Federated Learning for Adaptive Road Efficiency (FLARE)

Tyler Trott - V01040680
CSC 466 - Overlay and Peer-to-Peer Networking
Instructor: Jianping Pan
April 11, 2025

Abstract

As cities grow and urban transportation systems become increasingly complex, managing traffic efficiently has become a significant challenge—particularly in environments with constrained computing and communication capabilities, such as Wireless Sensor Networks (WSNs). This project introduces Federated Learning for Adaptive Road Efficiency (FLARE), a framework designed to optimize traffic flow predictions using a decentralized mesh of traffic sensor nodes. By leveraging federated learning with gossip-based protocols, nodes can collaboratively improve local models without relying on a central server, reducing communication overhead and enhancing scalability. Each node in the system is trained on localized traffic data, gathered from real intersections in downtown Victoria, BC, and then shares model updates with its immediate neighbors using a fully connected cluster-based topology called QuadMesh.

The simulation integrates OMNeT++ for network modeling and SUMO for realistic traffic generation, with Python scripts handling data preprocessing and linear regression model training. Results show promising performance in early simulation rounds, with model convergence occurring through iterative neighborhood updates. The system demonstrates potential for scalable, distributed traffic control and lays the groundwork for future enhancements such as anomaly detection and real-time adaptation. This project highlights the viability of combining federated learning and gossip communication as a foundation for next-generation smart city traffic solutions.

1 Introduction

Efficient traffic management is a critical aspect of urban infrastructure, especially as cities grow and transportation networks become more congested. Traditional centralized traffic control systems often struggle with scalability, latency, and single points of failure. As a result, there is growing interest in decentralized, intelligent systems that can respond to traffic conditions in real time.

Wireless Sensor Networks (WSNs) offer a promising foundation for distributed traffic monitoring and control, but they come with significant limitations. Nodes in WSNs typically have constrained processing power, limited storage, and restricted communication bandwidth. These limitations make it challenging to apply traditional machine learning techniques, which often require centralized data aggregation and substantial computational resources.

The objective of this project is to design and simulate a distributed traffic optimization framework that works within the constraints of WSNs. The proposed system, GossipGrid, combines federated learning with gossip-based communication protocols to enable nodes to collaboratively learn traffic patterns while keeping raw data local. This approach reduces communication overhead and enhances scalability and fault tolerance.

The system uses real traffic data from intersections in Victoria, BC, and is simulated using a combination of SUMO for realistic traffic flow and OMNeT++ for network behavior. A cluster-based topology called QuadMesh enables fully connected local learning and inter-cluster communication. The project demonstrates how decentralized learning strategies can be applied in low-resource environments to support smart city applications like dynamic traffic control.

2 Background and Related Work

The development of intelligent transportation and decentralized learning systems brings together a number of research areas, including wireless sensor networks (WSNs), federated learning (FL), and gossip-based protocols. This section reviews the foundational work in each of these areas and discusses how they interrelate for applications such as traffic prediction and control.

2.1 Wireless Sensor Networks for Traffic Management

Wireless sensor networks serve as the physical backbone of many smart transportation systems by collecting real-time environmental and traffic data. WSNs are typically composed of a large number of low-power nodes with limited computation and communication capabilities (Akyildiz, Su, Sankarasubramaniam, & Cayirci, 2002). Their constrained nature has prompted research into lightweight data processing and communication protocols that are essential for deploying distributed intelligence in traffic management. Researchers have shown that by optimizing data collection and processing in WSNs, it is possible to reduce energy consumption and improve the accuracy of traffic flow predictions (Lv, Duan, Kang, Li, & Wang, 2015).

2.2. Federated Learning in Distributed Systems

Federated learning has emerged as an effective approach for training machine learning models on distributed data without the need to centralize raw data (McMahan, Moore, Ramage, Hampson, & y Arcas, 2017). In FL, individual nodes (or sensors) update local models which are then aggregated to form a global model. This paradigm is particularly relevant for applications in WSNs and intelligent transportation, where preserving data privacy and reducing communication overhead are critical. By enabling local training on traffic data, FL allows for adaptive and region-specific traffic predictions while safeguarding sensitive information (Konečný, McMahan, Yu, Richtárik, Suresh, & Bacon, 2016).

2.3. Gossip Protocols for Decentralized Aggregation

Gossip-based protocols provide a decentralized alternative to the classical client–server model in federated learning. In gossip learning, nodes periodically exchange model updates with randomly selected neighbors, which not only distributes the communication load but also improves network robustness by eliminating a single point of failure (Kempe, Dobra, & Gehrke, 2003). The random peer-to-peer exchange mechanism has been demonstrated to be effective at achieving consensus on model parameters across the network, even in highly dynamic environments typical of vehicular networks. Recent studies suggest that gossip learning can match or even outperform centralized federated approaches under uniform data distribution conditions (Hegedűs, Danner, & Jelasity, 2013).

2.4. Traffic Prediction and Control Using Distributed Learning

The integration of FL and gossip-based protocols is particularly attractive for traffic management applications. Traffic prediction models benefit from the localized learning capabilities of FL, as they can capture regional variations in traffic patterns more accurately. When coupled with the robust and scalable communication provided by gossip protocols, the system can dynamically adjust traffic control measures in response to real-time events (Lv et al., 2015; McMahan et al., 2017). Several studies have explored these ideas, underscoring the importance of leveraging decentralized learning paradigms to improve both predictive accuracy and operational efficiency in intelligent transportation systems.

3. Scalability and Real-World Traffic Data

In developing FLARE, an integral goal was to ensure that the system can scale effectively to cope with complex, large-scale traffic networks while remaining robust in real-world deployments. This section outlines the methods used to achieve scalability and describes how real-world traffic data are integrated into the simulation environment to validate the system's performance.

3.1. Scalability in FLARE

FLARE leverages a decentralized federated learning architecture to manage traffic signal control without the need for a centralized aggregation server. By distributing both data processing and model training across numerous edge nodes (e.g., traffic sensors or roadside units), the system minimizes communication overhead and reduces latency. The use of lightweight Python scripts for data preprocessing and model training further ensures that each node can operate efficiently even on constrained hardware.

Key scalability features include, decentralized model aggregation where Nodes use peer-to-peer model updates via the federated averaging (FedAvg) algorithm. This approach improves fault tolerance and eliminates the single-point-of-failure risk inherent in centralized systems (McMahan et al., 2017). Efficient data flow optimization which includes optimization of Python scripts has ensured seamless CSV file reading and preprocessing, enabling quick integration with traffic model updates. This optimization is critical when scaling up to networks with hundreds or thousands of nodes. Along with dynamic clustering for load balancing. FLARE can dynamically group nodes based on geographical proximity or data similarity. Such clustering not only balances computation and communication but also reduces the time required for model convergence—further demonstrating the system's scalability in large urban environments.

3.2. Integration of Real-World Traffic Data

To ensure that FLARE's performance remains robust under real-world conditions, real traffic data from metropolitan areas are used to simulate traffic patterns accurately. In our project, we integrate data from SUMO (Simulation of Urban Mobility), which provides detailed, micro-level traffic simulations based on actual road networks and vehicle trajectories.

Key aspects of real-world data integration include, data-driven simulation. This is done by incorporating historical and real-time traffic data (e.g., vehicle counts, flow rates, and signal timings), FLARE can adapt to varying traffic conditions. This ensures that the model is trained on accurate representations of urban traffic, thereby enhancing the reliability of the control logic in live environments. Validation with actual traffic patterns. Traffic models are benchmarked against real datasets (such as those available from Victoria, BC, or similar smart city initiatives). This comparison validates that the distributed learning approach can capture the nuances of traffic congestion and signal adjustments, allowing for improvements in delay reduction and throughput optimization. Iterative Performance Evaluation was also included. This means that the system is tested across multiple simulation rounds where local nodes continuously update models with real-world traffic measurements. Metrics such as convergence speed, accuracy of traffic flow predictions, and improvements in signal timing are monitored to evaluate FLARE's effectiveness.

The combined approach of advanced scalability and real-world data integration ensures that FLARE not only performs efficiently in controlled simulation environments but also proves its viability for real-time adaptive traffic management in increasingly complex urban settings.

Include diagrams of:

- QuadMesh topology
- Simulation flow between SUMO, Python scripts, and OMNeT++

4. Optimizing Traffic Control Logic

In FLARE, the fundamental goal of improving traffic management is realized by refining the control logic at intersections. This control logic is rooted in a decentralized federated learning (FL) model where traffic sensor nodes continuously train local predictive models and exchange updates through a peer-to-peer (P2P) protocol. The OMNeT++ simulation framework plays a central role in emulating this distributed environment. Each node in the simulation, implemented as an instance of a C++ module (GossipNode), not only performs local model training but also communicates with its neighbors by broadcasting model updates.

At simulation startup, nodes are instantiated according to specifications in the network description (.ned) file. The .ned file statically deploys 16 nodes arranged into four clusters (or circles) with predefined connections. Each node's behavior is encapsulated in the GossipNode module, which is defined in C++ (see the provided .cc file). Upon initialization, each node reads its configuration from a CSV file, thereby being assigned a specific location and a cluster of related nodes. This information is then used to update the graphical display of the node within the simulator. Moreover, the nodes initialize a set of model weights randomly and schedule a periodic training event, where the duration between events is chosen randomly to mimic realistic, asynchronous behavior.

The simulation implementation includes an embedded Python interpreter, which enables a cross-language integration between C++ and Python. This integration supports advanced model evaluation tasks, such as running a linear regression script that determines the accuracy of the current model weights. The following pseudocode summarizes the operational sequence of an individual node, capturing the essential procedures for initialization, local training, and communication:

```
Procedure NodeInitialize
    Begin
        // Initialize simulation and schedule first training event
        If node index equals 0 then
            Schedule gossip message at time 0.0
            Initialize Python interpreter and set up module search path
        End if
        // Initialize local training event with a random delay
        Schedule training event (TrainEvent) at simTime + random value
        // Initialize model weights with random values (e.g., between 0.5
and 1.5)
        Read CSV file to assign location and cluster membership based on
node index
        Update node display string with location and cluster details
    End
```

```
Procedure HandleMessage(message)
    Begin
        If message is TrainEvent then
            Invoke PerformLocalTraining
            Reschedule TrainEvent with a new random delay
        Else if message is ModelUpdate then
            Extract received weights
            Merge received weights with local weights by averaging
        End if
        Delete the processed message
    End

Procedure PerformLocalTraining
    Begin
        // Simulate local training by perturbing each model weight slightly
        For each weight in modelWeights:
            Update weight by adding a small random value within a defined
range
        End for
        Compute maximum change in weights to assess convergence; log
convergence if threshold met
        Evaluate model accuracy using a Python script via embedded
interpreter
        Create a ModelUpdate message and populate it with the updated
weight vector
        Record the simulation time before sending the message
        Broadcast the ModelUpdate message to all connected output gates
        Log the message transmission and estimate latency for diagnostics
    End
```

The actual code in the GossipNode module implements these procedures. In the `initialize()` function, for instance, the node calls `assignLocationAndCluster()`, which opens the CSV file and assigns relevant properties based on the node's index. The `handleMessage()` function distinguishes between training events and incoming ModelUpdate messages; when a ModelUpdate is received, the current model weights are averaged with the received values, contributing to a distributed learning process analogous to federated averaging (FedAvg) as described by McMahan et al. (2017).

The .ned file defines the network topology for the simulation. Nodes are interconnected according to a mesh-like pattern with delays assigned via a custom channel (set to 100 ms in the simulation) and are organized into distinct groups. Such a topology ensures that model updates propagate among nodes in a manner that closely mirrors real-world wireless sensor networks. Notably, the inter-cluster links (or "gateway" connections) facilitate exchange between groups, which is critical for synchronizing the distributed learning process across diverse geographic areas.

The code snippet below (extracted from the .cc file) illustrates the core functionality of local training and broadcasting within a node:

```
void GossipNode::performLocalTraining() {
    EV << "Node[" << getIndex() << "] performing local training\n";

    // Save previous weights for convergence checking
    std::vector<double> previousWeights = modelWeights;
    for (auto& weight : modelWeights) {
        weight += uniform(-0.1, 0.1);
    }

    double maxWeightChange = 0.0;
    for (size_t i = 0; i < modelWeights.size(); i++) {
        double weightChange = fabs(modelWeights[i] - previousWeights[i]);
        maxWeightChange = std::max(maxWeightChange, weightChange);
    }

    if (maxWeightChange < 0.01) {
        EV << "Node[" << getIndex() << "] model has converged! Max weight
change: " << maxWeightChange << "\n";
    }

    // Evaluate accuracy via Python integration
    double accuracy = evaluateModel(modelWeights);
    EV << "Node[" << getIndex() << "] accuracy: " << accuracy << "\n";

    // Create and broadcast model update
    ModelUpdate *update = new ModelUpdate("ModelUpdate");
    update->setSenderId(getIndex());
    update->setWeightsArraySize(modelWeights.size());
    for (size_t i = 0; i < modelWeights.size(); ++i) {
        update->setWeights(i, modelWeights[i]);
    }
    forwardModelUpdate(update);
}
```

In summary, the fusion of OMNeT++ for network simulation and embedded Python for model evaluation forms the backbone of FLARE's approach to optimizing traffic control logic. The nodes are created via the OMNeT++ network description, and each node autonomously performs local training and communicates model updates. These interactions, captured through the provided pseudocode and code excerpts, illustrate the operational flow that underpins FLARE's distributed and adaptive control strategy. This methodology not only ensures that the system scales to large, real-world traffic networks but also lays a solid foundation for the continuous refinement of traffic control at intersections.

5. Description of Simulation Setup

The simulation involves a wireless sensor network (WSN) where multiple nodes are located at various intersection points in a city. Each node represents a traffic monitoring unit, and the network is used to simulate the communication between nodes based on the federated learning protocol using gossip messages. The simulation is executed on a mesh network topology, where nodes communicate locally and share their model updates with each other.

Nodes are initialized with different model weights, corresponding to their local traffic data. A custom gossip-based protocol is used to propagate model updates between nodes. Each node performs local training, and its accuracy is reported during each round. The simulation includes model update messages, training events, and the exchange of models to simulate the federated learning process. The goal is to evaluate the performance of this protocol over multiple communication rounds.

Metrics Used:

- **Accuracy:** The percentage of correct predictions made by the node's local model.
- **Precision, Recall, F1:** These metrics are used to evaluate the classification performance of each model at every round.
- **Message Latency:** The time taken for a message (e.g., model update, training event) to be transmitted between nodes in the network.

Results (performance over rounds):

1. **Accuracy Evolution:**
    - Node accuracy fluctuates across the simulation rounds as each node trains on local data and receives updates from other nodes. For example, Node[5] started with an accuracy of 30.33%, Node[8] at 43.47%, Node[3] at 48.17%, and Node[2] at 38.96%. Over time, the nodes update their models, and the accuracy improves, reflecting the effectiveness of federated learning in aggregating local models.
2. **Model Convergence:**

    - Model convergence can be observed in the way accuracy and other metrics evolve with time. The nodes gradually adjust their models as they exchange updates. For example, Node[10] showed an accuracy of 43.86% at time t=1.529, which slightly improved as it received updates from other nodes.

3. **Communication Costs:**
    - The communication costs can be assessed through the time taken for model updates to propagate through the network. Messages were exchanged frequently between nodes. For example, Node[4] received models from Node[5], Node[8] received models from Node[10], and so on.

Interpretation of Results:

The results show that nodes, despite having initial local models with varying accuracies, progressively improve their models through federated learning and gossip protocols. As nodes exchange models, they achieve better consensus on the overall traffic pattern, which is reflected in the accuracy improvements. The federated learning mechanism allows the nodes to learn collaboratively while keeping their data private.

Additionally, communication overhead (message latency) seems to be a key factor in the synchronization of the model across nodes. As the simulation progresses, the propagation of updates between nodes becomes more frequent, leading to potential synchronization and latency issues.

Observations on Synchronization and Latency Issues:

- **Synchronization Issues:** The synchronization between nodes can be challenging when there is a delay in receiving model updates. For instance, at certain points in the simulation, Node[0] received updates from Node[1] after a delay, which indicates that some nodes were not fully synchronized in terms of training.

- **Latency Issues:** Latency is observed as a factor impacting the speed of model updates. For example, message delays were observed between nodes such as Node[0] and Node[10], affecting the timely sharing of model information. This delay could hinder the speed of model convergence and performance improvement.

6. Challenges and Limitations

The development of FLARE, while demonstrating an innovative decentralized approach to traffic control, has surfaced several challenges that underscore the complexity of deploying federated learning in dynamic vehicular environments. One of the central challenges lies in the inherent tension between achieving rapid model convergence and managing communication overhead across distributed nodes. In our current simulation framework, nodes exchange model parameters through a simple averaging mechanism, which, although effective in theory, can suffer from delays and potential inconsistencies when extended to larger, real-world networks. The assumption of uniform communication delays and ideal network conditions does not always hold in practice, and latency variations may lead to asynchronous updates that undermine the stability of the global model.

Another significant limitation arises from the integration of heterogeneous computing environments within the simulation. FLARE relies on a combination of C++ modules implemented in OMNeT++ and embedded Python scripts for model evaluation, a cross-language approach that introduces complexity and potential performance bottlenecks. The overhead associated with initializing and interfacing between the C++ simulation environment and the Python interpreter may obscure the genuine behavior of local model training, thereby affecting the accuracy of convergence assessments. This challenge is compounded by the preliminary nature of our local training routine, which uses a rudimentary weight perturbation strategy rather than more advanced learning algorithms that could better capture traffic dynamics.

The use of static CSV files for node configuration, while sufficient for initial experiments, imposes additional constraints on the scalability and adaptability of FLARE. In dynamic real-world scenarios, the spatial and clustering information of nodes is likely to evolve with changes in traffic patterns and sensor deployments. This static configuration model restricts the system's ability to reconfigure itself in real time, potentially leading to suboptimal performance when confronted with shifting network conditions and heterogeneous traffic data.

Moreover, the simulation environment itself, despite its controlled setting, does not fully encompass the variability and unpredictability encountered in real urban traffic systems. Factors such as environmental influences, unpredictable driver behavior, and sporadic sensor failures are difficult to model accurately in simulation. These limitations suggest that while FLARE establishes a promising proof-of-concept, further research is essential to refine local training schemes, develop more robust aggregation algorithms, and incorporate richer real-world data to evaluate system performance under realistic conditions.

In summary, the challenges identified in this work—including communication latency, integration complexity, static configuration, and the limitations of simulated data—highlight critical areas for future investigation. Addressing these issues will be crucial in advancing FLARE from a conceptual model to a fully functional and resilient traffic management system capable of operating effectively in complex, real-world settings.

7. Future Work and Improvements

While FLARE establishes a novel framework that integrates federated learning with decentralized control for urban traffic management, several avenues remain for further exploration and refinement. One promising direction involves enhancing the sophistication of the local training algorithms. The current model employs a relatively simple weight perturbation strategy to simulate local learning; future research should incorporate state-of-the-art machine learning techniques, such as more refined gradient descent variants or even specialized deep learning architectures optimized for time series prediction. Integrating adaptive learning rates and momentum-based updates could not only improve convergence speed but also help in capturing the complex temporal variations seen in real traffic data. This focus on advanced local training is expected to mitigate issues related to non-convergence and model drift, particularly in a heterogeneous sensor network (McMahan et al., 2017; Konečný et al., 2016).

Another critical aspect lies in the development of more robust aggregation methods. At present, FLARE relies on a simple averaging process to merge model updates from different nodes. However, more complex aggregation techniques, such as weighted averaging that accounts for node reliability or variance reduction methods, could significantly increase the robustness of the global model. It is also worth investigating approaches that incorporate outlier detection during aggregation, since sporadic communication failures or adversarial updates might skew the collective learning process. Techniques from robust statistics or consensus algorithms could be adapted to this decentralized setting (Kempe, Dobra, & Gehrke, 2003).

The dynamic nature of real-world urban traffic further motivates the need for adaptive network reconfiguration. Currently, the node topology and cluster memberships are derived from a static CSV configuration, which limits the system's ability to respond to changes in sensor deployments or evolving traffic patterns. Future work could focus on implementing dynamic clustering mechanisms where nodes periodically update their cluster memberships based on real-time traffic data and communication quality metrics. Incorporating such adaptive behavior would allow FLARE to better maintain optimal connectivity and to respond more effectively to events such as road closures or accidents.

Moreover, the integration of real-world traffic data remains an important challenge. While simulation environments such as OMNeT++ and SUMO offer a controlled setting for model development, experimental validation on data from actual traffic sensors would provide critical insights into the performance and limitations of FLARE. Future research should aim to establish partnerships with municipal traffic authorities or leverage publicly available datasets to test the system under diverse and unpredictable traffic conditions. This real-world testing would help in fine-tuning various parameters such as communication delay, model convergence thresholds, and robustness to network congestion.

Finally, the current approach to privacy and security within FLARE, though promising, can be further strengthened. Advances in differential privacy and secure aggregation protocols should be explored to provide stronger privacy guarantees without incurring significant computational overhead. In particular, research into lightweight encryption schemes suitable for resource-constrained sensor nodes could be instrumental in protecting sensitive traffic data. Similarly, investigating techniques that are resilient to adversarial attacks—especially in scenarios where communication is intermittent—would help ensure that FLARE operates securely even in challenging network environments.

While FLARE demonstrates the feasibility of applying federated learning to traffic management, the future work outlined above highlights crucial areas for improvement. Enhancements in local training sophistication, robust aggregation and dynamic clustering, integration of real-world data, and fortified privacy measures are expected to propel FLARE toward practical, scalable, and secure deployment in modern intelligent transportation systems.

8. Conclusion

In this paper, we introduced FLARE, a novel federated learning framework tailored to enhance urban traffic control through decentralized processing. By leveraging a distributed architecture based on wireless sensor networks and integrating peer-to-peer model updates within an OMNeT++ simulation environment, FLARE demonstrates a promising approach to real-time traffic optimization that preserves data privacy while mitigating communication bottlenecks.

The system design is characterized by its ability to autonomously initialize sensor nodes with location and cluster-specific parameters drawn from a configuration file, perform local training via lightweight Python-augmented routines, and exchange model updates using a simple yet effective averaging strategy. This decentralized aggregation method, influenced by established federated learning paradigms (McMahan et al., 2017; Konečný et al., 2016), affords FLARE the dual benefits of scalability and robustness by eliminating single points of failure and reducing the overall communication overhead.

Our investigation revealed several challenges, notably in achieving rapid convergence under variable network conditions, managing the complexities inherent in a cross-language integration between C++ and Python, and accommodating the static configurations derived from CSV files. These limitations, however, serve as catalysts for future research that aims to refine local training algorithms, develop more robust aggregation methods, and introduce dynamic network reconfiguration strategies. Moreover, the preliminary experimental results highlight the critical need for real-world data integration, as simulation environments, while useful, may not capture the full complexity of urban traffic dynamics.

Looking forward, enhancing FLARE with more advanced learning techniques and stronger privacy-preserving protocols will be essential in bridging the gap between simulation-based studies and practical deployments. As intelligent transportation systems evolve and the demand for decentralized, real-time control increases, FLARE offers a scalable foundation upon which future improvements can be realized. Ultimately, this work contributes to a broader understanding of how federated learning can be effectively applied in urban traffic management, setting the stage for subsequent studies that will refine and extend these methodologies to meet the rigorous demands of modern cities.

References

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks, 38*(4), 393–422. https://www.sciencedirect.com/science/article/abs/pii/S1389128601003024

Hegedűs, I., Danner, G., & Jelasity, M. (2013). Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience, 25*(4), 556–571. https://arxiv.org/abs/1109.1396

Kempe, D., Dobra, A., & Gehrke, J. (2003). Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science* (pp. 482–491). IEEE. https://www.cs.cornell.edu/johannes/papers/2003/focs2003-gossip.pdf

Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*. https://arxiv.org/abs/1610.05492

Lv, Y., Duan, Y., Kang, W., Li, Z., & Wang, F.-Y. (2015). Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems, 16*(2), 865–873. https://ieeexplore.ieee.org/document/6894591

McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (pp. 1273–1282). PMLR. https://arxiv.org/abs/1602.05629