

Federated Learning at the Edge for Smart City Sensing Systems

Raed Abdel-Sater

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
for the Ph.D. Degree in Information and Systems Engineering at
Concordia University
Montreal, QC, Canada

November 2025

© Raed Abdel-Sater, 2025

**CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES**

This is to certify that the thesis prepared

By: Raed Abdel Sater

Entitled: Federated Learning at the Edge for Smart City Sensing Systems

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy Information and Systems Engineering

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____	Chair
Ayhan Ince	
_____	External Examiner
Wassim Bouachir	
_____	Arm's Length Examiner
Jun Yan	
_____	Examiner
Nizar Bouguila	
_____	Examiner
Amr Youssef	
_____	Thesis Supervisor (s)
Abdessamad Ben Hamza	

Approved by _____

Chun Wang Chair of Department or Graduate Program Director

November 18, 2025

Date of Defence

Mourad Debbabi Dean, Gina Cody School

Abstract

Federated Learning at the Edge for Smart City Sensing Systems

Raed Abdel-Sater

Ph.D. Degree in Information and Systems Engineering, Concordia University, 2025

The proliferation of Internet of Things sensors in smart buildings and Internet of Vehicles networks generates privacy-sensitive time series data at unprecedented scales, posing significant challenges for traditional learning paradigms. Centralized learning approaches are hampered by privacy risks and difficulties in handling heterogeneous data, while federated learning incurs server computational bottlenecks. This thesis presents a comprehensive framework progressing from privacy-preserving edge analytics to fully decentralized coordination for anomaly detection and long-term times series forecasting. The thesis contributions are threefold. First, we introduce FSLSTM, a federated stacked long short-term memory approach for anomaly detection across heterogeneous building sensors (HVAC, lighting, access control). Without centralizing raw telemetry, FSLSTM achieves competitive performance on various real-world datasets while converging twice faster compared to centralized baselines and reducing communication costs by 45%. Second, building on this foundation, we develop FedTime, a federated large language model framework that adapts Llama-2 for long-term time series forecasting. Through cluster-aware federation, channel independence, patching and task alignment techniques, FedTime outperforms strong forecasting baselines across several benchmarks including a production-scale electric vehicle charging data. The FedTime framework demonstrates thrice faster convergence than centralized training, while reducing communication overhead by 50%. Third, to eliminate residual trust assumptions, we propose BLEND, a blockchain-enhanced decentralized system that replaces server-based aggregation with smart contract orchestration. Through a novel Proof-of-Forecast consensus mechanism that validates updates based on predictive accuracy, BLEND yields strong forecasting performance while reducing network overhead by 63% and demonstrating resilience to 20% Byzantine participants. Collectively, these contributions establish a complete pipeline from edge sensing to decentralized learning, demonstrating that privacy preservation, computational efficiency and decentralized trust can be achieved simultaneously without sacrificing predictive performance.

Table of Contents

List of Tables	vii
List of Figures	viii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.2.1 Anomaly Detection in Smart Systems	2
1.2.2 Federated Long-Term Time Series Forecasting	3
1.2.3 Consensus and Coordination without a Central Server	3
1.3 Objectives	4
1.4 Literature Review	4
1.5 Preliminaries	7
1.5.1 Time Series Fundamentals and Notation	7
1.5.2 Forecasting Strategies	8
1.5.3 Reversible Instance Normalization (RevIN)	9
1.5.4 Sequence Models	10
1.5.5 Federated Learning Setup	12
1.5.6 Server-Side Adaptive Optimization	13
1.5.7 Differential Privacy in Federated Learning	14
1.5.8 Parameter-Efficient Fine-Tuning (PEFT): LoRA and QLoRA	15
1.5.9 Decentralized Coordination	16
1.6 Applications	18
1.7 Overview and Contributions	19
1.8 Summary of Publications	21
1.8.1 Thesis Research	21
1.8.2 Non-thesis Research	21

2	Federated Learning for Anomaly Detection in Smart Buildings	22
2.1	Introduction	23
2.2	Related Work	26
2.3	Method	27
2.3.1	Multi-Task Learning	29
2.3.2	LSTM	31
2.3.3	Proposed Framework	32
2.4	Experiments	35
2.4.1	Results	38
2.4.2	Ablation Study	46
2.5	Discussion	48
3	Federated Large Language Model for Long-Term Time Series Forecasting	51
3.1	Introduction	51
3.2	Related Work	53
3.3	Method	54
3.3.1	Problem Statement and Approach Overview	54
3.3.2	Proposed Learning Framework	55
3.4	Experiments	59
3.4.1	Experimental Setup	59
3.4.2	Results and Analysis	62
3.4.3	Communication Overhead	65
3.4.4	Ablation Study	66
3.5	Discussion	68
4	Blockchain-Enhanced Network Decentralization with LLMs for Time Series Forecasting	70
4.1	Introduction	70
4.2	Related Work	72
4.3	Proposed BLEND Framework	74
4.3.1	System Overview	74
4.3.2	Proof-of-Forecast Consensus Protocol	77
4.3.3	Incentive Mechanism	77
4.3.4	Miner LLM Forecaster	80
4.3.5	Federated Optimization	83

4.4	Experiments	83
4.4.1	Experimental Setup	84
4.4.2	Results and Analysis	86
4.4.3	Federated Performance	90
4.4.4	Communication Overhead	92
4.4.5	Ablation Study	92
4.5	Discussion	96
5	Conclusions and Future Work	98
5.1	Contributions of the thesis	98
5.1.1	A Federated Learning Approach to Anomaly Detection in Smart Buildings	98
5.1.2	A Federated Large Language Model for Long-Term Time Series Forecasting	99
5.1.3	BLEND: Blockchain-Enhanced Network Decentralisation with Large Lan- guage Models for Long-Term Time-Series Forecasting	99
5.2	Limitations	100
5.3	Future Work	101
5.3.1	Personalisation, Fairness, and Transfer Across Sites	101
5.3.2	Continual, Streaming, and Uncertainty-Aware Learning	101
5.3.3	Multimodal Context and LLM–Time-Series Hybrids	101
5.3.4	Efficiency for Edge Deployment	102
5.3.5	MLOps at Scale and Human-in-the-Loop Operation	102
5.3.6	Decentralized Coordination: Extending BLEND	102
	References	103

List of Tables

1.1	Summary of symbols.	17
2.1	Notation.	30
2.2	FSLSTM vs. baselines on Sensors (classification) and Energy Usage (regression) . . .	39
2.3	Collective and contextual anomaly detection comparison.	45
2.4	Ablation: FSLSTM layers on sensors & energy datasets	48
3.1	Summary statistics of benchmark datasets.	61
3.2	FedTime Long-horizon Forecasting Performance Across Datasets	63
3.3	FedTime vs. Federated Baselines Prediction Performance	65
3.4	Latency performance evaluation of various FedTime model versions across different GPU setups with and without PEFT. Latency times, measured in milliseconds (ms), showcase the efficiency of response in edge (E) and server (S) configurations. Boldface numbers indicate the best latency performance on edge devices.	68
4.1	Summary statistics of benchmark datasets.	85
4.2	Forecasting performance (MSE and MAE) on seven benchmark datasets for varying prediction horizons $T \in \{96, 192, 336, 720\}$. Best results are bolded and second-best are <u>underlined</u>	86
4.3	Performance evaluation of BLEND compared to federated baselines. Average values are presented as (MSE/MAE).	91

List of Figures

2.1	IoT-enabled smart building.	28
2.2	Federated learning system for anomaly detection.	28
2.3	LSTM block architecture.	32
2.4	Federated stacked LSTM architecture	33
2.5	Distribution of sensors' categories.	37
2.6	ROC curves and AUC (Sensors Event Log)	41
2.7	Learning curves: LSTM vs. FSLSTM (Sensors Event Log)	43
2.8	Convergence time vs. number of sensors (FSLSTM/FGRU/LSTM)	44
2.9	Actual vs. predicted energy (FSLSTM)	46
2.10	Anomaly detection on three sensor types (FSLSTM)	47
2.11	Communication cost: FSLSTM vs. baselines (50 epochs)	48
3.1	FedTime architecture (DPO alignment; RevIN fine-tuning)	56
3.2	Long-term forecasting vs. lookback L (horizon $T = 720$)	64
3.3	Learning curves: centralized LLaMA vs. FedTime	64
3.4	ACN energy delivery distribution based on disconnection time.	66
3.5	Communication overhead comparison between FedTime and baseline methods.	66
3.6	Predicted vs. actual energy (Caltech weekdays; FedTime variants)	67
4.1	BLEND training round. (1) Publisher assigns forecasting task; (2) Oracle fetches signals (e.g., traffic, weather) and broadcasts with the current global LLM; (3) Miners fine-tune on private data and submit signed updates; (4) Evaluators audit via proof-of-forecast (PoF) and assign utility scores; (5) Byzantine-tolerant vote selects top update and FedAdam aggregates it into global model. (6) New model and audit trail sealed in blockchain, triggering next round.	75

4.2	The LLM forecaster in BLEND consists of three distinct phases. The offline Model Alignment phase (Phase 1, top) first adapts a pre-trained LLM for forecasting using a two-stage process of DEITA and DPO-TS to produce a specialized, task-aligned model. The online inference process begins with Input Representation (Phase 2, bottom left), where time-series and contextual data are tokenized and assembled into a unified prompt. Finally, in the Forecasting Workflow (Phase 3, right), this prompt is fed to the frozen, task-aligned backbone with QLoRA adapters to generate the final forecast patch used in PoF scoring and aggregation.	81
4.3	Cross-dataset error distribution. Box plots show the MSE achieved by each model on the seven public benchmarks for prediction horizons $T \in \{96, 192, 336, 720\}$. The box spans the IQR; whiskers extend to the non-outlier range; the red line marks the median; and the green diamond denotes the mean. BLEND records the lowest median and mean MSE while exhibiting the narrowest IQR, indicating both superior performance and robustness compared with Transformer-, LLM-, and linear-based baselines.	88
4.4	Convergence trajectories comparing centralized standalone LLM, vanilla federated LLM, and BLEND over 60 training epochs on the Electricity dataset. BLEND converges faster and to a lower final error by filtering out unreliable updates via its consensus protocol.	89
4.5	Byzantine fault tolerance evaluation of BLEND under varying concentrations of malicious nodes (3, 6, and 9 bad nodes) over 60 training epochs on the ETTh2 dataset, demonstrating BLEND’s resilience against adversarial participants.	90
4.6	MSE over 100 epochs on the Traffic dataset under data poisoning. Federated learning degrades from a clean-data MSE of 0.167 to 0.324 when 30% of clients submit poisoned updates. In contrast, BLEND under the same attack converges to an MSE of 0.201, recovering over 35% of the degraded accuracy.	91
4.7	Communication overhead (in MegaBytes) for BLEND, centralized LLM training, and federated LLM training on the Electricity dataset, over varying numbers of vehicles V and participation ratios ρ . BLEND achieves up to 63% reduction in data transfer compared to baseline methods, underscoring its scalability and efficiency in IoV federated learning.	93

4.8	Ablation study of BLEND’s model alignment components across multiple evaluation dimensions. (a-c) Forecasting trajectories for three representative load regimes on the Electricity dataset comparing baseline (no alignment) and complete BLEND (DEITA + DPO-TS) configurations against ground truth. (d) Performance comparison across evaluation metrics (RMSE, MAE, MSE, R^2) for all alignment configurations, demonstrating progressive improvements with baseline (red), DEITA only (orange), DPO-TS only (blue), and combined alignment (green). (e) Dataset-wise MSE comparison showing consistent improvements across all seven benchmark datasets. (f) Training convergence analysis over 100 epochs, illustrating accelerated convergence and superior final performance with alignment techniques. (g) Statistical significance analysis through box plots of MSE distributions across multiple experimental runs, with p-values indicating significant differences between configurations.	94
4.9	Ablation of the Oracle agent: (a-c): trajectories under three load regimes on the Electricity dataset. (d) Metric-wise improvements; (e) error-density comparison; (f) Improvement Matrix; (g) Oracle impact across traffic patterns vs. news relevance. . . .	95

Introduction

In this chapter, we establish the foundation of our research. We begin with the motivation for developing privacy-preserving distributed learning systems, followed by the problem statement addressing federated learning challenges at the edge. We then present our objectives for anomaly detection and time series forecasting, and review relevant literature in federated learning, large language models, and blockchain coordination. Essential preliminaries cover time series fundamentals, federated optimization, sequence models, and consensus mechanisms. The chapter concludes with an overview of the thesis structure and a summary of the key contributions.

1.1 Motivation

The rapid electrification at scale of transportation and smart systems in addition to the proliferation of public and private charging infrastructure are reshaping power distribution and mobility services. Accurate forecasting of spatiotemporal charging demand underpins grid balancing, dynamic pricing, siting, and maintenance planning for sensors in buildings; yet the data and operating constraints in these systems complicate the design of effective learning pipelines. Mobility traces and energy consumption profiles are privacy-sensitive; data across stations are heterogeneous and non independent and identically distributed (non-IID); edge devices operate under tight compute, memory, and networking budgets; and centralized training introduces both a communication bottleneck and a single point of failure. Conventional federated learning alleviates data movement but still relies on a server-centric architecture, which can throttle scalability and resilience in large deployments.

This thesis is motivated by the need for a principled, end-to-end approach that achieves high predictive utility while respecting privacy, resource, and trust constraints. Two observations guide the design. First, large language models (LLMs) can be adapted to time series forecasting when their tokenization and objectives are aligned with numerical sequences, but such models must be made parameter- and communication-efficient to run where the data reside. Second, coordination should not hinge on a single aggregator: auditability, liveness, and robustness can be strengthened by recording verifiable updates on a permissioned ledger and tying block acceptance to measurable forecasting quality. Building on these observations, the thesis develops a unified framework that (i) learns from heterogeneous sensor streams without exporting raw data by casting anomaly detection as a federated multi-task problem; (ii) adapts pre-trained LLMs for long-term time series forecasting with channel-aware tokenization, patching, and alignment while minimizing communication and compute at the edge; and (iii) replaces server-based aggregation with a blockchain-mediated coordination layer in which smart contracts govern roles and a forecast-quality criterion determines model acceptance. In combination, these components are intended to bridge the gap between high-accuracy modeling and the practical realities of distributed energy systems, providing a path toward scalable, privacy-preserving, and operationally robust forecasting.

1.2 Problem Statement

This thesis studies learning from distributed time series data in smart systems under drift, resource limits, and auditability constraints. We address anomaly detection, long-horizon forecasting, and decentralized consensus as joint challenges in co-designing models, training protocols, and coordination rules that are robust to heterogeneity and verifiable through trusted incentive mechanism.

1.2.1 Anomaly Detection in Smart Systems

Given multivariate telemetry streams (e.g., charging stations, HVAC, lighting, occupancy, etc...), we aim to detect sensor faults and anomalies without centralizing raw data. The problem entails learning client-specific models that share global inductive biases while retaining site-specific residuals, under scarce labels and drifting operating conditions. Key constraints include irregular samples, seasonality, and concept shift due to retrofits or policy changes. We consider distributed learning over K clients (e.g., stations, buildings, or microoperators) holding local datasets $\{\mathcal{D}_k\}_{k=1}^K$ that must remain on-device. The global objective is

$$\min_{\theta \in \Theta} F(\theta) \triangleq \sum_{k=1}^K p_k \mathbb{E}_{(x,y) \sim \mathcal{D}_k} [\ell(f_\theta(x), y)], \quad (1.1)$$

with $p_k \propto |\mathcal{D}_k|$, subject to (i) data heterogeneity (covariate/label shift), (ii) limited client compute and memory, (iii) intermittent connectivity and communication budgets, and (iv) privacy and auditability constraints.

1.2.2 Federated Long-Term Time Series Forecasting

We target horizon- H forecasting ($H \gg 1$) over mixed-frequency, multi-channel demand signals. The challenge is to fine-tune a pre-trained LLM for numeric sequence modeling under parameter, activation, and bandwidth budgets, while mitigating non-IID drift across clients. Desired properties include (i) channel-aware tokenisation, (ii) patching/segmentation for long contexts, (iii) alignment via instruction tuning for forecasting tasks, and (iv) robust aggregation under heterogeneous participation. Each client $k \in \mathcal{K}$ holds pairs $(\mathbf{X}_{t-L+1:t}^{(k)}, \mathbf{Y}_{t+1:t+H}^{(k)})$ with $\mathbf{X}_{t-L+1:t}^{(k)} \in \mathbb{R}^{L \times C}$ and $\mathbf{Y}_{t+1:t+H}^{(k)} \in \mathbb{R}^{H \times C'}$, $H \gg 1$. The task is to learn

$$f : \mathbb{R}^{L \times C} \rightarrow \mathbb{R}^{H \times C'}$$

that minimizes a horizon-appropriate loss aggregated across clients,

$$\min_f \sum_{k=1}^K p_k \mathbb{E}_{(X,Y) \sim \mathcal{D}_k} [\ell_{\text{reg}}(f(X), Y)], \quad p_k \propto |\mathcal{D}_k|.$$

Constraints include non-IID client distributions, mixed sampling rates, limited device resources, intermittent participation, and bounded uplink/downlink. The objective is stable long-horizon accuracy without centralizing raw data.

1.2.3 Consensus and Coordination without a Central Server

We consider iterative coordination over rounds $r = 1, 2, \dots$ among participants $\mathcal{M}_r \subseteq \mathcal{K}$. In round r , participant i proposes an update $u_i^{(r)}$ that an evaluation procedure maps to a quality scalar $q_i^{(r)}$ (e.g., a validation loss). The consensus problem is to select and finalize one update

$$u_{\star}^{(r)} \in \arg \min_{i \in \mathcal{M}_r} q_i^{(r)}$$

subject to a quorum constraint for finality (signature threshold) and to record the decision immutably for audit. A reward mapping $R_i^{(r)}$ may align incentives (e.g., accuracy-based payouts and evaluator correctness), but the core is deterministic selection and verifiable commitment of $u_{\star}^{(r)}$ under partial trust and bandwidth limits.

1.3 Objectives

Building on the three problem statements above, this thesis pursues four complementary objectives that co-design learning models, training protocols, and coordination rules for distributed time-series intelligence.

- **Privacy-preserving anomaly detection on non-IID sensor streams.** Formulate anomaly detection over multivariate sequences $\mathbf{X}_{1:T}^{(k)}$ with local labels $\mathbf{A}_{1:T}^{(k)}$ under client heterogeneity, missingness, and drift [1]. Develop a federated, multi-task sequence model that learns shared inductive biases and client-specific residuals, and establish a detection risk and evaluation protocol suitable for scarce labels and streaming constraints.
- **Federated LLMs for Long-term forecasting.** Adapt a pre-trained language model to numeric time series with channel-aware tokenisation, temporal patching for long contexts, and task alignment (instruction/preference tuning) subject to edge budgets [2]. Specify a cluster-aware federated objective $\min_f \sum_k p_k \mathbb{E}_{(X,Y) \sim \mathcal{D}_k} [\ell_{\text{reg}}(f(X), Y)]$ and parameter-efficient fine-tuning so that multi-step forecasting over horizons $H \gg 1$ remains accurate and communication-efficient.
- **Decentralized coordination without a central server.** Design a permissioned-ledger layer that replaces server aggregation with verifiable, accuracy-driven consensus. Formalize a Proof-of-Forecast (PoF) rule that deterministically selects $u_{\star}^{(r)} \in \arg \min_{i \in \mathcal{M}_r} q_i^{(r)}$ (subject to quorum) and define a transparent reward mapping that aligns miner and evaluator behavior with global forecasting objectives while preserving auditability.
- **Systematic evaluation and reproducibility.** Establish metrics and protocols that jointly assess forecasting performance (i.e. MAE, MSE, RMSE, etc...), convergence (rounds/time-to-target), communication/computation cost (bytes per round, on-device memory/latency), and robustness (faults, poisoning/outliers). Conduct ablations on clustering, tokenisation/patching, optimizer and participation settings, and document implementation details for reproducible experiments.

1.4 Literature Review

Anomaly Detection in Smart Systems. Anomaly detection in smart and distributed systems spans point, contextual, and collective anomalies, with goals ranging from fault diagnosis to

intrusion detection. Deep sequence models learn normal patterns and flag departures via prediction residuals (forecast-based detection) or reconstruction errors (autoencoder-based detection) [3, 4]. In operational environments, *missingness*, *irregular sampling*, and *concept drift* (seasonal/operational changes) complicate static detectors. Several centralized LSTM-style detectors show promise on logs, in-vehicle networks, and industrial plants [5–7], but centralization raises privacy concerns and creates bandwidth/latency bottlenecks. Federated variants therefore distribute training across devices/sites, aggregating models rather than data. For example, federated intrusion profiles for IoT devices [8], server-side detection of poisoned local updates [9], probabilistic aggregation schemes [10], and multi-task formulations for network traffic [11] and activity recognition [12] demonstrate that anomaly detection benefits from personalization and robust aggregation when data are non-IID and labels are scarce. These observations support the thesis choice to cast anomaly detection as a multi-task, privacy-preserving sequence modeling problem under federated constraints.

Federated Long-Term Time Series Forecasting. Early deep time-series forecasters relied on recurrent architectures (LSTM/GRU) to model temporal dependencies and nonlinearity, with successful applications to operations logs, industrial telemetry, and energy systems. The advent of Transformers prompted a shift toward attention mechanisms and long-context modeling. Representative LTSF models include Pyraformer, which introduces pyramidal attention with inter/intra-scale connections to balance capacity and cost [13]; Informer, which proposes ProbSparse attention to reduce $\mathcal{O}(T^2)$ complexity while preserving accuracy at long horizons [14]; Autoformer, which couples seasonal–trend decomposition with auto-correlation to stabilize long-range prediction [15]; and FEDformer, which leverages Fourier/Wavelet components for efficient global pattern extraction [16]. PatchTST reframes sequences as subseries-level patches (tokens) and advocates channel-independence (CI)—a design that shares embeddings and Transformer weights across univariate channels to preserve local semantics while improving efficiency and robustness [17]. More recently, iTransformer inverts the series–token relationship to attend across entire univariate series, aiming to capture inter-series structure with a simpler feed-forward backbone [18].

Pre-trained LLMs have begun to appear as backbones for numeric time series: GPT4TS adapts a GPT-style decoder to forecasting [19]; Time-LLM explores prompt design and instruction-style supervision for temporal tasks [20]; and LLM4TS investigates adapting a pre-trained language model to time-series via appropriate tokenization and alignment [21]. In all such approaches, three design fronts recur: (i) representation (context length, patching/segmentation, and CI vs. channel interaction), (ii) parameter-efficient adaptation (e.g., low-rank adapters) to fit edge mem-

ory/compute budgets, and (iii) task alignment (instruction tuning or preference-style objectives) to encourage numerically faithful forecasting with long horizons. These concerns motivate the thesis’s focus on CI patching and alignment for LLM-based forecasters deployed in federated or decentralized regimes. Federated learning [22] enables collaborative training while keeping raw data local; standard FedAvg minimizes a weighted sum of client objectives via local SGD with periodic aggregation. However, non-IID data induce client drift, slowing convergence and potentially harming generalization. Remedies include proximal regularization, personalized or multi-task objectives, meta-learning, and clustering to group similar clients [10–12, 23]. Communication constraints motivate gradient/parameter compression and secure aggregation [24]; privacy concerns motivate differential privacy (DP) noise addition and clipping; and adversarial robustness calls for defenses against backdoor/poisoning attacks [25]. In the LTSF domain, cluster-aware personalization and parameter-efficient fine-tuning have proved effective for long-term accuracy with reduced uplink cost, as demonstrated in federated LLM forecasters such as FedTime [2]. A recurring limitation of classic cross-silo FL is its *central aggregator*, which remains a performance and trust bottleneck—an issue addressed later via blockchain-backed coordination.

Consensus and Coordination without a Central Server. Permissioned blockchains offer an immutable audit layer, programmable coordination via smart contracts, and quorum-based finality, making them attractive to replace the central FL server in low-trust settings. Foundational works [26, 27] established the primitives for decentralized consensus and tamper-evident ledgers. BlockFL demonstrates how a blockchain can log and orchestrate federated updates [28]; DeepChain adds tokenized incentives for participation [29]; Biscotti secures signed, auditable model updates [30]; and on-chain contribution accounting/payouts have been explored in [31, 32]. Many such systems, however, treat the ledger chiefly as a passive registry and decouple block production from *model utility*. In contrast, “useful-work”-style consensus links finality to a measurable training objective. The thesis’s BLEND framework instantiates this idea with *Proof-of-Forecast* (PoF), where an update is eligible for commitment only if it achieves superior predictive accuracy on a fixed validation slice—thereby aligning consensus with the learning goal while preserving auditability and incentives.

Agentic Workflows with LLMs. Agentic workflows-rooted in multi-agent systems have resurged with LLMs that can plan, call tools, and coordinate roles [33]. Typical systems assign specialized responsibilities (e.g., planner, coder, verifier) and use structured protocols to reduce error and improve sample efficiency. Most existing implementations remain centrally orchestrated, which limits applicability in cross-silo settings with partial trust. BLEND pushes these ideas into a blockchain-native context: roles such as *Publisher*, *Miner*, *Evaluator*, and *Leader* are enforced

by smart contracts on a consortium ledger; proposals and audits are signed and recorded; and rewards/slashing align individual behavior with a global forecasting objective. At the modeling layer, LLMs are adapted to numeric sequences via channel-aware tokenization, CI patching, and task alignment (e.g., instruction-style tuning and preference-based objectives), echoing trends in GPT-style adaptation [19–21, 34, 35] while addressing edge-side budget and participation constraints. The literature indicates three converging needs for edge-native forecasting and anomaly detection: (i) long-context sequence modeling with resource-aware representations and parameter-efficient adaptation; (ii) collaboration under non-IID data with privacy, robustness, and bounded communication; and (iii) low-trust coordination with verifiable progress and incentive alignment. The thesis responds to these needs via a federated anomaly detection pipeline, a federated LLM forecaster for LTSF, and a blockchain-mediated coordination layer that ties consensus to predictive merit.

1.5 Preliminaries

This section establishes the technical background and notation used throughout the thesis. We first formalize multivariate time series and leakage-safe evaluation, then review forecasting strategies and sequence models (recurrent and attention-based). We summarize the federated learning (FL) objective under non-IID distributions together with standard privacy primitives, and we recall server-side adaptive optimization and parameter-efficient fine-tuning that make LLMs practical at the edge. Finally, we give the minimal machinery for decentralized coordination (quorum, update selection, and a normalized score) reused later.

1.5.1 Time Series Fundamentals and Notation

We consider multivariate sequences $\mathbf{X} \in \mathbb{R}^{T \times C}$ with time index $t \in \{1, \dots, T\}$ and channels $c \in \{1, \dots, C\}$. A forecasting task maps a look-back window of size L to a horizon H . A multivariate time series of length T with C channels is written as

$$\mathbf{X}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)^\top \in \mathbb{R}^{T \times C}, \quad \mathbf{x}_t \in \mathbb{R}^C.$$

Given a look-back window of length L and a forecast horizon H , a forecaster

$$f_\theta : \mathbb{R}^{L \times C} \longrightarrow \mathbb{R}^{H \times C'}$$

maps each window $\mathbf{X}_{t-L+1:t}$ to future targets $\mathbf{Y}_{t+1:t+H}$, typically with $C' = C$ for multi-output prediction. We denote windows by

$$\mathbf{w}_t \triangleq \mathbf{X}_{t-L+1:t} \in \mathbb{R}^{L \times C}, \quad \hat{\mathbf{Y}}_{t+1:t+H} = f_\theta(\mathbf{w}_t).$$

Unless noted, time indices obey $L \leq t \leq T - H$ to avoid leakage from the future.

Leakage-safe temporal splits. Evaluation uses time-ordered partitions. A basic split places training, validation, test as

$$\text{train: } 1:t_1, \quad \text{val: } (t_1+1):t_2, \quad \text{test: } (t_2+1):T,$$

or a rolling-origin (walk-forward) protocol: for cut points $1 < t_1 < t_2 < \dots < t_m < T$, train on $1:t_j$ and test on $(t_j+1):t_{j+1}$, aggregating errors across folds. These protocols prevent target leakage and reflect deployment.

1.5.2 Forecasting Strategies

A multi-step forecaster maps a window $\mathbf{w}_t = \mathbf{X}_{t-L+1:t} \in \mathbb{R}^{L \times C}$ to H future targets $\mathbf{Y}_{t+1:t+H} \in \mathbb{R}^{H \times C'}$. Let $\hat{\mathbf{Y}}_{t+1:t+H} = (\hat{\mathbf{y}}_{t+1}, \dots, \hat{\mathbf{y}}_{t+H})^\top$ denote the predictions. Training minimizes a horizon-aware loss

$$\mathcal{L}(\Theta) = \sum_t \sum_{h=1}^H \lambda_h \ell(\hat{\mathbf{y}}_{t+h}, \mathbf{y}_{t+h}), \quad \lambda_h \geq 0, \quad (1.2)$$

where ℓ is typically MAE/MSE and $\{\lambda_h\}$ optionally emphasize long horizons. Below we outline the four canonical strategies.

Direct (independent multi-step prediction). The direct strategy trains H horizon-specific predictors to avoid error propagation:

$$\hat{\mathbf{y}}_{t+h} = \phi_h(\mathbf{w}_t), \quad h = 1, \dots, H. \quad (1.3)$$

In practice, one shares a backbone g_ψ with H linear heads $\{\mathbf{W}_h\}$, i.e., $\phi_h(\mathbf{w}_t) = \mathbf{W}_h g_\psi(\mathbf{w}_t)$. This affords horizon-specific bias/variance control and robustness when dynamics differ across h , at the cost of training and storing H heads and potential cross-horizon incoherence.

Recursive (iterated one-step prediction). The recursive (autoregressive) strategy fits a single next-step model and rolls it forward:

$$\hat{\mathbf{y}}_{t+1} = \phi(\mathbf{w}_t), \quad \tilde{\mathbf{w}}_{t+1} = \text{shift}(\mathbf{w}_t, \hat{\mathbf{y}}_{t+1}), \quad \hat{\mathbf{y}}_{t+h} = \phi(\tilde{\mathbf{w}}_{t+h-1}), \quad (1.4)$$

where $\text{shift}(\cdot)$ drops the oldest row and appends the newest observation (or its prediction) to keep length L . Training uses *teacher forcing* (true targets in the window); inference is free-running and can accumulate error with increasing h , but only one model is trained and maintained.

DirRec (direct-recursive hybrid). DirRec uses direct predictors for the first H_d steps and a recursive model thereafter:

$$\hat{\mathbf{y}}_{t+h} = \phi_h(\mathbf{w}_t), \quad h = 1, \dots, H_d, \quad (1.5)$$

$$\hat{\mathbf{y}}_{t+h} = \phi(\tilde{\mathbf{w}}_{t+h-1}), \quad h = H_d+1, \dots, H, \quad (1.6)$$

with $\tilde{\mathbf{w}}$ initialized using $\hat{\mathbf{y}}_{t+1:t+H_d}$. This reduces early-horizon bias and error propagation while limiting the number of direct heads.

MIMO (multi-input multi-output, joint decoding). MIMO produces the entire horizon jointly in one pass,

$$\hat{\mathbf{Y}}_{t+1:t+H} = f_{\Theta}(\mathbf{w}_t) \in \mathbb{R}^{H \times C'}, \quad (1.7)$$

and optimizes (1.2) end-to-end. Joint decoding captures cross-step correlations and aligns naturally with sequence-to-sequence decoders (e.g., Transformer/LLM backbones) where the decoder state summarizes the history and emits a vector of future steps. It yields coherent trajectories and strong long-horizon accuracy, with higher per-update memory/compute than a single one-step model.

In this thesis, LSTM baselines are reported under Direct/Recursive settings, while LLM-based forecasters adopt the MIMO formulation with a shared backbone and horizon-wise heads or joint linear projection, ensuring consistent use of the loss in (1.2).

1.5.3 Reversible Instance Normalization (RevIN)

Reversible Instance Normalization (RevIN) is a per-instance, per-channel normalization/denormalization scheme designed for time-series forecasting under distribution shift. It removes instance-specific mean/variance before the model and restores them after prediction, thereby improving robustness to amplitude and scale changes while preserving invertibility.

Formulation. Let a window be $\mathbf{w}_t = \mathbf{X}_{t-L+1:t} \in \mathbb{R}^{L \times C}$ with time index $\tau \in \{t-L+1, \dots, t\}$ and channel $c \in \{1, \dots, C\}$. Compute instance statistics (per window, per channel):

$$\mu_c(\mathbf{w}_t) = \frac{1}{L} \sum_{\tau=t-L+1}^t x_{\tau,c}, \quad \sigma_c(\mathbf{w}_t) = \sqrt{\frac{1}{L} \sum_{\tau=t-L+1}^t (x_{\tau,c} - \mu_c(\mathbf{w}_t))^2 + \epsilon}.$$

RevIN applies an affine, channel-wise, *learnable* normalization

$$z_{\tau,c} = \alpha_c \frac{x_{\tau,c} - \mu_c(\mathbf{w}_t)}{\sigma_c(\mathbf{w}_t)} + \beta_c, \quad \text{for } \tau \in [t-L+1, t],$$

with parameters $\alpha_c, \beta_c \in \mathbb{R}$ trained with the forecaster. The normalized sequence $Z_{t-L+1:t} = \{z_{\tau,c}\}$ is fed to f_θ . Given a model output $\hat{\mathbf{Y}}_{t+1:t+H} \in \mathbb{R}^{H \times C'}$ in the normalized domain, RevIN reverses the transform channel-wise:

$$\hat{y}_{t+h,c} = \sigma_c(\mathbf{w}_t) \frac{\hat{y}_{t+h,c}^{(\text{norm})} - \beta_c}{\alpha_c} + \mu_c(\mathbf{w}_t), \quad h = 1, \dots, H.$$

Thus the overall mapping is invertible up to ϵ and the learned affine parameters. Forecasting error often deteriorates under covariate/target shift (seasonal amplitude changes, sensor re-scaling, device heterogeneity). RevIN explicitly factors out window-level first/second moments before modeling and restores them afterward, allowing f_θ to focus on shape/dynamics rather than absolute scale. The learned (α_c, β_c) adapt the strength of normalization per channel. We reuse this transform in later chapters when handling non-IID clients and drifting series.

1.5.4 Sequence Models

Long Short-Term Memory (LSTM). Recurrent neural networks model a sequence by iterating a state update $(\mathbf{h}_t, \mathbf{c}_t) = \Phi(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1})$. The LSTM augments the hidden state $\mathbf{h}_t \in \mathbb{R}^d$ with a memory cell $\mathbf{c}_t \in \mathbb{R}^d$ regulated by input/forget/output gates $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t \in \mathbb{R}^d$:

$$\begin{aligned} \mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{R}_i \mathbf{h}_{t-1} + \mathbf{b}_i), & \mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{R}_f \mathbf{h}_{t-1} + \mathbf{b}_f), \\ \tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_g \mathbf{x}_t + \mathbf{R}_g \mathbf{h}_{t-1} + \mathbf{b}_g), & \mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{R}_o \mathbf{h}_{t-1} + \mathbf{b}_o), \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, & \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \end{aligned}$$

Here $\mathbf{x}_t \in \mathbb{R}^p$ is the input at time t , $\sigma(\cdot)$ denotes the logistic sigmoid, and $\odot$ is the Hadamard product; $\mathbf{W}_\star \in \mathbb{R}^{d \times p}$, $\mathbf{R}_\star \in \mathbb{R}^{d \times d}$, and $\mathbf{b}_\star \in \mathbb{R}^d$ are learnable parameters. The gating mitigates vanishing gradients and enables long-range temporal dependencies—useful for both detection (e.g., residual-based alarms) and forecasting.

Gated Recurrent Unit. The GRU is a lighter gated variant without an explicit cell state:

$$\begin{aligned} \mathbf{z}_t &= \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{R}_z \mathbf{h}_{t-1}), & \mathbf{r}_t &= \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{R}_r \mathbf{h}_{t-1}), \\ \tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{R}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1})), & \mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t. \end{aligned}$$

Fewer parameters and lower compute make GRUs attractive on constrained clients, albeit with slightly less capacity for very long horizons than LSTMs.

Scaled Dot-Product Attention. Attention computes a content-based aggregation of *values* $\mathbf{V}$ weighted by the similarity between *queries* $\mathbf{Q}$ and *keys* $\mathbf{K}$. Given a sequence embedding $\mathbf{X} \in \mathbb{R}^{T \times d}$,

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

with $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$, the single-head attention is

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V},$$

where the softmax is taken row-wise over time. The quadratic complexity $\mathcal{O}(T^2d)$ in sequence length T is the main scaling challenge addressed by sparse, low-rank, or patch-wise variants in long-horizon forecasting.

Multi-Head Attention (MHA). Instead of a single similarity space, MHA aggregates H parallel heads,

$$\text{MHA}(\mathbf{X}) = \text{Concat}\left(\text{Attn}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h)\right)_{h=1}^H \mathbf{W}_O,$$

with per-head projections $(\mathbf{W}_Q^{(h)}, \mathbf{W}_K^{(h)}, \mathbf{W}_V^{(h)})$ and output $\mathbf{W}_O$. Heads specialize to distinct temporal patterns or channels, improving expressivity at nearly linear parameter cost in H .

Transformer Architectures for Time Series. A Transformer encoder layer composes residual blocks of MHA and a position-wise feed-forward network (FFN), with layer normalization (LN):

$$\mathbf{Z} = \text{LN}(\mathbf{X} + \text{MHA}(\mathbf{X})), \quad \mathbf{Y} = \text{LN}(\mathbf{Z} + \text{FFN}(\mathbf{Z})).$$

Positional encodings (sinusoidal or learned) inject order. Decoder-style (auto-regressive) Transformers apply causal masks for next-step prediction; encoder-only stacks are often used with *direct* multi-step heads. For multivariate series, a common adaptation is *channel-independent (CI) tokenization/patching*: each channel is segmented into fixed-length patches (tokens) and processed with shared Transformer weights. CI improves data efficiency and robustness to cross-channel scale mismatch; when cross-channel interactions are important, lightweight fusion (e.g., cross-attention or grouped mixing) can be added. In resource-constrained federated settings, parameter-efficient tuning (LoRA/QLoRA) replaces full weight updates with low-rank adapters, $\mathbf{W} \leftarrow \mathbf{W} + \alpha \mathbf{A}\mathbf{B}^\top$ with rank $r \ll d$, reducing communication and memory without sacrificing accuracy.

1.5.5 Federated Learning Setup

Let $\mathcal{K} = \{1, \dots, K\}$ index participating silos (e.g., buildings, stations), each holding a private dataset $\mathcal{D}_k$ with $N_k = |\mathcal{D}_k|$ samples. A parametric predictor f_θ with parameters $\theta \in \Theta$ is trained collaboratively without sharing raw data. The local empirical objective at client k is

$$f_k(\theta) = \frac{1}{N_k} \sum_{(x,y) \in \mathcal{D}_k} \ell(f_\theta(x), y), \quad (1.8)$$

and the global objective aggregates these losses with sample-proportional weights $p_k = N_k/N$ where $N = \sum_{j=1}^K N_j$:

$$\min_{\theta \in \Theta} F(\theta) \triangleq \sum_{k=1}^K p_k f_k(\theta). \quad (1.9)$$

Round structure and FedAvg. At communication round r , the server broadcasts $\theta^{(r)}$. Each selected client performs E epochs of local optimization (e.g., SGD/Adam) to obtain $\theta_k^{(r+1)}$, then uploads the model *delta* $\Delta_k^{(r)} = \theta_k^{(r+1)} - \theta^{(r)}$. FedAvg updates the global model by a weighted average of client models (equivalently, of deltas):

$$\theta^{(r+1)} = \sum_{k=1}^K p_k \theta_k^{(r+1)} = \theta^{(r)} + \sum_{k=1}^K p_k \Delta_k^{(r)}. \quad (1.10)$$

When client drift is severe, a proximal regularizer stabilizes local steps (FedProx):

$$\theta_k^{(r+1)} \approx \arg \min_{\theta} f_k(\theta) + \frac{\mu}{2} \|\theta - \theta^{(r)}\|_2^2, \quad \mu \geq 0. \quad (1.11)$$

Heterogeneity (non-IID) taxonomy. Data distributions $P_k(X, Y)$ generally differ across silos. Four common sources of heterogeneity are: (i) *covariate shift* ($P_k(X) \neq P_j(X)$), (ii) *label/target shift* ($P_k(Y) \neq P_j(Y)$), (iii) *quantity skew* (N_k vary widely, affecting p_k), and (iv) *temporal concept drift* (time-varying $P_k^{(t)}$). These induce biased local optima and slow or unstable convergence of (1.10); remedies include proximal terms (1.11), personalized/multi-task heads, or cluster-aware training.

Privacy primitives. *Secure aggregation* allows the server to recover only the *sum* of updates, not individual ones. Let each pair of clients (i, j) exchange a random mask $\mathbf{r}_{i \rightarrow j}$ and set $\tilde{\Delta}_k^{(r)} = \Delta_k^{(r)} + \sum_{j>k} \mathbf{r}_{k \rightarrow j} - \sum_{j<k} \mathbf{r}_{j \rightarrow k}$. Masks cancel in the aggregate:

$$\sum_{k=1}^K \tilde{\Delta}_k^{(r)} = \sum_{k=1}^K \Delta_k^{(r)}, \quad (1.12)$$

revealing only the total needed for (1.10). Client-level (ε, δ) -DP is obtained by per-client clipping and Gaussian noise:

$$\widehat{\Delta}_k^{(r)} = \text{clip}(\Delta_k^{(r)}, C), \quad \Delta_{\text{DP}}^{(r)} = \frac{1}{K} \sum_{k=1}^K \widehat{\Delta}_k^{(r)} + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}), \quad (1.13)$$

with privacy calibrated via a moments accountant. Communication can be further reduced with compression $q(\cdot)$ (e.g., s -sparsification, b -bit quantization), by sending $q(\Delta_k^{(r)})$.

Participation and personalization. Under partial participation, $\mathcal{S}_r \subset \mathcal{K}$ denotes the sampled clients at round r and (1.10) is taken over $k \in \mathcal{S}_r$ with re-normalized weights. Personalization mitigates heterogeneity by learning a shared backbone θ and client-specific heads ϕ_k , or by partitioning clients into clusters $\{\mathcal{C}_m\}$ and training cluster models $\theta^{(m)}$ with local adaptation; this connects directly to the cluster-aware designs used in the forecasting chapters.

1.5.6 Server-Side Adaptive Optimization

Beyond weighted averaging in (1.10), a server (or leader) can apply an adaptive optimizer to the aggregated client signal to damp client drift induced by non-IID data and to stabilize long-horizon training. We adopt *FedAdam*, the Adam update applied to the aggregated model delta. Let the local model delta from client k in round r be

$$\Delta_k^{(r)} \triangleq \theta_k^{(r+1)} - \theta^{(r)} \in \mathbb{R}^{\dim(\theta)},$$

and the aggregated pseudo-gradient be the FedAvg-weighted sum

$$\mathbf{g}^{(r)} = \sum_{k=1}^K p_k \Delta_k^{(r)}. \quad (1.14)$$

FedAdam maintains first and second moment estimates $(\mathbf{m}^{(r)}, \mathbf{v}^{(r)})$ with the *same shape as* θ :

$$\mathbf{m}^{(r+1)} = \beta_1 \mathbf{m}^{(r)} + (1 - \beta_1) \mathbf{g}^{(r)}, \quad (1.15)$$

$$\mathbf{v}^{(r+1)} = \beta_2 \mathbf{v}^{(r)} + (1 - \beta_2) (\mathbf{g}^{(r)} \odot \mathbf{g}^{(r)}), \quad (1.16)$$

with bias corrections

$$\hat{\mathbf{m}}^{(r+1)} = \mathbf{m}^{(r+1)} / (1 - \beta_1^{r+1}), \quad \hat{\mathbf{v}}^{(r+1)} = \mathbf{v}^{(r+1)} / (1 - \beta_2^{r+1}). \quad (1.17)$$

The server (leader) updates the global parameters by

$$\theta^{(r+1)} = \theta^{(r)} - \eta \hat{\mathbf{m}}^{(r+1)} \odot (\sqrt{\hat{\mathbf{v}}^{(r+1)}} + \varepsilon). \quad (1.18)$$

Using $\Delta_k^{(r)}$ in (1.14) is equivalent to aggregating (stochastic) gradients when locals are small-step SGD; the delta view matches secure aggregation settings where only sums of updates are recoverable. Quantized/compressed updates are dequantized before (1.14). Typical hyper-parameters are $\beta_1=0.9$, $\beta_2=0.999$, $\varepsilon \in [10^{-8}, 10^{-6}]$ with a server step size η tuned per task; decoupled weight decay (AdamW) can replace implicit ℓ_2 if desired. In the decentralized setting of Chapter 4, the elected *Leader* executes (1.15)–(1.18) on the winning, validated aggregate and then broadcasts $\theta^{(r+1)}$ on-chain for auditability and synchronization.

1.5.7 Differential Privacy in Federated Learning

A randomized mechanism $\mathcal{M}$ with domain $\mathcal{D}$ satisfies (ε, δ) -differential privacy (DP) if for all measurable S and for all neighboring datasets $D, D' \in \mathcal{D}$ differing in one record,

$$\Pr[\mathcal{M}(D) \in S] \leq e^\varepsilon \Pr[\mathcal{M}(D') \in S] + \delta.$$

In *client-level* DP for cross-silo FL, “neighboring” typically means adding/removing all records of one client, thereby protecting entire participants rather than single samples.

DP mechanisms for FL. In each communication round, client k computes an update (e.g., model delta) Δ_k from local data. DP-SGD style protection proceeds via:

$$\begin{aligned} (i) \text{ Clipping: } \quad \bar{\Delta}_k &= \Delta_k \cdot \min\left(1, \frac{C}{\|\Delta_k\|_2}\right), \\ (ii) \text{ Noise addition: } \quad \tilde{\Delta} &= \frac{1}{M} \sum_{k \in \mathcal{S}} \bar{\Delta}_k + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}), \end{aligned}$$

where $C > 0$ is the clipping norm, $\sigma > 0$ the noise multiplier, $\mathcal{S}$ the set of M sampled/participating clients, and $\mathbf{I}$ the identity matrix in parameter space. The Gaussian mechanism ensures (ε, δ) -DP for appropriate $(\sigma, \varepsilon, \delta)$ choices; privacy amplification by subsampling applies when clients are sampled with rate $q = M/K$.

Accounting. Over R rounds, composition theorems (e.g., Moments Accountant / Rényi DP) track the cumulative (ε, δ) under given q , C , and σ . Tighter accounting yields smaller ε for the same noise level, which is crucial for long training runs.

Secure aggregation and DP. *Secure aggregation* (cryptographic) ensures the server learns only the sum $\sum_k \bar{\Delta}_k$ (and not individual $\bar{\Delta}_k$), while DP (statistical) limits what can be inferred about any single client from the released aggregate. In practice, secure aggregation protects updates in transit; DP bounds worst-case inference from model releases or logs. Both are complementary and can be used together.

1.5.8 Parameter-Efficient Fine-Tuning (PEFT): LoRA and QLoRA

Definition. Parameter-efficient fine-tuning (PEFT) adapts a large frozen backbone by training a small set of auxiliary parameters, rather than updating all weights. This reduces memory/compute and, in federated settings, the number of parameters communicated per round.

Low-Rank Adapters (LoRA). Consider a linear layer with weight $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and input $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$. LoRA reparameterizes the effective weight as

$$\mathbf{W}_{\text{eff}} = \mathbf{W} + \alpha \mathbf{A}\mathbf{B}, \quad \mathbf{A} \in \mathbb{R}^{d_{\text{out}} \times r}, \mathbf{B} \in \mathbb{R}^{r \times d_{\text{in}}}, \quad r \ll \min(d_{\text{out}}, d_{\text{in}}), \quad (1.19)$$

where $\mathbf{W}$ is frozen and only the low-rank factors $(\mathbf{A}, \mathbf{B})$ are trained. The forward map becomes

$$\mathbf{y} = \mathbf{W}_{\text{eff}}\mathbf{x} = \mathbf{W}\mathbf{x} + \alpha \mathbf{A}(\mathbf{B}\mathbf{x}). \quad (1.20)$$

The trainable parameter count per layer is

$$P_{\text{LoRA}} = r(d_{\text{in}} + d_{\text{out}}), \quad \text{fraction } \rho_{\text{LoRA}} = \frac{P_{\text{LoRA}}}{d_{\text{out}}d_{\text{in}}} = \frac{r}{d_{\text{in}}} + \frac{r}{d_{\text{out}}},$$

which is typically orders of magnitude smaller than $d_{\text{out}}d_{\text{in}}$. The scaling α controls update magnitude and is often set proportional to r (e.g., $\alpha = r$) to keep the effective step size approximately rank-invariant. At inference, one may either keep the decomposition in (1.20) or merge the adapter into the base weight via $\mathbf{W} \leftarrow \mathbf{W} + \alpha \mathbf{A}\mathbf{B}$.

Quantized LoRA (QLoRA). QLoRA further reduces memory by storing the frozen backbone in low precision while keeping adapters in higher precision for stable optimization. Let $\text{Quant}_{b_w}(\cdot)$ denote b_w -bit quantization (e.g., 4-bit) with scale s and zero-point z . The quantized weight and its dequantization are

$$\tilde{\mathbf{W}} = \text{Quant}_{b_w}(\mathbf{W}; s, z), \quad \hat{\mathbf{W}} = s(\tilde{\mathbf{W}} - z) \approx \mathbf{W}. \quad (1.21)$$

The effective layer used in training is then

$$\mathbf{y} = (\hat{\mathbf{W}} + \alpha \mathbf{A}\mathbf{B})\mathbf{x}, \quad (1.22)$$

with gradients flowing to $(\mathbf{A}, \mathbf{B})$ only. The memory footprint per layer becomes

$$\underbrace{d_{\text{out}}d_{\text{in}} \cdot \frac{b_w}{8}}_{\text{quantized frozen backbone}} + \underbrace{r(d_{\text{in}} + d_{\text{out}}) \cdot \frac{b_a}{8}}_{\text{adapters in FP16/BF16}},$$

where b_a is the adapter precision (commonly 16). In cross-silo FL, communicating only $\{\mathbf{A}, \mathbf{B}\}$ (or their deltas) yields a round-trip payload proportional to P_{LoRA} rather than $d_{\text{out}}d_{\text{in}}$, substantially reducing bandwidth. Because all clients share the same quantized backbone, aggregation (e.g., FedAvg/FedAdam on adapters) remains well-posed and reproducible.

Placement and Practicalities. Adapters are typically inserted in attention projections ($\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o$) and MLP/FFN matrices, which concentrates capacity where most representational change is needed while keeping compute low via the rank- r bottleneck. In long-horizon forecasting, LoRA/QLoRA pairs naturally with channel-independent patching: the backbone captures generic sequence regularities, while adapters specialize to forecasting objectives and client-specific dynamics. In decentralized or federated regimes, this design also improves robustness and privacy by limiting the communicated state to small adapter tensors rather than full precision backbones.

1.5.9 Decentralized Coordination

We formalize consensus in a permissioned setting with a validator set $\mathcal{V}$ of size n and a quorum threshold q (e.g., $q = \lceil 2n/3 \rceil$ for BFT-style finality). Training proceeds in rounds $r = 1, 2, \dots$. In round r a subset of participants (the miners) $\mathcal{M}_r \subseteq \mathcal{K}$ each produces a signed update proposal. For notational economy we write the transmitted payload as

$$u_i^{(r)} \triangleq \bar{\Delta}_i^{(r)} \quad \text{for } i \in \mathcal{M}_r,$$

where $\bar{\Delta}_i^{(r)}$ is a (possibly quantized) model delta relative to the current global parameters. An evaluator committee $\mathcal{E}_r$ (which may be a subset of $\mathcal{V}$ or distinct) reconstructs candidate models and computes a scalar validation loss

$$\mathcal{L}_i^{(r)} = \ell(f_{\theta^{(r)} \oplus u_i^{(r)}}; V^{(r)}),$$

on a fixed held-out slice $V^{(r)}$ for that round, where $\theta^{(r)}$ denotes the pre-round global parameters and $\oplus$ applies the delta.

Deterministic update selection. Block contents for round r are chosen by the best-loss rule

$$u_\star^{(r)} \in \arg \min_{i \in \mathcal{M}_r} \mathcal{L}_i^{(r)}, \quad (1.23)$$

with deterministic tie-breaking (e.g., lower variance across evaluator scores, then smaller agent identifier). For comparability across rounds with different loss scales, we also compute a normalized, higher-is-better score

$$s_i^{(r)} = \begin{cases} \frac{L_{\max}^{(r)} - \mathcal{L}_i^{(r)}}{L_{\max}^{(r)} - L_{\min}^{(r)}}, & \text{if } L_{\max}^{(r)} > L_{\min}^{(r)}, \\ \frac{1}{|\mathcal{M}_r|}, & \text{otherwise,} \end{cases} \quad \begin{aligned} L_{\max}^{(r)} &= \max_{j \in \mathcal{M}_r} \mathcal{L}_j^{(r)}, \\ L_{\min}^{(r)} &= \min_{j \in \mathcal{M}_r} \mathcal{L}_j^{(r)}. \end{aligned} \quad (1.24)$$

The score $s_i^{(r)} \in [0, 1]$ is used for incentive accounting and, if needed, as a secondary ranking signal.

Table 1.1: Summary of symbols.

Symbol	Meaning
T, C	Sequence length; number of channels
L, H	Look-back window; forecast horizon
$X_{1:T} \in \mathbb{R}^{T \times C}$	Multivariate time series; $\mathbf{w}_t = X_{t-L+1:t}$ window
$Y_{t+1:t+H}$	Future targets for horizon H
f_θ	Forecaster with parameters θ ; $\hat{Y} = f_\theta(\mathbf{w})$
$\mathcal{K}, K$	Set of clients; number of clients
$\mathcal{D}_k, N_k$	Client k 's dataset and sample count
p_k	Aggregation weight $N_k / \sum_j N_j$
$\theta^{(r)}, \theta_k^{(r)}$	Global and local parameters at round r
$\mathbf{m}^{(r)}, \mathbf{v}^{(r)}$	First/second moments in FedAdam
P, S, N_p	Patch length, stride, and number of patches per channel
$\mathcal{M}_r$	Proposers (miners) in round r
$u_i^{(r)}$	Proposed update (e.g., delta) from participant i
$\mathcal{L}_i^{(r)}$	Validation loss for proposal i in round r
$s_i^{(r)}$	Normalized score from (1.24)
n, q	Number of validators; quorum threshold for finality

Quorum and finality. Validators independently verify the selected update $u_\star^{(r)}$ and produce signatures. Let $N_{\text{sig}}^{(r)}$ denote the number of valid endorsements gathered for round r . Finality is encoded by the indicator

$$\mathbf{1}_{\text{finalize}}^{(r)} = \begin{cases} 1, & \text{if } N_{\text{sig}}^{(r)} \geq q, \\ 0, & \text{otherwise,} \end{cases} \quad (1.25)$$

upon which the block is committed and the global parameters advance as $\theta^{(r+1)} = \theta^{(r)} \oplus u_\star^{(r)}$.

Role of stake vs. accuracy. Traditional proof-of-stake (PoS) elects proposers according to stake and finalizes blocks by stake-weighted voting. In contrast, the proposed proof-of-forecast (PoF) ties proposership to verifiable predictive merit: the leader for round r is the miner supplying $u_\star^{(r)}$ in (1.23). Stake may still regulate eligibility or penalties (slashing) but does not determine leadership or acceptance, which are governed by measured accuracy and the quorum condition (1.25).

The time-series notation in §1.5.1 (windows $X_{t-L+1:t}$, horizons H) and forecasting strategies (§1.5.2) are used throughout Chapters 2–4. The federated objective in Eq. (1.9) and the FedAvg update in Eq. (1.10) underpin the training setups of Chapter 2 (FSLSTM for anomaly detection) and Chapter 3 (FedTime for long-horizon forecasting), with the server-side optimizer in §1.5.6 reused verbatim in both. The PEFT formulation in §1.5.8 (LoRA/QLoRA) is the parameter-efficient mechanism adopted for LLM adaptation in Chapter 3. Finally, the decentralized coordination

primitives in §1.5.9—the update selection objective in Eq. (1.23) and the normalized score $s_i^{(r)}$ in Eq. (1.24)—provide the scoring and finality semantics instantiated by BLEND’s Proof-of-Forecast consensus in Chapter 4. We consistently reuse this notation across chapters to facilitate cross-reference and avoid ambiguity.

1.6 Applications

Anomaly Detection in Smart Systems. Anomaly detection has become indispensable in smart systems such as smart grids, autonomous driving platforms, and industrial IoT. These systems operate in highly dynamic environments where faulty sensor readings, adversarial intrusions, or abnormal system behaviors can lead to catastrophic failures if not detected in time. For example, in the energy sector, anomaly detection models are used to identify irregular consumption patterns or detect early signs of equipment malfunction in transformers and substations [36]. Similarly, in intelligent transportation systems, anomalies such as sudden deceleration patterns or irregular vehicular trajectories can signal safety hazards or malicious activities [37].

Recent advances in deep learning, particularly autoencoders and graph neural networks (GNNs), have shown promise in capturing complex dependencies across multi-modal sensor streams [38–40]. By decentralizing anomaly detection, our proposed approach allows computation to be carried out locally at the edge (e.g., roadside units, industrial gateways), reducing detection latency and minimizing the risk of data exfiltration. This paradigm also addresses privacy concerns in domains such as healthcare, where anomalies in patient vitals or imaging data must be flagged without transmitting sensitive raw data to centralized servers [41]. Collectively, these applications demonstrate how decentralized anomaly detection enables robust, real-time monitoring across critical infrastructures.

Federated Long-Term Time Series Forecasting. Long-horizon forecasting is a cornerstone in domains such as energy demand planning, mobility management, and healthcare. Traditional centralized approaches face two principal challenges: (i) sensitive time series data are distributed across multiple stakeholders and governed by strict privacy regulations, and (ii) long-term forecasting models are prone to data heterogeneity and concept drift. Federated learning (FL) provides a natural solution by enabling multiple parties to collaboratively train forecasting models without sharing raw data [42, 43].

In the energy sector, forecasting EV charging demand is essential to alleviate stress on the power grid and to optimize charging station dispatch. Studies have demonstrated that federated recurrent neural networks (RNNs) and transformers can achieve competitive performance while preserving

user privacy [44]. Similarly, in predictive maintenance of smart manufacturing plants, federated forecasting models can anticipate machinery degradation by leveraging time series sensor logs distributed across different factories. In healthcare, long-term federated forecasting of patient trajectories has been employed to monitor chronic conditions such as diabetes, where continuous glucose monitoring data remain siloed across hospitals [45].

Our work extends this literature by focusing on long-term horizons and decentralized coordination. By integrating advanced sequence models with federated optimization, we demonstrate that robust forecasts can be obtained even in the presence of non-IID data distributions and communication constraints. This capability is particularly important in IoV ecosystems, where accurate long-term demand forecasts directly translate into reduced congestion and improved energy utilization.

Consensus and Coordination Without a Central Server. Achieving reliable consensus in the absence of a trusted central server is a fundamental challenge in distributed intelligence. Applications span blockchain-based financial platforms, vehicular ad-hoc networks (VANETs), and collaborative robotics. In such environments, participants must reach agreement on a shared global state despite unreliable communication channels and potential adversarial actors [46].

In intelligent transportation systems, for instance, vehicles and roadside units (RSUs) must coordinate lane merging, hazard warnings, or traffic light synchronization in real-time. Conventional centralized control schemes introduce single points of failure, making them unsuitable for safety-critical applications [47]. Blockchain-inspired consensus algorithms, such as Byzantine Fault Tolerant (BFT) and Proof-of-Stake (PoS) mechanisms, provide fault-tolerant alternatives. For example, lightweight BFT protocols have been deployed in vehicular networks to ensure consistent hazard detection across multiple RSUs [48]. Similarly, in supply chain management, decentralized coordination improves transparency and trust by eliminating reliance on intermediaries [49].

Beyond consensus itself, coordination mechanisms are required to manage joint decision-making under uncertainty. Multi-agent reinforcement learning combined with decentralized consensus protocols has been explored for distributed sensor fusion, swarm robotics, and drone coordination [50]. By integrating consensus protocols with machine learning workflows, our work aims to bridge predictive intelligence with trustless coordination. This synergy is particularly valuable in IoV settings, where decisions must be both data-driven and robust to adversarial disruptions.

1.7 Overview and Contributions

The remainder of this thesis is structured as follows:

- In Chapter 2, we introduce a federated learning framework for anomaly detection in smart buildings. Motivated by the need for privacy-preserving and efficient monitoring, we propose a federated stacked long short-term memory (FSLSTM) model that leverages multi-task learning to capture both individual sensor patterns and shared representations across distributed datasets. Experimental results on real-world IoT sensor data from General Electric Current smart buildings demonstrate that the FSLSTM converges twice as fast as centralized baselines, while achieving superior performance in both classification and regression tasks. The proposed framework significantly reduces communication cost, supports contextual and collective anomaly detection, and highlights the benefits of decentralized architectures in intelligent infrastructure systems.
- In Chapter 3, we present FedTime, a federated large language model designed for long-term time series forecasting. The proposed framework integrates clustering-based personalization, patching strategies, and parameter-efficient fine-tuning techniques to adapt LLaMA-based backbones for multivariate forecasting tasks. FedTime addresses core challenges of centralized methods, including privacy leakage and communication bottlenecks, by training models collaboratively across edge devices. Comprehensive experiments on seven benchmark datasets—including Weather, Traffic, Electricity, and ETT—show that FedTime consistently outperforms state-of-the-art centralized and federated baselines. The results validate its robustness for long-horizon forecasting while ensuring reduced communication overhead and enhanced scalability.
- In Chapter 4, we propose BLEND, a blockchain-enhanced decentralized learning framework for long-term time series forecasting. BLEND eliminates reliance on a central server by embedding a Proof-of-Forecast (PoF) consensus mechanism into a consortium blockchain. This ensures that block finality is tied directly to predictive accuracy, incentivizing honest participation and filtering out poisoned updates at commit time. The framework employs an agentic workflow, where Publisher, Miner, Evaluator, and Leader roles interact via smart contracts to ensure auditability, fairness, and resilience to Byzantine participants. Empirical evaluation on seven public benchmarks demonstrates that BLEND not only achieves state-of-the-art forecasting performance but also reduces communication overhead by up to 63% compared to federated baselines, while maintaining robustness under adversarial conditions. We provide a comprehensive comparative analysis of the BLEND framework for decentralized consensus—highlighting their respective advantages and trade-offs. This chapter consolidates insights into how federated and blockchain-enabled learning can jointly advance

reliability, privacy, and scalability in mission-critical applications such as smart buildings, Internet of Vehicles (IoV), and energy demand forecasting.

- Finally, in Chapter 5, we conclude the thesis by summarizing the main findings, discussing limitations, and outlining potential avenues for future work, including multimodal federated learning, incentive-compatible consensus protocols, and integration with emerging edge-cloud continuum systems.

1.8 Summary of Publications

1.8.1 Thesis Research

The contents of Chapter 2 consist of research ideas and results taken from:

Raed Abdel-Sater, A. Ben Hamza. “A Federated Learning Approach to Anomaly Detection in Smart Buildings,” *ACM Transactions on Internet of Things*, 2(4):1–23, 2021.

The contents of Chapter 3 consist of research ideas and results taken from:

Raed Abdel-Sater, A. Ben Hamza. “A Federated Large Language Model for Long-Term Time Series Forecasting,” *Proc. European Conference on Artificial Intelligence (ECAI)*, pp. 2452–2459, 2024.

The contents of Chapter 4 consist of research ideas and results taken from:

Raed Abdel-Sater, A. Ben Hamza. “BLEND: Blockchain-Enhanced Network Decentralization with Large Language Models for Time Series Forecasting,” *Under Review*, 2025.

1.8.2 Non-thesis Research

Additional publication carried out during the course of this Ph.D. program:

Raed Abdel-Sater, “Marriage Dissolution in the United States: A Survival Analysis Approach,” *Journal of Divorce & Remarriage*, pp. 235–261 2022.

Federated Learning for Anomaly Detection in Smart Buildings

Internet of Things (IoT) sensors in smart buildings are becoming increasingly ubiquitous, making buildings more livable, energy efficient, and sustainable. These devices sense the environment and generate multivariate temporal data of paramount importance for detecting anomalies and improving the prediction of energy usage in smart buildings. However, detecting these anomalies in centralized systems is often plagued by a huge delay in response time. To overcome this issue, we formulate the anomaly detection problem in a federated learning setting by leveraging the multi-task learning paradigm, which aims at solving multiple tasks simultaneously while taking advantage of the similarities and differences across tasks. We propose a novel privacy-by-design federated learning model using a stacked long short-time memory (LSTM) model, and we demonstrate that it is more than twice as fast during training convergence compared to the centralized LSTM. The effectiveness of our federated learning approach is demonstrated on three real-world datasets generated by the IoT production system at General Electric Current smart building, achieving state-of-the-art performance compared to baseline methods in both classification and regression tasks. Our experimental results demonstrate the effectiveness of the proposed framework in reducing the overall training cost without compromising the prediction performance.

2.1 Introduction

Smart buildings are making bold use of data collected by IoT sensors to assist in a wide range of positive outcomes, including cost reduction, improved safety and maintenance, and prevention of building equipment downtime. In simple terms, IoT refers to a network of sensors and other devices that are capable of sending and receiving data [51, 52], allowing diverse network components to cooperate and make their resources available to execute a common task. IoT helps create interoperable networks in smart buildings by connecting various types of sensors and other devices from which actionable insights can be extracted through collection and analysis of massive amounts of real-time data.

IoT-enabled smart buildings are changing the way we live, leveraging the power of intelligent devices to remotely monitor and control key equipment in the premises while ensuring greater energy and operational efficiency [53, 54]. In such an intelligent environment, a key objective is to provide support tools to help building managers and users make cost-effective decisions when utilizing, for example, electrical energy. It is estimated that more than half the global electrical energy is consumed by commercial buildings, and around 45% of that energy is generated by Heating, Ventilation and Air Conditioning (HVAC) systems [55]. In order to intelligently improve performance and create smarter buildings, there is a pressing need for developing efficient machine learning models that effectively learn the history patterns of IoT sensors in an effort to increase energy efficiency and help cut costs. Such models not only help facility managers make strategic decisions through data analysis and actionable insights to ensure buildings are working smarter and running at maximum efficiency, but can also help buildings self-diagnose and optimize. In particular, an anomaly detection model can be used to deliver insights on the present and future performance of critical assets in smart buildings.

Anomaly detection is the process of identifying anomalous observations, which do not conform to the expected pattern of other observations in a dataset. Detecting anomalies has become a central research question in IoT applications, particularly from IoT time-series data [56–58]. For instance, a lighting energy consumption pattern might vary in an office building. Smart lighting control systems use occupancy sensors in correlation with light sensors to dim light based on changing occupancy and daylight levels, with the benefit of conserving energy in the building [59]. Moreover, an observation of high energy consumption levels due to heating in a bank might be anomalous after working hours, but not during the day. Occupancy sensors can be used to measure occupancy and space utilization, as well as in other systems such as HVAC to detect such anomalies [60]. Multiple types of sensors are of great importance to building operations and can be used for a number of applications in an IoT environment, including energy consumption monitoring and control, and

security of critical systems. Most modern structures are equipped with a building automation system (BAS), which enables facility managers to automate and oversee the energy efficiency of a building by controlling various components within a building’s structure, such as HVAC.

Early works on detection and prediction tasks in smart buildings have focused on centralized approaches using IoT sensor data. Idé *et al.* [61] used correlation between multiple sensors to form neighborhood graphs for computing correlation anomaly scores, but they treat a multi-sensor system as a collection of centralized sensors. Bellido-Outeirino *et al.* [62] presented a building lighting automation system by integrating digital addressable lighting interface devices in wireless sensor networks using a centralized system, in which appliances are managed by a wireless sensor network that focuses on lighting automation, while considering maintenance and energy consumption costs. Yu *et al.* [63] developed a centralized real-time HVAC system to construct and stabilize virtual queues associated with indoor temperatures by minimizing the long-term total cost associated with the HVAC system in the smart grid, which is considered a key IoT application that involves the incorporation of a secure, two-way information and communication flow, along with a two-way power flow. Li *et al.* [64] proposed a centralized water leak detection method using a classifier based on artificial neural networks using acoustic emission sensor data in an effort to detect water leaks in municipal pipeline systems. Chandra *et al.* [65] presented a Bayesian approach to multi-task learning for dynamic time series prediction via cascaded neural networks by decomposing a single task learning problem into multi-task learning through subtasks that have inter-dependencies defined by the size of the window used for embedding. In order to detect spatial, temporal, and spatio-temporal anomalies in real-time sensor measurement data streams, Chen *et al.* [66] proposed an anomaly detection framework for real-world environmental sensing systems in a bid to identify outliers in raw measurement data.

In centralized systems, an abnormal sensor behavior is detected by a central model on a server. Typically, all the training data are collected from the sensors through gateways and saved on the cloud. Centralized systems are, however, prone to failures, vulnerable to cyber invasions and infections, and often require longer access times for training data coming from remote devices, leading to potential data exposure during the transmission from clients to server, and hence raising concerns about clients’ privacy [67]. Federated learning has recently emerged as a powerful alternative to centralized systems, as it enables the collaborative training of machine learning models from decentralized datasets in users’ devices such as mobile phones, wearable devices and smart sensors without uploading their privacy-sensitive data to a central server or service provider [12, 68–75], while reducing communication cost [24]. For example, in a federated learning system for smart buildings, each sensor performs model training using its own data and sends local updates to the

central server for aggregation in order to update the global model, which is then redistributed to the sensors. The training process of a federated learning model is usually carried out using the federated averaging algorithm [69], and is repeated iteratively until model convergence or the maximum number of training rounds is reached. Moreover, secure aggregation can be used to combine the outputs of local models on sensors for updating a global model by leveraging secure multiparty computation [76].

While federated learning has some privacy-enhancing advantages as compared to sharing private data with a central server, recent studies have shown that an honest-but-curious server can analyze the local model parameter updates to perform gradient leakage attacks [77, 78], which usually occur on comprised sensors, and hence the server may gain access to some private training data. To mitigate this issue of data leakage, federated learning can be used in conjunction with secure multi-party computation, homomorphic encryption, or differential privacy. With secure multi-party computation, for instance, the local model updates are securely combined into a single aggregate update in order to ensure that the data communicated through federated learning to the centralized server stays private [24]. Another effective strategy to prevent the gradients from leaking private training data is to set those with small magnitudes to zero [77].

In this chapter, we introduce a federated stacked long short-time memory model using federated learning on time series data generated by IoT sensors for classification and regression tasks such as lighting fault detection and energy usage prediction. In addition to learning long-term dependencies between time steps of sequence data, the proposed model learns individual feature correlations within each sensor and also shared feature representations across distributed datasets to improve model convergence for faster learning. We perform the training process, which involves input data from multiple heterogeneous data sources across numerous sensors, in a collaborative fashion on IoT sensors in lieu of the server. This strategy not only lowers the cost and anticipates failures, but also helps reduce computational demands and inherits the privacy-enhancing capabilities of federated learning. The proposed framework provides: (i) a reduced network traffic by sharing the weights solely with the federated server, (ii) a shorter convergence time thanks to the collaborative learning, and (iii) a federated methodology for training various types of sensors such as occupancy sensors. Moreover, we design a federated gated recurrent unit model and also a federated logistic regression to learn a common representation among multiple sensors. Our main contributions can be summarized as follows:

- We leverage the multi-task learning paradigm to formulate the anomaly detection problem in smart buildings.
- We propose a novel privacy-by-design federated stacked long short-time memory model for

anomaly detection in smart buildings using IoT sensor data.

- We demonstrate that our proposed federated stacked LSTM model converges 2x faster than the centralized LSTM during the training phase and significantly reduces the communication cost.
- We present experimental results to demonstrate the superior performance of our model in comparison with baseline methods on sensors event log and energy usage datasets.

The rest of this chapter is organized as follows. In Section 2.2, we provide a brief overview of centralized and federated learning approaches for anomaly detection in a smart building setting using IoT sensor data. In Section 2.3, we present our federated learning system as well as our problem formulation and propose a novel federated learning architecture for anomaly detection in smart buildings. We discuss in detail the main components and algorithmic steps of the proposed framework. In Section 2.4, we present experimental results to demonstrate the superior performance of our approach in comparison with baseline methods. Finally, we conclude in Section 2.5 with a discussion.

2.2 Related Work

The growing number of smart city centers has accelerated the proliferation of IoT sensors to observe and/or interact with their internal and external environments. In recent years, the advent of deep learning has sparked interest in the adoption of deep neural networks for learning latent representations of IoT sensor data. Du *et al.* [5] introduced a deep neural network using LSTM to model a system log as a natural language sequence by learning log patterns from normal execution in order to detect anomalies when log patterns deviate from the model trained from log data under normal execution. Zhu *et al.* [6] proposed a multi-task anomaly detection framework for control area network messages using LSTM on in-vehicle network data. Zhang *et al.* [7] also used an LSTM model to predict power station working conditions from data generated by industrial IoT sensors of a main pump in a power station. However, all the aforementioned approaches are centralized, as data generated by IoT devices are sent directly to the server, raising privacy concerns and resulting in network overload. Moreover, the training process of centralized approaches is bandwidth intensive and comes with significant privacy implications.

More recently, federated learning has emerged as a viable, compelling alternative to the centralized approach. Rather than aggregating increasingly large amounts and types of data into a central

location, federated learning distributes the global model training process such that each participating sensor’s data is used in situ to train a local model. Nguyen *et al.* [8] proposed an anomaly detection system for detecting compromised IoT devices using federated learning by aggregating anomaly-detection profiles for intrusion detection. Li *et al.* [9] presented an autoencoder based anomaly detection approach at the server side to detect anomalous local weight updates from the clients in a federated learning system. Yurochkin *et al.* [10] developed a probabilistic federated learning framework with a particular emphasis on training and aggregating neural network models by decoupling the learning of local models from their aggregation into a global federated model. Zhao *et al.* [11] presented a multi-task federated learning method for computer networks anomaly detection, as well as for traffic recognition and classification tasks. Chen *et al.* [12] introduced a federated multi-task hierarchical attention model for activity recognition and environment monitoring using multiple sensors.

2.3 Method

Motivated by LSTM networks’ capability of learning long-term dependencies between time steps of sequence data [79], we introduce in this section a novel privacy-by-design federated stacked LSTM model for anomaly detection in smart buildings using IoT sensor data. The proposed approach leverages the multi-task learning paradigm.

In a smart building setting with a federated learning system setup, we assume that there are K sensors, each of which has a dataset $\mathcal{D}_k$ that is kept private, where $k = 1, \dots, K$. In other words, the data $\mathcal{D}_k$ of the k -th sensor are not shared with the server. An IoT-enabled smart building is depicted in Figure 2.1, which shows several types of sensors for a variety of tasks, including lighting control, building access, water management, HVAC, fire suppression, and building monitoring. Unlike the traditional centralized learning that collects and uses all local data $\mathcal{D} = \cup_{k=1}^K \mathcal{D}_k$ from all sensors to train a model, the federated learning paradigm only collects and aggregates updated local models from the sensors to generate a global model.

An overall schematic layout of our federated learning system for anomaly detection is illustrated in Figure 2.2. The proposed system uses historic sensor data and contextual features to identify temporal anomalies in smart buildings using a multi-task federated recurrent neural network to classify anomalous sensors and predict energy consumption. The main building blocks of this system are local training, cloud aggregation, anomaly detection and global model broadcast.

Local training. Sensor data represent a time stamped input of energy consumption data recorded at regular time intervals. These sensor data are often noisy and incomplete due largely to faulty

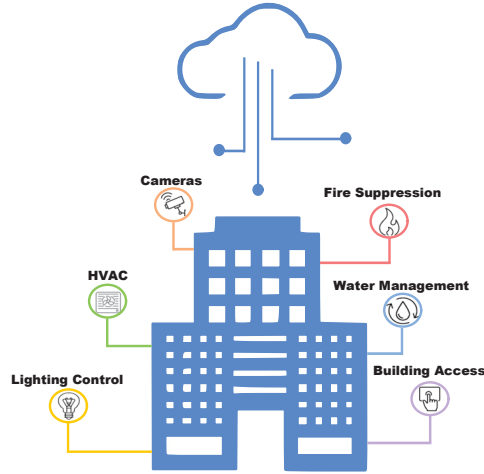


Figure 2.1: IoT-enabled smart building.

devices and/or communication errors [80]. To mitigate the negative impact of noisy and incomplete data on the performance of the federated learning system, these data instances are usually discarded or interpolated. Each sensor trains the local model on anomalous instances. The basic assumption is that historic sensor data are predominantly normal. The historical real datasets can then be split into training and test sets with the objective of using the test data to evaluate the capacity of the anomaly detection framework in order to identify normal behavior during the anomaly detection phase. The primary goal of training is to enhance engine recognition of normal input data patterns. At the end of the training phase, the participating sensors send the learned parameters of their local models for aggregation to a central server via Wireless Area Controller gateways that monitor and control data traffic between the sensors and the IoT system in the cloud [81].

Cloud aggregation. We use PySyft, a Python library for secure and private deep learning [82], which integrates federated learning with secure multi-party computation and differential privacy in an effort to protect against threats within the data center by ensuring that individual sensors' updates remain encrypted in memory. The secure multi-party computation protocol leverages encryp-

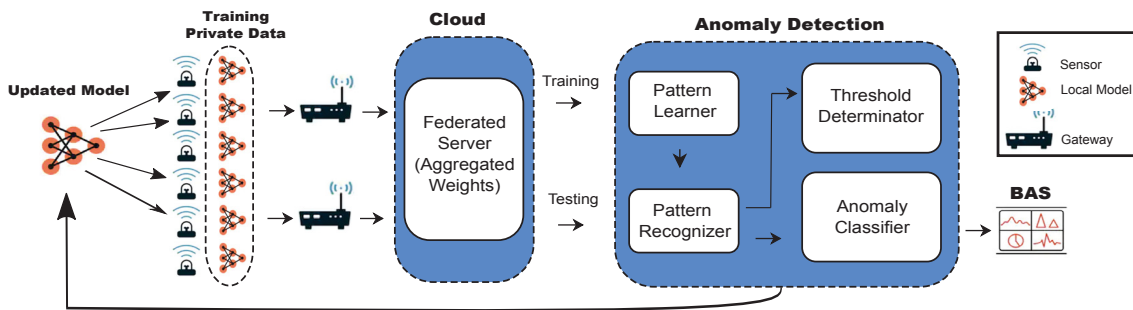


Figure 2.2: Federated learning system for anomaly detection.

tion to make individual sensors' updates uninspectable by a server [24]. At each communication round between the server and participating sensors, the server aggregates the learned parameters of the local models using the Federated Averaging algorithm, followed by updating the global model. This training process is repeated for a certain number of communication rounds until a desirable level of performance is achieved. Then, the updated global model is prepared for integration into the anomaly detection engine.

In PySyft, private data leakage is mitigated with secure aggregation [24] by leveraging secure multiparty computation to compute sums of local model parameter updates from individual sensors while maintaining privacy guarantees. With secure aggregation, the local model parameter updates are kept encrypted and their sum is only revealed to the server after a sufficient number of communication rounds. Homomorphic encryption and differential privacy can also be used in PySyft. However, only secure aggregation is used in our algorithm. In differential privacy, for instance, each sensor adds a carefully calibrated amount of noise to its local parameter update in an effort to mask its contribution to the learned global model.

Anomaly detection. The anomaly detection block of the proposed system is composed of four main components: the pattern learner process, pattern recognizer, threshold determinator, and anomaly classifier. The pattern learner receives the global model after the training has been completed and configured for testing. The pattern recognizer tests the global model and evaluates the performance based on the test data to help the threshold determinator find a suitable value for the specific problem. Based on the value of the threshold, the anomaly classifier generates the inference and predicts the anomalous data points that did not pass the threshold determinator and flags them in the anomaly detection system. The updated global model will then be ready for deployment on sensors after this round of training.

Global model broadcast. Once the model weights are updated using the Federated Averaging algorithm, the anomaly detection engine, located in the IoT system, sends back the updated global model to all sensors that are involved in the first round of training. These sensors receive the updated global model with new weights and then replace their local models' parameters with the global parameters to start a new round of training.

2.3.1 Multi-Task Learning

We leverage the multi-task learning paradigm, which aims at solving multiple tasks simultaneously while taking advantage of the similarities and differences across tasks, to formulate the anomaly detection problem in a federated learning setting. Our notation is summarized in Table 2.1.

Table 2.1: Notation.

Notation	Description
K	Number of participating sensors indexed by k
$\mathbf{X}^k$	Training data for the k -th sensor with N_k local examples
$\mathbf{y}^k$	Ground-truth label or target output vector
$\hat{\mathbf{y}}^k$	Predicted probability or response vector
$\mathcal{L}(\cdot, \cdot)$	Loss function
$\sigma(\cdot)$	Sigmoid activation function
B	Local minibatch size
E	Number of local epochs
w_G	Global model
FC	Fully Connected layer
BA	Balanced Accuracy
LR	Logistic Regression
FLR	Federated Logistic Regression
FGRU	Federated Gated Recurrent Unit
LSTM	Long Short-Term Memory network
FLSTM- ℓ	Federated LSTM with ℓ layers
FSLSTM	Federated Stacked LSTM (i.e. FLSTM-3)

Suppose that we have K learning tasks (i.e. one task per sensor) and denote by $(\mathbf{X}^k, \mathbf{y}^k)$ the training data for the k -th task, where $\mathbf{X}^k = (\mathbf{x}_1^k, \dots, \mathbf{x}_{N_k}^k)^T \in \mathbb{R}^{N_k \times F}$ is a data matrix consisting of N_k samples collected from multiple sensors of the same category (e.g. occupancy sensors), and $\mathbf{y}^k = (y_1^k, \dots, y_{N_k}^k)^T \in \mathbb{R}^{N_k}$ is a vector of outputs for the N_k samples. The training samples in $\mathbf{X}^k$ are generated by sensors connected to different gateways in the building. Further, we assume that the samples have the same feature dimension F across all tasks. It is important to note that $y_i^k \in \mathbb{R}$ for regression tasks, while $y_i^k \in \{0, 1\}$ for binary classification tasks with 0 and 1 representing “normal” and “anomalous” observations, respectively.

The goal of multi-task learning is to learn the weight parameters of a model by minimizing the following objective function across all sensors

$$\mathcal{E}_{\text{avg}} = \frac{1}{K} \sum_{k=1}^K \mathcal{L}(\mathbf{y}^k, \hat{\mathbf{y}}^k), \quad (2.1)$$

where $\mathcal{L}$ is a loss function and $\hat{\mathbf{y}}^k$ is a vector of predicted values by the model for the k -th task.

For binary classification tasks, we train a model to minimize the cross-entropy loss function

given by

$$\mathcal{L}(\mathbf{y}^k, \hat{\mathbf{y}}^k) = -\frac{1}{N_k} \sum_{i=1}^{N_k} y_i^k \log(\hat{y}_i^k), \quad (2.2)$$

where y_i^k and $\hat{y}_i^k$ are the ground-truth label and predicted probability, respectively, for the k -th task.

For regression tasks, we train a model to minimize the mean squared error (MSE) given by

$$\mathcal{L}(\mathbf{y}^k, \hat{\mathbf{y}}^k) = \frac{1}{N_k} \sum_{i=1}^{N_k} (y_i^k - \hat{y}_i^k)^2, \quad (2.3)$$

where y_i^k and $\hat{y}_i^k$ are the target output and predicted value by the model, respectively, for the k -th task.

2.3.2 LSTM

LSTM networks are a special type of recurrent neural networks (RNNs), capable of learning long-term dependencies between time steps of sequence data while being resilient to the vanishing gradient problem [79]. The key to an LSTM network is the cell state, which contains information learned from the previous time steps and has the ability to remove or add information using gates [83]. These gates control the flow of information to and from the memory. In addition to the hidden state, the architecture of an LSTM block is composed of a cell state, forget gate, memory cell, input gate and output gate, as illustrated in Figure 2.3. At each time step, the LSTM block takes as input the current input data vector $\mathbf{x}_t$ and both the hidden state (i.e. short-term memory) $\mathbf{h}_{t-1}$ and cell state (i.e. long-term memory) $\mathbf{c}_{t-1}$ from the previous cell. In order to decide which information to be retained or discarded at each time step before passing on the long-term and short-term information to the next cell, the LSTM block uses the forget, input and output gates, which are trainable functions with weights and biases. The forget gate decides which information from the long-term memory to forget, while the input gate can be regarded as a filter that selects what information can be kept and what information to be thrown out. The memory cell $\mathbf{g}_t$ is created by passing the current input and short-term memory into a tanh activation function, which is a shifted version of the sigmoid activation function. The new cell state $\mathbf{c}_t$ is obtained by adding two pointwise multiplication terms; the first term involves the input gate and memory cell, while the second one uses the forget gate and the previous cell state. The cell state $\mathbf{c}_t$ stores information about the input data across time steps. Finally, the hidden state $\mathbf{h}_t$ is obtained via pointwise multiplication of the output gate $\mathbf{o}_t$ and the new cell state through a tanh activation function. This hidden state (i.e. new short-term memory) is then passed on to the cell in the next time step.

Formally, given the input $\mathbf{x}_t$, current cell state $\mathbf{c}_{t-1}$ and hidden state $\mathbf{h}_{t-1}$ of the network, the LSTM updates at time step t are given by

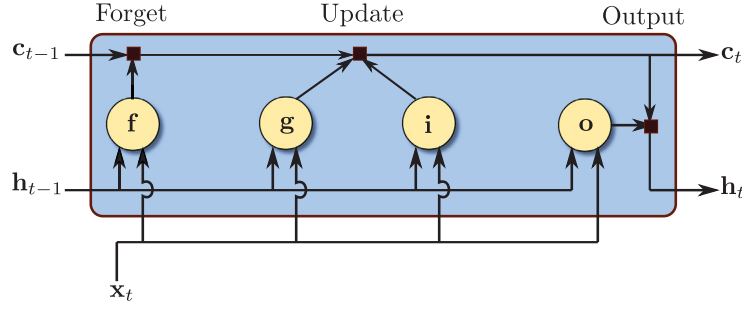


Figure 2.3: LSTM block architecture.

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{R}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{g}_t &= \tanh(\mathbf{W}_g \mathbf{x}_t + \mathbf{R}_g \mathbf{h}_{t-1} + \mathbf{b}_g) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{R}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{R}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{2.4}$$

where $\mathbf{f}_t$, $\mathbf{g}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$, $\mathbf{c}_t$ and $\mathbf{h}_t$ are the forget gate, memory cell, input gate, output gate, cell state and hidden state, respectively; $\sigma(\cdot)$ denotes the sigmoid activation function; $\odot$ denotes the point-wise product; $\mathbf{W}_\bullet$ and $\mathbf{R}_\bullet$ are the learnable input and recurrent weight matrices; and $\mathbf{b}_\bullet$ are the learnable bias vectors.

In summary, the input gate controls what new information is added to cell state from current input, while the forget gate controls what information to throw away from memory. The output gate controls what information encoded in the cell state is sent to the network as input in the following time step. An LSTM network with multiple LSTM layers is referred to as a stacked or deep LSTM, with the output sequence of one LSTM layer forming the input sequence of the next.

2.3.3 Proposed Framework

In order to overcome the centralized training issues such as computational demand, IoT device availability and network bandwidth limitation, we introduce a federated stacked long short-time memory (FSLSTM) model that contains two main components: a local model and a global model. The local model is a stacked LSTM network composed of three LSTM layers, and is applied on input data generated by each sensor to learn a latent feature representation. A fully connected (FC) layer is applied to the hidden representation of the last LSTM layer, followed by a softmax or linear activation function for classification and regression tasks, respectively. The global model,

on the other hand, aggregates the weights of all local models after each round. The architecture of the proposed FSLSTM model consists of three LSTM layers, as illustrated in Figure 2.4.

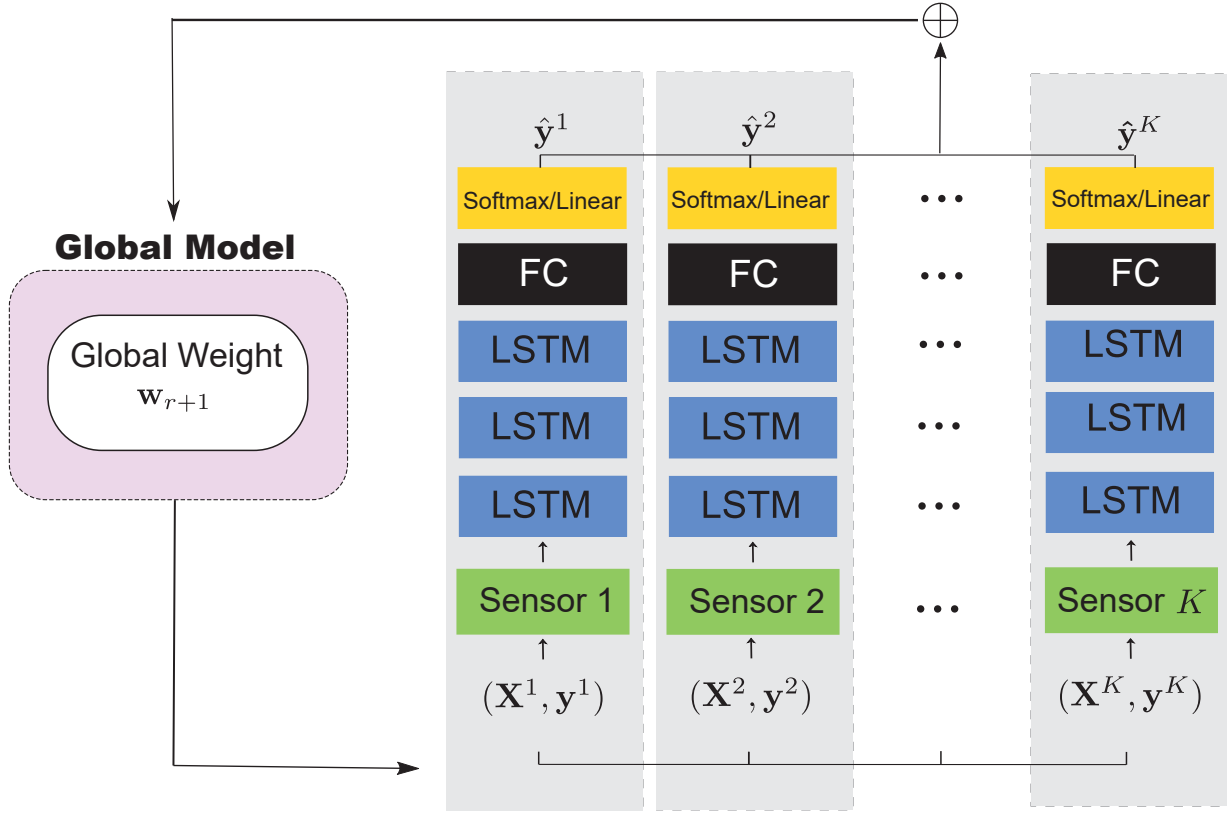


Figure 2.4: Architecture of the proposed federated stacked LSTM model. Only the local model parameter updates are sent back to the server.

In a federated learning setting, the training of a stacked LSTM model is distributed across the participating sensors by iteratively aggregating local models into a joint global model. Suppose we have training local datasets $\{(\mathbf{X}^k, \mathbf{y}^k)\}_{k=1}^K$ from K sensors, where the total number of samples distributed over these sensors is $N = \sum_{k=1}^K N_k$, with N_k denoting the number of samples in the training set $\mathcal{D}_k = (\mathbf{X}^k, \mathbf{y}^k) = \{(\mathbf{x}_i^k, y_i^k)\}_{i=1}^{N_k}$ of the k -th sensor. We consider a federated learning task, where these K sensors collaboratively train a model parameter vector $\mathbf{w}$ with the orchestration of a remote server. The goal is to minimize the following global loss function on all the distributed datasets

$$f(\mathbf{w}) = \sum_{k=1}^K \frac{N_k}{N} f_k(\mathbf{w}), \quad (2.5)$$

where $f_k(\mathbf{w})$ is the local loss function on the collection of data samples $\xi_i^k = (\mathbf{x}_i^k, y_i^k)$ from the k -th sensor

$$f_k(\mathbf{w}) = \frac{1}{N_k} \sum_{i=1}^{N_k} \ell(\mathbf{w}; \xi_i^k), \quad (2.6)$$

and $\ell(\mathbf{w}; \xi_i^k)$ is the loss function of the prediction on the data sample ξ_i^k made with model parameter vector $\mathbf{w}$. It is important to point out that in the case of the IID assumption (i.e. the training examples are distributed over the clients uniformly at random), we have $\mathbb{E}_{\mathcal{D}_k}[f_k(\mathbf{w})] = f(\mathbf{w})$, where the expectation is taken over the set of samples assigned to a fixed sensor k .

Each local LSTM model receives a local copy of the global model weights based on a time series window with step t .

Local model. Each participating sensor uses its local data to update parameters of the local model, which is a stacked LSTM network with three layers. As stated earlier, at each time step t , the LSTM block takes as input the current input data vector $\mathbf{x}_t$ and both the hidden state $\mathbf{h}_{t-1}$ and cell state $\mathbf{c}_{t-1}$ from the previous cell. Then, the LSTM network learns to predict the hidden and cell states of the next time step as follows:

$$\mathbf{h}_t, \mathbf{c}_t = \text{LSTM}(\mathbf{h}_{t-1}, \mathbf{x}_t, \mathbf{c}_{t-1}; \mathbf{w}), \quad (2.7)$$

where LSTM is an operator representing the operations in Eq. (2.4), $\mathbf{h}_t$ is the new hidden state, $\mathbf{c}_t$ is the new cell state, and $\mathbf{w}$ is the parameter vector of the LSTM model. We initialize $\mathbf{c}_0$ and $\mathbf{h}_0$ as random vectors.

The output of the last hidden state is the hidden vector of the last time step, and can be viewed as the representation of the whole sequence. Then, we pass this hidden representation to a fully-connected layer to get the predicted values for the k -th task as follows:

$$\hat{\mathbf{y}}^k = \sigma(\mathbf{W}^{\text{FC}} \mathbf{h}_{\text{last}} + \mathbf{b}^{\text{FC}}), \quad (2.8)$$

where σ is the softmax or linear activation function, $\mathbf{h}_{\text{last}}$ denotes the hidden vector in the last time step of LSTM, $\mathbf{W}^{\text{FC}}$ and $\mathbf{b}^{\text{FC}}$ are the learnable weight matrix and bias vector of the FC layer.

Global model. At each round r , the server randomly chooses a subset S_r of sensors for synchronous aggregation and broadcasts the global model parameter vector $\mathbf{w}_r$ to the participating sensors. For the choice of S_r , the random number generator is initialized with the same seed value for all training rounds. Each sensor updates the local model parameters by minimizing the loss function over its local data dataset, starting from the global model parameter vector $\mathbf{w}_r$ shared by the server and using stochastic gradient descent algorithm for E epochs and with a batch size B . At the end of the training phase, the parameters of local models are sent to the server for aggregation, which is done synchronously using the federated averaging algorithm in order to obtain the global model parameter vector $\mathbf{w}_{r+1}$ for the next round.

The update of local and global models is repeated for a certain number of rounds until the global loss function converges. The main algorithmic steps of our proposed federated stacked LSTM

network are summarized in Algorithm 1. This algorithm starts by randomly initializing the global model. Then, the server randomly selects a subset of sensors and distributes the current global model to these sensors. Each sensor trains the global model with its local data independently, and then the server collects the parameters of the locally trained models for all selected sensors and aggregates them using the federated averaging algorithm by computing a weighted average of these parameters in order to obtain a shared global model.

At the beginning of the training round of communication, each sensor reads the current parameter vector of the global model from the central server and updates it via stochastic gradient descent, where the stochastic gradient computed using a mini-batch sampled uniformly at random from the local dataset of the k -th sensor. At the end of the training round of communication, our proposed federated stacked LSTM returns a vector of predicted values. Then, we concatenate all the predicted outcomes from the K sensors to obtain a vector $\hat{\mathbf{y}} \in \mathbb{R}^N$ as follows:

$$\hat{\mathbf{y}} = \hat{\mathbf{y}}^1 \oplus \dots \oplus \hat{\mathbf{y}}^K, \quad (2.9)$$

where $\oplus$ denotes the concatenation operator.

Based on these predictions, the model sends back answers to the anomaly detection engine, where the values are passed on to the threshold determinator to make decisions regarding anomalous sensors, and subsequently take appropriate actions for isolation and maintenance.

2.4 Experiments

In this section, we conduct experiments to demonstrate and analyze the performance of the proposed FSLSTM model in anomaly detection and regression on three real-world datasets generated by the IoT production system at General Electric Current smart building in Montreal. For privacy concerns, we hashed the primary keys and a few privacy-sensitive features such as vendor name, device identification and some of the prototypes information. The effectiveness of our approach is validated by performing comparison with several baseline methods.

Datasets. We use a Sensors Event Log dataset for anomaly detection and an Energy Usage dataset for electricity consumption. We also use a Weather API dataset in conjunction with energy usage to enrich model learning. The weather data are collected from the building API that displays the basic weather parameters such as temperature, barometric pressure, humidity, precipitation, solar radiation and wind speed from May to August 2019. The weather data are used as a secondary source of information to enrich the Sensors Event Log dataset and help the model predict the energy usage. We simulate each zone in a separate category and allocate a specific task to it for learning.

Algorithm 1 Federated Stacked LSTM

Input: Training sets $\{(\mathbf{X}^k, \mathbf{y}^k)\}_{k=1}^K$ from K sensors, initial global model parameters $\mathbf{w}_0$, local minibatch size B , number of local epochs E , learning rate η , number of rounds R , number of sensors per round m .

Output: Vector $\hat{\mathbf{y}}$ of predicted values.

```
1: for  $r = 1$  to  $R$  do
2:   Server randomly selects a subset  $S_r$  of  $m$  sensors.
3:   Server broadcasts  $\mathbf{w}_r$  to the subset  $S_r$ 
4:   for each sensor  $k \in S_r$  in parallel do
5:      $\mathbf{w}_{r+1}^k \leftarrow \text{SensorUpdate}(k, \mathbf{w}_r)$ 
6:   end for
7:    $\mathbf{w}_{r+1} \leftarrow \sum_{k=1}^K \frac{N_k}{N} \mathbf{w}_{r+1}^k$  //aggregate updates
8:   end for //run on sensor k
9:    $\mathcal{B} \leftarrow$  partition local data  $(\mathbf{X}^k, \mathbf{y}^k)$  into batches of size  $B$ 
10:  for each local epoch from 1 to  $E$  do
11:    for mini-batch sample  $\xi \in \mathcal{B}$  do
12:       $\mathbf{h}_t, \mathbf{c}_t = \text{LSTM}(\mathbf{h}_{t-1}, \mathbf{x}_t, \mathbf{c}_{t-1}; \mathbf{w})$  //LSTM at time step t
13:       $\hat{\mathbf{y}}^k = \sigma(\mathbf{W}^{\text{FC}} \mathbf{h}_{\text{last}} + \mathbf{b}^{\text{FC}})$  //predicted outcomes
14:       $\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{B} \sum_{\xi \in \mathcal{B}} \nabla \ell(\mathbf{w}; \xi)$  //update local model
15:    end for
16:  end for return learned parameter vector  $\mathbf{w}$  to server
17: Concatenate predicted values:  $\hat{\mathbf{y}} \leftarrow \hat{\mathbf{y}}^1 \oplus \dots \oplus \hat{\mathbf{y}}^K$ 
```

The Sensors Event Log and Energy Usage datasets are generated by 180 sensors from 5 different categories: lights, HVAC thermostats, occupancy sensors, water leakage sensors and access sensors. The distribution of these categories is shown in Figure 2.5. We use these categories for classification tasks, as the target is to identify which sensor is faulty, rendering it unable to properly communicate with HVAC and other control devices. In addition, the energy usage metric of thermostat sensors is used for regression tasks, with the aim of predicting future energy consumption of IoT devices, such as temperature, humidity and pressure sensors which are the most commonly used sensors for HVAC and building equipment applications. The pre-processing phase of data generation and wrangling are performed to ensure the confidentiality of the datasets used in our experiments and the privacy of the company’s infrastructure.

- **Sensors Event Log Dataset:** Sensors data such as occupancy sensors, lights, thermometer and humidity are collected from 180 devices. These sensors are categorized into 5 different groups and are distributed all over the building. In our experiments, we select 1 million event logs with a window time of 4 months. We limit data collection to one particular season (summer in our case) to learn feature vectors of time and frequency domain variables for

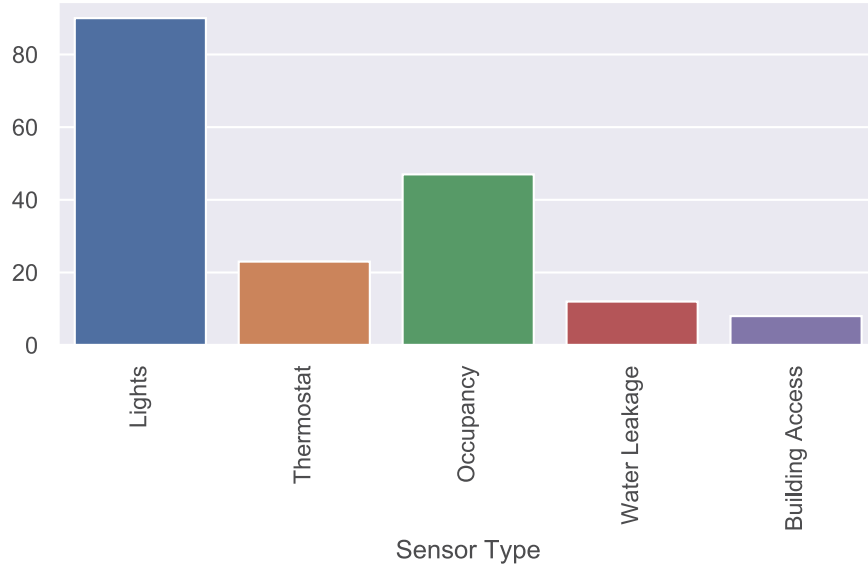


Figure 2.5: Distribution of sensors' categories.

this specific period of the year. We believe that each season should be considered separately for better learning purposes. For example, the heating pattern during the winter season is different from the cooling stages during summer. We consider each sensor as a separate task and predict its activities such as drop in temperature, excessive energy usage and running water.

- **Energy Usage Dataset:** Energy data gathered from sensors measure the electricity consumed per device. In our case, appliances can represent any equipment that is connected to the building automation system and falls under the five aforementioned categories. The unit is measured in kW/h for a variety of appliances, including LED light bulbs, rooftop units, humidity sensors, smart sensors that capture water leakage events, and occupancy activities around the building. The latter are sophisticated sensors, which are equipped with a smart dashboard for data analytics and different measuring tools to monitor important indicators related to indoor farms and laboratories. The data are aggregated every 15 minutes, stored in the Energy Usage table, and then merged into the Event Log dataset based on the sensor ID.

Baseline methods. We evaluate the performance of the proposed FSLSTM network consisting of three LSTM layers against several baseline models, including centralized logistic regression (LR), LSTM, federated logistic regression (FLR), federated gated recurrent unit (FGRU), federated LSTM with one LSTM layer (FLSTM-1), and federated LSTM with two LSTM layers (FLSTM-2). Both LSTM and GRU networks address the vanishing/exploding gradient problem

of traditional recurrent neural networks. While the LSTM block consists of three gates (forget, input and output), the GRU block has only two gates (update and reset). The update gate in the GRU block determines the amount of previous information that needs to be passed along the next state, while the reset gate decides how much of the past information is needed to neglect. It is worth pointing out that we designed the federated logistic regression and federated GRU baseline models for comparison purposes with FSLSTM, whereas the centralized logistic regression and LSTM are well-known prediction baselines in the literature. Our proposed framework leverages the popular federated averaging algorithm, which uses stochastic gradient descent as both sensor and server optimizers to update the local and global parameters, where the server learning rate is equal to 1. More recently, Sashank *et al.* [84] have introduced an adaptive optimization framework, in which federated versions of popular adaptive algorithms are incorporated for the sensor-side or server-side model updates.

Implementation details. All experiments are carried out on a Linux desktop computer running 4.4GHz and 64GB RAM with an NVIDIA GeForce RTX 2080 Ti GPU. The algorithms are implemented in PyTorch. The hyper-parameters are optimized using grid search. For fair comparison, we set the number of hidden units to 128 per layer in both LSTM and GRU blocks. We also set the number of nodes in the fully connected layer to 100. In addition, we use regular and recurrent dropouts of 20% and apply the ReLU activation function in an effort to avoid overfitting. Our grid search selects a batch size of 1024 for training and an initial learning rate of 0.001 as the best combination. We use cross-entropy and MSE as loss functions for classification and regression tasks, respectively. In all experiments, we split our datasets into 80% training and 20% testing.

2.4.1 Results

The effectiveness of our FSLSTM model (i.e. FLSTM-3) is assessed by conducting a comprehensive comparison with the baseline methods using several performance evaluation metrics. The results are summarized in Table 2.2.

For regression tasks, we use the mean absolute error (MAE), mean squared error (MSE) and root mean squared error (RMSE) as evaluation metrics, which are given by

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^m |y_i - \hat{y}_i|, \quad (2.10)$$

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2, \quad (2.11)$$

Table 2.2: Performance comparison results of FSLSTM against baseline models on the Sensors Event Log test set for anomaly detection (classification) and Energy Usage test set (regression). Boldface numbers indicate the best performance.

Methods	Sensors Data (Classification)				Energy Usage (Regression)		
	Precision	Recall	F1	BA	MAE	MSE	RMSE
LR	0.57	0.60	0.52	0.72	0.341	0.48	0.692
LSTM	0.66	0.61	0.58	0.71	0.243	0.33	0.574
FLR (ours)	0.65	0.71	0.70	0.69	0.339	0.34	0.583
FGRU (ours)	0.84	0.66	0.59	0.80	0.211	0.29	0.538
FSLSTM (ours)	0.89	0.79	0.87	0.90	0.162	0.19	0.435

and $RMSE = \sqrt{MSE}$, where m is the number of samples in the test set, y_i is the actual (ground truth) value, and $\hat{y}_i$ is the model’s predicted value. A small value of these error metrics indicates a better performance of the model. We rely on a 15 minute window aggregation to reduce buffer overflow over the network and also to remove missing values that are sometimes generated by some sensors due to inactivity. As shown in Table 2.2, FSLSTM significantly outperforms the baselines methods on the Energy Usage datasets. In terms of MAE, for example, FSLSTM yields a much smaller error compared to the one obtained by LSTM, and more than half the error obtained by LR and FLR. Moreover, FSLSTM yields a 4.9 percentage points performance improvement over FGRU.

For classification tasks, we use precision, recall, F1 score, balanced accuracy (BA), receiver operating characteristic (ROC) curve and area under the ROC curve (AUC) as evaluation metrics. Precision and recall are defined as

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{and} \quad \text{Recall} = \frac{TP}{TP + FN}, \quad (2.12)$$

where TP, FP, TN and FN denote true positives, false positives, true negatives and false negatives, respectively. TP is the number of correctly predicted anomalous observations, while TN is the number of correctly predicted normal observations. Recall, also known as true positive rate (TPR), is the percentage of positive instances correctly classified, and indicates how often a classifier misses a positive prediction.

AUC summarizes the information contained in the ROC curve, which plots TPR versus $FPR = FP/(FP + TN)$, the false positive rate, at various thresholds. Larger AUC values indicate better performance at distinguishing between anomalous and normal observations. FPR is the ratio of normal observations that were incorrectly classified as anomalous.

Since our datasets are highly imbalanced, we use the balanced F1 score defined as the harmonic

mean of precision and recall, and the balanced accuracy given by

$$BA = \frac{TPR + TNR}{2}, \quad (2.13)$$

where $TNR = 1 - FPR$, also called specificity, refers to the ability of a classifier to correctly identify the observations that are not anomalous.

As shown in Table 2.2, the proposed model outperforms all the baseline methods by a large margin in terms of F1 score, yielding performance improvements of 29 percentage points over LSTM and 28 percentage points over FGRU. In terms of balanced accuracy, FSLSTM also achieves notable performance improvements of 19 percentage points over LSTM and 10 percentage points over FGRU. This demonstrates the significant prediction ability of FSLSTM in anomaly detection. It is also important to note that LSTM yields comparable performance to FLR, due in large part to the fact that linear models cannot effectively represent temporal dependency, even in a federated learning setting.

Figure 2.6 displays the ROC curves, which show the better performance of FSLSTM in comparison with baseline methods on the Sensors Event Log test set. Each point on ROC represents different trade-off between false positives and false negatives. An ROC curve that is closer to the upper right indicates a better performance (TPR is higher than FPR). The overall performance of FSLSTM is significantly better than the baselines, as indicated by both the ROC curves and AUC values.

Federated vs. centralized learning. The LSTM model adopts a centralized approach, which requires the training data to be aggregated on the server, meaning that no training is performed on the edge devices. The learning is carried out by partitioning the data into training and test sets in a central machine. However, centralized training is privacy-intrusive, especially for clients with important sensors' data aggregated on the cloud, as some sensors may contain private patterns, such as occupancy and access information. In a centralized setting, sensors have to trade their privacy by sending and storing data on the cloud owned by a third-party service provider.

Unlike the centralized training framework, our federated learning based approach is decentralized and enables sensors located at different locations inside the building to collaboratively learn a federated model, while keeping all data that may contain private information on devices. Therefore, sensors can benefit from using a federated model without sending raw data to the cloud. Sensitive data may include the sensor's model number, manufacturer name or even serial numbers of newly produced prototypes that are still in the testing phase. Another limitation of centralized learning is the computing power [85]. In our setting, however, the computing power for each sensor is negligible compared to dedicated GPUs. In a centralized learning environment, a large amount

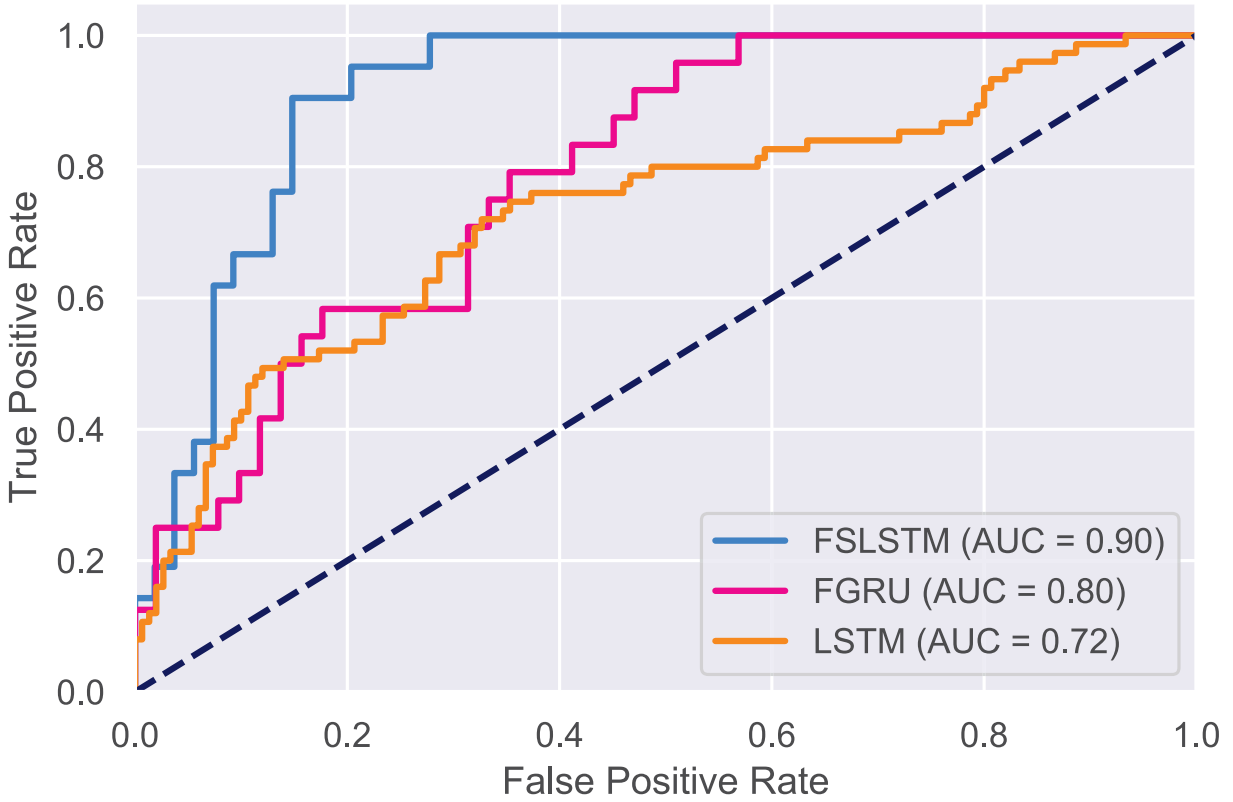


Figure 2.6: ROC curves for FSLSTM and baseline methods, along with the corresponding AUC values, on the *Sensors Event Log* test set.

of data collected from different IoT devices need to be merged into one dataset and then we wait for the machine to finish training. In our proposed federated framework, we train the model on the data of each device in parallel, resulting in better performance and faster convergence time. The better performance of our federated stacked LSTM model may be attributed not only to the strong capabilities of LSTM in prediction tasks, but also to the stacked LSTM layers that help generate a deep feature representation of the input data.

The ability to quickly detect and respond to anomalies is critical to the successful operation of smart buildings. Models trained on data of individual devices help improve critical incidents detection, such as technical glitches that may arise in smart buildings, where smart fire sensors, for instance, need not only be triggered in the case of an emergency, but also be capable of predicting specific scenarios and patterns. During the training phase, our FSLSTM model converges twice as fast as the centralized LSTM on the same datasets. As shown in Figure 2.7 (top), the centralized LSTM model does not seem to attain a stable state even after 50 epochs. The federated stacked LSTM model, on the other hand, is capable of significantly reducing the loss function and reaching a stable performance with higher accuracy in only 20 epochs, as shown in Figure 2.7 (bottom).

Each epoch represents a full round of the 180 sensors that are participating in the experiment. Moreover, notice that the learning curves of the federated model are less fluctuating compared to the centralized LSTM. This smoothness property is attributed, in large part, to the number of sensors involved in the learning process.

Convergence performance. An important factor in our experiments is the convergence speed of the model. We test the effect of the parameter K (i.e. number of sensors) on the convergence performance of the proposed approach. The results of convergence time for FSLSTM, FGRU and LSTM using a varying number of sensors on the Sensors Event Log dataset are shown in Figure 2.8. Note that unlike the centralized LSTM model, the convergence time of both federated models decreases when the number of sensors increases. Training a deep neural network involves using an optimization algorithm to find a set of weights to best map inputs to outputs, while convergence describes a progression towards a stable state where the network has learned to properly respond to a set of training patterns within some margin of error. The choice of the network’s hyperparameters, such as the number of sensors, play an important role in convergence. In order to assess the performance of FSLSTM with respect to the number of sensors K , we use the mini-batch stochastic gradient descent optimization algorithm with a fixed batch size for the local update by increasing the value of K from 20 to 200. For all sensors, even when the learning rate is tuned carefully, FSLSTM achieves, in all batch size settings, a better performance, as reported in Table 2.2. Figure 2.8 shows that an increase in the number of participating sensors positively impacts the convergence time of FSLSTM, reducing the time from 6 hours for the centralized LSTM to only 2 hours for FSLSTM using 200 sensors. However, a larger number of participating sensors requires higher local computation at the sensors, resulting in an increase in energy consumption by these sensors.

It is also important to point out that there is a slight increase of convergence time when the number of sensors is between 20 and 40, but then the convergence time decreases when the number of sensors exceeds 42. This is largely due to the delay in response of the rooftop units’ thermostats that are connected to the network through the Wireless Area Controller gateways. The latency time of these sensors is twice longer than other sensors in the buildings, leading to a slight increase in convergence time. Once these sensors are equipped with all the global parameters that are necessary for the training phase, the convergence time drops proportionally to the number of participating sensors.

Collective and contextual anomalies. A collective anomaly describes a group of data points that exhibits an anomalous behavior compared to the rest of the dataset [86]. An individual instance within the anomalous group is not necessarily anomalous on its own. If a data instance is

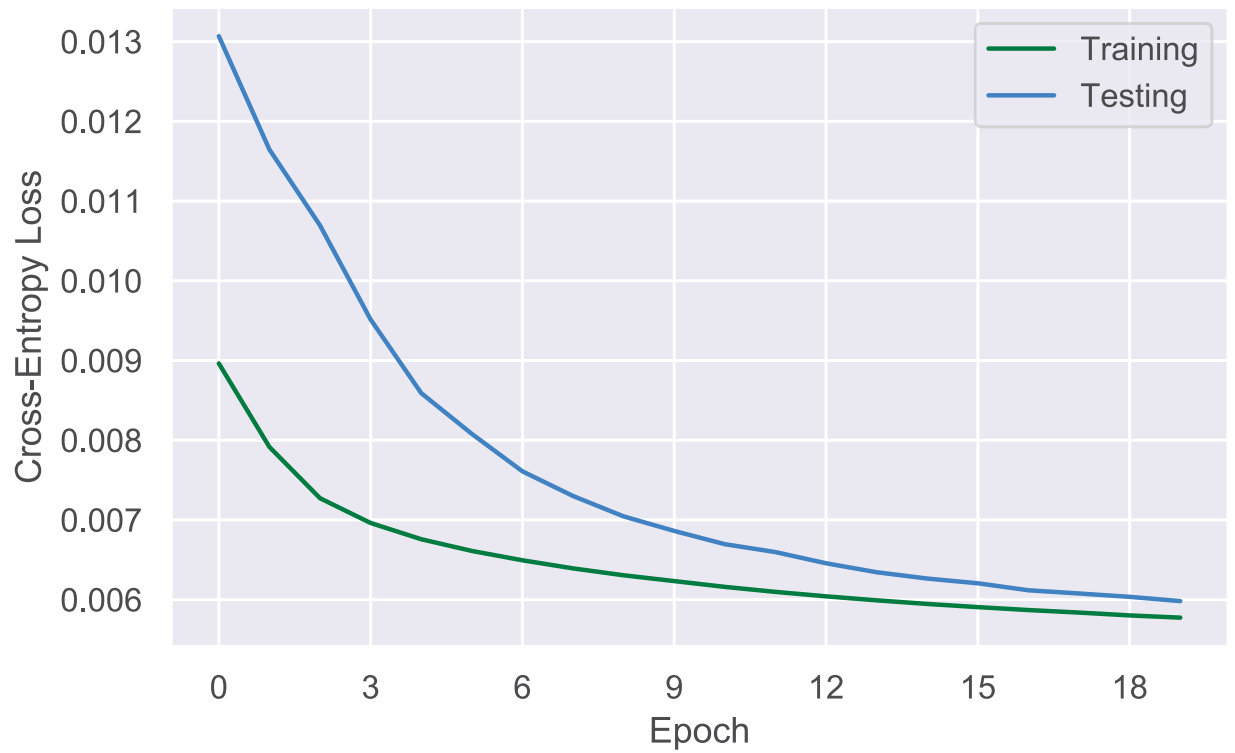
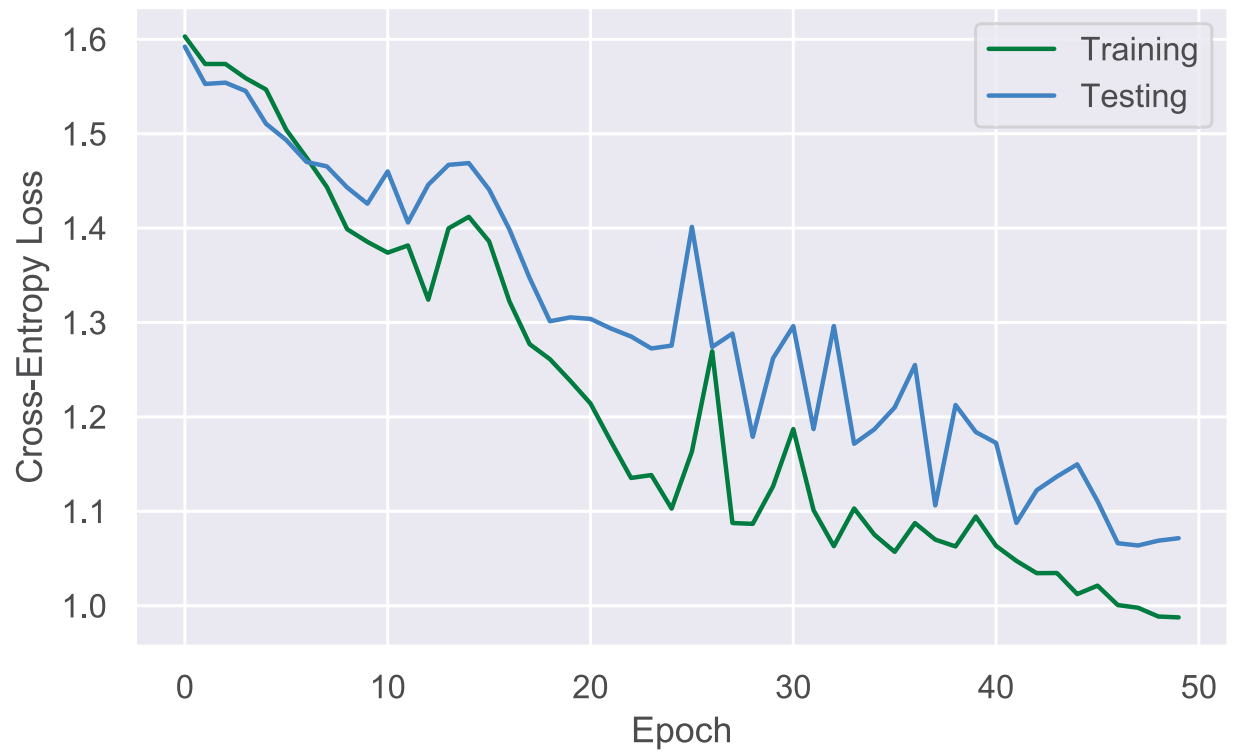


Figure 2.7: Training and testing learning curves of LSTM (top) and FSLSTM (bottom) on the *Sensors Event Log* dataset.

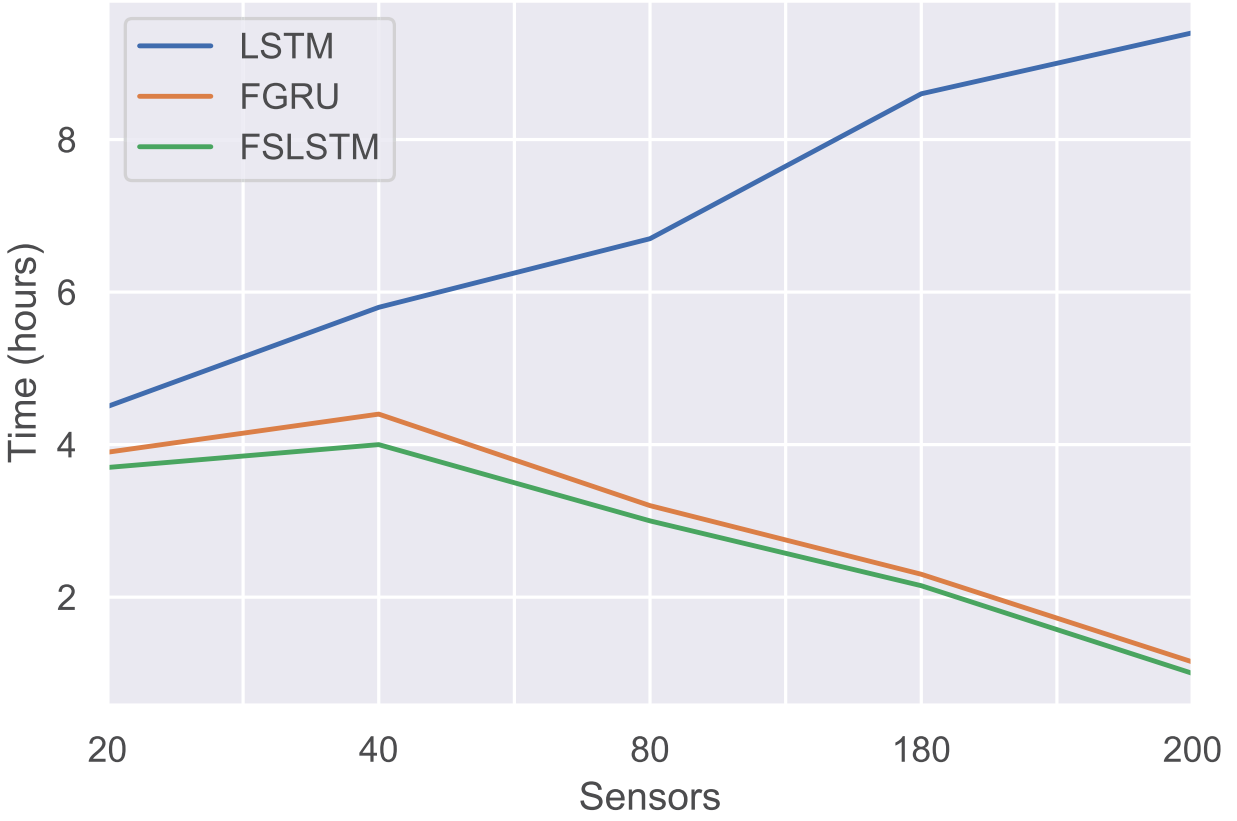


Figure 2.8: Convergence time comparison between FSLSTM, FGRU, and LSTM as a function of the number of sensors on the *Sensors Event Log* dataset.

anomalous in a specific context, but not otherwise, then it is termed as a contextual anomaly, also referred to as conditional anomaly [87]. We train FSLSTM on normal data before performing a live prediction for each time step, which is equal to 600 minutes. Instead of considering each time step separately, the observation of prediction errors from a certain number of time steps is now used for detecting collective anomalies. The prediction errors from a number of the latest time steps above a threshold, as set by the threshold determinator, will indicate a collective anomaly. We take a sample of our initial data, which includes four HVAC sensors, all measuring the electricity from power meters located in different areas within the building. In the first step, FSLSTM determines the point anomalies in real-time, while in the second step, the anomaly detection engine decides if these anomalies are contextual or collective based on different alarm profiles and system rules, as well as specific temperature profiles for each category of devices. Our proposed model efficiently determines context/collection based anomalies in real-time, as reported in Table 2.3.

As expected, we noticed through experimentation that it is possible to obtain a higher accuracy on collective anomalies detection, but the number of false alarms triggered tends to be quite high. As shown in in Table 2.3, FSLSTM yields superior performance by correctly detecting 88% of the

alarms out of 1000 instances in this experiment with only 9 false alarms. On the contextual side, FSLSTM achieves even a better performance of 90% with only 4 false alarms.

Table 2.3: Collective and contextual anomaly detection comparison.

Method	Collective Anomalies		Contextual Anomalies	
	Correct Alarms Triggered (%)	False Alarms Triggered (%)	Correct Alarms Triggered (%)	False Alarms Triggered (%)
LR	56	54	63	48
LSTM	66	33	74	29
FLR (ours)	65	21	78	18
FGRU (ours)	74	12	82	7
FSLSTM (ours)	88	9	90	4

Prediction performance. For regression tasks, we test the performance of FSLSTM on the Energy Usage dataset to predict the building energy consumption on a window time of 600 minutes for two main reasons. First, the confidence interval of the model shows a high accuracy of energy prediction, as illustrated in Figure 2.9, which displays the actual vs. predicted building energy consumption using FSLSTM. Second, it makes more sense from an industrial perspective as 600 min equals 10 hours, which roughly represents a full working day. This time frame provides building managers with plenty of insightful information to anticipate and predict the daily usage of energy, and allows them to plan activities accordingly. As shown in Figure 2.9, the FSLSTM model has a stable performance and is able to predict energy consumption in the building with 90% accuracy.

For the sensors fault detection, we run the classification task on the Sensors dataset to detect anomalous devices. The experiment includes lights, thermostats of rooftop units and water leakage sensors for a window time ranging from 4 to 6 days. Figure 2.10 demonstrates the effectiveness of the FSLTM model in detecting outstanding and malfunctioning behaviors expressed by these sensors at different times of the day. As can be seen, the model is able to capture outliers with high accuracy, along with the corresponding value and time stamp. During working hours (light orange color), the weather temperature (dark gray color) shows an increase during the day due to sunlight and heat, while the site energy demand (green color) is stable during the majority of hours. As shown in Figure 2.10 (top), the weather temperature is higher during working hours than during non-working hours (white bars). During this window time, the site has stable energy demands ranging between 0 and 35 Kw. FSLSTM is capable of capturing an anomalous amount of energy demands, caused by a cluster of malfunctioning devices, highlighting it as an outlier in real-time and eventually recording these devices with their manufacturing information and sending it back

to the federated learning system. Then, the threshold determinator makes the appropriate decision and sends it over the BAS system for maintenance.

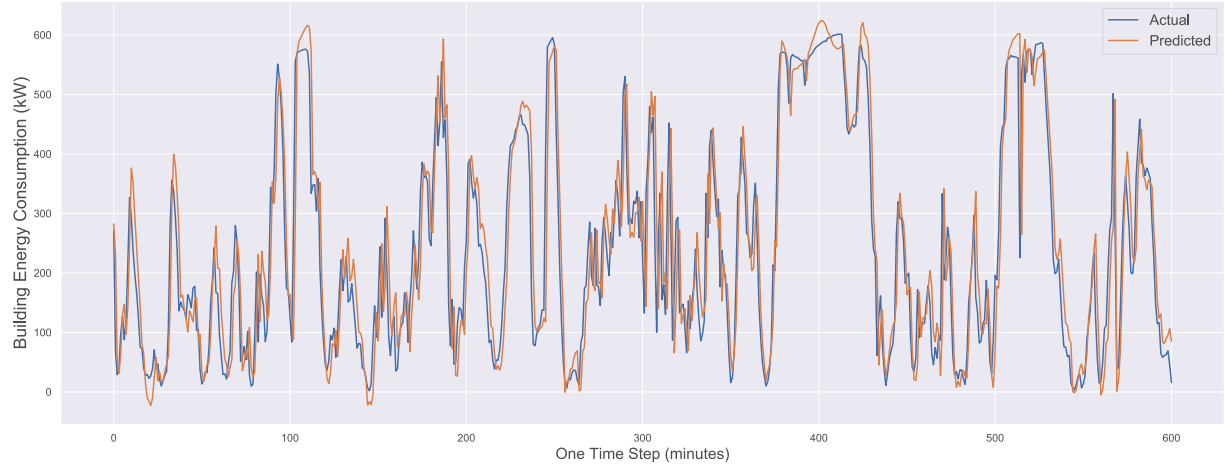


Figure 2.9: Actual vs. predicted building energy consumption using FSLSTM on the Energy Usage test set.

Communication Cost One of the key challenges in our federated learning system is the communication cost, which is typically expressed as a function of data volume, e.g. Megabytes. As shown in Figure 2.11, the proposed FSLSTM model significantly reduces the communication cost compared to the baseline methods on the Sensors Event Log dataset. We argue that the superior performance of FSLSTM over the centralized LSTM and LR models is largely attributed to the effective participation of sensors in the training phase [88]. Federated learning models enable decentralized IoT devices to collaboratively learn a shared prediction model without sending the actual data that is generated during the specific round to the central server. This property of federated learning enhances the communication quality and reduces the cost, while keeping all the training data on device and hence reducing concerns on data security and privacy.

2.4.2 Ablation Study

In order to validate the effectiveness of our FSLSTM framework, we perform an ablation study under different configurations by changing the number of LSTM layers and retraining the models on the same datasets. We design the FLSTM-1 and FSLSTM-2 models by removing one and two LSTM layers, respectively, from the proposed FLSTM architecture, which consists of three LSTM layers. The results are reported in Table 2.4, which shows that both FSLSTM-1 and FSLSTM-2 achieve better performance than LSTM and FGRU. It is important to note that with only a single LSTM layer, the federated LSTM model is able to outperform all baselines methods in terms of



Figure 2.10: Testing FSLSTM for anomaly detection on three sensor types: lights, thermostats, and water leakage. The red dot indicates an anomaly.

balanced accuracy and mean absolute error metrics. We also experimented with more than three LSTM layers, but we did not notice significant performance improvements. In addition, some

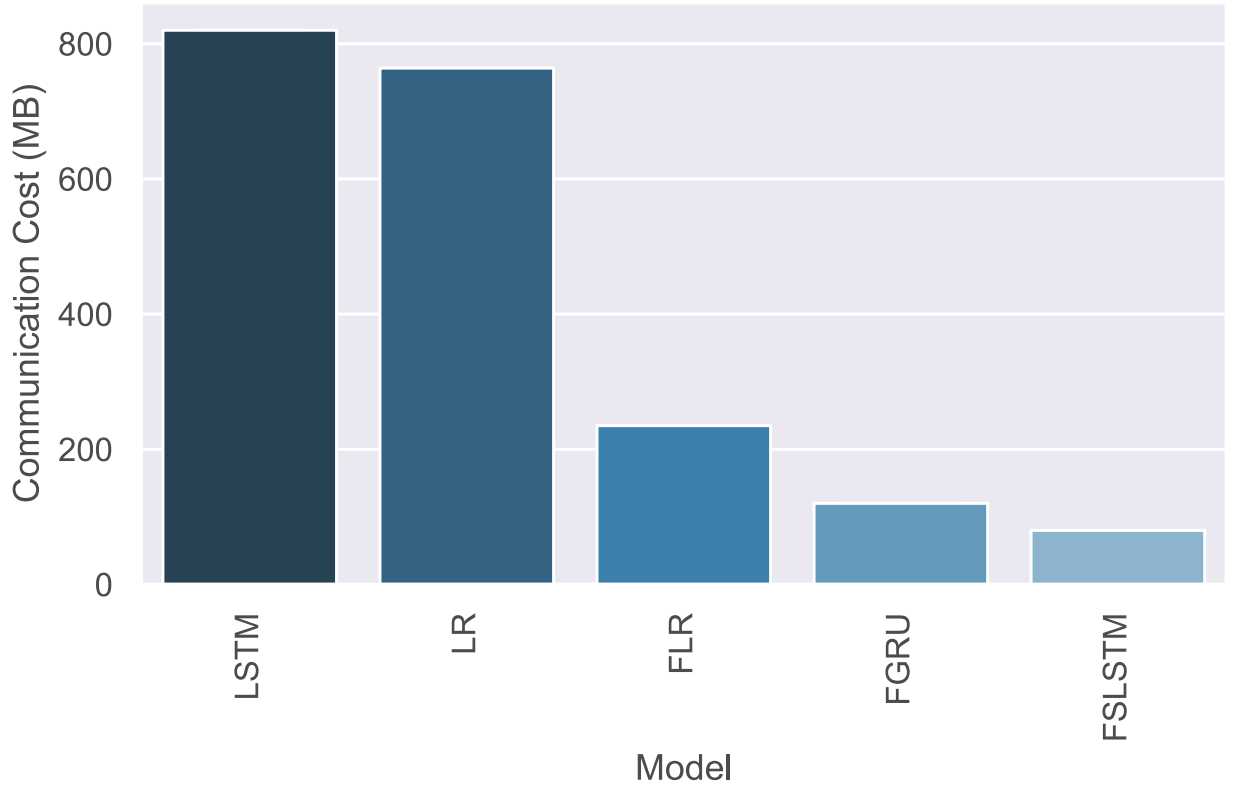


Figure 2.11: Communication cost comparison between FSLSTM and baseline methods, all trained for 50 epochs.

small sensors (e.g. water leakage sensors) are unable to support the growing size of the model.

Table 2.4: Ablation study of FSLSTM on the Sensors Event Log and Energy Usage test sets using different LSTM layers. Boldface numbers indicate the best performance.

Methods	Sensors Data				EU Data		
	Precision	Recall	F1	BA	MAE	MSE	RMSE
FLSTM-1	0.85	0.68	0.69	0.81	0.192	0.24	0.489
FLSTM-2	0.87	0.73	0.71	0.83	0.171	0.21	0.458
FSLSTM	0.89	0.79	0.87	0.90	0.162	0.19	0.435

2.5 Discussion

The proposed *federated stacked LSTM (FSLSTM)* framework for anomaly detection in smart buildings offers several advantages over centralized learning baselines. We highlight its merits in three key aspects: (1) privacy-preserving and task-aware learning across heterogeneous sensors, (2) ac-

celerated convergence with stronger predictive performance for both detection and forecasting, and (3) reduced communication cost that enables practical, on-device deployment in building automation systems.

- *Privacy-Preserving and Task-Aware Learning.* Unlike centralized approaches that aggregate raw sensor data in the cloud, the federated setup trains locally on each device and exchanges only model parameters. This *privacy-by-design* strategy keeps sensitive traces (e.g., occupancy or access patterns) on-device while still enabling collaborative training across sensors. In addition, casting each sensor/zone as a separate task within a multi-task learning formulation allows the model to exploit common structure across devices while preserving device-specific idiosyncrasies. Secure aggregation during cloud orchestration further limits leakage from individual updates.
- *Accelerated Convergence and Predictive Gains.* Stacking multiple LSTM layers yields a deeper temporal representation that improves both classification (fault/anomaly detection) and regression (energy usage forecasting). Empirically, the federated model converges in significantly fewer epochs than its centralized counterpart and achieves higher accuracy: for anomaly detection on the Sensors Event Log, FSLSTM attains an F1 score of **0.87** and balanced accuracy of **0.90**, surpassing centralized LSTM (F1 **0.58**, BA **0.71**). For energy forecasting, FSLSTM reduces error to **MAE 0.162**, **MSE 0.19** and **RMSE 0.435**, improving over centralized LSTM (MAE **0.243**). These gains are accompanied by smoother learning curves and about $2\times$ faster convergence during training.
- *Communication Efficiency and Real-Time Operations.* By transmitting only model updates, the framework lowers communication overhead relative to centralized training, which must stream all raw measurements. In practice, the communication budget decreases substantially while maintaining detection quality. The anomaly-detection pipeline (pattern learner, recognizer, threshold determinator and classifier) supports both *contextual* and *collective* anomalies, yielding high correct-alarm rates with few false alarms. A 600-minute prediction horizon (roughly a working day) provides actionable foresight for facility operations.

The federated design aligns with the realities of large buildings, where sensors are numerous, heterogeneous and sometimes intermittently connected. Training local models in parallel on 100s of devices reduces wall-clock training time and makes it feasible to refresh models frequently (e.g., by season or occupancy schedule). The ability to forecast energy usage 10 hours ahead at high accuracy can inform pre-cooling, load shifting and maintenance dispatch, while near-real-time anomaly flags can be forwarded to the BAS for isolation and remediation.

Despite its advantages, several constraints remain as limitations and trade-offs for FSLSTM, we highlight some of them as follows:

- *Resource constraints on devices.* Small sensors may struggle to host deep models; model size must balance representational power against memory/compute budgets.
- *Heterogeneity and non-IID data.* Device behaviors and usage patterns vary across floors and zones, which can slow or bias aggregation. Robust client sampling and aggregation are important.
- *Latency and connectivity.* Gateways (e.g., wireless area controllers) can introduce delays; a few slower devices may transiently degrade round timing.
- *Imbalanced labels and thresholding.* Fault events are rare. While balanced metrics (F1/BA) improve evaluation, threshold determinators require careful calibration to limit false alarms.
- *Interpretability.* Deep recurrent layers improve accuracy but are difficult to explain to operators. Lightweight attribution or rule extraction around detected events can aid trust.
- *Privacy add-ons.* Stronger protections (e.g., differential privacy or homomorphic encryption) may come with accuracy or efficiency penalties; these need case-by-case tuning.

Federated Large Language Model for Long-Term Time Series Forecasting

Long-term time series forecasting in centralized environments poses unique challenges regarding data privacy, communication overhead, and scalability. To address these challenges, we propose FedTime, a federated large language model (LLM) tailored for long-term time series prediction. Specifically, we introduce a federated pre-trained LLM with fine-tuning and alignment strategies. Prior to the learning process, we employ K-means clustering to partition edge devices or clients into distinct clusters, thereby facilitating more focused model training. We also incorporate channel independence and patching to better preserve local semantic information, ensuring that important contextual details are retained while minimizing the risk of information loss. We demonstrate the effectiveness of our FedTime model through extensive experiments on various real-world forecasting benchmarks, showcasing substantial improvements over recent approaches. In addition, we demonstrate the efficiency of FedTime in streamlining resource usage, resulting in reduced communication overhead.

3.1 Introduction

Long-term time series forecasting involves predicting future values based on historical data over an extended period. The primary goal is to provide reliable and accurate predictions for multiple timestamps ahead, which can be crucial for planning, decision-making, risk management, and resource allocation in various domains such as finance, energy management, transportation, and

environmental monitoring.

Recent advances in deep learning, particularly within Transformer- and LLM-based models [13, 14, 16, 21, 89, 90], have showcased significant progress in long-term time series forecasting. These models capitalize on the self-attention mechanism and its variants, providing a distinct advantage in capturing and modeling long-range dependencies within sequential data. While LLMs are adept at processing discrete tokens, time series data presents a unique challenge due to its continuous nature. Also, LLMs do not inherently possess the capability to interpret time series patterns, as this knowledge is not typically included in their pre-training. In addition, centralized learning models, including LLM-based methods, often require collecting and storing vast amounts of data in a central server, raising concerns about user privacy and data security. Thus, effectively leveraging LLMs for long-term time series forecasting, while ensuring accuracy and data privacy, highlights the challenges associated with centralized learning models, emphasizing the need for federated learning approaches [91]. Federated learning is an emerging machine learning paradigm that enables edge devices to utilize their local datasets to collaboratively train a global model with the help of a central server, while preserving data privacy by keeping the data on the respective devices. At each iteration, the server broadcasts the current global model to the devices for local training, and aggregates the local model updates from the devices to update the global model. Federated learning offers the potential to enhance long-term prediction accuracy [92]. However, certain challenges arise from skewed data distributions that may impact the training quality [93]. Another challenge in federated learning is systems heterogeneity, which arises from the variation in storage, computational, and communication capabilities influenced by differences in hardware specifications, and power constraints [91]. This leads to heightened concerns around stragglers and fault tolerance. Several strategies have been proposed to address this issue involving adaptive optimization techniques that cater to the varying capabilities of devices, optimizing resource usage based on each device's characteristics [91]. Additionally, employing techniques like asynchronous updates and gradient compression can reduce communication overhead and accommodate devices with limited connectivity or power [22]. These strategies collectively aim to enhance the efficiency and reliability of federated learning systems in heterogeneous environments. However, these techniques do not significantly enhance model accuracy in scenarios involving optimization and transfer learning, particularly in contexts where data heterogeneity arises from collective settings [94].

In this chapter, we introduce a federated learning framework, dubbed FedTime, for long-term time series forecasting, while preserving data privacy. Local data are used for training individual predictive LLM models on edge devices, allowing for rapid, localized predictions while reducing the load on cloud or central servers. The cloud server plays a dual role in initializing the system

and aggregating local LLMs for global updates, a key aspect of federated learning. To optimize the use of limited computational resources on edge devices, we employ parameter-efficient tuning techniques [95, 96], which are crucial in minimizing computational and communication overhead, facilitating local training. Our learning framework encompasses training procedures on local edge devices (e.g., EV charging stations) and subsequent aggregation processes on the central server side by leveraging the pre-trained Large Language Model Meta AI (LLaMA-2) [35]. This LLaMA model is a group of foundational pre-trained LLMs with various sizes (7B, 13B, 33B, and 65B parameters), offering a range of capabilities suited to different needs and computational resources. The key contributions of this chapter can be summarized as follows: (i) We introduce a federated large language framework, called FedTime, allowing for collaborative model training across edge devices while preserving data privacy; (ii) we design a model architecture using a two-phase fine-tuning strategy, leveraging quantized and low-ranking adaptation, and aligning the model with time series data via direct preference optimization; and (iii) we demonstrate through extensive experiments the superior performance of FedTime over strong baselines across various forecasting datasets, especially for long-range forecasts. Moreover, we empirically show that FedTime exhibits reduced communication overhead.

The rest of this chapter is structured as follows. In Section 3.2, we review important relevant work. In Section 3.3, we present the problem formulation and provide an approach overview, followed by introducing a federated large language model for long-term time series forecasting, with an emphasis on EV charging energy demand prediction. The empirical validation of our approach, in comparison with established baselines, is presented in Section 3.4. Finally, we conclude with a discussion in Section 3.5 .

3.2 Related Work

A sizable body of research has been developed to design Transformer-based methods for long-term time series forecasting [13–16, 90]. Pyraformer [13] presents a pyramidal attention module, leveraging inter-scale and intra-scale connections, which enhances model capability to capture temporal patterns. Informer [14] design a ProbSparse self-attention mechanism to improve model prediction capacity. Autoformer [15] adapts decomposition blocks and auto-correlation mechanisms to help separate the long-term trend information from predicted hidden variables. FEDformer [16] employs fast Fourier transform to decompose sequences, enhancing the extraction of long-term information from the data. PatchTST [90] segments time series data into subseries-level patches, which serve as input tokens to a Transformer-based architecture. It also employs channel-independence,

where each channel contains a single univariate time series that shares the same embedding and Transformer weights across all series, allowing efficient processing of multiple time series. More recently, various pre-trained LLMs, such as the generative pre-trained Transformer (GPT) [34] and LLaMA [35] models rooted in the Transformer architecture, have emerged as powerful methods in generating high-quality outputs in tasks spanning natural language processing [97–101] and time series forecasting [21], leveraging their ability to discern and comprehend intricate dependencies within data. Chang *et al.* introduce an LLM-based method, called LLM4TS, by adapting the GPT model as a foundational backbone to time series data and then fine-tuning it for time series forecasting tasks. Although our work also involves a two-stage fine-tuning strategy, it differs from LLM4TS in that our FedTime method not only focuses on efficient fine-tuning, but is also tailored for federated learning setups, aiming at long-term time series forecasting. The key strength of FedTime, compared to Transformer- and LLM-based approaches, lies in its ability to allow edge devices to collaborate in model training while keeping data decentralized. This preserves data privacy, a crucial aspect, especially in sensitive domains like healthcare. In addition, FedTime leverages parameter-efficient fine-tuning techniques, enabling it to update only a small number of local model parameters. This efficiency in training and model updates is crucial for real-world applications, especially in resource-constrained edge devices such as EV charging stations.

3.3 Method

In this section, we formulate the time series forecasting problem and provide an approach overview. Then, we present the main building blocks of the proposed federated learning framework.

3.3.1 Problem Statement and Approach Overview

Problem Statement. Time series forecasting refers to the process of predicting future values over a period of time using historical data. Let $\mathbf{X}_{1:L} = (\mathbf{x}_1, \dots, \mathbf{x}_L)^\top \in \mathbb{R}^{L \times M}$ be a history sequence of L multivariate time series, where for any time step t , each row $\mathbf{x}_t = (x_{t1}, \dots, x_{tM}) \in \mathbb{R}^{1 \times M}$ is a multivariate vector consisting of M variables or channels.

Given a history sequence $\mathbf{X}_{1:L}$ with look-back window L , the goal of multivariate time series forecasting is to predict a sequence $\mathbf{X}_{L+1:L+T} = (\mathbf{x}_{L+1}, \dots, \mathbf{x}_{L+T})^\top \in \mathbb{R}^{T \times M}$ for the future T timesteps. To this end, we introduce a federated pre-trained large language model by leveraging channel independence and patching [90], as well as supervised and downstream fine-tuning strategies [21].

Approach Overview. The proposed federated learning framework comprises two main components: local training operations conducted on the client side and aggregation operations carried out on the server side. These components collaborate to facilitate efficient training. Prior to the learning process, we apply K-means clustering as a pre-processing step to group similar edge devices or clients (i.e., based on cluster size and performance). Clustering not only helps reduce the impact of biased prediction [102], but also ensures that sensitive data remains localized within clusters, reducing the risk of data exposure during model training. Then, we employ a pre-trained LLaMA model [35] as the foundational backbone to leverage the inherent capabilities of LLMs in capturing data patterns, thereby facilitating its application to time series forecasting within our federated learning framework. To improve the global model convergence rate, we use federated averaging with adaptive learning rates using Adam optimizer [103], which dynamically adjusts the learning rates for each local device based on the model updates. For each cluster, a global model is produced by aggregating the model updates from the cluster members. We also employ parameter-efficient fine-tuning techniques [96], which offer an efficient way to adapt pre-trained LLMs to diverse downstream tasks by focusing on updating only a small number of model parameters, and hence transmit updates faster, leading to quicker convergence and reduced communication overhead. While our proposed approach builds upon these existing techniques, its technical novelties lie in how these techniques are integrated and adapted to address specific challenges in long-term time series forecasting. In addition to proposing a novel two-phase fine-tuning strategy that leverages quantized and low-ranking adaptation techniques, along with direct preference optimization, to align the model with time series data effectively, our federated learning framework streamlines resource usage by minimizing computational and communication overhead, thereby improving the scalability and efficiency of the training process.

3.3.2 Proposed Learning Framework

We present federated learning framework, designed to overcome the classical challenges within a centralized training environment, such as network congestion, low bandwidth, latency, and privacy. The proposed federated architecture is built on the LLaMA structure, enabling parallel feature learning across various time series forecasting benchmarks. Figure 3.1 depicts the main building blocks of our learning framework, including time series data normalization, patching and positional encodings, LLM encoder and robust fine-tuning strategies. Our two-phase approach consists of supervised fine-tuning, followed by forecasting fine-tuning.

Channel Independence. Channel independence refers to a strategy where the features or variables of a multivariate time series are treated separately [90], emphasizing independence rather

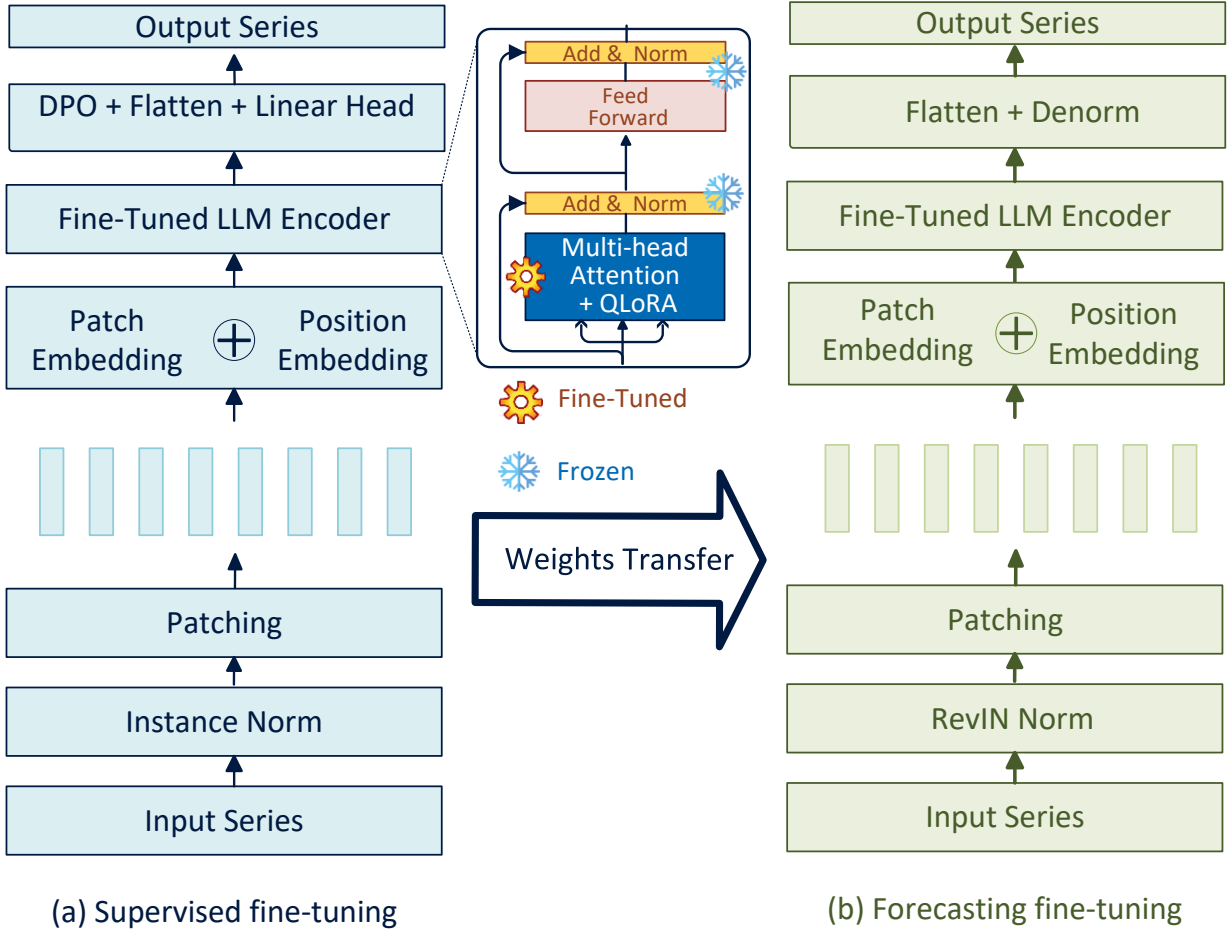


Figure 3.1: **FedTime Architecture.** (a) Supervised fine-tuning with Direct Preference Optimization (DPO) integration for backbone model alignment. (b) Forecasting fine-tuning using Reversible Instance Normalization (RevIN).

than mixing or amalgamating information from different channels or variables. This approach aims to retain the distinct characteristics of each variable without merging them together. Specifically, the input time series $\mathbf{X}_{1:L} = (\mathbf{x}_1, \dots, \mathbf{x}_L)^\top$ is split into M univariate series $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_L^{(i)})^\top \in \mathbb{R}^L$, where $\mathbf{x}^{(i)}$ is the i th column of $\mathbf{X}_{1:L}$. Each of these univariate series is fed into the model backbone. Our proposed federated large language model takes $\mathbf{x}^{(i)}$ as input and returns a T -dimensional vector of predictions $\hat{\mathbf{x}}^{(i)} = (\hat{x}_{L+1}^{(i)}, \dots, \hat{x}_{L+T}^{(i)})^\top$.

Normalization. In the supervised fine-tuning phase, we employ instance normalization before patching to normalize each data instance $\mathbf{x}^{(i)}$ with zero mean and unit standard deviation, which are then added back to the output prediction. In the subsequent forecasting fine-tuning phase, each input series is normalized using the reversible instance normalization (RevIN) technique [104], which addresses challenges related to shifts in data distributions over time. RevIN consists of two main steps: normalization and denormalization. In the first step, the input undergoes normalization

to standardize its distribution in terms of mean and variance. After the model generates output sequences, RevIN reverses the normalization process by denormalizing these outputs.

Patching. We adopt the patching strategy from [90], which involves dividing the sequential data into smaller patches. Specifically, each input series $\mathbf{x}^{(i)} \in \mathbb{R}^L$ is partitioned into patches, resulting in a sequence of patches $\mathbf{X}_p^{(i)} \in \mathbb{R}^{N \times P}$, where P is the patch length and N is the resulting number of patches, which also serves as the input sequence length for the Transformer encoder. The benefits of patching include better preservation of local semantic information in patches, computational and memory efficiency, and an expanded historical view.

Patch and Position Embeddings. Patch embeddings are obtained by mapping each patch in the sequence of patches to D dimensions with a trainable linear projection $\mathbf{W}_p \in \mathbb{R}^{P \times D}$. Position embeddings are added to patch embeddings via a learnable position encoding matrix $\mathbf{W}_{\text{pos}} \in \mathbb{R}^{N \times D}$ to retain positional information of the patches. Summing patch and position embeddings yields an embedding $\mathbf{X}_d^{(i)} \in \mathbb{R}^{N \times D}$ given by

$$\mathbf{X}_d^{(i)} = \mathbf{X}_p^{(i)} \mathbf{W}_p + \mathbf{W}_{\text{pos}}, \quad (3.1)$$

which serves as input to the multi-head self-attention mechanism of the Transformer encoder.

LLM Encoder. We employ the lightweight LLaMA2-7B model as our LLM encoder. This LLaMA model is fundamentally built on the Transformer architecture, distinguished by its 7 billion parameters, and benefits from a comprehensive training on an expansive corpus containing 2 trillion tokens. At their core, Transformers rely on self-attention, which operates on the matrix $\mathbf{X}_d^{(i)}$ of tokens (i.e., patch embeddings with added positional encodings) by first linearly projecting it into a query matrix $\mathbf{Q}^{(i)}$, a key matrix $\mathbf{K}^{(i)}$ and a value matrix $\mathbf{V}^{(i)}$ as follows:

$$\mathbf{Q}^{(i)} = \mathbf{X}_d^{(i)} \mathbf{W}_q, \quad \mathbf{K}^{(i)} = \mathbf{X}_d^{(i)} \mathbf{W}_k, \quad \mathbf{V}^{(i)} = \mathbf{X}_d^{(i)} \mathbf{W}_v, \quad (3.2)$$

where $\mathbf{W}_q \in \mathbb{R}^{D \times d_k}$, $\mathbf{W}_k \in \mathbb{R}^{D \times d_k}$ and $\mathbf{W}_v \in \mathbb{R}^{D \times d_k}$ are learnable weight matrices, and d_k is the projection dimension. Then, the self-attention (SA) output is a $N \times d_k$ matrix defined as the weighted sum of the value vectors for each token

$$\text{SA}(\mathbf{X}_d^{(i)}) = \text{softmax} \left(\frac{\mathbf{Q}^{(i)} \mathbf{K}^{(i)\top}}{\sqrt{d_k}} \right) \mathbf{V}^{(i)}, \quad (3.3)$$

where the weights are attention scores computed as the dot product between the query and key vectors, scaled by the square root of the projection dimension, and followed by softmax applied row-wise.

In the multi-head attention mechanism (MSA), multiple self-attention operations are performed in parallel, each with its own set of learnable weight matrices. For simplicity, we assume $d_k =$

D/h , where h is the number of attention heads. The outputs of h heads are concatenated and linearly transformed using a learnable weight matrix $\mathbf{W}_o \in \mathbb{R}^{D \times D}$ to obtain the MSA output:

$$\text{MSA}(\mathbf{X}_d^{(i)}) = \text{Concat}(\mathbf{Y}_1^{(i)}, \dots, \mathbf{Y}_h^{(i)})\mathbf{W}_o \in \mathbb{R}^{N \times D}, \quad (3.4)$$

where $\mathbf{Y}_j^{(i)} = \text{SA}_j(\mathbf{X}_d^{(i)}) \in \mathbb{R}^{N \times \frac{D}{h}}$ is the output of the j -th attention head.

As shown in Figure 3.1, the Transformer encoder architecture is comprised of a series of alternating layers, each containing an MSA mechanism and a feedforward neural network (FFN) block with SwiGLU activation function [105]. Before each block, layer normalization is applied, and after each block, residual connections are employed. Hence, the resulting output of the Transformer encoder is $\mathbf{Z}_d^{(i)} \in \mathbb{R}^{N \times D}$, which subsequently undergoes a flatten layer with linear head to produce the model predictions $\hat{\mathbf{x}}^{(i)} = (\hat{x}_{L+1}^{(i)}, \dots, \hat{x}_{L+T}^{(i)})^\top$.

Model Fine-Tuning. Parameter efficient fine-tuning (PEFT) methods offer an efficient way to adapt pre-trained LLMs to diverse downstream tasks without the need to fine-tune the entire set of model parameters. To fine-tune our pre-trained LLM, we employ quantized and low-rank adaptation (QLoRA) [96], a PEFT technique that aims to minimize the number of parameters updated during fine-tuning while preserving or improving performance on downstream tasks. QLoRA builds upon LoRA [95], which freezes the weight matrices of the linear projection layers within the self-attention mechanism of the Transformer encoder and represents each of their update matrices as a product of two trainable low-rank matrices. This low-rank matrix decomposition significantly reduces the number of trainable parameters, making the model more efficient for downstream tasks. In addition to low-rank decomposition, QLoRA also incorporates quantization techniques that reduce the precision of the weight values, thereby further decreasing the model’s memory footprint and computational requirements without substantially sacrificing performance. With QLoRA, only 1.2% of the model’s parameters are considered trainable, whereas using LoRA increases this percentage to 1.5% [21].

Model Alignment. Supervised fine-tuning is used as a fundamental precursor for subsequent forecasting fine-tuning of the LLaMA model. To enhance the alignment of our model with time series data after the supervised fine-tuning phase, we integrate direct preference optimization (DPO) [106] into our fine-tuning process. DPO, which involves the analysis of 10K comparison data pairs from the UltraFeedback dataset [107], is applied post-supervised fine-tuning to enhance model performance for specific data points. It optimizes its objective function, producing the optimal policy to an implicit reward function that is the best fit to the preference data. We use DPO to capture any change of variables, ensuring a more effective adaptation of the LLaMA model to the intricacies of time series forecasting. Finally, we transfer the updated weights of the backbone model to the forecasting fine-tuning phase.

Loss Function. The parameters of our FedTime model are learned by minimizing the training objective, which is the mean squared error (MSE) defined as

$$\mathcal{L} = \frac{1}{MT} \sum_{i=1}^M \sum_{\tau=L+1}^{L+T} \|\mathbf{x}_{\tau}^{(i)} - \hat{\mathbf{x}}_{\tau}^{(i)}\|^2, \quad (3.5)$$

where $\mathbf{x}_{\tau}^{(i)}$ and $\hat{\mathbf{x}}_{\tau}^{(i)}$ are the ground-truth and prediction, respectively, $\tau \in \{L+1, \dots, L+T\}$, L is the look-back window, T is the prediction length/horizon (i.e., future timesteps), and M is the number of time series variables (i.e., number of features or channels).

Algorithm. The main algorithmic steps of our federated learning framework are summarized in Algorithm 2. Prior to the learning process, K-means clustering is used as a pre-processing step to group edge devices into clusters. In practice, K-means clustering may occur in scenarios where a large number of edge devices are deployed in geographically dispersed locations, such as smart city infrastructures, industrial IoT deployments, or distributed sensor networks.

The FedTime algorithm initializes a global model, distributes it to selected edge devices for local training, aggregates the updated models from the participating devices at the central server, and iterates this process for multiple rounds, allowing the global model to gradually improve by aggregating insights from diverse edge devices while preserving data privacy. By training cluster-specific models, FedTime can capture patterns specific to different subsets of edge devices. Moreover, instead of transmitting updates from each individual edge device to the central server, FedTime aggregates updates from devices within the same cluster. This helps reduce the overall communication overhead by consolidating updates from devices with similar data distributions.

3.4 Experiments

In this section, we present experimental evaluation of the proposed federated learning approach by comparing it with recent state-of-the-art methods.

3.4.1 Experimental Setup

Datasets. We conduct a comprehensive evaluation of our FedTime model on various benchmark datasets: Weather¹, Traffic², Electricity³, and four ETT datasets (ETTh1, ETTh2, ETTm1, ETTm2) [15]. Dataset statistics are summarized in Table 3.1: (1) **Weather** is recorded at 10-minute intervals for the entire year of 2020, and contains 21 meteorological indicators, such as air

¹<https://www.bgc-jena.mpg.de/wetter>

²<http://pems.dot.ca.gov>

³<https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014>

Algorithm 2 FedTime: Federated LLM Algorithm

- 1: **Input:** Global model parameters θ ; number of edge devices S ; number of clusters K ; local datasets $\mathcal{D}_1, \dots, \mathcal{D}_S$
- 2: **Output:** Vector of aggregated global cluster-specific models θ_K
- 3: Apply K-means algorithm to split devices into K clusters $\mathcal{C} = \{c_1, c_2, \dots, c_K\}$ based on similar attributes.
- 4: Initialize global cluster model parameters θ_K
- 5: **for** each cluster c in $\mathcal{C}$ **do**
- 6: **for** each round $r = 1, 2, \dots$ **do**
- 7: **while** model θ_K has not converged yet **do**
- 8: **for** each edge device $s \in \{1, \dots, S\}$ **in parallel do**
- 9: $\theta_K \leftarrow \text{UpdateDevice}(\mathcal{D}_s, \theta_c)$
- 10: **end for**
- 11: **end while**
- 12: Server aggregates final global models from cluster-specific models using weighted averaging

$$\theta_c = \frac{\sum_s w_{s,c} \theta_K}{\sum_s w_{s,c}}$$

where $w_{s,c}$ is the weight of device s in cluster c .

- 13: **end for**
- 14: Send global cluster-specific model θ_c to local device s
- 15: **end for**
- 16: **UpdateDevice**($\mathcal{D}_s, \theta$)
- 17: **for** each cluster c in $\mathcal{C}$ **do**
- 18: **for** each local device s in cluster c **do**
- 19: Receive global model parameters θ from the server
- 20: Train local model using local data $\mathcal{D}_s$
- 21: Update local model parameters

$$\theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta; \mathcal{D}_s)$$

where η is the learning rate and $\mathcal{L}(\theta; \mathcal{D}_s)$ is the MSE local loss.

- 22: Send local model parameters θ to the server
 - 23: **end for**
 - 24: **end for**
 - 25: **Return:** θ_K
-

temperature and humidity. (2) **Traffic** is a collection of hourly data from California Department of Transportation, and describes the road occupancy rates measured by different sensors on San Francisco Bay area freeways. (3) **Electricity** contains the hourly electricity consumption of 321 customers from 2012 to 2014. (4) **ETT** contains data collected from electricity transformers, including load and oil temperature that are recorded every 1 hour for ETTh datasets and every 15

minutes for ETTm datasets between July 2016 and July 2018.

Table 3.1: Summary statistics of benchmark datasets.

Dataset	Features	Timesteps	Granularity
Weather	21	52,696	10 min
Traffic	862	17,544	1 hour
Electricity	321	26,304	1 hour
ETTh1	7	17,420	1 hour
ETTh2	7	17,420	1 hour
ETTm1	7	69,680	15 min
ETTm2	7	69,680	15 min

Baseline Methods. We evaluate the performance of our FedTime model against recent state-of-the-art methods, including Pyraformer [13], Informer [14], Autoformer [89], FEDformer [16], DLinear [108], N-HiTS [109], PatchTST [90], TiDE [110], and LLM4TS [21].

Evaluation Metrics. We use the mean squared error (MSE) and mean absolute error (MAE) as metrics for evaluating both long-term time series forecasting. MSE measures the average squared difference between the predicted and actual values. A lower MSE indicates that the predictions are closer to the actual values on average. On the other hand, MAE measures the average absolute difference between the predicted and actual values. Similar to MSE, a lower MAE indicates better predictive accuracy, with predictions closer to the actual values. We report the performance at various prediction horizons and look-back windows.

Implementation Details. In our federated learning setup, we assume the presence of 555 eligible edge devices for training. All experiments are carried out on a GPU server with an AMD EPYC 7502 processor, an NVIDIA RTX A6000 48GB GDDR6 and 512GB memory. The algorithms are implemented in PyTorch and PySyft to simulate a federated learning environment. For hyper-parameters optimization, we apply a grid search on a batch size of range {128, 256, 512, 1024}, a learning rate of range {0.01, 0.001, 0.0001}, and a local training round of range {40, 80, 200}. Using hyper-parameter tuning, the batch size and learning rate are set to 512 and 0.001, respectively. In all experiments, we split the datasets into 80% training and 20% testing. To update QLoRA parameters, we employ FedAdam [111], an adaptive optimization method that helps improve the convergence of federated learning.

3.4.2 Results and Analysis

Long-Term Time Series Forecasting. We evaluate the forecasting performance of our FedTime model against strong baseline methods on seven forecasting benchmark datasets, and the results are reported in Table 3.2. For Transformer-based baselines, the values of the MSE and MAE error metrics are taken from [90]. The best results are shown in bold, and the second best results are underlined. As can be seen, FedTime consistently outperforms the baselines on most datasets, particularly for long-term prediction horizons. In terms of MSE, Table 3.2 reveals that FedTime performs better than LLM4TS in 24 out of 28 evaluations spanning seven datasets and four forecast horizons, improving upon this best performing baseline by relative error reductions of 15.56% and 20% on the Traffic and Electricity datasets, respectively, for $T = 720$, while maintaining a smaller number of learnable parameters compared to LLM4TS. Similarly, our method outperforms LLM4TS in 25 out of 28 MAE evaluations, yielding relative error reductions of 10.98% and 7.36% on the ETTh1 and ETTh2 datasets, respectively, for $T = 720$. Moreover, FedTime performs better or on par with the best performing LLM4TS baseline on the Weather dataset across all the prediction horizons. It is worth pointing out that MAE is more interpretable than MSE since it directly represents the average magnitude of prediction errors. A low MAE implies that FedTime provides accurate estimates of forecasting time series, thus demonstrating the effectiveness of the our framework.

In Figure 3.2, we report the visual results in terms of MSE on six forecasting datasets for various values of the look-back window length L ranging from 24 to 720. As can be seen, FedTime exhibits consistent and superior performance, especially for long look-back windows. On the Traffic dataset, it is evident that FedTime holds a competitive edge, consistently outperforming Transformer-based models such as Informer, Autoformer and FEDformer. On the Weather dataset, FedTime stands out, especially between the look-back window lengths of 192 and 336. On the ETTh1 and ETTh2 datasets, FedTime’s performance peaks within the intermediate look-back window span of 48 to 192. Overall, FedTime yields superior performance over the baselines across the various forecasting datasets, especially for long-range look-back scenarios.

Federated Performance. In contrast to the centralized training of the LLaMA model, federated learning adopts a decentralized methodology, allowing for localized model updates and only transmitting the model weights, thus enhancing data privacy. In Figure 3.3, the training and testing learning curves show that our federated LLM model converges approximately $3\times$ faster than its centralized counterpart under analogous experimental conditions. We see that while the centralized LLaMA model faltered in convergence even post 200 epochs, the federated FedTime model adeptly minimized the loss function in merely 70 epochs. Also, the federated model’s uniform

Table 3.2: Long-term time series forecasting results for various prediction lengths $T \in \{96, 192, 336, 720\}$. The best results are in **bold**, while the second-best are underlined.

Method	Autoformer		FEDformer		DLinear		N-HiTS		PatchTST/64		TiDE		LLM4TS		FedTime		
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	
Weather	96	0.249	0.329	0.238	0.314	0.176	0.237	0.158	0.195	<u>0.149</u>	0.198	0.166	0.222	0.147	0.196	0.150	0.194
	192	0.325	0.370	0.275	0.329	0.220	0.282	0.211	0.247	0.194	0.241	0.209	0.263	<u>0.191</u>	<u>0.238</u>	0.180	0.217
	336	0.351	0.391	0.339	0.377	0.265	0.319	0.274	0.300	0.245	0.282	0.254	0.301	<u>0.241</u>	<u>0.227</u>	0.209	0.227
	720	0.415	0.426	0.389	0.409	0.323	0.362	0.401	0.413	0.314	0.334	0.313	0.340	0.313	0.329	<u>0.318</u>	<u>0.335</u>
Traffic	96	0.597	0.371	0.576	0.359	0.410	0.282	0.402	0.282	0.360	<u>0.249</u>	<u>0.336</u>	0.253	0.372	0.259	0.328	0.215
	192	0.607	0.382	0.610	0.380	0.423	0.287	0.420	0.297	0.379	<u>0.256</u>	<u>0.346</u>	0.257	0.391	0.265	0.339	0.233
	336	0.623	0.387	0.608	0.375	0.436	0.296	0.448	0.313	0.392	0.264	<u>0.355</u>	0.260	0.405	<u>0.275</u>	0.347	0.288
	720	0.639	0.395	0.621	0.375	0.466	0.315	0.539	0.353	0.432	0.286	<u>0.386</u>	<u>0.273</u>	0.437	0.292	0.369	0.239
Electricity	96	0.196	0.304	0.186	0.311	0.140	0.237	0.147	0.249	0.129	0.222	0.132	0.229	<u>0.128</u>	<u>0.223</u>	0.118	0.158
	192	0.211	0.324	0.197	0.311	0.153	0.249	0.167	0.269	0.147	<u>0.240</u>	<u>0.147</u>	0.243	<u>0.146</u>	<u>0.242</u>	0.133	0.315
	336	0.214	0.327	0.213	0.314	0.169	0.267	0.186	0.290	0.163	0.259	<u>0.161</u>	0.261	0.163	<u>0.258</u>	0.148	0.255
	720	0.236	0.342	0.233	0.344	0.203	0.301	0.243	0.340	0.197	0.290	<u>0.196</u>	0.294	0.220	<u>0.292</u>	0.176	0.288
ETTh1	96	0.435	0.446	0.376	0.415	0.375	0.399	0.378	0.436	0.379	0.401	<u>0.375</u>	<u>0.398</u>	0.371	0.394	0.392	0.402
	192	0.456	0.457	0.423	0.446	0.405	0.420	0.427	0.436	0.413	0.429	0.412	0.422	0.403	<u>0.412</u>	<u>0.404</u>	0.360
	336	0.486	0.487	0.444	0.462	0.439	0.443	0.458	0.484	0.435	0.436	0.435	0.433	<u>0.428</u>	<u>0.422</u>	0.425	0.421
	720	0.515	0.517	0.469	0.492	0.472	0.490	0.472	0.551	<u>0.446</u>	0.464	0.454	0.465	<u>0.422</u>	<u>0.444</u>	0.414	0.436
ETTh2	96	0.332	0.368	0.332	0.374	0.289	0.353	0.274	0.345	0.274	0.337	0.270	0.336	<u>0.269</u>	<u>0.332</u>	0.249	0.303
	192	0.426	0.434	0.407	0.446	0.383	0.418	0.353	0.401	0.338	<u>0.376</u>	0.332	0.380	<u>0.328</u>	0.377	0.318	0.343
	336	0.477	0.479	0.400	0.447	0.448	0.465	0.382	0.425	0.363	0.397	0.360	0.407	<u>0.353</u>	<u>0.396</u>	0.344	0.368
	720	0.453	0.490	0.412	0.469	0.605	0.551	0.625	0.557	0.393	0.430	0.419	0.451	<u>0.383</u>	<u>0.425</u>	0.376	0.428
ETTm1	96	0.510	0.492	0.326	0.390	0.299	0.343	0.302	0.350	0.293	0.346	0.306	0.349	<u>0.285</u>	<u>0.343</u>	0.283	0.339
	192	0.514	0.495	0.365	0.415	0.335	0.365	0.347	0.383	0.333	0.370	0.335	<u>0.366</u>	<u>0.324</u>	<u>0.366</u>	0.301	0.338
	336	0.510	0.492	0.392	0.425	0.369	0.386	0.369	0.402	0.369	0.392	0.364	<u>0.384</u>	<u>0.353</u>	0.385	0.319	0.365
	720	0.527	0.493	0.446	0.458	0.425	0.421	0.431	0.441	0.416	0.420	0.413	<u>0.413</u>	<u>0.408</u>	0.419	0.328	0.373
ETTm2	96	0.205	0.293	0.180	0.271	0.167	0.260	0.176	0.255	0.166	0.256	<u>0.161</u>	<u>0.251</u>	0.165	0.254	0.148	0.227
	192	0.278	0.336	0.252	0.318	0.224	0.303	0.245	0.305	0.223	0.296	<u>0.215</u>	<u>0.289</u>	0.225	0.292	0.197	0.261
	336	0.343	0.379	0.324	0.364	0.281	0.342	0.295	0.346	0.274	0.329	<u>0.267</u>	<u>0.326</u>	0.268	0.326	0.219	0.279
	720	0.414	0.419	0.410	0.420	0.397	0.421	0.401	0.413	0.362	0.385	<u>0.352</u>	0.383	0.356	<u>0.380</u>	0.305	0.352

training trajectory, which is largely attributed to the incorporation of a cohesive cluster of edge devices, diverges sharply from the inconsistency seen in the centralized counterpart.

For the sake of federated comparison, we implemented a federated version of PatchTST [90], which we refer to as Fed-PatchTST. We evaluate the performance of FedTime against federated stacked LSTM (FSLSTM) [1] and Fed-PatchTST, and the results are reported in Table 3.3, which shows that FedTime consistently outperforms these federated baselines across all the forecasting benchmark datasets. This better performance is particularly notable for longer prediction horizons, highlighting the robustness and adaptability of FedTime to long-term time series forecasting. For instance, the percentage gap and percentage difference between FedTime and Fed-PatchTST, in

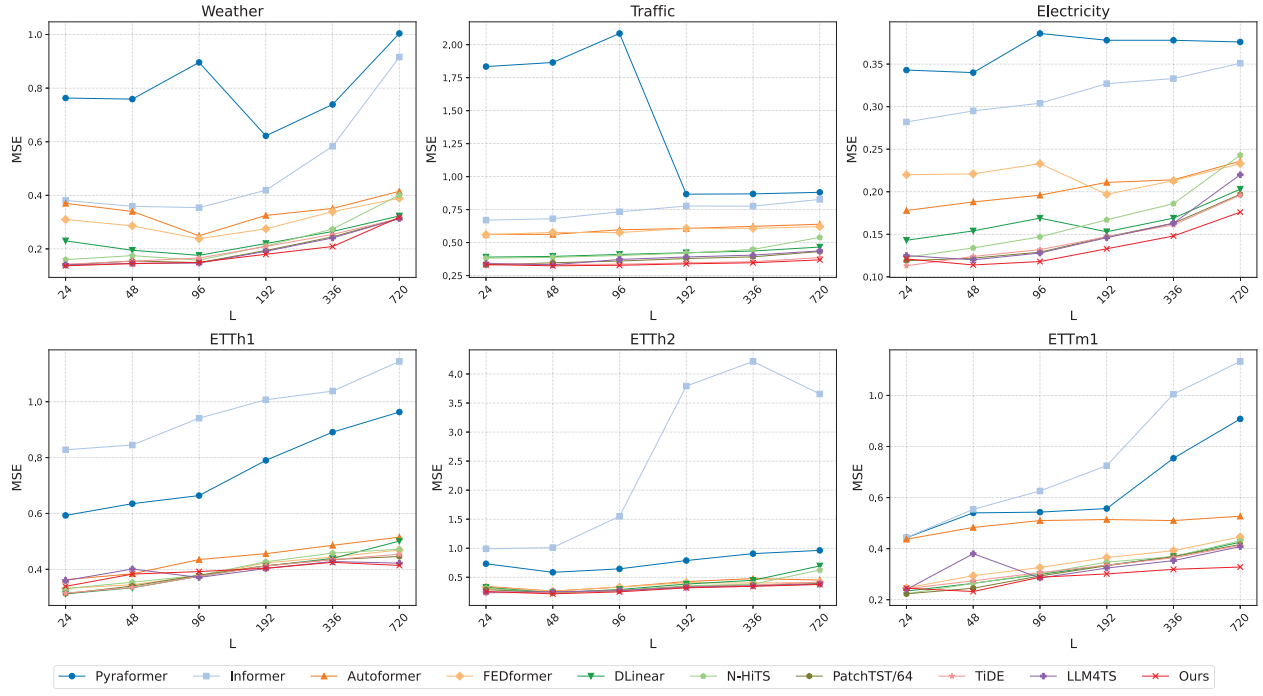


Figure 3.2: Long-term forecasting performance with varying lookback window lengths $L \in \{24, 48, 96, 192, 336, 720\}$. The prediction horizon is fixed at $T = 720$.

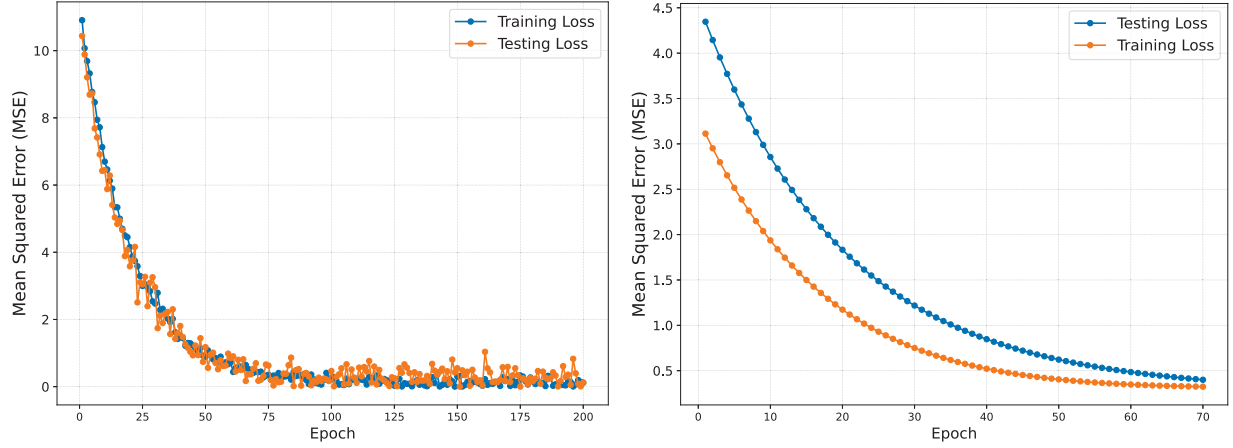


Figure 3.3: Training and testing learning curves of the centralized LLaMA model (left) and Fed-Time (right).

terms of MAE, on the Electricity dataset are 39.24% and 32.8%, respectively, indicating a significant improvement in prediction accuracy with our model.

Table 3.3: Performance comparison of FedTime and federated models across benchmark datasets with prediction length $T = 720$.

Method	Fed-PatchTST		FSLSTM		FedTime	
Metric	MSE	MAE	MSE	MAE	MSE	MAE
Weather	0.363	0.495	0.451	0.421	0.318	0.335
Traffic	0.388	0.272	0.450	0.386	0.369	0.239
Electricity	0.231	0.401	0.310	0.466	0.176	0.288
ETTh1	0.456	0.520	0.522	0.621	0.414	0.436
ETTh2	0.402	0.521	0.452	0.533	0.376	0.428
ETTh1	0.340	0.421	0.352	0.483	0.328	0.373
ETTh2	0.335	0.384	0.401	0.495	0.305	0.352

3.4.3 Communication Overhead

Evaluating the communication overhead is crucial in assessing the feasibility of a model’s deployment in federated environments. We use the real-world ACN dataset [112], which is an adaptive charging network dataset containing information about an electric vehicle (EV) single charging session at two distinct sites: California Institute of Technology (Caltech 540 stations) and Jet Propulsion Laboratory (JPL 400 stations). An exploratory data analysis of the dataset shows that the data distribution has an increasing trend at both Caltech and JPL sites. These upward trends in the distribution of energy demand at both sites underscores the need for enhanced capacity to accommodate these surges and mitigate escalated demand charges. The ACN dataset consists of 1.5 million charging sessions with a window time of 19 months. Both Caltech and JPL sites show a repetitive usage pattern with much higher utilization during weekdays than on weekends, as shown in Figure 3.4, which depicts the distribution of the energy delivered.

One of the key metrics of communication overhead is the data volume transferred, measured in Megabytes. As shown in Figure 3.5, our FedTime model yields reduced communication overhead compared to the baseline methods. This reduction is even more pronounced when considering metrics like message count and communication time. A key factor contributing to FedTime’s efficient performance is the strategic engagement of edge devices in the learning process. This decentralized approach, leveraging the inherent distributed nature of IoT-enabled devices, facilitates a system where updating model weights incurs a significantly lower overhead than transferring entire datasets. Such an optimized model structure and design strategy make FedTime notably efficient in comparison with centralized architectures such as Informer, Pyraformer, Autoformer, and LLM4TS. Moreover, Figure 3.5 alludes to the efficiency of models with optimized weight structures using PEFT techniques.

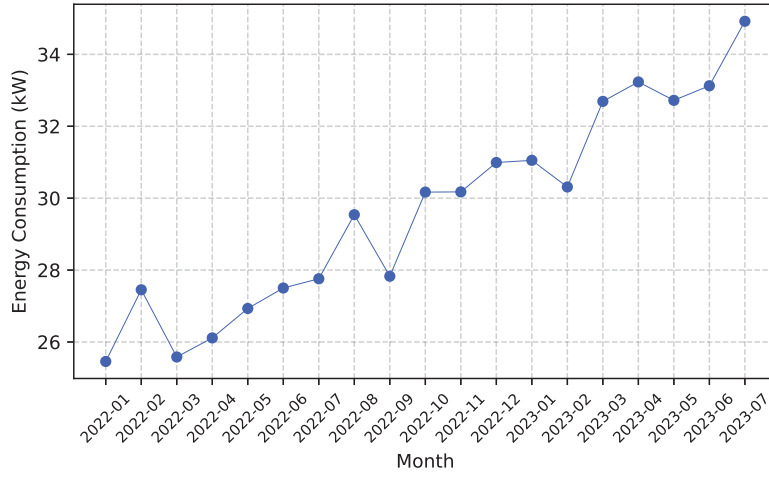


Figure 3.4: ACN energy delivery distribution based on disconnection time.

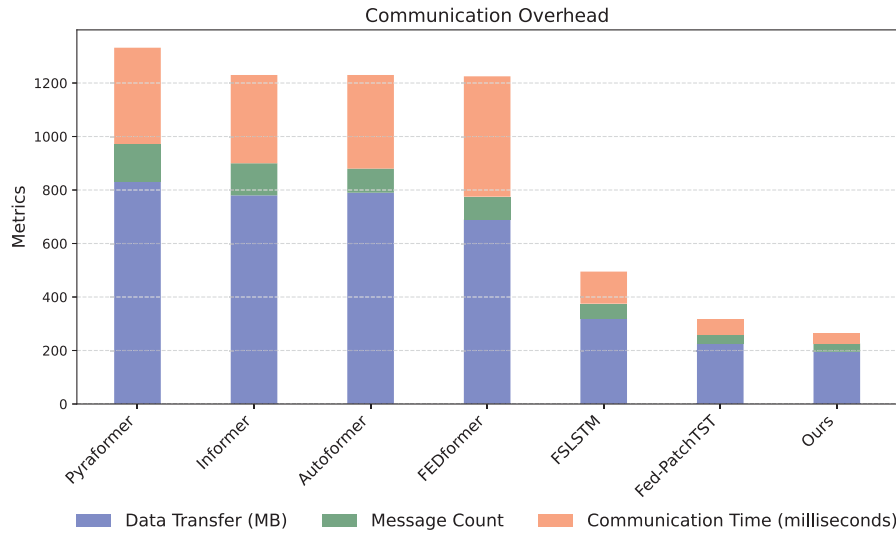


Figure 3.5: Communication overhead comparison between FedTime and baseline methods.

3.4.4 Ablation Study

In Figure 3.6, we evaluate the performance of various FedTime variants on the ACN dataset for Caltech site. In the following, we introduce these variants and analyze their respective performances:

- **FedTime without Clustering:** This model aligns relatively well with the actual energy consumption in general up to the 20-hour mark. Yet, between the 20- and 40-hour intervals, the model tends to underestimate consumption. Between 60 to 80 hours, the model’s predictions come closer to the actual data, but discrepancies are noticeable toward the end of the 100-hour period. The curve’s fluctuations are likely due to the mix of devices from different-sized clusters participating in this specific training round.

- **FedTime without PEFT:** The performance without parameter efficient fine-tuning generally tracks the consumption trend. Yet, certain intervals display noticeable over-estimations, and there are slight variances from the actual energy consumption, especially during peak and trough periods.
- **FedTime with Clustering+PEFT:** This variant manifests a notable performance over the 100-hour duration. Distinctly evident is its aptitude in closely mirroring the authentic consumption, especially in regions characterized by pronounced fluctuations, such as the 20-40 and 60-80 hour intervals. The superior performance of this model makes it the best choice for energy consumption prediction over the scrutinized 100-hour interval at Caltech site.

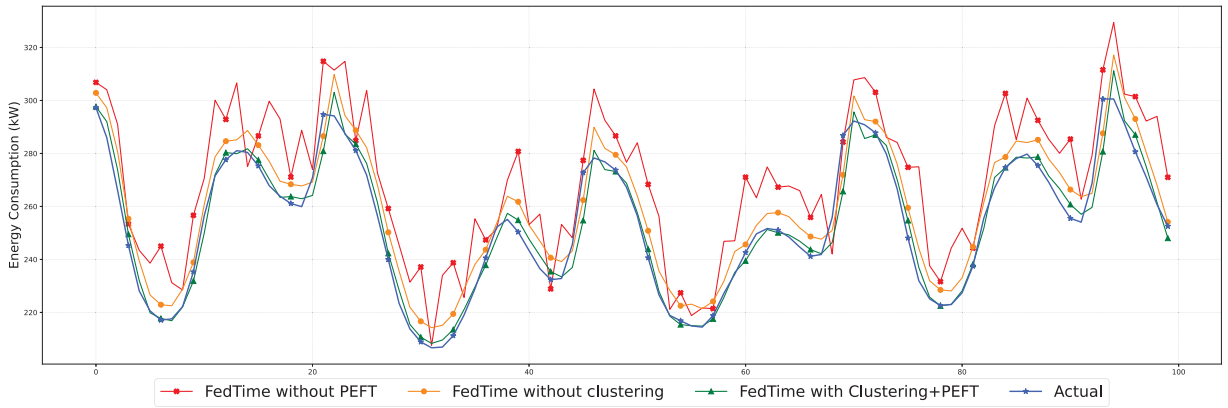


Figure 3.6: Comparison of predicted and actual energy consumption at the Caltech site during weekdays (100-hour time step) using FedTime variants. The horizontal axis represents the different time periods in hours.

We also investigate two variants of FedTime integrated with the LLaMA model: one with a parameter size of 7B, and the other with 13B. These models are subsequently denoted as FedTime-7B and FedTime-13B, respectively. It is worth mentioning that further experimentation was conducted with larger configurations, specifically 34B and 70B. However, the intricate architecture of these expanded models poses challenges in deployment at the edge devices, primarily due to limitations in memory resources and computational capacity. The comparative results of our model variants are reported in Table 3.4. Latency results offer insightful comparisons of our FedTime variants on diverse GPU setups, distinguishing between edge configurations, marked as (E), and server settings, denoted by (S). A striking observation from our analysis is the commendable performance of FedTime-7B across all GPU setups. Notably, this configuration exhibits the shortest latency, making it a method of choice for energy demand forecasting on edge devices. Furthermore, specific configurations, especially FedTime-34B and FedTime-70B, are unfeasible on certain GPU setups,

such as the RTX A6000 for the former and both RTX A6000 and ml.g5.12xlarge for the latter. This infeasibility is largely attributed to memory limitations juxtaposed with the demands of the model’s intricate architecture. As shown in Table 3.4, while the FedTime model without PEFT consistently yields higher latency values across various setups, the integration of PEFT into the model demonstrates a tangible reduction in latency. For edge devices, in particular, the FedTime-7B configuration stands out with its efficient performance, underscoring the potential and effectiveness of compact models in real-world deployment scenarios.

Table 3.4: Latency performance evaluation of various FedTime model versions across different GPU setups with and without PEFT. Latency times, measured in milliseconds (ms), showcase the efficiency of response in edge (E) and server (S) configurations. Boldface numbers indicate the best latency performance on edge devices.

Compute Instance	RTX A6000 (E)		ml.g5.12xlarge (S)		ml.g5.24xlarge (S)		ml.p4d.24xlarge (S)	
Method	w/o PEFT	FedTime	w/o PEFT	FedTime	w/o PEFT	FedTime	w/o PEFT	FedTime
FedTime-7B	41	28	34	22	26	19	19	8
FedTime-13B	65	41	48	36	32	22	24	13
FedTime-34B	x	x	81	62	76	45	69	39
FedTime-70B	x	x	x	x	135	81	85	54

3.5 Discussion

FedTime framework introduces a federated large language model tailored for long-term time series forecasting. It offers several advantages over both centralized and federated baselines, which we summarize in three main aspects: (1) improved accuracy on long-horizon prediction tasks, (2) accelerated convergence and communication efficiency, and (3) scalable privacy-preserving deployment across heterogeneous edge devices.

- *Enhanced Forecasting Accuracy.* FedTime consistently outperforms strong Transformer-based and LLM-based baselines (Autoformer, FEDformer, PatchTST, LLM4TS) across multiple benchmark datasets, including Weather, Traffic, Electricity, and ETT variants. For example, FedTime achieves up to 20% relative error reduction in long-range forecasts on Traffic and Electricity datasets at horizon $T = 720$. Its two-phase fine-tuning strategy—combining quantized low-rank adaptation (QLoRA) with direct preference optimization—effectively aligns pre-trained LLaMA models with time series dynamics. This allows the model to capture both global temporal patterns and local variations with high fidelity.

- *Efficient Convergence and Communication.* Compared to centralized training of LLaMA, the federated setup reduces the number of training epochs by nearly threefold while improving stability of the learning curves:contentReference[oaicite:2]index=2. Empirical evaluation shows that FedTime converges within ~ 70 epochs, whereas centralized baselines remain unstable even after 200 epochs. Communication cost is substantially lowered: only parameter updates are transmitted rather than full datasets, yielding reductions in data volume, message count, and latency. Ablation studies confirm that combining clustering with parameter-efficient fine-tuning offers the best trade-off between accuracy and overhead.
- *Scalability and Practical Deployment.* By leveraging device clustering and parameter-efficient updates, FedTime scales effectively across hundreds of heterogeneous clients (e.g., EV charging stations, sensors in smart grids). This design enables privacy-preserving, decentralized training while maintaining global performance improvements. Notably, the framework adapts well to long look-back windows, which are often critical in real-world operations such as energy demand prediction and traffic monitoring.

Alongside these benefits, the framework still faces certain challenges, which we outlined below:

- *Model complexity.* Even with parameter-efficient tuning, large-scale LLaMA backbones (7B+) require non-trivial memory and compute resources, which may not be feasible on all edge devices.
- *Non-IID and heterogeneous data.* Variations in device distributions and capabilities can bias updates or slow convergence, especially under extreme heterogeneity. While clustering mitigates this, it introduces additional system complexity.
- *Interpretability.* The use of LLM backbones enhances accuracy but limits transparency. Explaining forecast deviations to operators remains challenging without supplementary attribution mechanisms.
- *Overhead of alignment.* Direct preference optimization and reversible normalization improve alignment but add extra computation and memory overhead, potentially hindering lightweight deployments.
- *Privacy guarantees.* While raw data remain on-device, stronger privacy mechanisms such as differential privacy or secure aggregation were not the primary focus. These may affect accuracy and efficiency if integrated.

Blockchain-Enhanced Network Decentralization with LLMs for Time Series Forecasting

Time series forecasting plays a pivotal role in applications requiring predictive insights from sequential data, such as energy demand prediction in distributed systems. However, centralized forecasting models suffer from single points of failure and require aggregating sensitive data, while federated learning, though privacy-preserving, still relies on central servers that introduce computational bottlenecks. To address these limitations, we introduce BLEND, a decentralized framework for time series forecasting that makes two core contributions. First, we adapt large language models for time series forecasting using data-efficient instruction tuning and direct preference optimization. Second, we propose a fully decentralized architecture employing an agentic workflow where specialized agents coordinate via smart contracts governed by a proof-of-forecast consensus protocol, eliminating central aggregators while maintaining privacy guarantees. Experimental results on seven benchmark datasets demonstrates superior prediction performance with up to 63% reduction in communication overhead compared to federated baselines, showing that integrating task-aligned models with agent-based, performance-driven consensus mechanisms significantly enhances long-term time series forecasting.

4.1 Introduction

Long-term time series forecasting aims to predict future values over extended horizons using historical data. This task plays a pivotal role in strategic planning and decision-making across numer-

ous domains such as energy management, transportation, healthcare, and finance [113–116]. Accurate forecasts facilitate efficient resource allocation, enhance operational resilience, and enable proactive strategies. For instance, in energy systems, precise energy demand predictions ensure grid stability, while in transportation, traffic flow forecasts improve route planning and mitigate congestion. However, centralized forecasting methods [16–20, 108, 117, 118] encounter significant limitations in distributed settings, where data is generated across numerous devices and often contains sensitive information. These methods, which rely on aggregating sensitive data from multiple sources into a single processing hub, raise privacy concerns [22, 42], incur high communication overhead [119], and are vulnerable to single-point failures, particularly in large-scale, heterogeneous systems such as Internet of Vehicle (IoV) networks [91].

Federated learning (FL) addresses the limitations of data centralization by enabling local model training on edge devices while sharing only model updates [22]. Despite its privacy advantages, the widely used client-server architecture introduces its own challenges: a central server is responsible for coordinating training and aggregating updates, creating both a communication bottleneck and a single point of failure [43, 91]. This centralized coordination layer not only limits scalability, but also increases vulnerability to poisoning and backdoor attacks [25, 91]. Such risks pose significant barriers to deploying federated learning in mission-critical applications such as energy demand prediction, where reliability and auditability are essential.

To address the aforementioned challenges, we propose BLEND, a consortium-blockchain framework that integrates blockchain with federated learning to decentralize aggregation and ensure auditable coordination [26, 27, 120]. By recording verified model updates on an immutable ledger, BLEND reduces dependence on a trusted central server and enables transparent, programmable incentives through smart contracts [28, 121]. BLEND incorporates three core components: (i) a decentralized agentic workflow, where Publisher, Miner, Evaluator, and Leader roles are governed by smart contracts to ensure fair participation and verifiable coordination; (ii) a Proof-of-Forecast protocol that ties block acceptance to forecasting accuracy on hold-out validation data, incentivizing performance over participation; and (iii) a task-aligned LLM forecaster, adapted to time series through data-efficient instruction tuning (DEITA) [122] and direct preference optimization (DPO) [106], with parameter-efficient fine-tuning via quantized low-rank adaptation (QLoRA) [96] for scalable edge deployment. At the system layer, blocks commit signed updates and evaluation traces, enabling reproducible aggregation and incentive accounting. The main contributions of this chapter are as follows:

- We introduce Proof-of-Forecast, a novel performance-driven consensus protocol that selects updates by held-out validation error under a Byzantine-resilient vote. Coupled with a con-

sortium blockchain, it provides immutable auditability and aligns agent incentives through performance-based stake updates, enhancing trust and participation.

- We develop a task-aligned LLM forecaster that fuses tokenized historical time series with exogenous contextual signals, enabling context-aware predictions.
- We validate BLEND on various time series datasets, demonstrating superior forecasting performance over recent centralized and federated learning baselines. BLEND maintains robust performance even with 20% Byzantine clients, showcasing its resilience under adversarial conditions.

The rest of the chapter is organized as follows. Section 4.2 reviews relevant work. In Section 4.3, we first formalize the time series forecasting task, and then present our BLEND system and its main architectural components. We also describe the server-side FedAdam update and on-chain recording. Section 4.4 reports the experimental results. Finally, we present a discussion in Section 4.5.

4.2 Related Work

LLM- and FL-based Forecasting Methods. Significant progress has been made in long-term time-series forecasting through Transformer-based architectures, which excel at capturing long-range dependencies [13–18]. For instance, FEDformer [16] combines seasonal-trend decomposition with Fourier and Wavelet transforms to capture global temporal patterns, while iTransformer [18] inverts the conventional Transformer structure by treating entire time series of individual variates as tokens, using attention to model inter-series correlations and feed-forward networks for feature learning. PatchTST [17] segments inputs into subseries-level patches and introduces channel independence, which improves robustness and adaptability by isolating attention pathways for each channel, in contrast to channel-mixing strategies. Recently, pre-trained LLMs have been adapted for time series forecasting [19, 20]. Since LLMs are pretrained on text, their application to numerical time series requires careful task alignment and prompt design. In our BLEND framework, we leverage data-efficient instruction tuning and direct preference optimization to fine-tune LLM backbones for forecasting while preserving the underlying architecture. On the other hand, FL has emerged as a promising paradigm for forecasting with sensitive data by enabling decentralized training without raw data centralization [22]. Recent FL-based methods have demonstrated the effectiveness of FL in residential load forecasting [123, 124] and explored

clustering-based personalization strategies [23]. FedTime [2] introduces a federated LLM forecaster that incorporates K-means clustering and fine-tuning with alignment strategies to enhance forecasting performance, while Time-FFM [125] proposes a federated foundation model by transforming time series into text tokens and leveraging pre-trained language models with a prompt adaptation module and personalized federated training. However, these FL-based methods still rely on a central server for coordination, introducing vulnerabilities to single-point failures and poisoning attacks. Such risks are particularly concerning in IoV applications, where malicious updates could disrupt energy management. BLEND addresses these challenges by integrating a consortium blockchain with a Proof-of-Forecast consensus protocol, decentralizing coordination, ensuring auditability, and enhancing robustness against Byzantine faults.

Blockchain-Enabled Federated Learning. To address FL’s reliance on a central aggregator, various approaches have adopted blockchain [26, 27] to enable decentralized aggregation and enhance security. Early frameworks like BlockFL [28] and incentive-driven systems such as DeepChain [29] demonstrated that distributed ledgers can replace or complement centralized coordination. Subsequent work introduced mechanisms for signed and auditable updates [30], on-chain contribution tracking [31], and token-based incentives [32]. While these approaches focus on secure logging and rewards, they largely decouple block production from model performance. In contrast, our BLEND’s Proof-of-Forecast ties consensus directly to predictive accuracy, ensuring that block finality reflects verifiable model utility rather than mere participation.

Agentic AI Systems and Workflows. Agent-based problem solving, rooted in multi-agent systems, has resurfaced with the advent of LLMs in the form of agentic workflows [33], where specialized agents coordinate to complete complex tasks. These LLM-based workflows, however, often depend on centralized orchestration, creating coordination bottlenecks and vulnerabilities in distributed settings. BLEND addresses this limitation by embedding agentic workflows into a blockchain-native framework, using smart contracts to enforce decentralized coordination and incentive alignment. It introduces four distinct roles, Publisher, Miner, Evaluator, and Leader, responsible for data submission, model training, forecast validation, and consensus facilitation, respectively. These roles interact exclusively via smart contracts on a consortium blockchain, ensuring transparency, auditability, and resilience to malicious actors. This modular, decentralized workflow not only supports robust coordination without a central authority but also integrates seamlessly with standard time series forecasting models, making BLEND well-suited for privacy-preserving applications, such as energy demand prediction in IoV systems.

4.3 Proposed BLEND Framework

In this section, we first describe the context-aware multivariate time series forecasting task and then outline our solution strategy through a system-level overview using the IoV network as a running example. Subsequently, we present the core components of our BLEND framework, showing how federated learning and consortium blockchain are integrated with LLMs.

Problem Statement. Multivariate time series forecasting aims to predict a future sequence $\mathbf{X}_{L+1:L+T} \in \mathbb{R}^{T \times M}$ of length T (prediction horizon) from a historical sequence $\mathbf{X}_{1:L} \in \mathbb{R}^{L \times M}$ of length L (look-back window), where $\mathbf{x}_t \in \mathbb{R}^M$ represents a vector of M variates at time t . Adapting LLMs for this task involves discretizing $\mathbf{X}_{1:L}$ into tokens and training autoregressively to generate future sequences [19]. However, this approach often ignores exogenous contextual factors like traffic, weather, or events, which are critical for IoV energy demand forecasting. We extend this formulation to model the conditional distribution $P(\mathbf{X}_{L+1:L+T} | \mathbf{X}_{1:L}, \mathbf{C}_{1:L+T})$, where $\mathbf{C}_{1:L+T}$ includes contextual signals (e.g., weather forecasts, holidays). By fusing tokenized history $\mathbf{X}_{1:L}$ and context $\mathbf{C}_{1:L+T}$ into a unified prompt, BLEND generates forecasts that are both statistically plausible and responsive to IoV’s dynamic, event-driven environment.

4.3.1 System Overview

BLEND is a *system model* that couples federated learning with a consortium blockchain. The chain acts as the coordinator and source of truth: roles are registered on-chain, model proposals are signed and recorded as transactions, consensus selects the winning update, and a smart contract disburses rewards. Figure 4.1 depicts a training round of BLEND for time series forecasting, using the IoV network as an illustrative example.

Let $\mathcal{A}$ denote a set of agents, where each agent is equipped with a public-private key pair (pk, sk) [126]. We define the agent roles as follows: A *Publisher* agent is responsible for global model deployment and task initialization; an *Oracle* agent continuously scrapes web/news, filters context-aware events (energy, weather, traffic, etc.) and supplies validated textual prompts to the Publisher; *Miner* agents $\mathcal{M} = \{M_1, \dots, M_n\}$ take care of local training on private data and proposal generation; *Evaluator* agents $\mathcal{E} = \{E_1, \dots, E_m\}$ are designated Miners that rigorously evaluate forecast proposals; and a *Leader* agent is elected among the Miners to finalize block creation and broadcast the updated global model. These agents collectively address a forecasting task through the following six steps:

Step 1: Pool Formation & Task Initialization. The Publisher creates an Agents Pool by registering all participating agents in $\mathcal{A}$, defines the time series forecasting task (lookback, horizon,

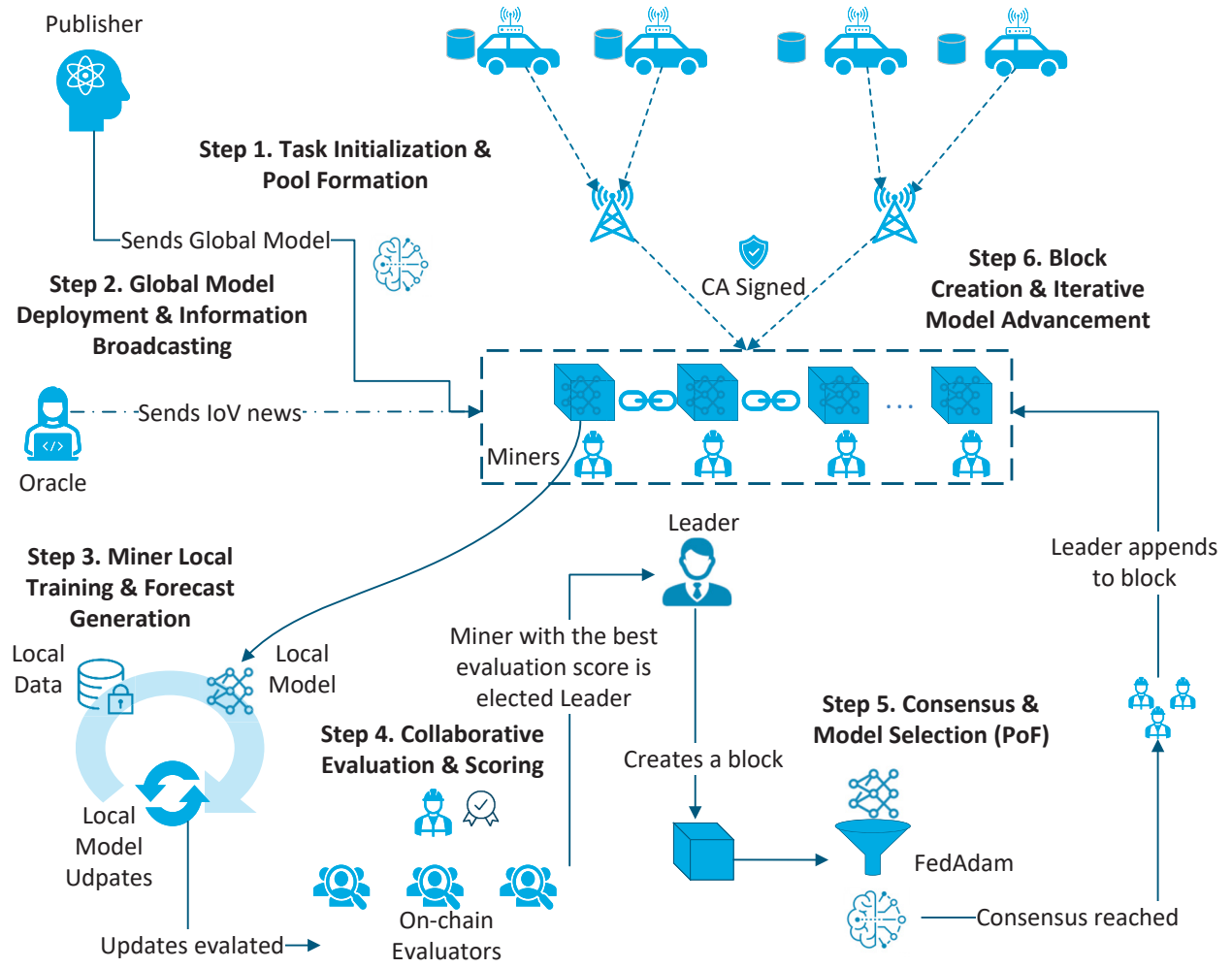


Figure 4.1: **BLEND training round.** (1) Publisher assigns forecasting task; (2) Oracle fetches signals (e.g., traffic, weather) and broadcasts with the current global LLM; (3) Miners fine-tune on private data and submit signed updates; (4) Evaluators audit via proof-of-forecast (PoF) and assign utility scores; (5) Byzantine-tolerant vote selects top update and FedAdam aggregates it into global model. (6) New model and audit trail sealed in blockchain, triggering next round.

metrics), and assigns roles to the remaining agents.

Step 2: Global Model Deployment & Information Broadcasting. The Publisher initiates a global model w_0 , which is an LLM adapted for time series forecasting to the blockchain along with the information retrieved by the Oracle and the metadata describing the task. The Publisher signals round $r = 0$, once recorded on-chain, all Miners retrieve w_0 as the starting point for local training as their initial weights. In parallel, the Oracle scrapes additional signals (e.g., Web sources, social networks, and weather/energy APIs) and bundles them with the current global LLM w_r , broadcasting the package to all Miners.

Step 3: Miner Local Training & Forecast Generation. Each Miner M_i verifies the context

signature shared by the Publisher and appends the summary information to their local prompt. The Miner trains or fine-tunes the global model $\mathbf{w}_r$ on its local dataset $\mathcal{D}_i$ to generate a local model update $\mathbf{w}_i^r$, which represents the Miner’s forecast proposal for the current round r . The Miner uses its private key sk_i to create a digital signature for the proposal. This signature, denoted $\text{Sig}_{\text{sk}_i}(\mathbf{w}_i^r)$, is generated by applying a cryptographic signing algorithm [126] to the model update. Subsequently, the Miner submits a signed proposal $\hat{\mathbf{w}}_i^r = (\mathbf{w}_i^r, \text{Sig}_{\text{sk}_i}(\mathbf{w}_i^r))$, a tuple containing the model update and its signature, to the Evaluator agents for verification.

Step 4: Collaborative Evaluation & Scoring. Evaluators, typically the top three performers with the lowest validation error, collect the signed proposals $\{\hat{\mathbf{w}}_i^r\}$ and compute evaluation scores $\{s_i^r\}$ for each proposal (forecast) (e.g., using MSE on a validation set), for all $i \in \mathcal{M}$. In parallel, tokenized rewards $\{R_i^r\}$ are assigned to agents according to their contributions, with distribution managed by a smart contract under the authority of a designated Leader agent. For auditability with fewer lookups, we bundle the three elements into a single record $\text{tx}_r = \{(\hat{\mathbf{w}}_i^r, s_i^r, R_i^r)\}$, which is collectively signed by Evaluators in preparation for the forecast evaluation to produce a preliminary block B after consensus is achieved [30–32].

Step 5: Consensus & Model Selection. A Leader is elected from the pool of Miners and is responsible for broadcasting the preliminary block B to all Miners for validation. Consensus is achieved if at least 7 out of 10 Miners (i.e., $\lceil 0.67 \cdot 10 \rceil = 7$ endorsements) approve B, in which case the block is appended to the blockchain. The forecast proposal with the highest evaluation score is then selected as the best forecast model, as it yields the most accurate prediction. If consensus is not reached within r debate rounds, the process restarts from Step 2, where the debate is repeated up to $r_{\max}$ rounds and a new leader is chosen.

Step 6: Block Creation & Iterative Model Advancement. The chosen Leader finalizes the updated global model by integrating the selected best forecast into a new global model $\mathbf{w}_r$. This block is then published to the blockchain, and all agents update their local records accordingly. The updated model serves as the starting point for the next iteration, enabling continuous refinement in time series forecasting through federated LLM training. Using FedAdam [127], the Leader applies bias correction to the moment estimates of the gradient and performs a single update to derive the new global model $\mathbf{w}_r$ for the subsequent round. The Leader embeds tx_r and the new model $\mathbf{w}_{r+1}$ in a block, publishes it, and disburses rewards. Then, all agents proceed to the next round.

In the following subsections, we present how blocks are produced and finalized via Proof-of-Forecast, how agents are incentivized, and the Miner LLM forecaster that generates proposals.

4.3.2 Proof-of-Forecast Consensus Protocol

In order to establish collaboration in BLEND, we propose a specialized Proof-of-Forecast protocol, where predictive accuracy serves as the scarce resource guiding block production. As Step 5 of BLEND’s training paradigm, each round begins with the Publisher broadcasting an immutable digest that includes the current global model, the Oracle-curated context, and a randomly sampled forecast horizon. Miners fine-tune the global model on their private data and submit signed proposals to Evaluators. These Evaluators score every submission on a hold-out validation slice and assemble a candidate block containing the tuple the signed proposals, evaluation scores and tokenized rewards for all miners at a given round. A lightweight Byzantine-resilient vote (weighted by historical stake and slashed for dishonest behavior) elects the forecast with the minimum error as the *winning proof*. The Miner supplying this proof becomes the temporary Leader, merges its update into the global weights, and gossips the new block. BLEND binds block finality to verifiable predictive accuracy and deterministically appoints as Leader the miner whose proposal achieves the lowest validation loss. Proof-of-Forecast simultaneously (i) incentivises agents to contribute genuinely informative updates, (ii) thwarts Sybil or model poisoning attacks, and (iii) guarantees that every committed block tangibly advances the federated forecasting objective.

4.3.3 Incentive Mechanism

The incentive layer converts evaluation outcomes from Step 4, as depicted in Figure 4.1, into on-chain rewards and stake updates, which, in turn, shape participation and voting in Steps 5-6. In each round r , every agent $i \in \mathcal{A}$ holds a nonnegative stake $\text{stk}_i^r \geq 0$, which determines their influence in the consensus process. Let $\mathcal{M}^r$ denote the set of Miners that submit signed proposals in round r . Evaluators score these proposals by verifying signatures and computing a task-specific loss $\ell(\mathbf{w}_i^r)$, such as MSE, on a held-out validation set. This update-score-reward pipeline follows established practice in blockchain-enabled federated learning [30]. We normalize the losses $\ell(\mathbf{w}_i^{(r)})$ via min-max normalization to produce evaluation scores s_i^r as follows:

$$s_i^r = 1 - \frac{\ell(\mathbf{w}_i^r) - \min_{j \in \mathcal{M}^r} \ell(\mathbf{w}_j^r)}{\max_{j \in \mathcal{M}^r} \ell(\mathbf{w}_j^r) - \min_{j \in \mathcal{M}^r} \ell(\mathbf{w}_j^r)} \in [0, 1], \quad (4.1)$$

where higher scores indicate better predictive performance. These normalized scores ensure stable consensus thresholds across diverse datasets and forecasting horizons, aligning incentives and supporting robust leader selection.

Reward Vector. At every round r , an agent $i \in \mathcal{A}$ earns at most one type of compensation,

determined by the specific role it has played in that round. We define the reward vector as

$$\mathbf{R}_i^r = (R_i^{\text{Miner},r}, R_i^{\text{Eval},r}, R^{\text{Oracle},r}, R^{\text{Lead},r}), \quad (4.2)$$

where $R_i^{\text{Miner},r}$ is the Miner performance reward, $R_i^{\text{Eval},r}$ is the evaluation-honesty reward (or penalty), $R^{\text{Oracle},r}$ is the Oracle information-gain reward (external-Oracle incentive), and $R^{\text{Lead},r}$ is the Leader commitment reward (block-leadership stipend). In each round, the reward vector has exactly one non-zero component, corresponding to the role performed by the related agent, with the other three entries set to 0. This vector notation ensures a unified update rule for stake and accounting, while preserving the exclusivity of rewards per role:

(a) *Miner Performance Reward.* To remunerate forecasting quality, we define the Miner performance reward as

$$R_i^{\text{Miner},r} = \beta_r s_i^r, \quad (4.3)$$

where s_i^r is the normalized evaluation score and $\beta_r > 0$ is a round-dependent scale (annealed over r to prevent inflation). The proposed reward is monotone in predictive performance, budget-bounded as it lies in the interval $[0, \beta_r)$, and scale-invariant due to the normalization of the evaluation scores.

(b) *Evaluation-Honesty Reward.* To ensure the integrity of the evaluation process, we define an evaluation-honesty reward as

$$R_i^{\text{Eval},r} = \begin{cases} \gamma_r, & \text{if } i = \arg \max_{j \in \mathcal{M}^r} s_j^r \text{ and } i = E^r, \\ -\lambda_r, & \text{if } i \in \mathcal{M}^r \setminus E^r \text{ and } |s_i^r - \bar{s}_r| > \tau, \\ 0, & \text{otherwise,} \end{cases} \quad (4.4)$$

where $\bar{s}_r$ and $\mathcal{M}^r$ are the average evaluation score and the set of all Miners of the r -th round, respectively, $\gamma_r > 0$ is the bonus for correctly ranking the best model, $\lambda_r > 0$ is the slashing coefficient, and $\tau > 0$ is a tolerance threshold. Intuitively, the reward is determined based on the agent's score relative to others. If agent i achieves the highest score s_j^r among all Miners for round r , and this agent is also the designated evaluator E^r for that round, then it receives a positive reward of γ_r , which is a predefined reward value for round r , incentivizing the agent for producing the best evaluation. On the other hand, if agent i is a Miner but not the designated evaluator E^r , and its score s_i^r deviates from the average score $\bar{s}_r$ across all Miners by more than the threshold τ , then the agent incurs a penalty of $-\lambda_r$, discouraging evaluations that are significantly inconsistent with the consensus. Our proposed reward function is designed to both incentivize truthful reporting and defend against malicious contributions. It employs a peer prediction-style incentive [128, 129] to reward the evaluator who correctly identifies the round's best-performing model. It also provides a

crucial defense against model poisoning and backdoor attacks [25, 130, 131] by penalizing Miners whose submissions are outliers. In other words, this dual approach encourages accurate evaluations while deterring bad-faith actors whose submissions deviate significantly from the consensus.

(c) *Oracle Information–Gain Reward.* The reward for the Oracle is proportional to the value of the information it provides. This aligns with principles of equitable data valuation, where a contributor’s reward is based on their marginal contribution to the model’s performance [132]. Concretely, let p_{without}^r and p_{with}^r denote the prior (baseline) and posterior predictive distributions of the forecaster without and with the Oracle’s context for round r . We quantify the information gain (IG) as the Kullback-Leibler divergence of the prior distribution from the posterior distribution:

$$\text{IG}_r = \text{KL}(p_{\text{with}}^r || p_{\text{without}}^r), \quad (4.5)$$

and define the Oracle’ reward as

$$R^{\text{Oracle},r} = \eta_r \text{IG}_r, \quad (4.6)$$

where $\eta_r > 0$ is the evaluation stipend. Eqs. (4.5)–(4.6) instantiate a pay-for-information mechanism grounded in the D_{KL} divergence: if the Oracle’ context does not shift the predictive distribution ($\text{IG}_r = 0$), no reward is issued.

(d) *Leader Commitment Reward.* To incentivize the final step of block creation, a Leader commitment reward is issued. This stipend is a core component of blockchain incentive design [133] and is made conditional on achieving network consensus. We enforce this through a signature quorum, a standard mechanism for ensuring Byzantine fault tolerance [46]. Formally, we define the Leader commitment reward as

$$R^{\text{Lead},r} = \begin{cases} \kappa_r, & \text{if } N_{\text{sig}} \geq q, \\ 0, & \text{otherwise,} \end{cases} \quad (4.7)$$

where N_{sig} is the number of validator signatures received for the proposed block, q is the minimum quorum threshold, and $\kappa_r > 0$ is the block-creation stipend. The reward is disbursed only upon successful block finalization, encouraging the Leader to propose a valid block that the network will accept. In any given round, only the reward component corresponding to the role an agent performs is non-zero; all other entries in their reward vector are explicitly set to 0.

Stake Dynamics. Each agent holds a non-negative stake stk_i^r , updated after each round according to its performance, following the proof-of-stake principle where participation and leader eligibility are weighted by stake [134]. We define the stake update as follows:

$$\text{stk}_i^{r+1} = \max \left\{ 0, (1 - \sigma) \text{stk}_i^r + \sum_{c \in \mathcal{C}} R_i^{c,r} \right\}, \quad (4.8)$$

where $\mathcal{C} = \{\text{Miner}, \text{Eval}, \text{Oracle}, \text{Lead}\}$, and $\sigma \in [0, 1)$ is a mild stake decay parameter that prevents the perpetual dominance of early high-performing agents. An agent’s stake determines its eligibility to participate; only agents whose stake ranks within the top $\lceil \alpha |\mathcal{M}| \rceil$ fraction are designated as Miners for the subsequent round. Furthermore, a Miner whose stake is reduced to zero is temporarily suspended for a certain number of rounds. This form of “stake slashing” serves as a significant penalty for consistently poor or malicious performance.

Robustness Against Noise. To guide the system towards a high-quality, stable consensus, BLEND employs an annealing policy for its reward coefficients, balancing exploration and exploitation [135]. In the initial exploration phase, large values for the accuracy bonus β_r and evaluation stipend η_r encourage diverse contributions. As training progresses into the exploitation phase, these coefficients gradually decay while the stake-weighted amplification γ_r and leader subsidy κ_r increase, prioritizing convergence and stability. Robustness against adversarial behavior is ensured via τ , and it is tied to the inter-quartile range of the evaluation score distribution, mitigating the impact of outliers. The slashing penalty is bounded by the performance reward ($\lambda_r \geq \beta_r$), ensuring a malicious forecast results in an expected net stake loss. Block finality requires a super-majority quorum of $q > \frac{2}{3} |\mathcal{M}|$, making the protocol tolerant to up to one-third Byzantine miners [46].

The design of our stake dynamics provides inherent security properties. Sybil resistance is a direct consequence of the stake-update rule in Eq. (4.8). A new identity enters the system with zero stake and cannot gain influence without first repeatedly earning rewards through valuable contributions, thereby preventing stake inflation via identity splitting [136]. The design also guarantees oracle verifiability. Since the Oracle’s compensation is proportional to the realized information gain $\Delta I^{(r)}$, it accrues no benefit when its data does not measurably improve the forecast. This comprehensive mechanism incentivizes accurate local forecasts, rewards honest evaluation, and gradually filters out malicious or low-effort agents, all of which is fully on-chain and auditable within the BLEND framework.

4.3.4 Miner LLM Forecaster

With consensus, incentives and stake dynamics described in Subsections 4.3.2-4.3.3, we now present the *Miner agent’s* forecasting LLM that operates within the BLEND system (Figure 4.1, Step 3). The model produces round-level proposals evaluated and rewarded by the PoF protocol. The LLM forecaster follows a three-phase pipeline, as shown in Figure 4.2: a one-time, offline *Model Alignment* of a time series forecaster; followed by an online workflow comprising *Input Representation and Prompt Construction*, which fuses historical and contextual signals into a uni-

fied prompt; and *Autoregressive Forecasting*, which generates the final prediction used in scoring and aggregation.

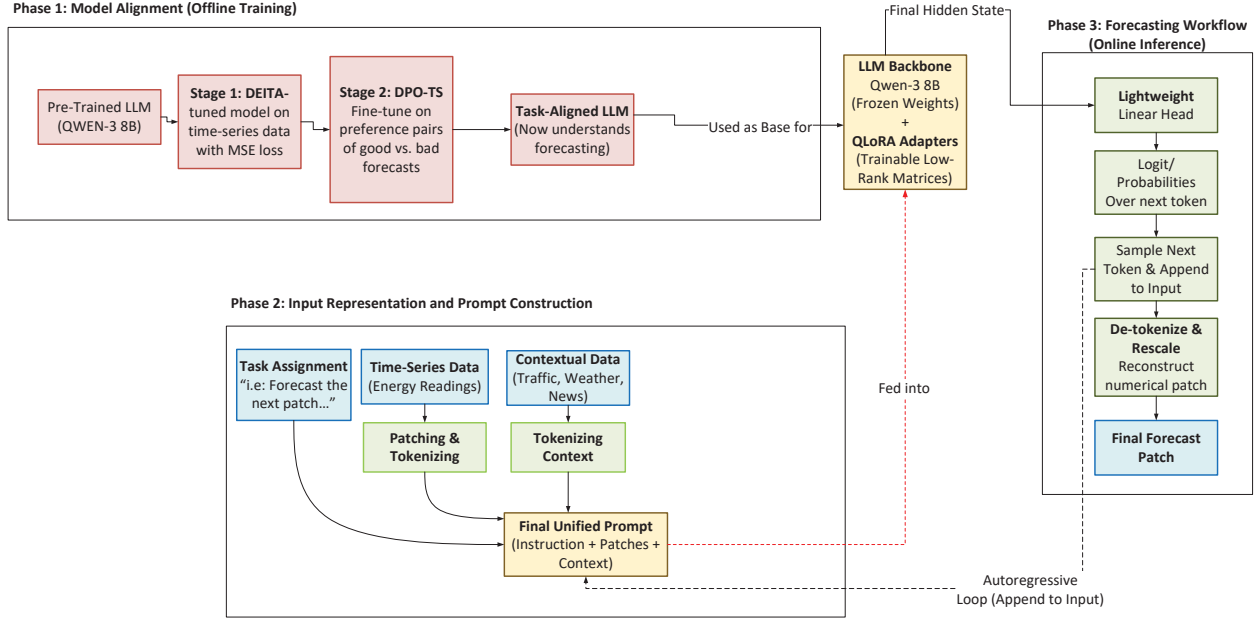


Figure 4.2: The LLM forecaster in BLEND consists of three distinct phases. The offline **Model Alignment** phase (Phase 1, top) first adapts a pre-trained LLM for forecasting using a two-stage process of DEITA and DPO-TS to produce a specialized, task-aligned model. The online inference process begins with **Input Representation** (Phase 2, bottom left), where time-series and contextual data are tokenized and assembled into a unified prompt. Finally, in the **Forecasting Workflow** (Phase 3, right), this prompt is fed to the frozen, task-aligned backbone with QLoRA adapters to generate the final forecast patch used in PoF scoring and aggregation.

Phase 1: Model Alignment (Offline). The objective of this initial phase is to adapt a pre-trained LLM for the specific time-series forecasting task. This is achieved through a sequential two-stage fine-tuning process.

1. *Instruction Tuning with DEITA*: We first perform supervised fine-tuning using the Data-Efficient Instruction Tuning (DEITA) methodology [122]. The model is trained on a curated dataset of instruction-response pairs where the objective function is explicitly set to minimize MSE, a canonical regression metric, rather than a standard language modeling loss function. This stage teaches the model to respect temporal dependencies and prioritize forecast performance.
2. *Preference Tuning with DPO-TS*: Next, we refine the model’s behavior by adapting the Direct Preference Optimization (DPO) framework [106] for the time series domain, a process we term DPO-TS. Standard DPO is designed for language-based preferences. To apply it to

forecasting, we generate pairs of “winning” (more accurate, i.e., lower MSE) and “losing” (less accurate) forecast trajectories for a given context. The model is then fine-tuned using these preference pairs, enabling it to discern high-quality forecast patterns directly from data, rather than from an explicit reward model.

The output of this offline process is the Task-Aligned LLM, a specialized model that serves as a frozen, expert backbone during the online inference phase.

Phase 2: Input Representation and Prompt Construction. At each inference step, a context-aware prompt is dynamically constructed by fusing information from different sources:

1. *Historical Time Series Data:* The most recent history of the time series (e.g., energy readings) is partitioned into overlapping patches, following PatchTST [17]. These patches are subsequently tokenized into sequences of discrete digits.
2. *Exogenous Contextual Data:* Concurrent, unstructured data related, for instance, to the IoV network (e.g., traffic updates, weather alerts, or news feeds) are tokenized into a separate sequence to provide real-time context.
3. *Task Instruction:* A static, natural-language instruction prompt (e.g., “Forecast the next patch given the following history and context...”) is provided to define the model’s objective for the current task.

The tokenized time series patches, the contextual tokens, and the task instruction are then concatenated to form the Final Unified Prompt. This unified input structure enables the LLM to reason over historical patterns, real-world events, and a specific objective simultaneously.

Phase 3: Autoregressive Forecasting (Online). The final forecast is generated in an autoregressive loop. The unified prompt from Phase 2 is fed into the frozen Task-Aligned LLM, which is augmented with a parameter-efficient fine-tuning (PEFT) method. Specifically, we use QLoRA [96], which inserts small, trainable low-rank adapter matrices into each layer of the LLM architecture while keeping the main backbone weights frozen. This 4-bit quantized tuning approach significantly reduces the computational and memory footprint, making it practical for deployment on edge devices such as Roadside Units (RSUs). The model’s final hidden state is passed through a lightweight linear head to produce a probability distribution over the next token. A token is sampled, appended to the input sequence, and the process repeats until a full forecast patch is generated. This patch is then de-tokenized and rescaled to yield the final, context-aware time series forecast.

4.3.5 Federated Optimization

In the BLEND framework, the distributed model optimization is orchestrated by the Leader agent, which acts as the server in the adaptive federated optimization (FedAdam) [127]. The Miner agents act as the clients, performing local computations. As illustrated in Algorithm 3, the process for each communication round r is as follows:

Miner’s Local Computation. Each participating Miner agent M_i receives the current global model, $\mathbf{w}_r$, from the Publisher, computes an unbiased estimate of the gradient of its local loss function with respect to the model parameters on its local dataset $\mathcal{D}_i$, updates the local model $\mathbf{w}_i^r$ using stochastic gradient descent, and then sends the model delta $\Delta_i^r = \mathbf{w}_r - \mathbf{w}_i^r$ (i.e., total local update) to the designated Leader agent for the current round.

Leader’s Aggregation and Adam Update. The Leader agent collects the model deltas $\{\Delta_i^r\}$ from all participating Miners, and then computes the weighted average delta as follows:

$$\Delta_r = \sum_{i=1}^{|\mathcal{M}|} \frac{n_i}{N} \Delta_i^r, \quad (4.9)$$

where $n_i = |\mathcal{D}_i|$ and $N = \sum_i n_i$ is the total number of samples across all Miners in $\mathcal{M}$. Then, the Leader applies the Adam update rule to update the first and second moment of gradients:

$$\mathbf{m}_r = \beta_1 \mathbf{m}_{r-1} + (1 - \beta_1) \Delta_r, \quad (4.10)$$

$$\mathbf{v}_r = \beta_2 \mathbf{v}_{r-1} + (1 - \beta_2) \Delta_r^2, \quad (4.11)$$

where $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the moment estimates.

Global Model Update. The Leader performs bias correction on the moments and updates the global model. This single server-side update step yields the new global model for the next round:

$$\mathbf{w}_{r+1} = \mathbf{w}_r - \eta \frac{\mathbf{m}_r}{\sqrt{\mathbf{v}_r + \epsilon}}, \quad (4.12)$$

where $\mathbf{m}_r$ and $\mathbf{v}_r$ are the bias-corrected moment estimates, η is the global learning rate, and ϵ is a small constant ensuring numerical stability. The value of ϵ governs the level of adaptivity in the algorithm, with smaller ϵ values corresponding to greater adaptivity.

4.4 Experiments

In this section, we present a comprehensive empirical evaluation of BLEND, assessing its forecasting performance against state-of-the-art baselines, and its robustness under adversarial conditions. These evaluations are carried out on multiple public benchmark datasets, complemented by an ablation study to assess the contributions of key architectural components.

Algorithm 3 BLEND: Blockchain-Enhanced Federated Training for IoV with FedAdam Optimization

```

1: Input: Vehicles  $\mathcal{V}$ , Certification Authority CA; Oracle, Publisher, Miner, Evaluator, Leader agents; Blockchain ledger  $\mathcal{B}$ ; Pretrained model  $\mathbf{w}_0$ ; Number of rounds  $R$ ; Number of local epochs  $E$ ; Learning rate  $\eta$ ; QLoRA( $r, \alpha$ ), Quantization  $q$ ; FedAdam hyperparameters  $(\beta_1, \beta_2, \epsilon)$ .
2: Output: Converged global model  $\mathbf{w}_r$ 
3: Initialization & Broadcast:
4:   For each  $i \in \mathcal{V}$ : Register with CA, get stake  $s_i$ 
5:   Initialize ledger  $\mathcal{B}$ ; Upload  $\mathbf{w}_0$ 
6:   Oracle fetches initial IoV news
7:   Publisher broadcasts  $\mathbf{w}_0$  with news
8:   Initialize FedAdam states:  $\mathbf{m} \leftarrow \mathbf{0}$ ,  $\mathbf{v} \leftarrow \mathbf{0}$ 
9: Local Training & Submission:
10: for each Roadside Unit do
11:   for each round  $r = 1, \dots, R$  do
12:     while model not converged do
13:       for all  $v \in \mathcal{V}$  in parallel do
14:          $\mathbf{w}_i^r \leftarrow \mathbf{w}_r$ ; Attach QLoRA( $r, \alpha$ )
15:         for each epoch  $r = 1, \dots, E$  do
16:           Compute gradient  $g_i^r$  of local loss function
17:           Update model  $\mathbf{w}_i^r \leftarrow \mathbf{w}_i^r - \eta \text{QLoRA}(g_i^r)$ 
18:         end for
19:          $\Delta_i^r \leftarrow \mathbf{w}_r - \mathbf{w}_i^r$ 
20:          $\Delta_i^r \leftarrow q(\Delta_i^r)$ ; Compress via  $q$ 
21:         Submit  $(\Delta_i^r, s_i)$  to Evaluators
22:       end for
23:     end while
24:   end for
25: end for
26: Evaluation & Voting:
27: for all Evaluators in parallel do
28:   Verify each stake  $s_i$  on-chain; Vote accept/reject each  $\Delta_i^r$ 
29: end for
30: Aggregation & FedAdam Update:
31: Leader collects and dequantizes approved  $\{(\Delta_i^r, s_i)\}$ 
32: Aggregate model deltas with FedAdam:
    
$$\Delta_r = \sum_{i=1}^{|\mathcal{M}|} \frac{n_i}{N} \Delta_i^r$$

33: Update first and second moments to  $(\mathbf{m}, \mathbf{v})$ 
34: Update model  $\mathbf{w}_r \leftarrow \mathbf{w}_{r-1} - \eta \mathbf{m} / (\sqrt{\mathbf{v}} + \epsilon)$ 
35: New Block Creation
36: Leader creates new block with  $(\mathbf{w}_r, \mathbf{m}, \mathbf{v})$ 
37: Appends updated model to ledger  $\mathcal{B}$ 
38: Publisher broadcast  $\mathbf{w}_r$ ;
39: Oracle fetches news at next  $r + 1$ 
40: return  $\mathbf{w}_r$ 

```

4.4.1 Experimental Setup

Datasets. We evaluate BLEND on seven widely adopted time series benchmarks: the four ETT datasets (ETTh1, ETTh2, ETTm1, ETTm2) [14], Weather¹, Electricity², and Traffic³ datasets. The ETT datasets contain transformer load and oil-temperature readings, recorded hourly for ETTh1/2 and every 15 minutes for ETTm1/2. The Weather dataset comprises 21 meteorological features (e.g., air temperature, humidity) sampled every 10 minutes. The Electricity data contains the hourly power consumption of 321 households from 2012 to 2014. Finally, the Traffic dataset describes the hourly occupancy rate of road segments on San Francisco Bay Area freeways, as

¹<https://www.bgc-jena.mpg.de/wetter>

²<https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014>

³<http://pems.dot.ca.gov>

recorded by California’s PeMS. For all experiments, we partition each dataset into training (80%), validation (10%), and test (10%) sets. Key statistics for each dataset are summarized in Table 4.1.

Table 4.1: Summary statistics of benchmark datasets.

Dataset	Features	Timesteps	Granularity
ETTh1	7	17,420	1 hour
ETTh2	7	17,420	1 hour
ETTm1	7	69,680	15 min
ETTm2	7	69,680	15 min
Weather	21	52,696	10 min
Electricity	321	26,304	1 hour
Traffic	862	17,544	1 hour

Baselines. We benchmark BLEND against a broad set of state-of-the-art methods: (i) LLM-based: GPT4TS [19], Time-LLM [20]; (ii) Transformer-based: PatchTST [17], FEDformer [16], iTransformer [18]; (iii) Linear/MLP-based: DLinear [108], TimesNet [117], TimeMixer [118], SimpleTM [137]; and (iv) frequency-based: FITS [138]. We also compare BLEND against federated baselines, including Time-FFM [125], FedTime [139], and Fed-PatchTST (federated version of PatchTST).

Implementation Details. All experiments were performed on a cluster of three NVIDIA A5000 GPUs (48 GB RAM each). To emulate a distributed IoV deployment, every vehicle node is instantiated as a separate process and pinned to a simulated Jetson Orin with 64 GB RAM. The end-to-end wall-clock for one federated round, including local training, consensus voting, FedAdam aggregation, and block finalization, is ≈ 1.3 seconds. The blockchain layer uses Hyperledger Fabric v2.4 with an endorsement policy requiring 7 of 10 miners to validate each new block. The agent pool (Publisher, Miners, Evaluators, Leader) is implemented with a 4-bit quantized Qwen-3 8B model to reduce memory while preserving on-chain task performance; the Oracle runs a lightweight distilled 1.7B Qwen model for low-latency context retrieval. Parameter-efficient fine-tuning employs QLoRA with rank $r=16$ and scaling factor $\alpha=32$. On each miner, local training runs for 3 epochs with batch size 64. Global optimization follows FedAdam with server learning rate $\eta=10^{-3}$ and Adam moments $\beta_1=0.9$, $\beta_2=0.999$; the system is trained for up to 200 federated rounds to ensure convergence.

Evaluation Metrics. We evaluate forecasting performance using two standard regression metrics: Mean Squared Error (MSE) and Mean Absolute Error (MAE). MSE is chosen for its sensitivity to large deviations, making it particularly relevant for penalizing significant forecast errors that

could impact critical infrastructure. In contrast, MAE offers a more robust measure of the average forecast bias. Consistent with the literature, we report both metrics across various forecasting horizons of length T , each conditioned on a look-back window of length L . For both metrics, lower values indicate higher forecasting accuracy.

Table 4.2: Forecasting performance (MSE and MAE) on seven benchmark datasets for varying prediction horizons $T \in \{96, 192, 336, 720\}$. Best results are **bolded** and second-best are underlined.

Dataset	T	BLEND		Time-LLM		PatchTST		GPT4TS		iTransformer		TimeMixer		FITS		SimpleTM		DLinear		TimesNet		FEDformer	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTth1	96	<u>0.363</u>	0.391	0.362	<u>0.392</u>	0.370	0.399	0.376	0.397	0.386	0.405	0.375	0.400	0.372	0.398	0.366	<u>0.392</u>	0.375	0.399	0.384	0.402	0.376	0.419
	192	0.395	0.414	<u>0.398</u>	0.416	0.413	0.421	0.416	0.418	0.441	0.436	0.429	0.421	0.404	0.418	0.422	0.421	0.405	<u>0.416</u>	0.436	0.429	0.420	0.448
	336	<u>0.425</u>	0.427	<u>0.430</u>	0.427	0.422	<u>0.436</u>	0.442	0.433	0.487	0.458	0.484	0.458	0.427	0.435	0.440	0.438	0.439	<u>0.443</u>	0.491	0.469	0.459	0.465
	720	0.442	0.456	0.442	0.457	0.447	<u>0.466</u>	0.477	0.456	0.503	0.491	0.498	0.482	0.424	0.460	0.463	0.462	0.472	0.490	0.521	0.500	0.506	0.507
	Avg	0.406	0.422	<u>0.408</u>	0.423	0.413	<u>0.430</u>	0.465	0.455	0.454	0.447	0.447	0.440	0.407	0.428	0.422	0.428	0.422	0.437	0.458	0.450	0.440	0.460
ETTth2	96	<u>0.270</u>	0.331	0.268	0.328	0.274	<u>0.336</u>	0.285	0.342	0.297	0.349	0.289	0.341	0.271	0.330	0.281	0.338	0.289	0.353	0.340	0.374	0.358	0.397
	192	<u>0.332</u>	0.380	0.329	0.375	0.339	<u>0.379</u>	0.354	0.389	0.380	0.400	0.372	0.392	0.331	0.378	0.355	0.387	0.383	0.418	0.402	0.414	0.429	0.439
	336	0.329	0.380	0.368	0.409	0.329	0.380	0.373	0.407	0.428	0.432	0.386	0.414	0.354	0.390	0.365	0.401	0.448	0.465	0.452	0.452	0.496	0.487
	720	0.321	0.416	<u>0.372</u>	0.420	0.379	<u>0.422</u>	0.406	0.441	0.427	0.445	0.412	0.434	0.377	0.425	0.413	0.436	0.605	0.551	0.462	0.468	0.463	0.474
	Avg	<u>0.325</u>	<u>0.379</u>	0.334	0.383	0.330	0.379	0.381	0.412	0.383	0.407	0.364	0.395	0.333	0.381	0.353	0.391	0.431	0.446	0.414	0.427	0.437	0.449
ETTh1	96	0.272	0.334	0.272	0.334	<u>0.290</u>	<u>0.342</u>	0.292	0.346	0.334	0.368	0.320	0.357	0.303	0.345	0.321	0.361	0.299	0.343	0.338	0.375	0.379	0.419
	192	0.310	0.358	0.310	0.358	0.332	<u>0.369</u>	0.332	0.372	0.377	0.391	0.361	0.381	0.337	0.370	0.360	0.380	0.335	0.365	0.374	0.387	0.426	0.441
	336	0.349	0.381	<u>0.352</u>	<u>0.384</u>	0.366	0.392	0.366	0.394	0.426	0.420	0.390	0.404	0.366	0.395	0.390	0.404	0.369	0.386	0.410	0.411	0.445	0.459
	720	0.376	0.404	<u>0.383</u>	<u>0.411</u>	0.416	0.420	0.417	0.421	0.491	0.459	0.454	0.441	0.415	0.422	0.454	0.438	0.425	0.421	0.478	0.450	0.543	0.490
	Avg	0.327	0.371	<u>0.329</u>	<u>0.372</u>	<u>0.351</u>	0.380	0.388	0.403	0.407	0.410	0.381	0.395	0.355	0.383	0.381	0.396	0.357	0.378	0.400	0.406	0.448	0.452
ETTh2	96	0.161	0.253	0.161	0.253	<u>0.165</u>	<u>0.255</u>	0.173	0.262	0.180	0.264	0.175	0.258	0.162	0.254	0.173	0.257	0.167	0.269	0.187	0.267	0.203	0.287
	192	0.217	0.290	<u>0.219</u>	<u>0.293</u>	0.220	0.292	0.229	0.301	0.250	0.309	0.237	0.299	0.216	0.291	0.238	0.299	0.224	0.303	0.249	0.309	0.269	0.328
	336	0.264	0.322	<u>0.271</u>	<u>0.329</u>	0.274	0.329	0.286	0.341	0.311	0.348	0.298	0.340	0.268	0.325	0.296	0.338	0.281	0.342	0.321	0.351	0.325	0.366
	720	0.345	0.370	<u>0.352</u>	<u>0.379</u>	0.362	0.385	0.378	0.401	0.412	0.407	0.391	0.396	0.348	0.382	0.393	0.395	0.397	0.421	0.408	0.403	0.421	0.415
	Avg	0.247	0.309	<u>0.251</u>	0.313	0.255	0.315	0.284	0.339	0.288	0.332	0.275	0.323	0.249	0.313	0.275	0.322	0.267	0.333	0.291	0.333	0.305	0.349
Weather	96	0.145	0.198	<u>0.147</u>	0.201	0.149	<u>0.198</u>	0.162	0.212	0.174	0.214	0.163	0.209	0.143	0.195	0.162	0.207	0.176	0.237	0.172	0.220	0.217	0.296
	192	0.189	0.234	0.189	0.234	<u>0.194</u>	<u>0.241</u>	0.204	0.248	0.221	0.254	0.208	0.250	0.186	0.230	0.208	0.248	0.220	0.282	0.219	0.261	0.276	0.336
	336	0.245	0.261	0.262	0.279	0.245	0.282	0.254	0.286	<u>0.263</u>	0.290	0.251	0.287	0.236	0.275	0.263	0.290	0.265	0.319	0.280	0.306	0.339	0.380
	720	0.289	0.309	<u>0.304</u>	0.316	0.314	0.334	0.326	0.337	0.358	0.347	0.339	0.341	0.307	0.321	0.340	0.341	0.333	0.362	0.365	0.359	0.403	0.428
	Avg	0.217	0.250	<u>0.225</u>	0.257	0.225	<u>0.264</u>	0.237	0.270	0.258	0.278	0.240	0.271	0.218	0.255	0.243	0.271	0.248	0.300	0.259	0.287	0.309	0.360
Electricity	96	0.132	0.227	<u>0.131</u>	<u>0.224</u>	0.129	0.222	0.139	0.238	0.148	0.240	0.153	0.247	0.134	0.230	0.141	0.235	0.140	0.237	0.168	0.272	0.193	0.308
	192	0.152	0.243	0.152	<u>0.241</u>	0.157	0.240	0.153	0.251	0.162	0.253	0.166	0.256	0.149	0.245	0.151	0.247	0.153	0.249	0.184	0.289	0.201	0.315
	336	0.153	0.243	0.160	0.248	0.163	0.259	0.169	0.266	<u>0.160</u>	<u>0.248</u>	0.163	0.259	0.165	0.260	0.173	0.267	0.169	0.267	0.198	0.300	0.214	0.329
	720	0.184	0.290	0.192	0.298	0.197	0.290	0.206	0.297	<u>0.192</u>	0.298	0.197	0.290	0.203	0.295	0.201	0.293	0.203	0.301	0.220	0.320	0.246	0.355
	Avg	0.155	0.251	<u>0.158</u>	<u>0.252</u>	0.166	0.260	0.167	0.263	0.158	0.252	0.161	0.252	0.163	0.258	0.178	0.270	0.166	0.263	0.192	0.295	0.214	0.327
Traffic	96	0.364	<u>0.253</u>	0.362	0.248	0.360	0.249	0.388	0.282	0.395	0.268	0.462	0.285	0.385	0.260	0.410	0.274	0.410	0.282	0.593	0.321	0.587	0.366
	192	<u>0.380</u>	<u>0.256</u>	0.374	0.247	0.379	0.256	0.407	0.290	0.417	0.276	0.473	0.296	0.397	0.270	0.430	0.280	0.423	0.287	0.617	0.336	0.604	0.373
	336	0.379	0.263	<u>0.385</u>	<u>0.271</u>	0.392	0.264	0.412	0.294	0.433	0.283	0.498	0.296	0.410	0.275	0.449	0.290	0.436	0.296	0.639	0.336	0.621	0.383
	720	0.425	0.277	<u>0.430</u>	0.288	0.432	<u>0.286</u>	0.450	0.312	0.467	0.302	0.506	0.313	0.448	0.290	0.486	0.309	0.466	0.315	0.640	0.350	0.626	0.382
	Avg	0.387	<u>0.271</u>	<u>0.388</u>	0.264	0.421	0.277	0.414	0.294	0.388	0.264	<u>0.390</u>	<u>0.263</u>	0.428	0.282	0.410	0.274	0.433	0.295	0.620	0.336	0.610	0.376

4.4.2 Results and Analysis

Quantitative Results. We evaluate BLEND on seven public benchmarks across prediction horizons $T \in \{96, 192, 336, 720\}$ with a fixed look-back window $L = 720$. Table 4.2 presents a comparative analysis of BLEND against recent state-of-the-art time-series forecasting methods. Across all datasets, BLEND achieves the best or second-best performance in 12 of 14 average met-

ric categories (85.7%), demonstrating robust and consistent prediction performance across diverse forecasting horizons and dataset characteristics. On the ETT datasets, shows particularly strong results on the finer-grained 15-minute benchmarks (ETTh1, ETTm2), where it achieves the lowest average MSE and MAE among all methods. For instance, on ETTm2, its average MSE is the lowest among all baselines. While BLEND achieves the best average MSE on ETTh1, PatchTST delivers the best average performance on ETTh2. This suggests that while BLEND is a consistently strong performer, certain specialized architectures can gain an edge on datasets with specific granularities. Performance gains extend to larger, high-dimensional datasets. On the Weather dataset, BLEND attains the best average MSE and MAE. Similarly, on the Electricity dataset, BLEND again yields the best average MSE and MAE values, demonstrating its robustness in handling datasets with many variates. On the Traffic dataset, BLEND attains the best average MSE, indicating superior handling of large forecast errors, while iTransformer and Time-LLM achieve the best and second-best average MAE, respectively, suggesting a slight advantage in minimizing average forecast bias for this dataset. Across all experiments, the consistent performance of BLEND validates its architectural design, which synergizes a task-optimized LLM for capturing complex temporal patterns with a decentralized, performance-driven proof-of-forecast consensus mechanism. This integration enables BLEND to effectively model long-term dependencies, particularly in long-horizon forecasting, while maintaining scalability and robustness.

Qualitative Analysis. We present a visual analysis of model performance in Figure 4.3, displaying the MSE distribution across seven benchmark datasets for prediction horizons $T \in \{96, 192, 336, 720\}$. BLEND achieves the lowest median and mean MSE, with a notably narrow interquartile range (IQR), reflecting both superior accuracy and consistent performance across diverse forecasting scenarios. In contrast, models such as TimesNet and FEDformer exhibit higher median MSE values (approximately 0.37-0.39) with wider interquartile ranges (IQRs), indicating both degraded performance and increased prediction uncertainty. Most notably, FEDformer performs poorly, with the highest median MSE (approximately 0.4), and significant distributional spread, highlighting its instability under varying conditions. This analysis demonstrates BLEND’s robustness and reliability, driven by its task-aligned LLM and blockchain-enhanced consensus mechanism, which outperform Transformer-, LLM-, and linear-based baselines in both central tendency and stability.

Convergence Analysis. We compare the convergence behavior of three LLM training paradigms over 60 epochs on the Electricity training dataset: (i) centralized (standalone) learning; (ii) vanilla federated learning; and (iii) our BLEND framework, which integrates blockchain-based aggregation. As shown in Figure 4.4, BLEND consistently achieves the lowest validation MSE at each

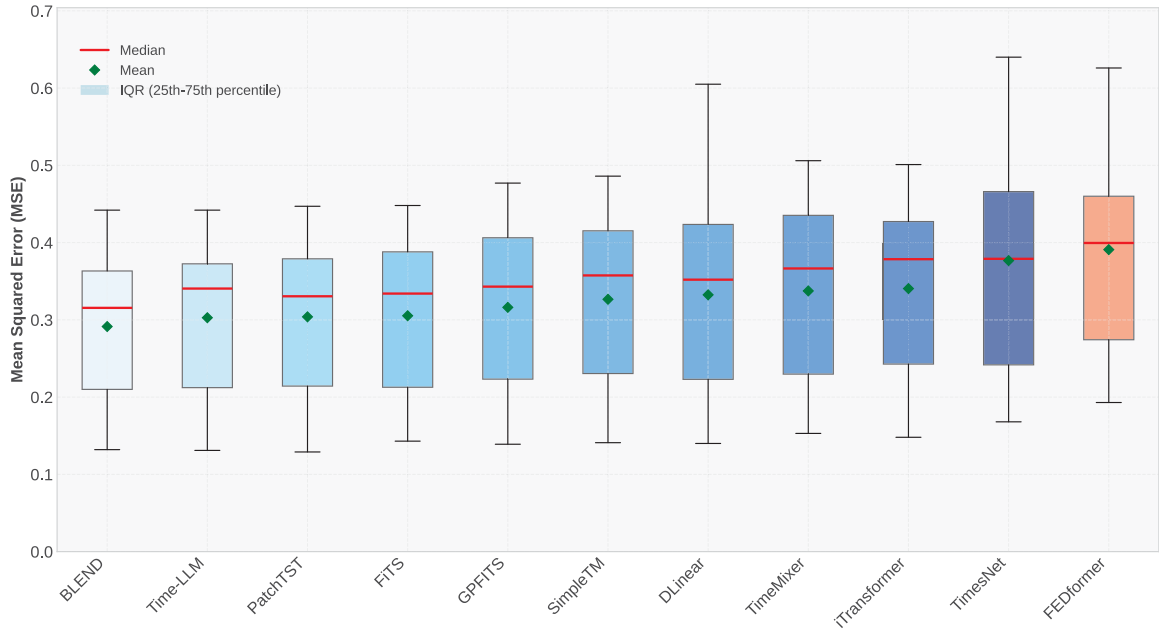


Figure 4.3: **Cross-dataset error distribution.** Box plots show the MSE achieved by each model on the seven public benchmarks for prediction horizons $T \in \{96, 192, 336, 720\}$. The box spans the IQR; whiskers extend to the non-outlier range; the red line marks the median; and the green diamond denotes the mean. BLEND records the lowest median and mean MSE while exhibiting the narrowest IQR, indicating both superior performance and robustness compared with Transformer-, LLM-, and linear-based baselines.

epoch and converges significantly faster than both standalone and federated approaches. After 60 epochs, BLEND converges to a 0.08 MSE, outperforming vanilla federated learning and standalone by 53% and 84%, respectively. This accelerated convergence stems from BLEND’s tamper-resilient consensus mechanism, which filters out corrupted or noisy updates before aggregation, thereby enhancing the stability and robustness of the global model. In contrast, vanilla federated learning exhibits slower improvement and higher variance, reflecting its vulnerability to heterogeneous and potentially malicious client updates. Centralized training, while stable, plateaus at a higher error due to its reliance on a single data source and lack of cross-client diversity. Hence, integrating blockchain into the aggregation loop not only ensures privacy and auditability, but also significantly boosts convergence speed.

Framework Resilience Evaluation. To evaluate the robustness of the BLEND framework under adversarial conditions, we conducted experiments on the ETTh2 dataset, progressively increasing the proportion of Byzantine clients and measuring the resulting performance degradation relative to a clean, centralized training baseline. Figure 4.5 displays the MSE “performance gap” over 60 epochs for three paradigms: centralized learning, vanilla federated learning, and our blockchain-enhanced federation. In the centralized setting, introducing just 3 malicious clients (out of 50)

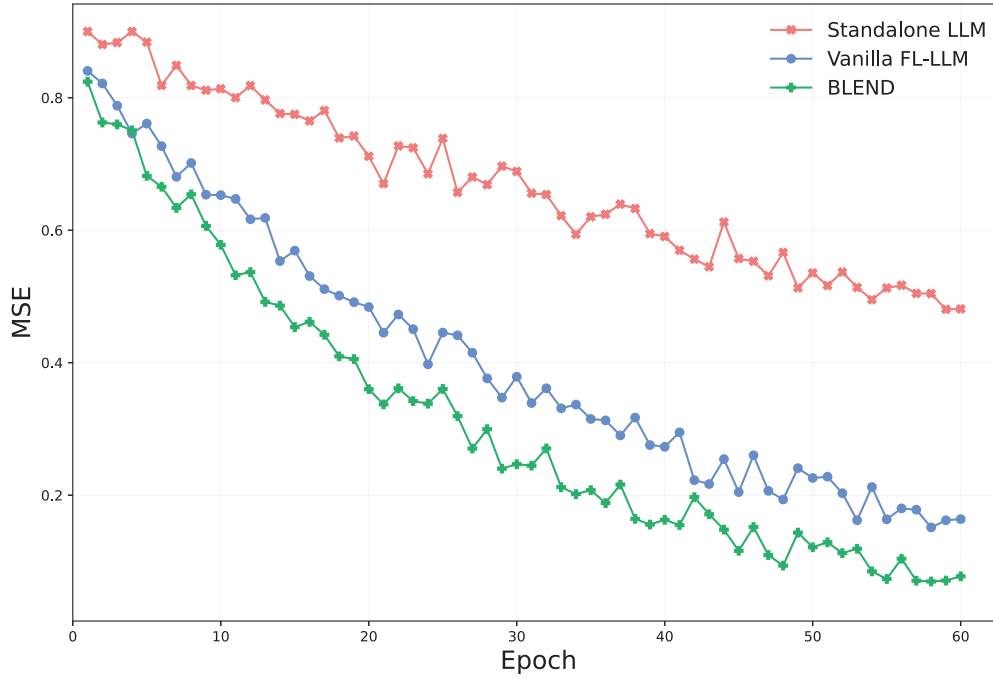


Figure 4.4: Convergence trajectories comparing centralized standalone LLM, vanilla federated LLM, and BLEND over 60 training epochs on the Electricity dataset. BLEND converges faster and to a lower final error by filtering out unreliable updates via its consensus protocol.

causes the performance gap to exceed 0.6 by epoch 10, remaining in an “unacceptable” range for the remainder of training. Vanilla federated learning shows slight improvement: with 3 Byzantine clients, the gap reaches a “degraded” but tolerable level of approximately 0.45 by epoch 20. However, when 6 clients (12%) act adversarially, the gap surpasses 0.6 by epoch 30 and continues to widen, showing federated learning vulnerability to moderate collusion. In contrast, BLEND’s tamper-resilient consensus mechanism effectively mitigates the impact of malicious updates. In the absence of any Byzantine actors, the performance gap steadily decreases from approximately 0.8 at epoch 1 to below 0.2 by epoch 60, staying well within the acceptable range. When up to 6 clients (12% of the federation) turn malicious, BLEND maintains the gap below 0.4 after epoch 30 and never slips into the “degraded” band thereafter. Strikingly, even under a severe 20 % Byzantine fraction (9 of 50 clients), BLEND briefly approaches a gap of 0.55 near epoch 25, momentarily entering degraded performance, but recovers into acceptable range by epoch 40. These results demonstrate that integrating a blockchain-based aggregation layer not only preserves privacy and provides an immutable audit trail but also substantially accelerates convergence and sustains low error rates in the presence of significant adversarial participation. This makes BLEND a robust and secure solution for federated training in challenging scenarios with realistic threat models.

Robustness to Data Poisoning. We test resilience to a 30% poisoning attack on the Traffic dataset.

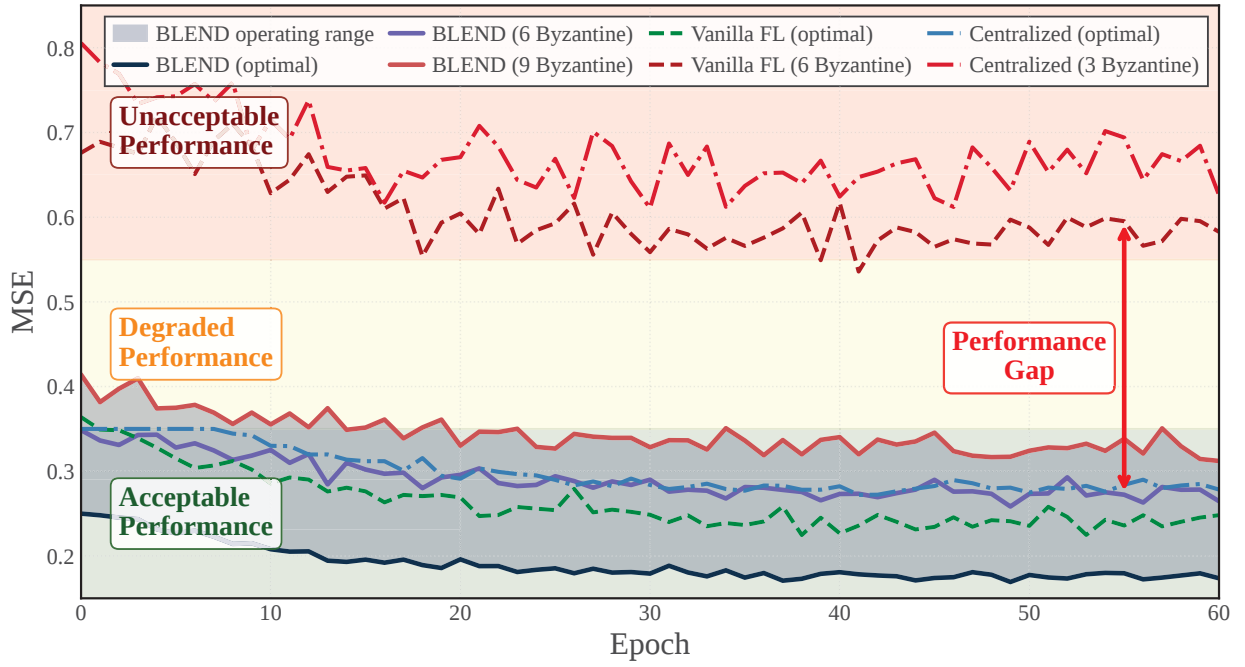


Figure 4.5: Byzantine fault tolerance evaluation of BLEND under varying concentrations of malicious nodes (3, 6, and 9 bad nodes) over 60 training epochs on the ETTh2 dataset, demonstrating BLEND’s resilience against adversarial participants.

As shown in Figure 4.6), we compare three scenarios over 100 training epochs: (i) Federated Learning without poisoning, (ii) Federated Learning with 30% poisoned clients, and (iii) BLEND with 30% poisoned clients. Under benign conditions, standard federated learning converges to an MSE of 0.167. However, introducing 30% corrupted updates significantly degrades performance, with MSE rising to 0.324 by epoch 100, indicating that malicious data severely compromises the global model. In contrast, BLEND mitigates the attack’s effect: despite the same 30% corruption rate, it converges to a 0.201 MSE, which is only 20% worse than its clean baseline and a full 0.123 MSE improvement over the poisoned federated baseline. This 35% relative recovery highlights BLEND’s ability to filter anomalous gradients through consensus mechanisms, preserving both convergence stability and final accuracy under adversarial conditions.

4.4.3 Federated Performance

We benchmark BLEND against three federated approaches: The federated version of PatchTST [17], denoted as Fed-PatchTST; Time-FFM [125]; and FedTime [139]. Table 4.3 presents the comparative results across multiple datasets. BLEND achieves the best results on 10 out of 12 metric comparisons (83.3%), exhibiting effectiveness across varied dataset characteristics, with the most pronounced gains observed on the Weather and Electricity datasets. On

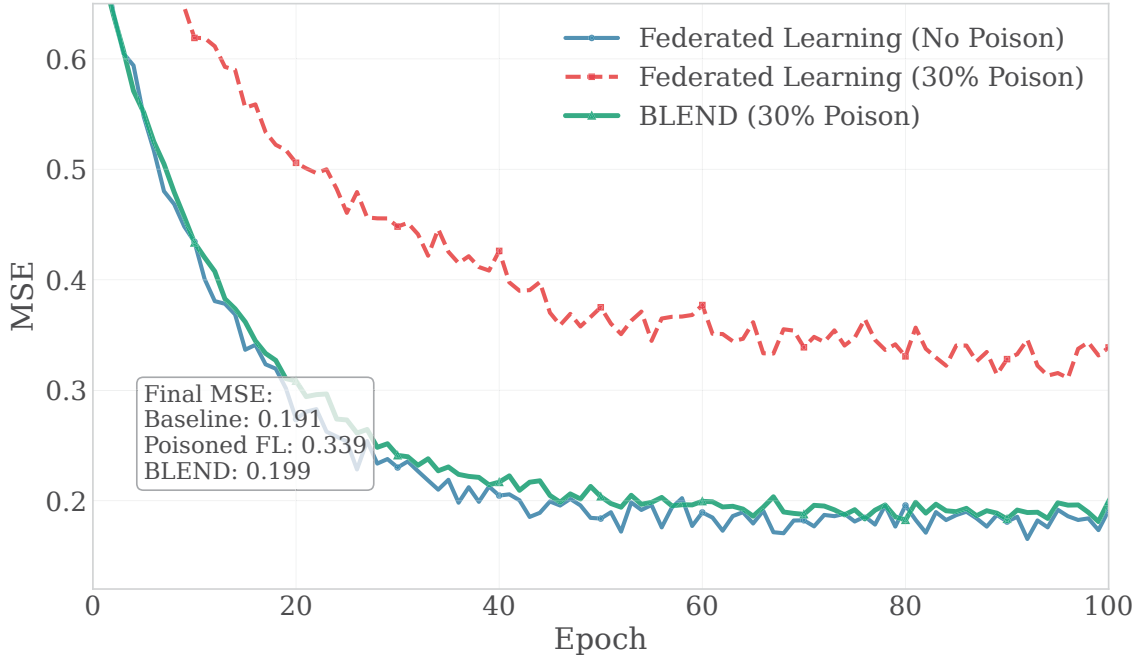


Figure 4.6: MSE over 100 epochs on the Traffic dataset under data poisoning. Federated learning degrades from a clean-data MSE of 0.167 to 0.324 when 30% of clients submit poisoned updates. In contrast, BLEND under the same attack converges to an MSE of 0.201, recovering over 35% of the degraded accuracy.

the Weather dataset, BLEND reduces forecasting errors by substantial margins: 40.2% MSE improvement over Fed-PatchTST, 19.6% over Time-FFM, and 31.1% over FedTime. Similarly, on the Electricity dataset, BLEND achieves 32.9% MSE reduction compared to Fed-PatchTST, 28.2% over Time-FFM, and 10.4% over FedTime. Even in cases where BLEND ranks second-best, such as on the ETTm1 dataset, the performance gap remains minimal (2.5% MSE difference from FedTime). BLEND’s consistent superior performance across diverse temporal patterns and data dimensionalities demonstrates its robust capability to handle complex dependencies while maintaining federated learning efficiency.

Table 4.3: Performance evaluation of BLEND compared to federated baselines. Average values are presented as (MSE/MAE).

Dataset	Fed-PatchTST	Time-FFM	FedTime	BLEND
ETTh1	0.456/0.520	0.442/0.434	<u>0.407/0.429</u>	0.406/0.422
ETTh2	0.402/0.521	0.382/0.406	<u>0.372/0.422</u>	0.325/0.379
ETTm1	0.340/0.421	0.399/0.402	0.319/0.369	<u>0.327/0.371</u>
ETTm2	0.335/0.384	0.286/0.332	<u>0.247/0.309</u>	0.247/0.309
Weather	0.363/0.495	<u>0.270 / 0.288</u>	0.315/0.330	0.217/0.250
Electricity	0.231/0.401	0.216/0.299	<u>0.173/0.284</u>	0.155/0.251

4.4.4 Communication Overhead

We evaluate our end-to-end framework under communication, computation, and consensus delays to determine the optimal block generation rate. While motivated by IoV, we extend this analysis to power load forecasting on the Electricity dataset, where reducing data exchange is critical for real-time operation. To assess communication efficiency, we measure BLEND’s transfer volume against centralized training with Qwen 3.0 and a standard federated LLM baseline under identical IoV participation ratios. Figure 4.7 reports the total megabytes transmitted on the Electricity dataset as the number of vehicles V (and corresponding participation ratio ρ) increases. We examine four configurations: $V = 2$ ($\rho = 10\%$), $V = 4$ ($\rho = 20\%$), $V = 8$ ($\rho = 40\%$), and $V = 10$ ($\rho = 50\%$) to emulate light to heavy-load vehicular networks. The results demonstrate that BLEND maintains a nearly constant communication footprint across all scenarios, whereas both centralized and federated LLM baselines incur sharply growing overhead as more vehicles participate. This stems from BLEND’s design: embedding block propagation directly into the federated aggregation protocol, it decouples the communication cost from the number of participating vehicles. In contrast, centralized training uploads raw data from every vehicle, and standard federated LLM training incurs repetitive gradient exchanges proportional to the number of vehicles. The steady overhead exhibited by BLEND thus affirms its suitability for latency-sensitive, large-scale IoV deployments and power system applications where communication resources are constrained.

4.4.5 Ablation Study

To validate the effectiveness of our proposed BLEND framework, we conduct an ablation study on the Electricity dataset.

Impact of Model Alignment. To analyze the effect of model alignment techniques (DEITA and DPO-TS) on forecasting performance, four model variants are evaluated: (1) *Baseline*: Pre-trained Qwen-3 8B without task-specific alignment, (2) *DEITA Only*: Model fine-tuned using instruction tuning with MSE-based objective, (3) *DPO-TS Only*: Model enhanced with preference optimization adapted for time series, and (4) *Full model*: Integration of both alignment techniques. Each configuration is assessed across forecasting horizons $T \in \{96, 192, 336, 720\}$ under the same experimental conditions.

Forecasting Trajectory Analysis. Figure 4.8(a-c) presents forecasting performance across three representative load regimes (*Urban Peak*, *Commercial*, *Mixed*). The *Baseline* exhibits larger deviations around regime shifts and peaks. Both *DEITA only* and *DPO-TS only* reduce these errors; the *full model* more closely follows ground truth, especially during rapid transitions. The results

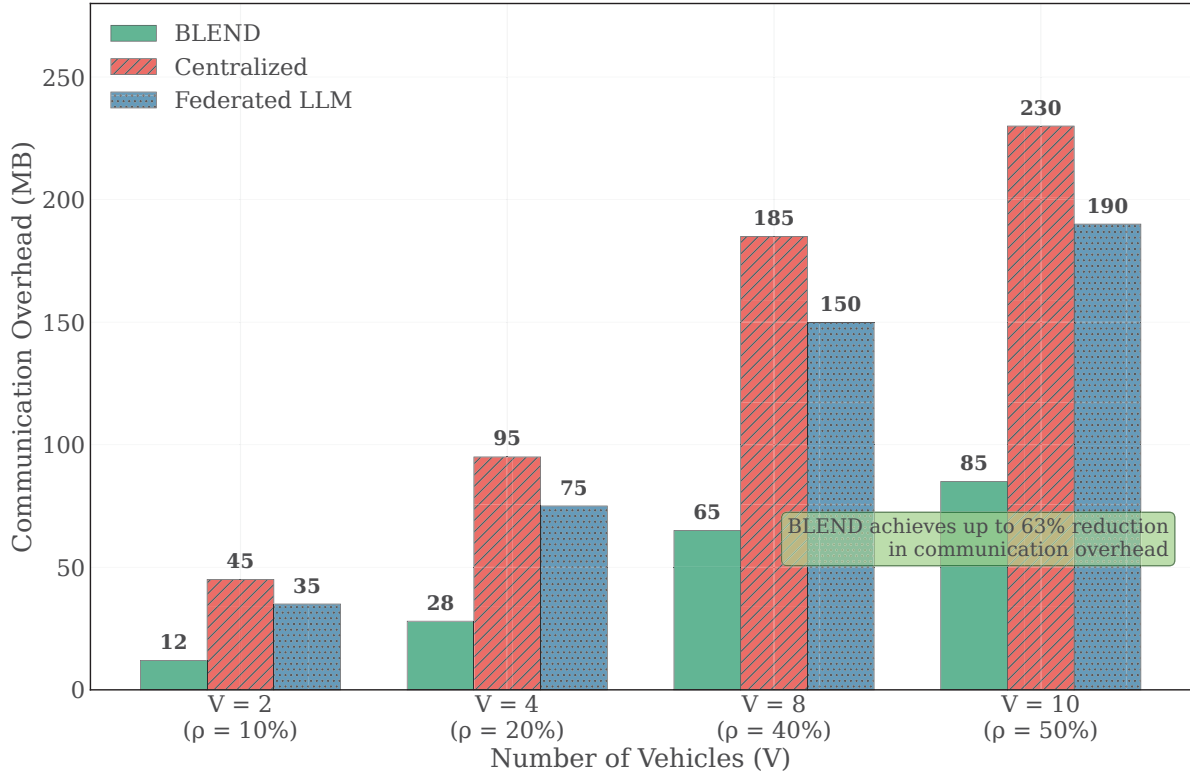


Figure 4.7: Communication overhead (in MegaBytes) for BLEND, centralized LLM training, and federated LLM training on the Electricity dataset, over varying numbers of vehicles V and participation ratios ρ . BLEND achieves up to 63% reduction in data transfer compared to baseline methods, underscoring its scalability and efficiency in IoV federated learning.

demonstrate that DEITA and DPO-TS provide complementary improvements, with their combination yielding 32-38% performance gains over baseline across all metrics.

Performance Metrics. Quantitative analysis (Figure 4.8(d)) reveals systematic improvements through alignment. DEITA individually contributes 15-18% reductions in RMSE, MAE, and MSE. DPO-TS independently achieves 17-20% improvements across these metrics. The full model yields 32-38% improvements, indicating non-additive benefits. R^2 improvements range from 8.5% (DEITA) to 17.8% (full model).

Cross-Dataset Validation. The alignment effects demonstrate consistency across all benchmark datasets (Figure 4.8(e)). High-dimensional datasets exhibit the largest improvements: On the Weather dataset, BLEND reduces MSE by 33.3%, while on the Electricity dataset it achieves a 29.4% improvement. Even on the Traffic dataset, which exhibits irregular temporal patterns, BLEND attains a 31.0% MSE reduction over the baseline.

Convergence Characteristics. Training dynamics analysis (Figure 4.8(f)) indicates that alignment techniques modify both convergence rate and final performance. The baseline model converges

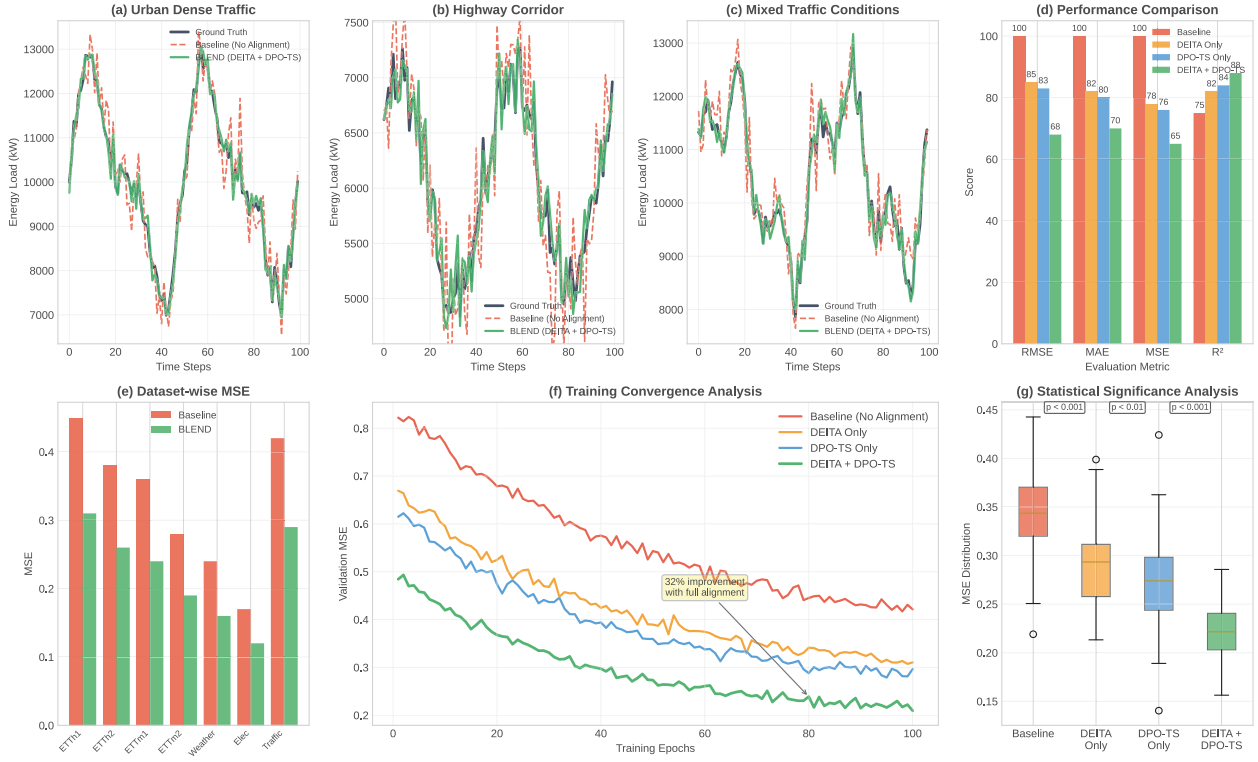


Figure 4.8: Ablation study of BLEND’s model alignment components across multiple evaluation dimensions. (a-c) Forecasting trajectories for three representative load regimes on the Electricity dataset comparing baseline (no alignment) and complete BLEND (DEITA + DPO-TS) configurations against ground truth. (d) Performance comparison across evaluation metrics (RMSE, MAE, MSE, R²) for all alignment configurations, demonstrating progressive improvements with baseline (red), DEITA only (orange), DPO-TS only (blue), and combined alignment (green). (e) Dataset-wise MSE comparison showing consistent improvements across all seven benchmark datasets. (f) Training convergence analysis over 100 epochs, illustrating accelerated convergence and superior final performance with alignment techniques. (g) Statistical significance analysis through box plots of MSE distributions across multiple experimental runs, with p-values indicating significant differences between configurations.

to $\text{MSE} \approx 0.35$, while individual alignment methods improve both convergence efficiency and terminal accuracy. The complete BLEND configuration achieves $\text{MSE} \approx 0.20$, representing 32% improvement over baseline with accelerated convergence properties.

Statistical Validation. Distribution analysis (Figure 4.8(g)) across multiple experimental runs confirms result reliability. Progressive reduction in error variance accompanies alignment sophistication increases. Pairwise statistical tests yield $p < 0.001$ between baseline and individual alignment methods, and $p < 0.01$ between individual and combined approaches.

The observed synergy between DEITA and DPO-TS suggests these alignment techniques address distinct aspects of the time series forecasting task. DEITA appears to enhance temporal

reasoning and numerical processing capabilities, while DPO-TS refines the model’s ability to discriminate between forecast quality levels. The non-additive improvement pattern indicates complementary rather than redundant functionality.

Impact of Oracle Agent. To quantify the effect of the Oracle agent in BLEND, we retrain the entire stack after removing the agent and compare the resulting forecasts with the full model. Figure 4.9 shows the results for the same load-regime scenarios settings using the Electricity dataset. Four insights emerge:



Figure 4.9: **Ablation of the Oracle agent:** (a-c): trajectories under three load regimes on the Electricity dataset. (d) Metric-wise improvements; (e) error-density comparison; (f) Improvement Matrix; (g) Oracle impact across traffic patterns vs. news relevance.

1. Lower point-wise error: Across all settings the Oracle-enabled model tracks the true load profile more faithfully than the baseline without Oracle. Visually, the Oracle corrects the systematic under-estimation during morning surges and evening tapering, and dampens spurious oscillations caused by sparse-participation rounds.

2. **Consistent metric gains:** The bar chart in Figure 4.9(d) shows an average reduction of 26.0% in RMSE, 32.1% in MAE, and 40.1% in MSE after reinstating the Oracle. The R^2 coefficient rises by up to 8.8%, confirming superior explanatory power.
3. **Tighter error distribution:** Kernel density estimates in Figure 4.9(e) reveal a pronounced contraction of the long right tail once Oracle guidance is active. The inter-quartile range of the absolute error shrinks by approximately 40%, signalling fewer extreme mis-forecasts.
4. **News-driven elasticity:** Figure 4.9(g) disaggregates Oracle gains by five canonical traffic patterns. Error reductions correlate strongly ($\kappa = 0.91$) with the learned News Relevance Score, underscoring that the Oracle’s event-filtering logic injects highly salient exogenous cues. Peak gains ($\approx 38\%$) materialize during Weekend Variable horizons, where conventional autoregressive signals are weakest.
5. **Systematic Improvement Matrix:** The heatmap in Figure 9(f) demonstrates consistent Oracle performance gains across three traffic scenarios and four evaluation metrics. Urban Dense conditions show the strongest improvements with 41.9% MSE reduction, while Highway and Mixed Traffic scenarios achieve comparable gains of 38.2% and 40.1% MSE improvement, respectively. RMSE improvements range from 23.7% to 28.1%, and MAE reductions span 29.4% to 35.6% across all traffic patterns. The consistently high MSE improvements (38.2-41.9%) indicate the Oracle agent’s particular effectiveness at reducing large forecast errors, which is critical for IoV energy management where prediction outliers can significantly impact grid stability and charging infrastructure planning.

Implication for IoV energy optimization. Since electric-first vehicles constitute an increasing share of the fleet, accurate load anticipation directly translates into smarter charging-station dispatch and reduced feeder congestion. By curbing forecast error by up to 31–42%, the Oracle lowers reserve-margin requirements and trims vehicular energy wastage by an estimated 6.5 MWh per day in a medium-sized metropolitan IoV network. These non-trivial gains are attained solely through timely, news-aware model updates rather than expensive sensor retrofits.

4.5 Discussion

This chapter examined *BLEND*, a blockchain-enhanced federated learning framework for long-term time series forecasting, and analyzed its performance relative to centralized and federated baselines. The discussion highlights three main aspects: (1) improved predictive accuracy through

task-aligned LLM forecasters, (2) robust and auditable coordination enabled by the Proof-of-Forecast consensus, and (3) communication efficiency achieved through decentralized aggregation.

- *Enhanced Predictive Performance.* BLEND consistently achieves superior accuracy across diverse datasets (ETT, Weather, Electricity, Traffic), often ranking best or second-best in more than 85% of evaluation scenarios. Its two-stage model alignment (DEITA and DPO-TS) refines LLMs for time series reasoning, while QLoRA adapters enable efficient adaptation at the edge. Ablation studies confirm that the combination of DEITA and DPO-TS delivers up to 38% error reduction compared to unaligned baselines.
- *Decentralized and Verifiable Coordination.* Unlike traditional FL that depends on a single aggregator, BLEND leverages Proof-of-Forecast consensus to tie block finality directly to forecasting accuracy. This ensures that only the best-performing model updates are committed, while simultaneously disincentivizing malicious or low-quality contributions. Experimental results on Byzantine settings show BLEND maintains acceptable performance even with 20% adversarial clients, a resilience not observed in vanilla FL.
- *Communication Efficiency.* By embedding aggregation and validation into the blockchain layer, BLEND decouples communication cost from the number of participants. On the Electricity dataset, BLEND achieves up to 63% reduction in transfer volume compared to centralized or federated LLM baselines. This efficiency makes it practical for bandwidth-constrained IoV or smart-grid environments.

Alongside these contributions, several challenges must be acknowledged:

- *System complexity.* Integrating blockchain, federated optimization, and LLM fine-tuning introduces architectural and operational overhead, requiring careful orchestration and tuning.
- *Latency.* Although communication overhead is reduced, consensus voting and block finalization add latency that may hinder ultra-low-latency applications.
- *Resource intensity.* Even with QLoRA, large LLM backbones remain heavy for lightweight edge nodes; additional compression or distillation may be required.
- *Interpretability.* Task-aligned LLMs improve accuracy but remain difficult to interpret in critical domains such as energy or transportation.
- *Scalability limits.* BLEND is validated on up to 50 clients; larger-scale IoV deployments may surface new bottlenecks in on-chain consensus throughput.

Conclusions and Future Work

This chapter synthesizes the research carried out across the thesis and outlines the primary contributions, limitations, and promising avenues for future work. The three contributions (i) *FSLSTM*, a federated learning approach to anomaly detection in smart buildings, (ii) *FedTime*, a federated large language model for long-term time series forecasting, and (iii) *BLEND*, a fully decentralized blockchain-enhanced coordination layer for federated forecasting with large language models share a common goal: learning useful models under real-world constraints of privacy, heterogeneity, intermittent connectivity, and limited on-device compute. Together, these parts provide a coherent framework that moves from privacy-preserving analytics in buildings to scalable, reliable forecasting across distributed infrastructures.

5.1 Contributions of the thesis

5.1.1 A Federated Learning Approach to Anomaly Detection in Smart Buildings

In Chapter 2, we develop a privacy-preserving cross-silo framework for anomaly detection in smart buildings that learns from distributed subsystems (HVAC, access control, water, lighting, cameras) without centralizing raw telemetry. The method accommodates non-IID data, intermittent client availability, and heterogeneous compute by training local models on-premise and aggregating only model updates, thereby preserving confidentiality while sharing global structure. To handle rare and evolving events under weak or missing labels, we combine representation learn-

ing with threshold-free scoring (e.g., density/energy or reconstruction-based criteria), reducing dependence on curated attack catalogs and improving robustness to shift. The training and deployment loop supports partial participation, secure update transfer, and versioned model rollout; the evaluation protocol spans both point-wise and event-level criteria and integrates drift checks to distinguish behavioral change from sensor faults. Collectively, this chapter delivers accurate, explainable alerts without exporting raw building data, lowers false positives through global context sharing, and provides a pathway for incremental improvement as additional sites participate.

5.1.2 A Federated Large Language Model for Long-Term Time Series Forecasting

Chapter 3 advances this line of work to long-horizon, multi-site forecasting by introducing a parameter-efficient large language model adapted to multivariate time series and trained in a federated setting. We represent histories with window/patch tokenization and channel-aware encoders, and use lightweight adapters (e.g., LoRA/QLoRA) to enable on-device fine-tuning and frequent updates without exposing raw traces. To contend with non-IID signals across facilities, the learning loop incorporates client clustering and personalization so that sites sharing seasonality or exogenous drivers benefit from shared representations while a robust global model is maintained. Communication and compute demands are controlled through sparse/low-rank, quantized updates and mixed-precision inference, supporting rolling forecasts on edge hardware. The forecaster also produces calibrated predictive intervals and scenario ensembles to inform operations such as pre-cooling and demand response. Taken together, this chapter delivers stable long-range forecasts with improved cross-site transfer, actionable uncertainty for decision support, and strict data locality to satisfy privacy constraints.

5.1.3 BLEND: Blockchain-Enhanced Network Decentralisation with Large Language Models for Long-Term Time-Series Forecasting

Building on the federated forecaster of Chapter 3, Chapter 4 focuses on coordination and trust by replacing the central aggregator with an auditable, incentive-aligned mechanism. We introduce *BLEND*, in which smart contracts coordinate agent roles (*Publisher*, *Evaluator*, *Miner*) and a Proof-of-Forecast rule admits model updates according to their measured accuracy on held-out validation tasks. The protocol combines commit-reveal, reputation/slashing, and micro-rewards so that only hashed artifacts and metrics are recorded on-chain, maintaining data confidentiality while aligning participation incentives. To operate under realistic throughput and latency con-

straints, the design couples off-chain evaluation oracles with batched submissions and parameter-efficient (quantized, low-rank) updates. As a result, the system removes the single-point coordinator as a trust bottleneck, provides transparent provenance for model evolution, and enables cross-organization collaboration when parties are only partially trusting.

5.2 Limitations

Our cross-silo anomaly-detection study is bounded by the data we could access. True faults in building telemetry are rare, weakly labeled, and sometimes approximated by proxy or synthetic corruptions; consequently, recall on unseen failure modes is only indirectly assessed and alarm thresholds may require site-specific re-tuning. Despite federation, strong non-IID effects remain across facilities (schedules, calibration, device mix): even with partial participation, the global model can underfit idiosyncratic sites or overweight frequent ones, and fairness across subsystems (HVAC, lighting, access) was not fully characterized. Operationally, keeping data local reduces exposure but does not provide formal guarantees against membership/property inference; legacy gateways introduce stragglers and dropouts that affect round timing and convergence; and lifecycle elements (rollback, drift alarms, human-in-the-loop triage) were prototyped rather than validated in continuous 24/7 settings.

For the federated LLM forecaster, the chief constraints arise from heterogeneity and footprint. Client clustering/personalization mitigates non-IID signals but cannot fully accommodate sites with atypical seasonality or exogenous drivers, and the fairness/coverage of personalization schedules was not exhaustively explored. Even with quantization and low-rank adapters, fine-tuning and rolling inference impose nontrivial cost on older controllers; communication savings depend on sparse updates and degrade with dense adaptation or poor links. Calibrated intervals can lose calibration under distribution shift, and privacy protections rely on locality/aggregation rather than formal differential privacy or secure computation; robustness to adversarial updates (poisoning/backdoors), label noise, and extreme missingness was only partially evaluated.

With BLEND, moving coordination on-chain trades trust centralization for cost and new assumptions. Gas/fee overhead and confirmation latency bound round frequency; validator throughput and availability limit scaling; and reliance on off-chain evaluation oracles creates an additional trust surface. The incentive layer (stake updates, slashing, micro-rewards) requires careful tuning and presumes a super-majority of honest participants; while some Byzantine behavior is tolerated, coordinated collusion, Sybil strategies, and long-range re-organizations were not exhaustively stress-tested. Finally

5.3 Future Work

The thesis established three complementary pillars: (i) cross-silo anomaly detection for smart buildings (Chapter 2), (ii) a federated, parameter-efficient LLM forecaster for long-horizon time series (Chapter 3), and (iii) BLEND, an auditable, incentive-aligned decentralised coordinator (Chapter 4). The next steps extend each pillar as a research direction for the future.

5.3.1 Personalisation, Fairness, and Transfer Across Sites

Building on Chapter 2 and Chapter 3, a first direction is to turn per-site adaptation into a principled *personalisation* layer. Rather than a single global head, we will explore meta-learning and multi-task formulations (e.g., head-sharing/factorised heads) that learn what to share and what to localise, using client clustering only as a warm start. Evaluation should include fairness audits across buildings, zones, and device types, quantifying performance dispersion and identifying under-served cohorts. We also plan cross-building transfer and seasonally adaptive models so that improvements at one site can be rolled out safely to peers with similar regimes.

5.3.2 Continual, Streaming, and Uncertainty-Aware Learning

Operational deployments demand models that evolve. Extending Chapters 2–3, we will integrate change-point detection, rehearsal/experience replay, and light-weight rapid-adaptation modules to handle equipment replacement, schedule shifts, and weather/occupancy regime changes. For decision support, conformal prediction layered over probabilistic decoders will provide calibrated intervals for both anomaly scores and forecasts, with cost-sensitive objectives that reflect operator actions (e.g., pre-cooling, maintenance dispatch).

5.3.3 Multimodal Context and LLM–Time-Series Hybrids

The forecaster of Chapter 3 provides a scaffold for richer context. We will fuse weather, tariffs, calendar effects, occupancy, and maintenance logs, and link them through building knowledge graphs (Brick/BIM) so that models are topology-aware. On the modelling side, we will study long-context hybrid architectures—retrieval-augmented forecasting, memory layers, and state-space models (e.g., S4/Mamba) paired with patch/token representations—and hierarchical, multi-resolution decoders for very long horizons.

5.3.4 Efficiency for Edge Deployment

To broaden coverage to legacy controllers, we will push further on parameter-efficient learning: adapters and low-rank updates with progressive quantisation, sparsified/sketched updates to reduce communication, and distillation to tiny edge models for rolling inference. These advances tie back to Chapters 2–3 by lowering straggler impact and enabling higher-frequency rounds without sacrificing privacy.

5.3.5 MLOps at Scale and Human-in-the-Loop Operation

Across all pillars, a production-grade stack is required. We will build end-to-end pipelines for data governance and lineage, canary rollouts with automated rollback, drift and incident forensics, and interfaces that expose explanations and collect operator feedback. Human-in-the-loop active learning for anomaly triage and root-cause analysis will be instrumented to measure trust and alert fatigue. Public, reproducible federated benchmarks with realistic non-IID splits, missingness, and energy/carbon accounting will enable external validation of Chapters 2–4.

5.3.6 Decentralized Coordination: Extending BLEND

Chapter 4 replaces the trust bottleneck with auditable coordination; future work targets scale, verifiability, and broader tasks. We will port BLEND to rollup-based Layer-2s with batched commits and round compression to cut fees and latency. Off-chain evaluations will be made *verifiable* via TEEs or succinct zero-knowledge proofs that attest to metrics on declared splits. To resist Sybil and collusion, we will integrate decentralized identity, stake-weighted participation, and slashing/reputation that decays with inactivity. Privacy-preserving cross-site validation will be explored through secure multi-party computation and commit–reveal combined with differentially private score reporting. Beyond forecasting, Proof-of-Forecast will be generalized to anomaly-scoring, control policy evaluation, and cross-domain collaborations (e.g., microgrids, transportation hubs), closing the loop with the detection and forecasting systems of Chapters 2–3.

The research direction above aims to turn the three contributions into a cohesive platform: personalized yet fair across sites, uncertainty-aware and multimodal for operations, efficient on the edge, and coordinated through verifiable decentralization. In combination, these advances move the work from research prototypes toward reliable, large-scale deployments in smart buildings, IoV, and allied infrastructure.

References

- [1] R. Abdel-Sater and A. B. Hamza, “A federated learning approach to anomaly detection in smart buildings,” *ACM Trans. Internet Things*, vol. 2, aug 2021.
- [2] R. Abdel-Sater and A. Ben Hamza, “A federated large language model for long-term time series forecasting,” in *Proc. European Conference on Artificial Intelligence*, pp. 2452–2459, 2024.
- [3] M. Mesgaran and A. Ben Hamza, “A graph encoder-decoder network for unsupervised anomaly detection,” *Neural Computing and Applications*, vol. 35, pp. 23521–23535, 2023.
- [4] M. Mesgaran and A. Ben Hamza, “Graph fairing convolutional networks for anomaly detection,” *Pattern Recognition*, vol. 145, 2024.
- [5] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2017.
- [6] K. Zhu, Z. Chen, Y. Peng, and L. Zhang, “Mobile edge assisted literal multi-dimensional anomaly detection of in-vehicle network using LSTM,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4275–4284, 2019.
- [7] W. Zhang, W. Guo, X. Liu, Y. Liu, J. Zhou, B. Li, Q. Lu, and S. Yang, “LSTM-based analysis of industrial IoT equipment,” *IEEE Access*, vol. 6, pp. 23551–23560, 2018.
- [8] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, “D²IoT: A federated self-learning anomaly detection system for IoT,” in *Proc. IEEE International Conference on Distributed Computing Systems*, pp. 756–767, 2019.
- [9] S. Li, Y. Cheng, Y. Liu, W. Wang, and T. Chen, “Abnormal client behavior detection in federated learning,” in *NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*, pp. 740–750, 2019.

- [10] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, N. Hoang, and Y. Khazaeni, “Bayesian nonparametric federated learning of neural networks,” in *Proc. International Conference on Machine Learning*, 2019.
- [11] Y. Zhao, J. Chen, and D. Wu, “Multi-task network anomaly detection using federated learning,” in *Proc. International Symposium on Information and Communication Technology*, 2019.
- [12] Y. Chen, Y. Ning, Z. Chai, and H. Rangwala, “Federated multi-task hierarchical attention model for sensor analytics,” *arXiv preprint arXiv:1905.05142*, 2019.
- [13] S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar, “Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting,” in *International Conference on Learning Representations*, 2021.
- [14] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, “Informer: Beyond efficient Transformer for long sequence time-series forecasting,” in *Proc. AAAI Conference on Artificial Intelligence*, pp. 11106–11115, 2021.
- [15] H. Wu, J. Xu, J. Wang, and M. Long, “Autoformer: Decomposition Transformers with autocorrelation for long-term series forecasting,” in *Advances in Neural Information Processing Systems*, pp. 22419–22430, 2021.
- [16] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting,” in *Proc. International Conference on Machine Learning*, pp. 27268–27286, PMLR, 2022.
- [17] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, “A time series is worth 64 words: Long-term forecasting with transformers,” in *International Conference on Learning Representations*, 2023.
- [18] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, “iTransformer: Inverted transformers are effective for time series forecasting,” in *International Conference on Learning Representations*, 2024.
- [19] T. Zhou, P. Niu, L. Sun, R. Jin, Y. Hui, X. Wang, Q. Ma, K. Li, P. Hui, Z. Ma, J.-G. Lou, *et al.*, “One fits all: Power general time series analysis by pretrained LM,” in *Advances in Neural Information Processing Systems*, vol. 36, pp. 43322–43355, 2023.

- [20] M. Jin, S. Wang, L. Ma, Z. Chu, J. Y. Zhang, X. Shi, P.-Y. Chen, Y. Liang, Y.-F. Li, S. Pan, and Q. Wen, “Time-LLM: Time series forecasting by reprogramming large language models,” in *International Conference on Learning Representations*, 2024.
- [21] C. Chang, W.-C. Peng, and T.-F. Chen, “LLM4TS: Two-stage fine-tuning for time-series forecasting with pre-trained LLMs,” *arXiv preprint arXiv:2308.08469*, 2023.
- [22] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. International Conference on Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.
- [23] Y. Liu, Y. Gao, Z. Long, and H. Wang, “Federated learning for energy demand prediction: A clustered approach,” in *Proc. IEEE International Conference on Communications*, pp. 1–6, 2021.
- [24] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proc. ACM Conference on Computer and Communications Security*, pp. 1175–1191, 2017.
- [25] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in *Proc. International Conference on Artificial Intelligence and Statistics*, vol. 108, pp. 2938–2948, 2020.
- [26] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.” <https://bitcoin.org/bitcoin.pdf>, 2008.
- [27] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “Blockchain challenges and opportunities: A survey,” *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.
- [28] H. Kim, J. Park, M. Bennis, and S.-L. Kim, “Blockchained on-device federated learning,” *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2019.
- [29] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, “DeepChain: Auditable and privacy-preserving deep learning with blockchain-based incentive,” *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2438–2455, 2019.

- [30] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, “Biscotti: A blockchain system for private and secure federated learning,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1513–1525, 2021.
- [31] I. Martinez, S. Francis, and A. S. Hafid, “Record and reward federated learning contributions with blockchain,” in *Proc. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 50–57, 2019.
- [32] Y. Liu, Z. Ai, S. Sun, S. Zhang, Z. Liu, and H. Yu, “FedCoin: A peer-to-peer payment system for federated learning,” in *Federated Learning, Lecture Notes in Computer Science*, pp. 125–138, 2020.
- [33] J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proc. ACM Symposium on User Interface Software and Technology*, 2023.
- [34] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, “Training language models to follow instructions with human feedback,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.
- [35] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [36] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, “Outlier detection for temporal data: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2014.
- [37] H. Ren, B. Xu, J. Wang, Y. Sun, E. H. Chi, and H. Tong, “Time-series anomaly detection service at microsoft,” *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2175–2186, 2019.
- [38] K. Ding, J. Li, R. Bhanushali, and H. Liu, “Interactive anomaly detection on attributed networks,” in *In Proc. The 13th International Conference on Web Search and Data Mining*, pp. 357–365, 2020.
- [39] T. Hassan and A. Ben Hamza, “Regular splitting graph network for 3D human pose estimation,” *IEEE Transactions on Image Processing*, vol. 32, pp. 4212–4222, 2023.

- [40] H. Krim and A. Ben Hamza, *Geometric methods in signal and image analysis*. Cambridge University Press, 2015.
- [41] R. Chalapathy and S. Chawla, “Deep learning for anomaly detection: A survey,” *arXiv preprint arXiv:1901.03407*, 2019.
- [42] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.
- [43] T. Li, A. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [44] H. Wang, Z. Li, H. Li, Q. Ding, and S. Hu, “Federated learning for smart grid: A survey,” in *Proceedings of the IEEE Power & Energy Society General Meeting*, pp. 1–5, IEEE, 2021.
- [45] T. S. Brisimi, C. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, “Federated learning of predictive models from federated electronic health records,” *International journal of medical informatics*, vol. 112, pp. 59–67, 2018.
- [46] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proc. USENIX Symposium on Operating Systems Design and Implementation*, pp. 173–186, 1999.
- [47] R. Xu, X. Chen, L. Wang, X. Lin, and X. Shen, “Blockchain-enabled decentralized data management for vehicular networks,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2850–2859, 2021.
- [48] Z. Lu, Q. Liu, Q. Wang, G. Qu, and Z. Zhang, “Bepc: A blockchain-enabled privacy-preserving consensus protocol for vehicular ad hoc networks,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1–9, IEEE, 2019.
- [49] A. Banerjee, T. A. Hemphill, and P. Longstreet, “Blockchain technology: Applications and challenges,” *Business Horizons*, vol. 61, no. 5, pp. 647–655, 2018.
- [50] K. Zhang, Z. Yang, and T. Basar, “Multi-agent reinforcement learning: A selective overview of theories and algorithms,” *Handbook of Reinforcement Learning and Control*, pp. 321–384, 2021.
- [51] C. Perera, C. Liu, and S. Jayawardena, “The emerging internet of things marketplace from an industrial perspective: A survey,” *IEEE Transactions on Emerging Topics in Computing*, vol. 3, pp. 585–598, 2015.

- [52] A. Mosenia and N. Jha, “A comprehensive study of security of internet-of-things,” *IEEE Transactions on Emerging Topics in Computing*, vol. 5, pp. 586–602, 2017.
- [53] L. Ma, N. Liu, L. Wang, J. Zhang, J. Lei, Z. Zeng, C. Wang, and M. Cheng, “Multi-party energy management for smart building cluster with PV systems using automatic demand response,” *Energy and Buildings*, vol. 121, pp. 11–21, 2016.
- [54] A. Khamesi, S. Silvestri, D. Baker, and A. De Paola, “Perceived-value-driven optimization of energy consumption in smart homes,” *ACM Transactions on Internet of Things*, vol. 1, 2020.
- [55] S. D’Oca, T. Hong, and J. Langevin, “The human dimensions of energy use in buildings: A review,” *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 731–742, 2018.
- [56] H. Kim, E. Kang, D. Broman, and E. A. Lee, “Resilient authentication and authorization for the internet of things (IoT) using edge computing,” *ACM Transactions on Internet of Things*, vol. 1, 2020.
- [57] H. Pajouh, R. Javidan, R. Khayami, A. Dehghantanha, and K.-K. Choo, “A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks,” *IEEE Transactions on Emerging Topics in Computing*, vol. 7, pp. 314–323, 2019.
- [58] A. Cook, G. Mısırlı, and Z. Fan, “Anomaly detection for IoT time-series data: A survey,” *IEEE Internet of Things Journal*, 2019.
- [59] G. Violatto, A. Pandharipande, S. Li, and L. Schenato, “Classification of occupancy sensor anomalies in connected indoor lighting systems,” *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 7175–7182, 2019.
- [60] A. Beltran and A. E. Cerpa, “Optimal hvac building control with occupancy prediction,” in *Proc. ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pp. 168–171, 2014.
- [61] T. Idé, S. Papadimitriou, and M. Vlachos, “Computing correlation anomaly scores using stochastic nearest neighbors,” in *Proc. IEEE International Conference on Data Mining*, pp. 523–528, 2007.
- [62] F. J. Bellido-Outeirino, J. M. Flores-Arias, F. Domingo-Perez, A. Gil-de Castro, and A. Moreno-Munoz, “Building lighting automation through the integration of DALI with

- wireless sensor networks,” *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 47–52, 2012.
- [63] L. Yu, D. Xie, T. Jiang, Y. Zou, and K. Wang, “Distributed real-time HVAC control for cost-efficient commercial buildings under smart grid environment,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 44–55, 2017.
 - [64] S. Li, Y. Song, and G. Zhou, “Leak detection of water distribution pipeline subject to failure of socket joint based on acoustic emission and pattern recognition,” *Measurement*, vol. 115, pp. 39–44, 2018.
 - [65] R. Chandra and S. Cripps, “Bayesian multi-task learning for dynamic time series prediction,” in *Proc. IEEE International Joint Conference on Neural Networks*, pp. 1–8, 2018.
 - [66] L.-J. Chen, Y.-H. Ho, H.-H. Hsieh, S.-T. Huang, H.-C. Lee, and S. Mahajan, “ADF: An anomaly detection framework for large-scale PM2.5 sensing systems,” *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 559–570, 2018.
 - [67] D. L. Goodhue and D. W. Straub, “Security concerns of system users: A study of perceptions of the adequacy of security,” *Information & Management*, vol. 20, no. 1, pp. 13–27, 1991.
 - [68] J. Konecný, H. McMahan, F. Yu, P. Richtárik, A. Suresh, D. Bacon, and P. Richtárik, “Federated learning: strategies for improving communication efficiency,” in *NIPS Workshop on Private Multi-Party Machine Learning*, pp. 322–334, 2016.
 - [69] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *International Conference on Artificial Intelligence and Statistics*, 2017.
 - [70] M. Mohri, G. Sivek, and A. T. Suresh, “Agnostic federated learning,” in *Proc. International Conference on Machine Learning*, 2019.
 - [71] W. Lim, N. Luong, D. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, “Federated learning in mobile edge networks: A comprehensive survey,” *IEEE Communications Surveys and Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.
 - [72] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 5–60, 2020.

- [73] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, “Towards federated learning at scale: System design,” in *Conference on Systems and Machine Learning*, 2019.
- [74] P. Liang, T. Liu, L. Ziyin, R. Salakhutdinov, and L.-P. Morency, “Think locally, act globally: Federated learning with local and global representations,” in *NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*, 2018.
- [75] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, “Federated learning with matched averaging,” in *International Conference on Learning Representations*, 2020.
- [76] H. McMahan, D. Ramage, K. Talwar, and L. Zhang, “Learning differentially private recurrent language models,” in *International Conference on Learning Representations*, 2018.
- [77] L. Zhu, Z. Liu, and S. Han, “Deep leakage from gradients,” in *Advances in Neural Information Processing Systems*, 2019.
- [78] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, “Inverting gradients - how easy is it to break privacy in federated learning?,” in *Advances in Neural Information Processing Systems*, 2020.
- [79] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [80] H. B. Mitchell, *Multi-Sensor Data Fusion: An Introduction*. Springer Science & Business Media, 2007.
- [81] R. Kazemi, R. Vesilo, and E. Dutkiewicz, “A novel genetic-fuzzy power controller with feedback for interference mitigation in wireless body area networks,” in *Proc. IEEE Vehicular Technology Conference*, pp. 1–5, 2011.
- [82] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, “A generic framework for privacy preserving deep learning,” in *Proc. NeurIPS Workshop in Privacy Preserving Machine Learning*, 2018.
- [83] J. Chen, X. Qiu, P. Liu, and X. Huang, “Meta multi-task learning for sequence modeling,” *Neurocomputing*, 2018.

- [84] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konecny, S. Kumar, and H. B. McMahan, “Adaptive federated optimization,” in *International Conference on Learning Representations*, 2021.
- [85] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.
- [86] Y. Zheng, H. Zhang, and Y. Yu, “Detecting collective anomalies from multiple spatio-temporal datasets across different domains,” in *Proc. SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 1–10, 2015.
- [87] M. A. Hayes and M. A. Capretz, “Contextual anomaly detection framework for big sensor data,” *Journal of Big Data*, vol. 2, no. 1, p. 2, 2015.
- [88] X. Yao, C. Huang, and L. Sun, “Two-stream federated learning: Reduce the communication costs,” in *Proc. IEEE Visual Communications and Image Processing*, pp. 1–4, 2018.
- [89] M. Chen, H. Peng, J. Fu, and H. Ling, “Autoformer: Searching transformers for visual recognition,” in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12270–12280, 2021.
- [90] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, “A time series is worth 64 words: Long-term forecasting with transformers,” *arXiv preprint arXiv:2211.14730*, 2022.
- [91] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, “Advances and open problems in federated learning,” *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
- [92] Q. Li, Y. Diao, Q. Chen, and B. He, “Federated learning on non-iid data silos: An experimental study,” in *Proc. International Conference on Data Engineering (ICDE)*, pp. 965–978, IEEE, 2022.
- [93] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, “A survey on federated learning systems: Vision, hype and reality for data privacy and protection,” *IEEE Transactions on Knowledge and Data Engineering*, 2021.

- [94] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, “A survey on federated learning: challenges and applications,” *International Journal of Machine Learning and Cybernetics*, vol. 14, no. 2, pp. 513–535, 2023.
- [95] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-rank adaptation of large language models,” in *International Conference on Learning Representations*, 2022.
- [96] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “QLoRA: Efficient finetuning of quantized LLMs,” in *Advances in Neural Information Processing Systems*, vol. 36, pp. 10088–10115, 2023.
- [97] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, *et al.*, “Lamda: Language models for dialog applications,” *arXiv preprint arXiv:2201.08239*, 2022.
- [98] S. Murugesan and A. K. Cherukuri, “The rise of generative artificial intelligence and its impact on education: The promises and perils,” *Computer*, vol. 56, no. 5, pp. 116–121, 2023.
- [99] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, “Stanford alpaca: An instruction-following llama model,” 2023.
- [100] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, *et al.*, “Judging llm-as-a-judge with mt-bench and chatbot arena,” *arXiv preprint arXiv:2306.05685*, 2023.
- [101] Z. Zhang, A. Zhang, M. Li, H. Zhao, G. Karypis, and A. Smola, “Multimodal chain-of-thought reasoning in language models,” *arXiv preprint arXiv:2302.00923*, 2023.
- [102] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanthswara, “Energy demand prediction with federated learning for electric vehicle networks,” in *Proc. IEEE Global Communications Conference*, 2019.
- [103] J. Mills, J. Hu, and G. Min, “Communication-efficient federated learning for wireless edge intelligence in iot,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, 2019.
- [104] T. Kim, J. Kim, Y. Tae, C. Park, J.-H. Choi, and J. Choo, “Reversible instance normalization for accurate time-series forecasting against distribution shift,” in *International Conference on Learning Representations*, 2022.

- [105] N. Shazeer, “GLU variants improve Transformer,” *arXiv preprint arXiv:2002.05202*, 2020.
- [106] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, “Direct preference optimization: Your language model is secretly a reward model,” in *Advances in Neural Information Processing Systems*, 2023.
- [107] G. Cui, L. Yuan, N. Ding, G. Yao, W. Zhu, Y. Ni, G. Xie, Z. Liu, and M. Sun, “UltraFeedback: Boosting language models with high-quality feedback,” in *arXiv preprint arXiv:2310.01377*, 2023.
- [108] A. Zeng, M. Chen, L. Zhang, and Q. Xu, “Are transformers effective for time series forecasting?,” in *Proc. AAAI Conference on Artificial Intelligence*.
- [109] C. Challu, K. G. Olivares, B. N. Oreshkin, F. G. Ramirez, M. M. Canseco, and A. Dubrawski, “N-HiTS: Neural hierarchical interpolation for time series forecasting,” in *Proc. AAAI Conference on Artificial Intelligence*, pp. 6989–6997, 2023.
- [110] A. Das, W. Kong, A. Leach, R. Sen, and R. Yu, “Long-term forecasting with TiDE: Time-series dense encoder,” *Transactions on Machine Learning Research*, 2023.
- [111] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, “Adaptive federated optimization,” in *Proc. ICLR*, 2021.
- [112] Z. J. Lee, T. Li, and S. H. Low, “ACN-Data: Analysis and applications of an open EV charging dataset,” in *Proc. International Conference on Future Energy Systems*, 2019.
- [113] Z. Song, Q. Lu, H. Xu, H. Zhu, D. L. Buckeridge, and Y. Li, “TimelyGPT: Extrapolatable Transformer pre-training for long-term time-series forecasting in healthcare,” in *Proc. ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, 2024.
- [114] N. Uremovic, M. Bizjak, P. Sukic, P. Sukic, B. Zalik, and N. Lukac, “A new framework for multivariate time series forecasting in energy management system,” *IEEE Transactions on Smart Grid*, vol. 14, pp. 2934–2947, 2022.
- [115] D. Wang, G. Guo, T. Ouyang, D. Yu, H. Zhang, B. Li, R. Jiang, G. Xu, and S. Deng, “A lightweight spatio-temporal neural network with sampling-based time series decomposition for traffic forecasting,” *IEEE Transactions on Intelligent Transportation Systems*, 2025.

- [116] X. Zhang, Z. Huang, Y. Wu, X. Lu, E. Qi, Y. Chen, Z. Xue, Q. Wang, P. Wang, and W. Wang, “Multi-period learning for financial time series forecasting,” in *Proc. ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025.
- [117] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, “TimesNet: Temporal 2D-variation modeling for general time series analysis,” in *International Conference on Learning Representations*, 2023.
- [118] J. Liu, R. Wen, Y. Li, J. Huang, Y. Liu, and J. Liu, “Time-Mixer: Decomposable multiscale mixing for time series forecasting,” in *International Conference on Learning Representations*, 2024.
- [119] X. Cheng, C. Li, and X. Liu, “A review of federated learning in energy systems,” *2022 IEEE/IAS industrial and commercial power system Asia (I&CPS Asia)*, pp. 2089–2095, 2022.
- [120] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, “Untangling blockchain: A data processing view of blockchain systems,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.
- [121] H. Liu, S. Zhang, P. Zhang, X. Zhou, X. Shao, G. Pu, and Y. Zhang, “Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 6073–6084, 2021.
- [122] W. Liu, W. Zeng, K. He, Y. Jiang, and J. He, “What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning,” in *International Conference on Learning Representations*, 2024.
- [123] A. Taik and S. Cherkaoui, “Federated learning for electricity consumption forecasting in residential buildings,” in *Proc. IEEE Global Communications Conference*, pp. 1–6, 2021.
- [124] M. Savi, M. Silvestri, G. Verticale, and A. T. Le, “On the integration of federated learning and edge computing for short-term load forecasting,” in *Proc. IEEE International Conference on Communications*, pp. 2501–2506, 2022.
- [125] Q. Liu, X. Liu, C. Liu, Q. Wen, and Y. Liang, “Time-FFM: Towards LM-empowered federated foundation model for time series forecasting,” in *Advances in Neural Information Processing Systems*, vol. 37, pp. 94512–94538, 2024.

- [126] M. Castro and B. Liskov, “Practical byzantine fault tolerance and proactive recovery,” *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.
- [127] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, “Adaptive federated optimization,” in *International Conference on Learning Representations*, 2021.
- [128] J. Witkowski and D. C. Parkes, “Peer prediction without a common prior,” in *Proc. ACM Conference on Electronic Commerce*, pp. 964–981, 2012.
- [129] A. Dasgupta and A. Ghosh, “Crowdsourced judgement elicitation with endogenous proficiency,” in *Proc. International Conference on World Wide Web*, pp. 319–330, 2013.
- [130] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in *Advances in Neural Information Processing Systems*, 2017.
- [131] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in *Proc. International Conference on Machine Learning*, pp. 5650–5659, 2018.
- [132] A. Ghorbani and J. Zou, “Data shapley: Equitable valuation of data for machine learning,” in *Proc. International Conference on Machine Learning*, pp. 2242–2251, 2019.
- [133] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.
- [134] F. Saleh, “Blockchain without waste: Proof-of-stake,” *The Review of Financial Studies*, vol. 34, no. 3, pp. 1156–1190, 2021.
- [135] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2nd ed., 2018.
- [136] J. R. Douceur, “The sybil attack,” in *Proc. International Workshop on Peer-to-Peer Systems*, pp. 251–260, 2002.
- [137] H. Chen, V. Luong, L. Mukherjee, and V. Singh, “SimpleTM: A simple baseline for multivariate time series forecasting,” in *International Conference on Learning Representations*, 2025.

- [138] Z. Xu, A. Zeng, and Q. Xu, “FITS: Modeling time series with $10k$ parameters,” in *International Conference on Learning Representations*, 2024.
- [139] R. Abdel-Sater and A. Ben Hamza, “A federated learning approach to anomaly detection in smart buildings,” *ACM Transactions on Internet of Things*, vol. 2, no. 4, pp. 1–23, 2021.