End-to-end Privacy Preserving Training and Inference for Air Pollution Forecasting with Data from Rival Fleets

Gauri Gupta
MIT
gaurii@mit.edu

Krithika Ramesh
Johns Hopkins University
kramesh3@jh.edu

Anwesh Bhattacharya
Microsoft Research India
t-anweshb@microsoft.com

Divya Gupta
Microsoft Research India
divya.gupta@microsoft.com

Rahul Sharma
Microsoft Research India
rahsha@microsoft.com

Nishanth Chandran
Microsoft Research India
nichandr@microsoft.com

Rijurekha Sen
Indian Institute of Technology Delhi
riju@cse.iitd.ac.in

ABSTRACT

Privacy-preserving machine learning (PPML) promises to train machine learning (ML) models by combining data spread across multiple data silos. Theoretically, secure multiparty computation (MPC) allows multiple data owners to train models on their joint data without revealing the data to each other. However, the prior implementations of this secure training using MPC have three limitations: they have only been evaluated on CNNs, and LSTMs have been ignored; fixed point approximations have affected training accuracies compared to training in floating point; and due to significant latency overheads of secure training via MPC, its relevance for practical tasks with streaming data remains unclear.

The motivation of this work is to report our experience of addressing the practical problem of secure training and inference of models for urban sensing problems, e.g., traffic congestion estimation, or air pollution monitoring in large cities, where data can be contributed by rival fleet companies while balancing the privacy-accuracy trade-offs using MPC-based techniques.

Our first contribution is to design a custom ML model for this task that can be efficiently trained with MPC within a desirable latency. In particular, we design a GCN-LSTM and securely train it on time-series sensor data for accurate forecasting, within 7 minutes per epoch. As our second contribution, we build an end-to-end system of private training and inference that provably matches the training accuracy of cleartext ML training. This work is the first to securely train a model with LSTM cells. Third, this trained model is kept secret-shared between the fleet companies and allows clients to make sensitive queries to this model while carefully handling potentially invalid queries. Our custom protocols allow clients to query predictions from privately trained models in milliseconds, all the while maintaining accuracy and cryptographic security.

KEYWORDS

pollution forecasting, secure MPC, Privacy Preserving ML

Proceedings on Privacy Enhancing Technologies 2023(4), 436–451

https://doi.org/10.56553/popets-2023-0118

1 INTRODUCTION

Urban sensing applications like air pollution sensing and forecasting [3, 24, 39], road surface quality monitoring [17, 44, 57], traffic congestion and travel time information collection [30, 68], have been shown to scale using vehicles instrumented with sensing platforms. Higher coverage and more data from diverse sources can play a vital role in improving machine learning (ML) accuracy [18, 34, 64]. However, fleet companies have a real privacy concern before taking part in vehicle-mounted sensing projects, that of sharing location trajectories and fleet size information among rivals. For example, cab-sharing companies like Uber and Lyft can instrument their vehicles with pollution sensors. But training a joint model to forecast air pollution combining their datasets, is privacy-invasive. Knowing where Uber cabs are, Lyft can optimize cab routes for higher profits, affecting Uber's business. Their sensed data will remain in silos unless privacy-preserving ML training and inference methods are devised to process the datasets.

Problem Setting. We focus on the same important use-case of air pollution monitoring as in [1]. Spatio-temporal *particulate matter* (PM) data sensed by each vehicle has to be aggregated across city locations and times of the day, and then fed into ML models for different applications. One useful application is *pollution forecasting* i.e. answering queries about PM values at some future time point, based on historical and current values. Prior work [1] limitedly focussed on one fleet company, which processed its own vehicles' sensor data with ML models in its own server. Privacy played a role in only answering statistical queries from clients using that processed data. In this work, we focus on securely combining data from multiple (rival) fleet companies to train more accurate ML models, while hiding the location of individual sensors or preserving the local privacy of individual data points.

As pollution is a dynamically evolving phenomenon, a pre-trained model is not sufficient to effectively infer the pollution values after a long duration. Hence we need to periodically fine-tune the model with fresh incoming data. This model fine-tuning should also preserve privacy at low latency, to handle continuous streaming data. Finally, (potentially invalid) queries from clients need to be answered using the most recent fine-tuned ML model, without revealing model details to the client and client location to any

server or data owner. The two problems have been well-studied separately in Privacy Preserving Machine Learning (PPML) literature, as secure training [33, 37, 45, 63] and secure inference [31, 45, 55].

Shortcomings of prior works in PPML. SecureML [45] enabled mutually distrusting parties to jointly train DNN models on their combined sensitive data without revealing the data to any of the parties. Since its introduction, there have been designs of specialized MPC protocols for this problem [33, 37, 59, 60, 63] as well as better programming support [33, 37]. However, the current state-of-the-art in secure training suffers from three main limitations that hinder its direct application to the problem of secure training of models for air pollution forecasting. First, it only focuses on simple convolutional neural networks (CNNs). Our setting deals with time-series data and recurrent neural networks (RNNs), e.g., LSTMs, which are standard model architectures used in this domain. None of the secure training protocols or frameworks support RNNs [54].

Second, all of the existing works use ad-hoc fixed-point approximations for ML training algorithms that can potentially lose accuracy. For instance, Piranha [63], the latest work on secure training, reports that secure training of AlexNet on CIFAR suffers from 19% accuracy loss compared to cleartext training.

Third, secure training via MPC has significant latency overheads, making it challenging to process streaming data like that of air pollution.

Our approach and its generalizability. To handle the shortcomings in prior PPML papers, we adopt a co-design approach between ML modeling and secure multi-party computation (MPC). At every step of our end-to-end system design, we carefully balance trade-offs between ML accuracy vs. MPC computation and communication overheads. Using real vehicle-mounted 3-month PM datasets [13] from one of the most polluted cities in the world, we show Graph Convolutional Network (GCN) architectures capture well the spatial relations of particulate matter (PM) and Long Short Term Memory (LSTM) cells capture temporal information for forecasting. To address the shortcomings of existing frameworks for secure training, especially to *provably guarantee* the absence of accuracy loss between secure training and cleartext training, we use the state-of-the-art implementations of MPC protocols for floating-point arithmetic, SecFloat [53], that implements floating-point operations like addition, multiplication, exponentiation, etc., with formal correctness guarantees. In particular, SecFloat is up to two orders of magnitude more efficient than works in secure floating-point that predate it. Implementation of ML training in SecFloat defines our cost model, that is MPC computation and communication overheads.

Our carefully co-designed GCN-LSTM architecture-based ML model and SecFloat based MPC protocol allow model training in floating point, maintaining the accuracy of plain-text training while achieving cryptographic guarantees for the privacy of training data. We also achieve practical runtime to support model fine-tuning with streaming data (7 minutes per training epoch). Clients get query responses in negligible time (less than 5 milliseconds). Our paper uses air pollution forecasting as a use-case, so that the end-to-end system design, implementation, and evaluation of accuracy and runtimes can be demonstrated using real datasets. However, the neural network architectures we use for pollution forecasting are highly generic and useful for many applications. For example, GCN has been used for text classification [66], image classification [11], recommendation systems [41, 61]; LSTM for keyword spotting [8, 10] and gesture detection and human activity recognition [21, 69]; and GCN-LSTM for travel time estimation [32, 70]. If in any such application, data is collected by more than one party which cannot share the data for training or inference in plain text, our ML-crypto co-design principles and MPC protocols will be useful.

1.1 Main contributions

Our **first contribution** is to design an ML model for air pollution forecasting that can be trained using protocols provided by SecFloat with an acceptable end-to-end latency. PM depends on green cover, industrial activities, traffic volumes, and other spatial phenomena, as well as temperature variations, wind speed, traffic densities, and other temporal phenomena. How should the city be split into location grids, to capture the spatial dependencies between locations and PM? How many buckets should the time of the day be split into, to capture temporal factors affecting PM? What parameters of the NN models (e.g. number of convolution layers if used, number of memory cells if used, filter sizes, etc.) will balance ML accuracy with MPC-based training/fine-tuning overheads? We answer these questions empirically in Section 4, using one of the richest spatio-temporal datasets for PM 2.5 and PM 10 collected using sensor-instrumented public buses in Delhi [13].

Next, as a **second contribution**, we build an end-to-end pipeline to securely train our models using SecFloat. SecFloat provides support for basic floating-point operations. One can write SecFloat code for LSTMs that works on one scalar value at a time, but that would be extremely inefficient. We show how to use SecFloat APIs to build ML training APIs with forward pass and backpropagation, for convolution, memory cells and other neural network operations, efficiently. We in-fact have observed while training a GCN model with SecFloat, and another fixed-point based industrial MPC framework, Crypten [37], that floating-point MPC training converges the same way as plaintext model training in Keras, while Crypten never converges. Recall that Crypten, like all prior secure training implementations, doesn't support LSTM. Our ML training secure APIs can be more directly used for applications that use either convolution or LSTM or both. Our solution works with 2 rival data providers and can be generalized to n-data providers in the client-server model using standard techniques [45, 48, 49].

After completion of secure training, the two servers hold secret shares of the trained/fine-tuned model. Our **third contribution** is efficient querying of this secret shared model. The main focus of prior work [1] was to support such queries in the setting of a single server; the client and the server engage in secure 2-party computation that requires the client to communicate MBs of data and results in one-second latency per query. We make two observations. First, even though we perform training using a secure floating-point, the query processing can work over integer/fixed-point arithmetic. This is particularly true for our setting because the PM values are integers between 1-1000 and we only need to compute some simple statistics over these. To enable private query processing over integers, we provide a novel 2-party protocol to convert from secret shared floating-point values to secret shared integers. These protocols act as glue between floating-point training

using SecFloat and integer query processing using EMP's garbled circuits [62]; enabling such private queries over integer values can reduce overheads by an order of magnitude. Second, we leverage the two servers to do the heavy lifting and allow a client to be lightweight. In particular, the client gets a response in milliseconds, i.e., more than two orders of magnitude improvement over [1].

In summary, this is a novel experience paper on end-to-end PPML solution for practical real-time training and inference of air pollution forecasting models. Our main technical contributions are interdisciplinary, at the intersection of ML and MPC. The research challenges arise from balancing ML accuracy vs. MPC latencies, which we resolve through co-design of neural network architecture and MPC protocols. This successful demonstration of privacy-preserving ML model training on streaming data and query responses in milliseconds, shows great promise for all urban sensing problems. Any application using convolution or LSTM models, that need secure training among non-trusting data-owners, can benefit from our ML-MPC co-design approach and SecFloat based secure APIs for ML.

1.2 Other Related Works

Secure MPC for location privacy: Secure Multi-Party Computation (MPC) has been extensively used to preserve location privacy, viz. detecting proximate users without revealing individual locations [14, 56], detecting users with overlapping trajectories without revealing individual trajectories in ride sharing [22, 28, 51]. These works have only location sensor values and do not have utility concerns, unlike our use case with pollution sensor values, where the accuracies of forecasting results are important. Works like [4, 9, 12, 29, 46] specifically explore MPC for mobile applications with different optimizations like an extra server for outsourcing garbled operations or using a dedicated compiler.

Alternate Approaches for Privacy. The use of approaches such as differential privacy [15] and federated learning [43] results in loss of either accuracy of the task, aka, utility, and/or end-to-end formal security guarantees, and hence, are unattractive for this task. Systems researchers are trying hard to build accurate low cost PM and GPS sensing devices. ML practitioners are designing DNN models to use the sensed information for the most accurate forecasting possible. As privacy researchers, we cannot negate their efforts by adding noise to the PM or GPS sensor data values or the ML model weights, as done in DP or FL. More details are below.

Differential Privacy: A potential alternate approach towards enabling the training of joint models in a privacy-preserving manner is that of Differential Privacy (DP) [15]. Several works have explored adding noise to the location coordinates [2, 5, 23, 27, 40], or adding dummy trajectories to actual trajectories [35, 47], or using chaff vehicles for variable traffic density [58] in order to not reveal raw sensor data. One could potentially do something similar – 1) add DP noise to the raw sensor data before the ML training process 2) train an ML model to obtain a pollution level grid map of the city; and finally 3) Add DP noise to the entries before releasing it publicly thus enabling client's to perform queries without revealing their location to the servers. However, an MPC-based approach to the problem has several important advantages: 1) As already empirically demonstrated in [1], the amount of DP noise to be added for privacy preservation undermines the utility of the pollution sensing system. 2) Even this would only provide privacy for the individual data points in the data set and not about the entire data set provided by each data provider. In contrast, an MPC-based solution allows us to train over the raw trajectory data with associated PM values as if all the data were present in the clear in one place. Hence, no change is needed in the ML training algorithm, and hence no accuracy is lost in the training process. Further, our end-to-end MPC provides a much stronger privacy guarantee. The data contributors learn no information through the process – in particular, the final trained model is also kept hidden (or secret shared amongst the servers). Finally, the clients learn the query results alone and nothing else about sensor data, and not even noisy grids.

Federated Learning: Federated learning (FL) [43] is another alternative approach to training joint models from multiple data sources. While being attractive from a data movement point of view, it suffers from serious limitations on the privacy front [16], especially when the number of data providers is small (FL by itself does not provide any privacy guarantees and can be combined with DP techniques and/or secure aggregation to provide some level of privacy). Further, it is significantly inferior to centralized training on the accuracy front, when the data present with the different sources is heterogeneous [71].

2 CRYPTOGRAPHY BACKGROUND

2.1 Secret Sharing Schemes

At a high level, 2-out-of-2 secret sharing schemes allow a secret x to be split into two values x_0 and x_1 with the property that each x_b ($b \in \{0,1\}$) contains no information about x, while x_0 and x_1 can be combined through a reconstruction procedure (typically addition over an appropriate ring) to recover x. We will make use of 2 types of 2-out-of-2 secret sharing schemes to share integer values (over different rings) and to share floating-point values. To share a secret ℓ–bit integer $\langle x \rangle^\ell$, viewed as an element $\in \mathbb{Z}_{2^\ell}$, sample 2 random ring elements $\langle x \rangle_0^\ell, \langle x \rangle_1^\ell$ from $\mathbb{Z}_{2^\ell}$ such that $\langle x \rangle_0^\ell + \langle x \rangle_1^\ell = \langle x \rangle^\ell$, where + denotes addition over the ring $\mathbb{Z}_{2^\ell}$ (i.e. $\bmod 2^\ell$). The share of party S_b ($b \in \{0,1\}$) is denoted by $\langle x \rangle_0^\ell$. When $\ell = 1$, we sometimes use the superscript B, instead of ℓ. Following SecFloat [53], a floating-point number x is represented as a 4-tuple (z, s, e, m) has has value $(1-z) \cdot (-1)^s \cdot \frac{2^e \cdot m}{2^q}$. Here, s is the sign bit, e is the p-bit exponent, z is the zero indicator bit equivalent to $e = -2^{p-1} + 1$, m is the fixed-point mantissa in $q+1$ bits and scale q, and the division is over reals. A secret shared floating-point number x is then a tuple $\langle x \rangle^{\mathsf{FP}} = (\langle z \rangle^B, \langle s \rangle^B, \langle e \rangle^{p+2}, \langle m \rangle^{q+1})$ of 4 secret-shared values. For standard 32-bit floating-point, $p = 8$ and $q = 23$.

2.2 Secure 2PC

Secure 2-party computation (2PC) allows two parties S_0 and S_1, each holding private inputs x_0 and x_1 respectively, to compute any arbitrary function $y = f(x_0, x_1)$ on these inputs while revealing only the function output y to either party. This is done through a protocol execution π. We consider a static probabilistic polynomial time semi-honest adversary that corrupts one of the parties. That is,

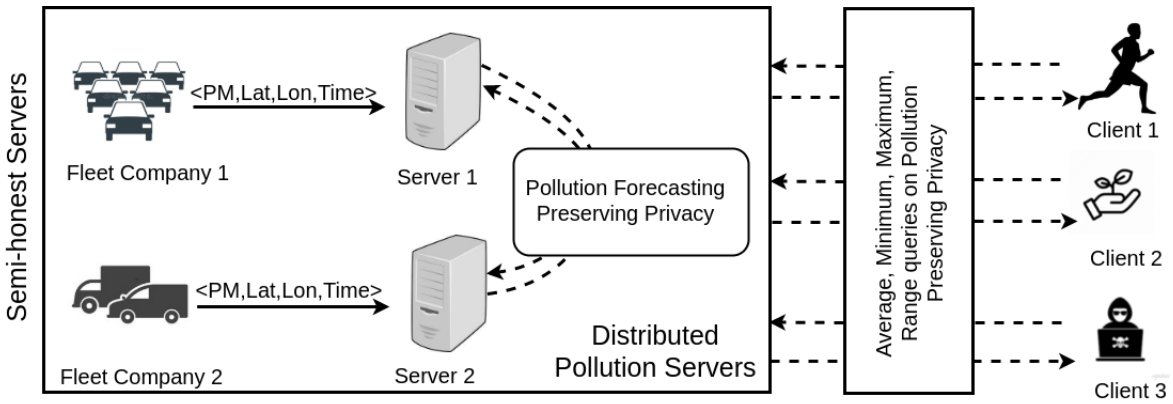


Figure 1: *System architecture with stake-holders and threat model. Solid lines denote clear-text computation and communication between servers and their respective fleets. Dotted lines indicate MPC among servers and between servers and clients.*

at the beginning of the protocol, an adversary $\mathcal{A}$ can corrupt either S_0 or S_1. While $\mathcal{A}$ will try to learn information about the other party's input, it will follow the protocol specification faithfully. Proof of security of π is shown via the ideal/real paradigm [42] that considers two interactions: a real interaction in which S_0 and S_1 interact with each other in the presence of an adversary $\mathcal{A}$ (corrupting S_b) and the environment $\mathcal{Z}$; and an ideal interaction, in which parties send their inputs to a trusted functionality that computes the function faithfully. Security is proved by showing the existence of an ideal adversary $\mathcal{S}$ (which does not know S_{1-b}'s input), known as the simulator, such that no $\mathcal{Z}$ can distinguish between the two interactions.

Various 2PC protocols follow the blueprint of both participants starting with secret shares of inputs of a function/computation and running a 2PC protocol to end with secret shares of outputs. The work of SecFloat [53] provides such secure protocols for various floating-point functions such as addition, multiplication, division and exponentiation. These protocols compose while preserving security and we use them to implement the operations occurring in our ML workloads.

3 SYSTEM OVERVIEW

3.1 Stakeholders, Threat Model, Problem Setting

Using vehicles instrumented with low-cost Particulate Matter (PM) sensors, fleet companies collect air pollution data in the format ⟨PM, Lat, Lon, T⟩, where PM denotes the Particulate Matter value sensed by an instrumented vehicle at some location given by latitude Lat and longitude Lon at a particular time T. These raw traces coming from individual instrumented vehicles are aggregated by their corresponding fleet companies at their respective servers as a grid with desired granularity as explained below. The servers want to train ML models for interesting tasks such as air pollution forecasting. Instead of using only the data streamed at their own server, combining data across multiple servers makes the dataset richer, with better spatiotemporal coverage and granularity, and subsequently having higher accuracy in answering the ML questions. However, the servers cannot share their collected data in the clear as that would reveal business-sensitive information, e.g., spatiotemporal fleet densities. Thus, we use MPC as a privacy-preserving technique to enable joint training of ML models

As the servers periodically compute ML models using streaming sensor data, we also allow clients to query the model with various aggregate queries discussed below. Fine-tuned models are kept secret-shared between the servers and not revealed to either of the servers. Similar to [1] our system forbids querying pollution data at a single location as that can reveal sensitive information. Our protocols for secure querying perform the checks necessary to ensure that the query is valid. Queries are also processed via MPC between the 2 servers and the client.

Figure 1 shows our system architecture with different stakeholders for the setting of 2 rival fleet companies. Our MPC-based solution easily generalizes to more fleets [45, 48, 49] as follows: The rival fleet companies or the data owners can begin by secret sharing their data between the two non-colluding servers, and the rest of the workflow remains the same as above. The complexity of this extension is identical to the case of 2 fleets. For simplicity of exposition, our formal description is based on 2 fleet companies.

Threat Model. Since the servers would be owned by renowned fleet companies, we assume that the servers are semi-honest and non-colluding during secure training as well as secure inference. However, it is unreasonable to assume that all of thousands of clients would only send well-formed valid queries. There is a clear incentive for clients to be malicious and send smartly crafted queries to learn sensitive information about fleet companies. Hence, we consider corrupt clients that can deviate arbitrarily from the system specifications, i.e., clients that can send arbitrary queries. Then, the query processing needs to first ensure that the query is valid before sending back the result.

Formally, we consider the setting of 2 servers and an arbitrary number of clients. We consider an adversary $\mathcal{A}$ that statically corrupts either one of the servers or an arbitrary collection of the clients. We formally show that our protocols for model training and query processing realize the ideal functionality defined in Figure 2. Note that here we define the end-to-end ideal functionality of training and query processing, and we also define the ideal functionality of query processing in more detail in Figure 6.

As is clear from the description of the ideal functionality that we realize, the servers learn no information about the model being trained. In contrast, in the federated learning-based model training, the servers would learn aggregate gradients or model updates after each iteration and hence, it does not satisfy this strong security that we achieve with MPC. In the setting of two data contributors, these gradient leakages have been used to launch explicit attacks to recover the underlying secret data of the other party [19, 67].

Consider the setting of 2 servers S_0 and S_1 and clients $\{C_i\}$.
Model Training: The input of S_b is a part of the dataset, denoted by D_b.

1: For $b \in \{0, 1\}$, S_b sends D_b to the ideal functionality $\mathcal{F}$.
2: $\mathcal{F}$ computes the joint data $D = (D_0, D_1)$. (We discuss exact data formatting/pre-processing for our setting in Section 5.1.)
3: $\mathcal{F}$ runs the machine learning training algorithm M on dataset D to obtain a model M_D. (Note that the trained model M_D is not sent to anyone.)

Query Processing: An algorithm V checks whether a query is valid or not. In particular, for a query q, $V(q) = 1$ if q is valid and 0 otherwise. We define this algorithm $V(\cdot)$ in Figure 6.

1: A client C_i sends a (potentially invalid) query $q = (\text{qType}, q')$ to $\mathcal{F}$. Here, qType denotes the type of the query, e.g., average, max, etc.
2: $\mathcal{F}$ sends qType to S_0, S_1. Then, $\mathcal{F}$ checks if q is a valid query by computing $V(q)$. If not, it sends reject to C_i. Else, it computes $y = M_D(q)$ and sends y to C_i. (Note that only the client gets the result, and the servers only learn the type of the query made.)

Figure 2: Ideal Functionality $\mathcal{F}$ for training ML models and query processing.

3.2 Detailed Problem Formulation

First, we view the urban area of interest as a rectangular grid of side lengths l and b with origin at Lat, Lon values (x, y). Next, we divide the area into small square grids of side z meters as shown in Figure 3 that defines the granularity of measurement agreed upon by all parties involved. When the servers collect the sensor information from their respective fleets (in cleartext), they locally aggregate the information into locally maintained grids by mapping to the nearest grid point. A day is bucketed into desired time slots as well. Reformatted and aggregated data held by each server thus takes the form $\langle Lat_{grid}, Lon_{grid}, t, PM\rangle$ tuples, where Lat_{grid}, Lon_{grid} correspond to cell index in the grid.

To begin with, the servers may have an incomplete grid or values at a grid location with very low confidence due to fewer sensors at that location. Via the ML models for forecasting, we learn complete grids for future times that would be kept secret shared between the servers.

Once the grids are learned, clients can perform several kinds of queries. Examples of these include current or future *average*, *minimum*, or *maximum* PM values over a set of grid locations. They can also ask *count* queries, where they specify a threshold θ value of PM and ask how many grid locations in the set exceed θ, or *range* queries with two thresholds θ_1 and θ_2 to ask how many grid locations fall within those thresholds. Note that all the queries have to specify a set of locations (grids) and we need to ensure that the set has some minimum cardinality.

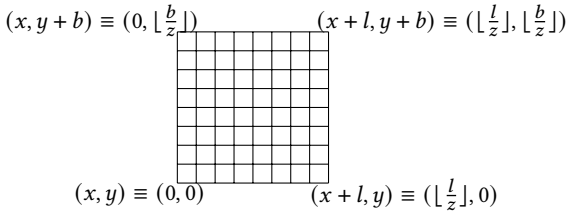


Figure 3: *Urban area divided into grid squares.*

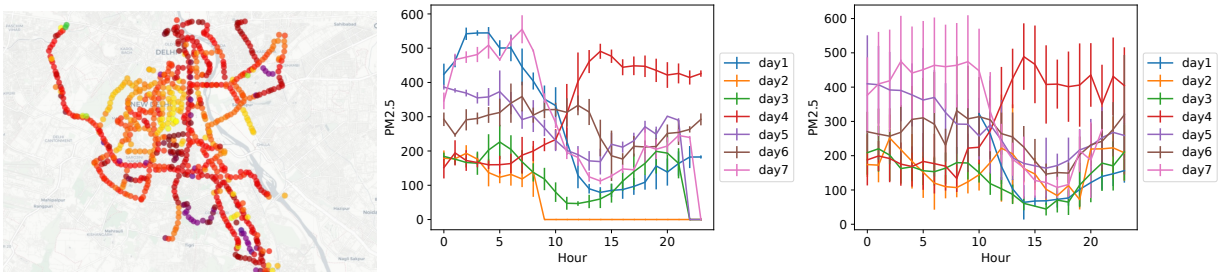
4 PM FORECASTING

State-of-the-art Machine Learning methods can solve useful problems processing PM data in intelligent ways. Predicting PM values for the future, or *forecasting*, is a significant problem. When making decisions like whether to use an indoor air purifier, wear a face mask outside, or decide to exercise indoors rather than outside, which alternate route to take from work to home in the next one hour, citizens in various urban regions should be able to access their local PM values in near future. In this paper, we study PM forecasting as target ML tasks to be solved, using state-of-the-art neural network-based ML models, while maintaining the privacy of training data across rival fleet companies.

4.1 Spatio-temporal patterns in PM data

In order to forecast PM data, there must be some underlying patterns in the PM data for ML models to capture. Figure 4(a) shows the average PM 2.5 values over the month of January 2020, as collected using instrumented public buses in a major metropolitan city. Darker colors indicate higher values of PM. Spatial patterns are evident from this plot, and nearby locations are similar in color. As pollution depends on spatial features like green cover vs. built environment in an area or the presence of local pollutant sources like industries or traffic hotspots, similarity in PM for nearby locations is intuitive. ML models for forecasting should be able to capture this spatial relation.

Figure 4(b) shows the hour of the day along the x-axis and average PM 2.5 values on the y-axis, each curve representing one day, where the averaging is done on all mobile sensors deployed in buses. Figure 4(c) plots the same graph for the same days, but with averaging done on reference grade static sensors within a 1 Km radius of the trajectories of the mobile bus sensors. The correspondence between pairwise curves in the two plots shows that mobile sensing data is as reliable as static sensor data while having much higher spatial coverage. The plots also show three interesting temporal characteristics. First, most curves (other than the red curve for day 4) have higher PM values at night, and the lowest is around 3 pm every day. This validates the known meteorological effect that temperature is negatively correlated with PM as heat expands the air, reducing PM concentration. Second, while the diurnal trends



(a) *Spatial patterns in PM data* (b) *Temporal patterns in mobile PM data* (c) *Temporal patterns in static PM data*

Figure 4: *Spatio-temporal patterns in PM data collected using instrumented buses and static sensors in a metropolitan city.*

(other than day 4) hold, the absolute PM values are not ordered in increasing or decreasing order for subsequent days. The ordering of the colors in the legend for successive days, and the ordering of the colors within the plot, do not match. Thus forecasting PM values later in the future, even for the next day, seems non-trivial, using PM data itself. Our third observation: PM for a given day has a slow rise and fall pattern without abrupt spikes, showing promise for short-term forecasting. The ML model should capture these short-term temporal trends.

4.2 ML Model for forecasting

GCN-LSTM: The GCN-LSTM model integrates the graph convolutional network and Long Short Term Memory (LSTM) units. The model consists of multiple graph convolution layers stacked over multiple LSTM to predict pollution values. The dropout layers and a dense layer at the end improve the model performance and manage over-fitting. Using historical time series data as input, GCN models the spatial features. Next, dynamic change over time is obtained by information transmission between the LSTM units.

4.3 GCN-LSTM model specifics for forecasting

The urban grid as described in Section 3.1 is modeled as a graph $G = \{V, W\}$, where V is the set of nodes, $V = \{v_i, i = 1, \ldots, N\}$, where N is the total number of nodes and W the adjacency matrix. Nodes on the graph represent grid points and the edges represent the connection relationships between these grid locations. An adjacency matrix A is used to represent the connection between adjacent grid points. A is a 0-1 adjacency matrix where

$$A(v_i, v_j) = \begin{cases} 1, & \text{if } v_i \text{ and } v_j \text{ are connected} \\ 0, & \text{otherwise} \end{cases}$$

We say v_i and v_j are connected if v_j is among the first 8 (excluding self) closest nodes to v_i based on the Euclidean distance. Now, the problem of spatiotemporal PM forecasting can be considered as learning the mapping function f on the grid G and feature matrix X and then predicting the PM values on the next time moment, as shown in equation:

$$X_{t+1} = f(G; (X_{t-n}, \ldots X_{t-1}, X_t))$$

where $X_i \in R^N$ is used to represent the PM values on each grid node at time i and n is the length of the historical time series. We use the GCN model to capture the complex spatial dependencies in the grid architecture and the PM values. A single GCN layer captures these dependencies between the nodes and its first-order neighborhood, which can be expressed as:

$$H^{k+1} = \sigma(\hat{A}H^k\theta_w^k + \theta_b^k)$$

where $\hat{A} = A+I_N$ is the adjacency matrix with added self-connections, I_N is the identity matrix, H^k is the output of k^{th} layer, θ_w^k and θ_b^k are the weight matrix and bias vector which are the trainable parameters of the k^{th} layer, and $\sigma(\cdot)$ is the nonlinear activation function. Now the GCN model is built by stacking multiple convolution layers. For example, the 2-layer GCN model can be expressed as:

$$f_{GC}(X, A) = \sigma_1(\hat{A}\sigma_0(\hat{A}X\theta_w^0 + \theta_b^0)\theta_w^1 + \theta_b^1)$$

Next LSTM is used to obtain temporal dependence from the PM data. Figure 12 shows a single LSTM cell at time t where X_t is the vector of PM values at all grid locations at time t; h_{t-1} denotes the hidden state at time $t-1$; i_t is the input gate which tells what new information is stored in the cell state; f_t is the forget gate which tells the information to throw away from the cell state; o_t is the output gate which provides the activation to the final output of the LSTM unit at time t; c_t is the cell state or memory at time t and $\tilde{c}_t$ represents the candidate of the cell state at time t. The equations for a single LSTM unit are shown as:

$$i_t = \sigma(W_i[f_{GC}(A, X_t), h_{t-1}] + b_i)$$
$$f_t = \sigma(W_f[f_{GC}(A, X_t), h_{t-1}] + b_f)$$
$$o_t = \sigma(W_o[f_{GC}(A, X_t), h_{t-1}] + b_o)$$
$$\tilde{c}_t = tanh(W_c[f_{GC}(A, X_t), h_{t-1}] + b_i)$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$
$$h_t = o_t * tanh(c_t)$$

where $f_{GC}(A, X_t)$ represents the graph convolution process defined above and W and b are the weights and biases in the training process. The output of the LSTM layer is the output of the final LSTM unit which is passed through dropout and dense layers to avoid over-fitting. We give a diagram for the GCN-LSTM architecture in Appendix, in Figure 12.

We show that 1 Km x 1 Km spatial grids and three-hour temporal buckets capture well the spatiotemporal characteristics of PM in our dataset. We model this using 2 GCN layers with [4,4] parameters and 1 LSTM layer with 4 parameters, after careful empirical analysis (Section 7.3), to get the lowest MPC latency and low average RMSE for the forecasting task.

5 PRIVATE TRAINING & QUERYING

Given the sensor data collected by multiple (possibly rival) fleet companies, we want to jointly train a useful ML model like the GCN-LSTM-based PM forecasting model discussed in the last section, in a privacy-preserving manner. We focus on the 2-server setting where these two servers could be the data owners or multiple data owners can secretly share their data with the two servers who participate in secure training via 2PC (Section 2.2). Since the final trained model might reveal sensitive information regarding the data used for training, we would keep the final model secret shared it between the two entities participating in secure training. Later on, we enable secure query computation over the secret shared pre-trained model.

Overview. Given a public mapping from (Lat, Lon) to grid location, as discussed in Section 3.1, the 2 data owners first locally aggregate the sensed data. Next, the GCN-LSTM model needs to be jointly and securely trained/fine-tuned for a certain number of epochs using forward and backward passes on new data. A final forward pass is needed to get the forecasted PM values for each grid point, with this output kept secret shared between the two servers. Client queries about average, and min/max pollution come for a subset of these points, which are responded to by the two servers jointly, again through MPC. Our system with 2 data owners can be trivially extended to multiple data owners and 2 non-colluding servers using standard techniques [45, 48, 49].

We discuss the important steps of secure training starting with data pre-processing and privacy-preserving query responses next. Our secure training builds on SecFloat [53] is a generic ML-independent library for 2PC of floating-point operations It provides 2PC protocols for both primitive operations like multiplying/adding/dividing floating-point values and for math functions like exponentiation, square root, etc. Additionally, SecFloat provides SIMD (single instruction multiple data) counterparts of these operations as well as protocols for compound operations such as matrix multiplications.

5.1 Pollution data pre-processing

Both servers collect (new) data and pre-process it into a secret shared common grid to be fed into the secure training.

- For $b \in \{0, 1\}$, Server S_b creates two grids G_b, B_b: G_b stores accumulated pollution level at a point and B_b indicates how many data points does it correspond to (to allow for computing the correct average value across data collected by both servers, see below).
- For $b \in \{0, 1\}$, S_b secret shares G_b, B_b with the other server. With this, S_0 and S_1 hold $\langle G_0 \rangle^{\mathsf{FP}}, \langle B_0 \rangle^{\mathsf{FP}}, \langle G_1 \rangle^{\mathsf{FP}}, \langle B_1 \rangle^{\mathsf{FP}}$.
- S_0 and S_1 invoke SIMD addition protocol on $\langle G_0 \rangle^{\mathsf{FP}}, \langle G_1 \rangle^{\mathsf{FP}}$ to compute $\langle G \rangle^{\mathsf{FP}}$, where $G[i][j] = G_0[i][j] + G_1[i][j]$. Similarly, they invoke SIMD addition protocol on $\langle B_0 \rangle^{\mathsf{FP}}, \langle B_1 \rangle^{\mathsf{FP}}$ to compute $\langle B \rangle^{\mathsf{FP}}$, where $B[i][j] = B_0[i][j] + B_1[i][j]$.
- S_0 and S_1 invoke SIMD division protocol on $\langle G \rangle^{\mathsf{FP}}$ and $\langle B \rangle^{\mathsf{FP}}$ to compute $\langle X \rangle^{\mathsf{FP}}$ such that $X[i][j] = G[i][j]/B[i][j]$ if $B[i][j] \neq 0$ and $X[i][j] = 0$ otherwise.

Secret shares of X are input to the secure training phase.

5.2 Secure training of GCN-LSTM

Once servers have secret shares of the grid with PM data, they input these secret shares in a secure 2-party computation (2PC) and get the secret shares of the trained model as output. For the case of fine-tuning, they start training using secret shares of the pre-trained model. Once the model is trained/fine-tuned, parties run one forward pass to learn new updated values of the grid used for forecasting. Secure training is done by invoking the appropriate APIs that we have built on top of SecFloat [53]. Our API decomposes the operations required in the training of GCN-LSTM, i.e., the operations required in the forward pass and the backward pass, to the generic operations provided by SecFloat.

This decomposition is done using the textbook descriptions of the ML operators. Convolutions are translated to matrix multiplications. Sigmoid and hyperbolic tangents are decomposed into exponentiations, additions, multiplications, and reciprocals. SecFloat protocols performance is bottlenecked by communication. The communication requirement of each individual floating-point operation (flop) is in kilobytes [53]: addition costs 11.1 KiB, multiplication costs 3.1 KiB, division costs 10.3 KiB, comparison costs 1.1 KiB, and exponentiation is over 30 KiB. The most expensive operation in the training of GCN-LSTM is the convolution which requires many multiplications and additions. We discuss how the GCN-LSTM model is implemented using SecFloat APIs in Section 6 and evaluate the runtime and communication overhead of the forward and backward passes for training in Section 7.

5.3 Computing the integer grid for queries

The secure training phase is followed by one forward pass to compute the final forecasting grid. Note that each grid value will be a secret shared floating-point value. We note that each grid point corresponds to a PM value taking values from 0 to 2000 and hence, to support subsequent queries, it is sufficient to quantize these values to integer values. This can enable much faster querying as now the secure queries can run on integer/fixed-point values instead of more expensive floating-point operations. For this, we provide a new protocol to convert floating-point values to integers. The protocol can be easily generalized to fixed-point values. Let ℓ be the length in bits required by the integer. We run the following protocol for each value in the grid in parallel.

Protocol overview for the float to integer. Recall that the numerical value of a floating-point number x represented as a 4-tuple (z, s, e, m) is $(1 - z) \cdot (-1)^s \cdot \frac{2^e \cdot m}{2^q}$ (Section 2.1). Now, to convert this to an ℓ-bit integer, we simply compute the above expression and round it to an integer. Note that not all possible floating-point values can have a meaningful integer representation. For instance, values smaller than 1 in magnitude (i.e., $e < 0$) need to be rounded to 0, and values with magnitude greater than $2^{\ell-1}$ (i.e., $e > \ell - 1$) can have garbage results. Formal protocol description is in Figure 5.

The protocol uses many 2PC building blocks over integer computations that we explain as we use them. To compute 2^e, in Step 2, we use a lookup table (LUT) EXP of size ℓ that maps i to 2^i for $i \in [0, \ell - 1]$. We use Π^{LUT} to denote the 2PC protocol for lookup tables that, given a secret shared index, outputs secret shares of the value of the lookup table at that index. This protocol is based on

Protocol Float-to-Int

Input: S_0 and S_1 hold $\langle x\rangle^{\mathsf{FP}} = (\langle z\rangle^{\mathsf{B}}, \langle s\rangle^{\mathsf{B}}, \langle e\rangle^{p+2}, \langle m\rangle^{q+1})$.
Output: S_0 and S_1 hold $\langle y\rangle^{\ell}$, where ℓ is output bit length.

1: For $b \in \{0,1\}$, S_b sets $\langle e'\rangle_b^k = \langle e\rangle_b^{p+2} \mod k$ for $k = \log \ell$.
2: $\langle n'\rangle^{\ell} = \Pi^{\mathsf{LUT}}(\mathsf{EXP}, \langle e'\rangle^k)$.
3: $\langle \mathrm{gt}_1\rangle^{\mathsf{B}} = \Pi^{>}(\langle e\rangle^{p+2}, -1)$.
4: $\langle n\rangle^{\ell} = \Pi^{\mathsf{MUX}}(\langle \mathrm{gt}_1\rangle^{\mathsf{B}}, \langle n'\rangle^{\ell}, 0)$.
5: $\langle \mathrm{gt}_2\rangle^{\mathsf{B}} = \Pi^{>}(\langle e\rangle^{p+2}, q)$.
6: $\langle m\rangle^{\ell} = \Pi^{\mathsf{ZXT}}(\langle m\rangle^{q+1}, q+1, \ell)$.
7: $\langle t_1\rangle^{\ell} = \Pi^{\mathsf{MULT}}(\langle m\rangle^{\ell}, \Pi^{\mathsf{ARS}}(\langle n\rangle^{\ell}, q))$.
8: $\langle t_2\rangle^{\ell} = \Pi^{\mathsf{ARS}}(\Pi^{\mathsf{MULT}}(\langle m\rangle^{\ell}, \langle n\rangle^{\ell}), q)$.
9: $\langle t\rangle^{\ell} = \Pi^{\mathsf{MUX}}(\langle \mathrm{gt}_2\rangle^{\mathsf{B}}, \langle t_1\rangle^{\ell}, \langle t_2\rangle^{\ell})$.
10: $\langle \mathrm{res}\rangle^{\ell} = \Pi^{\mathsf{MUX}}(\langle s\rangle^{\mathsf{B}}, \langle -t\rangle^{\ell}, \langle t\rangle^{\ell})$.

Figure 5: *Protocol for floating-point to integer conversion*

ℓ-choose-1 oblivious transfers [38, 54]. We perform this step using only the lower $\log \ell$ bits of the exponent (as the final integer value is only ℓ bits) to learn n'. Next, we note that above calculation is meaningless if $e < 0$. Hence, we check for this in Step 3 using a protocol for comparison $\Pi^{>}$ that returns boolean shares of the result gt_1 [55]. Then, we use a multiplexer protocol Π^{MUX} with choice bit gt_1 to choose between the value looked up n' and 0 and denote the result by n (Step 4) [55]. Note that $n = 2^e$ when $0 \le e \le \ell - 1$, $n = 0$ when $e < 0$, and can be arbitrary for other values of e.

Multiplication of n and m needs to be done carefully to avoid unnecessary overflows, that lead to the multiplexer in Step 9. In particular, the mantissa is a fixed-point value with scale q and hence, after multiplying with 2^e we need to right shift the result by q. Now if e is large enough (that is, $e > q$), we right shift 2^e before multiplying by m (Step 7). Else, we first multiply and then right shift the result (Step 8). Here, Π^{ARS} denotes the protocol for arithmetic right shift and Π^{MULT} denotes the protocol for integer multiplication [55]. Furthermore, before multiplication, we extend the mantissa from $q+1$ bits to ℓ bits using a protocol Π^{ZXT} for zero extension (Step 6) [54]. We also assume that $q < \ell/2$ so that the multiplication does not overflow. Finally, we set the sign of the result using the sign bit of the floating point value in Step 10.

Theorem 1. *For a floating-point value $\langle x\rangle^{\mathsf{FP}} = (\langle z\rangle^{\mathsf{B}}, \langle s\rangle^{\mathsf{B}}, \langle e\rangle^{p+2}, \langle m\rangle^{q+1})$ representing real value r, if c_1 and c_2 are two consecutive $\ell > 2q$ bit integers such that $c_1 \le r \le c_2$ then the above protocol returns $\langle c_1\rangle^{\ell}$ if $r \ge 0$ and $\langle c_2\rangle^{\ell}$ otherwise. Note that for our case we use standard floats with $p = 8, q = 23$, and $\ell = 64$-bit integers, and $\ell > 2q$ holds.*

Proof. There are three different cases here. First, when $r = 0$, we have $e < -1$ and line 3 ensures that the output is zero. Second, when $e > q$ then the protocol exactly evaluates the integer $r = (-1)^s \cdot 2^{e-q} \cdot m$. Third, when $e \le q$ then the protocol truncates the least significant bits of m, which causes positive numbers to be rounded down and negative numbers to be rounded up. □

Once we obtain secret shares of the grid as shares of 64-bit integers, the servers can convert them to lower bitwidth, e.g., $\ell = 16$, by discarding the top 48 bits of their respective shares locally. This step is sound because the PM value is between 0 and 2000 and standard properties of arithmetic secret sharing.

5.4 Privacy preserving query response

For a grid with d cells, the client wants to query aggregate statistics over some cells in the grid. For example, the client wishes to know the average/min/max PM value along the intended path to travel without revealing the path to the servers. In this secure computation, the client's input is the path and the servers input their secret shares of the PM values of the grid. The output is the aggregate PM result over the path which is only revealed to the client. The query result is provided only if the number of cells the client is interested in is greater than some threshold, say T. We need this step to ensure that the client is computing aggregate statistics over a threshold T number of cells and not simply querying a single grid point. [1] discussed smartly crafted consecutive client queries that overlap at one grid, to get PM values of individual grids, and how to handle it for average queries adding DP noise to the output. Similar DP noise can be added in our setting. The queries supported in this paper are listed in Table 1. The ideal 3-party functionality for secure queries is described in Figure 6.

Query	Inputs	Output
Average	$[G]$, $[X]$	$\frac{\sum_{X_i=True} G[i]}{\sum_i 1[X_i = True]}$
Minimum	$[G]$, $[X]$	$\min_{X_i=True} G[i]$
Maximum	$[G]$, $[X]$	$\max_{X_i=True} G[i]$
Count	$[G]$, $[X]$, θ	$\sum_{X_i=True} 1[G[i] \ge \theta]$
Range	$[G]$, $[X]$, θ_1, θ_2	$\sum_{X_i=True} 1[\theta_1 \le G[i] \le \theta_2]$

Table 1: *Types of queries supported*

Parameters. Public area grid with d cells and pollution values are over $\ell = 16$ bits. Threshold T.
Input: Servers S_0 and S_1 respectively hold secret shares $\langle [G]\rangle_0^{\ell}$ and $\langle [G]\rangle_1^{\ell}$ of d pollution values $[G]$. Client holds a list of d boolean values $[X]$ and a type of query Q.
Output: The functionality sends the type of the query Q to the servers S_0 and S_1. Next, it checks whether at least T indices in $[X]$ are true. If the check passes, the functionality reconstructs the pollution grid $[G]$ and returns the result of the query Q on $[G]$ and $[X]$ to the client.

Figure 6: *Ideal Functionality for Grid Queries.*

Queries are computed using MPC in the following three phases.
Phase 1: The client creates an array $[X]$ of d Booleans where X_i is *true* iff the i^{th} cell lies on the path. This array is then secretly shared between the two servers, i.e., the client gives boolean share $\langle [X]\rangle_b^B$ to S_b, $b \in \{0,1\}$.
Phase 2: The two servers realize the following ideal functionality via secure 2-party computation (2PC) where the input of S_b is $(\langle [G]\rangle_b^{\ell}, \langle [X]\rangle_b^B)$. It reconstructs the pollution grid $[G]$ and the client grid of interest $[X]$. Then, it checks whether $[X]$ has at least T *true* indices. Next, if the check passes, it computes the required statistic corresponding to the query result. This step involves computing the average/min/max etc of PM values of the cells with indices i such that $X_i = true$. Finally, it secret shares the query result between the two servers.

Phase 3: The servers send the secret shares of the result to the client who can then reconstruct the result.

Note that only the client learns the result and 2PC ensures that the servers learn nothing about the client's input. Also, the client learns nothing about the PM values in the grid beyond what is revealed by the aggregate result. As is clear, the client is quite lightweight and is only involved in phases 1 and 3 and the total communication incurred is $2d + 2\ell$ bits, which is less than a KB in our evaluation. Unlike prior work [1], where the client participates in 2PC and suffered from high communication overheads, here the 2PC in phase 2 is between the servers and does not involve the client. Thus, the servers communicate an MB of data (Section 7.5) but the communication overheads of the client are negligible.

5.5 Security

We need to show that putting together our protocols in Sections 5.1, 5.2, 5.3, 5.4, we realize the ideal functionality defined in Figure 2. We show security using the standard *hybrid model* [42] that replaces the calls to sub-protocols with respective ideal functionalities with the same input and output behavior. Moreover, we note that after each of these phases of our overall protocol, i.e., data pre-processing, model training, quantization, and query processing, the outputs are kept 2-out-of-2 secret shared between the servers (and not revealed to any server).

First, we show that composing the protocols for data pre-processing in Section 5.1 and secure training in Section 5.2 provides a secure protocol for the model training phase, i.e., if one were to reconstruct the secret shares of the output held by the servers, it would result in the same model (or, the pollution grid) as learned by $\mathcal{F}$ and we can simulate the view of the corrupt server. This trivially follows from the semi-honest security of protocols in SecFloat [53].

Next, by Theorem 1, at the end of the protocol in Figure 5, servers learn the shares of the correct quantized grid and nothing else. Finally, composing this with our 3-party query processing protocol from Section 5.4, we get a secure protocol for query processing that is secure against a semi-honest server or client. More formally, since the client only secret shares its input and only receives the output of 2PC between the servers, the protocol is trivially secure against a client (while this client may choose its input in any arbitrary manner; since the servers run the query validity check using a secure 2PC, it is guaranteed that the client's query must have been valid). Also, since the servers compute the check for query validity and the query itself using a secure 2PC, the protocol is also secure against a semi-honest server (i.e., the client's privacy is protected against a semi-honest server).

Finally, by composing protocols for training and query processing using the hybrid model, we get that the overall protocol securely realizes ideal functionality in Figure 2.

6 IMPLEMENTATION

With the GCN-LSTM model shown in Figure 12, we implement the forward and backward passes using SecFloat APIs (code snippet in Appendix). In the backward pass, first, the error gradient is computed from the outputs and labels, then the gradient of the sigmoid function is computed as element-wise multiplication of the activation values ($z \cdot (1 - z)$). Subsequently, the gradients are backpropagated through the final linear layer of the network and also backpropagated-through-time [65] in the LSTM layer and so on. For training over multiple epochs, we implement the Adam optimizer on top of SecFloat with the default β_1, β_2 parameter values as specified in [36]. Moreover, since most of the computations involved in the forward/backward passes are embarrassingly parallel *i.e.,* they involve multiple independent computations such as (SIMD) additions, (SIMD) multiplications, and point-wise activations, we parallelize them over 16 threads. During query time, there is no floating-point arithmetic and the query computations are over integer arithmetic. For queries, we use EMP [62] as the 2-party cryptographic backend. We have observed that, among the available 2PC implementations, EMP provides the best concrete performance for our queries. This is because almost all queries do boolean computations such as comparisons, and none of the queries require integer multiplications.

7 EVALUATION

We explore the trade-offs between good forecasting accuracy and MPC overheads (Figure 8, Section 7.3) to generate a GCN-LSTM model that a) provides better accuracy than baseline models (Table 2, Section 7.4), b) the queries can be processed in real-time with MPC (Table 6, Section 7.5), and c) the MPC-based training is efficient enough that the model can be fine-tuned within 3 hours. Ideally, one would like to train from scratch instead of fine-tuning (Figure 11), but then the models won't be ready on time. We show a schedule that combines secure training from scratch and secure fine-tuning to ensure the availability of accurate models (Table 8, Section 7.7).

7.1 Experimental setup

PM Dataset The Particulate Matter (PM) dataset used in this paper to evaluate forecasting accuracy, has been recorded over three months from November 2020 to January 2021 for a total of 91 days. The dataset contains more than 12.5 million sensor data points and spans an area of 559 square km. We use PM 2.5 values along with the GPS location and time of the collected data, to evaluate forecasting accuracy. To evaluate MPC performance, we split the dataset into parts, to emulate different fleet companies holding different parts of the dataset in their respective servers (Figure 1).

Train test split As in any time-series data, we need to respect chronological order. We thus combine data from 7 consecutive days, the first 6 days form the train set and the model is tested on the 7^{th} day. This choice of using 6 days in historical data in forecasting will be further analyzed in Section 7.3. Our test days are 15 Nov, 20 Nov, 28 Nov, 7 Dec, 15 Dec, 7 Jan, 10 Jan, and 24 Jan. We have a common DNN architecture for predictions on weekday vs. weekend, but our model weights are constantly updated using streaming data of the last 6 days.

Evaluation metrics To evaluate MPC overheads, we measure end-to-end latency/runtimes and communication incurred between participants. To quantify forecasting accuracy, we use the standard Root Mean Squared Error (RMSE) metric defined as $\sqrt{\Sigma_{i=1}^{n}\left(\frac{\hat{y}_i - y_i}{n}\right)^2}$ where $\hat{y}$ is the forecasted value, y is the observed value, and n is the number of observations. Lower RMSE indicates more accurate forecasting. Figure 7 shows the time of the day along the x-axis

with actual or sensed PM values along the y-axis for some days. The forecasting RMSE values are shown in the legend, corresponding to each day. Some days show a lot of variation in PM values (e.g. Jan 15 and Nov 24). For these days, the forecasting RMSE is higher, as the high variation of PM values is difficult to model and predict for the ML algorithm. As RMSE is data dependent, our evaluations present the average RMSE values with standard deviations, over the eight test days spread over the three months dataset.

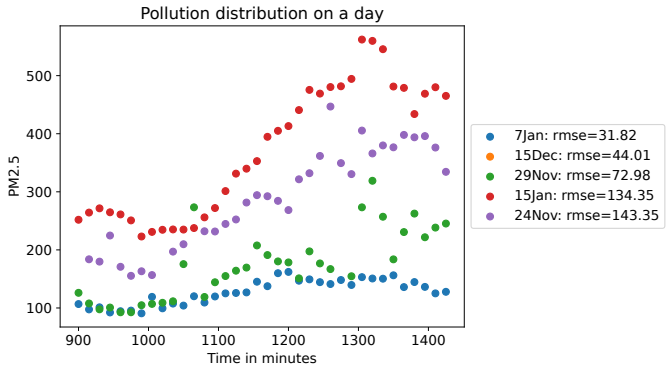


Figure 7: *RMSE for forecasting task depends on PM value fluctuations. Higher fluctuations result in higher RMSE.*

Hardware specifications MPC latency and communication depend on the hardware specifications of the experimental platforms. We use two 32-core 2.4GHz machines, with 128 GB RAM each, connected with a 16 Gbps LAN connection.

7.2 Data representation for forecasting

We need to construct a time series from the raw PM dataset described above, to input to the ML model for training the model and forecasting future values. Since this data representation affects the input to the ML model, and the input dimensions to the ML model in turn affect the model training overhead in MPC, clarifying the representation is necessary.

Spatial grids We divide the 559 square Kms of our dataset into a square of size (25*25) so that each small grid represents a region with an area of approximately $0.9Km^2$. Our forecasting and querying granularity is at the level of this approximately $1Km^2$ square grid. Any sensor reading whose latitude and longitude fall within the boundaries of a grid is binned as an event of that grid. Since the local buses that collect these PM values move along certain defined trajectories, certain grid locations that do not lie on these trajectories have no corresponding PM readings. 270 out of 625 grids are actually populated across 3 months. We use these 270 spatial grids as input to our forecasting model.

Temporal buckets Similar to how to discretize lat-long location space into spatial grids, we also discretize the time-space into buckets of 3-hr duration. As PM readings are collected at a high frequency of 0.2 Hz, for a given grid location, all PM 2.5 readings in a 3-hr window lying within the boundaries of the grid location are averaged for forecasting. We have only 20% and 4% values on average, collected in the first two temporal buckets respectively, as most buses are not operational from 12 AM-6 AM. We, therefore, drop the first two temporal buckets from our time-series construction and consider the remaining 6 buckets totaling 18 hours each day. The time series for a single day is thus constructed such that, each grid location is a vector of size 6, filled with aggregated PM2.5 values collected over an interval of 3-hr from 6 AM-12 midnight and 0 wherever missing. We show the sample time series for one day (Dec 7, 2020) for some grid locations in Appendix, which forms the input to our ML model to train and evaluate forecasting.

If we have bigger temporal and spatial grids, number of grids will be smaller. This will reduce both model training and query latencies through MPC, as computation time depends on number of grids. However, with bigger temporal and spatial grids, PM forecasting will be at a coarser granularity. Fig. 4(a) shows that PM 2.5 has significant variations across nearby locations, and Fig. 4(b) shows PM 2.5 has significant variations within nearby times. Thus there will be an accuracy trade-off in PM forecasting, if we choose bigger grids to reduce MPC latencies. We therefore choose 3 hours forecasting time buckets, and about 1 sqKm. spatial grid size, and empirically show that both training and query times with MPC are practical for this grid granularity.

7.3 Balancing RMSE and latency

As discussed in Section 4, spatiotemporal patterns in PM data can be modeled with GCN-LSTM for the forecasting problem, the GCN capturing the spatial characteristics in the data, and the LSTM capturing the temporal characteristics. How many GCN and LSTM layers to use in the model, what should be the dimensions of those layers, how much historical data to use for model training, and how the chosen model compares in forecasting RMSE to other ML models as a baseline, are empirically analyzed in this section. The goal is to examine trade-offs between forecasting RMSE and MPC latency, and understand how to balance such trade-offs if any.

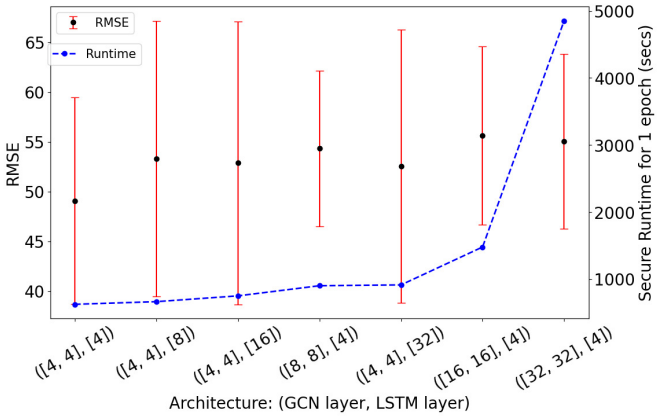


Figure 8: *Forecasting RMSE vs. MPC latency to train 1 epoch, for different GCN-LSTM architectures. RMSE across different architectures are comparable, but latency varies.*

Forecasting model architecture and parameters: Figure 8 shows some sample GCN-LSTM models along the x-axis, with 2 GCN layers with [i, i] parameters and 1 LSTM layer [j] parameters. The left y-axis shows the average forecasting RMSE with standard deviation over all test days, and the right y-axis shows the MPC runtime for

training the model for 1 epoch. The x-axis is ordered in increasing order of MPC runtime from left to right.

Forecasting RMSE is similar across the different GCN-LSTM model parameter choices, with the mean RMSE near 50 and s.d. of ±20 for all models. These mean and s.d. of RMSE across the test days can be explained from the trends we saw in Figure 7, where each day had a different forecasting RMSE based on the fluctuation of PM within that day.

While the mean and s.d. of RMSE are similar across ML models, their MPC latencies vary depending on how many private computations each ML model parameter choice entails. Smaller models with e.g. only one GCN layer suffer from higher RMSE and bigger models with more GCN-LSTM layers have even higher MPC latency. We, therefore, use 2 GCN layers with [4,4] parameters and 1 LSTM layer with 4 parameters, which has the lowest MPC latency and low average RMSE in subsequent experiments.

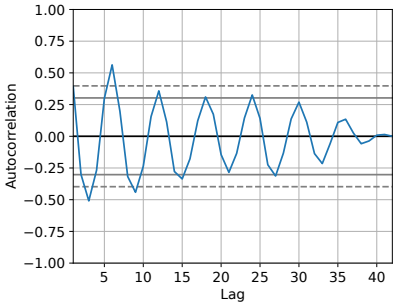


Figure 9: *PM data shows maximum auto-correlation at sequence length of 6. Given we use 6-time buckets for 24 hours, this sequence length indicates the same time on the previous day. Also, there is no correlation beyond 40-time buckets, which is approximately 7 days. We, therefore, use 6 days of historical data to forecast for the 7^{th} day.*

How much temporal history to use in the LSTM cell? Parameter *seq_len* determines the size of the past window of information to be used for LSTM modeling. The auto-correlation function helps to identify what time lags have significant correlations, indicating that past values influence the current value. Figure 9 shows that lag 6 has large positive correlations and correlations taper off slowly as the lags increase. That means, the observations y_t and y_{t-6} are highly correlated. This can also be explained semantically as the observations y_t and y_{t-6} correspond to the same time-buckets 1 day apart, as a single day has 6 time-buckets. Thus, we choose $seq\_len = 6$ for our LSTM cell in subsequent experiments.

How much temporal history to use in terms of past days? While we described the train test data split in Section 7.1, we mentioned using the past 6 days of training data for each test day. Why we choose this value of 6 days in training, is also explained in Figure 9. As can be seen towards the right of the figure, auto-correlation drops down to almost 0 after 40-time buckets. With each day having 6-time buckets, 42-time buckets amount to almost 7 days. i.e. a week. Longer temporal history means larger inputs to the GCN-LSTM model training, increasing MPC latency. As beyond a week PM values are hardly correlated, 6 days of historical data for forecasting PM values on the 7^{th} day seems optimal. We thus take our three months' data in 7 days chunks, use 6 days for training the forecasting model, and the 7^{th} day for testing.

7.4 Forecasting RMSE baseline comparison

We use a GCN-LSTM model comprising 2 GCN and 1 LSTM layer with ([4,4], [4]) parameters, with a sequence length of 6 for LSTM cell, and using 6 days of history as training data. Many statistical and ML models have the ability to forecast values in a time series, and therefore we need to compare the forecasting RMSE of our model against other baseline models. We use two baseline models for comparison - ARIMA, and N-Beats by Darts. *(a) Auto-Regressive Integrated Moving Average (ARIMA) [26]* uses a linear regression model. The data is prepared by a degree of differencing in order to make it stationary, i.e. to remove trend and seasonal structures. The parameters of the ARIMA model are: p or the number of lag observations included in the model, also called the lag order, d is the number of times that the raw observations are differenced, also called the degree of difference and q is the size of the moving average window, also called the order of moving average. We use ARIMA with parameters (5, 1, 1) after careful empirical analysis. *(b) N-BEATS by Darts* Darts [25] is a Python library for easy manipulation and forecasting of time series. We use the state-of-the-art deep learning model N-BEATS from Darts as a baseline. N-BEATS DNN architecture comprises backward and forward residual links and a very deep stack of fully-connected layers. More details on the N-Beats model architecture and forecasting accuracies on a variety of use cases can be found in [50]. We use ARIMA as a baseline, as it is the state-of-the-art method for pollution forecasting [6, 20]. We additionally use N-BEATs as a baseline, though it has not been used for pollution but for other spatiotemporal data like travel time estimation, and is the state-of-the-art method for graph neural network-based forecasting tasks.

Test Day	GCN-LSTM	N-BEATS	ARIMA
7-Jan	**31.816**	34.563	51.714
10-Jan	**38.646**	51.002	48.720
15-Dec	**44.014**	59.245	68.044
7-Dec	49.160	**44.510**	119.986
24-Jan	**52.639**	70.700	73.510
15-Nov	**55.204**	157.184	114.708
28-Nov	**59.229**	64.645	78.415
20-Nov	**62.014**	62.117	101.589
Average	**49.090**	67.996	82.086

Table 2: *RMSE for different forecasting models*

Table 2 shows that the GCN-LSTM forecasting RMSE is lower on the different test days, compared to the two baselines ARIMA and N-BEATS. The reasons for outperforming the two baseline models are different. ARIMA does not capture spatial relations of similarity in PM for nearby regions. It models the PM dataset solely as time series and thus can capture only the LSTM part of our model, not the GCN. N-BEATS models both spatial and temporal relations. But being a very complex model with millions of parameters, it needs significantly large training datasets to ensure generalization on test data. PM datasets are typically small, as sensor data collection with hardware instruments is costly. Thus instead of using a complex DNN model like N-BEATS, a smaller architecture like our 3-layer GCN-LSTM with carefully chosen layer dimensions, sequence length, training data size, and other hyper-parameters, gives promising forecasting accuracies.

Step	GCN-LSTM	Arima	NBeats
Data pre-processing	23.35	24.49	22.08
Training	5.04	15.34	44.37
Inference	0.68	0.34	0.89

Table 3: Data pre-processing, training (one epoch), and inference times in seconds, for different forecasting models.

Table 3 shows that our carefully designed GCN-LSTM model has much lower training time per epoch than ARIMA or NBeats. There is a marginal increase in data pre-processing time compared to NBeats and inference time compared to ARIMA, such that inference is still sub-second. These runtime trends of plaintext computation for these ML models will hold in secure computation. As PM data will be collected by sensors in a streaming fashion, continuous pre-processing of the fresh incoming data, and training with newly added data will be important. Low inference times are also important for scaling query handling on the trained model. Thus both in terms of RMSE and model training times, the GCN-LSTM model performs better than the baselines. Our cryptographic protocols will maintain the RMSE advantage of the GCN-LSTM model but will increase the runtimes. We therefore carefully evaluate next the MPC cost of training our GCN-LSTM model and inference latencies for handling client queries, on streaming PM data.

7.5 MPC cost for model training and queries

Forecasting RMSE for our carefully designed GCN-LSTM model is better than state-of-the-art baselines [25, 26]. However, the cost overhead of MPC in training this model on streaming PM data, as well as answering a variety of client queries, needs to be evaluated to understand the practicality of the proposed system.

Secure training for an epoch. Table 4 and Table 5 show the runtime and communication overhead for each layer of the GCN-LSTM network, for forward-pass and backward-pass respectively, when computed using 2PC for floating-point [53]. A combination of one forward and one backward pass constitutes one training epoch. Typically any ML model is trained over several epochs. Thus the total runtime and communication will be the sum of the total values in these two tables (7 minutes, 241 GiB) multiplied by the number of epochs for which training happens. Given the cost of securely running training for an epoch, we discuss the number of epochs required for various scenarios in Sections 7.6 and 7.7.

NN layer	Latency (s)	Percent	Comm. (GiB)	Percent
GCN Layer 1	176.00	54.8%	112.31	57.44%
GCN Layer 2	114.02	35.5%	76.78	39.27%
LSTM	29.46	9.17%	5.88	3.01%
Dense Layer	1.62	0.50%	0.56	0.29%
Total	321.11	100%	195.53	100%

Table 4: *2PC cost for GCN-LSTM forward-pass for each layer.*

Secure queries. Table 6 shows the runtime and communication overhead of servers for answering client queries through MPC. As seen from the table, the query response times are negligible (in milliseconds). Communication cost between servers is also small. Recall that for each of the queries, the servers need to first check that the client query is an aggregate over more than a threshold number of cells. This check takes roughly 0.73 ms and 0.68 MB of communication. The numbers in the table for each of the queries are inclusive of this check. For instance for average 0.68 MB is taken by the check and rest of 0.4 MB is taken by average computation. In contrast, if we had not quantized to an integer grid, and ran the query with secure floating-point, the check would take 0.68 MB and 625 additions would take about 7 MB of communication, which is an order of magnitude higher. Recall that the communication between the client and servers is already negligible; the client only communicates <1KB of data with the servers (Section 5.4). Hence, the overheads of MPC are bearable for real-time query processing.

NN layer	Latency (s)	Percent	Comm. (GiB)	Percent
GCN Layer 1	13.61	14.63%	9.16	20.17%
GCN Layer 2	5.63	6.05%	3.36	7.39%
LSTM	31.66	34.04%	9.92	21.83%
Dense Layer	3.18	3.42%	0.81	1.79%
Weight Update	38.91	41.84%	22.17	48.82%
Total	93.00	100%	45.42	100%

Table 5: *2PC cost for GCN-LSTM backward-pass for each layer.*

Function	Latency (ms)	Comm(MB)
Average	3.57	1.08
Maximum	3.62	1.09
Minimum	3.72	1.09
Count	4.34	1.22
Range	4.78	1.37

Table 6: *MPC cost on the servers for answering client queries. The client communicates < 1 KB for each of these queries.*

7.6 Model updation with streaming data

Consider the problem of building the ML model for forecasting of air pollution using data from the past 6 days, i.e., the model trained on data of days $T-6$ to $T-1$ needs to forecast for the day T. Assume that there exists such a model. Next, we get streaming data on day T and need to build a model to forecast for the day $T+1$. We shift the training window to days $T-5$ to T. We consider two ways of training. **1. Training from scratch:** Each time that we move the window (per day), we train the model from scratch using randomly initialized weights. **2. Fine-tuning:** Here, we start training with previously learned weights for the last window.

Figure 10 shows training plots for both methods on a sample day of Dec 8, 2020. Training from scratch is done with a higher learning rate of 0.01 and fine-tuning with a lower learning rate of 0.004, which are the best hyper-parameters found in grid-search. As seen from the figure, fine-tuning converges faster and more smoothly than training from scratch. We see this pattern on all test days, that 20 epochs of fine-tuning at a learning rate of 0.004 seem sufficient to converge the training losses.

Next, we discuss the time required to securely run 20 epochs between 2 servers on newly streamed data end-to-end. This includes all 4 phases: pre-processing the data, training for 20 epochs, one additional forward pass for forecasting the grid based on the trained model, and final one-time conversion of values across the grid from float-to-integer (to handle client queries). Table 7 shows the secure training time of our GCN-LSTM model for 20 epochs for all 4 phases. We see that training in MPC for 20 epochs takes roughly 2.5 hours and is practical. In particular, as no new data comes between 12-6 am of the day $T+1$, that window can be easily used for fine-tuning the new model which takes about 2.5 hours.

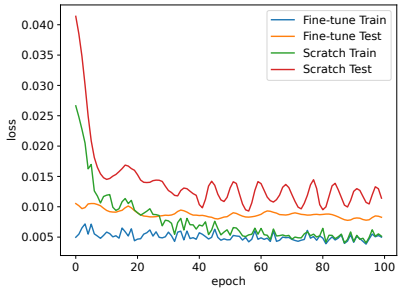


Figure 10: *Fine-tuning vs. training from scratch on Dec 8*

Step	Latency (s)	Comm. (GiB)
Data pre-processing	1.476	1.043
Training (20 epochs)	8282.2	4819
Float grid computation	321.11	195.53
Float-to-integer conversion	0.422	0.057
Total	8605.21	5015.63

Table 7: *End-to-end GCN-LSTM training latency for 20 epochs using MPC, including data pre-processing and model post-processing to obtain final integer grid for client queries.*

7.7 Training from scratch plus fine-tuning

Is fine-tuning an old model over and over again through weeks and months good for forecasting RMSE in the long run? Figure 9 showed PM values have maximum auto-correlation to the same hour on the past day. This correlation gradually decreases and becomes negligible after a week. Figure 11 shows forecasting RMSE from Dec 8 to Dec 16, when the same model is continued to be fine-tuned to forecast for the day T adding data from day $T-1$, for the day $T+1$ adding data from day T and so on. The RMSE starts to rise after day 5, showing fine-tuning over longer periods reduces forecasting accuracy.

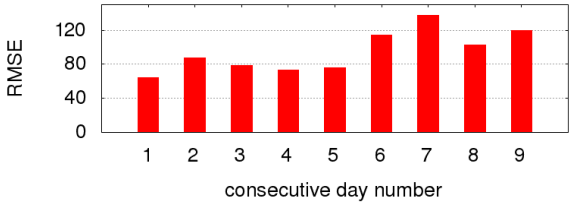


Figure 11: *RMSE rise with continued fine-tuning of a model*

But training from scratch takes 100-120 epochs to converge (Figure 10), which will take close to 15 hours in MPC! Data for the day $T-1$ finishes streaming in at 12 am of the day T. If one starts training the model from scratch at this time, the trained model will not be ready in 6 hours by 6 am of the day T, when new data starts streaming in requiring fresh forecasts. How can we regularly update our GCN-LSTM model with PM data from recent past days, maintaining privacy through MPC, if training in MPC is unable to keep up with streaming data?

We devise a practical training procedure combining training from scratch with fine-tuning so that we can use a regularly updated model incorporating the latest data while tolerating significant MPC runtime latency. At 6 am of the day T, we need model M_t ready to use the new incoming data as features, for forecasting for the next time bucket. Our procedure to train M_t is as follows: we start training M_t with randomly initialized weights i.e. train from scratch, at 6 am on the day $T-1$, using buffered values from day $T-7$ to $T-2$ as training data and a faster learning rate of 0.01 which we have found to be good for training from scratch. We continue this training from scratch till 3 am of the day T, over 21 hours, during which more than 160 epochs of training in MPC can be executed (20 epochs take 2.5 hours in MPC as per Table 7). As training from scratch converges in 100-120 epochs, M_t will already have fairly good weights at this point. We further fine-tune M_t with day $T-6$ to day $T-1$ as training data, as values for the day $T-1$ have finished streaming at 12 am of the day T. We use a faster learning rate (0.004-0.007) for this fine-tuning, and complete more than 20 epochs in the 3 hours between 3 am and 6 am. Thus an up-to-date model M_t is ready at 6 AM of the day T, incorporating data from day $T-7$ to day $T-2$ using training from scratch, and day $T-6$ to $T-1$ using fine-tuning. Figure 16 in the Appendix, shows a flowchart for this procedure to create a new model every day, to be used in that day's forecasts.

Test Day T	**Scratch Train** $T-6$ **to** $T-1$	**Scratch Train** $T-7$ **to** $T-2$, **Fine-tune** $T-6$ **to** $T-1$
7-Jan	31.816	29.443
10-Jan	38.646	38.331
15-Dec	44.014	53.659
7-Dec	49.160	40.946
24-Jan	52.639	49.626
15-Nov	55.204	47.005
28-Nov	59.229	62.079
20-Nov	62.014	81.523
Average	49.090	50.327

Table 8: *Forecasting RMSE with practical training procedure in MPC, combining train from scratch and fine-tuning*

Table 8 shows the forecasting RMSE for our test days T using this revised training procedure. The second column repeats the RMSE values for GCN-LSTM from Table 2, where the train from scratch was used (100 epochs, learning rate 0.01, day $T-6$ to $T-1$ as train data). The third column shows RMSE for the revised procedure combining train from scratch (100 epochs, learning rate 0.01, day $T-7$ to day $T-2$ as train data) with fine-tuning (20 epochs, learning rate 0.007, day $T-6$ to day $T-1$ as train data). The revised procedure can handle streaming data in MPC where the original procedure lags by many hours, while their forecasting RMSE values show negligible differences.

8 CONCLUSION AND FUTURE WORK

This paper shows that if ML model architecture, parameters, and training procedures are carefully co-designed considering MPC overhead at every step then the benefits of highly accurate ML predictions can be kept intact while additionally ensuring strong privacy guarantees with MPC. Faster MPC algorithms for floating-point arithmetic will further improve the training latency [7, 52]. Our work will be extended in the future to handle larger geographical areas. Extending the threat model to handle malicious servers will also be explored, based on whether floating point MPC protocols and tools become available for the malicious setting.

ACKNOWLEDGMENTS

We acknowledge the help from Delhi Integrated Multi-modal Transport Systems (DIMTS), who allowed us to instrument their buses with sensors to collect the data used in this paper. We also thank Department of Science and Technology (DST) for funding the PM monitoring device manufacturing and deployment, through SERB-IMPRINT grant.

REFERENCES

[1] Ismi Abidi, Ishan Nangia, Paarijaat Aditya, and Rijurekha Sen. 2022. Privacy in Urban Sensing with Instrumented Fleets, Using Air Pollution Monitoring As A Usecase. In *NDSS*.

[2] Gergely Acs and Claude Castelluccia. 2014. A case study: Privacy preserving release of spatio-temporal density in paris. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1679–1688.

[3] Matthew D. Adams and Denis Corr. 2019. A Mobile Air Pollution Monitoring Data Set. *Data* 4, 1 (2019). https://doi.org/10.3390/data4010002

[4] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and Tong Tong Wu. 2016. I-pic: A platform for privacy-compliant image capture. In *Proceedings of the 14th annual international conference on mobile systems, applications, and services*. 235–248.

[5] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 901–914.

[6] Andreea Badicu, George Suciu, Mihaela Balanescu, Marius Dobrea, Andrei Birdici, Oana Orza, and Adrian Pasat. 2020. PMs concentration forecasting using ARIMA algorithm. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. 1–5. https://doi.org/10.1109/VTC2020-Spring48590.2020.9129390

[7] Marina Blanton, Michael T. Goodrich, and Chen Yuan. 2023. Secure and Accurate Summation of Many Floating-Point Numbers. *PoPETs* (2023).

[8] Théodore Bluche, Maël Primet, and Thibault Gisselbrecht. 2020. Small-Footprint Open-Vocabulary Keyword Spotting with Quantized LSTM Networks. https://doi.org/10.48550/ARXIV.2002.10851

[9] Henry Carter, Charles Lever, and Patrick Traynor. 2014. Whitewash: Outsourcing garbled circuit generation for mobile devices. In *Proceedings of the 30th Annual Computer Security Applications Conference*. 266–275.

[10] Guoguo Chen, Carolina Parada, and Tara N. Sainath. 2015. Query-by-example keyword spotting using long short-term memory networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5236–5240. https://doi.org/10.1109/ICASSP.2015.7178970

[11] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. 2019. Multi-Label Image Recognition With Graph Convolutional Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5172–5181. https://doi.org/10.1109/CVPR.2019.00532

[12] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. 2012. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *Cryptographers' Track at the RSA Conference*. Springer, 416–432.

[13] IIT Delhi. 2020. Particulate Matter Dataset Collected with IoT devices on Delhi public buses. Retrieved Mar 7, 2023 from https://www.cse.iitd.ac.in/pollutiondata

[14] W. Dong, V. Dave, L. Qiu, and Y. Zhang. 2011. Secure friend discovery in mobile social networks. In *2011 Proceedings IEEE INFOCOM*.

[15] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284.

[16] Ahmed Roushdy Elkordy, Jiang Zhang, Yahya H. Ezzeldin, Konstantinos Psounis, and Salman Avestimehr. 2023. How Much Privacy Does Federated Learning with Secure Aggregation Guarantee? *PoPETs* (2023).

[17] Jakob Eriksson, Lewis Girod, Bret Hull, Ryan Newton, Samuel Madden, and Hari Balakrishnan. 2008. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services* (Breckenridge, CO, USA) *(MobiSys '08)*. Association for Computing Machinery, New York, NY, USA, 29–39. https://doi.org/10.1145/1378600.1378605

[18] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. A survey of data augmentation approaches for NLP. *arXiv preprint arXiv:2105.03075* (2021).

[19] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting Gradients - How Easy is It to Break Privacy in Federated Learning?. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 1421, 11 pages.

[20] Pooja Gopu, Rama Ranjan Panda, and Naresh Kumar Nagwani. 2021. Time Series Analysis Using ARIMA Model for Air Pollution Prediction in Hyderabad City of India. In *Soft Computing and Signal Processing*, V. Sivakumar Reddy, V. Kamakshi Prasad, Jiacun Wang, and K. T. V. Reddy (Eds.). Springer Singapore, Singapore, 47–56.

[21] Yu Guan and Thomas Plötz. 2017. Ensembles of Deep LSTM Learners for Activity Recognition Using Wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2, Article 11 (jun 2017), 28 pages. https://doi.org/10.1145/3090076

[22] P. Hallgren, C. Orlandi, and A. Sabelfeld. 2017. PrivatePool: Privacy-Preserving Ridesharing. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*.

[23] Xi He, Graham Cormode, Ashwin Machanavajjhala, Cecilia M Procopiuc, and Divesh Srivastava. 2015. DPT: differentially private trajectory synthesis using hierarchical reference systems. *Proceedings of the VLDB Endowment* 8, 11 (2015), 1154–1165.

[24] W. Hedgecock, P. Völgyesi, A. Ledeczi, X. Koutsoukos, A. Aldroubi, A. Szalay, and A. Terzis. 2010. Mobile Air Pollution Monitoring Network. In *Proceedings of the 2010 ACM Symposium on Applied Computing* (Sierre, Switzerland) *(SAC '10)*. Association for Computing Machinery, New York, NY, USA, 795–796. https://doi.org/10.1145/1774088.1774253

[25] Julien Herzen, Francesco Lässig, Samuele Giuliano Piazzetta, Thomas Neuer, Léo Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan Kościsz, Dennis Bader, Frédérick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and Gaël Grosch. 2022. Darts: User-Friendly Modern Machine Learning for Time Series. *Journal of Machine Learning Research* 23, 124 (2022), 1–6. http://jmlr.org/papers/v23/21-1177.html

[26] S. L. Ho and M. Xie. 1998. The Use of ARIMA Models for Reliability Forecasting and Analysis. *Comput. Ind. Eng.* 35, 1–2 (oct 1998), 213–216. https://doi.org/10.1016/S0360-8352(98)00066-7

[27] Jingyu Hua, Yue Gao, and Sheng Zhong. 2015. Differentially private publication of general time-serial trajectory data. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 549–557.

[28] Yan Huang, David Evans, and Jonathan Katz. 2012. Private set intersection: Are garbled circuits better than custom protocols?. In *NDSS*.

[29] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster secure two-party computation using garbled circuits.. In *USENIX Security Symposium*, Vol. 201. 331–335.

[30] Manas Joshi, Arshdeep Singh, Sayan Ranu, Amitabha Bagchi, Priyank Karia, and Puneet Kala. 2022. FoodMatch: Batching and Matching for Food Delivery in Dynamic Road Networks. *ACM Trans. Spatial Algorithms Syst.* 8, 1, Article 6 (mar 2022), 25 pages. https://doi.org/10.1145/3494530

[31] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1651–1669.

[32] Hiroki Katayama, Shohei Yasuda, and Takashi Fuse. 2022. Traffic Density Based Travel-Time Prediction With GCN-LSTM. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)* (Macau, China). IEEE Press, 2908–2913. https://doi.org/10.1109/ITSC55140.2022.9922259

[33] Marcel Keller and Ke Sun. 2022. Secure Quantized Training for Deep Learning. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*. PMLR, 10912–10938.

[34] Cherry Khosla and Baljit Singh Saini. 2020. Enhancing performance of deep learning models with different data augmentation techniques: A survey. In *2020 International Conference on Intelligent Engineering and Management (ICIEM)*. IEEE, 79–85.

[35] Hidetoshi Kido, Yutaka Yanagisawa, and Tetsuji Satoh. 2005. An anonymous communication technique using dummies for location-based services. In *ICPS'05. Proceedings. International Conference on Pervasive Services, 2005.* IEEE, 88–97.

[36] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[37] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. CrypTen: Secure Multi-Party Computation Meets Machine Learning. arXiv:2109.00984 [cs.LG]

[38] Vladimir Kolesnikov and Ranjit Kumaresan. 2013. Improved OT Extension for Transferring Short Secrets. In *CRYPTO*.

[39] Jason Jingshi Li, Boi Faltings, Olga Saukh, David Hasenfratz, and Jan Beutel. 2012. Sensing the Air We Breathe: The Opensense Zurich Dataset. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (Toronto, Ontario, Canada) *(AAAI'12)*. AAAI Press, 323–325.

[40] Meng Li, Liehuang Zhu, Zijian Zhang, and Rixin Xu. 2017. Achieving differential privacy of trajectory data publishing in participatory sensing. *Information Sciences* 400 (2017), 1–13.

[41] Jie Liao, Wei Zhou, Fengji Luo, Junhao Wen, Min Gao, Xiuhua Li, and Jun Zeng. 2022. SocialLGN: Light Graph Convolution Network for Social Recommendation. *Inf. Sci.* 589, C (apr 2022), 595–607. https://doi.org/10.1016/j.ins.2022.01.001

[42] Yehuda Lindell. 2016. How To Simulate It - A Tutorial on the Simulation Proof Technique. Cryptology ePrint Archive, Paper 2016/046. (2016). https://eprint.iacr.org/2016/046 https://eprint.iacr.org/2016/046.
[43] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA (Proceedings of Machine Learning Research, Vol. 54)*. PMLR, 1273–1282.
[44] Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. 2008. Nericell: Rich Monitoring of Road and Traffic Conditions Using Mobile Smartphones *(SenSys '08)*. Association for Computing Machinery, New York, NY, USA, 323–336. https://doi.org/10.1145/1460412.1460444
[45] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE S&P*.
[46] Benjamin Mood, Lara Letaw, and Kevin Butler. 2012. Memory-efficient garbled circuit generation for mobile devices. In *International Conference on Financial Cryptography and Data Security*. Springer, 254–268.
[47] Sashank Narain and Guevara Noubir. 2019. Mitigating location privacy attacks on mobile devices using dynamic app sandboxing. *Proceedings on Privacy Enhancing Technologies* 2019, 2 (2019), 66–87.
[48] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. 2013. Privacy-preserving matrix factorization. In *CCS*.
[49] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. 2013. Privacy-Preserving Ridge Regression on Hundreds of Millions of Records. In *IEEE S&P*.
[50] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2019. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. *CoRR* abs/1905.10437 (2019). arXiv:1905.10437 http://arxiv.org/abs/1905.10437
[51] Elena Pagnin, Gunnar Gunnarsson, Pedram Talebi, Claudio Orlandi, and Andrei Sabelfeld. 2019. TOPPool: Time-aware Optimized Privacy-Preserving Ridesharing. *Proceedings on Privacy Enhancing Technologies* (2019).
[52] Deevashwer Rathee, Anwesh Bhattacharya, Divya Gupta, Rahul Sharma, and Dawn Song. 2023. Secure Floating-Point Training. In *USENIX Security*.
[53] Deevashwer Rathee, Anwesh Bhattacharya, Rahul Sharma, Divya Gupta, Nishanth Chandran, and Aseem Rastogi. 2022. SecFloat: Accurate Floating-Point meets Secure 2-Party Computation. In *IEEE S&P*.
[54] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. 2021. SiRnn: A Math Library for Secure RNN Inference. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 1003–1020.
[55] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS*.
[56] S. Stirbys, O. A. Nabah, P. Hallgren, and A. Sabelfeld. 2017. Privacy-Preserving Location-Proximity for Mobile Apps. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*.
[57] Saurabh Tiwari, Ravi Bhandari, and Bhaskaran Raman. 2020. *RoadCare: A Deep-Learning Based Approach to Quantifying Road Surface Quality*. Association for Computing Machinery, New York, NY, USA, 231–242. https://doi.org/10.1145/3378393.3402284
[58] Christian Vaas, Mohammad Khodaei, Panos Papadimitratos, and Ivan Martinovic. 2018. Nowhere to hide? Mix-zones for private pseudonym change using chaff vehicles. In *2018 IEEE Vehicular Networking Conference (VNC)*. IEEE, 1–8.
[59] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *PoPETs* (2019).
[60] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *PoPETs* (2021).
[61] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge Graph Convolutional Networks for Recommender Systems. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 3307–3313. https://doi.org/10.1145/3308558.3313417
[62] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.
[63] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. 2022. Piranha: A GPU Platform for Secure Computation. https://eprint.iacr.org/2022/892
[64] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. 2020. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478* (2020).
[65] P.J. Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560. https://doi.org/10.1109/5.58337
[66] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph Convolutional Networks for Text Classification *(AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Article 905, 8 pages. https://doi.org/10.1609/aaai.v33i01.33017370
[67] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov. 2021. See through Gradients: Image Batch Recovery via GradInversion. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 16332–16341. https://doi.org/10.1109/CVPR46437.2021.01607
[68] Chak Fai Yuen, Abhishek Pratap Singh, Sagar Goyal, Sayan Ranu, and Amitabha Bagchi. 2019. Beyond Shortest Paths: Route Recommendations for Ride-Sharing. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 2258–2269. https://doi.org/10.1145/3308558.3313465
[69] Tahmina Zebin, Matthew Sperrin, Niels Peek, and Alexander J. Casson. 2018. Human activity recognition from inertial sensor time-series using batch normalized deep LSTM recurrent networks. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 1–4. https://doi.org/10.1109/EMBC.2018.8513115
[70] Jiansen Zhao, Zhongwei Yan, Xinqiang Chen, Bing Han, Shubo Wu, and Ranxuan Ke. 2022. k-GCN-LSTM: A k-hop Graph Convolutional Network and Long–Short-Term Memory for ship speed prediction. *Physica A: Statistical Mechanics and its Applications* 606 (2022), 128107. https://doi.org/10.1016/j.physa.2022.128107
[71] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. 2021. Federated learning on non-IID data: A survey. *Neurocomputing* 465 (2021), 371–390.

A APPENDIX

A.1 MPC Implementation details for training

With the GCN-LSTM model shown in Figure 12, we provide a condensed version of the forward pass of the model expressed in our APIs (built on SecFloat) in Figure 13. Recall that SecFloat only provides generic floating-point operations and we built the ML-specific APIs on top of SecFloat. The corresponding code for the backward pass is shown in Figure 14. To support a new model, one only needs to write code similar to what is shown in Figure 13 and Figure 14 by hand.

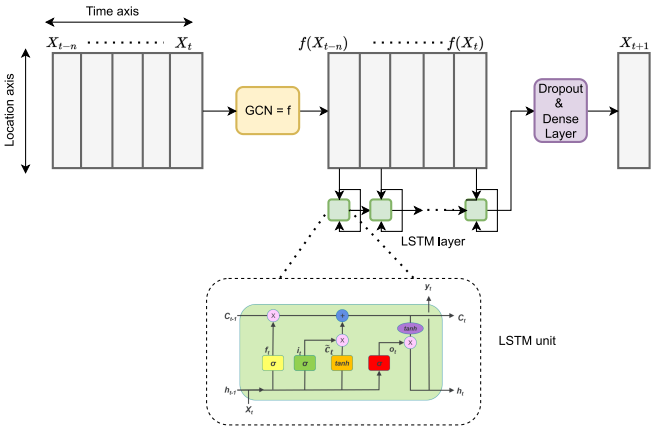


Figure 12: *GCN-LSTM Architecture*

```
void forward (...)
    GCN-Layer1 (adj1, ...)
    GCN-Layer2 (adj2, ...)
    LSTM (...)
    Linear-Layer (...)
    output = Sigmoid (...)
    Compute-Loss (output, labels)
```

Figure 13: Forward pass of GCN-LSTM

```
void backward (...)
    Error-Gradient (output, labels)
    ElemWiseMul (...)
    Linear-Backprop (...)
    Backprop-through-time (...)
    GCN-Backprop (adj2, ...)
    GCN-Backprop (adj1, ...)
```

Figure 14: Backward pass of GCN-LSTM

A.2 Temporal buckets for forecasting

We show the sample time series for one day (Dec 7, 2020) for some grid locations in Table 9, which forms the input to our ML model to train and evaluate forecasting.

lat-lon grid	tb-1	tb-2	tb-3	tb-4	tb-5	tb-6
0, 21	0.0	0.0	0.0	0.0	0.0	441.8
0, 22	420.1	502.7	349.3	263.3	261.7	459.3
1, 19	377.6	502.0	394.1	240.8	0.0	474.3
1, 20	431.0	546.9	392.3	231.6	0.0	456.6

Table 9: Snippet of time-series for a day (Dec 7, 2020), where tb denotes temporal buckets of 3 hours each, totaling 18 hours for a day. The remaining 6 hours 12 am - 6 am have too few sensor readings to be useful.

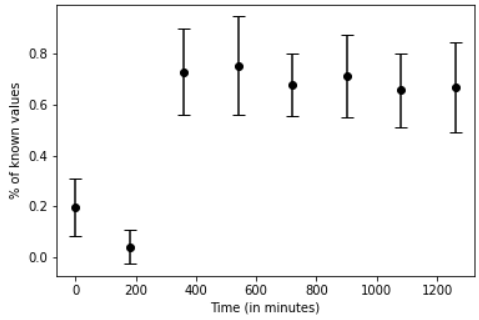


Figure 15: Temporal data distribution in a day

We use 6 time windows for each day, each window being of 3 hours. The remaining two windows between 12 am - 6 am have very few sensor readings, as the buses are idle during the night. Figure 15 shows the hour-wise PM data distribution, where these 6 hours of data are scarce. Hence we omit those 2 windows from our time-series modeling.

A.3 Practical Private Training Procedure

Figure 16 shows our practical private training procedure to create a new model every day, to be used in that day's forecasts.

We start training M_t with randomly initialized weights at 6 am on the day $T-1$, using buffered values from day $T-7$ to $T-2$ as training data and a learning rate of 0.01. We continue this scratch train till 3 am of the day T, over 21 hours, during which more than 160 epochs of training in MPC can be executed (20 epochs take 2.5 hours in MPC as per Table 7). As training from scratch converges in 100-120 epochs, M_t will already have fairly good weights at this point. We further fine-tune M_t with day $T-6$ to day $T-1$ as training data, as values for the day $T-1$ have finished streaming at 12 am of

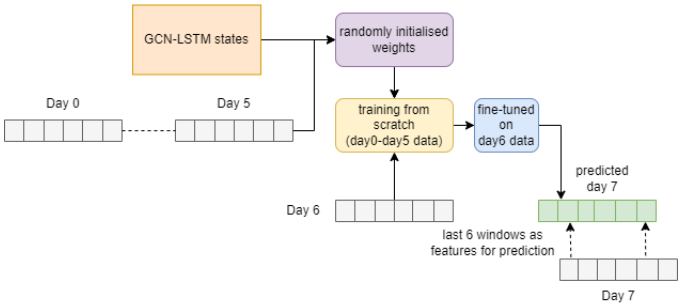


Figure 16: *Practical training procedure in MPC, combining train-from-scratch and fine-tune*

the day T. We use a learning rate (0.004-0.007) for this fine-tuning, and complete more than 20 epochs in the 3 hours between 3 am and 6 am. Thus an up-to-date model M_t is ready at 6 AM of the day T, incorporating data from day $T-7$ to day $T-2$ using the scratch train, and day $T-6$ to $T-1$ using fine-tune.