基于区块链的联邦学习算法

在联邦学习过程，如果恶意参与方提供错误的梯度参数更新来破坏模型训练的正确性，或者懒惰的节点不积极贡献模型数据都会造成模型训练的偏差。如何让多方持续积极参与模型数据训练过程，进行公平，公正的生产协作，成为制约行业发展的重要挑战之一。

区块链共识算法及智能合约等技术具有天然的激励作用，通过此类区块链技术特性能够激励多方积极参与协作。区块链和联邦学习相互融合，建立基于区块链的联邦学习算法，有望能解决这一行业难题。

本节根据联邦学习激励方式的不同，分别从模型质量评估，权重值以及激励分配等几个角度，介绍目前几种基于区块链的联邦学习算法。

1） 模型质量评估激励算法

联邦学习要把每一个数据孤岛组织起来，以此纳入联邦学习协作的体系中来。首先要对本地数据进行预处理，其次对训练数据特征化，然后再对模型数据进行质量评估。在联邦学习过程中，评估数据模型质量的精准度是影响训练效果的关键指标。

模型评估的数据量级、数据有效性、数据信息密度、数据真实性等指标，同时起到了数据监测与量化评估的作用。

在中心化的联邦学习过程，中央聚合器需要汇集所有的上传本地模型数据，以生成新的全局模型。这个过程中存在着模型投毒等安全攻击，可能导致整个模型训练的故障。

如何让参与者一直持续地贡献模型数据？在区块链智能合约里发布一个任务奖励，依据模型质量评估算法的结果，对完成任务的训练者给予相应的奖励。智能合约为联邦学习构建了一个模型数据交易的市场，诚实的参与者可以通过完成任务来盈利。一个良好的收益分配机制是各方高效协作的关键，这里介绍一种根据参与方所产生效益进行分配的激励方法。

联邦参与方的效益是指它加入联邦训练团队时所产生的效用。

一般而言，一个参与方 i 在给定收益分享轮次 t 中，从总预算 B(t)得到的分期收益可以表示为$\hat{u}_i(t)$，计算公式为：

$$\hat{u}_i(t) = \frac{u_i(t)}{\sum_{i=1}^{N} u_i(t)} B(t)$$，

式中，$u_i(t)$表示参与方 i 对收益 B(t)产生的效用，假定每一个参与方 i 对集合体作出的边际收益（假设集合体只包含参与方 i），用于计算它能得到的收益的分成。其数值计算方式：

$$u_i(t) = v(\{i\})$$

式中，v（X）表示评估集合体 X 效用的函数。X 表示参与方的集合。

本节以 MSE(Mean Squared Error)作为评估集合体 X 效用的函数，介绍一种模型质量评估激励算法。

MSE（Mean Squared Error）均方误差是线性回归中常用的损失函数，线性回归过程中尽量让该损失函数最小。模型评估之间的对比也可以用它来比较。通过计算观测值与真实值偏差的平方和与观测次数的比值，得到每次模型数据评估精准度 MSE。

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(f_i - y_i)^2$$

其中 m 为样本的个数，MSE 值范围[0,+∞)。MSE 值越小，说明模型数据的精准值越高。当预测值与真实值完全相同时，MSE 值为 0;

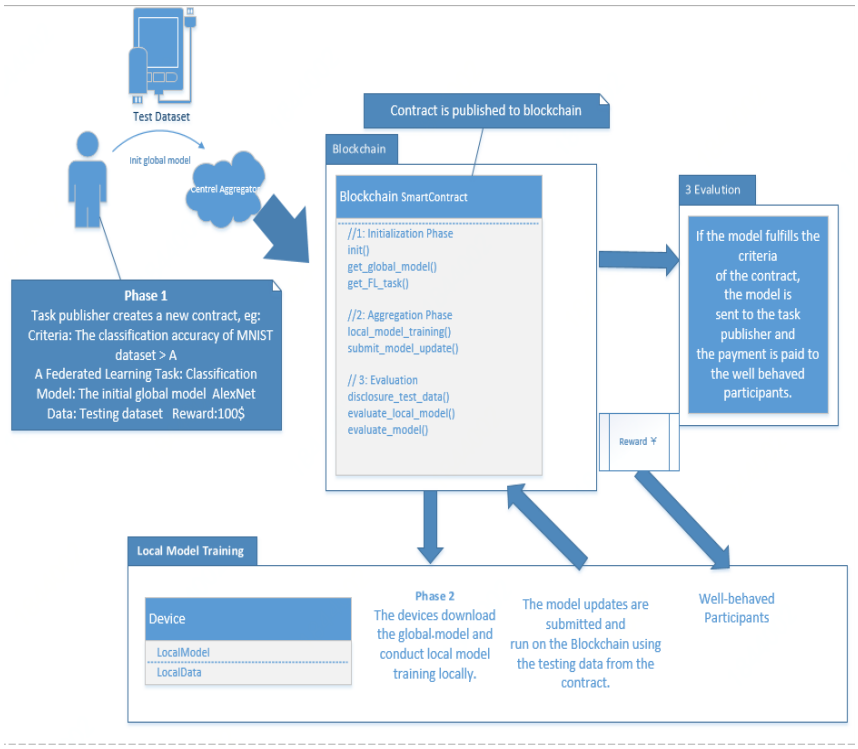
下图是在区块链上部署的一个智能合约，将联邦学习训练过程中的参与者，训练任务，训练状态及任务奖励等存放在智能合约中。通过智能合约发布模型训练任务。在联邦学习过程中，根据模型质量评估算法对模型数据进行评估后，将正确的 MSE 值更新到智能合约中。当完成本轮训练后，通过智能合约的预置事件，触发支付合约进行有序的交易执行。

```
/**
  Precondition: the ids of the data owners are their addresses.
*/

contract SmartContract {

    //model Data where to assign  validators, trainers, mse , status, fedratedAggregator ...
    struct ModelData {

            string modelId;

            address[] trainers;

            address[] validators;

            uint[] msesByIter; // El MB chequea con su mse contra esto

            mapping(address => uint[]) partialMsesByIter;

            uint currIter;

            uint improvement; // Percentage

            uint frozenPayment;

            mapping(address => uint) contributions; // Percentages

            Status status;

            address owner; // Model Buyer's are the owners

            address federatedAggregator; // federatedAggregator orchestrator

        }

        enum Status {INITIATED, STOPPED, FINISHED}

        constructor() public {
            prices["LINEAR_REGRESSION"] = 1000000000000000000;
        }

        /** EVENTS **/
        event ModelCreationPayment(address owner, uint256 amount);
        event ContributionPayment(address receiver, uint256 amount);
        event ValidationPayment(address receiver, uint256 amount);
}
```

智能合约中的模型数据结构

整个联邦学习过程主要通过数据初始化，交换本地训练的模型数据和

模型数据质量评估及激励分配三个阶段，在智能合约中实现训练任务安全协作和有效激励。



模型质量评估激励过程

步骤 1 初始化阶段

中央聚合器调用 init()函数初始化联邦学习的配置参数，测试数据集以及全局模型等设置。联邦学习任务发布者在智能合约里创建任务奖励，指定训练用的模型和聚合器等。聚合器将初始的全局模型数据任务发送给所有参与者，对一些敏感的中间传输数据加密安全传输处理。

```
/** ContractService() init phase: 1 deploy a smart contract 2 init model in SC **/
init():
    var SmartContract = artifacts.require("SmartContract");
    module.exports = function(deployer) {
        deployer.deploy(SmartContract);
    };


function initModel(string memory modelId, address[] memory validators, address[] memory trainers,
    address modelBuyer, address federatedAggregator) private pure returns (ModelData memory) {
     ModelData memory model = ModelData({
        trainers: trainers,
        validators: validators,
        modelId: modelId,
        msesByIter: new uint[](200),
        improvement: 0,
        currIter: 0,
        frozenPayment: 0,
        status: Status.INITIATED,
        owner: modelBuyer,
        federatedAggregator: federatedAggregator
     });

     return model;
}
```

```
#whole federated aggregator flow with python

app = create_app()

api.init_app(app)

encryption_service = EncryptionService(is_active=app.config["ACTIVE_ENCRYPTION"])

data_owner_service = DataOwnerService()

contract_service = ContractService()

contract_service.init(app.config)

federated_aggregator = FederatedAggregator()

data_owner_service.init(encryption_service, app.config)

federated_aggregator.init(encryption_service=encryption_service,data_owner_service=data_owner_service,
                          contract_service=contract_service,config=app.config)

logging.info("federated_aggregator running")
```

初始化过程

步骤 2 本地训练阶段

每个设备收到新的模型训练任务后，执行 localModelTraining()函数对本地数据进行训练。为防止联邦学习过程的模型精度损失，通常使用梯度压缩等算法来传输本地的模型数据参数。

当设备完成本地训练后，执行 submitModelUpdate() 将本地模型数据更新提交给聚合器，以赚取该设备本轮次的收益机会。

步骤 3 评估阶段

中央聚合器调用 evaluateModel()评估来自设备端上传的数据。如果该设备的模型数据通过评估，则将本次评估的结果 MSE 值保存在智能合约中作为贡献依据。在计算本轮训练贡献值函数 calculateContributions()中，按照对模型质量的精度值提升的效用，作为边际效益奖励的标准。这样只有高质量模型设备才会获得本轮次的收益；不安全的模型数据提供者将会被排除在外，从而失去本轮收益的机会。

在模型更新次数达到上限时，触发 ContributionPayment 事件。智能合约调用 finalizeContract()执行对设备的奖励。模型购买者依据智能合约中初始的协定，将本轮奖励 B(t)分配给模型训练者。然后中央聚合器调用 postEvaluation() 发布此任务的结果。最终任务发布者得到一个更高质量的全局模型。

```
/**
    finalize Contract to payment training contributors
*/
function finalizeContract(string memory modelId) public onlyModelBuyer isFinished(modelId) {
    ModelData storage model = models[modelId];

    _calculateContributions(modelId);

    // Pay trainers
    for (uint i = 0; i < model.trainers.length; i++) {
        address payable trainer = address(uint160(model.trainers[i]));
        _calculatePaymentForContribution(modelId, trainer);
    }
    // Return payments to model buyer
    returnModelBuyerPayment(modelId, address(uint160(model.owner)));
}

/**
    Accoding to U(i) / N methods, pay for the Data Owner with his address
    for his work done training the model.
*/
function _calculatePaymentForContribution(string memory modelId, address dataOwner)
    private isFinished(modelId) view returns (uint) {
      uint paymentForImprov = (payments[modelId] * getImprovement(modelId)) / 100;
      uint paymentForImprovForTraining = (paymentForImprov * 70) / 100;  //training contribution percent
      return paymentForImprovForTraining;
}
//core calcuate contribution func
function _calculateContributions(string memory modelId) public isFinished(modelId) {
    ModelData storage model = models[modelId];
    uint iter = model.currIter;  //current iteritor of this training process
    // improvement = ((initialMse - currMse) * 100) / initialMse
    model.improvement = calculateImprovement(model.msesByIter[0], model.msesByIter[iter]);

    //caculate the total contributions of the training
    uint contributionsSum = 0;
    for (uint i = 0; i < model.trainers.length; i++) {
        address trainer = model.trainers[i];
        //difference = intialMse - currMse;
        model.contributions[trainer] = mseDifference(model.msesByIter[0], model.partialMsesByIter[trainer][iter]);
        contributionsSum = contributionsSum + model.contributions[trainer];
    }

    //update the contributotion percecent of every training
    for (uint i = 0; i < model.trainers.length; i++) {
        address trainer = model.trainers[i];
        //contributionPercentage = (mseDiff * 100) / totalDifference
        model.contributions[trainer] = contributionPercentage(model.contributions[trainer], contributionsSum);
    }
 } ? end  calculateContributions ?
```

MSE 边际效益分配方法

2 权重值激励算法

中心化聚合器需要在许多客户端协作下才能完成聚合操作，需要不断地给这些客户端广播新的全球模型。这需要较高的网络带宽，会受到单一服务器稳定性的影响。中心化聚合过程可能偏向某些客户，有的会提供恶意的有毒模型，甚至从更新的模型数据中收集客户的隐私数据。

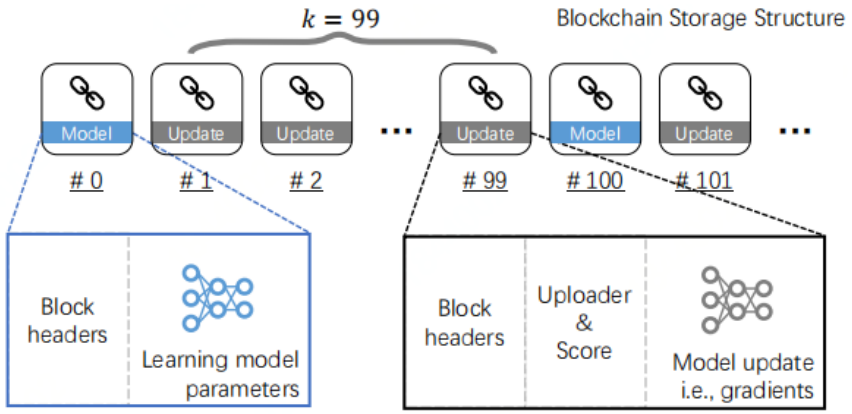
如何建立更加安全公平可靠的激励机制？可以用区块链网络来替代中心化聚合器。通过选举一个区块链共识委员会来验证和交换客户端设备的本地模型更新，通过后上链保存，并且依据该上链的模型数据作为后续的激励权重参考。以减少一致性的计算量，防御恶意模型攻击，进行更加安全高效的协作。

基于区块链的权重分值激励设计

将联邦学习的模型数据放在链上，保证了多协作方数据互访的一致性，形成了一个公开透明的模型数据账本。每一个区块负责记录联邦学习训练过程中的一份合格的数据证据，并将全局和本地模型数据分别保存在不同类型的区块上。

开始时初始化模型被放在#0 块，然后进行第一轮的训练。设备获取当前要训练的全局模型任务并执行本地训练，当有足够的更新块时，触发智能合约开始聚

合。设备通过选举一个共识委员会来验证和交换各自提交的模型数据，验证节点将验证的模型数据结果上传到新的区块上，完成本轮训练后将会生成一个新的全局模型块放置在链上。联邦学习只依赖最新的模型块数据和历史块数据，历史块数据仅作为故障回退或块验证用途。



区块生产过程

k：是每轮训练验证需要的更新块数(total number of Update，例如 100)
t： 是本轮训练的轮次 (t = 0，1，2, ...,)

模型块(Model)：是存放本轮所产生的全局模型数据的区块。
公式：# k×(t + 1) 是本轮训练第 t 轮次的所产生的模型块。

更新块(Update)：是存放设备提交的本地模型数据的区块。
公式：# [k×(t+1)+1, ((t+1) × (k+1))-1] 是第 t 轮次的更新的区块集合。

如下图所示，区块链前 100 区块#[0,100]块信息说明。

区块数	类型	计算公式	描述
#0	模型块	# t×(k+ 1) = #0 当 k = 99，t = 0 时	#0，是区块链上第 0 块 存放联邦学习模型初始化数据，在联邦训练的初始化过程生成。
#[1, 99]	更新块	#[t×(k+1)+1, (t+1)× (k+1))-1] = #[1, 99] 当 k = 99，t = 0 时	#[1, 99]，是区块链上第 1-99 区块集合(k 为轮次，t 为 100), 联邦学习过程评估通过的更新块，由设备提交生成。
#100	模型块 块 #	#t×(k+ 1) = #100 当 k = 99，t = 1 时	#100 表示区块链上第 100 块 是第一轮联邦训练生成的模型块

从实现的角度来看，一个模型块应包括：块标头、t 轮次和全局模型数据。一个更新块应包括：t 轮次，本地更新的模型梯度数据，权重值(weight score)这些基本的联邦训练数据。

区块链的链式结构保证了数据的确定性。基于竞争机制将正确的块追加到链上是共识的关键。考虑计算，通信成本和协商一致，建立一个高效和安全的委员会共识机制，在设备的模型数据上链之前对其进行验证。委员会由几个诚实的共识节点构成，负责验证设备提交的数据和协商出块。其余节点执行本地训练，将本地更新发送给委员会。开始一轮新的训练时，上一轮委员会不会再次当选。根据得分情况重新选举一个新的委员会。委员会成员通过质量模型评估算法，对设备提交的模型数据进行评估打分。只有通过评估的设备模型数据才会被委员会接收并给与激励。

权重值(weight score)计算

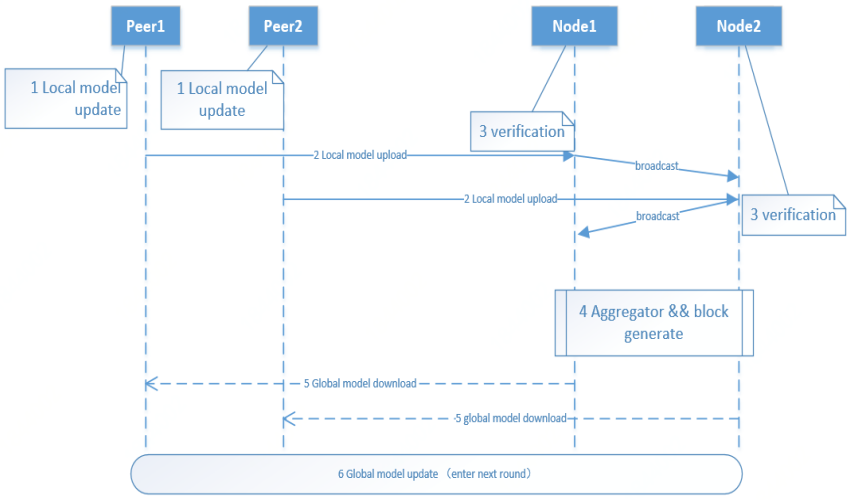
基于上述区块链委员会的联邦学习场景，考虑到完整性和稳定性等，可以根据每个设备的本地学习准确性或者参与频率评估其权重值，对联邦学习进行对比打分，来更新全局模型数据。

这里介绍以学习准确性作为评估权重值的实现方式。

Peer 是参与联邦训练的设备，主要负责训练本地数据，并将训练好的本地模型数据上传(1 Local mode update)，发给共识 Node 进行模型质量评估(2 Local model upload)。每轮通过委员会共识生成新的区块(4 Aggregator && block generate)，Peer 更新下载最新的全局模型(5 global model download)，并进行下一轮的本地更新，直到整个模型训练完成。

Node 是所有设备选举出来的共识委员会节点，负责验证设备上传的模型数据。和其它 Node 节点达成共识后生成新的区块，存储模型数据。在具体联邦学习过程中，当 Node 收到设备节点发来的模型数据，使用给定的测试数据集验证，删除可能损坏全局模型的有毒本地模型更新。交易验证通过后，Node 将加上自己签名广播给其它共识 Node 节点，进行验证(3 Cross Validation)。只有精度值高于给定阈值的合格模型才会被接受，生成下一次迭代新的、可靠的全局模型(6 global model update)。

整个交互过程如下图所示：



模型数据更新流程

以 MSE 评估模型数据作为权重值评价依据，权重值计算公式如下(详见 https://arxiv.org/abs/1808.03949)：

$$w^{(\ell)} = w^{(\ell-1)} + \sum_{i=1}^{N_D} \frac{N_i}{N_S} \left(w_i^{(\ell)} - w^{(\ell-1)} \right)$$

该式表示为在 l 代次最后一轮迭代中，本地权重和全局权重之间的计算。其中 w 为权重，S 为数据样本，D 表示设备，N 为设备数，Sk ∈ S，基于 MSE 函数 f(w) 的权重值算式。

在 l 代次迭代中，设备节点上传的本地模型数据更新为：

$$(w_i^{(\ell)}, \{\nabla f_k(w^{(\ell)})\}_{s_k \in \mathcal{S}_i})$$

完成本轮所有本地结果验证后，本轮次的全局模型更新为：

$$(w^{(\ell)}, \nabla f(w^{(\ell)}))$$

以每个设备提交的本地模型数据长度大小作为权重值标准，当通过委员会验证后，记录在链上的更新块(Update)中。本轮训练结束后，设备总权重值即为已验证通过得更新块模型数据大小总和。生成新的模型块，记录所有的全局模型数据相关更新，并依据权重值对各个设备的贡献进行量化激励。

在每轮训练结束时，重新选举一个新的委员会来生成下一个全局数据模型块。在分布式的模型训练场景下，这个委员会的决策效率会显著地提升整个全局模型训练的性能。

3 激励分配算法

在一些联邦学习的场景中，有些公司可能已经在当前市场上拥有众多的客户设备，积累了大量的高质量数据资源。在构造联邦学习模型过程，这些公司的资源是非常宝贵的。但是通过协作训练共同的全局模型，这些市场领导者可能会无意地帮助到它的竞争者们。因为联邦学习模型在所有参与方之间共享，从而导致市场领导者竞争优势的损失。因此，为联邦学习设计的收益分配方法应该考虑参与方加入联邦会产生的代价。这需要更有效的激励机制，鼓励设备更新本地模型数据，有偿支付费用给这些全局模型数据的贡献者，建立更加公平的按贡献分配利润的经济激励机制。

考虑到上述的需求，结合第一节中提到的基于模型质量评估 MSE 算法和边际效用分配方法。可以在该激励分配方法基础上，增加设备许可访问费和对其它参与方的奖励，使其更加符合现实场景需求。

- 许可费：

  每个设备支付一些全局模型的访问许可费，这些费用支付给模型保管者后，设备节点可以无限制地访问最新的模型。在联邦学习网络中，设备加入时，需要先购买一个模型许可费，只有购买后设备训练者才被允许访问模型任务并且参与全局模型的训练过程。如下图所示：

```
function newModel(string memory modelId, address[] memory validators, address[] memory trainers,
    address modelBuyer) public onlyFederatedAggr {
     models[modelId] = initModel(modelId, validators, trainers, modelBuyer, msg.sender);
}


/**
   Adds a new Data Owner to the DataOwners set.
   @param doAddress the data owner address used as value in the mapping
*/

function setDataOwner(address doAddress) public {
    dataOwners[doAddress] = true;
}

/**
   Adds a new setModelBuyer to the ModelBuyer set.
   @param doAddress the modelBuyerAddress address used as value in the mapping
*/

function setModelBuyer(address modelBuyerAddress) public {
    modelBuyers[modelBuyerAddress] = true;
}


/**
    Called from ModelBuyer when ordering training of model.
*/
function payForModel(string memory modelId, uint256 pay) public payable onlyModelBuyer {
    //check parameter ....
    if (payments[modelId] == 0) {
        payments[modelId] = 0;
    }

    payments[modelId] += pay;
    emit ModelCreationPayment(address(this), pay); //event ModelCreationPayment(address owner, uint256 amount);
}
```

购买模型许可费

- 利润分配

每轮聚合器完成聚合后，基于设备提交的模型数据精度贡献值，购买者将奖励分配给相应的设备。经常提供更新的设备有机会赚取更多奖励，并能不断下载更新全局模型数据，这样会以吸引更多设备参与。这种激励机制具有很高的可扩展性，可以适应不同的实际应用。

如果参与方付出的代价非常高，联邦带来的收益可能不够补偿这一代价，因此联邦学习可能要求分期地支付给参与方。这将会进一步导致“利息”的问题，因为从本质上来说，参与方是在将各自的资源（如数据）借给联邦以产生收益。

为了维持数据联邦的长期稳定，并且在以后逐渐吸引更多高质量的参与方加入，需要一种强调公平性，并且适合联邦学习环境的激励机制。在这里，我们基于第一节激励算法基础上，综合实现了一种更加优化的分配激励方法。它允许在联邦学习环境中灵活地解决上述涉及收益分享的问题。通过最大化可持续的经营目标，动态地将给定的预算奖励分发给联邦中的各个参与方，同时最小化参与方间的不平等问题。

```
/**
   payment functions, according to contribution of tranners, validators, orchestrator by ModelBuyer
*/
function generateTrainingPayments(string memory modelId) public onlyModelBuyer isFinished(modelId) {
    ModelData storage model = models[modelId];
    _calculateContributions(modelId);
    // Pay trainers
    for (uint i = 0; i < model.trainers.length; i++) {
        address payable trainer = address(uint160(model.trainers[i]));
        executePayForContribution(modelId, trainer);
    }
    // Pay validators
    for (uint i = 0; i < model.validators.length; i++) {
        address payable validator = address(uint160(model.validators[i]));
        executePayForValidation(modelId, validator);
    }
    // Pay orchestrator
    executePayForOrchestration(modelId, address(uint160(model.federatedAggregator)));
    // Return payments to model buyer
    returnModelBuyerPayment(modelId, address(uint160(model.owner)));
} ? end generateTrainingPayments ?
```

为了鼓励参与方加入联邦学习，区块链可以通过超时检查和经济惩罚机制为参与者提供公平性。对于不能准时到达和执行错误提交的参与者，会采用经济惩罚机制，撤销或冻结不诚实参与者的预存款，将其重新分配给诚实参与者。以此来进一步避免不诚实参与者和激励参与者盈利的积极性。

通过这样的市场化经济激励模型，可以在智能合约中比较灵活地实现了各方利益之间的平衡。任务激励，抵押和超时惩罚机制，从经济利益方面防止参与者进行模型攻击等恶意行为。促使联邦学习过程在模型数据质量，活跃度，有效协作方面形成的良性发展的生态。