情况下, 嵌入大小为 512, 每个工作节点的批量大小为 128、缓存比例为 8%.

2) 基线方法. 本文将 ESD 与以下机制进行了比较:

HET[49]. HET 是一个在参数服务器架构下通过跟踪嵌入版本以容忍有界陈旧性的嵌入缓存机制. 当检索嵌入时, 会将本地版本与 PS 版本进行比较, 如果差异超过阈值, 则从 PS 拉取嵌入. 同样的方法也适用于梯度同步.

FAE[50]. FAE 设计采用了静态缓存策略. 在训练前, 缓存的嵌入会被离线分析并固定下来. 所有工作节点在 FAE 中缓存相同的嵌入, 并使用全局规约同步所有缓存的嵌入.

LAIA[37]. LAIA 旨在多个云端工作节点之间调度嵌入. LAIA 计算一个分数来量化每个输入样本与工作节点之间的相关性, 并将每个输入分配给得分最高的工作节点.

HET 和 FAE 通过牺牲一定的模型准确性来减少嵌入传输代价. 在本文的实验中, HET 和 FAE 中采用批量同步并行. 在本节中, ESD 是一个更通用的说法, 无论α为何值, 均可以称为 ESD. 具体来说, ESD($\alpha=1$)、ESD($\alpha=0.5$)、ESD($\alpha=0.25$)、ESD($\alpha=0.125$) 和 ESD($\alpha=0$) 分别代表具有不同α值的机制. 这里的α指的是 HybridDis 混合决策方法中 Opt(匈牙利算法)和 Heu(启发式算法)的分配比例. 不同的α值反映了在求解质量和资源消耗之间不同的权衡策略. 例如, ESD($\alpha=1$) 意味着完全采用 Opt 方法, 而 ESD($\alpha=0$) 则表示完全使用 Heu. 通过调整α, 可以在不同的应用场景下找到合适的配置.

3)负载. 如表 3 所列, 本文使用 3 个负载(即 S1, S2 和 S3)进行端到端实验, 其中每个负载均由代表性的深度学习推荐模型和数据集组成. 为了排除初始阶段的影响, 本文排除了最初的 10 次迭代作为预热, 并报告随后迭代的性能表现. 本文没有展示模型准确率的结果, 因为正如第 1.1 节分析的那样, ESD 不会影响模型准确率. 表 3 所示的数据集均以用户行为日志(如广告点击、搜索转化)为核心, 包含用户 ID、设备信息、上下文属性等在内的匿名化的高维稀疏特征. 其中 Criteo Kaggle 数据集中每个样本有 26 个 ID, 总样本数为 41 256 556; Avazu 数据集中, 每个样本有 18 个 ID, 总样本数为 36 386 071; 在 Criteo Sponsored Search 数据集中, 每个样本有 17 个 ID, 总样本数为 14 396 071.

Table 3 Workloads in Experiment

表 3 实验所用负载

负载序号	所用模型	数据集
S1	WDL[26]	Criteo Kaggle[36]
S2	DFM[27]	Avazu[79]
S3	DCN[68]	Criteo Sponsored Search[80]

4)指标. 用于评估不同方法的性能指标是每秒训练迭代次数(Iterations per second, ItpS)和总嵌入传输代价(Cost). 为了便于比较, 使用 LAIA 的 ItpS 作为参考基准, 其他机制的性能提升则以相对于这个参考值的比例来表示.

$$\text{机制A的加速比}=\frac{\text{机制A的ItpS}}{\text{LAIA的ItpS}}$$

同样, LAIA 也被用作减少嵌入传输代价的参考基准.

$$\text{机制A的代价降低}=\frac{\text{LAIA的代价}-\text{机制A的代价}}{\text{LAIA的代价}}$$

5.2 总体性能

本文首先评估了在默认设置下, ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 的整体性能. 图 5(a)展示了不同工作负载下, ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 相对于基线方法的训练加速比.

观察发现, 相比 LAIA(用作参考且未在图 5 中显示), ESD 可以实现 1.03 倍到 1.74 倍的加速. 具体来说, 本文注意到随着α值的减小, 加速比也随之下降. 这是因为较大的α值意味着更多的嵌入样本由 Opt 处理, 从而降低了每次迭代的嵌入传输代价. 如图 5(b)所示, 与 LAIA 相比, ESD($\alpha=1$) 可以减少至多 36.76% 的传输代价, 而 ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 分别将嵌入代价降低了 10.81% 和 7.03%. 实验结果还表明, FAE 和 HET 相比于 LAIA 表现较差. 因此, 为了专注于 ESD 相对于 LAIA 的改进, 本文在后续的实验分析中省略了 FAE 和 HET 的结果.

5.3 命中率和传输代价组成

为了探讨影响这些机制性能的原因, 本文在图 6 中展示了命中率和嵌入传输操作的组成. 本文假设,

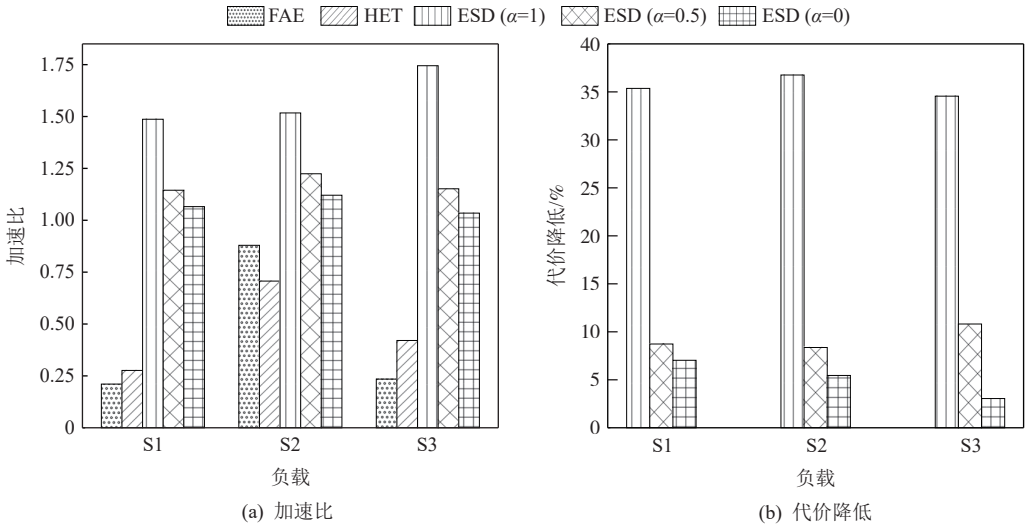


Fig. 5 Overall performance

图 5 总体性能

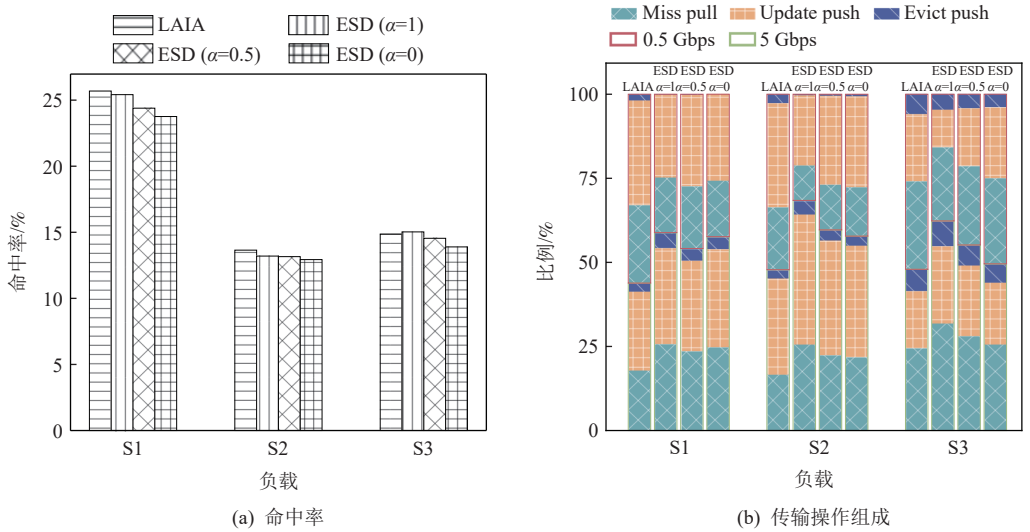


Fig. 6 Hit ratio and ingredient of transmission operations

图 6 命中率和传输操作组成

若输入嵌入样本中存在x_i且最新版本的 $\boldsymbol{Emb}(x_i)$ 已经缓存于工作节点w_j的嵌入缓存中，则视为一次命中. 命中率定义为相对于总查找次数的缓存命中的查找比例. 本文在图 6(a)中展示了 LAIA、ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 的命中率. 与 LAIA 相比，ESD 并没有实现更高的命中率. 然而，如图 5 所示，ESD 相比 LAIA 降低了代价并加速了端到端训练. 这与引言部分讨论的内容一致，因为嵌入传输代价不仅来自于嵌入未命中，还来自于 Update Push 和 Evict Push 操作. 因此，更高的命中率并不一定能降低嵌入传输代价.

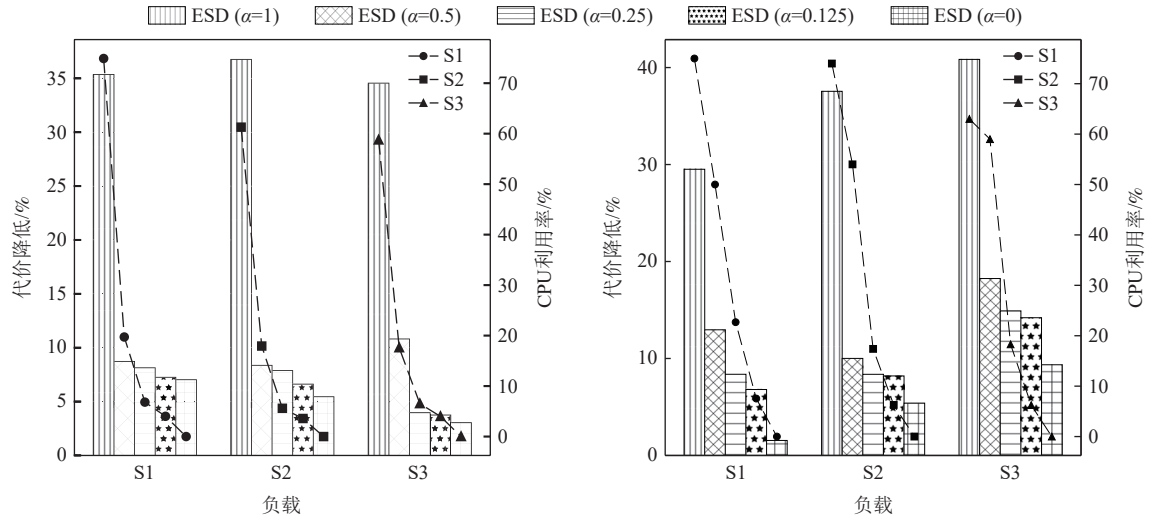
图 6(b)说明了不同机制的嵌入传输操作的组成，即未命中拉取(Miss Pull)、更新推送(Update Push)和逐出推送(Evict Push)操作在总传输操作中的比例. 在每个条形图中，底部三个部分代表 5 Gbps 工作节点的结果，而上部部分对应 0.5 Gbps 工作节点. 如图 6(b)所示，与 LAIA 相比，ESD 在所有三种工作负载下，5 Gbps 工作节点的操作比例更大，而在 LAIA 中，5 Gbps 工作节点的操作比例小于 0.5 Gbps 工作节点. 这表明 ESD 有效地考虑了异构网络的情况. 图 6(b)还显示，未命中拉取和更新推送占嵌入传输操作的 90% 以上，而逐出推送的占比不到 10%. 此外，实验表明，每节点批量大小的增加会使得未命中拉取的比例增加，缓存比例越大则逐出推送操作占比越少，嵌入大小的变化一般不影响各操作占比

5.4 代价降低和 GPU 资源消耗

由于边缘工作节点上的资源有限，本文提出了

一种名为 HybridDis 的混合方法来决定输入嵌入样本的调度,其中α表示使用 Opt 进行调度的比例.

图 7 展示了,当α为 1、0.5、0.25、0.125 和 0 时,每个工作节点的批量大小分别为 128 和 256 时,代价降低和 GPU 资源消耗(以 GPU 利用率表示,GPU 利用越高表示该方法占用 GPU 资源越多). 对于 ESD($\alpha=0$), GPU 利用率为 0. 当每个工作节点的批量大小为 128 时,如图 7(a)所示,在不同工作负载下,较大的α值会带来更大的代价降低和更高的 GPU 利用率. 在 S1 工作负载上,ESD($\alpha=1$) 带来的代价降低最多,GPU 利用率为 74.89%. 当每个工作节点的批量大小为 256 时,如图 7(b)所示,ESD 可以将嵌入传输代价减少多达 40.84%. 对于负载 S1,当每个工作节点的批量大小为 256 时,对于 ESD($\alpha=0.5$), 代价降低为 12.96%, GPU 利用率为 50%, 对于 ESD($\alpha=1$), 代价降低为 29.51%, GPU 利用率为 75%. 总体而言,较大的α值会带来更大的代价降低和更高的 GPU 利用率. α的设置取决于边缘工作节点上 GPU 资源的多少以及多个任务之间的竞争情况. 可以看到,即使$\alpha=0$, ESD 相对于 LAIA 仍然可以减少传输代价.



(a) 节点批量大小为128 (b) 节点批量大小为256

Fig. 7 Cost reduction and GPU resource consumption

图 7 代价降低和 GPU 资源消耗

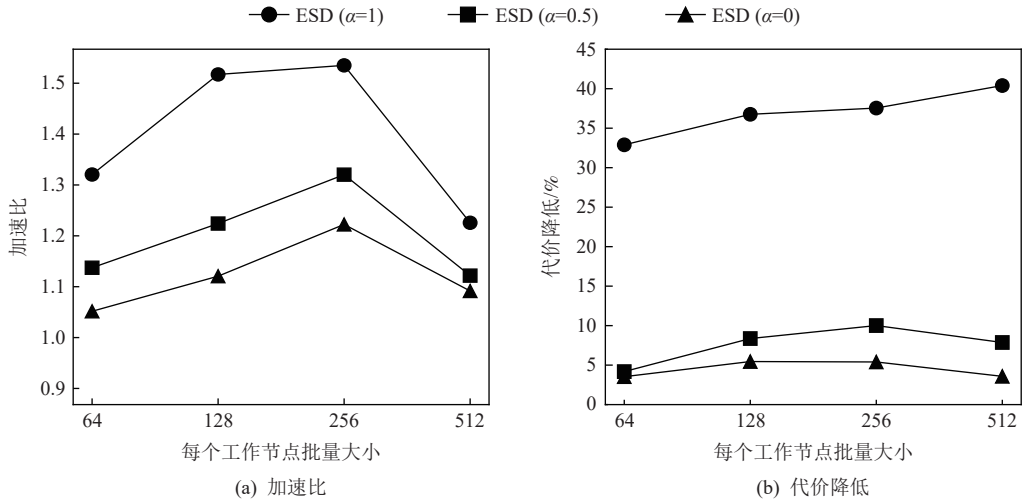
需要注意的是,准确测量 GPU 的资源消耗是计算机系统中的一个极具挑战性的问题[81-82]. 在图 7 中,本文使用执行命令 nvtop 得到的 GPU 利用率作为衡量 GPU 资源消耗的标准,显然,这是一个粗粒度的资源消耗测量标准. 需要说明的是,图 7 中 ESD 的 GPU 利用率是在 GPU 上单独执行 ESD 的过程中而非在 DLRM 训练过程中测得的. 实际上,当节点批量大小为 128, 在负载 S3 上进行训练时,当在训练过程中同时执行 ESD($\alpha=1$) 时,通过 nvtop 观察到的 GPU 利用率约为 55%, 而图 7(a)中展示的单独执行 ESD($\alpha=1$) 时 GPU 利用率约为 58%, 因此,图 7 中显示的 GPU 利用率旨在说明随着α值的减小,使用的 GPU 资源也随之减少,并没有反应真实的 GPU 资源消耗比例,也就是说, ESD($\alpha=1$) 这一调度机制实际上并不会像图 7 所示那样消耗高达 70% 的 GPU 资源.

5.5 敏感性分析

接下来,本文将探讨当每个工作节点的批量大小、缓存比例和嵌入大小发生变化时 ESD 的性能. 此外,本文还评估了在四个工作节点以及工作节点与参数服务器之间具有相同带宽(同构网络)的情况下 ESD 的性能表现. 当某一设置发生变化时,其他设置保持默认值. 在这一部分中,本文主要关注工作负载 S2.

5.5.1 每个工作节点的批量大小

如图 8 所示,为了研究每个工作节点批量大小的影响,本文将其从 64 变化到 512, 并展示了加速比和代价降低的情况. 当每个工作节点的批量大小从 64 增加到 256 时, ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 的加速比呈增长趋势,最高可达 1.54 倍. 然而,当批量大小从 256 增加到 512 时, ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 的加速比并未超过 256 时的表现(尽管仍保持在 1 以上). 这是因为增加每个工作节点的批量大小会增加决策时间,并且随着批量大小的增加, ESD($\alpha=0$) 的解决方案质量下降,导致这三种机制的加速比均有所下降. 对于传输代价,随着每个



(a) 加速比　　(b) 代价降低

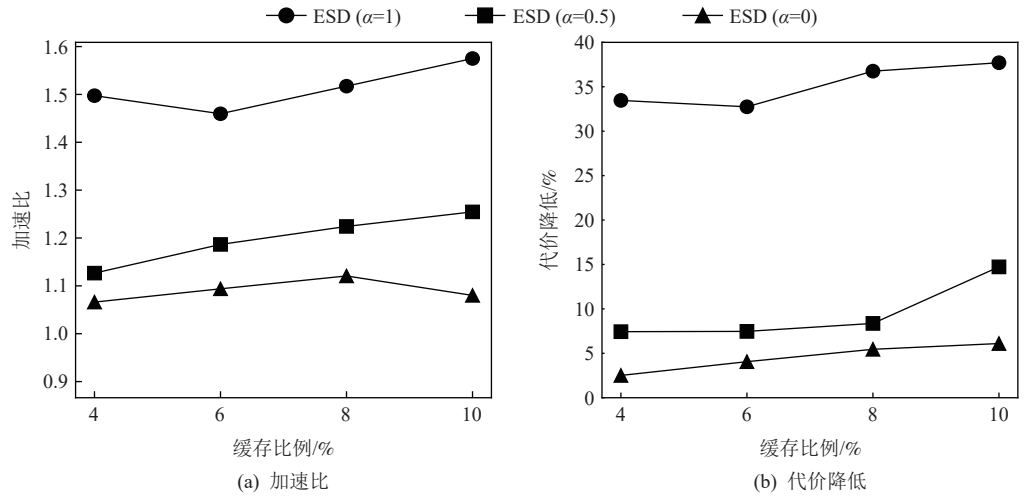
Fig. 8　Impact of batch size per worker on performance

图 8　每工作节点批量大小对性能的影响

工作节点批量大小的增加，代价降低的趋势与加速比的趋势相似. 当批量大小从 256 增加到 512 时，虽然 ESD($\alpha=1$) 实现了更大的代价降低，但增加的决策时间使得加速比从 1.54 降至 1.23.

5.5.2　缓存比例

如图 9 所示，本小节展示缓存比例的影响. 缓存比例，即缓存中嵌入数量与总嵌入数量的比例，范围 4%~10%.



(a) 加速比　　(b) 代价降低

Fig. 9　Impact of cache ratio on performance

图 9　缓存比例对性能的影响

图 9 表明，ESD 在不同大小的嵌入缓存下相对于 LAIA 保持了性能上的优越性，并且对于相同的α值，加速比在不同缓存大小下并没有显著变化. 这个实验验证了 ESD 在不同缓存比例下的持续有效性.

5.5.3　嵌入大小

嵌入大小，即嵌入向量的维度，是影响 DLRM 模型性能的重要因素. 如图 10 所示，本文测试了嵌入大小为 128、256、512 和 1 024 时，ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 的加速比和代价降低的变化情况. 当嵌入大小增加时，如图 10(a)所示，三种机制相对于 LAIA 的加速比均有所增加，最高可达 1.59 倍. 原因是随着嵌入大小的增加，每次嵌入拉取和推送的数据量(即 D_{tran})变大，从而增加了嵌入传输的代价. 当每次嵌入传输代价增加时，ESD 的效果变得更加显著. 对于代价降低，嵌入大小的变化仅影响 D_{tran}，并不影响 ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 相对于 LAIA 的代价降低效果. 例如，当嵌入大小从 256 增加到 512 时，D_{tran}翻倍，导致 LAIA 和 ESD 的嵌入传输代价也都翻倍. 然而，ESD 相对于 LAIA 的代价降低比例保持不变.

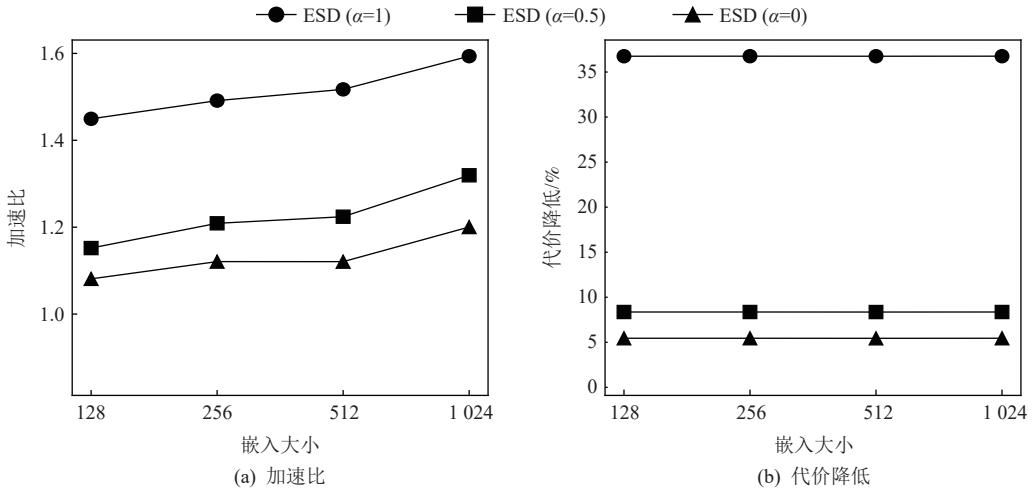


Fig. 10 Impact of embedding size on performance

图 10 嵌入大小对性能的影响

5.5.4 工作节点数量和网络带宽

之前实验是在 8 个边缘工作节点上进行的, 这些工作节点与参数服务器(PS)之间的网络带宽是异构的. 为了验证 ESD 在不同数量的工作节点以及工作节点与 PS 之间具有相同网络带宽的情况下的有效性, 本文测试了以下设置下的 ESD($\alpha=1$)、ESD($\alpha=0.5$) 和 ESD($\alpha=0$) 的性能. 一是使用 4 个边缘工作节点, 其中 2 个带宽为 5 Gbps, 另外两个带宽为 0.5 Gbps. 二是使用 4 个带宽均为 5Gbps 的工作节点. 实验结果如图 11 所示. 在这两种设置下, 对于 3 种机制和 3 种工作负载, 加速比分别在 1.07~1.31 倍和 1.03~1.23 倍之间. 对于代价降低, 当使用两个 5 Gbps 工作节点和两个 0.5 Gbps 工作节点时, 代价降低范围为 6.06% 到 42.15%. 相比之下, 当使用四个 5 Gbps 工作节点时, 代价降低范围为 0.33% 到 29.11%. 实验表明, ESD 在具有四个工作节点的同构网络中是有效的, 并且在异构网络中的表现更好, 这与本文的动机一致.

6 相关工作

6.1 为边缘智能计算系统设计的推荐模型训练方法

为边缘智能计算系统设计的推荐模型训练方法通常专注于隐私保护和效率. Yuan 等人提出了无参数传输联邦推荐框架 PTF-FedRec, 通过在客户端与参数服务器之间传输预测来促进协作学习, 同时通过本地模型预测的采样和交换机制确保隐私[83]. HFSA 是一个半异步层次化的联邦推荐系统, 通过边缘聚合减少通信开销, 采用半同步机制提高性能, 并允许慢速客户端异步贡献其参数以确保全局模型收敛, 适用于动态移动和边缘环境[24]. DualRec 是一个结合联邦学习与高效模型聚合和降噪机制的协同训练框架, 允许设备用本地数据训练轻量级模型, 而较大的模型则在云端训练, 通过生成伪用户交互和在设备与云端模型之间进行知识蒸馏来提升推荐系统性能[84]. PREFER[23] 是一个针对兴趣点推荐的边缘加速联邦学习框架, 其中用户的独立参数在用户间共享并在边缘服务器上进行聚合. PEPPER[59] 是一个基于 gossip learning 的去中心化推荐系统, 使用户能够异步训练个性化模型. 与已有工作类似, 本文专注于在边缘智能计算系统中为物联网应用提供推荐服务, 不同之处在于本文更关注于通过调度嵌入样本减少嵌入传输代价从而加速模型训练, 而现有工作主要关注训练方法设计和联邦推荐系统设计.

6.2 深度学习推荐模型训练加速方法

先前的部分工作集中在最小化传输次数上[49,50,85]. FAE[50] 通过在 GPU 上存储热门嵌入来减少从 CPU 到 GPU 的嵌入传输. Persia[85] 使用混合同步和异步机制分别更新 MLP 和嵌入表. HET[49] 设计了一个缓存一致性模型, 允许缓存读写操作中的陈旧性以最小化传输开销. LAIA[37] 对嵌入进行调度以减少传输次数而不牺牲准确性. 其他工作则专注于减少每次传输的延迟. Ugache[35] 引入了一种分解提取机制以防止带宽拥塞, 并利用 NVLink 和 NVSwitch 加速远程 GPU 内存访问. AdaEmbed[86] 和 CAFE[31] 旨在通过压缩和剪枝减少嵌入的传输代价. ScaleFreeCTR[29] 采用混合缓存机制消除主机与 GPU 之间的传输代价, 并利用虚拟稀疏 ID 减少传输量. 本文关注在边缘智能计算系统中训练 DLRM, 并提出了 ESD 来应对边缘

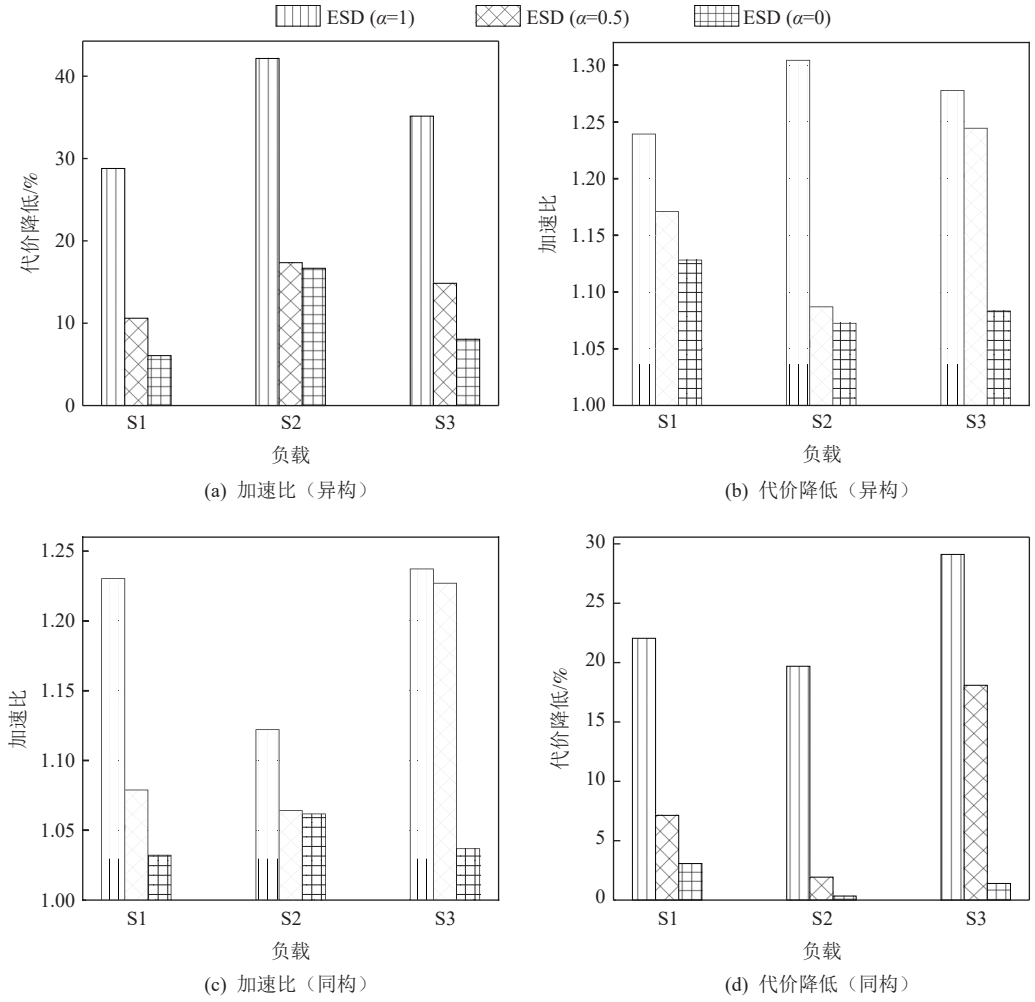


Fig. 11 Experiment results when using four workers

图 11 当使用 4 个工作节点时的实验结果

侧的独特挑战，如资源受限和异构网络，以最小化总的嵌入传输代价，从而加速 DLRM 训练.

7 讨论和未来工作

7.1 缓存替换策略

在 DLRM 的训练过程中，当嵌入缓存达到容量时，必须将一个或多个已缓存的嵌入替换出去以容纳新的嵌入. 在按需同步的情况下，如果被替换出去的嵌入的梯度尚未与参数服务器(PS)同步，则会触发 Evict Push 操作. 本文设计了 Emark，一种基于标记的策略来管理每个工作节点中的缓存嵌入，以减少 Evict Push 操作的数量，该策略考虑了最近性、频率和嵌入版本. 当嵌入 ID x_i被调度到w_i时，Emark 为$\boldsymbol{Emb}(x_i)$分配一个标记(target)，target 是一个从 1 开始的正整数. 当缓存满且所有嵌入的标记都是 target 时，target 加 1. 对于缓存替换，当缓存满时，Emark 首先逐出过期的嵌入，然后按标记升序比较，最后比较访问频率. 本文在 C++中重载了<运算符以实现这一点. 此外，通过调整 target 的递增方法和重载<运算符中的比较顺序，Emark 可调整为优先考虑频率、最近性或嵌入版本. 由于图 5(b)显示逐出推送对传输操作的贡献不到 10%，本文在此讨论缓存替换策略而不是在第 3 节中讨论. 此外，将 Evict Push 和替换策略纳入预期传输代价的计算是减少 Evict Push 操作的有效方式，本文将此作为未来的工作方向.

7.2 非均一嵌入大小

本文假设所有嵌入向量维度(即嵌入大小)是均一的. 然而，最近的趋势表明，使用非均一嵌入大小可以减少 DLRM 的内存占用、训练时间和推理时间[87-88]. 当嵌入大小不均一时，ESD 可以通过以下调整有效处理这种情况：首先，对于计算预期传输代价，可以使用不同的数据大小(D_{tran})来考虑非均一的嵌入大小. 其次，关于缓存替换策略，非均一嵌入大小会导

致嵌入缓存中不同嵌入占用的的内存大小不同. 这可以通过优先级指标进行管理, 缓存中每个嵌入的优先级的计算方式如下, 使用频率、最近性和版本(最新或过时)的相关数值作为分子, 而每个嵌入的内存占用作为分母, 当缓存达到容量时, 优先级最低的嵌入将首先被逐出. 在未来的研究中, 本文将进一步探索边缘智能计算系统中训练和部署 DLRM 时嵌入大小的选择问题.

7.3 与其他边缘计算优化方法的异同

在边缘智能计算系统中, 资源分配、任务卸载和缓存策略是常见的优化方法. 然而, 这些方法与本文提出的 ESD 机制在优化目标、技术路径和适用场景上存在差异. 边缘计算中的资源分配方法通常聚焦于存储、计算、网络资源的动态分配以提升系统吞吐率[89], 其核心在于平衡多任务间的资源竞争. 与之相比, 虽然 ESD 中 HybridDis 也考虑了资源受限场景中多任务争用资源的情况, 但其并不直接干预资源分配, 而是采用α来平衡求解质量和资源消耗. 任务卸载方法通常将计算任务迁移至云端或其他边缘节点以降低延迟或能耗[90], 其卸载决策基于任务计算量与节点处理能力的匹配. 与任务卸载相似, ESD 基于预期传输代价将嵌入样本调度到不同的工作节点上, 不同之处在于, 任务卸载中多个任务间一般无关联, 而对嵌入样本进行调度会影响更新推送操作的数量. 缓存策略主要基于数据的历史访问模式来提升缓存命中率[91], 在 ESD 中, 如引言部分和 7.1 节所述, 与传统缓存问题不同, DLRM 训练中的嵌入具有动态更新特性, ESD 需要同时考虑未命中拉取、更新推送和逐出推送三类操作代价. 总的来说, 本文主要考虑在边缘智能计算系统中加速 DLRM 训练, 面临嵌入传输代价组成复杂、异构网络、资源受限等挑战, 资源分配、任务卸载和缓存策略等边缘计算优化工作也会考虑网络异构、资源受限等问题, 但尚无研究工作面向在边缘节点上训练 DLRM 这个场景进行深入研究.

8 结 论

在边缘智能计算系统中训练和部署 DLRM 是在确保隐私保护和提供低延迟服务的情况下, 为物联网应用提供个性化推荐服务的有效方式. 本文专注于基于预期嵌入传输代价调度输入嵌入样本, 以最小化总的嵌入传输代价, 进而加速 DLRM 训练. 本文强调了在边缘智能系统中设计调度机制所面临的挑战, 包括代价构成、异构网络和资源受限. 为应对这些挑战, 本文提出了一种名为 ESD 的调度机制, 其中包含作为关键组件的混合决策方法 HybridDis. 本文使用 C++(包括 CUDA)和 Python 实现了 ESD, 并通过典型的负载在边缘工作节点上进行了实验, 以验证其相较于最先进机制的改进. 构建一个高效的边缘智能计算系统用于服务物联网应用需要在数据收集、处理、模型训练、部署等方面进行优化. 本文希望这项工作能为构建高效的边缘智能计算系统贡献一份力量.

作者贡献声明: 李国鹏明确研究挑战, 提出解决思路, 设计并完成实验和撰写论文; 谈海生提出问题, 参与解决方案设计, 对论文撰写与实验方案设计进行指导; 张弛参与方案设计和撰写论文; 倪宏秋、王子龙、章馨月、徐洋调研文献以及参与算法设计, 田晗提供问题背景、对实验、论文写作进行指导; 陈国良对整体研究方向和思路进行指导.

参 考 文 献

[1] Gu Yulong, Bao Wentian, Ou Dan, et al. Self-supervised learning on users' spontaneous behaviors for multi-scenario ranking in e-commerce[C]//Proc of the 30th ACM Int Conf on Information & Knowledge Management. New York: ACM, 2021: 3828—3837

[2] Wang Jizhe, Huang Pipei, Zhao Huan, et al. Billion-scale commodity embedding for e-commerce recommendation in alibaba[C]//Proc of the 24th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2018: 839–848

[3] Smith B and Linden G. Two decades of recommender systems at amazon. Com[J]. IEEE Internet Computing, 2017, 21(3): 12−18

[4] Gomez-Uribe C and Hunt N. The netflix recommender system: Algorithms, business value, and innovation[J]. ACM Trans on Management Information System, 2015, 6(4): 1−19

[5] Covington P, Adams J, and Sargin E. Deep neural networks for youtube recommendations[C]//Proc of the 10th ACM Conf on Recommender Systems. New York: ACM, 2016: 191−198

[6] Schedl M, Knees P, and Gouyon F. New paths in music recommender systems research[C]//Proc of the 11th ACM Conf on Recommender Systems. New York: ACM, 2017: 392−393

[7] Sharma A, Jiang J, Bommannavar P, et al. Graphjet: Real-time content recommendations at twitter[J]. Proc of the VLDB Endowment, 2016, 9(13): 1281−1292

[8] Boeker M and Urman A. An empirical investigation of personalization factors on tiktok[C]//Proc of the ACM Web Conf 2022. New York: ACM, 2022: 2298−2309

[9] Ying R, He Ruining, Chen Kaifeng, et al. Graph convolutional neural networks for web-scale recommender systems[C]//Proc of the 24th

ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2018: 974−983

[10] Peng Yingtao, Meng Xiaofeng, Du Zhijuan. Survey on Diversified Recommendation[J]. Journal of Computer Research and Development, 2025, 62(2): 285−313 (in Chinese)
(彭迎涛, 孟小峰, 杜治娟. 多样化推荐综述[J]. 计算机研究与发展, 2025, 62(2): 285−313)

[11] Wang Siqi, Feng Tianyu, Yang Hailong, et al. Atrec: Accelerating recommendation model training on cpus[J]. IEEE Trans on Parallel and Distributed Systems, 2024, 35(6): 905−918

[12] Sayed A, Himeur Y, Alsalemi A, et al. Intelligent edge-based recommender system for internet of energy applications[J]. IEEE Systems Journal, 2021, 16(3): 5001−5010

[13] Himeur Y, Alsalemi A, Al-Kababji A, et al. A survey of recommender systems for energy efficiency in buildings: Principles, challenges and prospects[J]. Information Fusion, 2021, 72: 1−21

[14] Pourpanah F and Etemad A. Exploring the landscape of ubiquitous in-home health monitoring: a comprehensive survey[J]. ACM Trans on Computing for Healthcare, 2024, 5(4): 1−43.

[15] Su Xin, Giancarlo S, Vincenzo M, Antonio Picariello, et al. An edge intelligence empowered recommender system enabling cultural heritage applications[J]. IEEE Trans on Industrial Informatics, 2019, 15(7): 4266−4275

[16] Yin Hongzhi, Chen Tong, Qu Liang, et al. On-device recommender systems: A tutorial on the new-generation recommendation paradigm[C]//Proc of the ACM Web Conf 2024. New York: ACM, 2024: 1280−1283

[17] Cai Qiqi, Cao Jian, Xu Guandong, et al. Distributed recommendation systems: survey and research directions[J]. ACM Trans on Information Systems, 2024, 43(1): 1−38,

[18] Long Jing, Ye Guanhua, Chen Tong, et al. Diffusion-based cloud-edge-device collaborative learning for next poi recommendations[C]//Proc of the 30th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2024: 1324−1337

[19] Yuan Wei, Qu Liang, Cui Lizhen, et al. Hetefedrec: Federated recommender systems with model heterogeneity[C]//Proc of the 40th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2024: 2976−2987

[20] Yongbo Yu, Fuxun Yu, Xiang Sheng, et al. Eaglerec: Edge-scale recommendation system acceleration with inter-stage parallelism optimization on gpus[C]//Proc of the 60th Design Automation Conf. Piscataway, NJ: IEEE, 2023: 1−6

[21] Gong Yu, Jiang Ziwen, Feng Yufei, et al. Edgerec: recommender system on edge in mobile taobao[C]//Proc of the 29th ACM Int Conf on Information & Knowledge Management. New York: ACM, 2020: 2477−2484

[22] Himeur Y, Sohail S, Bensaali F, et al. Latest trends of security and privacy in recommender systems: a comprehensive review and future perspectives[J]. Computers & Security, 2022, 118: 102746,

[23] Guo Yeting, Liu Fang, Cai Zhiping, et al. Prefer: Point-of-interest recommendation with efficiency and privacy-preservation via federated edge learning[J]. Proc of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2021, 5(1): 1−25

[24] Li Youhuizi, Yu Haitao, Zeng Yan, et al. Hfsa: A semi-asynchronous hierarchical federated recommendation system in smart city[J]. IEEE Internet of Things Journal, 2023, 10(21): 18808−18820

[25] Wu Jiang, Yang Yunchao, Hu Miao, etal. FCER: A federated cloud-edge recommendation framework with cluster-based edge selection [J]. IEEE Trans on Mobile Computing, 2025, 24(3): 1731−1743

[26] Cheng Heng-Tze, Koc L, Harmsen J, et al. Wide & deep learning for recommender systems[C]//Proc of the 1st workshop on deep learning for recommender systems. New York: ACM, 2016: 7−10

[27] Guo Huifeng, Tang Ruiming, Ye Yunming, et al. Deepfm: a factorization-machine based neural network for ctr prediction[J]. arXiv preprint arXiv: 1703.04247, 2017

[28] Jiang Jiazhi, Tian Rui, Du Jiangsu, et al. Mixrec: Orchestrating concurrent recommendation model training on cpu-gpu platform[C]//Proc of the 41st Int Conf on Computer Design. Piscataway, NJ: IEEE, 2023: 366−374

[29] Guo Huifeng, Guo Wei, Gao Yong, et al. Scalefreectr: Mixcache-based distributed training system for ctr models with huge embedding table[C]//Proc of the 44th Int Conf on Research and Development in Information Retrieval. New York: ACM, 2021: 129−1278

[30] Zhao Xiangyu, Wang Maolin, Zhao Xinjian, et al. Embedding in recommender systems: A survey[J]. arXiv preprint arXiv: 2310.18608, 2023.

[31] Zhang Hailin, Liu Zirui, Chen Boxuan, et al. Cafe: Towards compact, adaptive, and fast embedding for large-scale recommendation models[J]. Proc of the ACM on Management of Data, 2024, 2(1): 1−28

[32] Miao Xupeng, Zhang Minxu, Shao Yingxia, etal. PS-Hybrid: Hybrid communication framework for large recommendation model training[J]. Journal of Tsinghua University(Science and Technology), 2022, 62(9): 1417−1425 (in Chinese)
(苗旭鹏, 张敏旭, 邵蓥侠, 等. PS-Hybrid: 面向大规模推荐模型训练的混合通信框架[J]. 清华大学学报(自然科学版), 2022, 62(9): 1417−1425)

[33] Zhang Yuanxing, Chen Langshi, Yang Siran, et al. Picasso: Unleashing the potential of gpu-centric training for wide-and-deep recommender systems[C]//Proc of the 38th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2022: 3453−3466

[34] Acun B, Murphy M, Wang Xiaodong, et al. Understanding training efficiency of deep learning recommendation models at scale[C]//Proc of the 27th IEEE Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2021: 802−814

[35] Song Xiaoniu, Zhang Yiwen, Chen Rong, et al. Ugache: A unified gpu cache for embedding-based deep learning[C]//Proc of the 29th Symp on Operating Systems Principles. New York: ACM, 2023: 627−641

[36] Kaggle. Click-through rate prediction[EB/OL]. [2025-2-24]. https://www.kaggle.com/c/avazu-ctr-prediction.

[37] Zeng Chaoliang, Liao Xudong, Cheng Xiaodian, et al. Accelerating neural recommendation training with embedding scheduling[C]//Proc

of the 21st USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: Association, 2024: 1141−1156

[38] Agarwal S, Yan Chengpo, Zhang Ziyi, et al. Bagpipe: Accelerating deep recommendation model training[C]//Proc of the 29th Symp on Operating Systems Principles. New York: ACM, 2023: 348−363

[39] Youngeun K and Minsoo R. Training personalized recommendation systems from gpu scratch: Look forward not backwards[C]//Proc of the 49th Annual Int Symp on Computer Architecture. New York: ACM, 2022: 860−873

[40] Adam P, Sam G, Francisco M, et al. Pytorch: An imperative style, high-performance deep learning library[C]//Proc of the 33rd Inter Conf on Neural Information Processing Systems. Red Hook: Curran Associates Inc, 2019: 8026−8037

[41] Ma Kaihao, Yan Xiao, Cai Zhenkun, et al. Fec: Efficient deep recommendation model training with flexible embedding communication[J]. Proc of the ACM on Management of Data, 2023, 1(2): 1−21,

[42] Saeed G, Lan Guanghui, and Zhang Hongchao. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization[J]. Mathematical Programming, 2016, 155(1): 267−305

[43] Chanwon P and Jemin L. Mobile edge computing-enabled heterogeneous networks[J]. IEEE Trans on Wireless Communications, 2020, 20(2): 1038−1051

[44] Li Yun, Ma Hui, Wang Lei, et al. Optimized content caching and user association for edge computing in densely deployed heterogeneous networks[J]. IEEE Trans on Mobile Computing, 2020, 21(6): 2130−2142

[45] Taegeon U, Byungsoo O, Minyoung K, et al. Metis: Fast automatic distributed training on heterogeneous gpu[C]//Proc of the 2024 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2024: 563−578

[46] Ling Neiwen, Wang Kai, He Yuze, et al. Rt-mdl: Supporting real-time mixed deep learning tasks on edge platforms[C]//Proc of the 19th ACM Conf on Embedded Networked Sensor Systems. New York: ACM, 2021: 1−14

[47] Z K, Xu Qiang, Meng Jiayi, et al. Accumo: Accuracy-centric multitask offloading in edge-assisted mobile augmented reality[C]//Proc of the 29th Annual Int Conf on Mobile Computing and Networking. New York: ACM, 2023: 1−16

[48] Zhao M, Choudhary D, Tyagi D, et al. Recd: Deduplication for end-to-end deep learning recommendation model training infrastructure[J]. arXiv preprint, arXiv: 2211.05239, 2022

[49] Miao Xupeng, Zhang Hailin, Shi Yining, et al. Het: scaling out huge embedding model training via cache-enabled distributed framework[J]. Proc of the VLDB Endowment, 2021, 15(2): 312−320

[50] Adnan M, Maboud Y, Mahajan D, et al. Accelerating recommendation system training by leveraging popular choices[J]. Proc of the VLDB Endowment, 2021, 15(1): 127−140

[51] Wang Chunnan, Wang Hongzhi, Wang Junzhe, et al. Autosr: Automatic sequential recommendation system design[J]. IEEE Trans on Knowledge and Data Engineering, 2024, 36(11): 5647−5660

[52] Li Jiayu, He Zhiyu, Cui Yumeng, et al. Towards ubiquitous personalized music recommendation with smart bracelets[J]. Proc of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2022, 6(3): 1−34

[53] Wang Qinyong, Yin Hongzhi, Chen Tong, et al. Next point-of-interest recommendation on resource-constrained mobile devices[C]//Proc of the ACM Web Conf 2020. New York: ACM, 2020: 906−916

[54] Long Jing, Chen Tong, Nguyen Q, et al. Decentralized collaborative learning framework for next poi recommendation[J]. ACM Trans on Information Systems, 2023, 41(3): 1−25

[55] Muhammad K, Wang Q, O'Reilly-Morgan D, et al. Fedfast: Going beyond average for faster training of federated recommender systems[C]//Proc of the 26th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2020: 1234−1242

[56] Sun Zehua, Xu Yonghui, Liu Yong, et al. A survey on federated recommendation systems[J]. IEEE Trans on Neural Networks and Learning Systems, 2024, 36(1): 6−20

[57] Zhang Chunxu, Long Guodong, Zhou Tianyi, et al. Gpfedrec: Graph-guided personalization for federated recommendation[C]//Proc of the 30th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2024: 4134−4142

[58] Ding Yuchen, Zhang Siqing, Fan Boyu, et al. Fedloca: Low-rank coordinated adaptation with knowledge decoupling for federated recommendations[C]//Proc of the 18th ACM Conf on Recommender Systems. New York: ACM, 2024: 690−700

[59] Belal Y, Bellet A, Mokhtar S B, et al. Pepper: Empowering user-centric recommender systems over gossip learning[J]. Proc of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2022, 6(3): 1−27

[60] Xia S, Wei P, Liu Yanchen, et al. Reca: A multi-task deep reinforcement learning-based recommender system for co-optimizing energy, comfort and air quality in commercial building[C]//Proc of the 10th ACM Int Conf on Systems for Energy-Efficient Buildings, Cities, and Transportation. New York: ACM, 2023: 99−109

[61] Gao Ye, Ma Meiyi, Gordon K, et al. A monitoring, modeling, and interactive recommendation system for in-home caregivers: Demo abstract[C]//Proc of the 18th ACM Conf on Embedded Networked Sensor Systems. New York: ACM, 2020: 587−588

[62] Matam K, Ramezani H, Wang Fan, et al. Quickupdate: a real-time personalization system for large-scale recommendation model [C]//Proc of the 21st USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2024: 731−744

[63] Wang Zheng, Wang Yuke, Deng Jiaqi, et al. Rap: Resource-aware automated gpu sharing for multi-gpu recommendation model training and input preprocessing[C]//Proc of the 29th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2024: 964−979

[64] Yang Chen, Chen Jin, Yu Qian, et al. An incremental update framework for online recommenders with data-driven prior[C]//Proc of the 32th ACM Int Conf on Information & Knowledge Management.

New York: ACM, 2023: 4894−4900

[65] Sima C, Fu Y, Sit M K, et al. Ekko: Alarge-scale deep learning recommender system with low-latency model update[C]//Proc of the 16th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2022: 821−839

[66] Yu Keping, Guo Zhiwei, Shen Yu, et al. Secure artificial intelligence of things for implicit group recommendations[J]. IEEE Internet of Things Journal, 2021, 9(4): 2698−2707

[67] Deng Yongheng, Wang Guanbo, Yu Sheng, et al. Relayrec: Empowering privacy-preserving ctr prediction via cloud-device relay learning[C]//Proc of the 23rd ACM/IEEE Int Conf on Information Processing in Sensor Networks. Piscataway, NJ: IEEE, 2024: 188−199

[68] Wang Ruoxi, Fu Bin, Fu Gang, et al. Deep & cross network for ad click predictions[C]//Proc of the 23rd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2017: 1−7

[69] Chen Wenqiang, Zhan Lizhang, Ci Yuanlong, et al. Flen: leveraging field for scalable ctr prediction[J]. arXiv preprint arXiv: 1911.04690, 2019

[70] Adnan M, Maboud Y E, Mahajan D, et al. Heterogeneous acceleration pipeline for recommendation system training[C]//Proc of the 51st Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2024: 1063−1079

[71] He Gongshan, Zhao Chuanlei, Jiang Jinhu, etal. A survey of data storage technologies for deep learning[J/OL]. Chinese Journal of Computers, 2025 (in Chinese)
(贺巩山, 赵传磊, 蒋金虎, 等. 面向深度学习的数据存储技术综述[J/OL]. 计算机学报, 2025.)

[72] Xie Minhui, Lu Youyou, Wang Qing, et al. Petps: Supporting huge embedding models with persistent memory[J]. Proc of the VLDB Endowment, 2023, 16(5): 1013−1022

[73] Wei Yingcan, Langer M, Yu Fan, et al. A gpu-specialized inference parameter server for large-scale deep recommendation models[C]//Proc of the 16th ACM Conf on Recommender Systems. New York: ACM, 2022

[74] Goyal P, Dollár P, Girshick R, et al. Accurate, large minibatch sgd: Training imagenet in 1 hour[J]. arXiv preprint arXiv: 1706.02677, 2017.

[75] Kuhn H. The hungarian method for the assignment problem[J]. Naval research logistics quarterly, 1955, 2(1-2): 83−97

[76] Lopes P, Yadav S, Ilic A, et al. Fast block distributed CUDA implementation of the hungarian algorithm[J]. Journal of Parallel and Distributed Computing, 2019, 130: 50−62.

[77] Lawler E. Combinatorial optimization: networks and matroids[M]. Courier Corporation, 2001.

[78] Munkres J. Algorithms for the assignment and transportation problems[J]. Journal of the society for industrial and applied mathematics, 1957, 5(1): 32−38

[79] Kaggle. Display advertising challenge[EB/OL]. [2025-02-23]. https://www.kaggle.com/c/criteo-display-ad-challenge

[80] Tallis M and Yadav P. Reacting to variations in product demand: An application for conversion rate (cr) prediction in sponsored search[J]. arXiv preprint, arXiv: 1896.08211, 2018

[81] Delestrac P, Battacharjee D, Yang Simei, et al. Multi-level analysis of gpu utilization in ml training workloads[C]//Proc of 2024 Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2024: 1−6

[82] Shubha S, Shen Haiying, and Iyer A. Usher: Holistic interference avoidance for resource optimized ml inference[C]//Proc of the 18th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2024: 947−964

[83] Yuan Wei, Yang Chaoqun, Qu Liang, et al. Hide your model: A parameter transmission-free federated recommender system[C]//Proc of the 40th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2024: 611−624

[84] Zhang Ye, Deng Yongheng, Yue Sheng, et al. Dualrec: A collaborative training framework for device and cloud recommendation models[J]. IEEE Trans on Mobile Computing, 2025

[85] Lian Xiangru, Yuan Binhang, Zhu Xuefeng, et al. Persia: An open, hybrid system scaling deep learning-based recommenders up to 100 trillion parameters[C]//Proc of the 28th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2022: 3288−3298

[86] Lai Fan, Zhang Wei, Liu Rui, et al. Adaembed: Adaptive embedding for large-scale recommendation models[C]//Proc of the 17th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2023: 817−831

[87] Zhao Xiangyu, Liu Haochen, Fan Wenqi, et al. Autoemb: Automated embedding dimensionality search in streaming recommendations [C]//Proc of the 21st Int Conf on Data Mining. Piscataway, NJ: IEEE, 2021: 896−905

[88] Luo Qinyi, Wang Penghan, Zhang Wei, et al. Fine-grained embedding dimension optimization during training for recommender systems[J]. arXiv preprint arXiv: 2401.04408, 2024.

[89] Bahreini T, Badri H, Grosu D. Mechanisms for resource allocation and pricing in mobile edge computing systems[J]. IEEE Trans on Parallel and Distributed Systems, 2021, 33(3): 667−682.

[90] He Ying, Fang Jingcheng, Yu F R, et al. Large language models (llms) inference offloading and resource allocation in cloud-edge computing: an active inference approach[J]. IEEE Trans on Mobile Computing, 2024, 23(12), 11253−11264.

[91] Tan Haisheng, Wang Yi, Zhang Chi, et al. Asymptotically tight approximation for online file caching with delayed hits and bypassing[J]. IEEE Trans on Networking, 2025.

Li Guopeng, born in 1997. PhD candidate. His main research interests include edge intelligence, large language model-based agent, and machine learning system.

李国鹏, 1997 年生. 博士研究生. 主要研究方向为边缘智能、大模型驱动的智能体、机器学习系统.

Tan Haisheng, born in 1981. PhD, professor. Member of CCF. His main research interests include edge intelligence, and system and networking for AI.

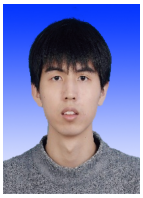
谈海生, 1981 年生. 博士, 教授. CCF 会员.主要研究方向为边缘智能、人工智能系统与网络.

Zhang Chi, born in 1995. PhD, associate professor. His main research interests include edge computing and network algorithms.

张　弛, 1995 年生. 博士, 副教授. 主要研究方向为边缘计算、网络算法.

Ni Hongqiu, born in 2000. PhD candidate. Her main research interests include edge computing, large language model inference and machine learning system.

倪宏秋, 2000 年生, 博士研究生. 主要研究方向为边缘计算、大语言模型推理、机器学习系统.



Wang Zilong, born in 2000. Master candidate. His main research interests include edge computing, scheduling mechanism, and machine learning system.

王子龙, 2000 年生. 硕士研究生. 主要研究方向为边缘计算、调度机制、机器学习系统.

Zhang Xinyue, born in 2000. PhD candidate. Her main research interests include edge computing, serverless computing, and machine learning system.

章馨月, 2000 年生, 博士研究生. 主要研究方向为边缘计算、服务器无感知计算、机器学习系统.

Xu Yang, born in 2003. Master candidate. His main research interests include edge computing, machine learning system, and large language model.

徐　洋, 2003 年生. 硕士研究生, 主要研究方向为边缘计算、机器学习系统、大语言模型.

Tian Han, born in 1989. PhD, associate professor. His main research interests include machine learning and its applications in networking, system and private computing.

田　晗, 1989 年生. 博士, 副研究员. 主要研究方向为机器学习及其在网络、系统、隐私计算中的应用.

Chen Guoliang, born in 1938. Professor. Fellow of CCF. His main research interests include parallel algorithms, computer architectures, and computational intelligence.

陈国良, 1938 年生. 教授. CCF 会士. 主要研究方向为并行算法, 计算机体系结构、计算智能.