用于区块链上人工智能的有效工作量证明

Andrei Lihu[1], Jincheng Du[2], Igor Barjaktarević[1],
Patrick Gerzanics[1] and Mark Harvilla[2]

[1] *Up and Running Software*
[2] *Oben*
Email: andrei.lihu@upandrunningsoftware.com, jincheng@oben.com,
igor.barjaktarevic@upandrunningsoftware.com,
patrick.gerzanics@upandrunningsoftware.com, mark@oben.com

比特币的开采是一个十分消耗时间和资源的过程。每添加一个区块链交易区块，矿工们都必须花费大量精力。比特币协所用的工作量证明（**Proof of Work, PoW**）形如一个抽签机制，底层的计算工作并没有其他用途。在本文中，我们基于在区块链上训练机器学习模型，描述了一种新颖的"有效工作量证明"（**Proof of Useful Work, PoUW**）协议。矿工在诚实地进行一定量的机器学习训练工作之后可以得到一次铸造新虚拟货币的机会。客户向网络提交机器学习训练任务并支付所有贡献者们。这是对于参与网络的一种额外激励，因为系统不仅仅依赖于抽签机制。通过我们的共识协议，各方利益相关者可以在分布式环境中订购，完成和验证有效工作量。同时，我们概述了奖励有效工作量和惩罚恶意行为者的机制，因为我们致力于利用区块链的安全性创建更好的人工智能系统。

Keywords: Machine learning, blockchain, proof of useful work, mining, PAI coin, project PAI, artificial intelligence.

1. 概述

比特币 [1] 矿工正在花费大量电力来向区块链添加新的区块。区块链是一条区块的序列，每个区块都包含若干笔交易。区块的头部包含前一个区块的加密哈希，网络目标值和 nonce。在传统的 *工作量证明* (PoW) 协议中，如果要添加新区块，区块头部的 *双重 SHA-256 哈希*必须小于所有网络参与者都知道的网络目标值。nonce 是可变的以获得不同的哈希值。对于每个成功开采的区块，挖出该区块的矿工被允许凭空创造特定数量的新币（区块奖励），并收取区块中所含交易的费用。有时候，新的区块会创建在相同的前一个区块的上方，从而出现分叉。根据共识机制，矿工应在最长的链上继续挖矿，因此将丢弃较短的分叉。PoW 方便验证，但很难产生 [2]。重复计算哈希（通过改变 nonce）本身并没有用途，因此 PoW 只是纯粹的抽签，不能保证为所有花费计算资源的参与者提供公平的报酬。我们希望可以奖励给有实际效用生产能力的矿工。为此，PoW 应该替换或者与有效工作 (例如*机器学习*) 结合使用，矿工应竞争提供*有效工作量*（PoUW）。

使用机器学习（ML）作为有效工作量的基础来设计 PoUW 系统时需要解决很多障碍。与哈希计算相比，机器学习任务更复杂而多样化；比特币难题 [2] 的复杂程度会随着时间的推移而增长，但是在现实世界中，客户的机器学习问题决定了任务的复杂性；由于机器学习任务的多样性，很难验证参与者是否诚实地执行了工作；在机器学习算法中的许多步骤中恶意行为者可以逃避工作、草率完成或实施恶意行为；区块链在设计上不能保存机器学习训练所需的大量数据，而且大多数参与者对大多数数据也不感兴趣；在区块链点对点（P2P）网络的不受信任的环境中分发和协调机器学习训练过程也很困难。

大量的商用机器学习平台分布式分配并执行操作（例如 Facebook [3]，Amazon [4]，Google [5]）。他们需要基于大数据在多个节点之间分配工作负载

和业务流程。这些系统利用了数据并行：每个节点内部都拥有训练过的机器学习模型的副本，而数据则通过工作节点 [6] 进行分配。区块链可以提供比标准机器学习云服务更多的计算资源。

我们的研究受到以下问题的启发：*我们可以使用区块链的安全性构建更好的人工智能系统吗？我们能否提供基于 PoUW 的更好的区块链协议？*

我们使用区块链的架构来协调深度神经网络（DNN）的训练。我们设计了一个具有共识协议的分布式网络，以执行和验证基于机器学习的有效工作。该协议的核心是一种新的通过有效工作来构建 nonce 值的方法。

我们的论文主要结构如下：在第 2节，我们简要介绍了将人工智能与区块链相结合的工作。在第 3节，我们描述了系统环境和协议。我们将在第 4节论述机器学习训练的详细步骤。在有效工作量证明 PoUW 部分（第 5节），我们描述了挖矿和验证的过程。我们还实现了一个概念证明级产品，其详细信息在第 6中有详细描述。在第 7节，我们描述了本提案的潜在性能和安全问题。最后，我们在第 8节给出了有关未来工作的论述。

2. 相关工作

目前，已经有一系列不同的 PoUW 协议被提出，但它们都缺乏实用性：Ball 等人在 [2] 中提出了在区块链上解决 *正交向量*问题, *全源最短路径*问题和 *3SUM* 问题的方法; 在质数币 [7] 中矿工们可以在挖矿的同时寻找质数的 Cunningham 链。

DML [8] 和 SingularityNET [9] 是利用智能合约构建的分布式 AI 服务市场。SingularityNET 需要管理服务。它们都没有与区块链紧密集成，而是将 AI 作为单独的服务。

Gridcoin [10] 是一个开源的基于权益证明（PoS）的区块链，用于解决 Berkeley 网络计算开放基础架构上的任务，其系统是集中式的（白名单，中央服务器，集中式验证）。

Coin.AI [11] 只是一个矿工可以单独训练 DNN 理论建议。深度神经网络的架构是根据最后一个挖掘的块生成的，其验证涉及检查模型的性能。他们的系统容易出现安全风险，例如矿工从事廉价工作或不工作。

CrowdBC [12] 是一个使用以太坊智能合约的众筹框架。参与者存入一定金额的货币以降低出现潜在不良行为的可能性，但是矿工并不能执行有用的工作。

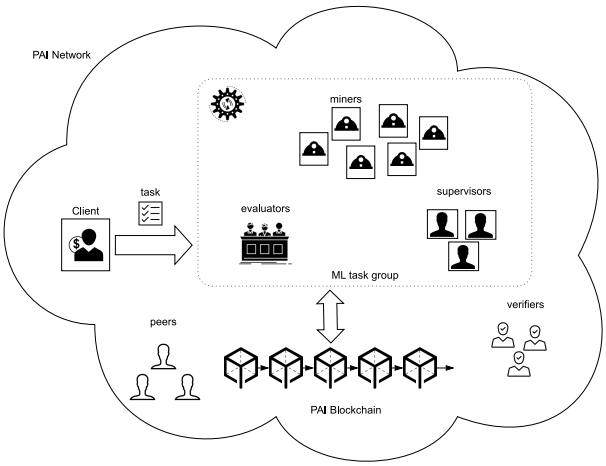

图 1: **系统概览.** 客户提交一个机器学习任务到 PAI 网络。工作节点执行训练，评估者决定支付方式。PAI 链保证机器学习过程的安全性。

据我们了解，我们的 PoUW 提案是唯一与区块链紧密集成的协议。它不需要智能合约，而且矿工可以执行有效的计算工作，并确定性地对其进行验证。我们的 PoUW 协议可以借助分布式网络、更高的安全性和适当的激励措施更好地促进人工智能发展。

3. 系统概览

3.1. 系统环境

本系统的运行环境是 PAI（Personalised Artificial Intelligence）区块链，一条使用混合工作量证明/权益证明的区块链。这是一个由以下几种角色组成的去中心化网络（1）：

客户： 支付一定费用以在 PAI 链上训练其机器学习模型的节点。

矿工： 执行训练的节点。他们可以在每个 iteration 后得到特殊的 nonce 值用来尝试挖出新区块。训练是分布式进行的，所有矿工通过共享他们本地模型的更新来进行合作。

监督者： 将一个机器学习任务期间所有消息记录到“消息历史”日志中的节点。因为环境中可能存在拜占庭节点，所以监督者也可以在训练期间防

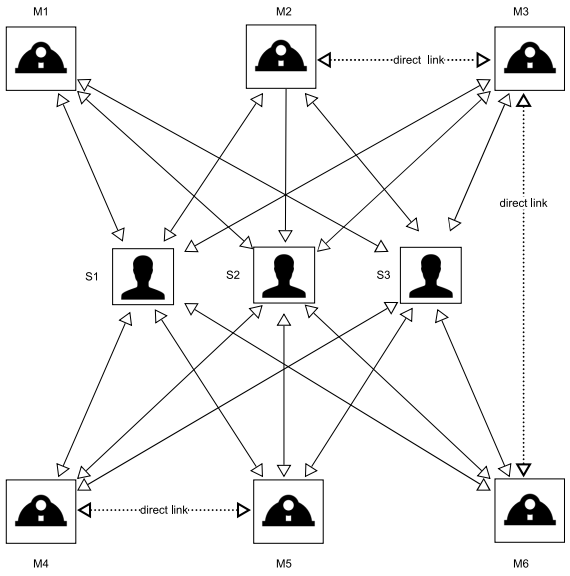

图 2: **一个通信拓扑结构的例子。**矿工们（M1-M6）与监督者们（S1-S3）全连接，并且与监督者间双向传递数据。某些矿工之间也会直接建立连接（例如 M2-M3, M4-M5, M3-M6）。

止某些恶意行为的发生。

评估者： 测试来自每个矿工的最终模型的表现、并将最好的模型发送给客户的独立节点。他们也负责将客户支付的费用合理分配给所有参与的节点。

验证者： 检验一个区块是否合格的节点。我们需要他们的原因是验证是一个计算量很大的过程，不能让所有节点都参与验证。

普通节点： 不属于上述任何一种的节点。他们进行常规的区块链交易。

矿工与监督者因为在训练任务中参与度较高又被称为**工作节点**。工作节点之间用高速信道通信。我们推荐在完整 VPN（虚拟私人网络）网格拓扑结构（例如 PeerVPN – `https://peervpn.net`）中运行所有工作节点以支持直接通信。矿工向监督者发送消息，后者记录消息历史。矿工之间直接通信是非必需的（请参考图 2）。

3.2. 交易

一笔比特币交易是一种经过数字签名的，包含版本信息、输入和输出的数据结构。如果一笔交易的输入可以追溯到另外一个或多个未经使用的输出（UTXO），那么这笔交易就是有效的。**OP_RETURN** 是一个脚本操作域（在比特币中有 80 字节长），它用来标记一个无效的交易输出，可以被用来在区块链上储存任何数据。

在 PAI 链中，我们用特殊的交易来处理机器学习任务中的训练，验证，评估和支付。为了达到这个目的，我们将所需的信息编码到一笔交易输出的 OP_RETURN 码中（在我们的协议中有 160 字节长）[3]。

在被归入区块前，交易会在 *mempool* 这个待定交易缓冲池中等待。节点们互相转发 mempool 中的交易直至广播到真个 P2P 网络。

链下信息和自己的交易用参与者的 ECDSA 私钥签名。在多方交易（例如监督者需要在一个决定上达成共识）中我们使用 *Boneh-Lynn-Shacham*（BLS）密钥与签名（ [13]）方案。为了降低特殊交易的数量，我们要求多方交易需由一个**组长**发出。

3.3. 质押

除了普通节点之外的其他所有节点都需要首先存入一些虚拟货币。我们将这个过程称为“**质押**”。存款代表着一笔被锁定的资金，如果参与者按照要求完成了工作他们就可以收回这笔资金，以及额外的奖励。

质押就是购买券。节点可以发起 *BUY_TICKETS* 交易，它是一笔包含身份（矿工，监督者等）信息和任务偏好信息（编码在 OP_RETURN 中）的特殊交易。券在被记录在区块链中并且经过一定量（例如 128 个）的新区块确认之后变为“激活”状态。通过每挖出 144 个新区块后调整券的价格，*mempool* 中券的数量稳定在约 40960 张左右。我们采用 Decred 的算法（ [14]）来调整券价。一个新挖出的区块中不能包含超过 50 张券。

另外一种质押交易是 *PAY_FOR_TASK* 交易，它由客户在提交一个新任务时发起。它包含任务描述信息和一笔训练手续费。

质押是一种用来避免恶意行为的方式，因为不诚实参与者的质押金会被没收且分给诚实的参与者。在训练的最后，评估者会决定如何分配客户支付的

[3]我们在 Appendix A的补充材料中提供了特殊交易的实现案例，还在 Appendix B中描述了缩短任务等待时间的方法。

手续费以及如何惩罚恶意参与者。评估者发起一笔 *CHARGE_FOR_TASK* 交易来将手续费支付给诚实参与者。如果任务不能被执行手续费将退还给客户。如果任务被恶意参与者破坏，诚实工作节点的质押金同样会被退还。每笔质押金都有一个有效期限，如果在该期限内质押者没有被分配任何任务，质押金将被全额退还。被选中参与任务的节点如果离线就会损失一部分质押金（例如 10%）。

3.4. 任务

一个客户向区块链以一笔特殊交易的形式（*PAY_FOR_TASK*）提交一份任务说明 T 和一笔手续费 F，具体包括：

- 一份待训练模型的描述：模型类型（例如：多层感知机 MLP、卷积神经网络 CNN）和它的结构：网络层类别（例如：Dense 核心网络层、dropout 层、卷积层、批标准化层、池化层），神经元数量，激活函数（例如：*relu*、*sigmoid*、*tanh*），损失函数（例如：softmax 交叉熵）。
- 优化器（例如：SGD、Adam）及其参数（例如：初始化参数、学习率）。
- 训练停止条件（提前终止、预设 epoch 数量）
- 验证方式（例如：交叉验证、holdout 验证）以及训练集中作为验证集的比例–$D_{pct}^{(val)}$。
- 模型评价指标 K（例如：准确率、损失）；客户会为在测试集上按照此评价指标表现最好的模型支付手续费。
- 数据集信息：格式（例如：CSV、MNIST [15]），mini-batch 大小，完整数据集中训练集所占比例（$D_{pct}^{(tr')}$），将原有数据集 D 分为至少十等份后的各自哈希值，还有完整数据集 D 的大小。
- 表现：期待训练时间（t_{exp}），硬件偏好（例如：CPU 或 GPU）。

手续费 F 按照一套奖励机制 R 分配给工作节点和评估者：一部分根据模型表现分给矿工，一部分分给监督者，一部分分给评估者。所有节点都受到激励参与到 PAI 链之中：客户得到训练好的模型，矿工可以得到区块奖励和客户手续费的一部分，监督者和评估者同样会得到客户手续费的一部分。矿工支付区块奖励的一部分给验证者。

3.5. 协议

在我们的系统中，一个客户利用 PAI 链训练一个机器学习模型并为此项服务付费。在客户将任务广播到 PAI 网络中之后，矿工和监督者将被网络按照工作节点的偏好 P 和任务说明 T 随机选择。

首先数据集 D 被分成：

- 一个训练集，后来会发给矿工以执行机器学习工作。
- 一个验证集，从最初的训练集中选出来以进行机器学习验证。
- 一个测试集，后来会发送给评估者以测试最终模型表现。

训练集会被进一步分成大小相同的 mini-batches。mini-batch 的典型大小由 10 条到 100 条记录不等。每一轮 iteration 处理一个 mini-batch。一个 epoch 是一轮完整的训练周期，即训练集的所有 mini-batch 都被处理一次。

在训练开始后，矿工们基于自己在 mini-batch 上的工作以及从别的矿工处接收到的消息不断提高自己本地模型的性能。每名矿工都与其他参与者分享自己对本地模型的修改。他们还可以在每轮 iteration 后用几个 nonce 值进行挖矿尝试。

监督者们在一项任务期间记录所有消息并监测恶意行为。评估者测试最终模型、选择最好的模型发送给客户并分配客户的手续费。

矿工们还是按照比特币协议的模式构造区块，但同时完成了有效工作量，即诚实地执行了机器学习训练任务的一轮 iteration。一名矿工扫描 mempool，收集交易，创建新区块并添加有效工作量证明的附加信息（额外的域）。在我们的有效工作量系统中，nonce 值是 32 字节长的 SHA-256 哈希。

区块有效性验证工作由验证者进行，他们收到输入数据，重新执行幸运的 iteration，并检查输出数据是否可重现。一名矿工会向验证者们发送有效工作量证明协议验证的所有数据。

4. 工作流

机器学习任务工作流有四个阶段：注册，初始化，训练，终止

4.1. 注册

在注册阶段，客户提交一个 PAY_FOR_TASK 交易。这笔交易被记录在一个叫做任务定义块的区块中。客户可以在任务开始前通过提交一个指向之前的交易的 REVOKE_TASK 交易来撤销一个任务，对应的 PAY_FOR_TASK 交易就会被取消。如果客户想调整对一个任务的报价，他可以在任务开始或者过期前再次提交一次任务。任务的过期时间也会被重新计算。

要开始一个任务至少有 2 名矿工和 3 名监督者。监督者的数量可以在满足 $|S| \in [3, \max(3, sqrt|M|)]$ 的范围内，矿工的数量在 $[2, \omega_D * disk\_size(D) + \omega_t * t_{exp} + \omega_F * F]$ 的范围内，其中 $disk\_size$ 是一个返回数据集占磁盘空间的函数（以 kB 为单位），$|S|$ 代表监督者的数量，$|M|$ 代表矿工的数量，ω_D ω_t 和 ω_F 是网络系数。一个任务定义块如果不满足这些要求会被视作无效区块。

一个工作节点需要自己计算检查他的激活券中是否有被选中参与训练的。一张在任务提交后才被购买的券无法参与到此任务的训练之中。对任意一张券来说，检查流程如下：

1. 计算本券的偏好 P 和对应的任务偏好最大子集 $P_{max} \subseteq T$（P 和 P_{max} 以非负向量的形式表示）之间的余弦相似度：$s(P, P_{max}) = \frac{\sum_{i=1}^{n} P_i P_{max\,i}}{\sqrt{\sum_{i=1}^{n} P_i^2} \sqrt{\sum_{i=1}^{n} {P_{max}}_i^2}}$; $s(P, P_{max}) \in [0, 1]$
2. 以一个可验证随机函数（VRF）[16]、任务定义块的哈希（$hash_{TDB}$）和券所属的身份（监督者、矿工等）作为输入，计算出一个哈希和一个证明：$(hash, proof) = VRF_{sk}(hash_{TDB}||role)$（参考 [17]）。请注意：哈希看起来是随机的，但实际上依赖于一个密钥 sk 和输入字符串。已知 sk 对应的公钥 pk 和证明就可以验证一个哈希是否是正确生成的。
3. 如果 $s(P, P_{max}) \geq \omega_S \frac{hash}{2^{hashlen}-1}$，该券就会被选中参与任务 T。ω_S 是一个全网参数，$hashlen$ 是哈希的比特长度。

被选中的工作节点会发送一个名为应用（*JOIN_TASK* 的交易），它会被包含在下一个区块中，该区块被称为参与块。我们将任务定义块的哈希作为随机数生成器的种子以供将来重现或验证选择过程之用。

4.2. 初始化

4.2.1. 密钥交换与生成

工作节点相互交换他们的公钥来验证收到的消息和特殊交易。

监督者通过 *t-n* 门限 *BLS* 签名来达成共识，其中 $n = |S|$，t 代表 2/3 的门限（$t = \frac{2}{3}n$）。n 个监督者中的 t 人子集即可给一笔交易签名使其生效，而签名人数小于 t 时会使交易无效。监督者们执行 Joint-Feldman 分布密钥生成（DKG）协议（ [18]）的一个改版来合作生成他们的 BLS 私钥（参考补充材料的 Appendix C和 Appendix D）。因为 BLS 门限签名方案的性质，对任何一笔多方交易 tx，一旦收集到 t 个签名，一个组长就可以在交易上重建全局签名并验证交易，达到和用全局私钥（未知）签名一样的效果。在 DKG 协议的最后，所有监督者需向区块链发送用他们自己 ECDSA 密钥签名的、记录了本地 t-n 公钥的 *DKG_SUCCESSFUL* 交易。如果所有监督者在一个预定时间窗口内发送了包含相同公钥的交易，则协议已成功执行，大家可以进入到下一个阶段。否则出错的监督者将被替代，他们的质押金也会被没收，DKG 协议需重新执行。

4.2.2. 数据准备

在提交一个任务之前，客户需私下将数据集 D 分成 $p \geq 10$ 的连续几个部分 $(d_1, d_2..d_p)$，使得 $|d_1| = |d_2| = .. = |d_{p-1}|$;，且 $|d_p| = |D| - (p-1)|d_1| < |d_1|$，随后计算每个部分的哈希 $H(d_1), H(d_2)..H(d_p)$，将哈希附到任务定义中的数据集部分。哈希是公开的，但是数据集 D 只有客户知道。

在初始化阶段，客户和工作节点将任务定义块的哈希作为随机数种子来随机排列 D 的哈希。前 $D_{pct}^{(tr')}$ % 的哈希对应初始训练集，客户随后会向工作节点公开，其他的对应测试集，直到机器学习任务完成后才能公开。

验证集由矿工根据验证方式从初始训练集独立确定地推导而来。例如采用 *holdout* 验证时，验证集

$D^{(val)}$ 包含初始训练集 $D^{(tr')}$ 中 $D^{(val)}_{pct}$ % 的 mini-batch。最终训练集是初始训练集减去验证集后的剩余数据集：$D^{(tr)} = D^{(tr')} \setminus D^{(val)}$。

最终训练集被进一步分成 m 个 mini-batch：$b_1 ... b_m$；他们满足 $|b_1| = |b_2| = .. = |b_{m-1}|$ 以及 $|b_m| = |D^{(tr)}| - (m-1)|b_1| \leq |b_1|$（大小由 client 在任务偏好中决定）。我们用下列两个方法中的一个给矿工们分配 mini-batch：

有限负载一致性哈希 这是一个受到 [19] 和 [20] 启发的方法，它规定一个矿工先计算 mini-batch 附上 epoch 编号 ξ 的哈希：$H(b_1||\xi)..H(b_n||\xi)$，然后用模函数 $H \bmod 2^\kappa$（κ 是映射空间的指数，是一个全网统一的值，例如 $\kappa = 10$）将每个哈希映射到一个大小为 2^κ 的一致性哈希环上。每个矿工应该处理的 mini-batch 对应的哈希值取值范围在哈希环上该矿工自己的哈希值与下一个矿工的哈希值之间。为了避免被分配工作量存在差异，我们不允许一个矿工处理超过 $cm/|M|$ 个 mini-batch，其中 $1 \leq c \leq 2$ 是一个公开常量。为了达到这个目的当前 mini-batch 会被分配给下一个矿工直至满足此条件。

交错数据 这个方法规定矿工和 mini-batch 分别根据 ID（$M = \{M_1..M_n\}$）和哈希值的字典序排序。随后第一个矿工分到第一个 mini-batch，第二个矿工分到第二个，以此类推，直到所有 $|M|$ 个矿工一起分到了前 $|M|$ 个 mini-batch。接着，第一个矿工再分到第 $|M|+1$ 个 mini-batch，如此往复，直到所有 m 个 mini-batch 都被分配。对每个 epoch ξ，一个矿工 M_i 首先分到 mini-batch $b_{i+\xi-1}$，然后周期性地分到 $b_{i+\xi-1+|M|}, b_{i+\xi-1+2|M|} ...$，同时也包括之前没有被覆盖到的位置 $[1..i+\xi]$（mini-batch 被视作一个环状数据结构）。

如果一个矿工退出或者有心的矿工加入任务，上述分配过程需要重新执行。我们使用以任务定义块作为种子的随机数生成器来保证矿工们可以提前直到自己将被分配到的 mini-batch 以及顺序。这些步骤在区块验证时也可以被重现。

4.2.3. 数据储存

监督者在机器学习训练期间储存数据，他们可以下列冗余方案中的一种：

完全复制 每个监督者都保存一个 $D^{(tr')}$ 的副本。这将会导致一个有监督者数量 $\beta = \frac{|data_{red}|}{|data|} = |S|$ 这么高的冗余系数（[21, p. 36]）。一个矿工可以选择性下载 mini-batch。

Reed-Solomon 监督者们用$(k+h)$ *Reed-Solomon 储存方案*（[22]）来储存 $D^{(tr')}$。他们将训练集分成 $k+h$ 个大小相同的部分，监督者的总数约为 $|S| \approx k+h$ and $k \approx [1/3..2/3] * |S|$。为了得到完整的数据集，一个矿工需要从 k 个监督者处下载并重建数据。本方案冗余度较低：$\beta = \frac{k+h}{k} = [1.5..3]$，但是矿工需要为重建数据集付出额外计算开销。

4.3. 训练

矿工们初始化他们本地模型的权重与偏差 $\boldsymbol{\theta}$ 以开始一项机器学习任务。在每一轮 iteration 之中，他们下载一个 mini-batch，在其上执行*随机梯度下降 SGD*（[23, p. 147]），然后与其他工作节点通信，发送他们对 $\boldsymbol{\theta}$ 的更新（权重更新）。在大多数机器学习算法中代价函数是一系列训练数据的损失函数 L 输出之和，全局梯度是函数的期望。为了最小化全局损失，可以从数据中取样一个 mini-batch b_i 来计算近似全局梯度的偏梯度 $\mathbf{g^{(i)}}$，而不是直接使用全部数据：$\mathbf{g^{(i)}} = \frac{1}{|b_i|} \nabla \sum_{j=1}^{|b_i|} L(x^{(j)}, y^{(j)}, \theta)$, 其中 x 和 y 代表每条数据的特征与目标。得到梯度后经典 SGD 算法以如下方式更新模型：$\theta \leftarrow \theta - \epsilon \mathbf{g^{(i)}}$，其中 ϵ 是学习率。

因为梯度的大小和本地模型的大小一样，让矿工把所有梯度更新都发送给网络是不现实的。为了克服这个困难，我们使用 [4] 的“dead-reckoning”方案，即在本地模型中仅更新且分享超过一个特定 τ 值的权重。低于 τ 的梯度会被累加起来在以后应用。这个延时更新的方案不会影响到学习。

一轮 iteration 由改进自 [4] 的**算法 1**中的几个步骤组成，符号定义请参考表 2。

- 矿工按 4.2.2节所述方法下载一个 mini-batch(b_i)。
- 根据从及其他节点处接收到的权重更新信息更新本地模型。
- 用向后传播计算本地梯度，并把它累加到剩余梯度之中。
- 定义一个消息映射：一个 $\boldsymbol{\delta^{(\ell)}}$ 列表。这个列表

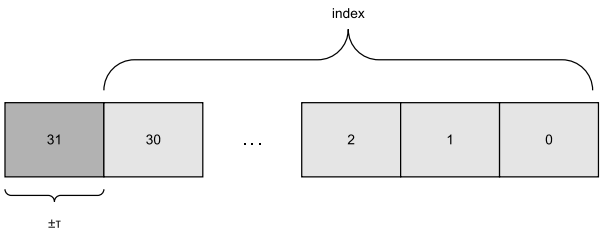

图 3: **消息映射**最高位比特与 τ 相关，其余用来表示索引。

记录了剩余梯度中大小超过门限值 $\pm\tau$ 的梯度的地址（索引），τ 由客户指定。列表的一个元素 $e(0|1, i)$ 有 4 字节长，当索引为 i 的剩余梯度的值小于 $-\tau$ 时，最高位比特为 0；而当他大于 $+\tau$ 时，最高位比特为 1。其他的 31 个比特用来记录索引 i（取值从 0 到 $2^{31}-1$ 的无符号整数）（请参考图 3）。

- 将消息映射中的权重更新应用到本地的深度学习网络模型副本之上。
- 从其他节点处接收并解压消息。
- 向网络发送包含如下信息一条消息：当前实现版本、机器学习任务 ID，消息类型（*IT_RES*）、当前 epoch–ξ、其他节点更新的数量、权重更新的消息映射–$\delta^{(\ell)}$、训练指标–$K^{(tr)}$、开始与结束时间–$t^{(s)}$、$t^{(f)}$、mini-batch 的哈希、在当前 iteration 收到的其他节点消息组成的列表的哈希、模型初始状态 θ 的哈希、初始剩余梯度的哈希、nonce 值设为 0 的在 k 个 iteration 后待挖区块的哈希–$hash(ZNB)$、用于下一轮 iteration 的未压缩其他节点消息（在第 20 部收到）组成的列表的哈希、以及矿工的 DER 编码 ECDSA 签名（请参考表 1）。
- 生成多个 nonce 值并尝试挖矿。

如果一个矿工较早完成了一个 epoch 的工作，他需要等待其他节点、临时应用来自他们的更新，直至大多数节点同样都完成本 epoch 工作。

在每一个 epoch ξ 中，用 $(i-1) \mod \xi + 1$ 公式轮流选出一个组长，i 是将所有监督者按字典序排序后的监督者的 ID 的排名。如果下一个被选出的组长（索引为 $(i-1) \mod \xi + 2$）提交一个有效的多

表 1: IT_RES 消息组成

域	字节	类型	注
$version$	2	short	消息版本
$task\_id$	16	uuid	任务 id
msg_type	1	char	IT_RES
ξ	2	ushort	epoch
$\|\delta^{(\ell)}\|$	2	ushort	其他节点更新数量
$\delta^{(\ell)}$	$4 * \|\delta^{(\ell)}\|$	list	请参考图 3
$\|K^{(tr)}\|$	2	ushort	模型指标数量
$K^{(tr)}$	4 * $\|K^{(tr)}\|$	list	模型指标
$t^{(s)}$	4	uint	UNIX 时间戳
$t^{(f)}$	4	uint	UNIX 时间戳
$hash(b_i)$	32	uint256	SHA256
$hash(\Delta^{(p)})$	32	uint256	SHA256
$hash(\theta)$	32	uint256	SHA256
$hash(g^{(r)})$	32	uint256	SHA256
$hash(ZNB)$	32	uint256	SHA256
$hash(\delta^{(p)})$	32	uint256	SHA256
sgn	65	string	DER 编码 ECDSA

方 *REPLACE_LEADER* 交易且被至少 2/3 的监督者同意，则当前组长被替换成该组长。交易记录了旧组长被替换的原因（离线、反应时间过长、恶意行为等），这个过程可以持续直至找到一个合理的组长。

监督者记录下来在训练期间传递的**消息历史**以此帮助矿工证明他们诚实地执行了机器学习任务。验证者只关心消息历史中可以证明有效工作量的部分。因此，在一个 epoch 期间组长会将一组连续的消息封装到**消息槽**。监督者们用偏好投票（例如 Schulze 方法 [24]）来共同确定一个消息槽内消息的顺序。投票完成后，组长计算每个消息槽的结果并将它的哈希以 *MESSAGE_HISTORY* 交易的形式公开。消息槽的原始数据由监督者保管，可以被矿工或

表 2: 算法 1符号定义

符号	定义
$\mathbf{X^{(tr)}}$	mini-batch 中的特征
$y^{(tr)}$	mini-batch 中的目标
θ	iteration 开始时 模型的状态（权重、偏差）
θ'	应用其他节点更新后 模型的状态（权重、偏差）
θ''	应用本地更新后 模型的状态（权重、偏差）
ϵ	学习率
$\boldsymbol{\Delta^{(p)}}$	从其他节点处获得的权重更新
$\boldsymbol{\Delta^{(\ell)}}$	本地权重更新
$\boldsymbol{g^{(\ell)}}$	本地梯度
$\mathbf{g^{(r)}}$, $\mathbf{g^{(r)'}}$, $\mathbf{g^{(r)''}}$	不同阶段的剩余梯度
$\boldsymbol{\delta^{(\ell)}}$	消息映射
$\boldsymbol{\delta^{(p)}}$	从其他节点处获取的消息映射
$K^{(tr)}$	mini-batch 上的模型指标
$t^{(s)}$	iteration 开始时间
$t^{(f)}$	iteration 结束时间

验证者们下载。

工作节点如果跳过一定量的 iteration（矿工 5%，监督者 10%）就会损失他们的质押金。一个工作节点若想重新加入一个任务就需要同步至最新的权重更新。如果矿工数量降到初始大小的 80% 则监督者需要添加一个心的矿工。一个监督者若漏记超过 10%iteration 的消息也会被取代。组长通过公开一个 *RECRUIT_WORKER_NODE* 交易来启动替代过程。我们在补充材料（Appendix E）中详细描述了离线节点检验与替代过程。

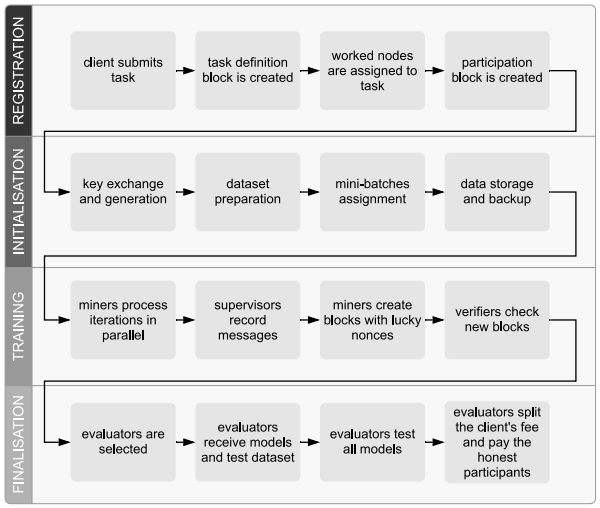

图 4: **机器学习任务步骤**一个典型的在 PAI 链上进行机器学习训练的流程。

4.4. 终止

客户向验证者提供测试集，后者会验证客户是否有撒谎（即检查哈希是否与保留的测试 mini-batch 匹配）。他们在测试集上测试来自每一个矿工的模型并将表现最好的一个模型发送给客户。评估者按照与选择工作节点类似的算法被选择出来。

评估者必需生成包含机器学习指标的相同的结论。一个结论以 *CHARGE_FOR_TASK* 交易的形式公开。如果少于 2/3 的评估者生成相同的结论，重新选择下一组评估者直至达成 2/3 共识。一个评估者可以通过等待 mempool 中出现结论再公开相同结论的方式来作弊。为了避免这个问题我们将本流程分成两个阶段：1）所有评估者将自己的结论加密后再发送到 mempool；2）他们看到所有结论都收集齐了之后公开自己的揭秘密钥。我们将这个方法称为 *先承诺后揭秘*。

我们在图 4中对上述所有步骤进行了总结。

5. 有效工作量证明

5.1. 挖矿

一个矿工在每轮 iteration 之后有挖矿的权力。在经典比特币机制中，矿工通过变化 nonce 值来获取不同的区块头哈希。我们将 nonce 的数量限制在 $a = \omega_B * disk\_size(b_i) + \omega_M * model\_size(\boldsymbol{\theta''})$ 以

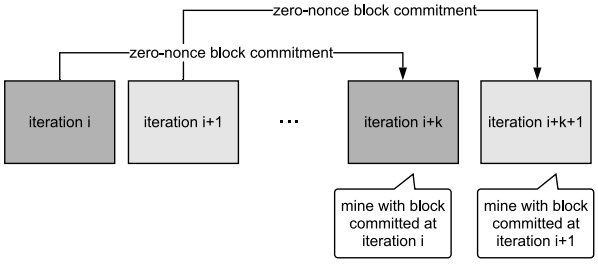

图 5: **零 nonce 区块承诺**一个矿工提前 k 个 iterations 宣布他将要进行挖矿活动的零 nonce 区块。

下，其中 $model\_size$ 是一个返回模型权重与偏差数量的函数，ω_B 和 ω_M 全网范围的系数。我们旨在保证进行哈希运算的计算量远低于用于机器学习训练的计算量。

一个矿工可以通过修改区块中的交易（例如更改时间戳）而快速生成数以百万计的哈希。因此我们要求矿工如图 5所示在 k 个 iterations 之前需要“承诺”一个零 *nonce* 区块（ZNB）。一个 ZNB 由一个固定的区块头和一组固定的交易构成，将 nonce 和其他 PoUW 所需信息设置为 0。一个矿工需要在 iteration i 时将 ZNB 的哈希记录在 IT_RES 消息中。在 iteration $i+k$ 时，矿工可以将 ZNB 中的 nonce 替换成根据算法 2从机器学习训练中间值计算生成的 nonce。nonce 是将 iteration 结束时本地模型 θ'' 和本地梯度连在一起、再进行双重哈希的结果。我们把机器学习中间值第一次哈希的结果称为前驱 *nonce*。

在成功挖矿后，矿工将模型初始状态（权重与偏差），mini-batch，初始剩余梯度和其他节点更新保存下来，同时也要从监督者处下载并保存相关消息历史槽。接着矿工创建一个新区块，在其中记录 $hash(h_i)$ 和 $hash(msg)$ 以供将来查询,其中 $hash(h_i)$ 指向包含了从 iteration i 到 iteration $i+k$ 之间所有 IT_RES 消息的消息历史槽的 Merkel 树,$hash(msg)$ 是对应幸运 iteration 的 IT_RES 消息的哈希。经过成功的验证后，区块被添加到 PAI 链上。图 6展现了比特币区块头和 PoUW 区块头的对比。

结算即交易通过挖矿从 mempool 被添加到区块链上。在没有客户提交任务的时期，矿工会进行一些全网范围预定义的默认任务，例如利用深度学习网络

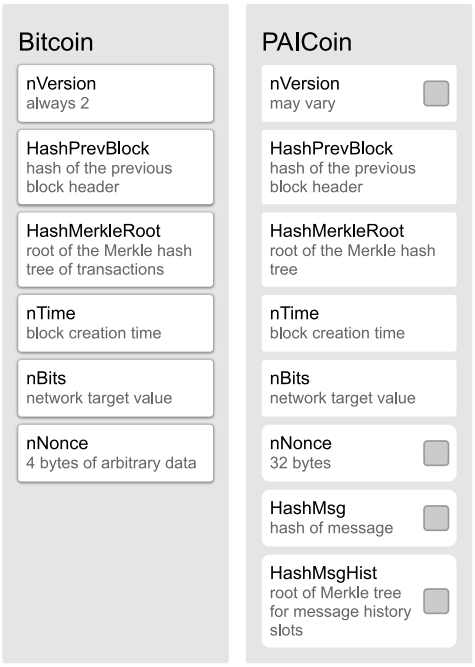

图 6: **PAI 币与比特币区块头对比**不同的域用方形图案标注。我们使用了和 [25] 相同的符号和区块结构。

进行蛋白质结构预测，以维持挖矿进行。这种情况下矿工不会收到客户手续费奖励，而只会收到区块奖励和交易手续费。特殊情况下监督者、验证者和评估者也获得区块奖励。在所有区块奖励不为零的区块被挖出来后，他们将仅从手续费获取奖励。为了避免矿工数量不足，会有一定数量属于 Project PAI 的矿机始终参与到网络中。

5.2. 区块验证

因为重新运行一个 iteration 的计算量十分大，所以验证一个新挖出区块的工作仅由验证者们完成。区块验证有如下几个步骤:

- 十名验证者根据新挖出区块的哈希（作为随机数种子）被自动选出。
- 验证者首先检查区块的哈希是否低于网络目标 T（和比特币协议一样）。
- 他们从区块中提取出: $nonce$ $hash(h_i)$ 和 $hash(msg)$，并接收到 iteration 开始时的 mini-batch 和本地模型、剩余梯度、其他节点权重更新以及消息历史的相关部分。

- 从消息历史提取出信息后，他们检查矿工身份、券和机器学习任务的有效性。
- 每个验证者通过比较哈希来决定消息历史槽是否在前一个区块中以特殊交易形式被注册，前一轮 iteration 中是否声明了本轮 iteration 开始前模型的哈希、剩余梯度以及其他节点更新。
- 他们检查 batch b_i 是否存在、是否按照之前收集的信息所规定的时间与顺序被处理。
- 在消息历史中，验证者可以通过与当前挖出区块的零 nonce 区块版本进行比较来检查 $i + k$ 承诺是否有效。
- 给定 mini-batch、iteration 开始前模型、其他节点更新以及剩余梯度，验证者重新运行算法 1的 2-18 步以检查得到的指标与发送给网络的指标是否一致。
- 验证者压缩并计算其他节点消息的哈希以检查其是否与在前面消息中记录的哈希一致。
- 在得到 iteration 结束时模型状态和本地权重映射后，一个验证者就可以重建前驱 nonce 以及 nonce 并与新挖出区块中的 nonce 进行比较。
- 验证者同样需要检查在一个时间窗口 Δt 内机器学习指标是否有提高。
- 每个验证者向 mempool 提交一个包含前驱 nonce 的加密摘要（以交易的形式）。
- 矿工等待摘要的出现并在某个时间提交一笔 *COLLECT_VERIFICATIONS* 交易。矿工要尽可能多的获取摘要确认，因为他能获得的下一笔区块奖励正比与他所收集的摘要数量（每增加一个摘要奖励增加 10%，直至十个摘要都收集到则可获得 100% 的区块奖励）。矿工同时也会期望其他矿工将来会在他的区块之上延伸区块链。
- 验证者会在另一笔交易中通过公布自己的加密密钥来公开他们的摘要。摘要也包含了机器学习任务的指标。

其他的普通节点和在比特币协议中一样检查区块哈希是否低于网络目标，还要通过验证者的摘要检查 nonce 是否由前驱 nonce 生成。当前节点收纳了包含上一个区块摘要信息的交易以用来验证 *coinbase 交易*（支付挖出区块的矿工的交易）中的区块奖励。

6. 实现

我们实现了一个 PoUW 的早期概念验证性产品。您可以在**pouw.projectpai.com** 看到我们的工作。本实现主要有三个软件库，所有工程都有详细的安装与使用教程:

PoUW 核心 的代码涵盖了分布式机器学习训练（算法 1）、验证（5.2节）以及挖矿（算法 2）。机器学习训练部分我们使用了 MXNet [26]，因为它适用于高性能机器学习并且支持动态图计算。我们的预先实验显示 MXNet 比其他机器学习框架更快。MXNet 作为 Apache Incubator 的一部分是一个开源项目，也被 Amazon 用在他们的机器学习云计算方案中。

PoUW 区块链 是我们系统实现使用的的底层区块链，它是一个 Project PAI 区块链的改版，不同之处包括区块头、特殊交易和其他一些区块链处理逻辑。PoUW 核心是 PoUW 区块链的充分必要条件。

PoUW 模拟 是一个基于 Kubernetes [27] 用来模拟 PoUW 环境下不同角色参与者的工程。它可以被用来在本地或云端机器上学习 PAI 网络是如何运转的。图 7是一个我们的机器学习系统如何在训练中提升指标的例子。为了收集这些指标，三名矿工在一个 NC6 Microsoft Azure 机器（Intel Xeon E5-2690v3 CPU, 56 GB 内存以及 1 x K80 GPU）上使用 MNIST 数据集 [15] 训练了一个全连接深度学习网络。收敛速度以及分布式网络在不同数目的训练节点以及不同数据集的条件下的表现不在本文的讨论范围内。有关这些话题请参考 [4]。

PoUW 现在是 Project PAI 计划的一部分。有关 Project PAI 的更多细节可以在**projectpai.com** 上面找到。

7. 实验讨论

我们的协议相比经典比特币有几大优势。我们使用了安全性更高的改进版混合工作量/权益证明共

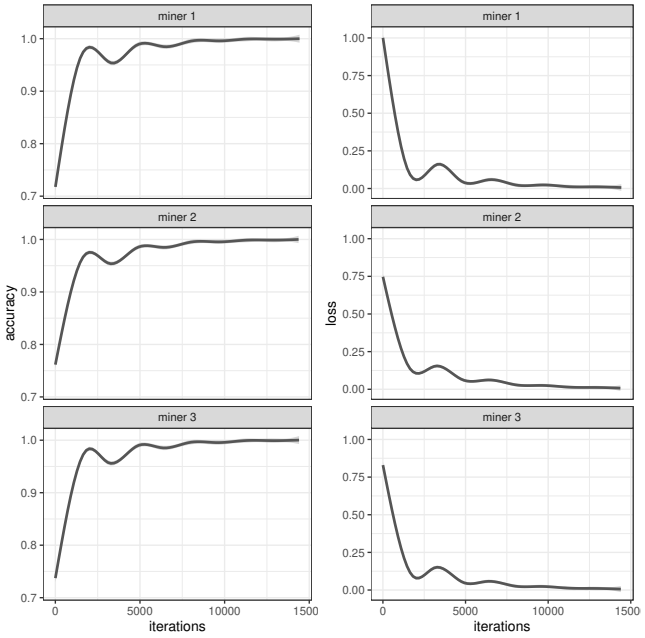

图 7: **收敛**在这个例子中，一个由三个矿工组成的分布式机器学习系统随着训练的进行逐渐提高准确率，降低损失。

识方案，并且给网络参与者提供了更多奖励。我们的方案也考虑了由于系统内在的复杂性而产生的性能风险与安全性风险。在接下来的几节中我们会提供一个经济学分析并且讨论性能优化方案以及对于系统安全性至关重要的恶意攻击场景。

7.1. 经济学分析

我们的 PoUW 方案较比特币挖矿收益更高，较机器学习云计算方案成本更低。为了证明这个观点，我们将使用云计算方案与使用我们方案的客户成本进行了比较，也对矿工按比特币协议与按 PoUW 协议挖矿的投资回报率（ROI）进行了比较。我们假定客户每小时支付的手续费不会超过云服务商支持机器学习的虚拟机的报价，矿工不会在投资与操作成本超过收益时参与系统，如果比特币挖矿收益更高矿工会转向后者。

我们发布了一个在线 PoUW 成本计算器来研究投资回报率与收益能力，可以通过在项目主页下“ROI 计算器”的链接访问。变量定义与公式在在线文档中有所解释。在这里我们简单介绍下这个计算器：

- 第一张表记录了全局参数，包括客户的每小时手续费（美元）、每个任务平均收费参与者人数、机器学习分布式系统效率、PAI 币与比特币当前价值、电价、PAI 网络活跃矿工人数以及 PAI 链区块奖励。
- 第二张表研究了几款最流行的比特币挖矿设备的收益。
- 第三张表比较了亚马逊、微软与谷歌的机器学习云计算与四种利用本地深度学习工作站（两种来自 [28]，另外两种来自 [29]）。
- 第四张表展示了客户在使用我们方案时的收益。

我们假定一个 PoUW 矿工在机器学习硬件上的投资摊销到三年的时间中。矿工购买并使用本地工作站的成本要比使用类似云计算服务的要低。图 8展示了两者在一年后与三年后的总成本比较。**投资回报率**（ROI）的计算公式是：

$$R = 100(\frac{g\left(\frac{F_h}{Q} + 6\frac{W}{U}\right)}{\frac{C_H}{26280} + E} - 1)$$

，其中 g 是 GPU 的数量，F_h 是客户每小时支付的手续费（成本计算器中的 *fee_usd_client_1h*），Q 是每个任务平均被支付参与者的人数（*paid_participants_per_task*），W 是矿工的区块奖励（*revenue_pai_block * price_usd_paicoin*），U 是任意时间平均矿工人数（*miners_active*），C_H 是硬件初始成本（*cost_usd_hwd*），E 是每小时电价（*power_kwh_1h * cost_usd_1h*）。为了评估投资回报率如何随其变量变化而变化，我们计算出它的偏导：

$$\frac{dR}{dF_h} = \frac{2628000g}{Q\left(C_H + 26280E\right)}$$
$$\frac{dR}{dQ} = -\frac{2628000g \cdot F_h}{Q^2\left(C_H + 26280E\right)}$$
$$\frac{dR}{dW} = \frac{15768000g}{U\left(C_H + 26280E\right)}$$
$$\frac{dR}{dU} = -\frac{15768000g \cdot W}{U^2\left(C_H + 26280E\right)}$$
$$\frac{dR}{dC_H} = -\frac{2628000g \cdot \left(F_h \cdot U + 6 \cdot Q \cdot W\right)}{QU\left(C_H + 26280E\right)^2}$$
$$\frac{dR}{dE} = -\frac{69063840000g\left(F_h \cdot U + 6 \cdot Q \cdot W\right)}{QU\left(C_H + 26280 \cdot E\right)^2}$$

为了量化投资回报率变量的重要性，我们需要考虑一些现实限制：电价是区间计价的，区块奖励会在

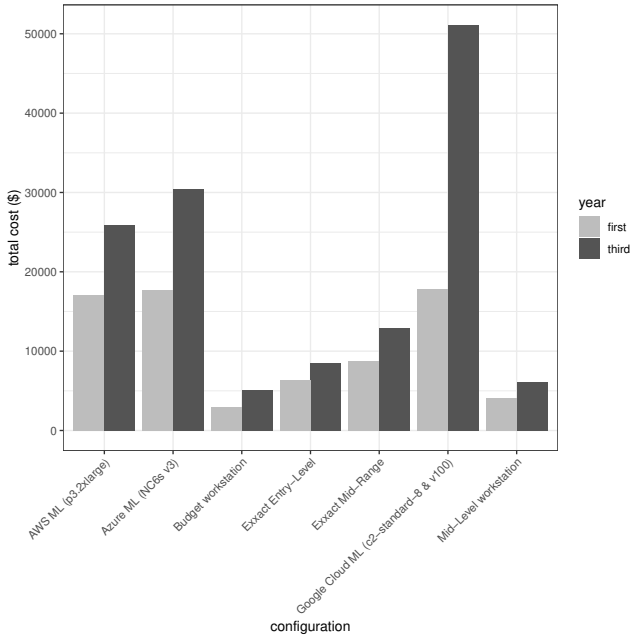

图 8: **一年与三年后总成本** Budget、Mid-Level 和 Exxact 是本地方案。AWS、Azure 和 Google 是云方案。

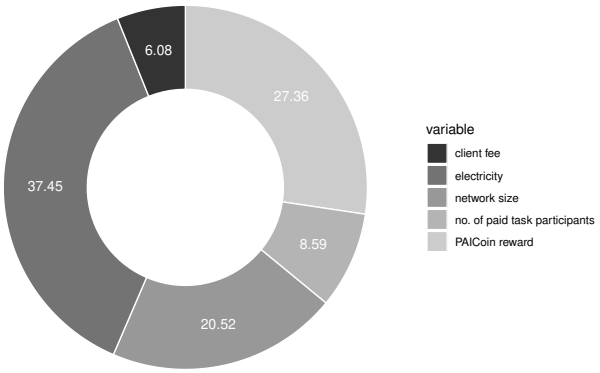

图 9: **变量重要性**本图绘制了最重要的几个投资回报率变量（其余的 <1%）。变量以绝对值评估。

一定时间后减半以及有静态硬件配置。当 $F_h = 0.50$ \$, $Q = 20$, $W = 1155$ \$（revenue_pai_block=1500 以及 PAI 币历史平均币价 price_usd_paicoin=0.77 \$），$U = 10000$, $C_H = 1945$ \$ 以及 0.12 \$ 每 kWh 时，我们可以得到如图 9所示的重要性评分。可以看出电价是对收益性影响最大的因素。图 10说明了随着每 kWh 价格的上升收益会急剧下降。在经典比特币中同样如此。PAI 币奖励（区块奖励乘以 PAI 币价格）是影响力第二大的因素，接着是网络大小。其他因素包括平均被支付参与者数量（矿工、监督者与评估者）以及客户手续费。

图 9没有考虑到初始硬件投资的影响。从图 11看出，在硬件上投资更少的矿工可能获得更高的 PoUW 收益。为了降低这种风险，我们的方案在特定任务硬件偏好与矿工系统能力（相见 4.1节）之间设计了一种配对机制。更贵的硬件会配有更多 GPU（$g > 1$），这样矿工就可以选择同时参与多个任务（一个 GPU 一个任务）。另外奖励机制 R 会给更好的硬件分配更高的奖励。

一名矿工在他的每小时收益 $P_h^{(pai)}$ 为正且大于他用市面上最好的挖矿设备进行比特币挖矿的收益时会参与到我们的网络中（$P_h^{(btc)}$）：$P_h^{(pai)} > P_h^{(btc)}$ 且 $P_h^{(pai)} > 0$）。每小时收益按如下公式计算：

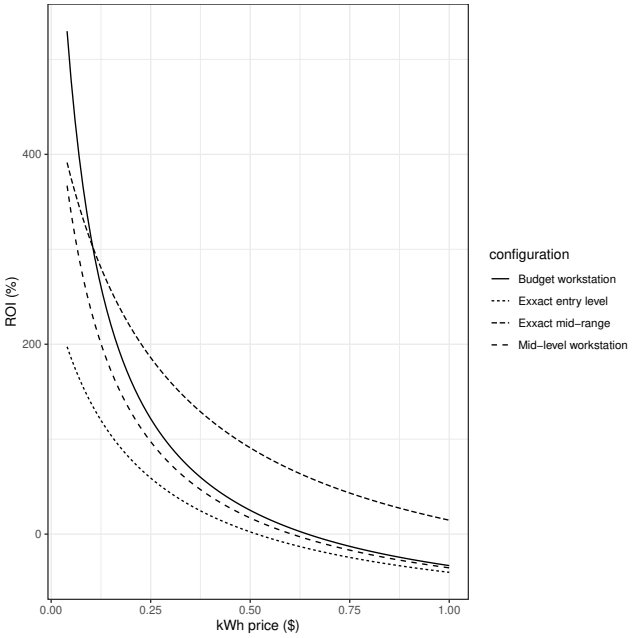

图 10: **投资回报率（%）与电价关系。**

$$P_h^{(pai)} = g\left(\frac{F_h}{Q} + 6\frac{W}{U}\right) - \left(\frac{C_H}{26280} + E\right)。$$

客户在支出比使用最便宜的云计算方案 $F_{h_{min}}^{(cloud)}$ 少时会使用 PoUW 系统（在我们的分析中，GCP 即为 $F_{h_{min}}^{(cloud)} = 2.36$）。变量 E_f 描述分布式系统效率，即 $F_h < F_{h_{min}}^{(cloud)} \cdot E_f$。一名客户使用我们方案时的支出节省为 $100\left(1 - \frac{F_h}{F_{h_{min}}^{(cloud)} \cdot E_f}\right)$。

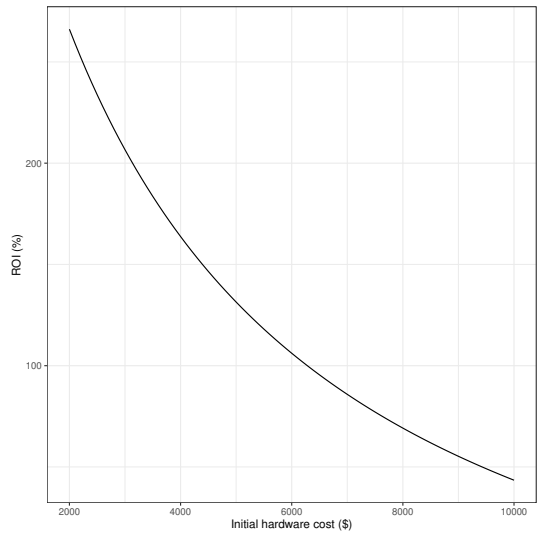

图 11: 投资回报率（%）与硬件成本关系。

在我们的计算中，当比特币币价为 $8243.41（2019 年 11 月）时，市场上最好的比特币挖矿设备（蚂蚁矿机 S17 Pro）可以实现 18% 的投资回报率，而 PoUW 挖矿设备在上述参数下可以达到平均 200% 的投资回报率。因此，PAI 币 PoUW 挖矿收益性大约为比特币的 10 倍。对于用户来说，手续费比使用最便宜的机器学习云计算方案还要少 30%。

7.2. 性能考虑

在分布式系统上进行机器学习的效率永远比不上在一个本地机器上运行完整数据集的效率，但是有很多地方可以优化以提高性能: 矿工们可以在整个 iteration 期间异步接收梯度，mini-batch 可以提前加载，创建消息映射可以并行进行。挖矿计算可以可以在另一条线程或进程中执行。工作节点可以舍弃不必需的数据。只有验证者需要执行完整的区块验证，其他普通节点不需执行高开销运算。我们在补充材料中描述了更多性能特征以及提高效率 E_f 的方法。

为了最优化带宽，我们使用了 Strom [4] 中的“dead-reckoning”方案。根据 [6]，此技术用在带有 1460 万个参数的模型数据传输时可以节省 99.6 % 带宽。

7.3. 安全性

我们假定绝大多数的参与者是诚实的。PoUW 协议被设计成可以避免来自拜占庭节点的各种威胁。我们在下文几段中列出了可能恶意行为与遏制手段:

客户提交提前训练好的模型

在这种情况下，客户有一个已经训练过的表现良好的模型。他讲机器学习任务提交 PAI 网络后再扮作矿工。如果这个恶意行为者被选中参与任务，他就可以不做机器学习工作而提交他的训练好的模型，以此获得 PAI 币挖矿的机会。

同态加密可以提供足够的保护但因计算量过大而不实用 [30]。为了降低这种风险，系统规定每名矿工必须提交每一轮的 iteration 权重更新以及幸运 nonce 的输入。因为恶意行为者的进度依赖于其他节点的更新，他难以逃避训练工作。

客户使用满足另一个数据分布的测试集

客户可能通过这种行为来免费获得训练好的模型，因为模型在这个测试集上的表现会很差。训练集与测试集理应满足相同的数据分布。

在我们的协议中，客户提交整个数据集，该数据集根据任务定义块的哈希（如 4.2.2节描述）被打散并切分为训练集和测试集。任务定义块是未知的，它在任务提交与随后记录在区块上之前是不存在的。

客户提交不符合规定的任务说明

我们允许工作节点们检查并报告不符合规定的任务 T。如有发现，客户的手续费会被没收并分配给工作节点与评估者们。

有害梯度攻击

有害梯度攻击意思是一个矿工试图以发送大量或者伪造梯度的方式来干扰学习过程。多次重复发送相同消息也被视为一种有害梯度攻击。Blanchard 等人在 [31] 中证明了仅一个拜占庭节点就可以显著地影响学习过程，并提出了 *Krum* 函数来检测这种攻击。Damaskinos 等人也在 [32] 中提出了 *Kardam* 过滤器来应对这种攻击。

监督者们会监控这种攻击并且把恶意工作节点添加到一个公开黑名单中。节点们会忽略来自黑名单中节点的梯度更新，验证者也会没收他们的质押金。矿工们不会应用对应同一轮 iteration 的多条 IT_RES 消息。为了排除没有取得学习进步的矿工，验证者会要求幸运矿工证明他的本地模型随着之前 iteration 的进行表现有所提升。

矿工仅执行挖矿计算

我们限制了 nonce 的使用数量因此使得经典挖矿计算量占比极小。一个矿工如果仅进行伪造的机器学习工作而专注于挖矿的话，他将不能证明产生的区块的有效性。伪造的机器学习工作包括：转发收到的权重更新、在任务完成前退出或不遵守机器学习训练的步骤。对于矿工来说这种行为会给他带来经济上的伤害，因为他不仅会损失质押金还无法得到任何客户手续费。

女巫攻击

恶意行为者可以在网络上设置多个虚假身份来合作产生一个劣质模型。他们也可以通过在所有节点间重复一个单位工作量来降低工作量。为了避免这种攻击，工作节点不允许自己挑选任务，他们只能陈述自己的偏好（详见 4.1节）。这样一来他们就不能挑选简单的任务。

拜占庭组长

如果监督者的组长延迟发布 *MESSAGE_HISTORY* 交易，则另一个组长会被立马选出。如果组长发布了无效的 MESSAGE_HISTORY 交易，他会被替代并被加入到黑名单中。

DOS 攻击

当工作节点没有收到足够的其他节点更新或当训练过程十分慢甚至停滞时，可能是受到了 DOS 攻击。他们可以暂停训练过程并在攻击结束后回复训练。

对抗 DOS 攻击的步骤是：几个工作节点发起一个带有原因 *DOS_ATTACK* 的 *CONSIDER_PAUSE* 交易。当多数节点同意在一个预先定义的时间范围内暂停时，诚实的工作节点们发起一个 *PAUSE* 交易，随后所有人暂停训练。在暂停模式下，所有参与节点会向之前机器学习任务小组中的节点在链下发送 *HEARTBEAT* 消息。当激活节点形成新的多数代表后，他们可以发送 *CONSIDER_RESUME* 消息，并最终发送 *RESUME* 交易来继续训练过程。

在 DOS 攻击期间如果一个验证者有足够的理由怀疑一个区块由攻击者挖出，他可以拒绝此区块。

区块链垃圾信息

一个恶意矿工可以用海量伪造区块堵塞区块链使得验证者付出大量的精力验证区块。这也是 DOS 攻击的一种因为很难快速地验证区块。我们采取了如下的反制措施：

- 赋予开销小的验证工作更高的优先级，提前执行
- 一名矿工如果提交无效区块他会被没收质押金并列入黑名单。
- 我们限制一个矿工在预定义时间范围内可以发布的区块的数量。

长范围攻击

在长范围攻击中，一个攻击者分叉出一条有大量区块甚至自创世块起所有区块的区块链分支。如果攻击者有足够大的计算力，他甚至能以超过主链的速度发布他自己的分支链。

我们系统底层的区块链是使用了受最长链规则保护的混合 PoS/PoW 区块链。新区块的生成需要为有效工作量付出大量的能量。我们同样规定每 1024 个区块创建一个检查点：检查点之前的历史都是真正不可修改的（即不能对区块链 1024 个区块前的内容作出任何修改）。12 名验证者会被随机选择出来并投票生成一个检查点。至少 12 人中的 9 人需要参与检查点投票。

8. 结论

我们呈现了一套新颖的在区块链上运行分布式去中心化机器学习的有效工作量证明方案。我们的提案很容易被拓展到其他 AI 算法上。

我们简单回顾了相关工作并且列出了我们方案的特征。我们设计了一套不同的区块链框架，矿工因有效工作量而获益。我们还解释了防止并惩罚恶意行为者的机制。我们展示了用来将机器学习与区块链挖矿结合的角色分工、环境设置以及共识协议，以此创造并奖励有效工作量。

在我们的系统中，一名客户可以用一个由工作节点组成的分布式网络训练一个机器学习模型。在他们执行了预先分配的单位工作量后，矿工可以用特殊的 *nonce* 值尝试挖出新区块。在每一轮 iteration 中，nonce 由一个考虑了机器学习训练输入与中间值的公式生成。如果一名矿工找到了一个幸运 nonce，

他必须证明他诚实地执行了本轮 iteration，这样他的区块才能被网络其他参与者接受。区块验证意味着重新执行幸运的那一轮 iteration。因为矿工们使用数据并行训练他们的模型，他们需要用链下消息互相交换信息。这些消息主要涵盖了一个矿工对他本地模型的更新。这些更新会被同一个机器学习任务小组中的其他工作节点复制并采纳。消息历史被记录下来并被用在将来的区块验证之中。与其他区块链相比，一个节点总能够收到来自客户的经济补偿，而解决区块链难题则能带来额外收入。尽管我们限制了 nonce 的数量，但是网络的目标难度同时也被降低以保证与比特币协议相同的 10 分钟出块速度。我们将计算重点从挖矿过程转移到了机器学习训练，因此实际哈希运算的计算量占比极小。

我们还实现了一个 PoUW 的概念验证级产品。我们证明了借助我们的 PoUW 方案，客户可以比使用普通机器学习云计算方案节约更多成本，矿工可以比进行比特币挖矿获得更多收益。我们的方案也证明了在个人消费级硬件上合作训练出的机器学习模型可以有不错的表现。我们相信这个方案可以借助区块链技术的安全性使人工智能更加民主化。

在本文中我们以深度神经网络（DNN）为例，但是主要技术很容易被推广到其他基于 iteration 机制的 AI 算法。我们未来的工作包括给协议加入安全多方计算技术以及实现一个工业级产品。

致谢

本文作者向南加州大学计算机科学助理教授 Muhammad Naveed 表达衷心的感谢，本文的完成离不开他的杰出帮助与审阅。

REFERENCES

[1] Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system.

[2] Ball, M., Rosen, A., Sabin, M., and Vasudevan, P. N. (2017) Proofs of useful work. *IACR Cryptology ePrint Archive*, **2018**, 203.

[3] Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017) Accurate, Large Minibatch SGD Training ImageNet in 1 Hour. *CoRR*, **arXiv**.

[4] Strom, N. (2015) Scalable distributed dnn training using commodity gpu cloud computing. *INTERSPEECH*, pp. 1488–1492. ISCA.

[5] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012) Large scale distributed deep networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, USA NIPS'12, pp. 1223–1231. Curran Associates Inc.

[6] Skymind (2018) *Introduction to Distributed Training of Neural Networks*.

[7] King, S. (2013). Primecoin: Cryptocurrency with prime number proof-of-work. http://primecoin.io/bin/primecoin-paper.pdf.

[8] Decentralized machine learning: whitepaper. **https://decentralizedml.com**.

[9] SingularityNET: A decentralized, open market and inter-network for AIs. **https://public.singularitynet.io/whitepaper.pdf**.

[10] Gridcoin whitepaper. **https://gridcoin.us/assets/img/whitepaper.pdf**.

[11] Baldominos, A. and Saez, Y. (2019) Coin.AI: A proof-of-useful-work scheme for blockchain-based distributed deep learning. *CoRR*, **arXiv**.

[12] Li, M., Weng, J., Yang, A., Lu, W., Zhang, Y., Hou, L., Liu, J., Xiang, Y., and Deng, R. H. (2019) Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, **30**, 1251–1266.

[13] Boneh, D., Lynn, B., and Shacham, H. (2004) Short signatures from the weil pairing. *Journal of Cryptology*, **17**, 297–319.

[14] developers, D. Decred documentation. **https://docs.decred.org/**.

[15] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[16] Micali, S., Rabin, M., and Vadhan, S. (1999) Verifable random functions. *Proceedings of the 40th Annual Symposium on the Foundations of Computer Science*, New York, NY, October, pp. 120–130. IEEE.

[17] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. (2017) Algorand: Scaling byzantine agreements for cryptocurrencies. *Proceedings of the 26th Symposium on Operating Systems Principles*, New York, NY, USA SOSP '17, pp. 51–68. ACM.

[18] Pedersen, T. P. (1991) A threshold cryptosystem without a trusted party. *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, Berlin, Heidelberg EUROCRYPT'91, pp. 522–526. Springer-Verlag.

[19] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. (1997) Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, New York, NY, USA STOC '97, pp. 654–663. ACM.

[20] Mirrokni, V., Thorup, M., and Zadimoghaddam, M. (2018) Consistent hashing with bounded loads. *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 587–604.

[21] Duminuco, A. (2009) Data Redundancy and Maintenance for Peer-to-Peer File Backup Systems. Phd thesis Télécom ParisTech.

[22] Reed, I. and Solomon, G. (1960) Polynomial codes over certain fnite felds. *Journal of the Society for Industrial and Applied Mathematics*, **8**, 300–304.

[23] Goodfellow, I., Bengio, Y., and Courville, A. (2016) *Deep Learning*. The MIT Press.

[24] Schulze, M. (2011) A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, **36**, 267–303.

[25] Okupski, K. (2014). Bitcoin developer reference.

[26] Chen, T., Li, M., Cmu, U. W., Li, Y., Lin, M., Wang, N., Wang, M., Xu, B., Zhang, C., Zhang, Z., and Alberta, U. MXNet : A Flexible and E f cient Machine Learning Library for Heterogeneous Distributed Systems. *arXiV*, **2015**, 1–6.

[27] What is Kubernetes. **https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/**.

[28] Chen, J. (2019). How to build the perfect Deep Learning Computer and save thousands of dollars. **https://medium.com/the-mission/how-to-build-the-perfect-deep-learning-\\computer-and-save-thousands-of-dollars-\\9ec3b2eb4ce2**.

[29] Exxact Corp – The Best Deep Learning Workstations in the Business. **https://www.exxactcorp.com/Deep-Learning-NVIDIA-GPU-Systems**.

[30] Rist, L. Jan Encrypt your Machine Learning. *Medium*, **2018**.

[31] Blanchard, P., El Mhamdi, E. M., Guerraoui, R., and Stainer, J. (2017) Machine learning with adversaries: Byzantine tolerant gradient descent. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Red Hook, NY, USA NIPS' 17 118—128. Curran Associates Inc.

[32] Damaskinos, G., Mhamdi, E. M. E., Guerraoui, R., Patra, R., and Taziki, M. Asynchronous Byzantine Machine Learning. *arXiv*, **2018**.

[33] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002) Seti@home: An experiment in public-resource computing. *Commun. ACM*, **45**, 56–61.

[34] Miller, A., Juels, A., Shi, E., Parno, B., and Katz, J. (2014) Permacoin: Repurposing bitcoin work for data preservation. *Proceedings of the IEEE Symposium on Security and Privacy*, May. IEEE.

[35] Labs, P. (2017). Filecoin: A decentralized storage network. **https://filecoin.io/filecoin.pdf**.

[36] Vorick, D. and Champine, L. (2014). Sia: Simple decentralized storage. **https://sia.tech/sia.pdf**.

[37] Schrijvers, O., Bonneau, J., Boneh, D., and Roughgarden, T. (2017) Incentive compatibility of bitcoin mining pool reward functions. In Grossklags, J. and Preneel, B. (eds.), *Financial Cryptography and Data Security*, Berlin, Heidelberg, pp. 477–498. Springer Berlin Heidelberg.

[38] Miller, A., Juels, A., Shi, E., Parno, B., and Katz, J. (2014) Permacoin: Repurposing bitcoin work for data preservation. *Proceedings - IEEE Symposium on Security and Privacy*, may, pp. 475–490. IEEE.

[39] (2020) *Bitcoin Core version 0.9.0 released*.

[40] Du, J. (2018) *PoUW Research Report*.

[41] Zhang, B. and Srihari, S. N. (2003). Properties of binary vector dissimilarity measures.

[42] Prechelt, L. (2012) Early Stopping — But When? In Montavon, G., Orr, G. B., and Müller, K.-R. (eds.), *Neural Networks: Tricks of the Trade: Second Edition*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[43] Chiu, J. and Koeppl, T. V. (2018). Incentive compatibility on the blockchain.

[44] Waters, A., Harvilla, M., and Gerzanics, P. (2018) *PAI Data Storage and Sharing*.

[45] Castro, M. and Liskov, B. (1999) Practical byzantine fault tolerance. *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA OSDI '99, pp. 173–186. USENIX Association.

[46] Das, A., Gupta, I., and Motivala, A. (2002) Swim: Scalable weakly-consistent infection-style process group membership protocol. *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, Washington, DC, USA DSN '02, pp. 303–312. IEEE Computer Society.

[47] Backes, M. and Cachin, C. (2003) Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. *In Proc. of Intl. Conference on Dependable Systems and Networks (DSN-2003*, pp. 37–46. IEEE.

Algorithm 1 一轮 iteration 中需完成的工作

procedure ITERATION()
  $\mathbf{X}^{(\mathbf{tr})}, \mathbf{y}^{(\mathbf{tr})} \leftarrow Load(b_i)$ ▷ 加载一个 mini-batch
  for each $\boldsymbol{\Delta}_{\boldsymbol{i}}^{(\boldsymbol{p})} \in \boldsymbol{\Delta}^{(\boldsymbol{p})}$ **do**
    $\boldsymbol{\theta'} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{\Delta}_{\boldsymbol{i}}^{(\boldsymbol{p})}$ ▷ 用其他节点信息更新本地模型副本
  $\boldsymbol{g}^{(\boldsymbol{\ell})} \leftarrow Backprop()$ ▷ 用向后传播计算本地梯度
  $\mathbf{g}^{(\mathbf{r})'} \leftarrow \mathbf{g}^{(\mathbf{r})} + \boldsymbol{g}^{(\boldsymbol{\ell})}$ ▷ 更新剩余梯度
  $\mathbf{g}^{(\mathbf{r})''} \leftarrow \mathbf{g}^{(\mathbf{r})'}; \boldsymbol{\Delta}^{(\boldsymbol{\ell})} \leftarrow \mathbf{0}_{\boldsymbol{n}}; \boldsymbol{\delta}^{(\boldsymbol{\ell})} \leftarrow$ ▷ 初始化最终剩余梯度、本地权重列表和消息映射（n 代表梯度的长度）
  for each $g_i^{(r)'} \in \mathbf{g}^{(\mathbf{r})'}$ **do**
    if $g_i^{(r)'} > +\tau$ **then**
      $\boldsymbol{\delta}^{(\boldsymbol{\ell})} \leftarrow \boldsymbol{\delta}^{(\boldsymbol{\ell})} \cup e(\mathbf{1}, i)$
      $\Delta_i^{(\ell)} \leftarrow +\tau$
      $g_i^{(r)''} \leftarrow g_i^{(r)'} - \tau$
    else if $g_i^{(r)'} < -\tau$ **then**
      $\boldsymbol{\delta}^{(\boldsymbol{\ell})} \leftarrow \boldsymbol{\delta}^{(\boldsymbol{\ell})} \cup e(\mathbf{0}, i)$
      $\Delta_i^{(\ell)} \leftarrow -\tau$
      $g_i^{(r)''} \leftarrow g_i^{(r)'} + \tau$
  $\boldsymbol{\theta''} \leftarrow \boldsymbol{\theta'} - \epsilon \boldsymbol{\Delta}^{(\boldsymbol{\ell})}$ ▷ 用本地信息更新本地模型副本
  $\boldsymbol{K}^{(\boldsymbol{tr})} \leftarrow FwdPass(\boldsymbol{\theta''}, \mathbf{X}^{(\mathbf{tr})}, \mathbf{y}^{(\mathbf{tr})})$ ▷ 在训练 mini-batch 上评估模型指标
  $\boldsymbol{\delta}^{(\boldsymbol{p})} \leftarrow$ *Get peers message maps()* ▷ 接收并解压其他节点消息，提取消息映射
  $\boldsymbol{\Delta}^{(\boldsymbol{p})} \leftarrow Rebuild\ gradients(\boldsymbol{\delta}^{(\boldsymbol{p})})$
  $msg \leftarrow$ *Build message* (将表 1中的变量序列化到一个 IT_RES 消息，签名并压缩)
  $Send(msg)$ ▷ 向网络广播消息
  $Mine(...)$ ▷ 挖矿

Algorithm 2 挖矿操作

procedure MINE(...)
  $nonce_{precursor} \leftarrow hash(\boldsymbol{\theta''}|\Delta^{(\ell)})$ ▷ 计算前驱 nonce
  $nonce \leftarrow hash(nonce_{precursor})$ ▷ 计算 nonce
  $a = \omega_B * disk\_size(b_i) + \omega_M * model\_size(\boldsymbol{\theta''})$ ▷ 允许用来尝试挖矿的 nonce 数量
  for $j \leftarrow 0$ to $a - 1$ **do**
    $success = MineWithNonce(nonce + j)$
    if *success* **then**
      $Store(\boldsymbol{\theta}, b_i, g^{(r)}, \Delta^{(p)}, h_i)$ ▷ 矿工需要保存：iteration 开始时的模型、mini-batch、剩余梯度、其他节点更新以及相关消息历史 ID
      $block \leftarrow MakeBlock(...nonce, hash(h_i), hash(msg)...)$ ▷ 包含 PoUW 域的新区块
      $AddToPAIBlockChain(block)$ ▷ 将区块添加到 PAI 链上

表 A.1: BUY_TICKETS

交易	说明
txin[0..n]	指向已经存在的 UTXO
txout[0]	包含一个价值，它是当前相应类型（矿工券、监督者券等）券价的整数倍；脚本域为空！
txout[1]	以 P2PKH 形式向质押者返还的找零；这也是挖矿收益的收款地址。
txout[2]	一个声明券类型与偏好的 OP_RETURN 脚本：”TICKET-TYPE:preferences”

表 A.2: PAY_FOR_TASK

交易	说明
txin[0..n]	指向已经存在的 UTXO
txout[0]	仅声明质押的金额；脚本域为空！
txout[1]	向质押者返还的找零
txout[2]	一个声明任务的 OP_RETURN 脚本

APPENDIX A. 交易解析

PAI 节点通过读取交易最后的交易输出 TXOUT 来区分交易类型，TXOUT 的 OP_RETURN 脚本包含了描述信息。这里有一个实现案例，详情请参考表 A.1、表 A.2和表 A.3。

比特币的 OP_RETURN 输出规定了第一个操作码是 OP_RETURN，随后是入栈操作码。我们引入了一个被称为*结构化数据输出*的更丰富的格式，其中 OP_RETURN 域后面是 OP_STRUCT 操作码，再是一个包含数据项的数组。结构化数据输出的格式要求第一个数据项必须指明版本，随后是交易的详情。比如，PAY_FOR_TASK 交易的这个个域包含有关数据集、验证方式、优化器等信息。

APPENDIX B. 缩短任务等待时间

我们旨在缩短开始一个任务所需的等待时间。我们要求一项任务的时间戳必须早于包含这项任务的区块的时间戳，且这项任务最终必须出现在所有区块链分叉中。

每个节点在本地保存一个记录任务与任务发现时间（不同节点间可能有几秒的差距）的列表。需要往这个

表 A.3: CHARGE_FOR_TASK

交易	说明
txin[0..n]	指向任务相关质押金（包括客户手续费与恶意参与者被罚款项）；脚本域为空
txout[0..n]	向参与者支付款项的经典 P2PKH 输出；请注意只有一条 CHARGE_FOR_TASK 交易是可花费的
txout[n+1]	一条发布供客户下载的最佳模型的 OP_RETURN 脚本：”RESULT:taskHash:url”

列表添加元素时，节点从任务的时间戳开始搜索区块链，且需要遍历孤儿分叉。任何包含了任务提交的被挖出区块都是一个任务定义块，但是节点只能确认其中的一个，因此要选择不论在哪个分支上的最老的那个区块。如果出现多个包含相同任务提交的区块且他们的时间戳相同，则选择哈希值最小的那一个区块。

被选中的工作节点通过发送 JOIN_TASK 交易加入任务，这个交易包含了他们认定的任务定义块哈希与任务哈希。当选择某个任务定义块哈希的工作节点形成多数后他们就会开始训练。其他工作节点如果想参与会被拒绝。

训练结束后，任务提交时的账本已经稳定下来。节点们需要找到任务手续费区块的 PAY_FOR_TASK 费用：从任务的时间戳开始搜索区块链，仅遍历当前最长分支。第一个包含任务提交的区块即为任务手续费区块。

遵循这些步骤可以安全地在交易费用得到确认之前就开始机器学习任务。

APPENDIX C. 一个分布式密钥生成（DISTRIBUTED KEY GENERATION, DKG）方案

在我们的 PoUW 系统中，私钥的子部分由**分布式密钥生成**（DKG）协议在任务初始化阶段生成。这个协议是 Joint-Feldman 协议（ [18]）的一个改进版。它包括如下几个步骤:

- 每个监督者生成一个 $t-1$ 阶多项式 $\mathcal{S}_i(x) = s_{i,0} + s_{i,1}x + ... + s_{i,t-1}x^{t-1}$，除了 $s_{i,0}$ 之外的系数都是随机生成的私钥。$s_{i,0} = s_{k_i}$ 是每个参与者的 BLS 私钥。
- 基于他们的 $\mathcal{S}_i(x)$，所有监督者计算并广播他们自己的 $\mathcal{P}_i(x) = p_{i,0} + p_{i,1}x + ... + p_{i,t-1}x^{t-1}$，这是一个同阶的多项式，由对应的公钥组成 $p_{i,j} = g_i \times s_{i,j}, j \in \{0, ..t-1\}$。
- 每个人都通过把 x 替换成相应监督者的索引来计算包括自己在内所有人的 $\mathcal{S}(x)$。例如在一个 5 选 3 方案中，监督者 2 需要计算:

$$\mathcal{S}_2(1) = s_{2,0} + s_{2,1} \cdot 1 + s_{2,2} \cdot 1$$
$$\mathcal{S}_2(\mathbf{2}) = \mathbf{s_{2,0} + s_{2,1} \cdot 2 + s_{2,2} \cdot 2^2}$$
$$\mathcal{S}_2(3) = s_{2,0} + s_{2,1} \cdot 3 + s_{2,2} \cdot 3^2$$
$$\mathcal{S}_2(4) = s_{2,0} + s_{2,1} \cdot 4 + s_{2,2} \cdot 4^2$$
$$\mathcal{S}_2(5) = s_{2,0} + s_{2,1} \cdot 5 + s_{2,2} \cdot 5^2$$

- 每个监督者将每一个 $\mathcal{S}_i(x)$ 加密并发送给相关方。例如监督者 2 会用监督者 1 的公钥加密 $\mathcal{S}_2(1)$ 并将其发送给监督者 1。如果在一个预先设置的时间窗口内某方没有收集到所有私钥子部分，他会通过向区块链提交包含发送方索引的 *DKG_COMPLAINT* 交易来控诉发送方。
- 每条私钥子部分被接收到时会通过 $\mathcal{P}(x)$（将 x 替换成自己的索引）解密并验证。如果结果与从接收到的私钥子部分推导出的公钥不匹配，节点会通过向区块链提交包含身份信息以及接收到的密钥子部分的 *DKG_COMPLAINT* 交易来控诉发送方。
- 为了得到全局公钥（$\mathcal{P}_\mathbf{k}$），所有监督者会将所有多项式 $\mathcal{P}(x)$ 的自由项合并到一起，$\mathcal{P}(x)$: $\mathcal{P}_\mathbf{k} = \sum_{i=1}^{n} p_{i,0} = p_{1,0} + p_{2,0} + ... + p_{n,0}$，并且公开它。
- 全局私钥 $\mathcal{S}_\mathbf{k} = \sum_{i=1}^{n} s_{i,0} = s_{1,0} + s_{2,0} + ... + s_{n,0}$ 对于任何人都是保密的。
- 在协议的最后，所有监督者必须提交包含本地计算的 n 选 t 公钥 $\mathcal{P}_\mathbf{k}$ 的 *DKG_SUCCESSFUL* 交易。当在一个预定义的时间窗口 Δt 内有 n 条带有相同公钥的交易被观察到时，DKG 协议视为已成功，所有参与者可以进行下一阶段。

DKG 协议在机器学习任务的初始化阶段（密钥交换之后）或者在监督者小组人员出现变化时执行。在 DKG 阶段出现错误的监督者会被网络禁止活动并且被没收质押金。

要检测一个节点是否收到了错误的子部分或者收到了正确的子部分但是假装没有以排除其他节点有一定难度。如果 2/3 的节点都举报同一节点，则该监督者会被自动取消资格。而如果只有一个节点举报另一个节点，则该节点会基于现有证据发起投票以决定排除哪一个节点：发送者或接收者。如果人数减少后的诚实成员们仍满足人数需求，或者有新的成员取代犯错的成员，DKG 协议将被重新执行。选择额外工作节点的过程与任务注册阶段里的相同。矿工们会在第二个参与区块中列入 JOIN_TASK 交易，并指向第一个参与区块。

APPENDIX D. *N* 选 *T* 交易签名

每名监督者都有几份私钥子部分用来给 n 选 t 交易签名。对于一笔交易 tx，每名监督者需要执行如下步骤:

1. 计算 $H(tx)$，交易在 BLS 曲线上的哈希。
2. 发布 $Sig_i(tx) = \sum_{j=1}^{n} \mathcal{S}_j(i) \times H(tx)$，$i$ 是当前节点的索引，j 是私有子部分的索引。请注意每个节点 i 要对他的私钥子部分求和 $\sum_{j=1}^{n} \mathcal{S}_j(i)$ 并用其给交易签名。
3. epoch 组长收集至少 t 个签名子部分。

给定一笔交易 tx，当任意 t 个签名子部分被收集到之后，借助 BLS 门限签名性质，组长就可以通过拉格朗日插值重建交易上的全局签名（$Sig(tx)$），达到和使用全局私钥给交易签名一样的效果（$Sig(tx) = (s_{1,0} + s_{2,0} + ... + s_{n,0}) \times H(tx)$）。全局签名可以用全局公钥 $\mathcal{P}(x)$ 的自由系数来验证。

APPENDIX E. 崩溃恢复模型

我们使用一个崩溃恢复模型来检测何时有哪些节点离线。这个模型收到 [47] 这个框架的启发，它考虑到节点可能重复崩溃再恢复，或者永久离线的状况。在现实中尽管可能存在网络问题或者硬件/软件故障，但节点最终还是会上线并继续机器学习训练。

Appendix E.1. 离线检测

如果一个工作节点怀疑另一个工作节点（矿工或监督者）速度过慢或者在一个时间范围 t_r 内没有发送一定量的消息，那么他就可以发起一个测试以检测该节点是否离线（崩溃）。探查算法（算法 3）受到 SWIM 算法（ [46]）的启发，我们将其修改成拜占庭错误容忍（BFT，Byzantine Fault Tolerant 的。和 PBFT（ [45]）中一样，我们要求多于 1/3（一个 BFT 系统可以接受的最大错误节点数量）的节点声明某一个节点是在线时整个任务小组才能认定该节点在线。我们假定一个和 [45] 中一样的弱同步，即网络错误最终会被修复且大多数节点会快速从离线状态恢复。

任意一个工作节点 w_i 都保存了一个已知活跃工作节点的列表 W。如算法 3所示，一名监督者 s_i 向另一个工作节点 w_j 发送信息。如果在一个时间范围 Δt 内没有收到应答 s_i 会随机选择一个有 k 个工作节点的子集 $W_k \subsetneq W$，且并行地要求他们也向那个问题节点发信息。为了避免偏差，k 个被选择的节点由一个随机数生成器决定，随机数种子为最后一个任务参与块的哈希。如果子集中超过 1/3 的节点声明问题节点是在线的，则所有节点仍在他们的本地列表中保存这个节点，每个应答都经发送者签名。否则，发起质疑的监督者公开联系组长并要求组长决定时候替换问题节点（ReportOfflineNode 过程）。为了达到此目的。发起质疑的节点需要发起一笔包含其他节点应答信息以及调查原因（比如“离线”）的 *CHECK_NODE* 交易。组长发起一轮投票让监督者们声明该节点处于在线还是离线状态。每名监督者会直接向被质疑的节点发送消息，如果在一个时间段后没有收到应答就会投票声明其为离线。组长会和所有节点一起发布一个 n 选 t 签名的 *NODE_STATUS*

交易。如果最终状态是离线（超过 2/3 的监督者节点投票“离线”）则组长会发布一条包含被替换工作节点 ID 与替换原因的 *RECRUIT_WORKER_NODE* 交易。工作节点们会从他们的本地列表移除该离线节点。

算法 4与算法 3类似，但是仅由监督者运行。

一个节点如果频繁地在离线-在线状态间切换则必须用 *CHECK_NODE* 和 *NODE_STATUS* 交易从工作小组中移除（以“离线-在线”为原因）。恶意节点通过相同的方式不同的理由（例如“DOS 攻击”，“有害梯度”–发送错误更新以干扰机器学习训练等）被举报及验证。

Algorithm 3 一个工作节点 w_i 运行本算法来检测一个可疑节点 w_j 是否为离线状态。

procedure DetectOfflineNode(...)
  $r \xleftarrow{\Delta t} ping(w_j)$ ▷ 向 w_j 发送消息并在时间范围 Δt 内等待应答
  if $r == online$ **then**
    $W \leftarrow W \cup w_j$ ▷ 保存或添加于本地列表
  else ▷ No response
    $W_k \leftarrow rnd(W \setminus \{w_i, w_j\})$ ▷ 选择 k 个随机工作节点（一个 W 的子集）
    $c \leftarrow 0$ ▷ 在线节点计数器
    $R \leftarrow \{r\}$ ▷ 应答
    parfor $w_k \in W_k$ **do** ▷ 并行
      $R_k \xleftarrow{\Delta t} ping\_req(w_k)$ ▷ 向每个被选中的节点发送消息
      if $status(R_k) == online$ **then**
        $c \leftarrow c + 1$
    end parfor
    if $c > |W|/3$ **then** ▷ 如果超过 1/3 的节点报告该节点为在线状态
      $W \leftarrow W \cup w_j$ ▷ 节点处于激活状态
    else
      $s \leftarrow ReportOfflineNode(w_j, W_k, R)$ ▷ 报告 k 个被选择的节点以及他们的应答
      if $s ==$ offline **then**
        $W \leftarrow W \setminus w_j$ ▷ 从激活节点列表移除该节点

Appendix E.2. 监督者离线

一名监督者可能因为网络连接问题离线。如果他错过了超过 10% 的训练 iteration，那么他就会失去他的质押金并且无法再重新加入任务。

如果一名离线的监督者在 10% 的训练 iteration 内重新上线，他可以与其他监督者同步，通过读取来自其他监督者的公开流/数据库信息来更新。

如果离线的监督者错过了 10% 的训练 iteration，那么其他监督者会开始一个招募过程来取代缺席的监督者。组长会发起一笔 *RECRUIT_WORKER_NODE* 交易来指明需要一名新的监督者。

Appendix E.3. 矿工离线

如果一名矿工错过了所有训练操作的 5%，他就会失去他的质押金。否则，他可以重新加入任务。矿工不需要进行模型同步，但是要同步至最新的权重更新。

如果矿工数量降至初始数量的 80% 则监督者需要召集一个新矿工。组长会发布一条被称为 *RECRUIT_WORKER_NODE* 的特殊交易。被替换的矿工的质押金会被转移到任务费用中并在训练的最

Algorithm 4 组长运行本算法来决定是否声明一个工作节点为离线状态。

procedure INVESTIGATENODESTATUS(...)
  $c_x \leftarrow 0$ ▷ 离线节点计数器
  $r \xleftarrow{\Delta t} ping(w_j)$ ▷ 向 w_j 发送消息并在时间范围 Δt 内等待应答
  if $r \neq online$ **then**
    $c_x \leftarrow 1$ ▷ 离线节点计数器
  $R \leftarrow \{r\}$ ▷ 应答
  parfor $s_k \in S_k \setminus \{s_i\}$ **do** ▷ 并行
    $R_k \xleftarrow{\Delta t} ping\_req(s_k)$ ▷ 向每个监督者发送消息
    if $status(R_k) == offline$ **then**
      $c_x \leftarrow c_x + 1$
  end parfor
  if $c_x \geq 2/3|S|$ **then** ▷ 如果超过 2/3 的节点报告该节点为离线状态
    $Tr(RECRUIT\_WORKER\_NODE, R)$ ▷ 发起交易招募一个新的工作节点
  else
    $Tr(NODE\_STATUS\_ONLINE, R)$

后分配给评估者。

Appendix E.4. 恢复

离线但及时上线且没有错过太多 iteration 以致被取代的监督者或者矿工需要与其他成员同步他自己的内部数据库。重新接受以前的成员不需要投票的过程。

APPENDIX F. 分布式机器学习系统性能

分布式机器学习系统的效率比不上在单一机器上进行训练的效率。我们令 $E_f = 1$ 表示训练完全在一个机器上进行。在分布式系统中 E_f 会低很多，但是可以通过如图 F.1所示来并行一些操作以提高 E_f。

在图的左边部分我们列出了在单一机器上需要执行的操作；在中间部分，我们列出了一个未经优化、所有操作顺序执行的场景。在右边部分，我们改进了执行顺序：在整个 iteration 期间其他节点消息异步地被接收，其他节点更新被一次性求和并应用，同时消息映射由其他相关操作并发地处理。

我们分别在只使用 CPU 和配备了 GPU 的机器上测试了主算法中多个步骤的运行时间。第一个机器是一台 2017 款 MacBook Pro 2017，配有 2.8 GHz Quad-Core Intel Core i7 CPU 和 16 GB RAM。GPU 机器是一台 NC6 Microsft Azure machine（E5-2690v3 Intel Xeon CPU，56 GB RAM，1 x K80 GPU）。我们在图 F.2与图 F.3中列出测试结果。如所预想的一样，大部分与机器学习相关的操作（例如反向传播、模型更新）在 GPU 机器上执行得更快。

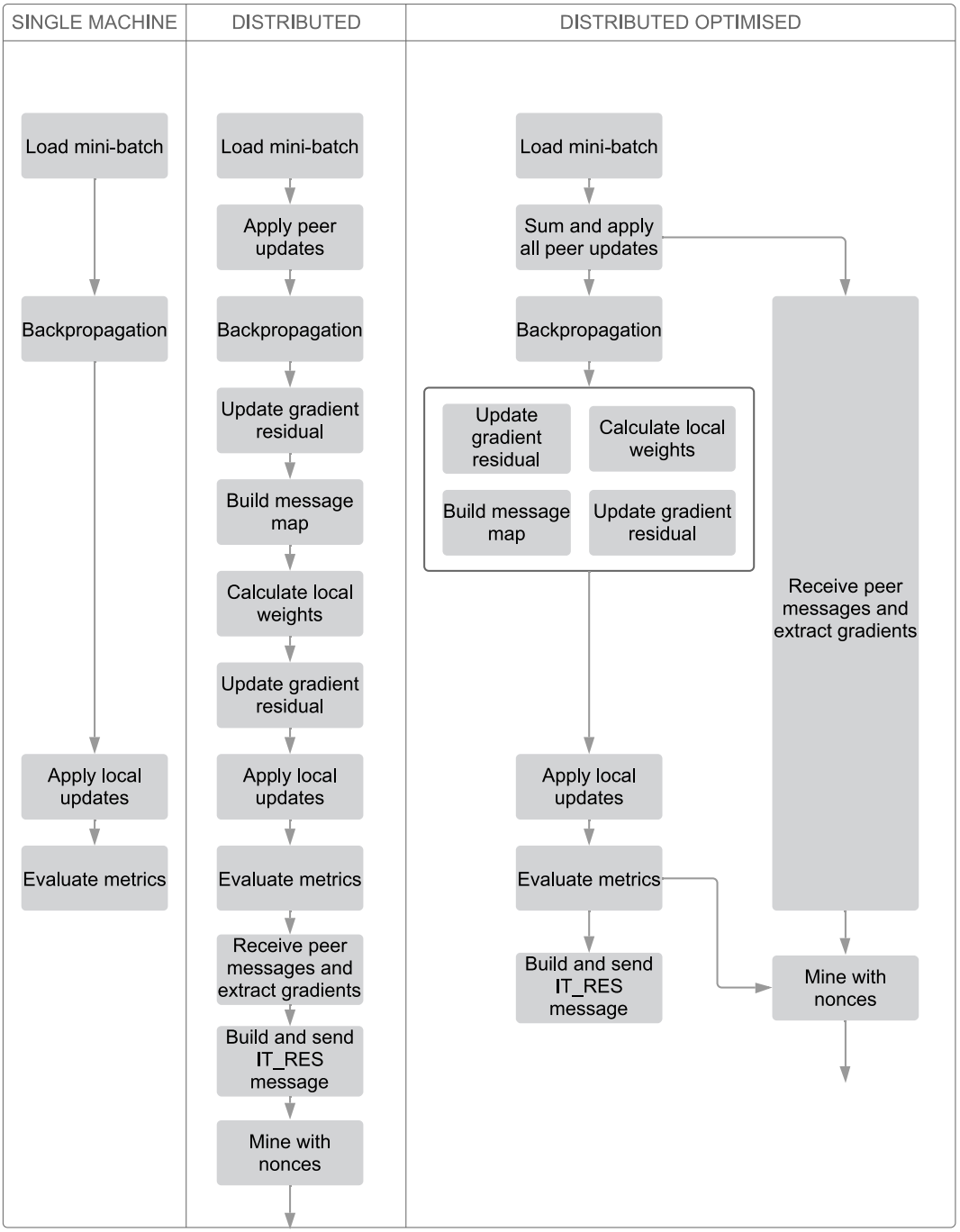
SINGLE MACHINE
DISTRIBUTED
DISTRIBUTED OPTIMISED
Load mini-batch
Backpropagation
Apply local updates
Evaluate metrics
Load mini-batch
Apply peer updates
Backpropagation
Update gradient residual
Build message map
Calculate local weights
Update gradient residual
Apply local updates
Evaluate metrics
Receive peer messages and extract gradients
Build and send IT_RES message
Mine with nonces
Load mini-batch
Sum and apply all peer updates
Backpropagation
Update gradient residual
Calculate local weights
Build message map
Update gradient residual
Apply local updates
Evaluate metrics
Build and send IT_RES message
Receive peer messages and extract gradients
Mine with nonces

图 F.1: 性能优化

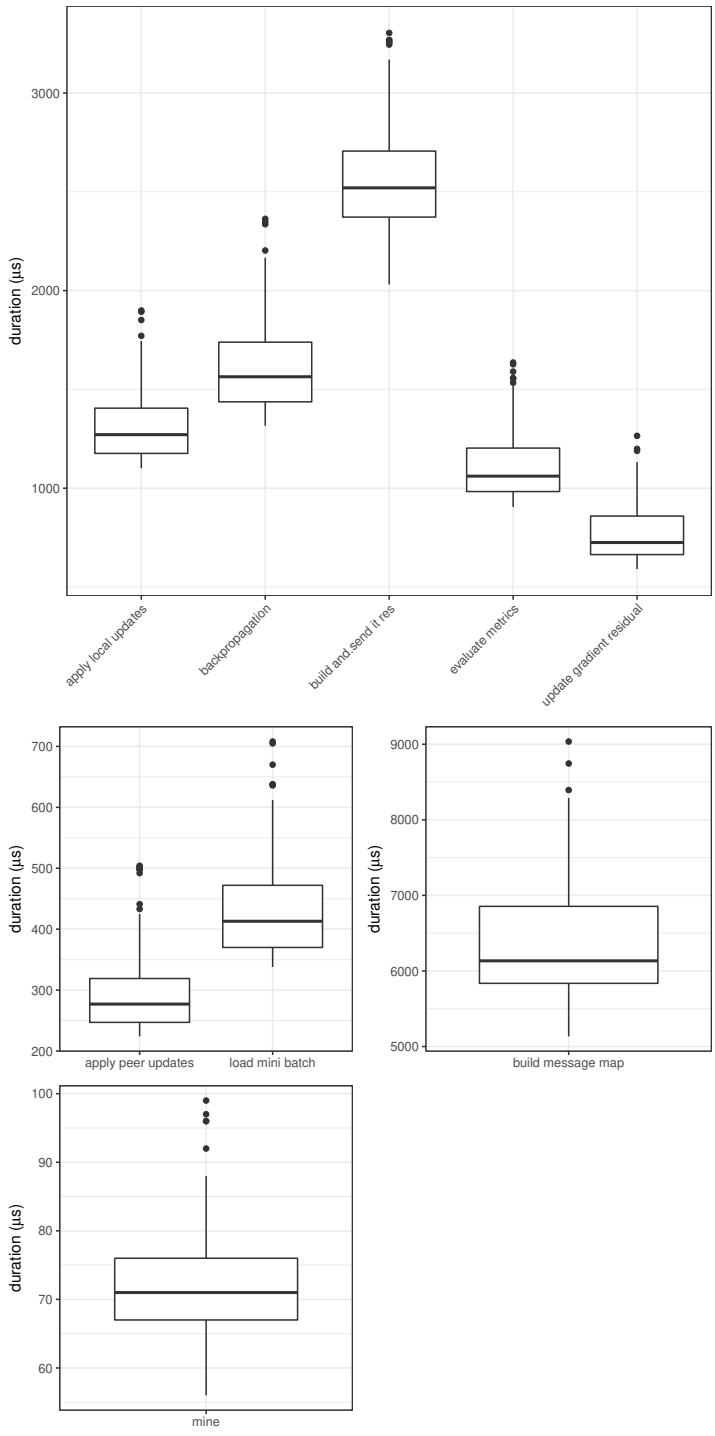

图 F.2: **CPU** 上不同运算步骤的时间测试

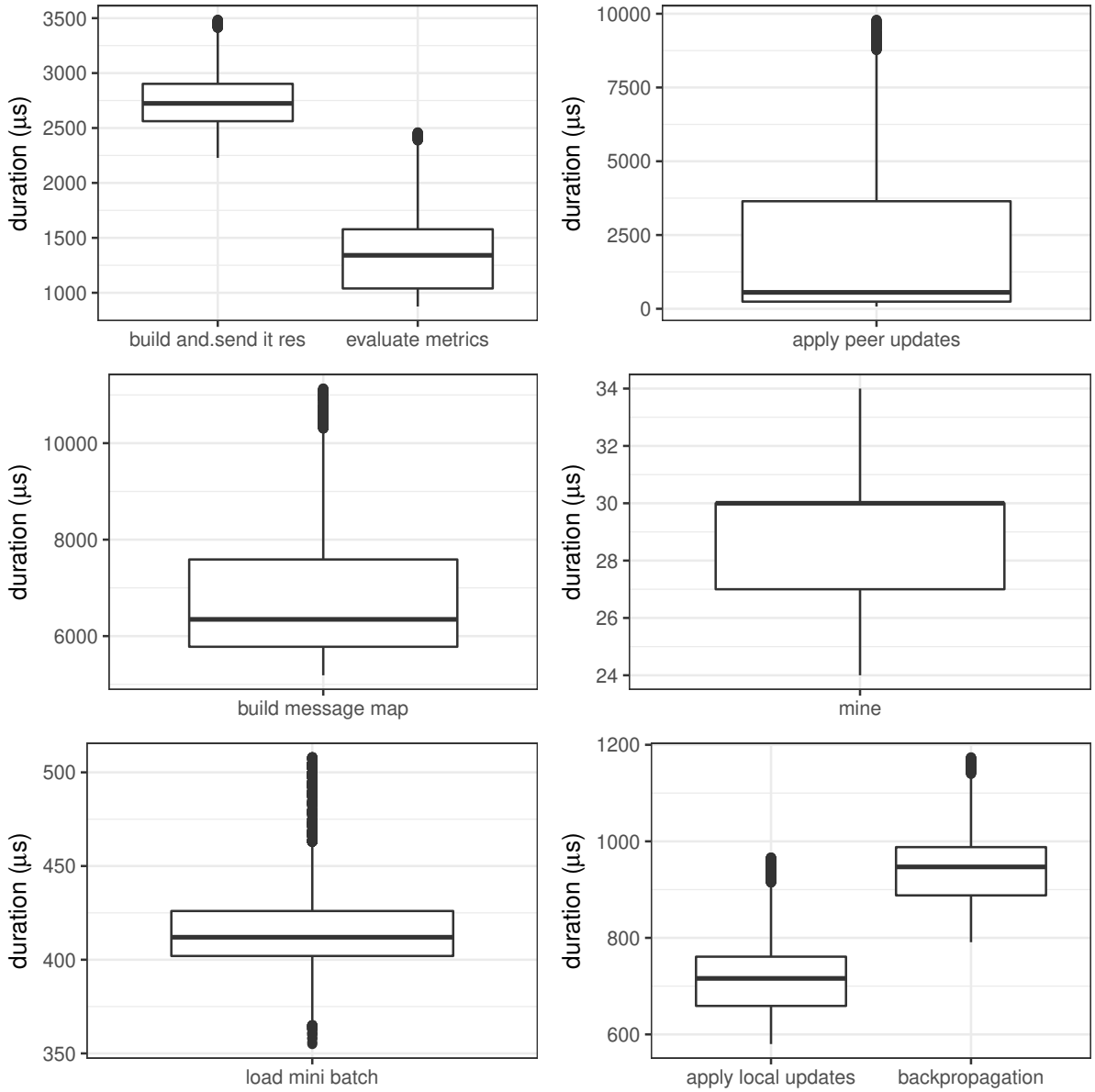

图 F.3: **GPU** 上不同运算步骤的时间测试