Wireless Networks
https://doi.org/10.1007/s11276-026-04224-1

ORIGINAL PAPER

Optimizing resource allocation for federated LLM training via workload forecasting and autoscaling in edge environments

Bablu Kumar[1] · **Anshul Verma**[1,2] · **Rajkumar Buyya**[2]

Received: 18 September 2025 / Accepted: 28 August 2026

Abstract

The growing deployment of applications in real-world edge environments is driving an increasing demand for large language model (LLM) training in distributed systems. This demand creates significant challenges for resource management, particularly in resource-constrained edge cloud environments. Federated learning (FL) enables privacy-preserving LLM fine-tuning across multiple clients, but it also introduces dynamic computational and communication loads that can strain system resources. To address these challenges, we propose an integrated framework that combines LLM fine-tuning using federated adaptive local low-rank adaptation (FedALoRA) with forecasting-driven autoscaling policies. Differential privacy (DP) is incorporated to ensure secure aggregation, balancing privacy guarantees with system performance. Using the WikiText-2 dataset for FL-based LLM training and the Bitbrains dataset for CPU usage forecasting, we evaluate deep learning models (LSTM, GRU, BiLSTM, CNN, CNN-LSTM, Transformer). Results demonstrate that FedALoRA enables fast federated convergence, reducing perplexity from 199.3 to 85.2 without DP and from 211.6 to 92.2 with DP, incurring only a 7–13% privacy overhead. Among forecasting models, CNN-LSTM achieves the lowest MSE (0.25) and consistently requires the fewest pods (as low as 11 replicas), highlighting its superior accuracy and resource efficiency for proactive autoscaling. Comparative analysis demonstrates that proactive autoscaling consistently outperforms reactive autoscaling, with non-DP settings achieving slightly faster adaptation and DP settings offering stronger privacy guarantees. Overall, the proposed framework balances performance, adaptability, and privacy, making it well-suited for federated-LLM Autoscaling in dynamic edge environments.

Keywords Federated learning · Large language models · FedALoRA · Differential privacy · Proactive autoscaling · Cloud-edge environments

1 Introduction

The rapid growth of intelligent and connected devices has accelerated the adoption of edge computing, which processes data closer to end-users to reduce latency and improve responsiveness [1–6]. While edge computing offers substantial benefits, it also introduces challenges such as workload heterogeneity, dynamic resource fluctuations, and stringent privacy requirements [7–9]. These challenges become particularly critical when scaling computationally intensive applications such as large language models (LLMs) in real-time, resource-constrained edge environments [10].

Containerization and orchestration frameworks such as Docker and Kubernetes have become the standard tools for managing distributed cloud–edge applications [11, 12]. In Kubernetes, applications are deployed as containers

✉ Anshul Verma
anshul.verma@unimelb.edu.au; anshul.verma@bhu.ac.in

Bablu Kumar
bablu.kumar@bhu.ac.in

Rajkumar Buyya
rbuyya@unimelb.edu.au

[1] Department of Computer Science, Banaras Hindu University, Varanasi, Uttar Pradesh 221005, India

[2] Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC 3053, Australia

grouped into pods, and autoscaling mechanisms dynamically adjust the number of pods based on system demand [13]. Autoscaling strategies are broadly classified into reactive approaches, which respond to observed resource usage, and proactive approaches, which rely on predictive models to anticipate future demand [14–16]. Prior studies have shown that proactive autoscaling can significantly outperform reactive methods, particularly in environments with highly variable workloads, by preventing underprovisioning and overprovisioning before performance degradation occurs [17–20].

At the same time, training LLMs in distributed edge environments imposes substantial computational and communication overhead [21, 22]. Federated learning (FL) has emerged as a promising paradigm to enable collaborative model training across multiple clients without sharing raw data, thereby preserving privacy and reducing data transfer costs [23, 24]. However, FL introduces highly dynamic workloads due to non-independent and identically distributed (non-IID) data, varying convergence rates, and communication bursts across training rounds. These challenges are further amplified when privacy-preserving mechanisms such as differential privacy (DP) and secure aggregation are applied, as they increase computational overhead and slow convergence [25–27].

To mitigate these issues, parameter-efficient fine-tuning (PEFT) techniques have been proposed to reduce the cost of adapting LLMs. Methods such as LoRA, ALoRA, LLRA, and QLoRA limit training to low-rank adapter parameters instead of updating the full model [28]. While these approaches significantly reduce memory and computation requirements, they do not address the system-level challenge of managing the dynamic resource demands generated during federated LLM training. Recently, FedALoRA [29] has emerged as an effective solution that integrates adaptive low-rank fine-tuning with federated optimization, reducing communication overhead while maintaining model accuracy and personalization [30].

Despite advances in both federated learning and autoscaling, existing studies largely treat these two domains independently. Federated learning research focuses primarily on convergence, privacy, and communication efficiency, while autoscaling research targets generic cloud workloads without considering the unique and highly dynamic behavior of federated LLM training. As a result, current approaches lack a unified framework that jointly addresses model-level learning objectives and system-level resource management. In particular, proactive autoscaling has not been sufficiently explored as a mechanism to adaptively manage the fluctuating workloads induced by federated LLM training, especially under privacy constraints.

To evaluate the proposed framework, we simulate federated LLM training using the WikiText-2 dataset with and without differential privacy, capturing perplexity and convergence behavior across federated rounds. In parallel, we use the Bitbrains dataset to model CPU utilization and train deep learning-based forecasting models, including LSTM, GRU, BiLSTM, CNN, CNN-LSTM, and Transformer. These forecasts drive both reactive and proactive autoscaling strategies implemented through Kubernetes pod and replica management. Experimental results demonstrate that proactive autoscaling—particularly when driven by CNN–LSTM forecasting—outperforms reactive approaches by improving resource efficiency, accelerating convergence, and maintaining stable performance even under DP constraints.

The study is structured into three phases: (i) federated LLM fine-tuning with and without DP, (ii) workload forecasting and autoscaling evaluation, and (iii) integrated analysis linking federated learning dynamics with system-level resource management. Together, these contributions establish an adaptive, scalable, and privacy-preserving framework for deploying and training LLMs in federated cloud–edge environments. This paper makes the following contributions:

- This paper proposes an integrated framework that combines federated LLM training with FedALoRA and forecasting-driven autoscaling, enabling dynamic and adaptive resource allocation in edge environments.
- We evaluate FL performance under both DP and non-DP settings, showing that DP provides stronger privacy guarantees with a minor trade-off in convergence speed, while non-DP training achieves faster adaptation.
- This work demonstrates that proactive autoscaling, when guided by FL workload dynamics, consistently outperforms reactive autoscaling by improving resource performance, ensuring system stability, and maintaining privacy across dynamic workloads.
- Through a hybrid experimental setup, leveraging the WikiText-2 dataset for federated LLM fine-tuning and the Bitbrains dataset for CPU utilization forecasting, we comprehensively evaluate deep learning models (LSTM, GRU, BiLSTM, CNN, CNN-LSTM, Transformer).
- Results show that CNN-LSTM provides the most accurate predictions to support proactive autoscaling, and replica analysis further highlights that CNN-LSTM requires the fewest pods across all scenarios, making it the most resource-efficient model. Together, these findings establish an adaptive and privacy-preserving framework for LLM training and deployment across federated edge environments.

The rest of the paper is organised as follows: Sect. 1 introduces the work and highlights its contributions. Section 2 reviews the related literature. Section 3 presents the preliminary background and architecture necessary to understand the study. Section 4 focuses on the core methodology proposed in this study. Section 5 provides details and analysis of the results obtained. Finally, Sect. 6 concludes the paper by discussing its limitations and offering suggestions for future research directions.

2 Related work

This section reviews prior research across three key domains: (i) edge and fog computing for latency-sensitive services, (ii) predictive autoscaling strategies in cloud-native edge environments, and (iii) federated fine-tuning of LLMs. We highlight how these works address specific challenges in performance, adaptability, and privacy, and position our proposed framework as a novel integration of FedALoRA-based LLM fine-tuning with forecasting-driven autoscaling for dynamic edge environments. Table 1 presents a comparison between related work and our proposed framework.

Table 1 Comparison of related work with our proposed framework

Work	Focus Area	Limitations	Our contribution
Shi et al. [2], Satyanarayanan [3]	Foundations of edge computing, latency reduction	No model-level optimization, limited system adaptivity	Integrate model-level fine-tuning with system-level autoscaling
Ju et al. [32], Jeong and Jeong [16]	Proactive Kubernetes autoscaling for cloud/edge	Resource-centric, no learning-aware scaling	Combine predictive autoscaling with federated LLM fine-tuning
Dang-Quang and Yoo [33], Kumar et al. [18, 19]	Deep learning-based workload forecasting	System-level focus, ignore learning dynamics	Use FL-induced workload traces for autoscaling
McMahan et al. [23], Abreha et al. [34], Dritsas and Trigka [24]	Federated learning frameworks	Limited focus on LLMs and system scaling	Extend FL with FedALoRA and proactive autoscaling
Luo et al. [37]	Communication-efficient FL with adaptive aggregation	No autoscaling or LLM-aware resource management	Integrate adaptive FL with forecasting-driven autoscaling
Zhang et al. [38], Jin et al. [39]	Privacy–accuracy trade-offs in adaptive FL	Privacy studied without system-level adaptation	Joint privacy-aware FL and resource management
Wu et al. [21], Kuang et al. [22], Yi et al. [29]	PEFT-based federated LLM fine-tuning	No autoscaling integration	Combine FedALoRA with predictive autoscaling
Dogani and Khunjush [12]	FL-assisted autoscaling in edge systems	No LLM or PEFT support	Unified FL–LLM–autoscaling framework

The growing adoption of edge computing has been motivated by the need to process data closer to end-users, thereby reducing latency and improving responsiveness. Foundational works such as Shi et al. [2] and Satyanarayanan [3] outlined the vision and challenges of edge computing, emphasizing its role in latency-sensitive services. Bonomi et al. [8] introduced fog computing to extend cloud capabilities to the edge, while Mao et al. [9] surveyed mobile edge computing from a communication perspective, providing the foundation for subsequent work. More recently, Qu et al. [10] explored deploying LLMs at the mobile edge, highlighting constraints in resource-limited environments. Zhang et al. [7] proposed Edgeshard, a collaborative edge framework for efficient LLM inference, addressing computational bottlenecks during real-time inference. Recently, integrated cloud-edge federated learning frameworks have emerged. Parra-Ullauri et al. [31] proposed *kubeFlower*, a Kubernetes-based privacy-preserving federated learning framework for cloud-edge environments. Although kubeFlower provides orchestration and privacy support for FL workloads, it does not incorporate forecasting-driven autoscaling or address the dynamic resource demands introduced by federated LLM fine-tuning.

Containerization and orchestration frameworks have further advanced distributed edge and cloud-native systems. Kubernetes has emerged as the standard for orchestration, enabling workload replication and autoscaling across clusters [11]. Early surveys, such as Lorido-Botran et al. [14] and Al-Dhuraibi et al. [15], reviewed the principles of elasticity in cloud systems. Later works introduced intelligent scaling methods: Ju et al. [32] proposed proactive Kubernetes autoscaling for edge workloads, while Jeong and Jeong [16] provided a comprehensive review of autoscaling strategies in cloud-native environments. Deep learning has also been integrated into predictive autoscaling: Dang-Quang and Yoo [33] used BiLSTM for Kubernetes workload prediction, and Kumar et al. [18–20] extended this with Transformer-based and hybrid deep learning models for multivariate forecasting in dynamic workloads. However, these works remain largely system-focused and do not directly link workload forecasting with LLM training dynamics.

In parallel, FL has emerged as a key paradigm for distributed training without centralized data sharing. McMahan et al. [23] introduced the federated averaging (FedAvg) algorithm, establishing a foundation for collaborative model training. Surveys such as Abreha et al. [34], Brecko et al. [35], and Dritsas and Trigka [24] provide systematic overviews of FL in edge environments, focusing on privacy,

heterogeneity, and communication efficiency. Trindade et al. [36] further examined resource management strategies for FL, underscoring the importance of workload-aware scheduling. Recent advances in federated learning have focused on improving communication efficiency and privacy robustness in heterogeneous edge environments. Luo et al. [37] proposed adaptive aggregation mechanisms for client-edge-cloud networks to reduce communication overhead under heterogeneous client capabilities. Furthermore, Zhang et al. [38] and Jin et al. [39] analyzed the privacy-accuracy trade-off in adaptive federated learning, demonstrating how gradient clipping and differential privacy configurations influence convergence stability. While these studies enhance FL efficiency and robustness, they do not consider proactive autoscaling or system-level resource adaptation.

For LLMs, parameter-efficient fine-tuning (PEFT) has emerged as a scalable alternative to full model updates. Wang et al. [28] introduced FedITD, combining PEFT with FL for insider threat detection. Wu et al. [21] proposed FedBiot for efficient LLM fine-tuning at the edge, while Kuang et al. [22] developed FederatedScope-LLM for federated LLM adaptation. Yi et al. [29] introduced FedALoRA, integrating adaptive low-rank adaptation into FL to reduce memory and communication overhead. Yang et al. [40] surveyed LoRA-based FL for foundation models, and Raje et al. [41] presented Ravan, a multi-head low-rank adaptation method for LLM fine-tuning in FL. Wang et al. [42] also studied LoRA-based LLM training in wireless networks. While these approaches optimize model training efficiency, they do not address system-level autoscaling.

The convergence of autoscaling and federated LLM fine-tuning has recently attracted attention. Wu et al. [30] explored device-edge cooperative fine-tuning for foundation models in 6 G, while Yang et al. [43] reviewed synergies between IoT, FL, and LLMs for privacy-preserving intelligent systems. Dogani and Khunjush [12] investigated proactive autoscaling in edge environments using FL models, bridging model training and system-level scaling. However, these works stop short of tightly integrating PEFT-based LLM fine-tuning with forecasting-driven autoscaling for dynamic edge resource management.

In summary, prior research spans three interconnected domains: (i) edge resources for latency-sensitive services, (ii) predictive autoscaling for dynamic resource allocation, and (iii) federated fine-tuning of LLMs using PEFT strategies. Yet, limited efforts have unified these directions into a single framework. Our study advances the field by integrating FedALoRA-based LLM fine-tuning with CNN-LSTM-driven proactive autoscaling, enabling dynamic, privacy-preserving, and resource-efficient LLM training in federated edge environments.

3 Preliminaries

This section provides an overview of the foundational concepts and methodologies underpinning our proposed framework, which integrates federated fine-tuning of LLMs with predictive autoscaling in edge-cloud environments. The preliminaries are structured into four parts: the workflow of the framework, LLMs and federated fine-tuning, the predictive autoscaling framework, and proactive autoscaling for dynamic resource management.

3.1 Proposed workflow explanation

The workflow of our framework (Fig. 1) follows a closed-loop process that integrates FL-driven LLM fine-tuning using FedALoRA with forecasting-based autoscaling. Within this workflow, the Adaptive LoRA subfigure corresponds to the module embedded within the main backbone figure and represents the client-side low-rank adaptation component that drives federated model updates and represents the client-side low-rank adaptation component that drives federated model updates. Adaptive LoRA integrates local LoRA aggregation by training and clipping adaptive weights at the client side. Adaptive LoRA integrates local LoRA training and DP-based clipping at the client side, which directly influences the computational workload observed at edge resources. This workflow consists of the following eight sequential steps:

- *Step 1* - Local Edge Initialization: The process begins with local edge resources, where multiple FL clients (Client 1 to Client n) are deployed. Each edge node contains pods or containers responsible for handling data inputs (Data 1 to Data n) and local tasks (Task 1 to Task n). These nodes form the foundation for latency-sensitive training and inference in realistic edge environments.
- *Step 2* - Local LLM Training with FedALoRA: Each selected client hosts a local instance of the LLM with a frozen backbone and an adaptive LoRA module within the overall framework (Fig. 1). In this stage, federated fine-tuning is performed using FedALoRA, where updates are applied only to the low-rank adaptation matrices (ΔW), rather than the full model. This selective update strategy significantly reduces both computation and communication overhead on the client side. The local training generates intermediate outputs ($Y_1, Y_2, \ldots, Y_n$), which capture partial improvements in perplexity and convergence speed while preserving overall model performance. These adaptive weights are subsequently clipped and aggregated in the global federated learning process.

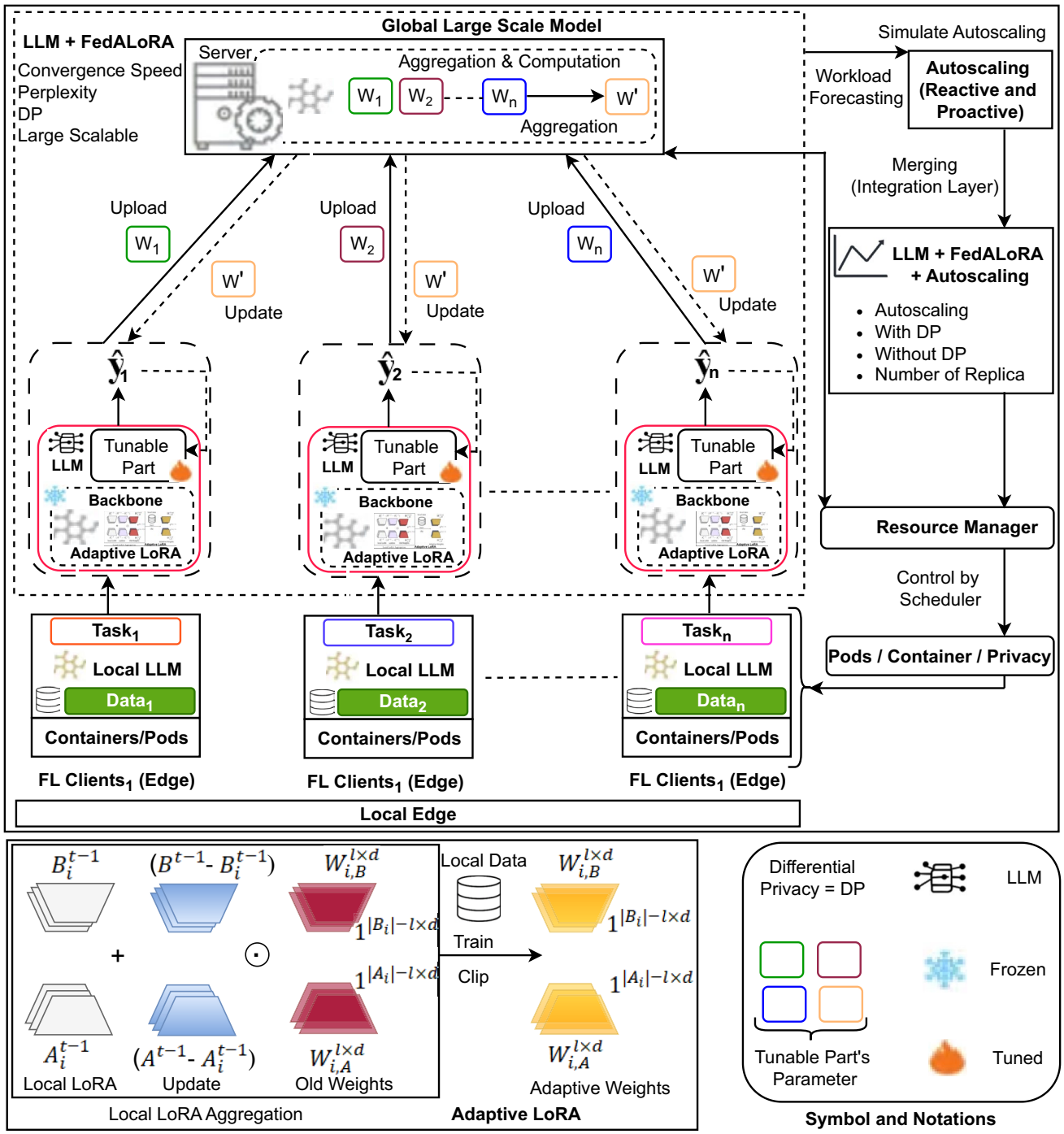


Fig. 1 Workflow of the proposed FedALoRA-enabled LLM system integrating edge resources, autoscaling, and resource management with privacy-preserving coordination

- *Step 3* - Upload and Aggregation: The local updates $(W_1, W_2, \ldots, W_n)$ are uploaded to the global LLM server, where aggregation is performed to obtain a refined global model (W'). This aggregation leads to improved convergence speed through efficient adaptation, enhanced adaptability to support many clients with limited resources, and the option of applying DP for secure gradient and weight sharing.
- *Step 4* - Workload Forecasting: Parallel to model training, workload forecasting is conducted using both WikiText2 traces, which represent training-related workload from FL fine-tuning, and Bitbrains traces, which provide

real CPU usage patterns. These datasets are combined to train forecasting models such as LSTM, GRU, and Transformer, which simulate both reactive and proactive autoscaling policies for predicting pod and container requirements under varying loads.

- *Step 5* - Integration Layer: The outcomes of LLM fine-tuning (WikiText-2 with FedALoRA) and forecasting-driven autoscaling (Bitbrains) are merged at the integration layer. This integration highlights how resource allocation dynamically adapts: with autoscaling, pod replicas scale up or down based on forecasted demand; with DP versus non-DP, trade-offs between privacy, performance, and accuracy are revealed; and without autoscaling, resource bottlenecks or over-provisioning emphasize the importance of prediction-driven resource management.
- *Step 6* - Resource Manager: The resource manager serves as a central control hub that ensures autoscaling decisions are consistently applied. It maintains bidirectional interaction with the global FL server to balance workload distribution, synchronize updates, and avoid SLA violations. The manager also adjusts allocations when anomalies or unexpected workload shifts are detected.
- *Step 7* - Scheduler and Execution: The scheduler enforces resource manager directives at the pod level, ensuring replica placement, load balancing, and privacy-preserving execution across edge nodes.
- *Step 8* - Loop Continuation: The updated global model and autoscaling feedback are returned to edge clients, sustaining the iterative cycle of FL fine-tuning, scaling, and privacy preservation.

Importantly, the federated learning and autoscaling processes are tightly coupled through a feedback-driven interaction. During each FL round, FedALoRA-based fine-tuning generates dynamic computational workloads at the edge nodes, reflected in CPU utilization and resource demand patterns. These workload traces are collected and fed into the forecasting models, which predict future resource requirements and drive proactive autoscaling decisions. The resulting scaling actions adjust pod replicas and container resources, directly impacting the execution efficiency and stability of subsequent FL training rounds. This closed-loop interaction ensures that model-level adaptation (FedALoRA fine-tuning) and system-level adaptation (forecast-driven autoscaling) operate in a coordinated manner, enabling scalable, stable, and privacy-preserving federated LLM training in resource-constrained edge environments.

3.2 LLMs, edge deployment, and federated fine-tuning

LLMs have demonstrated remarkable capabilities in natural language understanding and generation. Despite their success, deploying LLMs in edge environments is challenging due to limited computational resources, restricted bandwidth, and heterogeneous data distributions across clients. Full fine-tuning of these models in distributed environments is often computationally expensive and communication-intensive. To address these limitations, PEFT methods such as LoRA, ALoRA, LLRA, and QLoRA have been introduced [28]. These methods reduce the number of trainable parameters, memory footprint, and overall computational requirements, though each presents trade-offs in performance, personalization, and communication cost.

FL provides an effective solution by enabling multiple clients to collaboratively train models without sharing raw data, thereby preserving privacy. When combined with adaptive fine-tuning techniques, FL can efficiently personalize LLMs to handle diverse client datasets. Several approaches illustrate this balance between performance and adaptability. LoRA reduces computation and storage costs but offers limited personalization. ALoRA enhances personalization while maintaining performance, though it still introduces moderate communication overhead. LLRA improves accuracy with higher rank values but demands more memory and communication, making it less suitable for resource-constrained edge devices [30]. QLoRA applies 4-bit quantization to reduce memory consumption, allowing very large models to be fine-tuned, yet it does not fully address communication bottlenecks in federated settings. More recently, FedALoRA [29] has emerged as a promising approach. By integrating adaptive low-rank fine-tuning with federated optimization, FedALoRA achieves high accuracy while keeping computational and memory requirements low and minimizing communication overhead. This makes it especially suitable for distributed, IoT, and edge computing scenarios.

From a scalability perspective, the proposed framework is designed to efficiently handle both an increasing number of clients and larger LLM models. As the number of participating clients grows, communication overhead scales with the size of the low-rank adapter updates rather than the full model parameters, ensuring that aggregation remains efficient even in large-scale federated settings. Moreover, because the pretrained LLM backbone remains frozen, scaling to larger models does not proportionally increase client-side computation or memory usage; only the lightweight

LoRA parameters are trained and exchanged. This design enables the framework to generalize across different LLM sizes and client populations while maintaining stable training dynamics and manageable system overhead. Consequently, the proposed FedALoRA-based framework is well-suited for scalable and privacy-preserving LLM fine-tuning in large, heterogeneous edge environments.

Although the autoscaling mechanism can effectively manage underprovisioning and overprovisioning in isolation, its robustness is preserved when integrated with the proposed FedALoRA-based framework. In the integrated setting, FedALoRA-driven federated LLM training influences the intensity and temporal characteristics of system workloads, while the forecasting-driven autoscaling module continues to regulate resource allocation based on predicted CPU demand. As a result, scaling decisions remain stable and responsive under both DP and non-DP configurations. The use of predictive models and the resource removal strategy (RRS) further prevents abrupt scaling actions, ensuring that pod replicas are adjusted smoothly despite the additional computational overhead introduced by federated training. These results confirm that the proposed framework maintains autoscaling robustness while jointly supporting privacy-preserving federated LLM fine-tuning in dynamic edge environments.

Our proposed FedALoRA-based framework extends these PEFT methods by tightly integrating federated LLM fine-tuning with forecasting-driven autoscaling, enabling dynamic application scaling in edge-distributed environments by proactively adjusting resources based on workload variations induced by federated training.

3.3 Distributed fine-tuning and predictive autoscaling framework

The proposed framework integrates federated model training with intelligent system-level resource management in a closed-loop manner. In the first phase, FedALoRA is employed to fine-tune large language models (LLMs) across distributed clients [42]. Instead of updating the full model, each client modifies only the low-rank adapter layers while keeping the pretrained backbone parameters frozen. This design significantly reduces computational cost and memory consumption, making the approach suitable for resource-constrained edge devices. Differential privacy (DP) can be optionally applied during training to protect sensitive client data while enabling collaborative knowledge sharing across clients.

The outcome of this phase is a set of workload traces that characterize system behavior during federated LLM fine-tuning. These traces capture computational activity, communication overhead, and learning-related metrics such as CPU utilization, perplexity, convergence rate, and the additional cost introduced by privacy mechanisms. Collectively, these metrics provide a comprehensive and realistic foundation for predictive resource management in the subsequent phase. In the second phase, the framework focuses on forecasting-driven autoscaling using the workload traces generated during federated training [18, 19, 44]. The selection of forecasting models is motivated by the need to capture diverse workload characteristics observed in cloud–edge environments. Recurrent neural networks such as LSTM and GRU are selected due to their effectiveness in modeling temporal dependencies in CPU utilization time series. BiLSTM is included to enhance prediction accuracy by leveraging both past and future contextual information, which is particularly useful for highly dynamic and bursty workloads.

Convolutional Neural Networks (CNNs) are employed to extract local and short-term patterns from workload traces, while the hybrid CNN-LSTM model combines spatial feature extraction with temporal dependency learning to improve robustness under fluctuating resource demands. Transformer models are also incorporated due to their self-attention mechanism, which enables efficient modeling of long-range dependencies and complex workload interactions that may not be well captured by recurrent architectures alone. Together, these six models represent complementary forecasting paradigms, recurrent, convolutional, hybrid, and attention-based, allowing a systematic and fair comparison of predictive autoscaling performance. Based on the predicted resource demands, the autoscaling controller proactively adjusts Kubernetes pod replicas to allocate sufficient resources before performance bottlenecks occur. A resource removal strategy (RRS) is further applied to gradually release unused capacity, preventing abrupt scaling actions and maintaining system stability. Throughout this phase, predicted and actual CPU utilization, pod allocation efficiency, and scaling responsiveness are continuously monitored, enabling iterative refinement of both the forecasting models and the autoscaling policy.

By tightly coupling federated LLM fine-tuning with forecasting-driven autoscaling, the proposed framework achieves coordinated model-level and system-level adaptation. This integrated design enables scalable, stable, and privacy-preserving federated LLM training while

ensuring efficient resource utilization in dynamic cloud–edge environments.

3.4 Proactive autoscaling for dynamic resource management

Autoscaling is traditionally approached in two ways: reactive and proactive. Reactive autoscaling adjusts resources only after system metrics such as CPU or memory usage have already crossed critical thresholds [45]. Although simple, this approach often reacts too late during sudden workload surges, leading to service-level violations and unstable performance. Proactive autoscaling takes a different path by anticipating workload changes in advance through predictive modeling. By forecasting CPU demand and workload fluctuations, resources are provisioned ahead of time, reducing delays, preventing bottlenecks, and ensuring smoother operation [18–20]. This approach is particularly valuable in federated training of LLMs, where workloads are highly dynamic and often unpredictable.

In addition to communication overhead, the computational overhead of executing forecasting models in edge environments is also an important consideration. In this work, forecasting models were deliberately designed to remain lightweight and resource-efficient. Shallow recurrent architectures (LSTM, GRU, and BiLSTM with limited layers), compact CNN-based models, and a lightweight Transformer configuration were selected to balance prediction accuracy with computational feasibility on edge infrastructures. Moreover, forecasting is performed at coarse-grained control intervals rather than continuously, which significantly reduces runtime overhead. In practical deployments, these forecasting tasks can be executed at edge controllers or centralized management nodes instead of individual edge clients, further minimizing the computational burden on resource-constrained devices. As a result, the proposed forecasting-driven autoscaling framework introduces minimal additional computational overhead while enabling proactive and stable resource management in federated edge environments.

Proactive autoscaling is tightly integrated with FedALoRA fine-tuning. Workload traces that reflect convergence bursts and communication spikes are mapped to predicted CPU demands, allowing Kubernetes pod allocations to be adjusted before the system reaches a critical state. An RRS ensures that unused capacity is released in a controlled manner, avoiding waste while maintaining stability. Throughout this phase, system-level indicators such as CPU utilization, replica count, and service-level compliance are continuously observed. These insights strengthen the link between model-level fine-tuning and system-level resource management, creating a unified framework that sustains both accuracy and performance in distributed edge environments.

While DP is the primary privacy mechanism evaluated in this study, the proposed framework is not limited to DP alone. Other privacy-preserving techniques, such as secure aggregation and homomorphic encryption, can be seamlessly integrated into the federated learning phase. Secure aggregation can be incorporated at the server-side aggregation step to ensure that individual client updates remain inaccessible, while homomorphic encryption can enable encrypted computation over model updates at the cost of additional computational overhead. These techniques are complementary to FedALoRA and can be selectively applied depending on the desired privacy-performance trade-offs in edge environments.

In addition to replica counts and CPU utilization, the framework can be naturally extended to incorporate other system-level resource metrics, such as memory consumption and network bandwidth usage. Forecasting models can be trained on multivariate workload traces that include memory pressure and network throughput, enabling more comprehensive autoscaling decisions. Although the current experimental evaluation focuses on CPU-driven scaling to maintain clarity and reproducibility, future extensions of this methodology will support multi-resource autoscaling to further enhance system efficiency and robustness in cloud-native edge environments.

Overall, proactive autoscaling enhances resource efficiency, accelerates convergence, preserves privacy, and ensures system stability, making our framework suitable for real-world federated LLM deployment in edge environments.

4 Experiment setup and methodology

This study evaluates the synergy between FL for LLMs and forecasting-driven autoscaling under realistic workload traces. Two complementary datasets were used to evaluate the proposed framework.

- WikiText-2 dataset: The WikiText-2 dataset[1] was employed for federated fine-tuning of LLMs using FedALoRA. This dataset contains over 2 million tokens extracted from verified Wikipedia articles, offering natural language diversity suitable for language model training.

[1] https://www.kaggle.com/datasets/vivekmettu/wikitext2-data

In our setup, client devices fine-tuned only the low-rank adapter parameters to minimize communication and memory costs, with and without DP applied during global aggregation.

- Gwa Bitbrains dataset: The Bitbrains virtual machine traces dataset[2] was used for CPU utilization forecasting and workload-driven autoscaling. The dataset includes detailed performance logs such as CPU usage, memory, disk, and network throughput from a real private cloud environment. In this study, CPU usage traces were extracted, preprocessed into minute-level intervals, and used to train forecasting models (LSTM, GRU, BiLSTM, CNN, CNN-LSTM, Transformer). Forecasts were then mapped to pod replica allocations for both reactive and proactive autoscaling strategies.

The WikiText-2 dataset was used to evaluate FedALoRA fine-tuning performance with and without DP, capturing both convergence speed and perplexity across federated rounds. The Bitbrains dataset provided real-world workload dynamics to drive predictive autoscaling. Together, these datasets enabled end-to-end validation of federated training, forecasting, and scaling performance, including pod scaling decisions and replica requirements under DP and non-DP settings in dynamic edge–cloud environments. The workflow is structured into three interconnected phases, with each phase building upon the outcomes of the previous one.

4.1 Hyperparameter tuning and model configuration

To ensure a fair and reproducible evaluation of forecasting-driven autoscaling, a unified hyperparameter tuning strategy was applied across all models. All forecasting models were trained using the same preprocessed CPU utilization traces from the Bitbrains dataset, normalized using min-max scaling and segmented into fixed-length input windows with a consistent prediction horizon. Hyperparameter tuning was conducted via grid search on a validation set, with the objective of minimizing Mean Squared Error (MSE). For recurrent models (LSTM, GRU, and BiLSTM), the number of hidden units {32, 64, 128}, stacked layers {1, 2}, and learning rates $\{10^{-3}, 10^{-4}\}$ were evaluated. The final configuration selected 64 hidden units with a single recurrent layer, balancing accuracy and computational efficiency. CNN-based models employed one-dimensional convolutions with kernel sizes {3, 5} and filter counts {32, 64}, followed by max-pooling. The hybrid CNN–LSTM model combined these convolutional layers with an LSTM layer of 64 hidden units and consistently achieved the lowest validation error. For the Transformer model, embedding dimensions {32, 64}, attention heads {2, 4}, and encoder layers {1, 2} were tuned, with a lightweight configuration (64-dimensional embedding, 2 heads, 1 layer) selected for efficiency.

In the federated learning phase, FedALoRA was used to fine-tune LLMs by updating only low-rank adapter parameters while keeping the pretrained backbone frozen. The total number of federated clients was set to N, with multiple local training rounds followed by global aggregation. Differential privacy was applied to enhance privacy guarantees during FL. Specifically, client-side LoRA updates were clipped using an ℓ_2-norm bound of $C = 1.0$ before aggregation. The privacy budget was controlled using ϵ-differential privacy, with ϵ values set to {1.0, 2.0} to analyze the trade-off between privacy protection and model utility. Smaller ϵ values provided stronger privacy guarantees at the cost of slightly slower convergence, while larger values improved accuracy with relaxed privacy constraints.

Across all forecasting models, the Adam optimizer was used with a batch size of 32 and a maximum of 100 training epochs. Early stopping based on validation loss with a patience of 10 epochs was applied to prevent overfitting. The final hyperparameter settings reported in Sect. 5 correspond to the configurations achieving the lowest validation MSE, ensuring a fair comparison across forecasting architectures.

All experiments were implemented using Python (v3.10) and PyTorch (v2.x). The FedALoRA implementation was adapted from the official reference codebase, while forecasting models were developed using PyTorch along with standard scientific libraries such as NumPy and Pandas. Autoscaling logic was evaluated using Kubernetes-based resource management scripts with fixed control intervals. Experiments were conducted on a system equipped with an NVIDIA GPU (24 GB VRAM), 64 GB system memory, and a multi-core CPU, enabling efficient federated training, forecasting, and autoscaling evaluation under realistic cloud-edge workload conditions.

4.2 Phase 1: FL-based LLM fine-tuning with FedALoRA

In phase 1, we adopt the FedALoRA algorithm [29] to fine-tune a GPT-style model on the WikiText-2 dataset. FedALoRA extends FL by combining low-rank adaptation with client–server collaboration to efficiently fine-tune LLMs

[2] https://www.kaggle.com/datasets/gauravdhamane/gwa-bitbrains/data

(Algorithm 1). The process begins with the server initializing and broadcasting the pretrained model parameters θ and global LoRA factors (A_0, B_0) to all participating clients. Each client performs *WRound* warming rounds of stochastic fine-tuning (SFT) on local mini-batches $D_i^{(s)}$ to adapt its LoRA parameters (A_i, B_i) before global training begins. During each round *t*, clients compute adapted LoRA parameters as shown in Eq 1:

$$\hat{A}_i^t = A + \Delta A_i^t, \quad \hat{B}_i^t = B + \Delta B_i^t \tag{1}$$

here ΔA_i^t and ΔB_i^t are the local updates obtained by minimizing the LLM fine-tuning loss $\mathcal{L}$ (e.g., cross-entropy for next-token prediction) on the sampled mini-batch as shown in Eq 2:

$$\Delta A_i^t, \Delta B_i^t = \arg\min_{A_i, B_i} \mathcal{L}\big(f_{\text{LLM}}(D_i^{(s)}; \theta, A_i, B_i)\big) \tag{2}$$

To ensure privacy, DP clipping is applied to updates as shown in Eq 3:

$$\Delta A_i^t \leftarrow \mathcal{C}(\Delta A_i^t), \quad \Delta B_i^t \leftarrow \mathcal{C}(\Delta B_i^t) \tag{3}$$

After local updates, each client sends $(\hat{A}_i^t, \hat{B}_i^t)$ to the server, which aggregates the updates using weighted averaging based on dataset sizes, as shown in Eq 4:

$$A = \sum_{i=1}^{N} k_i \hat{A}_i^t, \quad B = \sum_{i=1}^{N} k_i \hat{B}_i^t, \quad k_i = \frac{|D_i|}{|D|} \tag{4}$$

In these equations, the local updates ΔA_i^t and ΔB_i^t correspond to fine-tuning adjustments of the LLM's low-rank adapter matrices, while the frozen pretrained parameters θ remain unchanged. By exchanging only these lightweight updates, FedALoRA reduces communication and memory overhead, making distributed LLM training feasible on heterogeneous clients.

Here, *N* denotes the total number of clients participating in FL, and *D* represents the full dataset across all clients, with D_i being the local dataset of client *i*. Each mini-batch sampled from client *i*'s dataset is denoted by $D_i^{(s)}$, and *WRound* indicates the number of warming rounds used for initial local adaptation. The global pretrained model parameters are represented by θ, while (A_0, B_0) are the initial global LoRA matrices for low-rank adaptation. The locally trained LoRA parameters for client *i* are (A_i, B_i), and their adapted versions at round *t* are $(\hat{A}_i^t, \hat{B}_i^t)$. The learning rate for LoRA updates is η, and $\mathcal{C}(\cdot)$ is the clipping function applied for DP.

During the warm-up stage, $W_{i,A}^l$ and $W_{i,B}^l$ denote the layer-wise trainable LoRA weights for client *i* at layer *l*. These parameters are optimized using supervised fine-tuning (SFT), which performs gradient-based updates of the low-rank adapters on local mini-batches while keeping the pretrained backbone θ frozen. The purpose of the warming rounds is to stabilize local LoRA representations before federated aggregation, thereby reducing update variance and improving convergence in subsequent FL rounds. For notational consistency, these layer-wise parameters are collectively represented by the compact matrix form (A_i, B_i) throughout the equations and Algorithm 1. The server aggregates client updates using the weighting factor $k_i = |D_i|/|D|$, and the total number of federated learning rounds is denoted by *T*. The supervised fine-tuning function applied to LoRA parameters is denoted by $\text{SFT}(\cdot)$, and the final updated local model for client *i* is given by $\theta + A_i^t B_i^t$, which combines the frozen pretrained weights with low-rank adaptations.

By freezing θ and exchanging only low-rank updates, FedALoRA enables personalized yet globally aligned LLM fine-tuning, providing a robust foundation for scalable federated LLM training. Algorithm 1 summarizes the FedALoRA workflow, highlighting the role of warming rounds for stable initialization, supervised fine-tuning (SFT) for local low-rank adaptation, and DP clipping for privacy preservation before federated aggregation. Phase I establishes efficient low-rank adaptation across distributed clients, providing a robust foundation for scalable FL. The resulting workload traces directly feed into phase 2 forecasting-driven autoscaling and subsequently inform phase 3 integrated FL–autoscaling analysis.

Algorithm 1 FedALoRA for LLM Fine-Tuning (adapted from [29])

Require: N clients with local datasets D_i, pretrained LLM weights θ, initial LoRA (A_0, B_0), learning rate η, warming rounds $WRound$, loss $\mathcal{L}$, DP clipping $\mathcal{C}(\cdot)$
Ensure: Local models $\theta + A_i^t B_i^t$ for all $i \in [N]$
Server broadcasts (θ, A_0, B_0) to all clients
for each client $i \in [N]$ **do**
  Initialize local LoRA parameters $(A_i, B_i) \leftarrow (A_0, B_0)$
  for $WRound$ iterations **do**
    Sample mini-batch $D_i^{(s)} \subset D_i$
    $(A_i, B_i) \leftarrow \mathrm{SFT}(f_{\mathrm{LLM}}(D_i^{(s)}; \theta, A_i, B_i))$ {Eq. 2}
  end for
end for
for round $t = 1$ to T **do**
  Server broadcasts (θ, A, B) to clients
  for each client $i \in [N]$ **do**
    Sample mini-batch $D_i^{(s)}$
    Compute local updates $\Delta A_i^t, \Delta B_i^t$ using Eq. 2
    Apply DP clipping: Eq. 3
    Update adapted LoRA parameters: Eq. 1
    Send $(\hat{A}_i^t, \hat{B}_i^t)$ to server
  end for
  Server aggregates updates: Eq. 4
end for
return Local models $\theta + A_i^t B_i^t$ for all $i \in [N]$

4.3 Phase 2: forecasting-driven autoscaling service

Building on the workload traces generated in phase 1 from FedALoRA-based FL, phase 2 leverages the Bitbrains dataset to train forecasting models (LSTM, GRU, BiLSTM, CNN, CNN-LSTM, Transformer). To evaluate the forecasting models, we compute the loss function using Mean Squared Error (MSE), defined as in Eq 5:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{5}$$

where y_i denotes the actual CPU utilization at time i, and $\hat{y}_i$ represents the predicted value.

These models predict CPU utilization for upcoming intervals, denoted as CPU_{t+1}. The predictions are then mapped to pod allocation decisions, where each pod is allocated a fixed CPU capacity, CPU_{perPod}. The predicted number of pods required at time $t+1$ is thus calculated as shown in Eq 6:

$$Pods_{t+1} = \frac{CPU_{t+1}}{CPU_{\mathrm{perPod}}}. \tag{6}$$

The system uses a RRS to adjust the number of pods incrementally, avoiding abrupt scaling changes. If the predicted pods exceed the current count $pods_t$, a scale-out operation is performed according to Eq 7:

$$pods_{t+1} = \min(pods_t + (pods_{t+1} - pods_t) \cdot RRS, pods_{\max}) \tag{7}$$

Conversely, if the predicted pods are fewer than $pods_t$, scale-in operation is executed as defined in Eq 8:

$$pods_{t+1} = \max(pods_t - (pods_t - pods_{t+1}) \cdot RRS, pods_{\min}) \tag{8}$$

Here, $pods_{\min}$ and $pods_{\max}$ define the minimum and maximum allowed pod counts, while $RRS \in (0, 1]$ controls the proportion of pods adjusted per iteration. The system waits for a fixed control delay time (CDT) before applying the next adjustment, ensuring stability under fluctuating workloads.

The Forecasting-Driven Autoscaling algorithm is summarized in Algorithm 2. At each iteration, predicted CPU usage is converted into pod requirements every 60 s or 1 min. The control delay time (CDT) defines the fixed control interval between consecutive executions of the autoscaling loop, ensuring that the system waits for the effects of a previous scaling action to stabilize before initiating the

next prediction and scaling decision. Based on the comparison with the current pod count, either scale-out or scale-in is executed following Eq. 7 and Eq. 8. The resulting scaling decisions enable proactive autoscaling, maintaining performance and resource performance even under dynamic workloads generated by distributed LLM training.

Algorithm 2 Forecasting-Driven Autoscaling

Require: CPU_{t+1} // Predicted CPU usage at time $t+1$
Require: CPU_{perPod} // CPU capacity per pod
while system is operational **do**
  $pods_{t+1} = \frac{CPU_{t+1}}{CPU_{\text{perPod}}}$
  if $pods_{t+1} > pods_t$ **then**
    $pods_{deficit} = (pods_{t+1} - pods_t) \times RRS$
    $pods_{t+1} = \min(pods_t + pods_{deficit}, pods_{\max})$
    EXECUTE_SCALE_OUT$(pods_{t+1})$
  else if $pods_{t+1} < pods_t$ **then**
    $pods_{redundant} = (pods_t - pods_{t+1}) \times RRS$
    $pods_{t+1} = \max(pods_t - pods_{redundant}, pods_{\min})$
    EXECUTE_SCALE_IN$(pods_{t+1})$
  else
    $pods_{t+1} = pods_t$ // No adjustment
  end if
  Wait for control delay time (CDT) = 60s before next iteration
end while

In this phase, CPU_{t+1} represents the predicted CPU utilization for the upcoming interval, and CPU_{perPod} indicates the CPU capacity assigned per pod. $pods_t$ is the current number of pods, and $pods_{t+1}$ is the calculated number of pods to deploy. The parameters $pods_{\min}$ and $pods_{\max}$ enforce lower and upper bounds for scaling operations, *RRS* determines the incremental adjustment factor for pod scaling, and *CDT* represents the CDT between successive scaling decisions. This delay allows resource utilization and workload distribution to stabilize after each scaling action, preventing rapid oscillations and ensuring stable proactive autoscaling. By leveraging workload traces from phase 1, this forecasting-driven approach anticipates spikes in distributed LLM training workloads and ensures system stability and resource performance before actual load increases occur.

This phase establishes the connection between predictive analytics and system-level adaptation. By using FL-generated workload traces as inputs, the proactive autoscaling mechanism can anticipate demand spikes caused by distributed LLM training. Unlike reactive scaling, which responds only after workload increases are observed, proactive scaling ensures that resources are allocated in advance, maintaining performance and enabling efficient pod assignment. The resulting scaling decisions and resource allocation patterns provide the foundation for phase 3, where FL workloads and autoscaling policies are integrated for end-to-end performance evaluation.

4.4 Phase 3: integrated FL-autoscaling analysis

Phase 3 integrates the outcomes of phases 1 and 2, linking FL-induced workload dynamics from FedALoRA-based LLM training with autoscaling decisions derived from CPU usage forecasts. Metrics such as convergence spikes, communication bursts, and DP overhead are translated into equivalent CPU demand ($CPU_{FL}(r)$), which is then combined with predicted CPU usage (CPU_{t+1}) from phase 2. This ensures that autoscaling decisions are synchronized with both anticipated and real-time workloads, maintaining performance and system stability.

Algorithm 3 Federated Learning + Autoscaling Integration Algorithm

Require: $w(r)$ // Workload intensity at FL round r
Require: CPU_{t+1} // Forecasted CPU usage from phase 2
Require: CPU_{perPod} // CPU capacity per pod
for each FL round $r = 1 \dots R$ **do**
  Collect FL metrics: $Perplexity(r)$, $CommCost(r)$, $DP\_overhead(r)$
  Map FL workload to equivalent CPU demand: $CPU_{FL}(r)$
  $CPU_{demand}(t+1) = \max(CPU_{t+1}, CPU_{FL}(r))$
  $pods_{t+1} = \frac{CPU_{demand}(t+1)}{CPU_{\text{perPod}}}$
  if $pods_{t+1} > pods_t$ **then**
    Apply scaling-out with RRS
  else if $pods_{t+1} < pods_t$ **then**
    Apply scaling-in with RRS
  else
    $pods_{t+1} = pods_t$
  end if
  Execute FL round r with updated pod allocation
end for

This phase 3 bridges phases 1 and 2 by feeding the LLM training workload metrics from FedALoRA (phase 1) into the autoscaling decisions derived from CPU forecasts (phase 2). Specifically:

- Metrics like perplexity changes, communication cost, and DP overhead from FedALoRA fine-tuning are converted to $CPU_{FL}(r)$, quantifying the FL-induced demand.
- Forecasted CPU usage CPU_{t+1} from predictive models ensures proactive scaling ahead of anticipated load.
- The combination via $\max(CPU_{t+1}, CPU_{FL}(r))$ guarantees that pod allocation accounts for both proactive predictions and FL-induced spikes.
- Scaling decisions are executed using the RRS to maintain stability and avoid oscillations during sudden workload changes.

Consequently, phase 3 ensures that distributed LLM training is resource-aware, autoscaling efficiency with perplexity, and speed convergence, avoiding under-provisioning during convergence peaks while minimizing over-provisioning during low-load periods. It represents a fully integrated FL–autoscaling framework, where phase 1 generates workload traces, phase 2 predicts future demand, and phase 3 enforces adaptive resource management.

Our experiments explore the interaction between FL for LLMs and proactive autoscaling in cloud-native edge environments. In phase 1, FedALoRA enables distributed LLM fine-tuning where clients update low-rank adapters locally while keeping pretrained weights frozen, optionally applying DP. In phase 2, workload traces from phase 1 are used to train forecasting models such as LSTM, GRU, BiLSTM, CNN, CNN-LSTM, and Transformer, which predict CPU usage and drive proactive pod allocation using the RRS. phase 3 maps FL workload spikes, including convergence surges and communication bursts, to predicted CPU demand, dynamically adjusting pod allocations to maintain system stability. Together, these phases provide a unified experimental methodology that captures both model-level (perplexity, communication, DP overhead) and system-level (CPU utilization, number of replicas allocated) behaviors, while addressing privacy concerns and enabling faster convergence, thereby validating the proposed framework in realistic federated and cloud-native edge environments.

5 Results

The results demonstrate the effectiveness of integrating FedALoRA-based federated LLMs training, workload forecasting, and proactive autoscaling within edge environments. Each figure highlights a core component of the proposed framework, providing comparative insights across DP and non-DP settings, perplexity, convergence speed, and the performance differences between reactive and proactive pod scaling strategies.

We begin by evaluating federated training using FedALoRA for LLM fine-tuning. Figure 2 presents a comparison of perplexity across federated rounds with and without DP, using FedALoRA. Perplexity is a widely used metric in language modeling, where lower values indicate better predictive performance. The x-axis represents federated rounds, while the y-axis shows perplexity values ranging from 80 to

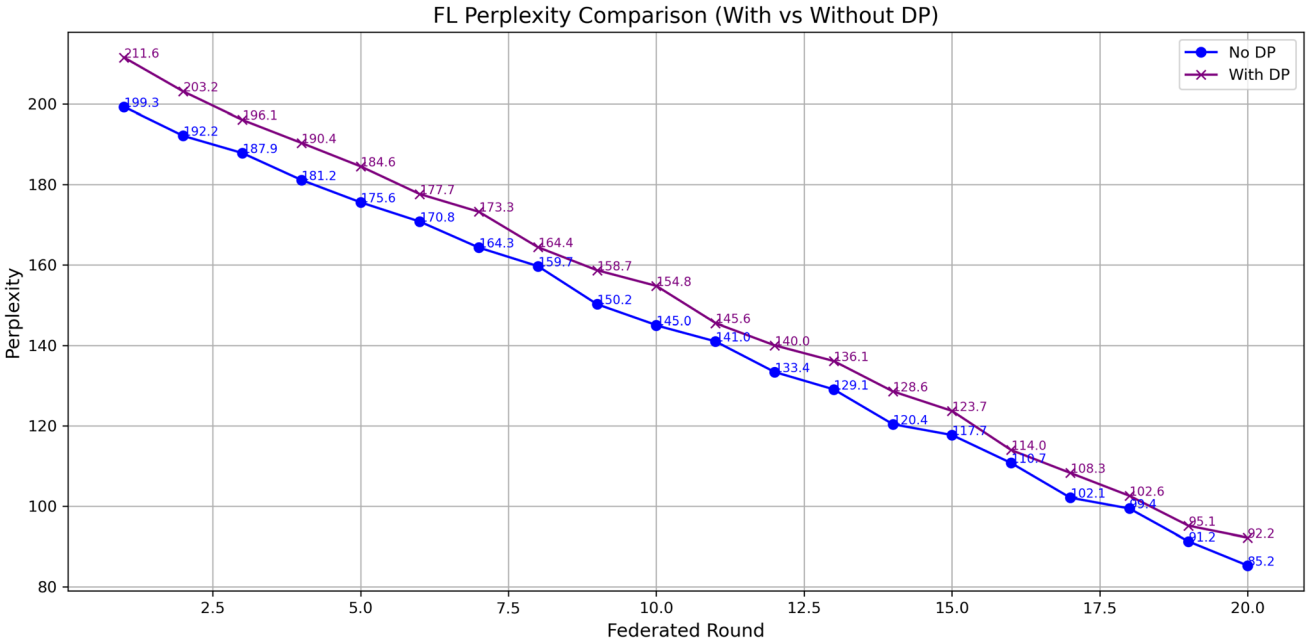


Fig. 2 Comparison of perplexity across FL rounds with and without DP using FedALoRA

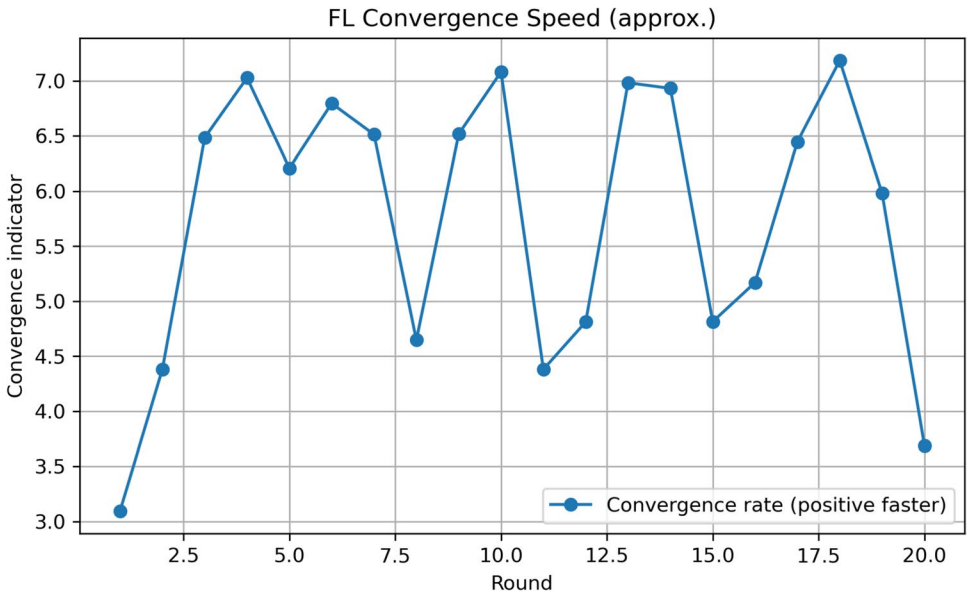


Fig. 3 Convergence speed of FedALoRA-based federated training

200. Blue circles denote results without DP, while another color indicates results with DP.

The results show that FL without DP achieves perplexity values ranging from a higher value of 199.3 to a lower value of 85.2, whereas with DP, the values range from 211.6 to 92.2. Beyond the absolute perplexity values, the impact of differential privacy can be quantitatively observed through relative accuracy degradation and convergence behavior. Compared to the non-DP setting, the DP-enabled configuration exhibits an average perplexity increase of approximately 7–13% across federated rounds, reflecting the noise introduced during privacy-preserving aggregation. Although the DP-enabled model converges slightly more slowly, it provides the added benefits of secure aggregation and enhanced privacy protection. Moreover, integrating FedALoRA for LLM fine-tuning improves convergence efficiency, helping the DP-enabled model approach near-optimized performance while maintaining privacy guarantees.

Figure 3 illustrates the convergence speed of FedALoRA-based federated LLM training. The diagram plots convergence indicators, with the x-axis representing training rounds (up to 20) and the y-axis showing convergence values. In this work, the convergence indicator is defined as the normalized reduction in training loss across federated

rounds, computed as the relative improvement in perplexity between consecutive rounds. Higher values indicate faster convergence, reflecting more rapid loss reduction per training round. As shown in Fig. 3, the blue curve demonstrates a clear upward trajectory, reaching a value of 8.0 within just 20 rounds with positive and steeper slope reflects faster convergence rates. The consistently positive convergence trend highlights FedALoRA's effectiveness in accelerating training compared to full fine-tuning, confirming that model adaptation can be achieved efficiently while preserving privacy.

Next, we trained forecasting models: LSTM, GRU, BiLSTM, CNN, CNN-LSTM, and Transformer, using workload traces from WikiText-2 for FL demand and the Bitbrains dataset for CPU utilization. Forecasting performance was evaluated using the MSE, as defined in Eq. 5. Figure 4 presents the results, where the x-axis represents time in federated rounds and the y-axis shows CPU usage percentage. In the graph, the blue curve represents the actual values, while the orange dashed curve denotes the predicted values. MSE was used as the error metric, where lower values indicate better forecasting performance. Among all models, the obtained MSE values were: LSTM (4.0), GRU (1.0), CNN (9.0), CNN-LSTM (0.25), Bi-LSTM (4.0), and Transformer (1.0). CNN-LSTM consistently achieved the lowest MSE, with its predictions closely matching the actual values, demonstrating superior predictive accuracy compared to the other approaches.

Figure 5 illustrates the actual versus predicted CPU utilization for all forecasting models over time, measured in minutes. The x-axis represents time (0–1400 min), while the y-axis shows CPU usage percentage, ranging from 0 to 55 depending on workload dynamics. In the plot, the black dashed line represents the true values, and the yellow line denotes the predicted values.

Starting with LSTM, the predicted CPU usage ranges between 31.8 and 50.6%. For CNN, the range is 26.6–39.4%, while CNN-LSTM achieves 33.2–50.8%. GRU produces values between 31.4 and 49.0%, BiLSTM ranges from 30.4 to 50.8%, and the Transformer spans 29.3–47.5%. Among these, the CNN-LSTM model provided the most accurate fit, with predictions closely matching the actual values. BiLSTM also performed competitively but showed slightly less stable predictions compared to CNN-LSTM. These results confirm CNN-LSTM as the most effective model for dynamic workload prediction in edge environments. The predicted CPU usage percentages were further used to calculate provisioning metrics and scaling decisions, as described in the experimental setup and methodology sections.

We have already discussed Fig. 2, which illustrates the perplexity of FL with FedALoRA training. In this setting, FL without DP achieved perplexity values ranging from 199.3 (maximum) to 85.2 (minimum), while FL with DP ranged from 211.6 to 92.2. Building on this, Fig. 6 connects FL perplexity with CPU demand and provisioned pods in the non-DP case, whereas Fig. 7 illustrates the DP-enabled case. In both figures, the x-axis represents FL rounds or time (up to 20 rounds), while the y-axis represents provisioned pods or CPU usage percentage, ranging from 0 to 2500. The

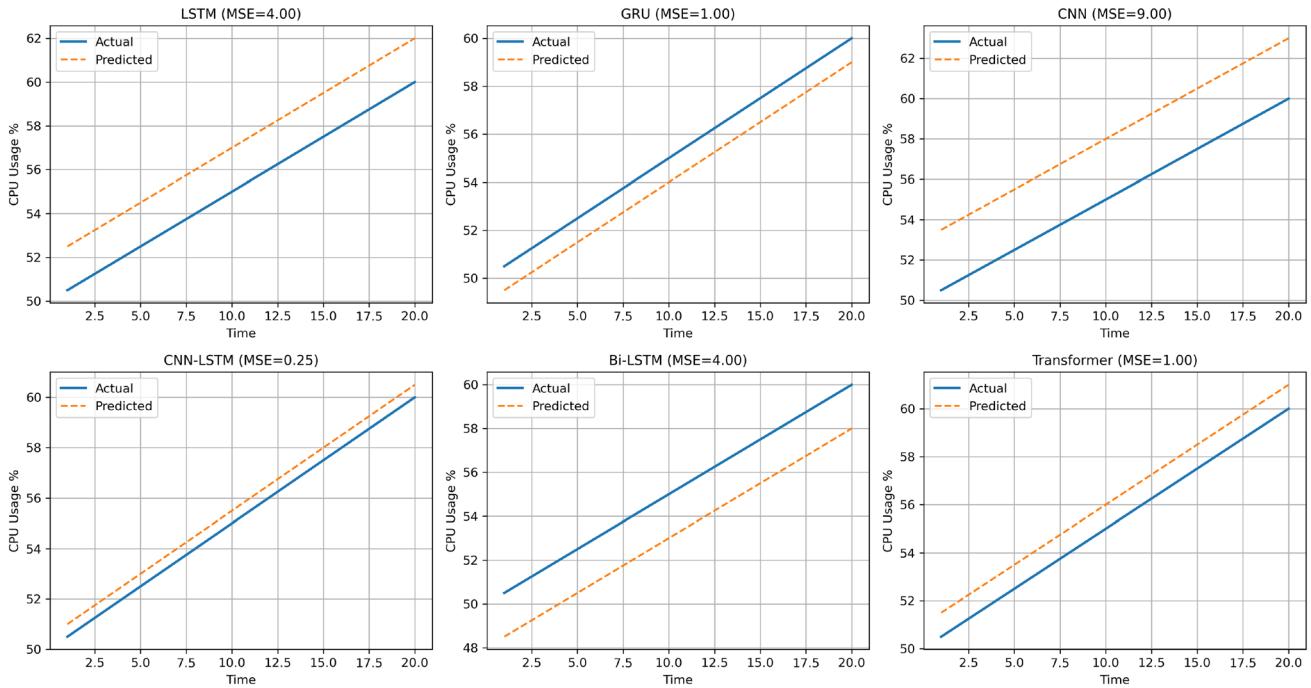


Fig. 4 Mean squared error (MSE) of forecasting models (LSTM, GRU, BiLSTM, CNN, CNN-LSTM, Transformer) for CPU utilization prediction

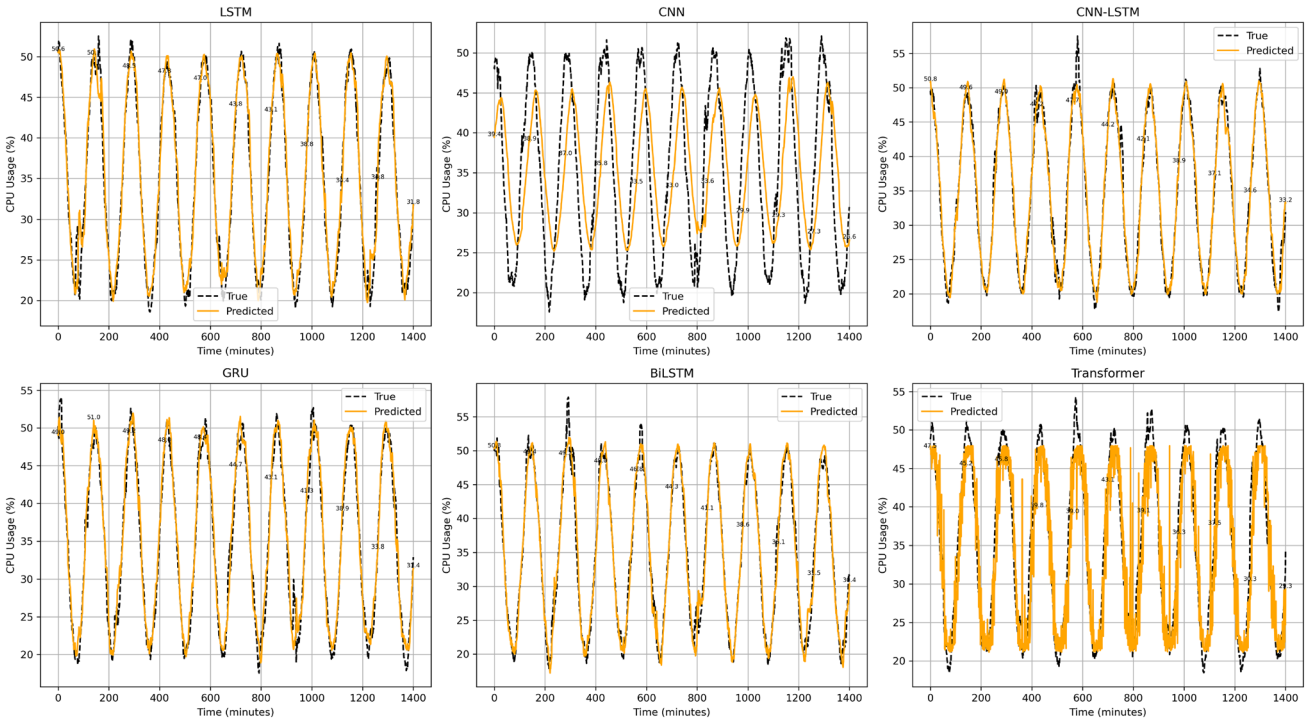


Fig. 5 Actual versus predicted CPU utilization for forecasting models (LSTM, GRU, BiLSTM, CNN, CNN-LSTM, and Transformer over the time

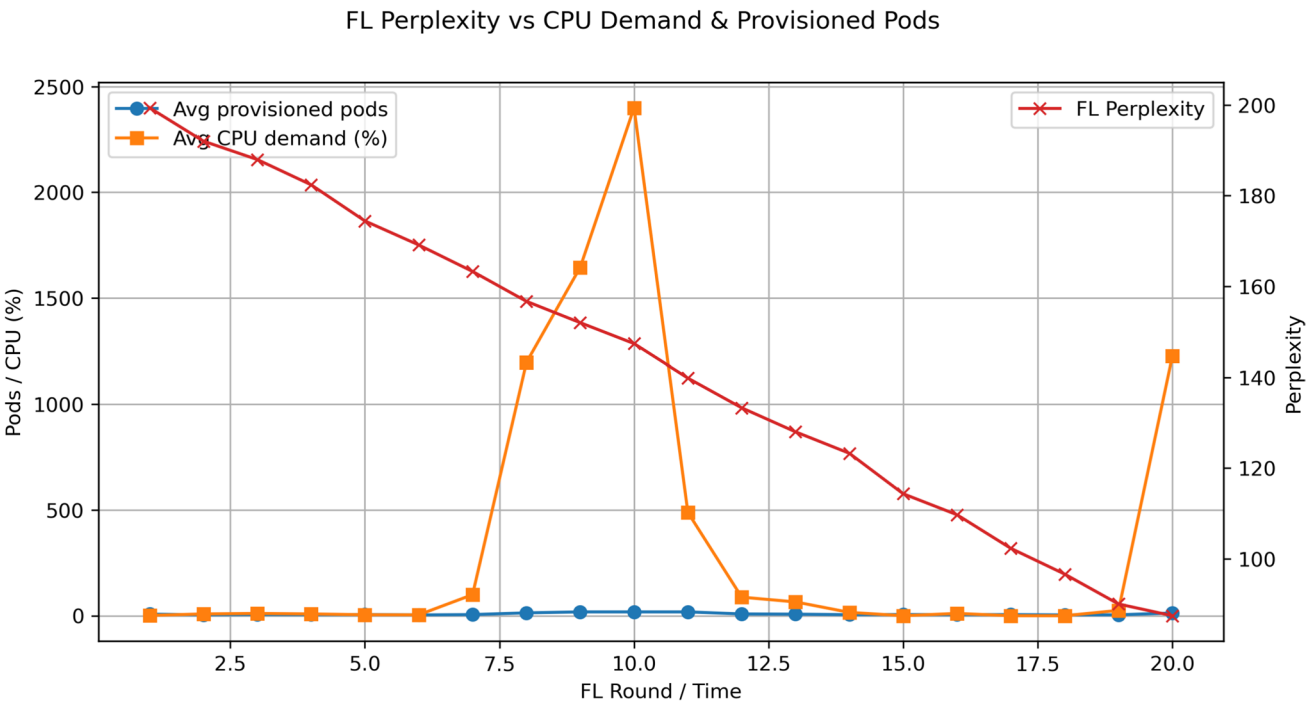


Fig. 6 FL with DP: perplexity versus CPU demand and provisioned pods

blue circles denote the average provisioned pods, the orange curve shows the average CPU demand, and the red line captures FL perplexity.

In Fig. 6, perplexity stabilizes around 200 while reflecting the interaction between workload demand and autoscaling behavior in the absence of DP. By contrast, Fig. 7 demonstrates the DP-enabled case, where perplexity exceeds 200 and peaks at 211.6, reflecting the additional overhead of privacy-preserving training.

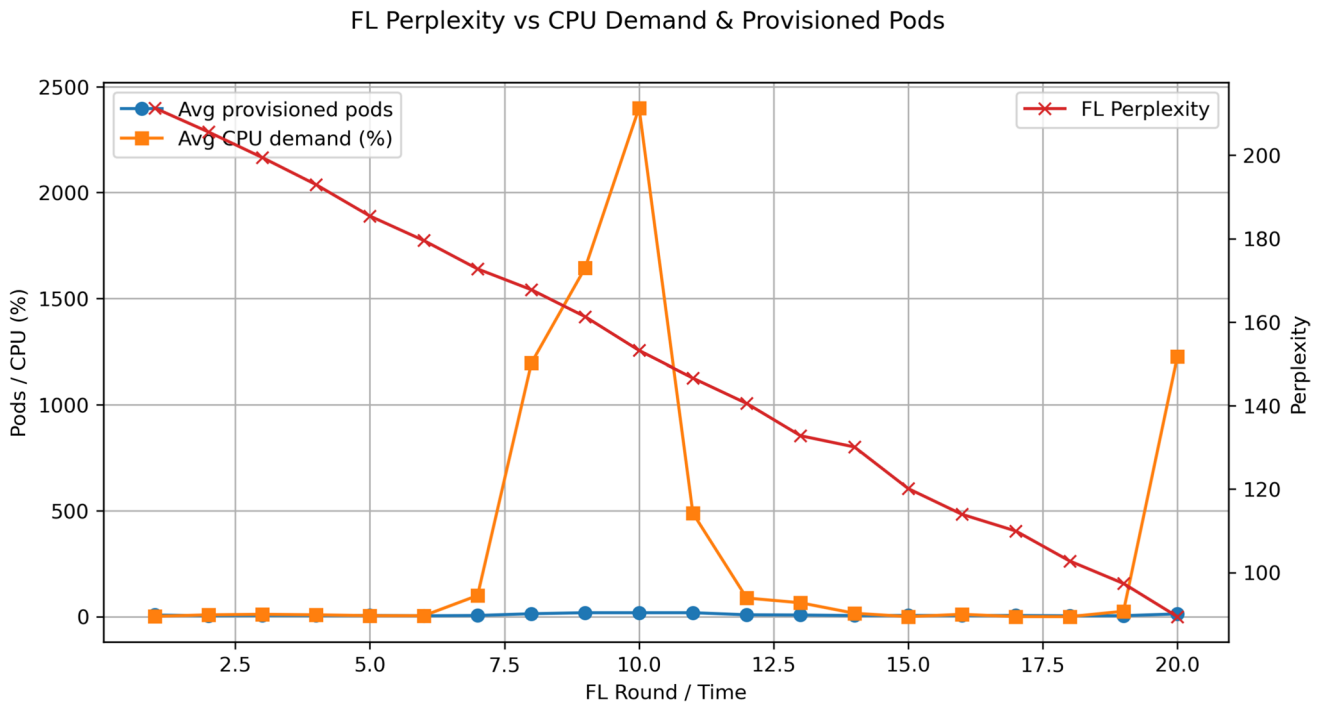
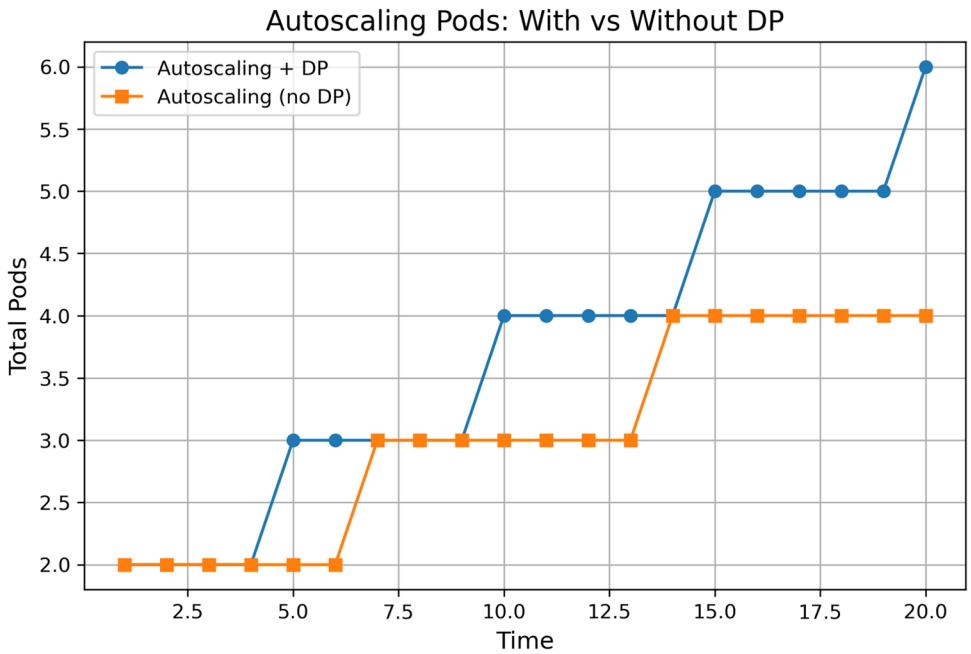


Fig. 7 FL without DP: perplexity versus CPU demand and provisioned pods

Fig. 8 Autoscaling behavior with and without DP in terms of scaling pods over time



These findings highlight that autoscaling through pod assignment adapts effectively to dynamic workloads under both DP and non-DP conditions. Integrating FedALoRA-based LLM training with predictive autoscaling sustains model accuracy while ensuring efficient resource utilization. The mapping from CPU_{t+1} to pod count demonstrates the CNN-LSTM model's capability to respond to fluctuating CPU demand with minimal over- or under-provisioning, thereby forming a solid foundation for proactive autoscaling.

Figure 8 compares autoscaling behavior with and without DP in terms of scaling pods over time. The x-axis represents time in minutes (up to 20 federated rounds with FedALoRA-based LLM training), while the y-axis shows the total number of assigned pods, ranging up to 6.0, with potential increases as scaling adapts dynamically. Autoscaling with DP is shown with blue circles, while autoscaling without DP is depicted with a contrasting orange line.

The results indicate that autoscaling without DP responds slightly faster and requires fewer pods overall. By contrast, the DP-enabled configuration triggers more frequent pod scaling due to the additional overhead introduced by secure aggregation and privacy mechanisms. Nevertheless, both setups demonstrate the effectiveness of autoscaling: the non-DP case highlights faster adaptation and reduced resource usage, while the DP-enabled case ensures stronger privacy protection despite the higher scaling demands.

Figure 9 presents the number of replicas generated through autoscaling across forecasting models such as (LSTM, CNN, CNN-LSTM, GRU, BiLSTM and Transformer), comparing reactive (red color) and proactive (green color) approaches. The x-axis represents time in minutes (0–1250), while the y-axis shows replica counts ranging from 1.0 to 2.0, evaluated at six discrete levels (1.0, 1.2, 1.4, 1.6, 1.8, and 2.0) and the experiment considers ten federated clients per model. The results indicate that CNN-LSTM achieves the most accurate replica scaling, with proactive autoscaling (green) dominating across eight out of ten clients, while only two clients show partial overlap with reactive scaling (red). Similarly, GRU demonstrates proactive superiority in seven clients, with three partially reactive cases. By contrast, LSTM, CNN, BiLSTM, and Transformer exhibit stronger reliance on reactive scaling, underscoring their limitations in predictive adaptation. Overall, CNN-LSTM consistently delivers the most efficient workload-to-replica mapping under dynamic conditions, confirming its suitability for federated edge environments.

Figure 10 extends this analysis by comparing replica scaling with and without DP. The x-axis again represents time (0–1250 min), while the y-axis shows replica counts between 1.0 and 2.0 and the experiment considers ten federated clients per model. The figure distinguishes four cases: reactive (no DP, dashed red line), proactive (no DP, dashed green line), reactive (with DP, solid red line), and proactive (with DP, solid green line). The results show that CNN-LSTM with DP maintains proactive superiority across nine clients, with only one client showing comparable reactive performance. In contrast, LSTM exhibits limited proactive improvements under DP, covering nine clients but lacking consistent predictive adaptation. GRU shows mixed outcomes, with four clients dominated by proactive scaling, four by reactive scaling, and two producing insufficient results. Other models, including CNN, BiLSTM, and Transformer, remain almost fully reactive in DP-enabled settings.

Together, these findings confirm that proactive autoscaling consistently outperforms reactive scaling under both

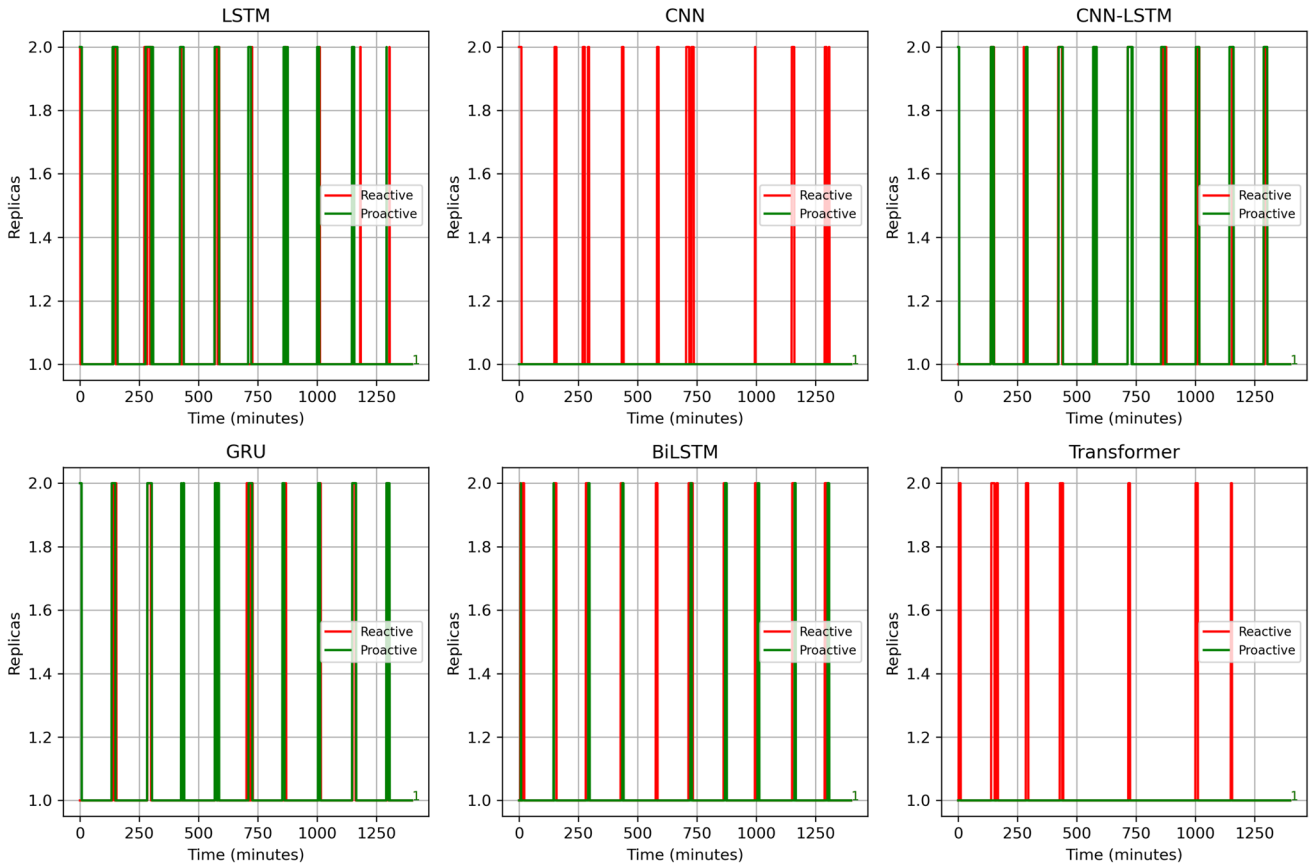


Fig. 9 Replica scaling across models, showing the number of replicas over time for reactive and proactive strategies

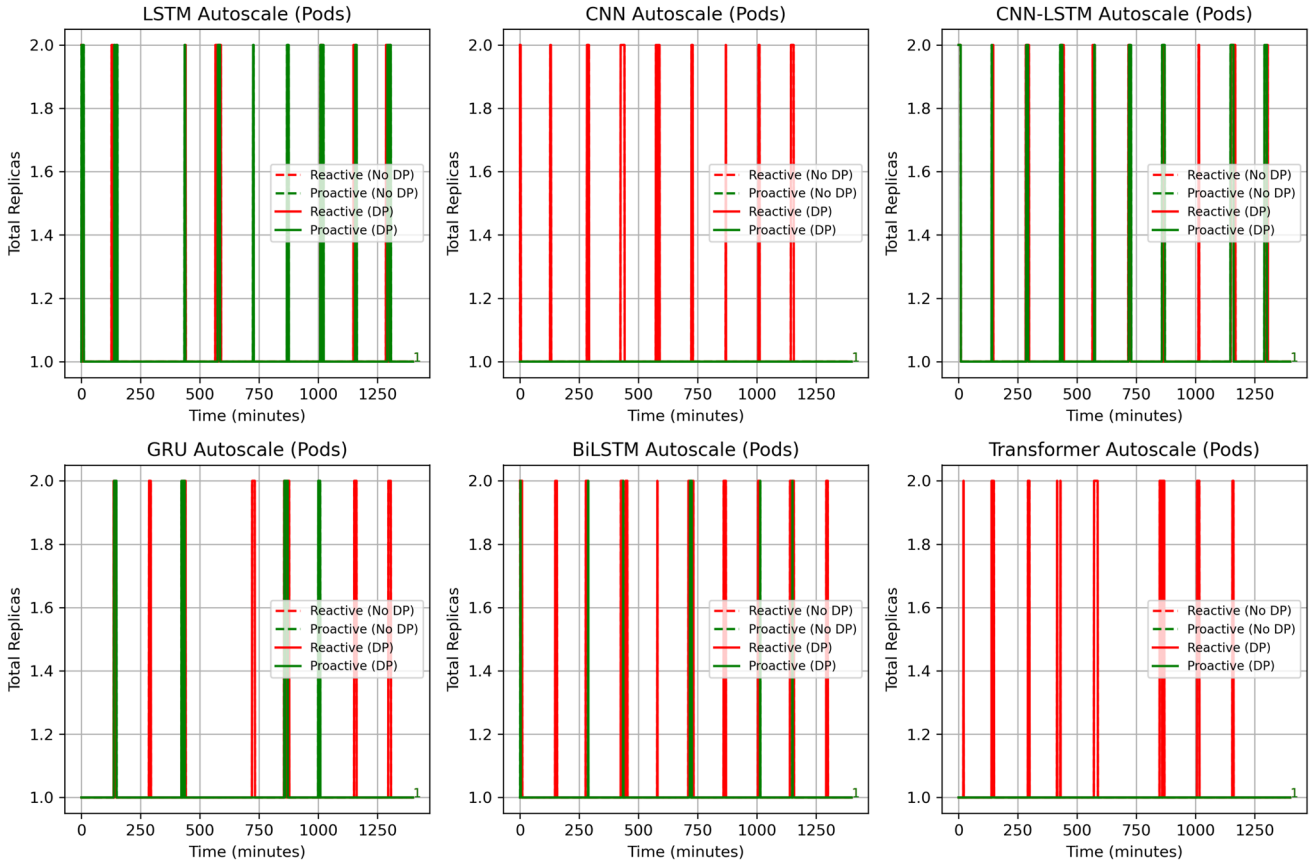


Fig. 10 Replica scaling across models with and without DP, showing the number of replicas over time for reactive and proactive strategies

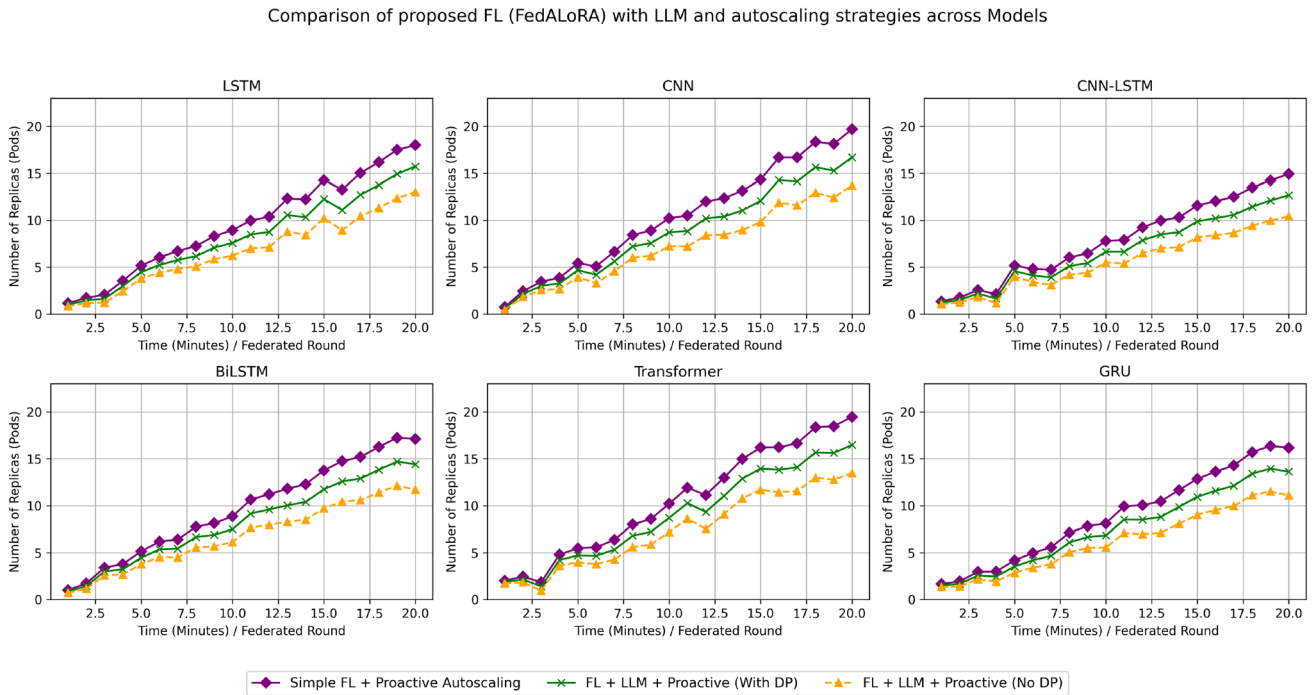


Fig. 11 Comparison of different strategies within the proposed framework: FL with proactive autoscaling, FL with LLM (FedALoRA) and proactive autoscaling with DP, and FL with LLM (FedALoRA) and proactive autoscaling without DP

DP and non-DP conditions. More importantly, integrating FedALoRA-based fine-tuning with CNN-LSTM-driven autoscaling provides robust and scalable performance for federated edge environments, even when DP introduces additional computational and communication overhead.

Figure 11 compares the proposed FL (FedALoRA) with LLM and proactive autoscaling strategies across six models: LSTM, CNN, CNN-LSTM, BiLSTM, Transformer, and GRU. The x-axis represents time in minutes and federated rounds (20), while the y-axis shows the number of replicas (Pods). Three strategies are considered: (i) simple FL with proactive autoscaling (purple), (ii) FL with LLM (FedALoRA) and proactive autoscaling with DP (green), and (iii) FL with LLM (FedALoRA) and proactive autoscaling without DP (orange). The results show that fewer pods lead to better performance with reduced cost and time, whereas higher numbers of replicas increase overhead and degrade autoscaling efficiency in cloud–edge environments. Replica counts across strategies are: LSTM (18, 16, 13), CNN (20, 17, 13), CNN-LSTM (15, 13, 11), BiLSTM (18, 14, 12), Transformer (19, 16, 14), and GRU (16, 14, 12). Among these, CNN-LSTM requires the fewest pods in all cases. While the "No DP" setting achieves better performance than "With DP," the DP-enabled strategy provides privacy preservation and enhanced security at the cost of higher complexity. Overall, pod scaling in combination with federated FedALoRA and LLM offers both performance efficiency and stronger security for cloud–edge environments.

Although the autoscaling mechanism can effectively manage underprovisioning and overprovisioning in isolation, its robustness is preserved when integrated with the proposed FedALoRA-based framework. In the integrated setting, FedALoRA-driven federated LLM training influences the intensity and temporal characteristics of system workloads, while the forecasting-driven autoscaling module continues to regulate resource allocation based on predicted CPU demand. As a result, scaling decisions remain stable and responsive under both DP and non-DP configurations. The use of predictive models together with the resource removal strategy (RRS) prevents abrupt scaling actions, ensuring smooth pod replica adjustments despite the additional computational overhead introduced by federated training.

By explicitly detecting under-provisioning and over-provisioning conditions through predicted CPU demand, the proposed framework dynamically adjusts container and pod allocations, while the resource removal strategy (RRS) proactively releases excess resources to alleviate network bottlenecks and prevent congestion in edge environments.

Overall, the findings demonstrate that proactive autoscaling consistently outperforms reactive autoscaling under both DP and non-DP settings. Without DP, proactive scaling achieves faster adaptation, while with DP it combines stable performance with stronger privacy guarantees, making it a reliable choice for dynamic resource management. Among the forecasting models, CNN-LSTM proved to be the most effective, accurately predicting CPU demand and supporting proactive autoscaling decisions. The replica analysis further shows that CNN-LSTM requires the fewest pods across all scenarios, highlighting its efficiency in resource utilization. The combined evaluation demonstrates that federated fine-tuning with FedALoRA achieves faster convergence and lower perplexity, forecasting models (particularly CNN-LSTM) provide accurate CPU utilization prediction, and proactive autoscaling ensures efficient resource management under dynamic workloads. Together, these findings establish an adaptive and privacy-preserving framework for LLM training and deployment across federated edge environments.

6 Conclusions and future work

The paper proposed an adaptive and privacy-preserving framework that integrates FedALoRA-based fine-tuning, LLM training, and forecasting-driven autoscaling for resource-constrained edge environments. By combining federated fine-tuning with workload forecasting and proactive autoscaling, the framework achieves a balanced trade-off between performance, adaptability, and privacy. Experimental results demonstrated that proactive autoscaling consistently outperforms reactive autoscaling under both DP and non-DP settings. Without DP, proactive scaling achieves faster adaptation, while with DP it combines stable performance with stronger privacy guarantees, making it a reliable choice for dynamic edge environments. Furthermore, CNN-LSTM emerged as the most effective forecasting model, delivering accurate CPU utilization predictions that enable reliable real-time scaling decisions with fewer pods across scenarios. Collectively, these findings validate the effectiveness of the proposed approach in supporting large-scale, efficient, and sustainable federated training of LLMs in cloud–edge environments.

Despite these promising results, several limitations have important implications for real-world deployment. First, the experimental evaluation relies on specific datasets (WikiText-2 for LLM fine-tuning and Bitbrains for workload forecasting), which may not fully capture the diversity, burstiness, and domain-specific characteristics of production edge workloads. Second, the current framework assumes relatively stable communication among federated clients, whereas real-world edge environments often experience heterogeneous network conditions, intermittent connectivity, and partial client failures, which can affect training

stability and aggregation efficiency. Third, while differential privacy strengthens data protection, the associated privacy-accuracy trade-off can become more pronounced as the number of clients and model scale increase, potentially impacting convergence speed and final model quality.

Future work will address these limitations through several targeted directions. First, the framework will be evaluated on larger, more heterogeneous datasets and real-world edge applications to better assess generalization and robustness. Second, communication efficiency will be improved by integrating adaptive gradient and parameter compression techniques, sparsification strategies, and dynamic client participation mechanisms to reduce bandwidth usage and aggregation latency. Third, intelligent client selection and hierarchical aggregation schemes can be explored to improve scalability under unreliable network conditions. Additionally, combining forecasting-driven autoscaling with reinforcement learning-based control policies may enable more context-aware and self-adaptive resource management. Finally, investigating advanced privacy-preserving techniques beyond standard DP, such as secure aggregation and lightweight cryptographic protocols, offers a promising direction to enhance privacy guarantees while minimizing performance degradation.

Author contributions Bablu Kumar: Conceptualization, Methodology, Software, Validation, Investigation, Writing - Original Draft, Visualization. Anshul Verma: Methodology, Resources, Writing - Review and Editing, Supervision, Project administration, Funding acquisition. Rajkumar Buyya: Investigation, Validation, Visualization, Supervision, Writing - Review and Editing.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. This research work is supported by the Institutions of Eminence (IoE) Scheme of Banaras Hindu University, Varanasi, India under Dev. Scheme No. 6031.

Data availability Publicly available databases are used which are available on the following references: WikiText-2 dataset - https://www.kaggle.com/datasets/vivekmettu/wikitext2-data and gwa-bitbrains - https://www.kaggle.com/datasets/gauravdhamane/gwa-bitbrains/data.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Sarthak, A. V., & Verma, P. (2025). An optimal three-tier prioritization-based multiflow scheduling in cloud-assisted smart healthcare. *Journal of Network and Computer Applications, 238*, Article 104143.
2. Shi, W., Cao, J., Zhang, Q., Li, Y., & Lanyu, X. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal, 3*(5), 637–646.
3. Satyanarayanan, M. (2017). The emergence of edge computing. *Computer, 50*(1), 30–39.
4. Sagar, S., Mahmood, A., Sheng, Q. Z., Zhang, W. E., Zhang, Y., & Pabani, J. K. (2024). Understanding the trustworthiness management in the social internet of things: A survey. *Computer Networks, 251*, Article 110611.
5. Kumar, B., Singh, M., Verma, A., & Verma, P. (2023). Optimal cloudlet selection in edge computing for resource allocation. *SN Computer Science, 4*(6), 745.
6. Kafunah, J., Verma, P., Ali, M. I., & Breslin, J. G. (2023). Out-of-distribution data generation for fault detection and diagnosis in industrial systems. *IEEE Access, 11*, 135061–135073.
7. Zhang, M., Shen, X., Cao, J., Cui, Z., & Jiang, S. (2024). Edgeshard: Efficient llm inference via collaborative edge computing. *IEEE Internet of Things Journal, 12*(10), 13119–13131.
8. Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16.
9. Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials, 19*(4), 2322–2358.
10. Qu, G., Chen, Q., Wei, W., Lin, Z., Chen, X., & Huang, K., (2025). Mobile edge intelligence for large language models: A contemporary survey. *IEEE Communications Surveys & Tutorials, 27*(6), 3820–3860.
11. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM, 59*(5), 50–57.
12. Dogani, J., & Khunjush, F. (2024). Proactive auto-scaling technique for web applications in container-based edge computing using federated learning model. *Journal of Parallel and Distributed Computing, 187*, Article 104837.
13. Kumar, B., Verma, A., & Verma, P. (2026). Critical insights into runtime scheduling, image, storage, and networking challenges in modern Kubernetes environments. *Computer Science Review, 59*, Article 100851.
14. Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing, 12*(4), 559–592.
15. Coutinho, E. F., Rubens, F., de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes, & José Neuman de Souza. (2015). Elasticity in cloud computing: a survey. *Annals of telecommunications-annales des télécommunications,70*(7), 289–309.
16. Jeong, B., & Jeong, Y.-S. (2025). Autoscaling techniques in cloud-native computing: A comprehensive survey. *Computer Science Review, 58*, Article 100791.

17. Bharot, N., Verma, P., Soderi, M., & Breslin, J. G. (2024). Dq-deeplearn: Data quality driven deep learning approach for enhanced predictive maintenance in smart manufacturing. *Procedia Computer Science, 232*, 574–583.
18. Kumar, B., Verma, A., & Verma, P. (2024). Optimizing resource allocation using proactive scaling with predictive models and custom resources. *Computers and Electrical Engineering, 118*, Article 109419.
19. Kumar, B., Verma, A., & Verma, P. (2025). A multivariate transformer-based monitor-analyze-plan-execute (mape) autoscaling framework for dynamic resource allocation in cloud environment. *Computing, 107*(3), 69.
20. Kumar, B., Verma, A., Verma, P., & Bennour, A. (2025). Optimizing resource allocation in cloud-native applications through proactive autoscaling with the informerautoscale model: B. kumar, et al. *The Journal of Supercomputing*,*81*(9), 1077.
21. Wu, F., Li, Z., Li, Y., Ding, B., & Gao, J. (2024). Fedbiot: Llm local fine-tuning in federated learning without full model. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3345–3355.
22. Kuang, W., Qian, B., Li, Z., Chen, D., Gao, D., Pan, X., Xie, Y., Li, Y., Ding, B., & Zhou, J. (2024). Federatedscope-llm: A comprehensive package for fine-tuning large language models in federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5260–5271.
23. McMahan, B., Moore, E., Ramage, D., Hampson, S., & Arcas, B.A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR.
24. Dritsas, E., & Trigka, M. (2025). Federated learning for iot: A survey of techniques, challenges, and applications. *Journal of Sensor and Actuator Networks, 14*(1), 9.
25. Bharot, N., Mehta, N., John, G., Breslin, P., & Verma. (2025). Cloudlock: Secure data sharing using a hybrid cryptosystem in multi-cloud data storage. *Cluster Computing, 28*(7), 464.
26. Verma, P., Vidyarthi, A., Mishra, A., Ali, J., Bharot, N., & Breslin, J. G. (2025). A secure adversarial attack detector framework for monitoring network intrusion in iot environment. *IEEE Transactions on Consumer Electronics, 71*(4), 11613–11623.
27. Verma, P., Tapaswi, S., Wilfred, W., & Godfrey. (2021). An impact analysis and detection of http flooding attack in cloud using bio-inspired clustering approach. *International Journal of Swarm Intelligence Research (IJSIR),12*(1), 29–49.
28. Zhi Qiang Wang, H., Wang, A. E., & Saddik. (2024). Feditd: A federated parameter-efficient tuning with pre-trained large language models and transfer learning framework for insider threat detection. *IEEE Access, 12*, 160396–160417.
29. Yi, X., Hu, C., & Cai, B. (2025). Fedalora: Adaptive local lora aggregation for personalized federated learning in llm. In *International Conference on Wireless Artificial Intelligent Computing Systems and Applications*, pp. 86–95. Springer.
30. Hai, W., Chen, X., & Huang, K. (2024). Device-edge cooperative fine-tuning of foundation models as a 6g service. *IEEE Wireless Communications, 31*(3), 60–67.
31. Parra-Ullauri, J. M., Madhukumar, H., Nicolaescu, A.-C., Zhang, X., Bravalheri, A., Hussain, R., Vasilakos, X., Nejabati, R., & Simeonidou, D. (2024). kubeFlower: A privacy-preserving framework for Kubernetes-based federated learning in cloud-edge environments. *Future Generation Computer Systems, 157*, 558–572.
32. Ju, L., Singh, P., & Toor, S. (2021). Proactive autoscaling for edge computing systems with kubernetes. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, pp. 1–8.
33. Dang-Quang, N.-M., & Yoo, M. (2021). Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. *Applied Sciences, 11*(9), 3835.
34. Abreha, H. G., Hayajneh, M., & Serhani, M. A. (2024). Federated learning in edge computing: a systematic survey. *Sensors, 22*(2), 450.
35. Brecko, A., Kajati, E., Koziorek, J., & Zolotova, I. (2022). Federated learning for edge computing: A survey. *Applied Sciences, 12*(18), 9124.
36. Trindade, S., Luiz, F., Bittencourt, N. L. D., & Fonseca. (2024). Resource management at the network edge for federated learning. *Digital Communications and Networks, 10*(3), 765–782.
37. Luo, L., Zhang, C., Yu, H., Sun, G., Luo, S., & Dustdar, S. (2024). Communication-efficient federated learning with adaptive aggregation for heterogeneous client-edge-cloud network. *IEEE Transactions on Services Computing, 17*(6), 3241–3255.
38. Zhang, B., Mao, Y., He, X., Ping, P., Huang, H., & Wu, J. (2025) Exploring the privacy-accuracy trade-off using adaptive gradient clipping in federated learning. *IEEE Transactions on Network Science and Engineering, 12*(3), 2254–2265.
39. Jin, Z., Xu, C., Wang, Z., & Sun, C. (2025). Towards robust differential privacy in adaptive federated learning architectures. *IEEE Transactions on Consumer Electronics, 71*(2), 4087–4099.
40. Yang, Y., Long, G., Lu, Q., Zhu, L., Jiang, J., & Zhang, C. (2025). Federated low-rank adaptation for foundation models: A survey arXiv:2505.13502 arXiv preprint.
41. Raje, A., Baris Askin, D., Jhunjhunwala, G., & Joshi. (2025). Ravan: Multi-head low-rank adaptation for federated fine-tuning arXiv:2506.05568 arXiv preprint.
42. Wang, Z., Zhou, Y., Shi, Y., & Letaief, K.B. (2024). Federated low-rank adaptation for large language model fine-tuning over wireless networks. In *GLOBECOM 2024-2024 IEEE Global Communications Conference*, pp. 3063–3068. IEEE.
43. Yang, H., Liu, H., Yuan, X., Kai, W., Wei Ni, J., Zhang, A., & Liu, R. P. (2025). Synergizing intelligence and privacy: A review of integrating internet of things, large language models, and federated learning in advanced networked systems. *Applied Sciences, 15*(12), 6587.
44. Shaby, S.M., Gomathi, T., & Saroo Raj RB, et al. (2025). Predictive maintenance in the internet of vehicles using transformer-based deep learning models. In *2025 International Conference on Networks and Cryptology (NETCRYPT)*, pp. 201–206. IEEE.
45. Taha, M. B., Sanjalawe, Y., Al-Daraiseh, A., Fraihat, S., & Al-E'mari, S. R. (2024). Proactive auto-scaling for service function chains in cloud computing based on deep learning. *IEEE Access, 12*, 38575–38593.

Bablu Kumar received an M.C.A. degree in Computer Science and Engineering from Pondicherry University, Kalapet, Puducherry. He qualified for JRF in 2021 and is currently a research scholar under the supervision of Dr. Anshul Verma in the Department of Computer Science at the Institute of Science, Banaras Hindu University, Varanasi (India). His primary research interests include Cloud Computing, Edge Computing, Autoscaling and Resource Allocation, Distributed Systems, Deep Learning, Federated Learning, and Large Language Models (LLMs).

Anshul Verma received M.Tech. and Ph.D. degrees in Computer Science and Engineering from ABV-Indian Institute of Information Technology and Management Gwalior, India. He has done Postdoctorate from Indian Institute of Technology Kharagpur, India. He is serving as an Assistant Professor in the Department of Computer Science, Institute of Science, Banaras Hindu University Varanasi, India and is presently on study leave, serving as a Visiting Researcher at the School of Computing and Information Systems, University of Melbourne, Australia. He has also served as a faculty member in the Computer Science and Engineering Department at Motilal Nehru National Institute of Technology (MNNIT) Allahabad and National Institute of Technology (NIT) Jamshedpur, India. His research interests include Cloud Computing, Edge Computing, Distributed Systems, Mobile ad-hoc Networks, and Formal Verification. He is serving as Associate Editor of the Journal of Scientific Research of the Banaras Hindu University and as Editorial Board Member of Scientific Reports journal of Springer.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory at the University of Melbourne, Australia. He served as a Future Fellow of the Australian Research Council during 2012–2016. He has authored over 850 publications, seven text books, and several edited books. He is one of the highly cited authors in computer science and software engineering worldwide (hindex: 177, g-index: 384, 167800+ citations). Dr. Buyya is recognized as a "Web of Science Highly Cited Researcher" for seven times since 2016, a Fellow of IEEE, Foreign Fellow of Academia Europaea, Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier. He has been recognised as the "Best of the World" twice for research fields (in Computing Systems in 2019 and Software Systems in 2021) as well as "Lifetime Achiever" and "Superstar of Research" in "Engineering and Computer Science" discipline twice (2019 and 2021) by the Australian Research Review. Recently, he received "Research Innovation Award" from IEEE Technical Committee on Services Computing and "Research Impact Award" from IEEE Technical Committee on Cloud Computing.