FedForecaster: An Automated Federated Learning Approach for Time-series Forecasting

Giza Systems

FedForecaster: An Automated Federated Learning Approach for Time-series Forecasting

Mohamed Maher*
Innovation Hub, Giza Systems
mohamed.maher@gizasystems.com

Osama Fayez Oun*
Innovation Hub, Giza Systems
osama.fayez@gizasystems.com

Mahmoud Saeed Mesmeh*
Innovation Hub, Giza Systems
mahmoud.mesmesh@gizasystems.com

Radwa ElShawi
Institute of Computer Science, University of Tartu
radwa.elshawi@ut.ee

ABSTRACT

This paper introduces `FedForecaster`, a novel automated machine learning (AutoML) engine for univariate time-series forecasting in federated learning (FL) environments. Our engine addresses the challenge of automating the full pipeline of time-series forecasting, including feature engineering, algorithm selection, and hyperparameter tuning, without centralized data collection. Leveraging a meta-model trained on a diverse knowledge base of synthetic and real univariate time-series datasets, the engine recommends the optimal forecasting algorithms based on statistical meta-features aggregated across multiple clients. Bayesian optimization is subsequently applied to refine the search space, optimizing performance within the constraints of federated learning environments. Our solution outperforms baseline approaches, including random search and N-beats model, as demonstrated in evaluations across various domains. We utilize the Flower framework to implement and evaluate our approach, highlighting its potential to scale and adapt across different client distributions and data types.

1 INTRODUCTION

Currently, machine learning (ML) is experiencing a paradigm shift from centralized cloud data centres to distributed edge computing environments [16]. With the advancement of mobile Internet of Things (IoT) [35], a substantial amount of valuable time series data is generated by distributed smart devices. Time series data consists of a sequence of data points organized in chronological order and has been widely applied in various domains, like anomaly detection [13, 15], and weather forecasting [6]. One of the major research areas in this field is time series forecasting [19]. However, the traditional process of building forecasting models is often labor-intensive and requires expert knowledge in feature engineering, algorithm selection, and hyperparameter tuning [32]. Additionally, the sensitivity and privacy concerns associated with users' data pose significant challenges for centralized model training. Traditional centralized ML approaches require all data to be aggregated on a central server for training, which not only increases the overhead of data transmission but also heightens the risk of privacy breaches.

To address the challenges of data confidentiality and communication efficiency, federated learning (FL) [1, 22, 34] has emerged as a promising distributed training paradigm. FL enables collaborative training on large, multi-source datasets without exchanging original data, thus preserving data privacy [17] while reducing communication overhead [21]. Specifically, edge devices retain their private data locally, and FL primarily achieves the training of a robust model through the aggregation and distribution of local models across multiple rounds of communication. FL holds significant potential for enabling collaborative model training; however, it encounters substantial challenges related to data heterogeneity in practical applications [22].

Achieving optimal performance in FL environments significantly relies on the careful optimization of hyperparameters [23, 37]. In contrast to traditional centralized settings, hyperparameter tuning in FL is further complicated by the distinctive characteristics of distributed environments, where data frequently exhibits non-IID properties. While hyperparameter optimization (HPO) is well-studied in centralized settings [11], FL presents distinct challenges due to limited communication. In FL settings, hyperparameters are adjusted dynamically across communication rounds to account for variations in local client data and global model aggregation [14]. Moreover, FL's privacy constraints further complicate the evaluation and tuning of hyperparameters across decentralized nodes.

The motivation behind this work stems from the need for a fully automated, privacy-preserving time-series forecasting engine that can be deployed in FL settings. Our research aims to address this gap by proposing `FedForecaster`, an automated machine-learning engine designed specifically for univariate time-series forecasting in federated environments. `FedForecaster` leverages a meta-model trained on a knowledge base of univariate time-series datasets to recommend the best-performing forecasting algorithms for any given dataset. This recommendation is based on statistical meta-features aggregated across all clients, ensuring that data privacy is maintained throughout the process.

The automation provided by `FedForecaster` simplifies the process of time-series forecasting by enhancing the scalability of FL systems, enabling forecasting model deployment across distributed clients. By integrating Bayesian optimization into the engine, we further refine the hyperparameter tuning process, allowing for more efficient exploration of the algorithmic search space. `FedForecaster` is evaluated on diverse time-series datasets, and the results show its superiority in terms of forecasting accuracy and efficiency when compared to baseline approaches like random search and the `N-beats` [26] model.

This work introduces the following list of **contributions**:

- We propose `FedForecaster`, a fully automated machine-learning engine for time-series forecasting in federated learning (FL) environments. To the best of our knowledge, this is the first work that investigates a fully automated

*Authors contributed equally to this research.

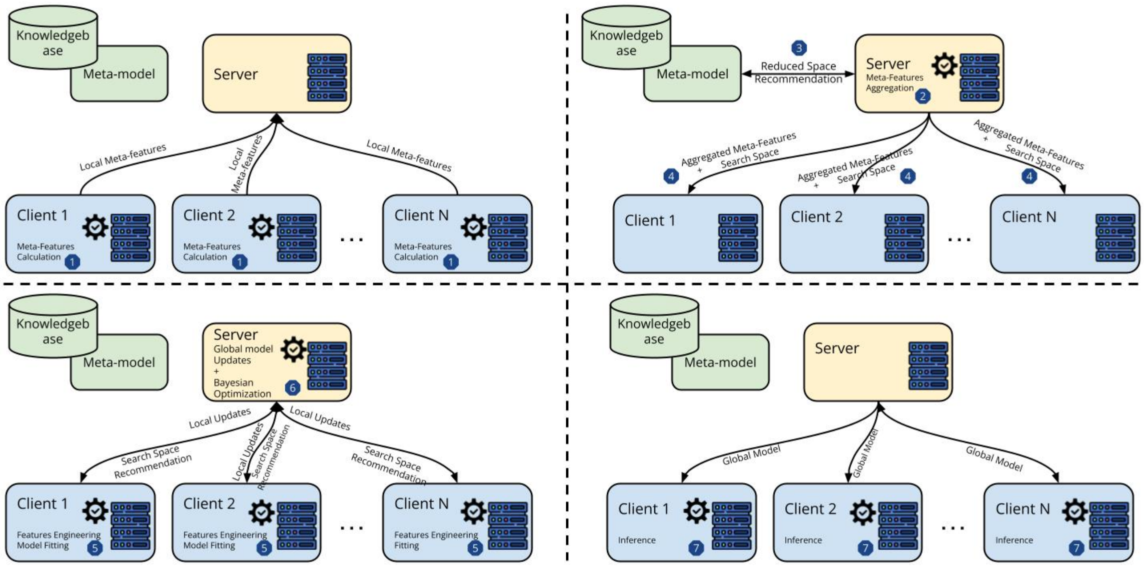

Figure 1: Overview of the FedForecaster framework. I) Meta-features are computed over each client. II) The centralized server aggregates the meta-features and recommends a search space using a meta-learning approach. III) The server recommends model instantiations alongside the aggregated meta-features to clients. The clients use the aggregated meta-features to perform automated feature engineering and fit the recommended model. The server computes the global loss and uses Bayesian optimization for the next model with hyper-parameters in an iterative way. IV) The server aggregates the local models with the best global performance to be deployed on clients for inference.

approach, encompassing the entire pipeline of time-series forecasting, including feature engineering, algorithm selection, and hyperparameter tuning.

- `FedForecaster` utilizes a novel meta-learning approach that employs globally aggregated meta-features to recommend the most promising forecasting algorithms. We integrate a Bayesian optimization approach on the server side to enable efficient hyperparameter tuning with reduced communication rounds, thereby enhancing both model accuracy and efficiency.
- We apply a methodology for a unified time-series feature engineering across the clients given the globally collected meta-features across the data splits.
- We provide extensive empirical evaluations across diverse datasets and FL scenarios, demonstrating the scalability and effectiveness of `FedForecaster` compared to baseline approaches like random search and `N-beats`.

This paper is structured as follows: Section 2 reviews the related work, highlighting key limitations that motivate the development of `FedForecaster`. Section 3 formulates the problem addressed by `FedForecaster`, outlining its objectives. Section 4 describes the architecture of `FedForecaster`, detailing its key components. Section 5 presents the empirical evaluation, including the experimental setup and performance analysis. Finally, Section 6 provides a conclusion and future research directions.

2 RELATED WORK

Time-series forecasting is a critical task across industries, requiring sophisticated techniques to model temporal dependencies in data [3]. However, traditional approaches to time-series forecasting have predominantly relied on centralized datasets, limiting their applicability in privacy-sensitive environments such as FL [20]. While centralized solutions like ARIMA and LSTMs have demonstrated strong performance in time-series forecasting, these models depend on access to aggregated data, which is infeasible in FL environments due to privacy constraints [39].

FL enables ML model training across distributed clients without centralized data aggregation. Though much of the existing research in FL focuses on general ML tasks such as classification and regression, relatively few works have addressed time-series forecasting in FL settings [38]. Recurrent neural networks (RNNs) are adapted for time-series forecasting across federated clients [18]. Similarly, `FedATM` employed temporal attention mechanisms for anomaly detection in time-series data [24]. Both studies show that deep learning models can be effectively used for time-series forecasting in FL environments. However, they do not focus on automating model selection or tuning hyperparameters.

Table 1: Suggested Meta-Features for Federated Time-Series Forecasting, their types [time-series (TS), statistical (Stat.)] and Aggregation Methods

Meta-Feature	Type	Aggregation Method
No. of Clients	Stat.	NA
Sampling Rate	TS	NA
No. of Instances	Stat.	Sum, Avg, Min, Max, Stddev
Target Missing Values %	Stat.	Avg, Min, Max, Stddev
Stationary Features	TS	Avg, Min, Max, Stddev
Target Stationarity	TS	Entropy across clients
Stationary Features after 1st Order Diff	TS	Avg, Min, Max, Stddev
Stationary Features after 2nd Order Diff	TS	Avg, Min, Max, Stddev
Significant Lags using pACF	TS	Avg, Min, Max, Stddev
Insignificant lags between 1st and last significant ones	TS	Avg, Min, Max, Stddev
Detected seasonality components	TS	Avg, Min, Max, Stddev
Skewness	Stat.	Avg, Min, Max, Stddev
Kurtosis	TS	Avg, Min, Max, Stddev
Fractal dimension analysis of target	Stat.	Avg
Periods of seasonality components	TS	Min, Max
KL Div. among clients' distribution	Stat.	Avg, Min, Max, Stddev

In FL environments, the heterogeneity of time-series data across clients poses additional challenges, as data distributions can vary significantly. This makes it difficult to develop a universal forecasting model that works well for all clients [31]. While centralized forecasting models benefit from consistent data distributions, FL scenarios require models that can adapt to the specific statistical characteristics of client data without violating privacy. Existing FL approaches for time-series forecasting focus on adapting specific neural network architectures but do not explore broader algorithm selection or automated solutions for heterogeneous client environments [5, 28].

AutoML has emerged as a powerful tool for automating the process of model selection and hyperparameter tuning [7, 8]. AutoML platforms such as `Auto-sklearn` [10] and `TPOT` [25] have been successfully applied in centralized contexts, offering significant time and resource savings. Other platforms have been developed specifically for time-series forecasting tasks like `AutoGluon` [9] and `GizaML` [29]. However, these platforms assume access to a centralized dataset, rendering them unsuitable for FL environments.

Recent research has started to explore AutoML in FL. For instance, `FedNAS` automates neural architecture search in FL settings but does not specifically address the challenges of time-series forecasting [12]. `FLASH` introduced an AutoML framework for the combined algorithm selection and hyperparameter tuning problem in the FL settings [2]. Existing AutoML methods in FL focus primarily on model training automation and hyperparameter tuning for general-purpose tasks [27], with little attention to time-series data, which requires specialized techniques for feature engineering and algorithm selection.

To the best of our knowledge, no AutoML solutions are specifically designed for time-series forecasting in FL settings. The lack of such solutions represents a gap in the current research landscape, as time-series data is particularly prevalent in industries that require distributed, privacy-preserving data analysis. `FedForecaster` fills this gap by introducing a fully automated engine that addresses time-series forecasting in FL, automating the entire pipeline from feature engineering to algorithm selection and hyperparameter tuning.

3 PROBLEM FORMULATION

We formulate the problem that `FedForecaster` addresses as follows. Given a set of machine learning forecasting algorithms $\mathbf{A} = \{A^{(1)}, A^{(2)}, ...\}$, and a time-series federated dataset $\mathbf{D}$ across N clients, where client j has a private data split $D_j = D_j^{train} \cup D_j^{valid}$ such D_j^{train} and D_j^{valid} are the training and validation time-series splits, respectively. The goal is to find the best-performing algorithm $A_\lambda^{(i)}$ with hyper-parameter configuration $\lambda \in \Lambda$ that minimizes the global aggregated loss L across all clients within a time budget T. Equation 1 summarizes the problem formulation such that $\alpha_j = \frac{|D_j|}{|\mathbf{D}|}$ represents the weight of client j loss value in the aggregated global loss.

$$A_\lambda^{(i)*} = \underset{A \in \mathbf{A}, \lambda \in \Lambda}{\arg\min} \sum_{j}^{N} \alpha_j L((A_\lambda^{(i)}, D_j^{train}), D_j^{valid}) \qquad (1)$$

4 METHODOLOGY

In this section, we present the methodology of the `FedForecaster` framework. Figure 1 illustrates the architecture of `FedForecaster`. The framework primarily comprises three phases: (a) the meta-learning phase (Section 4.1), (b) the feature engineering phase (Section 4.2), and (c) the optimization phase (Section 4.3). In the meta-learning phase, various instantiations of the machine-learning models are proposed based on globally aggregated meta-features that characterize the input dataset. These meta-features are communicated with the clients to support the feature extraction decisions and automate the feature engineering stage. The recommended instantiations are likely to yield strong performance and serve as a warm start to the optimization process, which employs Bayesian optimization for hyperparameter tuning.

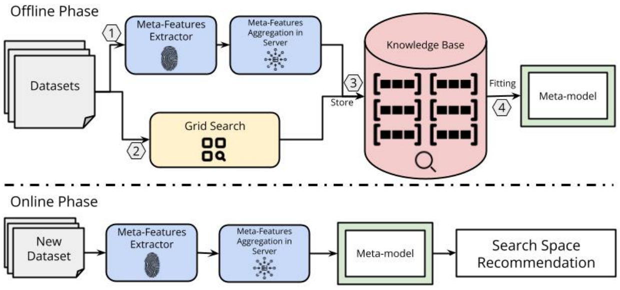

Figure 2: Offline Phase: A meta-model is fitted on a knowledge base to recommend algorithm instantiations. The knowledge base is constructed from the meta-features of a collection of datasets along with the best forecasting algorithm over each dataset obtained after applying grid search. Online Phase: The meta-features are extracted from the clients' data splits. The meta-model is used to recommend algorithm instantiations given the aggregated meta-features.

4.1 Meta-Learning Phase

We utilize a meta-learning approach to identify machine learning algorithm instantiations that are expected to perform effectively on new datasets. This meta-learning process consists of two phases: an offline phase, where a meta-model is trained using a diverse set of datasets, and an online phase, where the meta-model is applied to recommend suitable algorithm instantiations for the target dataset. The overall methodology is illustrated in Figure 2, and detailed in Algorithm 1.

4.1.1 Offline Phase. We use a collection of time-series datasets from the `GizaML` knowledge base [29]. The knowledge base consists of 512 synthetic datasets and 30 real-world datasets obtained from various open data platforms, including Kaggle[1] and the Nasdaq stock market[2]. The synthetic datasets were generated by varying several factors, such as seasonality components, sampling frequencies, signal-to-noise ratios, the percentage of missing values, and the nature of the signal components (additive or multiplicative). These variations aim to capture a wide range of meta-features that describe the characteristics of time-series data. To simulate FL environments, we split the datasets randomly into 5, 10, 15, or 20 clients with time-series splits, ensuring that each client receives at least 500 instances per split. If a dataset does not meet this minimum instance threshold, it is excluded from the knowledge base.

[1] https://kaggle.com/
[2] https://www.nasdaq.com

The core of our knowledge base consists of a collection of synthetic and real datasets, from which we extract statistical and time-series meta-features for each dataset. These meta-features capture various properties, including general trend, seasonality, and stationarity, as detailed in Table 1. These meta-features serve as the "fingerprint" of the time-series data, helping to identify the best-performing forecasting algorithms. It is important to note that the collected meta-features are anonymized, ensuring no sensitive data is shared across clients. Only the statistical properties of the data are aggregated without centralizing the full dataset. The server then aggregates and stores the meta-features from all clients in the knowledge base.

For each dataset in the knowledge base, we conduct a comprehensive grid search over a predefined search space of forecasting algorithms and hyperparameter configurations. The goal is to identify the best-performing algorithm and its corresponding hyperparameters. The search space of algorithms and hyperparameters used in this grid search is detailed in Table 2. A meta-model is trained on the knowledge base to recommend the best algorithms given the aggregated meta-features.

4.1.2 Online Phase. The meta-features are computed over each client split. Then, the server aggregates all these meta-features across the clients and feeds them into the pre-trained meta-model, which recommends algorithms that are most likely to perform well on the federated dataset. This recommendation serves as the warm initialization for the hyperparameter tuning phase (Algorithm 1: lines 3-10).

4.2 Feature Engineering

The feature engineering process is crucial for improving the predictive performance of time-series models in `FedForecaster`. Initially, linear interpolation is applied to handle any missing value gaps in the time-series data.

4.2.1 Feature Extraction. Given the aggregated meta-features from the server, each client independently extracts features from their local time-series data (lines 11-13) as follows:

1. **Trend Feature**: The trend component is extracted by first applying the Augmented Dickey-Fuller test to determine the stationarity of the time-series. Depending on whether the time-series is stationary, linear, or logistic, a `Prophet` model [30] is fitted to estimate the trend component.
2. **Time Features**: Temporal features such as day of the week, hour of the day, and month of the year are extracted to capture periodicity in the data.
3. **Lag Features**: The statistically significant global lags are computed using the partial autocorrelation function (ACF). The number of lags is determined by the maximum count of significant lags identified during the meta-feature calculation stage across all clients.
4. **Seasonality Features**: The top N seasonal components are extracted using a weighted periodogram across all clients. The most important seasonalities are included in the feature set.

While some extracted features, such as seasonality components and significant lags are already extracted during the meta-features extraction phase (Table 1), new features like time and trend components are not used in the meta-features.

4.2.2 Feature Selection. Each client computes the feature importance scores using a Random Forest regressor. The aggregated average feature importance scores are evaluated in the server, aiming to keep only the most important features contributing to 95% of the sum of feature importance scores and discarding the less important ones to reduce the dataset dimensionality.

4.3 Hyperparameter Tuning

The hyperparameter optimization process in `FedForecaster` leverages Bayesian optimization to efficiently tune forecasting algorithms across federated clients (Lines 14-22). The algorithm instantiations recommended by the meta-model serve as the initialization for Bayesian optimization, which is evaluated locally by each client on their validation sets. The resulting local losses are aggregated by the server to compute a global loss value. The global feedback is used to update the surrogate model, enabling it to balance exploration of new configurations with exploitation of promising ones. Through iterative updates, the server refines its recommendations, ensuring that each subsequent set of configurations is informed by prior results and aimed at improving performance. The process continues until a predefined time budget is exhausted, optimizing the average aggregated performance across clients.

4.4 Inference

Once the globally optimized hyperparameters are identified, they are shared with all clients, who then use these configurations to train their final models on local data (Lines 23:25). This approach efficiently navigates large search spaces while minimizing computational burden. The server then aggregates the locally trained models to generate the final federated forecast (Lines 26:27). This approach ensures that hyperparameter optimization is efficient and privacy-preserving while maintaining the global optimization objectives.

Table 2: Search Space for Forecasting Algorithms in `FedForecaster`

Algorithm	Hyperparameters	Values
Lasso Regressor	alpha	$(log(e^{-5}), log(10))$
	selection	{cyclic, random}
LinearSVR Regressor	C	$[1 : 10]$
	epsilon	$[0.01 : 0.1]$
ElasticNetCV Regressor	l1_ratio	$[0.3 : 10]$
	selection	{cyclic, random}
XGB Regressor	n_estimators	$[5 : 20]$
	max_depth	$[2 : 10]$
	learning_rate	$[0.01 : 1]$
	reg_lambda	$[0.8 : 10]$
	subsample	$[0.1 : 1]$
Huber Regressor	epsilon	{1.0, 1.35, 1.5}
	alpha	$[log_{10}(e^{-3}) : log_{10}(e^{2})]$
Quantile Regressor	alpha	$[log_{10}(e^{-3}) : log_{10}(e^{2})]$
	quantile	$[0.1 : 1]$

5 EMPIRICAL EVALUATION

5.1 Experimental Setup

We present the experimental evaluation of `FedForecaster` in comparison to the `N-beats` algorithm implemented in a federated context and random search within an FL setting. All experiments were conducted using `Flower` framework [4], and the source code is publicly available[3]. Each method was allocated a maximum time budget of $T = 5$ minutes for the hyperparameter

[3] https://github.com/giza-data-team/FedForecaster

Algorithm 1 Federated AutoML Framework (FedForecaster)

Input: Time-series data splits at clients $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_j$, pre-trained recommendation meta-model R at server, and Time budget T or Number of iterations I
Output: Best global model with hyperparameters $\hat{A}_{\lambda}^{(i)}$
for each client j **do**
  Split time-series data into training and validation sets. $D_j = D_j^{train} \cup D_j^{valid}$
  Compute statistical meta-features from client data splits
  Send meta-features to server
end for
Server aggregates meta-features from all clients
Server feeds aggregated meta-features into meta-model R
Meta-model R recommends a search space of forecasting algorithms $\mathbf{A'} \subset \mathbf{A}$
for each client j **do**
  Perform feature engineering on time-series data splits (see Subsection 4.2)
end for
Initialize Bayesian optimization with recommended search space ($\mathbf{A'}$) in the server
for Time Budget T OR Number of iterations I **do**
  Server recommends the next algorithm with hyperparameters configuration $A'_{\lambda} \in \mathbf{A'}$ using Bayesian Optimization to clients
  for each client j **do**
    Fit the model (A'_{λ_j}) on client j data split
    Send the fitted local model updates and performance to the server
  end for
  Server aggregates the global loss L
end for
for each client j **do**
  Final model and hyperparameter configuration with the best global performance ($\hat{A}_{\lambda_j}$) is fitted on client split.
end for
Server aggregates local models ($\hat{\mathbf{A}}_{\lambda}$)
Server deploys final global model ($\hat{A}_{\lambda}$) to all clients.

tuning. N-beats was tuned to achieve the best Mean Squared Error (MSE) global loss, and the final hyper-parameters were 256 for the batch size, learning rate $5e^{-4}$, and the number of seasonal and trend neurons were 512 and 64, respectively. The count of generic, trend, and seasonal layers was set to 2. The meta-model of FedForecaster is set to predict the most promising $K = 3$ algorithms. The Bayesian optimization algorithm was set to use the expected improvement as an acquisition function with the Gaussian processes surrogate model.

We evaluated the performance of the algorithms on 12 real-world datasets from Kaggle and the Nasdaq stock market that were not used in constructing the knowledge base for FedForecaster. The stock market datasets include prices of stocks within the same exchange-traded funds (ETFs) over a shared time period, while the other datasets were split across a number of clients ranging from the set of {5, 10, 15, 20} clients using time-series splits. To ensure the presence of enough samples per client, larger client numbers resulting in smaller splits than 500 instances are discarded. The aim was to minimize the Mean Squared Error (MSE) for all methods within the given time budget. The experiments were repeated 3 times with different random seeds and the final average results are reported. The experiments were done using 1 vCPU per client node with 2 GB memory running on Intel Xeon(R) Gold 6138 CPU@2GHz.

5.2 Results and Discussion

Table 3 summarizes the performance comparison of FedForecaster against the baselines of random search and N-beats in federated settings. The N-beats Cons. is the N-beats algorithm trained on the consolidated time-series clients' splits into a single dataset except for the last 3 ETFs datasets that were originally not a single time-series signal and concatenating them back into a single dataset could be misleading. The table presents the number of dataset instances (Len.), the number of clients (Clients), and the test Mean Squared Error (MSE) results for each method.

The results in Table 3 demonstrate that FedForecaster consistently outperforms random search and N-beats in most cases, particularly when dealing with complex datasets with varying client sizes. Notably, FedForecaster achieved the lowest test mean squared error (MSE) in 10 out of the 12 datasets within the constrained time budget, achieving an overall ranking of 1.17, compared to 2.17 for random search and 2.67 for N-beats. For example, on the USBirthsDaily dataset, FedForecaster achieved an MSE of 434.89, compared to 533.37 for random search and 983.36 for N-beats. Similarly, on the BOE-XUDLERD dataset, it achieved a significantly lower MSE of 0.006 compared to both baselines. Additional experiments were carried out on possible client counts and different time budgets in our repository 3. The results demonstrate consistency with the findings of our study.

However, there are few cases where random search or N-beats performed better, such as in the nasdaq Brazil Base Financial Rate and Energy Select Sector datasets, where random search outperformed FedForecaster slightly. These instances suggest that, while FedForecaster generally provides superior performance within limited time constraints, there may be room for further improvement in certain scenarios, especially where simpler models can perform well with less tuning.

FedForecaster demonstrated robust performance across a diverse set of datasets, highlighting its effectiveness in FL environments where data privacy and computational constraints are crucial. The overall ranking and average MSE confirm its potential as a reliable tool for automated time-series forecasting in such distributed settings.

The poor performance of N-Beats in the federated settings could be attributed to the small data splits on each client node, which is unsuitable for neural-based models requiring larger datasets for practical training. As the number of clients increases, the size of each split decreases, limiting N-Beats' ability to capture patterns accurately. However, as demonstrated in the N-Beats Cons. results, the performance is improved when training with longer data sets.

To statistically validate the performance of FedForecaster, we performed the Wilcoxon Signed-Rank test [33], that is a non-parametric statistical test used to compare paired samples to assess whether their population mean ranks differ. We compare FedForecaster average MSE results with those of random search and N-beats across the 12 datasets. The p-value for the comparison between FedForecaster and random search was $p = 0.034$, and between FedForecaster and N-beats, $p = 0.003$. Since both p-values are below the significance level of 0.05, we conclude that there is significant evidence to suggest that

Table 3: Performance Comparison of FedForecaster, Random Search, and N-beats in FL settings and N-beats Cons. on consolidated time-series datasets using 12 different datasets in terms of MSE

Dataset	Len.	N-Beats Cons.	Clients	FedForecaster	Random Search	N-Beats	Best Model
BOE-XUDLERD	15653	*0.004*	20	**0.006**	0.011	0.071	HuberRegressor
SunSpotDaily	73924	*16.51*	20	**29.37**	32.07	63.38	Lasso
USBirthsDaily	7305	820.02	5	**434.89**	533.37	983.36	LinearSVR
nasdaq_Brazil_Base_Financial_Rate	10091	*0.031*	10	0.058	**0.048**	0.153	LinearSVR
nasdaq_Brazil_Pr_Base_Financial_Rate	10091	*0.0014*	15	**0.008**	0.012	**0.008**	HuberRegressor
nasdaq_Brazil_Saving_Deposits1	812	*0.0252*	5	**0.028**	0.039	0.412	Lasso
nasdaq_Brazil_Saving_Deposits2	1182	*0.0057*	10	**0.020**	0.025	0.024	XGBRegressor
nasdaq_EIA_PET_RWTC	9124	*1.11*	5	**1.29**	1.40	8.66	LinearSVR
nasdaq_WIKI_AAPL_Price	9124	3.99	15	**3.76**	4.24	4.15	LinearSVR
Energy Select Sector ETF	2517	-	10	3.44	**2.87**	24.61	Lasso
The Technology Sector ETF	2517	-	10	**40.00**	101.70	75.98	QuantileRegressor
Utilities Select Sector ETF	2517	-	10	**1.30**	11.70	17.58	HuberRegressor

FedForecaster outperforms both baseline methods within the 5-minute time budget.

Runtime. there is an offline overhead in FedForecaster to construct the knowledge base and training the meta-model. While this effort runtime is unimportant as it is just made once for training a good meta-model, it is undoubtedly an additional effort to augment the knowledge base with more datasets to train a more robust meta-model and enhance its predictions. A single record in our constructed knowledge base takes around 114.53 seconds. For each new FL task, the client's meta-feature extraction cost depends on the client's hardware specs. However, this does not affect the inference stage and could be done at any time from the client side; it took, on average, 2.74 seconds for each client to construct its meta-features over our reported benchmarking datasets, which is insignificant time compared to the online phase (5 minutes). Although no offline effort is needed for the other baselines, N-beats require hyper-parameter optimization to its network architecture, which could be time consuming.

5.3 Meta-Model Evaluation

To select the best meta-model for recommending forecasting algorithms, we trained and evaluated several classifiers using the knowledge base constructed from the 512 synthetic and 30 real datasets. Our objective was to optimize the Mean Reciprocal Rank (MRR) at the top $K = 3$ predictions, ensuring that the best-performing forecasting algorithms were highly ranked in the recommendations. MRR is used to evaluate the effectiveness of retrieval systems, calculated as the average of the reciprocal ranks of the first relevant results [36]. The knowledge base was split into 80% training and 20% validation datasets, and hyperparameter tuning was performed using Random Search on the validation set.

Table 4 summarizes the performance of the different classifiers in terms of MRR@3 and F1 Score. The Random Forest Classifier outperformed the other classifiers, achieving the highest MRR@3 of 85.8% and an F1 score of 74%. This model demonstrated superior ability to accurately recommend the top-performing algorithms based on the meta-features of the datasets. Other classifiers, such as XGBoost and Logistic Regression, also demonstrated strong performance, but they have not reached the predictive power of the Random Forest Classifier.

The Random Forest Classifier, with its high MRR@3 and F1 score, was selected as the final meta-model for FedForecaster.

Table 4: Performance of Different Classifiers for the Meta-Model

Model	MRR@3	F1 Score
XGBClassifier	0.840	**0.74**
Logistic Regression	0.825	0.70
Gradient Boosting	0.825	0.73
Random Forest	**0.858**	**0.74**
CatBoost	0.813	0.69
LightGBM	0.790	0.66
Extra Trees	0.788	0.64
MLPClassifier	0.663	0.49

This meta-model efficiently identifies the top algorithms for any new federated time-series dataset based on its meta-features.

6 CONCLUSION

In this paper, we introduced FedForecaster, a novel automated machine-learning framework designed to address the challenges of time-series forecasting in FL environments. By utilizing a meta-model for algorithm recommendation based on statistical meta-features and Bayesian optimization for hyperparameter tuning, FedForecaster achieves efficient model training without centralized data aggregation. The experimental results demonstrate that FedForecaster outperforms both random search and the N-Beats algorithm in terms of minimizing MSE across multiple real-world datasets, all within a limited time budget. Additionally, statistical validation using the Wilcoxon Signed-Rank Test confirmed that FedForecaster provides statistically significant improvements over baseline methods.

Future Research Directions. While FedForecaster has shown promising results, several future research directions remain to be explored. One area of interest is expanding the framework to handle multivariate time-series data, as real-world applications often involve complex interactions between multiple variables. Another potential enhancement is exploring dynamic model adaptation to adjust for shifting data distributions over time.

ACKNOWLEDGMENTS

This work was supported by the innovation hub at Giza Systems [4] and the project "Increasing the knowledge intensity of Ida-Viru entrepreneurship" co-funded by the European Union.

[4] https://gizasystems.com

REFERENCES

[1] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N Whatmough, and Venkatesh Saligrama. 2021. Federated learning based on dynamic regularization. *arXiv preprint arXiv:2111.04263* (2021).
[2] Md Il Alam, Koushik Kar, Theodoros Salonidis, and Horst Samulowitz. 2023. FLASH: Automating federated learning using CASH. In *Uncertainty in Artificial Intelligence*. PMLR, 45–55.
[3] Ana Almeida, Susana Brás, Susana Sargento, and Filipe Cabral Pinto. 2023. Time series big data: a survey on data stream frameworks, analysis and algorithms. *Journal of Big Data* 10, 1 (2023), 83.
[4] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, et al. 2020. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390* (2020).
[5] Christopher Briggs, Zhong Fan, and Peter Andras. 2022. Federated learning for short-term residential load forecasting. *IEEE Open Access Journal of Power and Energy* 9 (2022), 573–583.
[6] Shengchao Chen, Guodong Long, Tao Shen, and Jing Jiang. 2023. Prompt federated learning for weather forecasting: Toward foundation models on meteorological data. *arXiv preprint arXiv:2301.09152* (2023).
[7] Hassan Eldeeb, Mohamed Maher, Radwa Elshawi, and Sherif Sakr. 2024. AutoMLBench: A comprehensive experimental evaluation of automated machine learning frameworks. *Expert Systems with Applications* 243 (2024), 122877.
[8] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. 2019. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287* (2019).
[9] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (2020).
[10] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2022. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research* 23, 261 (2022), 1–61.
[11] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. In *Automated Machine Learning*. Springer, Cham, 3–33.
[12] Chaoyang He, Erum Mushtaq, Jie Ding, and Salman Avestimehr. 2021. Fednas: Federated deep learning via neural architecture search. (2021).
[13] Xin Jie, Xixi Zhou, Chanfei Su, Zijun Zhou, Yuqing Yuan, Jiajun Bu, and Haishuai Wang. 2024. Disentangled Anomaly Detection For Multivariate Time Series. In *Companion Proceedings of the ACM on Web Conference 2024*. 931–934.
[14] Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina F Balcan, Virginia Smith, and Ameet Talwalkar. 2021. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. *Advances in Neural Information Processing Systems* 34 (2021), 19184–19197.
[15] Dongmin Kim, Sunghyun Park, and Jaegul Choo. 2024. When Model Meets New Normals: Test-Time Adaptation for Unsupervised Time-Series Anomaly Detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 13113–13121.
[16] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 19–35.
[17] Zexi Li, Feng Mao, and Chao Wu. 2022. Can we share models if sharing data is not an option? *Patterns* 3, 11 (2022).
[18] Yi Liu, JQ James, Jiawen Kang, Dusit Niyato, and Shuyu Zhang. 2020. Privacy-preserving traffic flow prediction: A federated learning approach. *IEEE Internet of Things Journal* 7, 8 (2020), 7751–7763.
[19] Zhen Liu, Wenbin Pei, Disen Lan, and Qianli Ma. 2024. Diffusion language-shapelets for semi-supervised time-series classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 14079–14087.
[20] Raúl Llasag Rosero, Catarina Silva, and Bernardete Ribeiro. 2023. Forecasting functional time series using federated learning. In *International Conference on Engineering Applications of Neural Networks*. Springer, 491–504.
[21] Wang Luping, WANG Wei, and LI Bo. 2019. CMFL: Mitigating communication overhead for federated learning. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 954–964.
[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.
[23] Hesham Mostafa. 2019. Robust federated learning through representation matching and adaptive hyper-parameters. *arXiv preprint arXiv:1912.13075* (2019).
[24] Kenji Nishimoto, Yi-Han Chiang, Hai Lin, and Yusheng Ji. 2023. FedATM: Adaptive Trimmed Mean based Federated Learning against Model Poisoning Attacks. In *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*. IEEE, 1–5.
[25] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.
[26] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2019. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437* (2019).
[27] Davy Preuveneers. 2023. AutoFL: Towards AutoML in a Federated Learning Context. *Applied Sciences* 13, 14 (2023), 8019.
[28] Marco Savi and Fabrizio Olivadese. 2021. Short-term energy consumption forecasting at the edge: A federated learning approach. *IEEE Access* 9 (2021), 95949–95969.
[29] Esraa Sayed, Mohamed Maher, Omar Sedeek, Ahmed Eldamaty, Amr Kamel, and Radwa El Shawi. 2024. GizaML: A Collaborative Meta-learning Based Framework Using LLM For Automated Time-Series Forecasting.. In *EDBT*. 830–833.
[30] Sean J Taylor and Benjamin Letham. 2018. Forecasting at scale. *The American Statistician* 72, 1 (2018), 37–45.
[31] Siqi Wang, Jiashu Li, Mian Lu, Zhao Zheng, Yuqiang Chen, and Bingsheng He. 2022. A system for time series feature extraction in federated learning. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 5024–5028.
[32] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. 2023. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics* 14, 2 (2023), 513–535.
[33] Robert F Woolson. 2005. Wilcoxon signed-rank test. *Encyclopedia of Biostatistics* 8 (2005).
[34] Chenrui Wu, Zexi Li, Fangxin Wang, and Chao Wu. 2023. Learning cautiously in federated learning with noisy and heterogeneous clients. In *2023 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 660–665.
[35] Chenrui Wu, Yifei Zhu, Rongyu Zhang, Yun Chen, Fangxin Wang, and Shuguang Cui. 2023. Fedab: Truthful federated learning with auction-based combinatorial multi-armed bandit. *IEEE Internet of Things Journal* 10, 17 (2023), 15159–15170.
[36] Yang Wu, Masayuki Mukunoki, Takuya Funatomi, Michihiko Minoh, and Shihong Lao. 2011. Optimizing Mean Reciprocal Rank for person re-identification. In *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 408–413. https://doi.org/10.1109/AVSS.2011.6027363
[37] Yingda Xia, Dong Yang, Wenqi Li, Andriy Myronenko, Daguang Xu, Hirofumi Obinata, Hitoshi Mori, Peng An, Stephanie Harmon, Evrim Turkbey, et al. 2021. Auto-FedAvg: learnable federated averaging for multi-institutional medical image segmentation. *arXiv preprint arXiv:2104.10195* (2021).
[38] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A survey on federated learning. *Knowledge-Based Systems* 216 (2021), 106775.
[39] Tuo Zhang, Chaoyang He, Tianhao Ma, Lei Gao, Mark Ma, and Salman Avestimehr. 2021. Federated learning for internet of things. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 413–419.

Thank You!

Giza Systems