

**本科生毕业设计（论文）参考文献译文**

译文出处：

*Tianyu Zhang, Kaige Liu, Jack Kosaian, Juncheng Yang, and Rashmi Vinayak. 2023. Efficient Fault Tolerance for Recommendation Model Training via Erasure Coding. Proc. VLDB Endow. 16, 11 (July 2023), 3137–3150. https://doi.org/10.14778/3611479.3611514*

|  |  |
| --- | --- |
| 院 系 | 计算机科学与技术 |
| 专业班级 | CS2103 |
| 姓 名 | 余正浩 |
| 学 号 | U202115404 |
| 指导教师 | 胡燏翀 |

2025年2月

**译文要求**

1. 译文内容须与课题（或专业内容）联系，并需在封面注明详细出处。
2. 出处格式为

图书：作者.书名.版本（第×版）.译者.出版地：出版者，出版年.起页～止页  
期刊：作者.文章名称.期刊名称，年号，卷号（期号）：起页～止页

1. 译文不少于5000汉字（或2万印刷符）。
2. 翻译内容用五号宋体字编辑，采用A4号纸双面打印，封面与封底采用浅蓝色封面纸（卡纸）打印。要求内容明确，语句通顺。
3. 译文及其参考文献原文一起装订，顺序依次为封面、译文、文献。
4. 翻译应在第七学期完成。

**译文评阅**

|  |
| --- |
| **导师评语**  应根据学校“译文要求”，对学生译文翻译的准确性、翻译数量以及译文的文字表述情况等做具体的评价后，再评分。 |
| 评分：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_（百分制） 指导教师（签名）：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  2025 年 2月 20 日 |

# 题目：通过纠删码实现推荐模型训练的高效容错

**摘要：**

基于深度学习的推荐模型（DLRM）被广泛用于提供个性化内容服务。除了使用神经网络，DLRM 还包含大型的、访问稀疏的嵌入表，这些嵌入表将分类特征映射到经过学习的密集向量表示。由于嵌入表规模庞大，DLRM 的训练通常会分散在数十或数百个节点的内存中。在如此大规模的系统里，节点故障很常见，为确保训练能在生产期限内完成，必须对其进行缓解。检查点机制是这类系统中实现容错的主要方式，但在正常运行期间和从故障中恢复时，都会产生显著的时间开销。随着 DLRM 规模的不断扩大，这些开销也会相应增加，对于未来预计规模还会更大的 DLRM 而言，检查点机制的开销将成为更严重的问题。这就需要重新思考 DLRM 训练中的容错策略。

我们提出了 ECRec，这是一个通过将纠删码与 DLRM 训练的独特特性相结合，实现高效容错的 DLRM 训练系统。ECRec 采用纠删码和复制不同 DLRM 参数的混合策略，能够正确且高效地更新冗余参数，让训练可以持续进行，同时保证恢复参数的一致性。我们在开源的工业级 DLRM 训练系统 XDL 之上实现了 ECRec。与检查点机制相比，ECRec 在大型 DLRM 上可将训练时间开销最多降低 66%，故障恢复速度提高 9.8 倍，并且在恢复过程中继续训练时，吞吐量仅下降 7 - 13%（而检查点机制在恢复时必须暂停训练）。

# 引言

基于深度学习的推荐模型（DLRMs）是在互联网规模上提供个性化内容服务的关键工具[10, 35]。DLRMs产生的价值依赖于其反映最新数据的能力，因此生产环境中的DLRMs需要频繁重新训练[2]。由于DLRMs应用广泛，其训练占用了大量计算资源，例如在Facebook，DLRM训练需求占比超过50%[34]。因此，减少DLRM训练时间对于维持模型的准确性和时效性[33]，以及降低资源消耗至关重要。

DLRMs由嵌入表和神经网络（NNs）组成。嵌入表将稀疏的分类特征（如城市名称）映射为经过学习得到的密集向量。嵌入表类似查找表，其中数百万甚至数十亿[17, 21]的稀疏特征各自映射到一个包含数十或数百个浮点数的小密集向量。一个小型神经网络对嵌入表“查找”得到的密集向量进行处理，以生成预测结果。我们将单个密集嵌入向量称为一个 “嵌入表条目”，简称 “条目”。

我们提出了ECRec，这是一个通过将纠删码与DLRM训练的独特特征相结合，实现高效容错的DLRM训练系统。ECRec在纠删码和复制不同DLRM参数之间采取混合策略，能够正确且高效地更新冗余参数，实现训练无中断进行，同时保持恢复参数的一致性。我们在开源的工业规模DLRM训练系统XDL之上实现了ECRec。与检查点机制相比，ECRec在大型DLRMs上最多可将训练时间开销降低66%，故障恢复速度提高9.8倍，并且在恢复过程中继续训练时，吞吐量仅下降7 - 13%（而检查点机制在恢复期间必须暂停训练）。

嵌入表通常非常大，大小从数百GB到数TB不等[21]。如此大规模的模型通常通过分布式方式在数十或数百个节点上进行训练[6, 21]，如图1a（小规模示例）所示。嵌入表和神经网络参数在服务器之间进行分片，并存储在内存中以实现快速访问。工作节点以数据并行的方式从服务器访问模型参数，并将梯度发送回服务器，通过优化器（如Adam[12]）更新参数。

由于模型参数存储在内存中，任何服务器故障都需要重新从头开始训练。鉴于DLRM训练资源消耗大、时间长，且大规模环境中故障频发，DLRM训练必须具备容错能力[16, 29]。例如，Facebook报告称，生产规模的DLRM训练中，故障间隔的中位数和平均值分别为8 - 17小时和14 - 30小时[29]。对于需要在紧迫期限内重新训练并部署的DLRMs来说，容错能力尤为重要[33]。在这项工作中，我们主要关注服务器故障的容错处理。服务器故障的处理至关重要，因为这会导致部分DLRM参数丢失。相比之下，工作节点故障的影响较小，因为工作节点不保存训练状态。

检查点机制是DLRM训练中用于容错的主要方法[16, 29]。这涉及定期暂停训练，并将当前参数和优化器状态写入稳定存储，如分布式文件系统。如果发生故障，整个系统将重置到最近的检查点，并从该点重新开始训练。虽然检查点机制简单，但频繁暂停训练以保存DLRM状态，且故障后需要重新进行部分训练。因此，在生产环境中训练DLRMs时，检查点机制会带来显著的开销，如在Facebook[29]。检查点机制还会消耗数据中心大量的网络和存储带宽，并且占用大量存储空间[16]。更令人担忧的是，这些开销会随着DLRM规模的增大而增加。考虑到模型规模不断增大的趋势[43, 48]（详见§2.2），检查点机制在未来训练更大规模的DLRMs时，预计会带来更大的开销。

另一种无需长时间恢复过程的替代方法是在单独的服务器上复制DLRM参数。然而，复制需要的内存至少是基于检查点机制系统的两倍，鉴于DLRMs规模庞大且还在不断增大，这种方法并不实际。

一种理想的DLRM训练容错方法应具备以下特点：（1）训练时间和内存开销低；（2）能快速从故障中恢复；（3）不会引入潜在的精度损失（以及相关的不确定性）。最后，这种解决方案应能随着DLRM规模的增加良好地扩展，以支持新兴的DLRMs。设计一个满足这些要求的系统是本文的目标。

纠删码是一种编码理论工具，它利用主动冗余（类似复制），但内存开销显著更低，已在存储和通信系统中广泛应用（例如[37, 41]）。与复制和传统检查点机制一样，纠删码不会改变训练精度。由于其内存开销低，纠删码为DLRM训练中的高效容错提供了潜力。如图1b所示，DLRM训练系统可以通过对来自不同服务器的k个参数进行编码，构建 “奇偶校验参数”。在这个例子中，通过编码函数p = e₀ + e₁ + e₂，由参数e₀、e₁和e₂形成一个奇偶校验值P，并存储在单独的服务器上。如果一台服务器发生故障，可以通过读取k个可用参数并执行纠删码的解码过程来恢复丢失的参数（例如，e₁ = p - e₀ - e₂ ）。

虽然在DLRM训练中利用纠删码看起来很有前景，但由于纠删码与DLRMs的独特方面之间的相互作用，这种应用场景也存在挑战。在这项工作中，我们深入研究了纠删码在DLRM训练系统中的应用，揭示了这些挑战，并提出了应对方案。最终成果是ECRec，这是一个基于对DLRM训练独特特征的深入理解，通过精心设计实现高效容错的DLRM训练系统。我们将在下面描述这些挑战以及ECRec如何克服它们。

**混合冗余**：我们在§3.2中表明，在DLRM训练中正确使用纠删码比复制需要更多的通信。因此，ECRec必须确定哪些参数应采用纠删码，以在内存和网络开销之间取得平衡。ECRec根据DLRMs的独特特征来做出这个决策：虽然嵌入表占据了DLRMs内存使用的绝大部分，但神经网络参数的梯度在更新过程中主导了网络带宽的消耗。例如，对于在MLPerf[30]中使用Criteo数据集[1]训练的DLRM，嵌入表占DLRM参数的99%，但其梯度仅占网络带宽的35%，而神经网络占所有参数的1%，但却消耗了65%的网络带宽（其他数据集/模型也有类似情况；详见§3.3）。基于这种不对称性，ECRec采用混合冗余策略，对嵌入表进行纠删编码，对神经网络参数进行复制。对嵌入表进行纠删编码可保持较低的内存开销，而复制神经网络参数可在不增加太多内存开销的情况下减少网络带宽消耗。

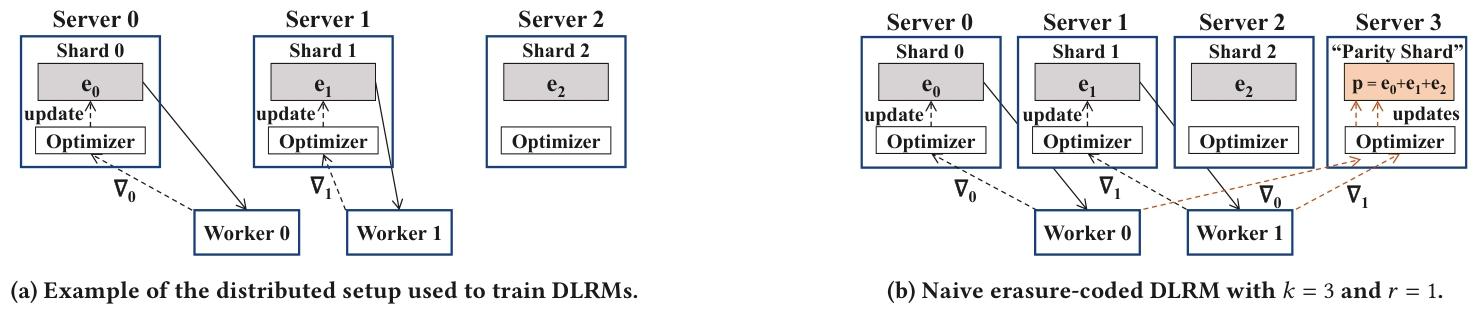
**保持正确性**：ECRec中的冗余参数必须与DLRM参数保持同步，以确保正确恢复。我们在§3中表明，在使用存储内部状态的优化器（如Adagrad、Adam）时，正确更新奇偶校验值且不产生大量内存开销是一项挑战。ECRec通过一种称为 “差异传播” 的方法，将更新奇偶校验值的责任从工作节点转移到服务器，并使用两阶段提交，从而克服了这些挑战。

**无暂停恢复**：纠删码的恢复过程可能资源消耗大，因为它涉及将所有可用数据读取到单个服务器并进行解码[40, 42]。这可能导致恢复时间过长，在此期间训练会停滞。ECRec通过在恢复过程中允许训练继续进行来实现快速恢复。ECRec利用按需重建丢失的DLRM参数来满足新的训练迭代需求，同时在后台进行完全恢复。同时，ECRec仔细确保新的训练更新不会与后台恢复过程冲突。

我们在开源的工业规模DLRM训练系统XDL[21]之上实现了ECRec，并使用MLPerf[30]中Criteo DLRM的变体进行评估。ECRec比检查点机制恢复得更快，且在训练大型DLRMs时训练时间开销更低。例如，ECRec从故障中恢复的速度比检查点机制的平均情况快9.8倍，并且在恢复过程中继续训练时，吞吐量仅下降7 - 13%，而检查点机制在恢复期间会暂停训练。这对于确保即使发生故障，DLRM训练也能在频繁生产部署所需的紧迫期限内完成至关重要。与检查点机制相比，ECRec在大型DLRM上最多可将训练时间开销降低66%。ECRec的优势随着DLRM规模的增大而更加明显，这表明它有望为当前和未来的DLRMs实现高效容错。

这项工作的主要贡献如下：

* 研究了容错机制与DLRMs独特特征之间的相互作用。
* 设计了ECRec，这是一个容错的DLRM训练系统，它明智地使用纠删码和复制，使训练在恢复过程中能够继续进行，并保持底层系统的一致性保证。
* 在各种场景下对ECRec进行评估，以展示其与检查点机制相比卓越的恢复性能，以及随着DLRM规模增大，在训练时间开销方面良好的扩展性，未来DLRMs的规模预计会继续增加。
* 据我们所知，ECRec是第一个采用纠删码提供容错能力的DLRM系统。



***图1：（a）正常训练和（b）（简单的）纠删码训练示例，涉及参数、、以及梯度和。为简化起见，图中仅展示了对嵌入表条目的访问和更新操作。***

# 深度学习推荐模型（DLRM）容错中的挑战

## DLRM 训练系统

回顾第 1 节内容，由于使用了嵌入表，DLRM 的规模较大，并且 DLRM 训练通常分布在一组服务器和工作节点之间（图 1a）。模型参数（嵌入表和神经网络参数）被分片存储在服务器内存中。在对一批数据进行训练迭代时，一个工作节点读取与该批次相关的嵌入表条目和所有神经网络参数，执行前向和反向传播以生成梯度（针对神经网络参数和嵌入表条目），并将梯度发送回存储被读取参数的服务器。每个服务器上的优化器（例如 Adam）使用这些梯度来更新参数。许多系统在训练 DLRM 时采用异步训练方式（例如 Facebook [6]、阿里巴巴 [21]）。因此，我们主要关注异步训练，但我们的工作也可以扩展到同步训练。

**有状态的优化器**。许多流行的优化器在更新参数时会使用每个参数的状态（例如 Adam [12]、Adagrad [13]、动量随机梯度下降法）。我们将这类优化器称为 “有状态的”。例如，Adagrad 会随时间跟踪每个参数的梯度平方和，并在更新参数时使用该值。优化器状态存储在服务器内存中，并且在相应参数更新时也会更新。随着 DLRM 规模的增长，这种状态会消耗大量内存。

## DLRM的独特特征

与其他深度网络相比，DLRM具有独特的特征。首先，如今许多深度网络都依赖大型神经网络，而DLRM通常使用小型神经网络，但嵌入表规模很大[20]。例如，一个常用于Criteo数据集[1]训练的DLRM，其嵌入表大小超过100GB，而神经网络参数却不到1GB。其次，DLRM的组件具有不同的访问模式。每个训练样本通常仅访问少量嵌入表条目，但会访问所有神经网络参数。例如，在Criteo数据集上，一批2048个训练样本平均仅访问约8900个嵌入表条目，这在DLRM约2亿个条目中仅占很小的比例。因此，嵌入表条目是稀疏更新的，而所有神经网络参数在每个训练批次中都会被更新。

**规模缩放趋势**。与其他深度模型类似，增加DLRM中的参数数量可以提高准确性。因此，近年来DLRM的规模急剧增大：2020年，Facebook使用的DLRM含有数千亿个参数，而如今的DLRM参数数量已远超一万亿个[33]。这种规模的增长主要是由DLRM中嵌入表条目的增加所驱动：Facebook报告称，2017年至2021年间，DLRM中嵌入表条目的数量增长了17.5倍[43]。预计这一规模增长趋势在未来还会持续[48]。

## 检查点机制及其缺点

回顾第1节内容，对于大规模DLRM训练而言，确保服务器的容错能力至关重要，而工作节点的容错则相对没那么受关注。同样回顾第1节可知，检查点机制是DLRM训练中用于容错的主要方法。

最近，Facebook报告称，检查点机制带来的开销平均占DLRM训练时间的12%，这些开销累计达到了超过1000机年的计算量[29]。检查点机制存在两个主要的时间成本：

**1. 正常运行期间的时间成本**。鉴于DLRM的规模庞大，将检查点写入稳定存储是一个缓慢的过程，并且在此期间训练会暂停，以确保保存的模型具有一致性。直观地说，正常运行期间检查点机制的开销会随着检查点的频率和DLRM的规模而增加，这在第5.2节中有实证说明。

**2. 恢复期间的时间成本**。发生故障时，使用检查点机制的系统必须通过从存储中读取检查点，将DLRM回滚到最近的检查点状态，并重新执行在该检查点与故障之间发生的所有训练迭代。在此期间，新的训练迭代会暂停。从存储中读取检查点所需的时间可能很长[29]，并且会随着DLRM规模的增大而增加。重新执行迭代的预期时间会随着检查点之间的时间间隔而增加：如果每隔T个时间单位写入一次检查点，那么这个时间最好的情况是0（在写入检查点后立即发生故障），最坏的情况是T（在写入检查点前发生故障），平均为 。我们在第5.3节展示了这种恢复性能。

**要点总结**。检查点机制在正常运行时的时间开销和恢复时的时间开销之间存在着根本性的权衡。增加检查点之间的时间间隔可以减少保存检查点时暂停的时间比例，但会增加恢复时预期要重新执行的工作量。为了在生产应用中部署DLRM时，即使发生故障也能满足紧迫的期限要求，需要在正常模式和恢复期间都保持较低的训练时间开销[33]。Facebook还指出，减少DLRM检查点机制所使用的存储和网络带宽，对于减轻这些共享资源的负载至关重要[16]。这些开销会随着模型规模的增大而增加。鉴于在第2.2节中提到的模型规模不断增大的趋势，检查点机制在未来DLRM训练中带来的开销预计会更大。

这就需要为DLRM训练寻找其他的容错方法，这些方法要能够适用于大规模的DLRM，并且在训练时间开销和恢复性能之间不存在严重的权衡问题。

## 降低检查点机制的开销

有多种技术可用于降低检查点机制的开销。接下来我们将对这些技术进行概述，并讨论将它们应用于DLRM训练时面临的挑战。

**近似方法**：最近的一些方法旨在通过采用近似检查点或近似恢复来降低检查点机制的开销[9, 16, 29, 38]。然而，在发生故障时，这些技术会将DLRM回滚到一个近似的状态，这可能会改变模型的收敛性和最终精度。鉴于DLRM产生的重大商业价值，先前的研究指出，即使是微小的精度下降也必须避免[52]。此外，我们与多位从事大规模DLRM训练的从业者进行交流时发现，这种潜在的精度下降会引入不确定性，使得调试生产环境中的DLRM变得困难。因此，理想情况下，人们希望在不影响精度的前提下降低检查点机制的开销。

**异步检查点**：另一种降低检查点机制开销的方法是采用异步检查点[5]。在这种方法中，训练在将模型状态写入稳定存储的同时继续进行。然而，这会导致存储在检查点中的模型状态不一致，因为检查点中的某些参数会反映比其他参数更新的迭代结果。这可能会导致恢复后的模型精度下降。

**日志记录**：有人可能会质疑，能否通过在梯度生成时将更新记录到稳定存储来降低检查点机制的开销。只有当写入存储的速度能够跟上梯度生成的速度时，这种方法才可行。由于DLRM训练系统中有许多工作节点是异步运行的，梯度生成的速度非常快，存储无法跟上。实际上，如果存储能够跟上速度，那么DLRM参数就可以存储在存储设备中，而不是内存中。因此，将更新记录到稳定存储对于DLRM训练来说并不可行。

## 内存冗余

除了检查点机制外，另一种方法是在系统中预留额外内存，以容错的方式在内存中冗余存储DLRM参数和优化器状态。

**复制**：如第1节所述，一种在内存中冗余保存参数的方法是在不同服务器上复制每个参数。然而，这需要的内存至少是无复制系统的两倍，考虑到DLRM的规模越来越大，这种方法并不实际。

**纠删码**：纠删码是一种编码理论工具，与复制相比，它能以显著更低的开销实现冗余[37, 41, 45]。纠删码对k个数据单元进行编码，生成r个冗余的 “奇偶校验单元”，这样在总共(k + r)个数据和奇偶校验单元中，任何k个单元都足以让解码器恢复原始的k个数据单元。纠删码的开销为，通过设置r < k，其开销比复制要低。这些特性使得纠删码在许多领域得到了广泛应用（例如[23, 37, 39, 41, 45, 47, 49]）。

例如，考虑图1b中简单的纠删码DLRM训练系统，其中参数、和存储在三个不同的服务器上。假设系统必须容忍一台服务器故障，一个参数为k = 3、r = 1的纠删码可以通过将奇偶校验单元编码为 ，并将这个奇偶校验单元存储在第四台服务器上来实现。假设存储的服务器发生故障，系统可以使用纠删码的减法解码器恢复： 。这种设置仅需多使用33%的内存，就能从任何一台服务器故障中恢复数据，而复制则需要多使用100%的内存。

**要点总结**。一种理想的DLRM训练容错方法应（1）在正常运行和恢复过程中都避免暂停；（2）内存开销低；（3）不会引入潜在的精度损失；（4）能够适用于大规模的DLRM。纠删码为实现这些目标提供了有前景的可能性。然而，在DLRM训练中使用纠删码存在一些挑战。我们将在接下来的部分描述这些挑战以及如何克服它们。

# ECRec：采用纠删码的 DLRM 训练系统

我们提出了 ECRec，这是一种容错的深度学习推荐模型（DLRM）训练系统，它在训练过程中无需暂停，恢复过程中也无需回滚，并且能提供与底层训练系统相同的精度保证。ECRec 是在研究纠删码与 DLRM 训练独特属性之间的相互作用的基础上设计出来的。

## ECRec概述

图2展示了传统DLRM训练系统和ECRec高级操作的示例，以及每个系统中单个服务器的详细内容。箭头表示单个工作节点执行更新时的数据流向。

在原始系统中（图2a），一个工作节点将神经网络参数的梯度和嵌入表条目的梯度发送到托管这些参数的服务器。如插图所示，服务器上的优化器读取相应条目的优化器状态和神经网络参数，并结合接收到的梯度来计算更新值，然后将其应用于相关参数。

**混合冗余**：图2b展示了ECRec维护嵌入表、神经网络参数和优化器状态的冗余版本（带阴影框）。ECRec根据每种参数在内存中的大小以及更新时使用的网络带宽，为其选择合适的冗余类型。

理想情况下，为了最小化内存开销，DLRM中的所有参数都应采用纠删码编码。然而，正如我们在3.3节中所示，在训练过程中正确更新奇偶校验比更新副本需要更多的通信。因此，对网络带宽占用较高的参数进行纠删码编码可能会增加相当大的开销。

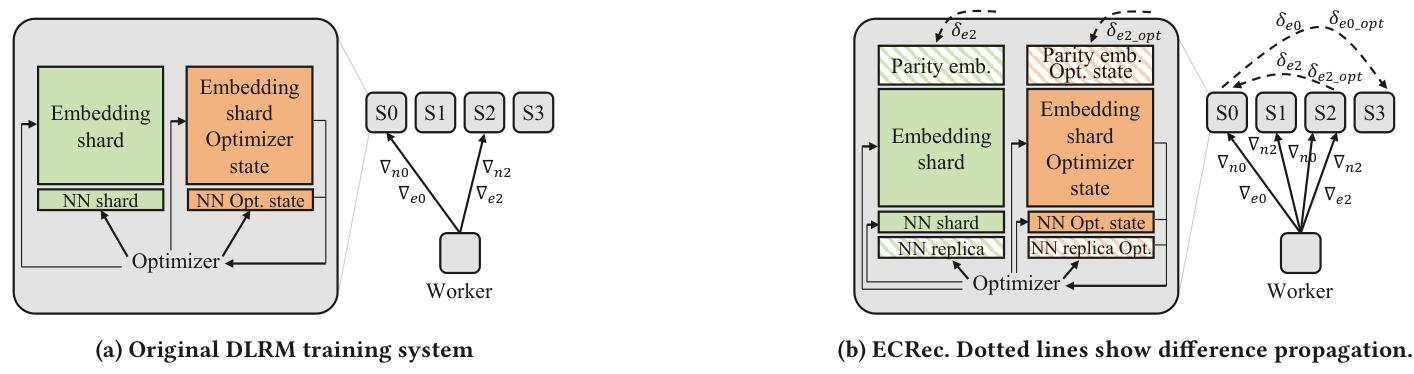
ECRec通过观察DLRM参数在内存和网络带宽占用上的不对称性来平衡这种权衡：嵌入表（及其优化器状态）占DLRM内存的大部分，而神经网络参数（及其优化器状态）只占一小部分。另一方面，神经网络参数的梯度在更新过程中占网络流量的大部分。我们将在3.3节中详细讨论这一点。

基于这种不对称性以及更新纠删码编码参数所需的额外网络流量，ECRec对嵌入表及其优化器状态进行纠删码编码，并复制神经网络参数及其优化器状态。这样做使得ECRec能够以较低的内存开销运行，因为DLRM的大部分内存占用都通过纠删码进行了处理，同时降低了网络带宽开销。我们将在3.2节中详细描述ECRec如何克服在嵌入表中使用纠删码的挑战，并在3.3节中解释为什么ECRec对神经网络参数采用复制的方式。

**更新冗余参数**：图2b还展示了ECRec如何使冗余参数保持最新。工作节点将神经网络参数的梯度发送到每个托管神经网络参数副本的服务器。然而，正如3.2节中所述，同样的过程不足以正确更新嵌入表条目及其优化器状态的奇偶校验。为了克服这个问题，ECRec利用了3.2节中提到的“差异传播”（图2b中的虚线），即托管嵌入表条目的服务器将该条目及其优化器状态更新所产生的差异转发到持有相应奇偶校验的服务器。

**恢复和一致性**：最后，ECRec能够在服务器故障恢复期间继续训练，并保持与底层DLRM训练系统相同的一致性保证。低开销的恢复对于在生产环境中部署DLRM时即使发生故障也能满足紧迫的期限至关重要。我们将分别在3.4节和3.5节中描述ECRec如何实现这些特性。

接下来，我们将详细描述ECRec的各个组件。



***图 2：（a）为在传统 DLRM 训练系统中，一名工作节点与 4 台服务器进行一次更新的示例；（b）为在 ECRec 中，k=3、r=1（包括对神经网络参数进行复制）时，一名工作节点与 4 台服务器进行一次更新的示例。左侧插图展示了单个服务器的详细情况。***

## 采用纠删码的嵌入表条目

如2.1节所述，嵌入表和优化器状态会在服务器间进行分片存储。ECRec对来自不同分片的k个嵌入表条目进行编码，生成一个 “奇偶校验条目”，并将其存储在单独的服务器上。优化器状态也会以类似的方式编码，形成 “奇偶校验优化器状态”，并与相应的奇偶校验条目存储在同一服务器上。

在ECRec中，只要k个相关的嵌入表条目中有任何一个发生更新，奇偶校验值就会被更新。因此，奇偶校验值的更新频率比原始条目更高，必须仔细安排其存储位置，以避免服务器之间出现负载不均衡的情况。ECRec采用轮转奇偶校验放置策略，在服务器之间分配奇偶校验值，使得每个服务器上的奇偶校验值数量相等。图3展示了k=3时的示例：每个服务器轮流被选中存储一个奇偶校验值，而用于编码该奇偶校验值的条目则存储在其他三台服务器上。这种方法的灵感来源于RAID-5系统中的奇偶校验放置策略[37]。

**编码器和解码器**：我们主要使用参数r=1的纠删码（即每k个条目对应一个奇偶校验值，可从单个故障中恢复数据），因为这代表了数据中心中最常见的故障场景[40]。不过，我们会在第4节介绍ECRec如何容忍多个故障。在r=1的设置下，ECRec使用图3所示的简单求和编码器以及相应的减法解码器：当k=3时，嵌入表条目、和被编码形成奇偶校验值。如果存储的服务器发生故障，可以通过 进行恢复。

3.2.1 正确更新奇偶校验值。图1b中简单的纠删码DLRM训练方法在使用有状态优化器（如Adam、Adagrad）时，无法正确更新奇偶校验条目。

以图1b中使用Adagrad[13]优化器的系统为例。设表示嵌入表条目在经过t次更新后的数值，表示的梯度。Adagrad使用梯度对进行的更新公式为：

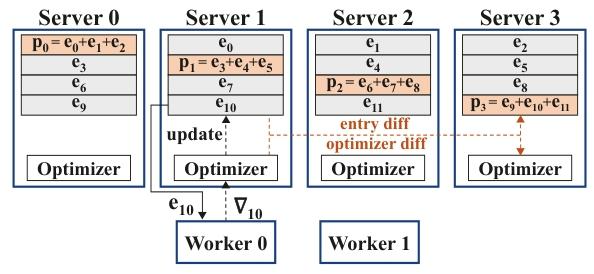
其中，，α是常数学习率，ϵ是一个小常数。，我们称之为的 “累加器”，是每个参数的优化器状态的一个示例。

如3.1节所述，ECRec为每k个原始优化器参数维护一个 “奇偶校验优化器参数”。在上述示例中，“奇偶校验累加器” 为。通过将每次对k个原始条目的更新所产生的梯度平方累加到上，很容易使它保持最新（例如，在对进行更新产生后，）。然而，使用这个奇偶校验累加器基于来更新奇偶校验条目（即在公式1中用代替，用代替）会导致奇偶校验条目错误，因为。

这个问题在任何有状态的优化器中都会出现，比如流行的Adagrad、Adam和动量随机梯度下降法（momentum SGD）。因此，ECRec在使用有状态优化器时，必须采取措施来维护正确的奇偶校验值。一种解决方法是在存储奇偶校验值的服务器上复制k个原始优化器参数。但是，嵌入表的优化器状态数据量很大，并且会随着嵌入表的增大而增加，使得这种复制方式并不实际。

**差异传播**。上述挑战源于直接将梯度发送到存储奇偶校验值的服务器：存储奇偶校验值的服务器仅接收到原始条目的梯度，必须正确更新奇偶校验条目和优化器状态。同时，将嵌入表条目从原始服务器发送到存储相应奇偶校验条目的服务器的方法也无法正确更新奇偶校验条目，因为存储奇偶校验条目的服务器需要访问嵌入表条目的先前版本才能生成梯度。为了克服这个挑战，ECRec采用了差异传播方法。

在差异传播中，工作节点仅将梯度发送到持有该梯度对应嵌入表条目的服务器。服务器在对条目应用优化器并更新优化器状态后，将条目和优化器状态的差异发送到持有相应奇偶校验条目的服务器。接收服务器将这些差异添加到奇偶校验条目和优化器状态中。如图2b所示，工作节点将条目的梯度发送到服务器0和2，然后服务器0和2将条目的差异和优化器状态的差异发送到存储相应奇偶校验值的服务器。通过向服务器发送差异而不是梯度，差异传播在使用有状态优化器时能够正确更新奇偶校验条目。



***图 3：k=3、r=1 时的轮转奇偶校验放置和差异传播示例***

## 复制的神经网络参数

接下来我们将介绍ECRec如何为神经网络参数及其优化器状态提供容错能力。

**为何神经网络需要额外的容错机制**：回顾2.1节内容，在DLRM训练中，每个训练迭代时，工作节点都会从服务器拉取所有神经网络参数。因此，每个工作节点都包含当前神经网络参数的近似副本。这可能会让人质疑，ECRec是否可以简单地将工作节点拉取的神经网络参数，当作跨服务器分片的神经网络的 “天然” 副本加以利用。虽然这种方法可以恢复神经网络参数，但它无法为神经网络参数使用的优化器状态提供容错能力。由于优化器状态保存在服务器上，工作节点不会读取，所以如果服务器发生故障，神经网络的优化器状态就会丢失。因此，需要一种能够同时使神经网络参数及其优化器状态保持冗余的替代方法。

**对神经网络参数进行纠删码编码导致的高开销**：有人可能会想，能否通过类似3.2节中的纠删码方式，来确保神经网络参数及其优化器状态的容错能力。毕竟，这些参数和嵌入表一样，都是在服务器之间进行分片存储的，所以同样的技术似乎也能适用。

然而，我们发现，按照3.2节中描述的方法对神经网络参数进行纠删码编码，会导致显著的性能开销。回顾2.2节内容，嵌入表是稀疏更新的，而所有神经网络参数及其优化器状态在每次训练迭代中都会被更新。这就导致了更新嵌入表条目和神经网络参数所消耗的网络带宽出现不平衡，神经网络参数消耗的带宽要大得多。例如，对于用于Criteo数据集训练的DLRM，我们发现，在一次给定的训练迭代中，更新神经网络参数所消耗的网络带宽，比更新嵌入表条目高出1.8倍以上。同样，对于在Avazu数据集上训练的DLRM的两个变体，我们发现神经网络在更新参数时消耗了80-90%的网络带宽。

使用差异传播的纠删码编码，在每次更新时需要额外200%的网络带宽，因为原始参数及其优化器状态的差异（两者大小都与梯度相同）都必须转发到存储奇偶校验的服务器。鉴于神经网络参数在网络带宽占用上的主导地位，对神经网络执行差异传播会增加相当大的训练时间开销。然而，正如3.2节所述，差异传播对于正确更新奇偶校验是必要的。

**复制神经网络参数**：ECRec利用了神经网络参数梯度所消耗的网络带宽，与神经网络参数大小之间的不对称性。虽然神经网络参数的梯度在更新过程中占据了大部分网络带宽，但神经网络参数在整个DLRM的规模中只占很小的一部分：对于用于Criteo和Avazu数据集的DLRM，神经网络参数及其优化器状态占整个DLRM规模的比例不到1%。因此，复制神经网络参数及其优化器状态，只会给整个系统增加很少的内存开销。

复制神经网络参数及其优化器状态，使ECRec在更新神经网络时，能够避免差异传播（及其相关的网络带宽开销）。在ECRec中，对于给定的神经网络参数，工作节点会将其梯度发送到托管该参数副本的两个服务器。每个持有副本的服务器会在本地更新参数及其优化器状态副本。通过这种方式，ECRec在更新神经网络参数时，所消耗的网络带宽只有采用差异传播的纠删码编码方式的一半：差异传播需要额外发送神经网络参数及其优化器状态的差异，而复制只需要多发送一次神经网络参数的梯度。

## 故障发生时的无暂停恢复机制

我们现在详细介绍ECRec如何在不暂停训练的情况下进行恢复。

由于ECRec中使用的纠删码具有这样的特性：在总共（k + 1）个原始单元和奇偶校验单元中，任意k个单元就足以恢复原始的k个单元。因此，即使有一台服务器发生故障，ECRec也能继续训练。例如，在图3中，即使服务器2发生故障，ECRec中的一个工作节点仍可以通过读取、和p，并解码来访问嵌入表条目。在纠删码系统中，这种需要解码的读取操作被称为 “降级读取”。同样，神经网络参数可以通过另一台服务器上的副本来访问。

**纠删码恢复中的挑战**。尽管ECRec能够进行降级读取，但它仍必须完全恢复发生故障的服务器，以应对未来可能出现的故障。对于神经网络参数而言，这很简单且高效，因为它们可以从副本服务器复制，并且规模较小。然而，先前关于纠删码存储的研究表明，完全恢复可能会耗费大量时间。在ECRec中，完全恢复需要解码故障服务器持有的所有嵌入表条目和优化器状态。这在传输用于解码的条目时会消耗大量网络带宽，在执行解码操作时会占用服务器CPU资源。鉴于嵌入表及其优化器状态的规模较大，在恢复训练之前进行完全恢复可能会显著暂停训练，这使得在发生故障时，难以满足生产训练部署的期限要求。

**通过粒度锁实现在恢复过程中继续训练**。ECRec并没有在故障发生后仅执行降级读取，也没有一直暂停直到完全恢复完成，而是在完全恢复的同时继续进行训练。一旦发生故障，ECRec就开始完全恢复丢失的嵌入表条目和优化器状态。与此同时，系统继续进行新的训练迭代，工作节点通过降级读取来访问故障服务器中的条目。

ECRec必须避免在某个条目用于恢复时对其进行更新。如果恢复过程读取到条目的新值，但奇偶校验的旧值（例如，因为更新尚未到达奇偶校验服务器），那么恢复的条目将是不正确的。ECRec使用粒度锁来避免这种情况。恢复过程会锁定一组它将解码的丢失条目。在锁定期间，对用于恢复这些锁定条目的更新会被缓冲在服务器内存中。工作节点通过缓冲区访问已更新但被锁定的条目。当锁被释放时，缓冲的更新会被应用到嵌入表中，然后锁定下一组条目。每个锁所涵盖的条目数量在切换锁的时间和服务器用于缓冲的内存开销之间引入了一种权衡，这可以根据部署级别的要求进行调整。

## 恢复一致性的DLRM

如2.4节所述，ECRec旨在避免为DLRM训练系统引入新的精度损失来源。接下来我们将描述ECRec如何维持其构建基础的异步DLRM训练系统的一致性保证。

在异步训练中，不同工作节点对参数的并发更新可能以任意顺序发生，并且可能相互覆盖。ECRec采用的冗余参数也是如此，这与原始系统的一致性保证相匹配。

然而，有一种情况需要特别注意：更新过程中服务器发生故障。为了更好地说明这一点，首先要注意每个训练迭代都会在不同服务器上更新参数。假设在当前迭代中，持有某个嵌入表条目的服务器在更新步骤中发生故障。如果在故障发生前，相应的奇偶校验条目已通过差异传播进行了更新，那么纠删码的解码函数将能够正确恢复。但是，如果服务器在传播其差异之前发生故障，恢复后的DLRM将不一致：其他参数将恢复到包含该迭代更新的状态，而有问题的恢复条目将不会反映该更新。

这个例子背后的问题在于，无法确定在故障发生前，奇偶校验条目是否已通过差异传播进行了更新。

**两阶段提交**。ECRec在更新参数时使用两阶段提交（2PC）来避免这种不一致情况。2PC协议由每个工作节点在其执行的每个训练迭代中进行协调。在ECRec中，2PC将在不同服务器上更新一组参数的过程分为两个阶段。在第一阶段，计算并暂存参数（原始参数和冗余参数）的更新。在第二阶段，将暂存的更新应用到参数（原始参数和冗余参数）上。这确保了在恢复开始之前，DLRM参数处于一致状态。

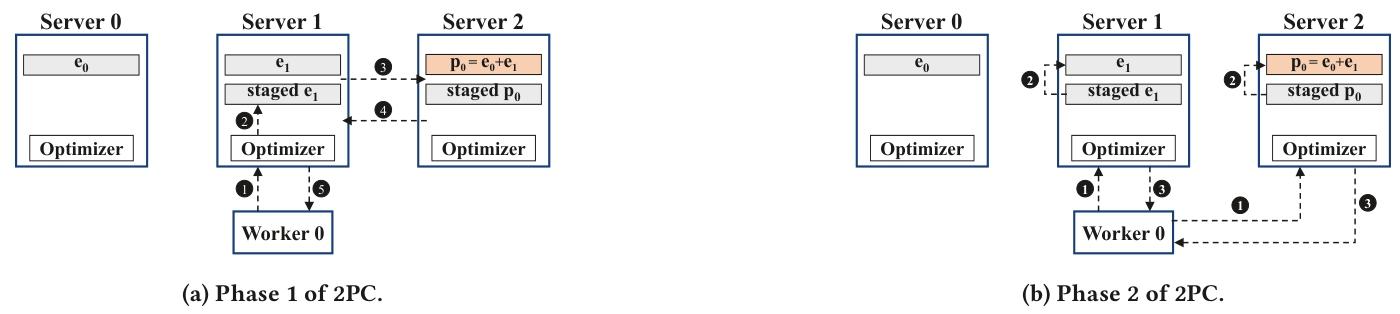
**2PC示例**。接下来我们通过一个简单的例子，来说明在k=2、r=1的ECRec中2PC是如何应用的，如图4所示。

图4a展示了协议的第一阶段。（1）工作节点将梯度发送到托管给定更新相关参数的服务器。（2）接收服务器使用优化器计算更新，并暂时暂存更新后的参数，而不是直接应用更新。（3）接收服务器执行差异传播，从而得到相应p参数的暂存版本。（4）通过差异传播接收到更新的服务器，向发送差异的服务器发送确认消息，以确认对p的更新已暂存。（5）在步骤1中接收到原始梯度的服务器向工作节点发送确认消息。一旦工作节点从所有它发送梯度的服务器收到确认消息，第一阶段就完成了，此时可以安全地开始第二阶段。

图4b展示了协议的第二阶段。（1）协调协议的工作节点向所有在上一阶段暂存了更新的服务器发送 “提交” 消息。（2）接收服务器将暂存的更新应用到相应的参数上。（3）接收服务器在应用暂存更新后，向协调工作节点发送确认消息。

**处理2PC中的故障**。如果在2PC的第一阶段发生故障，协调工作节点将中止并重新启动协议，这是标准操作。如果在第二阶段发生故障，协议将继续进行，但协调工作节点只会等待参与协议的非故障服务器的确认消息。这是安全的，因为该迭代的更新将已提交到足够的奇偶校验/副本和原始参数中，以便恢复过程能够重建故障服务器上的参数。

**权衡**。ECRec中的2PC为参数更新增加了一轮额外的通信。如第5章所示，这增加了训练时间开销。鉴于ECRec使用2PC仅仅是为了防止在发生故障时丢失部分更新，读者可能会质疑保留这些完整更新的重要性。毕竟，ECRec构建于其上的训练系统是异步的，这意味着工作节点的并发更新可能会相互覆盖。尽管如此，我们在ECRec中实现2PC是为了遵循2.4节中的设计目标，即在训练中不引入额外的不准确性来源。虽然这可能是一种比较严格的解决方案，但2PC确保了训练中使用的容错技术不会引入额外的不一致性来源（以及在调试准确性时相关的不确定性）。我们在5.2节中展示了，如果愿意放弃这些保证，可以通过关闭2PC来降低ECRec的开销。



***图4 ECRec中使用的两阶段提交协议示例，k=2，r=1***

# 讨论

**ECRec中的权衡**。ECRec将k个嵌入表条目编码为单个奇偶校验条目（r = 1的情况下）。参数k在ECRec中会产生以下权衡：

*增加k会降低内存开销和容错能力*：由于ECRec每k个条目编码一个奇偶校验条目，k值越大，所需的内存越少。然而，因为ECRec使用的纠删码可以从（k + 1）个故障中的任意一个中恢复，增加k会降低ECRec能够容忍的故障服务器比例。

*增加k不会改变正常操作期间的负载*：在ECRec中，每个嵌入表条目被编码以生成一个奇偶校验条目，因此每个应用于条目的更新也会应用于一个奇偶校验条目。因此，无论k值是多少，就执行的更新数量而言，总负载增加量为2。我们在5.2节中也展示了ECRec平衡了服务器间更新的总体负载。

*增加k会增加完全恢复的时间*：在ECRec中恢复嵌入表需要从不同服务器读取k个条目并进行解码。因此，恢复过程中使用的网络流量会随着k的增加而增加，这就增加了完全恢复的时间。不过，如3.4节所述，ECRec在此期间会继续训练。

**容忍多个故障**。回顾第3节内容，ECRec使用r = 1的纠删码，即可以从单个服务器故障中恢复。这一选择是基于先前对集群故障的研究，这些研究表明单节点故障是节点组中最常见的故障场景。

ECRec可以通过使用r > 1的纠删码轻松适应容忍更多故障的需求。另一种仍然利用r = 1的方法是，将训练中使用的整个集群划分为较小的服务器组，在每个组内执行r = 1的纠删码编码，这样每个组内发生多个故障的可能性就较小。最后，ECRec还可以采用比常规检查点方案低得多的频率进行检查点操作，作为应对并发故障的第二层防御措施。这种方法与多级检查点类似。

**利用细粒度访问频率**。先前的研究表明，在训练过程中，嵌入表条目的访问频率各不相同，少数条目占了总体访问的绝大部分。有人可能会考虑调整ECRec，使其考虑单个嵌入表条目的访问频率。例如，复制一部分访问频率最高的嵌入表条目，以减少对这些条目执行差异传播时的网络带宽开销，这可能是有益的。利用这种更细粒度的访问频率指标是未来工作的一个有前景的方向，但由于它会引入额外的权衡和系统复杂性，所以超出了本工作的范围。

# 评估

接下来我们对ECRec进行评估，主要结果如下：

* ECRec从故障中恢复的速度比检查点机制的平均恢复时间快高达9.8倍。快速恢复对于满足频繁重新训练和部署DLRM的生产应用的紧迫期限至关重要[33]。
* ECRec在恢复过程中能够继续训练，吞吐量仅下降7%-13%，而检查点机制在恢复期间必须暂停训练。
* 与检查点机制相比，ECRec在大型DLRM上的训练时间开销最多可降低66%。ECRec能够很好地适应DLRM规模的增长，显示出其在训练当前和未来DLRM方面的潜力。
* 虽然ECRec在更新奇偶校验时会引入额外负载，但通过改善集群负载平衡，减轻了增加的负载对训练吞吐量的影响。

## 评估设置

我们在阿里巴巴的开源DLRM训练系统XDL上用C++实现了ECRec。

**数据集和模型**：我们主要使用Criteo Terabyte数据集[1]进行评估，该数据集常用于评估DLRM训练系统。我们从数据集中随机抽取一天的样本，在一次遍历数据集中以的概率选取每个样本，并使用这个子集进行评估以减少存储需求。这种随机抽样得到的数据集能够模拟完整数据集。

我们专注于Criteo Terabyte数据集，因为它是DLRM最常用的公共数据集之一，并且其较大的规模更接近生产规模的数据集。虽然还有其他公共数据集可用，但许多数据集的嵌入表较小（例如小于1GB），不适合大规模实验。此外，Adnan等人[7]的研究表明，Criteo Terabyte数据集与许多其他公共数据集具有相似的特征。我们在3.3节中对DLRM训练的内存和网络使用情况的分析也反映了这种相似性。

我们基于MLPerf[35]中用于Criteo数据集的DLRM对各种DLRM进行评估，该DLRM有13个嵌入表，总共约2亿个条目，每个条目有128个密集特征。我们使用带动量的随机梯度下降（SGD）作为优化器，每个参数添加一个浮点值的优化器状态。也可以使用其他任何优化器。嵌入表和优化器状态的总大小为220GB。该DLRM使用一个七层的多层感知器作为神经网络，每层有128 - 1024个特征[4]。

我们评估不同规模的DLRM。首先，增加DLRM中嵌入表条目的数量（即稀疏维度），这会增加每个服务器所需的内存以及必须进行检查点存储/冗余保存和恢复的数据量。我们考虑上述原始Criteo DLRM的变体，其嵌入表条目数量分别为原始数量的1倍、2倍、4倍和8倍。我们将这些变体分别称为Criteo-Original、Criteo-2S、Criteo-4S和Criteo-8S，它们的大小分别为220GB、440GB、880GB和1760GB。我们主要关注嵌入表稀疏维度的扩展，因为这反映了当前观察到的一个显著扩展趋势：Facebook报告称，从2017年到2021年，DLRM中嵌入表条目的数量增加了17.5倍[43]。

为了全面评估，我们还对增加嵌入表条目大小（即密集维度）的DLRM进行评估。这会增加每个服务器所需的内存、必须进行检查点存储/冗余保存的数据量、传输条目/梯度所需的网络带宽以及工作节点和服务器的计算量。因此，这种扩展方式补充了模型在稀疏维度上的扩展。我们考虑上述原始Criteo-2S DLRM的一个变体，其中每个条目大小是原来的两倍，同时神经网络输入层的宽度也相应增加以适应更大的条目大小。我们将这个DLRM称为Criteo-2S-2D，其总大小为880GB。

**编码参数和基线**：我们评估了k值为2和4的ECRec，它们分别具有50%和25%的内存开销。尽管我们主要关注r = 1的情况，但这些实验也能让我们了解第4节中所述技术的性能，即通过将一组服务器划分为多个组，在每个组内执行r = 1的纠删码编码来容忍多个故障。默认情况下，我们在恢复过程中使用一个锁（见3.4节），但也评估了更细粒度的锁设置。

我们将ECRec与以下几种情况进行比较：（1）每30分钟向HDFS进行一次检查点存储（Ckpt-30）；（2）每60分钟向HDFS进行一次检查点存储（Ckpt-60）。我们还与（3）完全不进行检查点存储或容错（No FT）的情况进行比较。由于检查点机制的恢复时间取决于故障发生的时间（见2.3节），在评估恢复性能时，我们还将ECRec与检查点机制的最佳、平均和最差情况进行比较。向HDFS进行检查点存储代表了生产环境中DLRM训练的常见做法，因为生产中常使用类似HDFS的分布式文件系统[6, 16]。此外，我们使用的检查点基线具有竞争力：我们发现通过HDFS进行检查点存储仅比（特意设置的不切实际的）直接写入本地SSD的基线慢7%-27%。另外，对于代表当前DLRM的Criteo-Original，我们报告的检查点写入开销与Facebook在生产环境中报告的结果相似[29]。

我们还在5.4节中与使用近似检查点（可能导致精度损失）的方法进行比较。现有的关于训练中近似检查点的研究没有开源代码发布（例如[9, 16, 29, 38]），因此我们通过分析建模来模拟它们的性能，并与ECRec的实际系统性能进行比较（这种方式对基于近似检查点的基线更有利）。

我们不与复制嵌入表的方法进行比较，因为如第1节所述，复制大型嵌入表会带来2倍的内存开销，这在大规模场景中是不可行的。在愿意承担2倍内存开销进行复制的情况下，复制方法自然有望超越ECRec和基于检查点的方法的性能。

**集群设置**：我们在AWS上进行评估，使用5台r5n.8xlarge类型的服务器，每台服务器具有32个vCPU、256GB内存和25Gbps网络带宽（由于内存需求，对于大于440GB和880GB的DLRM，使用r5n.12xlarge和r5n.24xlarge类型的服务器）。我们使用15台p3.2xlarge类型的工作节点，每个工作节点配备一个V100 GPU、8个vCPU和10Gbps网络带宽。由于成本原因，我们无法进一步扩展集群规模，但我们根据实际部署（例如XDL[21]）选择了工作节点与服务器节点的比例，从而考虑到了相关的权衡。工作节点的批量大小为2048。对于检查点存储，我们使用15个额外的i3en.xlarge类型的HDFS节点，每个节点配备NVMe SSD和25Gbps网络带宽。所有实例均使用AWS ENA网络。

**评估指标**：对于正常运行期间的性能，我们测量训练吞吐量（样本/秒）和训练时间开销，训练时间开销是指训练一定数量样本所需时间增加的百分比。对于恢复过程，我们测量完全恢复一台故障服务器所需的时间以及恢复期间的训练吞吐量。

## 正常运行期间的性能

图5展示了在两小时的训练运行中，ECRec和检查点机制与无容错系统（因此没有开销）相比的训练时间开销。随着DLRM规模的增加，ECRec的训练时间开销增长非常缓慢，而检查点机制的训练时间开销显著增加。例如，从Criteo-Original到Criteo-8S，k = 4的ECRec训练时间开销仅增加了1.2，而Ckpt-30和Ckpt-60的训练时间开销分别增加了7.2倍和7倍。这使得ECRec显著降低了大型DLRM的训练时间开销：对于Criteo-8S，k = 4的ECRec训练时间开销为22%，而Ckpt-30和Ckpt-60的开销分别为65%和31%。虽然对于如此大规模的DLRM，可以减少检查点存储的频率，但这会带来恢复时间延长的负面影响（如5.3节所示）。

对于较小规模的DLRM，ECRec的训练时间开销确实比检查点机制高。然而，需要注意的是，DLRM的规模预计会不断增大[43]（见2.2节）。此外，如5.3节所示，与检查点机制相比，ECRec在恢复期间的性能有显著提升。因此，ECRec有望成为未来DLRM的可扩展解决方案，而无需在正常模式和恢复性能之间进行严重的权衡。

图6显示，ECRec的吞吐量略低于无容错系统（No FT），而Ckpt-30的吞吐量在与无容错系统的吞吐量之间波动，在进行检查点存储时会降至0。这使得Ckpt-30的平均吞吐量（虚线）低于ECRec。图7进一步表明，Ckpt-30的训练进度比ECRec慢。图8和图9显示，由于检查点存储频率降低，Ckpt-60在正常运行期间的吞吐量比Ckpt-30更接近ECRec。然而，这是以延长恢复时间为代价的，如5.3节所示。

**参数k的影响**：无论参数k的值是多少，ECRec在正常运行期间的网络带宽和CPU开销都是恒定的（见5.4节）。图5、图6和图7表明，k为2和4时，ECRec的性能相同。

**ECRec对负载不均衡的影响**：接下来，我们评估ECRec中奇偶校验放置对集群负载不均衡的影响。我们通过统计训练Criteo-Original时每个服务器上发生的更新次数来衡量负载。

在不使用纠删码的情况下，负载最高的服务器执行的更新次数是负载最低的服务器的2.28倍。相比之下，在k = 2和k = 4的ECRec中，这种负载差异分别为1.64倍和1.58倍。因此，ECRec引入的额外负载通过改善负载平衡得到了缓解。在ECRec中，与给定服务器的嵌入表条目对应的奇偶校验分布在所有其他服务器上。因此，一台服务器在非奇偶校验更新时所承受的相同负载量，也会分布到其他服务器上用于更新奇偶校验。虽然所有服务器的负载都会增加，但在没有纠删码的情况下负载最高的服务器，由于添加了纠删码，其负载增加可能最小，因为它为其托管奇偶校验的所有其他服务器的负载较低。因此，预计负载最高和最低的服务器之间的负载差异会减小。这样，虽然ECRec使服务器上的更新总数增加了一倍，但通过改善负载平衡，其影响得到了缓解。

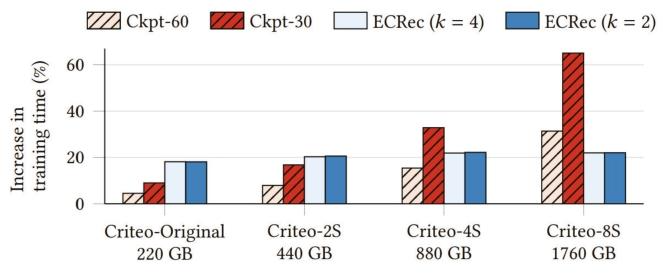
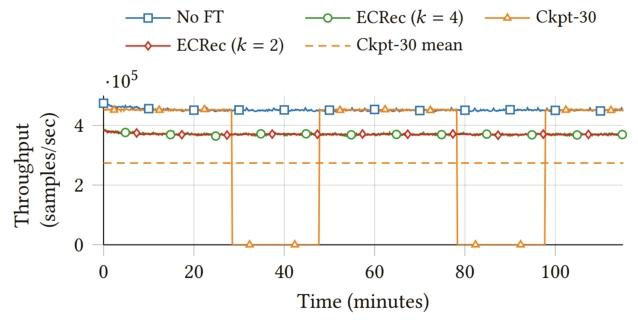
**条目宽度的影响**：我们现在评估增加每个嵌入表条目大小（即密集维度）对ECRec的影响。我们比较了在Criteo-4S和Criteo-2S-2D上k = 4的ECRec，这两个模型总大小相同，但Criteo-2S-2D的条目数量是Criteo-4S的一半，且每个条目大小是Criteo-4S的两倍。

k = 4时，ECRec在Criteo-4S上的训练时间开销为22%，在Criteo-2S-2D上为27%。Criteo-2S-2D上较高的开销可以用训练时增加的网络流量来解释：因为Criteo-2S-2D中的每个条目是Criteo-4S中条目的两倍大，通过差异传播传输嵌入表条目（及其梯度）在Criteo-2S-2D中消耗的网络带宽是Criteo-4S的两倍。

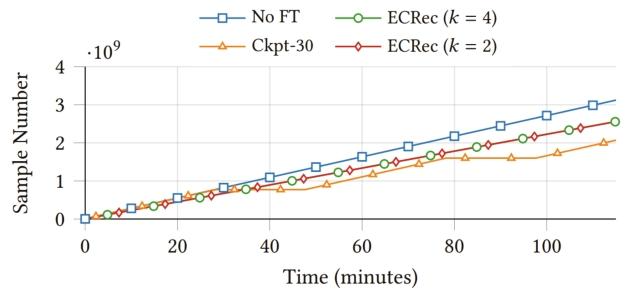
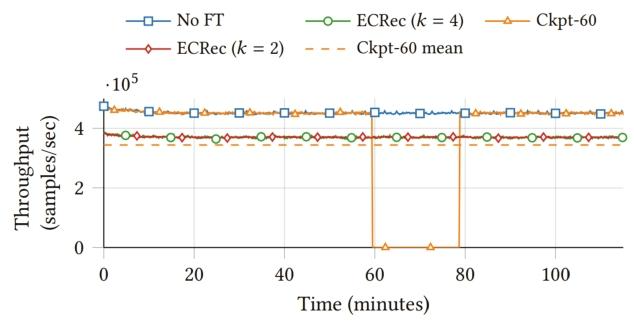
**消融研究**：接下来，我们研究ECRec各个组件对训练时间开销的贡献。

我们首先考虑ECRec中复制神经网络参数所带来的训练时间开销。我们将k = 4的ECRec与不复制神经网络的ECRec-NoRep版本进行比较。在Criteo-Original上，ECRec的训练时间开销为18.2%，而ECRec-NoRep的训练时间开销仅为5.2%。这表明在ECRec中保持神经网络副本最新所产生的额外网络流量，会带来相当大的训练时间开销。回顾3.3节内容，如果使用无状态优化器（例如SGD），则可以避免复制神经网络参数。使用此类优化器的用户可以通过关闭ECRec中的神经网络复制功能来潜在地降低开销。

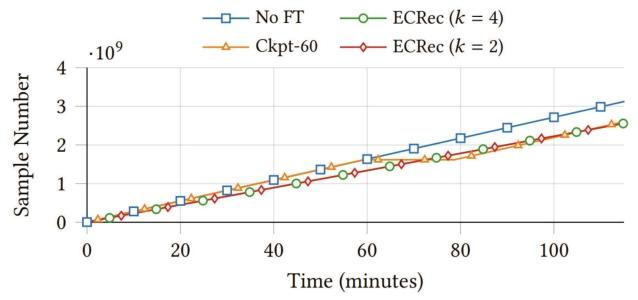
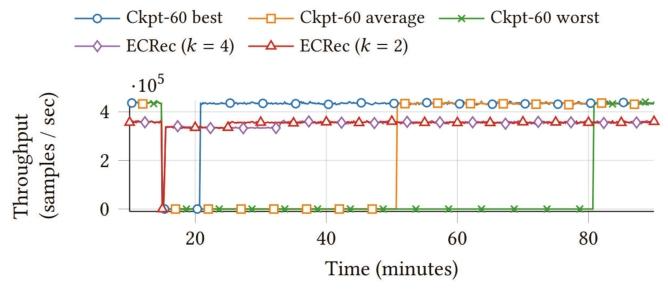
接下来，我们考虑在ECRec中使用两阶段提交（2PC）所带来的训练时间开销（见3.5节）。在Criteo-Original上，我们发现，在不使用2PC的情况下，ECRec-NoRep的训练时间开销从5.2%进一步降低到仅2.6%。如3.5节所述，ECRec中使用2PC是为了避免在发生故障时丢失正在进行的更新。对于那些愿意放弃这种（可能较小的）潜在精度损失的应用程序，可以关闭2PC。

***图5 训练时间开销 图6 Criteo-8S与Ckpt-30训练吞吐量对比***

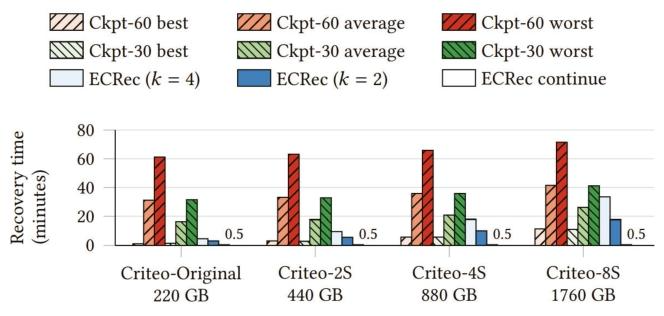
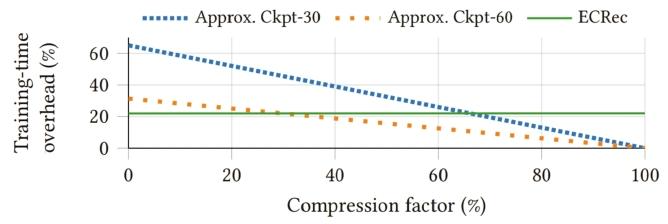
*** ***

***图 7 Criteo-8S 与 Ckpt-30 的训练进度对比 图8：Criteo-8S与Ckpt-60训练吞吐量对比。***

*** ***

***图9 Criteo-8S与Ckpt-60的训练进度对比 图 11 Criteo-4S 在 15 分钟出现故障后，与 Ckpt-60 的训***

***练吞吐量对比***

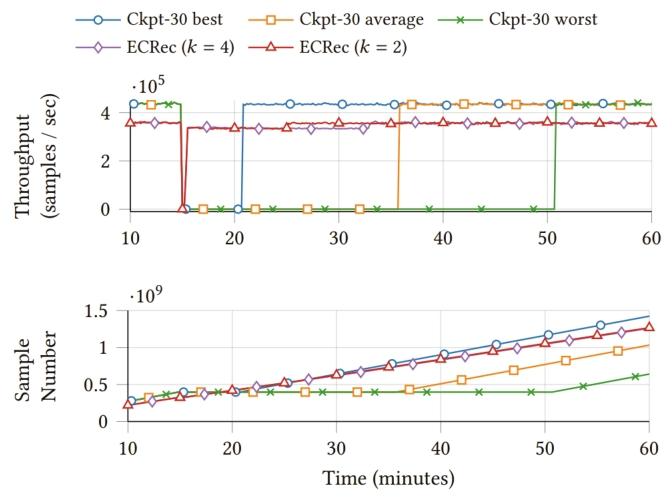
*** ***

***图12：完全恢复故障服务器所需的时间。“ECRec 图13：ECRec（k = 4；k = 2时情况类似）与基于近***

***continue”表示故障发生到ECRec在恢复过程中继 似检查点方法的训练时间开销对比。***

***续训练之间的时间。这个值很小，几乎难以察觉，因***

***此我们也在文中进行了说明。***

******

***图 10 Criteo-4S 在 15 分钟出现故障后，ECRec 与 Ckpt-30 的训练吞吐量（上图）和训练进度（下图）对比***

## 恢复期间的性能

接下来，我们评估ECRec和检查点机制在故障恢复方面的性能。图10最能体现两者的恢复性能对比，该图绘制了在Criteo-4S上，ECRec和Ckpt-30在15分钟服务器故障后的吞吐量和训练进度。ECRec的完全恢复速度比Ckpt-30的平均情况更快，关键是在恢复期间，其吞吐量能保持在正常运行期间的7%-13%以内。相比之下，Ckpt-30在恢复期间无法进行新的迭代。如图10底部所示，ECRec在恢复期间的高吞吐量使其训练进度比Ckpt-30的平均情况更快。图11表明，ECRec在恢复时间和吞吐量方面相对于Ckpt-60的提升更大。

图12展示了ECRec、Ckpt-30和Ckpt-60恢复一台故障服务器所需的时间。k = 4的ECRec比Ckpt-60和Ckpt-30的平均恢复时间分别快1.2 - 6.7倍和0.8 - 3.5倍（k = 2时，速度提升最高可达9.8倍）。更重要的是，与检查点机制不同，ECRec能够在故障发生后30秒内以高吞吐量继续训练。

**参数k和DLRM规模的影响**：图12显示，参数k的值越高，ECRec完全恢复所需的时间越长。然而，对于每个k值，在发生故障后，ECRec都能在恢复期间保持高吞吐量，并且能在30秒内恢复高吞吐量训练（另见图10）。

图12还表明，正如预期的那样（见2.3节和第4节），ECRec和检查点机制的完全恢复时间都会随着DLRM规模的增加而增加。由于ECRec在解码时单个服务器读取的数据和计算量增加了k倍，其恢复时间随DLRM规模的增长速度比检查点机制更快。但这对训练的影响不大，因为ECRec可以在恢复期间以高吞吐量继续训练。

**锁粒度的影响**：接下来，我们比较k = 4时，使用1个锁和10个锁的ECRec的完全恢复时间（见3.4节）。对于Criteo-8S，使用10个锁使恢复时间增加了6.5%，对于Criteo-Original则增加了24.8%。即使采用更细粒度的锁，从而导致恢复时间更长，ECRec在恢复期间仍能保持高训练吞吐量，这与检查点机制不同。切换锁涉及（1）暂时同步工作节点和服务器，以及（2）将更新后的嵌入表条目从缓冲区复制到原始条目。同步时间与DLRM规模无关，是恒定的，而复制缓冲区的时间会随着DLRM规模的增加而增长。因此，在较大规模的DLRM上，同步时间的分摊效果更好，从而降低了锁切换的开销。

## 与近似检查点的比较

我们现在将ECRec与使用近似方法来降低检查点开销的方法进行比较。如前所述，ECRec与这些方法的不同之处在于，它能保持与底层训练系统相同的精度保证，而近似方法可能会降低精度。这种潜在的精度下降可能需要更长的训练时间才能达到目标精度，并且会使调试模型性能变得困难。现有的关于训练中近似检查点的研究没有开源代码发布（例如[9, 16, 29, 38]），所以我们通过分析建模来模拟它们的性能，并与ECRec的实际系统性能进行比较（这种方式对基于近似检查点的基线更有利）。

**压缩检查点**：我们首先将ECRec与压缩检查点的方法（例如通过使用较低精度，如Check-N-Run[16]）进行比较。图13展示了在Criteo-8S上，不同压缩程度下ECRec和检查点基线的训练时间开销。由于ECRec不使用检查点，其开销与压缩因子无关，保持恒定。检查点基线的开销随着压缩因子的增加而线性下降（尽管这个分析模型没有考虑执行压缩的时间开销，因此对这些方法更有利）。图13显示，Ckpt-30需要超过60%的压缩因子才能与ECRec的训练时间开销相匹配。即使达到这种压缩水平，检查点机制仍然需要较长的恢复过程，并且从近似版本恢复的DLRM可能与原始DLRM的精度不匹配。

**部分恢复**：其他技术采用部分恢复，即仅将故障服务器上托管的参数在发生故障时回滚到检查点（例如 CPR [29]）。部分恢复的性能很难建模。一方面，部分恢复的时间可以潜在地建模为仅加载 DLRM 丢失部分的检查点所需的时间。另一方面，由于恢复后的 DLRM 状态不一致，使用部分恢复达到特定精度的总时间可能与 ECRec 或非近似检查点系统的时间不同。因为这种额外的训练时间取决于许多因素（例如数据集、神经网络、故障发生的时间、丢失的参数等），所以很难建模，从业者也难以推断。相比之下，ECRec 不会引入这些问题，并且具有更直接的恢复语义。

# 相关工作

**DLRM系统**：深度学习推荐模型（DLRM）训练和推理的系统支持最近受到了关注。关于改进DLRM推理的工作涵盖了从工作负载/系统分析（如[20, 28]）、模型 - 系统协同设计（如[17, 19]）到专用硬件（如[22, 46]）等多个方面。与DLRM训练相关的工作包括来自大型机构的系统（例如[6 - 8, 21, 24, 33, 35, 43, 52]）以及模型 - 系统协同设计（如[18, 48, 50]）。ECRec与这些工作的不同之处在于它专注于DLRM训练的容错性，并在其中使用了纠删码。ECRec可以在许多这些工作的基础上运行。

**检查点技术**：计算机系统长期以来一直使用检查点技术实现容错（例如[11, 25, 32]）。近期的工作对神经网络训练中的检查点技术进行了优化（如[31, 36]），但并未专注于DLRM。相比之下，ECRec利用DLRM训练的独特特性，运用纠删码实现容错。其他工作开发了基于近似技术的方法，以减少训练中检查点的开销（如[9, 38]）。而ECRec不会改变底层训练系统的准确性保证。有两项工作专注于降低DLRM训练中检查点的开销。Maeng等人（[29]）使用部分恢复的方法来减少故障后回滚的开销：只有故障节点回滚到其最近的检查点。Eisenman等人（[16]）使用增量检查点技术并降低检查点参数的精度。这两项工作在从故障中恢复时都可能降低DLRM训练的准确性。虽然这两项工作通过实证表明精度下降幅度较小，但它们无法提供与底层训练系统相同的准确性保证。ECRec在两个方面与这些工作不同：（1）ECRec保持与底层训练系统相同的准确性保证。这避免了关于容错方法是否能提供符合部署所需准确性的模型的不确定性，并且减少了在存在多种不准确来源时调试模型准确性的工作量。（2）ECRec利用内存冗余来降低容错的开销。

**纠删码系统**：纠删码在各个领域被广泛用于容错、负载均衡和缓解性能下降（例如[37, 39, 41, 47]）。近期的工作也将编码理论的思想应用于神经网络推理（[26]）以及某些类别的模型训练中（例如[15, 27, 44, 51]）。相比之下，ECRec专注于DLRM训练，这与这些工作所考虑的场景有显著差异。

# 结论

ECRec是一个容错的深度学习推荐模型（DLRM）训练系统，它采用纠删码技术来克服检查点技术的缺点。ECRec利用DLRM训练的独特特性，对DLRM的嵌入表使用纠删码，同时复制神经网络参数，从而采用混合方式实现内存冗余。ECRec以较低的开销维护最新的冗余参数，使训练在恢复过程中能够继续进行，同时保持与底层训练系统相同的准确性保证。与检查点技术相比，ECRec最多可将训练时间开销降低66%，故障恢复速度提高9.8倍，并且允许训练不间断地进行。随着DLRM规模的增加，ECRec能够很好地扩展，不会在训练时间开销和恢复性能之间进行严格的权衡。虽然ECRec的优势伴随着额外的内存需求和服务器负载，但内存开销只是一小部分，并且负载得到了均匀分布。ECRec展示了纠删码作为一种比检查点技术更优越的替代方案，用于当前和未来DLRM训练中的容错。探索ECRec在其他学习系统中的适用性是一个值得追求的令人兴奋的未来研究方向。

# 主要参考文献

1. Criteo Labs: Download Terabyte Click Logs <https://labs.criteo.com/2013/12/download-terabyte-click-logs/>. Last accessed 10 July 2023.
2. Introducing NVIDIA Merlin HugeCTR: A Training Framework Dedicated to Recommender Systems. <https://tinyurl.com/yy82pd2l>. Last accessed 10 July 2023.
3. Kaggle Avazu CTR Prediction Contest. <https://www.kaggle.com/c/avazu-ctrprediction>. Last accessed 10 July 2023.
4. MLPerf Inference Github Repository. <https://github.com/mlperf/inference>. Last accessed 10 July 2023.
5. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016.
6. Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim Hazelwood. Understanding Training Efficiency of Deep Learning Recommendation Models at Scale. In 2021 IEEE International Symposium on High Performance Computer Architecture (HPCA 21), 2021.
7. Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J. Nair. Accelerating Recommendation System Training by Leveraging Popular Choices, 2022.
8. Saurabh Agarwal, Ziyi Zhang, and Shivaram Venkataraman. BagPipe: Accelerating Deep Recommendation Model Training. arXiv preprint arXiv:2202.12429, 2022.
9. Yu Chen, Zhenming Liu, Bin Ren, and Xin Jin. On Efficient Constructions of Checkpoints. In Proceedings of the International Conference on Machine Learning (ICML 20), 2020.
10. Paul Covington, Jay Adams, and Emre Sargin. Deep Neural Networks for YouTube Recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, 2016.
11. John T Daly. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. Future Generation Computer Systems, 22(3):303–312, 2006.
12. Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In International Conference on Learning Representations (ICLR 15), 2015.
13. John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12(7), 2011.
14. Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The Design and Operation of CloudLab. In 2019 USENIX Annual Technical Conference (USENIX ATC 19), 2019.
15. Sanghamitra Dutta, Ziqian Bai, Haewon Jeong, Tze Meng Low, and Pulkit Grover. A Unified Coded Deep Neural Network Training Strategy Based on Generalized Polydot Codes for Matrix Multiplication. In Proceedings of the 2018 IEEE International Symposium on Information Theory (ISIT 18), 2018.
16. Assaf Eisenman, Kiran Kumar Matam, Steven Ingram, Dheevatsa Mudigere, Raghuraman Krishnamoorthi, Krishnakumar Nair, Misha Smelyanskiy, and Murali Annavaram. Check-N-Run: A Checkpointing System for Training Deep Learning Recommendation Models. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), 2022.
17. Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim Hazelwood, Asaf Cidon, and Sachin Katti. Bandana: Using Non-Volatile Memory for Storing Deep Learning Models. In The Second Conference on Systems and Machine Learning (SysML 19), 2019.
18. AA Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. In 2021 IEEE International Symposium on Information Theory (ISIT 21), 2021.
19. Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, GuYeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. DeepRecSys: A System for Optimizing End-to-End At-Scale Neural Recommendation Inference. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA 20), 2020.
20. Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. The Architectural Implications of Facebook’s DNN-based Personalized Recommendation. In 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA 20), 2020.
21. Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, et al. XDL: An Industrial Deep Learning Framework for High-Dimensional Sparse Data. In Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data, 2019.
22. Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B Preußer, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, et al. MicroRec: Accelerating Deep Recommendation Systems to Microseconds by Hardware and Data Structure Solutions. In The Fourth Conference on Systems and Machine Learning (MLSys 21), 2021.
23. Saurabh Kadekodi, Francisco Maturana, Suhas Jayaram Subramanya, Juncheng Yang, K. V. Rashmi, and Gregory R. Ganger. Pacemaker: Avoiding heart attacks in storage clusters with disk-adaptive redundancy. In Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation, OSDI’20, USA, 2020. USENIX Association.
24. Dhiraj Kalamkar, Evangelos Georganas, Sudarshan Srinivasan, Jianping Chen, Mikhail Shiryaev, and Alexander Heinecke. Optimizing Deep Learning Recommender Systems’ Training On CPU Cluster Architectures. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 20), 2020.
25. Richard Koo and Sam Toueg. Checkpointing and Rollback-Recovery for Distributed Systems. IEEE Transactions on software engineering, (1):23–31, 1987.
26. Jack Kosaian, K. V. Rashmi, and Shivaram Venkataraman. Parity Models: Erasure-Coded Resilience for Prediction Serving Systems. In Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP 19), 2019.
27. Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding Up Distributed Machine Learning Using Codes. IEEE Transactions on Information Theory, July 2018.
28. Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. Understanding Capacity-Driven Scale-Out Neural Recommendation Inference. In 2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 21), 2021.
29. Kiwan Maeng, Shivam Bharuka, Isabel Gao, Mark C Jeffrey, Vikram Saraph, Bor-Yiing Su, Caroline Trippel, Jiyan Yang, Mike Rabbat, Brandon Lucia, et al. CPR: Understanding and Improving Failure Tolerant Training for Deep Learning Recommendation with Partial Recovery. In The Fourth Conference on Systems and Machine Learning (MLSys 21), 2021.
30. Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. MLPerf Training Benchmark. In The Third Conference on Systems and Machine Learning (MLSys 20), 2020.
31. Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. CheckFreq: Frequent, Fine-Grained DNN Checkpointing. In 19th USENIX Conference on File and Storage Technologies (FAST 21), 2021.
32. Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R De Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC 10), 2010.
33. Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. Software-Hardware Co-design for Fast and Scalable Training of Deep Learning Recommendation Models. arXiv preprint arXiv:2104.05158, 2021.
34. Maxim Naumov, John Kim, Dheevatsa Mudigere, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, et al. Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems. arXiv preprint arXiv:2003.09518, 2020.
35. Bogdan Nicolae, Jiali Li, Justin M Wozniak, George Bosilca, Matthieu Dorier, and Franck Cappello. DeepFreeze: Towards Scalable Asynchronous Checkpointing of Deep Learning Models. In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid 20), 2020.
36. David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 88), 1988.
37. Aurick Qiao, Bryon Aragam, Bingjing Zhang, and Eric Xing. Fault Tolerance in Iterative-Convergent Machine Learning. In International Conference on Machine Learning, pages 5220–5230, 2019.
38. K. V. Rashmi, Mosharaf Chowdhury, Jack Kosaian, Ion Stoica, and Kannan Ramchandran. EC-Cache: Load-Balanced, Low-Latency Cluster Caching with Online Erasure Coding. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016.
39. K. V. Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A Hitchhiker’s Guide to Fast and Efficient Data Reconstruction in Erasure-Coded Data Centers. In Proceedings of the 2014 ACM SIGCOMM Conference (SIGCOMM 14), 2014.
40. Luigi Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. ACM SIGCOMM Computer Communication Review, 27(2):24–36, 1997.
41. Mahesh Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. Proceedings of the VLDB Endowment, 6(5), 2013.
42. Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. Recshard: Statistical feature-based memory optimization for industry-scale neural recommendation. In Proceedings of the Twenty-Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 22), 2022.
43. Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient Coding: Avoiding Stragglers in Distributed Learning. In International Conference on Machine Learning (ICML 17), 2017.
44. Hakim Weatherspoon and John D Kubiatowicz. Erasure Coding vs. Replication: A Quantitative Comparison. In International Workshop on Peer-to-Peer Systems (IPTPS 2002), 2002.
45. Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. RecSSD: Near Data Processing for Solid State Drive Based Recommendation Inference. In Proceedings of the Twenty-Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 21), 2021.
46. Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A Chien, and Haryadi S Gunawi. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In 15th USENIX Conference on File and Storage Technologies (FAST 17), 2017.
47. Jie Amy Yang, Jianyu Huang, Jongsoo Park, Ping Tak Peter Tang, and Andrew Tulloch. Mixed-Precision Embedding Using a Cache. arXiv preprint arXiv:2010.11305, 2020.
48. Juncheng Yang, Anirudh Sabnis, Daniel S. Berger, K. V. Rashmi, and Ramesh K. Sitaraman. C2DN: How to harness erasure codes at the edge for efficient content delivery. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), pages 1159–1177, Renton, WA, April 2022. USENIX Association.
49. Chunxing Yin, Bilge Acun, Xing Liu, and Carole-Jean Wu. TT-Rec: Tensor Train Compression for Deep Learning Recommendation Models. In The Fourth Conference on Systems and Machine Learning (MLSys 21), 2021.
50. Qian Yu, Netanel Raviv, Jinhyun So, and A Salman Avestimehr. Lagrange Coded Computing: Optimal Design for Resiliency, Security and Privacy. In Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS 19), 2019.
51. Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. AIBox: CTR Prediction Model Training on a Single Node. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM 2019), 2019.