Journal of Applied Informatics and Computing (JAIC)
Vol.10, No.3, June 2026, pp. 2376~2390
e-ISSN: 2548-6861

Attention-Enhanced Multivariate Forecasting for Intelligent Microservice Autoscaling

Nur Saifuddin [1]*, Mula Agung Barata [2]*, Ifnu Wisma Dwi Prastya [3]*
* Department of Informatics Engineering, Faculty of Science and Technology, Universitas Nahdlatul Ulama Sunan Giri
nsaifuddin29@gmail.com [1], mula.ab26@gmail.com [2], ifnuprastya@unugiri.ac.id [3]

Article Info

Article history:

Received 2026-03-10
Revised 2026-04-30
Accepted 2026-05-25

Keyword:

Attention Mechanism,
Cloud-Native Microservices,
Multivariate Forecasting,
Proactive Autoscaling,
Resource Utilization Prediction.

ABSTRACT

Proactive autoscaling in cloud-native microservices requires forecasts that are accurate enough to support reliable scaling decisions rather than merely low regression error. This study evaluates a leakage-aware forecasting-to-decision pipeline for CPU and memory utilization using a retained Alibaba microservice trace subset containing 48 service instances and 47 nodes. The experimental design applies chronological per-service train-validation-test splits, train-only preprocessing, one-step multivariate forecasting, fixed-backbone attention ablation, and NoPred/PredOnly/WithPred feature-set attribution for downstream action classification. The forecasting stage compares recurrent baselines, a vanilla Transformer Encoder, and BiLSTM encoder-decoder variants with Bahdanau, Luong, and multi-head attention. The best configuration, a BiLSTM encoder-decoder with Bahdanau attention and residual connection, achieves the lowest joint CPU-memory test RMSE of 0.017188, improving over the BiLSTM baseline (0.020649) and Transformer Encoder (0.019534). Target-specific results show CPU RMSE of 0.028924 and memory RMSE of 0.005452, while burst analysis indicates lower spike-segment RMSE and MAE despite stronger Transformer behavior on selected peak-alignment indicators. In the decision stage, tuned XGBoost with forecast-augmented features obtains 0.950602 accuracy and 0.951026 macro F1, outperforming NoPred and PredOnly settings. A read-only downstream simulation further reduces service-level objective violation compared with a horizontal autoscaling policy under Normal and Stress×3 scenarios while maintaining zero simulated downtime under the in-place update assumption. Overall, the findings indicate that forecast-derived signals can improve proactive autoscaling decisions when evaluated through strict temporal controls, explicit attribution, and bounded simulation.

I. INTRODUCTION

Modern cloud-native deployments commonly rely on Kubernetes autoscaling to sustain service performance under fluctuating demand [1], [2]. In practice, the Horizontal Pod Autoscaler adjusts the number of Pod replicas using observed workload signals such as CPU and memory utilization, or other configured metrics [1]. Complementarily, the Vertical Pod Autoscaler targets vertical adjustments by recommending, and depending on update policies applying, CPU and memory requests and limits based on observed resource usage and historical evidence [3]. Despite their practicality, reactive decision loops and interval-based updates can amplify sensitivity to noisy or rapidly changing CPU and memory dynamics, which may create decision lag and resource misallocation under highly dynamic microservice workloads [2], [4]. Consequently, there is strong motivation to make scaling decisions more data-driven by predicting future resource demand rather than reacting solely to recent observations [2], [5].

A growing body of research has investigated proactive autoscaling strategies that incorporate workload forecasting

to anticipate demand and reduce scaling delay in microservice platforms [2], [5]. Because CPU and memory are jointly evolving resource signals, this work formulates their prediction as a multivariate time-series problem and connects the resulting forecasts to downstream scaling decisions through a predict-then-optimize perspective [6], [7], [8], [9]. The novelty is therefore not claimed from the generic use of a two-stage pipeline, which is already well represented in recent autoscaling and prediction-to-decision literature [2], [5], [8], [9]. Instead, the study focuses on how forecasting gains are isolated, transferred, and evaluated at decision time under strict temporal controls and explicit feature-set attribution. To support trace-driven evaluation, the experiments use the Alibaba Cluster Trace Microservices v2021 release, while preprocessing includes validity controls motivated by known artifacts in the Alibaba trace ecosystem [10], [11], [12].

Recent forecasting studies show that recurrent, attention-based, and Transformer-style architectures can all be competitive for multivariate or long-sequence time-series prediction, although their relative performance depends on data characteristics, feature construction, and evaluation discipline [6], [7], [13], [14]. Informer demonstrates the strength of Transformer-based forecasting in long-sequence settings [13], whereas attention-based multivariate autoregression and encoder-decoder forecasting studies indicate that attention can improve dependency modeling when its configuration and training process are carefully controlled [6], [7], [14]. Consequently, the relevant question in this study is not whether attention is intrinsically novel, but whether an attention-enhanced BiLSTM encoder-decoder remains advantageous when compared with strong recurrent baselines and a vanilla Transformer Encoder under identical temporal splits, shared feature engineering, and comparable model-selection discipline [13], [15], [16]. This framing is also consistent with autoscaling work showing that automated featurization can improve forecasting quality and mitigate undesirable scaling dynamics [17].

At the same time, representative autoscaling studies frequently emphasize end-to-end outcomes such as utilization stability, service-level objective risk, uncertainty-aware scaling quality, or actuation efficiency, whereas the internal contribution of the forecasting stage versus the decision stage often remains only partially attributable [4], [18], [19]. That distinction is methodologically important because, in autoscaling pipelines, small forecasting errors can become decision-critical when they occur near operational boundaries, especially under bursty CPU regimes where the separation between hold and scale_up actions may become operationally narrow [4], [18], [20]. Forecast evaluation is also vulnerable to optimistic bias when temporal order is violated, when preprocessing is fit outside the training partition, or when threshold construction leaks future information into model development, and these issues have been explicitly identified as recurring pitfalls in recent forecasting and machine-learning evaluation guidance [20], [21]. For that reason, this study centers its contribution on a strict no-leakage protocol and a fair fixed-backbone attention ablation, both of which are aligned with recent concerns on forecasting evaluation and prediction-to-decision modeling [8], [9], [20], [21]. The NoPred, PredOnly, and WithPred settings are then introduced as the study's controlled attribution design for measuring how forecast-derived signals affect downstream autoscaling decisions. In this formulation, the core question becomes whether better forecasting quality can be shown, under controlled attribution, to produce measurable gains in autoscaling decision quality [8], [9].

Based on this problem setting, the objective of this study is to examine whether a controlled forecasting-to-decision pipeline can improve proactive autoscaling quality under trace-driven microservice workloads while making the source of improvement attributable at each stage [2], [8], [9], [20]. Forecasting is evaluated using RMSE, MAE, SMAPE, and R^2, whereas the downstream decision stage is assessed using Accuracy, Precision, Recall, F1 score, one-versus-rest AUC, and Log Loss [20], [22]. The contributions of this study are therefore fivefold: (1) a controlled benchmark of multivariate CPU and memory forecasting that contrasts recurrent baselines, a Transformer Encoder comparator, and attention-enhanced BiLSTM encoder–decoder variants under identical evaluation rules, (2) a fair fixed-backbone ablation that isolates the effect of attention design and residual pathways within the same BiLSTM encoder–decoder backbone, (3) a strict no-leakage protocol that enforces time-respecting splits and training-only fitting for preprocessing and oracle-threshold estimation, (4) an explicit prediction-to-decision transfer analysis that quantifies the decision value of predicted signals through NoPred, PredOnly, and WithPred feature sets, and (5) a supporting downstream validation that contrasts the proposed pipeline against Kubernetes-style horizontal and vertical autoscaling baselines and is reported as contextual evidence rather than the primary optimization target.

II. METHOD

This study adopts an experimental research design that benchmarks a two-stage machine learning pipeline for autoscaling decisions, where multivariate forecasting predicts future CPU and memory demand and a downstream multiclass classifier maps demand signals into scaling actions [2], [5], [8], [9]. The workflow is structured as data validity controls, feature engineering, time-respecting forecasting preparation, forecasting benchmarking and improvement, decision label construction, classification benchmarking and optimization, and standardized reporting. The overall workflow is summarized in Fig. 1. The diagram separates the forecasting stage, the feature-set attribution stage, and the downstream simulation stage so that the evaluation logic remains clear: forecasting quality is first measured directly, forecast-derived signals are then transferred into the classifier, and the selected pipeline is finally interpreted through read-only downstream validation.

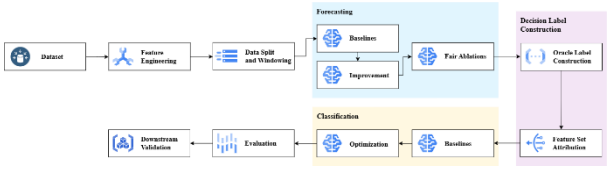

Figure 1. Research Workflow of the Two-Stage Pipeline

A. *Dataset*

This study uses the public Alibaba Cluster Trace Microservices v2021 release as the external source of production-grade workload traces and treats the retained subset as a trace-driven autoscaling benchmark rather than as a live deployment log collected by the authors [10]–[12]. The experimental subset retains five resource-oriented fields, namely timestamp, msinstanceid, nodeid, cpu_utilization, and memory_utilization, because these fields are sufficient to reconstruct ordered service-level resource dynamics under a controlled forecasting-to-decision pipeline [10], [11]. Rows with incomplete values are removed, and conservative validity controls are applied to reduce the risk that known artifacts in the Alibaba trace ecosystem bias the downstream analysis when interpreted as fully reliable ground truth [12].

The retained subset contains 183,757 valid observations from 48 service instances and 47 nodes. The raw timestamp increments are 60,000 units. Therefore, the experiments treat them as a regular relative sampling interval rather than as reliable wall-clock calendar time. However, because calendar-time conversion produced artifacts, the experiments use ordered per-service sequence indices rather than wall-clock calendar semantics. CPU utilization ranges from 0.0 to 1.0, while memory utilization ranges from 0.0 to 0.8930. Table I summarizes the dataset and windowed experimental subset used in this study.

TABLE I
DATASET AND EXPERIMENTAL SUBSET SUMMARY

Item	Value
Trace source	Alibaba Cluster Trace Microservices v2021
Retained raw fields	timestamp, msinstanceid, nodeid, cpu_utilization, memory_utilization
Initial records	184,237
Valid observations	183,757
Service instances / nodes	48 / 47
Unique timestamps	4,310
Effective interval	60,000 timestamp units
Train / validation / test windows	127,179 / 26,103 / 26,155
Sequence length	30
Forecasting features / targets	22 / 2

B. *Feature Engineering*

Feature engineering is treated as a primary design component because proactive autoscaling depends not only on current utilization, but also on short-horizon temporal context and contextual covariates that approximate service- and node-level operating conditions [17], [20]. This emphasis is also consistent with applied structured-signal classification studies, where optimized feature extraction, feature selection, and signal transformation have been shown to improve downstream classification behavior in C4.5-based gas detection, Chi-Square-assisted e-nose classification, and e-nose-based food quality identification tasks [23]–[25]. Accordingly, the feature set combines lagged resource values, rolling statistics, and contextual auxiliary covariates representing node-capacity proxies and service-level metadata. For both CPU and memory, lag features are constructed at 1, 5, and 10 steps, while rolling descriptors include the mean and standard deviation over windows of 5, 15, and 30 steps. This design captures instantaneous changes, short-term persistence, and local volatility, which are operationally relevant when scaling actions must react to rising resource pressure before sustained violations emerge [17], [20].

Contextual variables are incorporated as node-capacity proxies and service-level metadata. Because the retained trace subset does not provide complete operational metadata for all required covariates, these variables are generated as synthetic auxiliary covariates for experimental control and are interpreted as modeling support rather than directly observed production metadata. Timestamp-derived temporal fields are also generated; however, they are not interpreted as calendar seasonality because the converted timestamps do not preserve reliable wall-clock semantics. After feature engineering, the processed table contains 33 columns. For downstream classification, NoPred provides 22 classifier predictors from observed engineered features, whereas WithPred provides 24 predictors after appending predicted_cpu and predicted_mem generated by the frozen BiLSTM encoder-decoder with Bahdanau attention and residual connection.

C. *Data Split and Windowing*

Each service stream is divided chronologically into training, validation, and test partitions with proportions 0.70, 0.15, and 0.15, respectively, so that temporal order is preserved throughout model development and no future observations are allowed to influence earlier fitting stages [20], [21]. This design follows recent forecasting-evaluation guidance that discourages random reshuffling when the target process contains temporal dependence, local regime shifts, or short-horizon autocorrelation [20]. All feature scaling is therefore fitted on the training partition only and then transferred unchanged to the validation and test partitions, thereby preventing information leakage through preprocessing statistics [20], [21].

One-step-ahead multivariate forecasting is implemented through fixed-length windows of 30 timesteps. Let $x_t^{(s)} \in \mathbb{R}^d$ denote the multivariate feature vector of service stream $s$at time t, Ldenote the window length, and $y_{t+1}^{(s)} \in \mathbb{R}^2$ denote the

true next-step CPU–memory target pair. The input window and target pair are defined as

$$X_t^{(s)} = [x_{t-L+1}^{(s)}, x_{t-L+2}^{(s)}, \dots, x_t^{(s)}], y_{t+1}^{(s)} = [c_{t+1}^{(s)}, m_{t+1}^{(s)}]^T \quad (1)$$

The forecasting model then maps the 30-step input window into the next-step CPU–memory prediction as

$$\hat{y}_{t+1}^{(s)} = f_\theta(X_t^{(s)}). \quad (2)$$

In (1), $X_t^{(s)}$ denotes the input window for service stream s, $y_{t+1}^{(s)}$denotes the true next-step CPU–memory target, and $c_{t+1}^{(s)}$and $m_{t+1}^{(s)}$denote the true CPU and memory utilization at the next timestep. In (2), $\hat{y}_{t+1}^{(s)}$ denotes the model forecast consumed later by the downstream decision stage, and $f_\theta(\cdot$)denotes the forecasting model parameterized by θ. In this study, $L = 30$.

Under this protocol, the final forecasting tensors have shapes (127179, 30, 22) for training, (26103, 30, 22) for validation, and (26155, 30, 22) for testing. In the downstream classification stage, the feature mode is last, so the classifier consumes the most recent engineered observation from each window, with or without forecast-derived signals depending on the attribution setting.

D. Forecasting Models

The forecasting benchmark includes both non-neural and neural baselines under the same split, scaling, and reporting protocol. The non-neural reference consists of Support Vector Regression with an RBF kernel, implemented through MultiOutputRegressor so that CPU and memory can be predicted jointly using one regressor per target [20], [26], [27]. The recurrent neural baselines comprise LSTM, GRU, and BiLSTM models implemented with standard recurrent layers in Keras [28]–[30]. These baselines are retained because they provide strong recurrent reference models before any architectural enhancement is introduced.

To broaden the comparison beyond recurrent models, a vanilla Transformer Encoder comparator is additionally included [13], [31]. The selected comparator is a vanilla Transformer Encoder forecasting model with input projection, sinusoidal positional encoding, two stacked encoder blocks, four attention heads, a feed-forward dimension of 128, global average pooling, and a final dense readout for joint next-step CPU and memory prediction [13], [32]. This model is not introduced as a replacement for the proposed architecture, but as a contemporary external reference evaluated under the same temporal split, scaling protocol, and model-selection discipline.

The proposed forecasting enhancement is built on a BiLSTM encoder–decoder backbone whose encoder summarizes the 30-step multivariate input sequence and whose decoder produces the joint next-step CPU and memory forecast [7], [28], [30], [31]. The decoder is initialized from the concatenated forward and backward encoder states, after which the decoder representation is combined with an attention-mediated context vector before the final dense readout is generated. Hyperparameter optimization is divided into two stages to preserve attribution fairness. In stage B2, only the backbone hyperparameters are tuned, including encoder width, decoder width, dropout, learning rate, and batch size. In stage B3, attention-specific parameters are tuned while the selected B2 backbone is held fixed, so that any measured improvement can be attributed to the attention design rather than to hidden backbone changes. Hyperparameter search is carried out with Optuna using 50 trials for the backbone stage and 50 trials for the attention stage [15], [16]. The shared forecasting training protocol uses seed 42, 100 epochs, early stopping patience 15, and ReduceLROnPlateau patience 7 with a factor of 0.5.

Let $H = \{h_1, h_2, \dots, h_L\}$denote the encoder hidden-state sequence and $s$denote the decoder state used for next-step prediction. The attention computation is written as

$$e_i = a(s, h_i), \alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{L} \exp(e_j)}. \quad (3)$$

$$c = \sum_{i=1}^{L} \alpha_i h_i, \hat{y}_{t+1} = g([s; c]). \quad (4)$$

In (3) and (4), e_i is the alignment score, α_i is the normalized attention weight, c is the context vector, and $g(\cdot)$ is the dense prediction head for joint CPU and memory forecasting.

To isolate attention effects fairly, the ablation varies the attention operator A and residual indicator r while keeping the tuned backbone θ_{B2}^* fixed:

$$\hat{y}_{t+1}^{(A,r)} = f(X_t; \theta_{B2}^*, A, r), \quad (5)$$

$$A \in \{\text{Bahdanau,Luong,MultiHead}\}, r \in \{0,1\}. \quad (6)$$

In (5) and (6), θ_{B2}^* denotes the fixed tuned backbone, A denotes the attention family, and r indicates whether the residual pathway is disabled or enabled.

Table II summarizes the forecasting families evaluated in this study. The table is intentionally compact because the detailed architectural distinction is discussed in the surrounding text: the Transformer Encoder is included as a modern comparator, whereas the BiLSTM encoder–decoder variants are used to test whether attention and residual pathways improve joint CPU–memory forecasting under the same temporal protocol.

TABLE II
FORECASTING MODEL FAMILIES AND KEY CONFIGURATIONS

Model Family	Variants	Key Configuration
Kernel baseline	SVR MultiOutput	RBF kernel; one regressor per target

Recurrent baselines	LSTM, GRU, BiLSTM	64 units; dropout 0.2; validation monitoring
Modern comparator	Transformer Encoder	d_model 64; 2 blocks; 4 heads; FFN 128
Encoder–decoder	BiLSTM-ED without attention	tuned B2 backbone
Attention ablation	Bahdanau, Luong, Multi-head	fixed backbone; residual ON/OFF
Training discipline	Neural variants	train-only scaling; validation selection; test reporting

Thus, the forecasting comparison separates three roles: recurrent baselines establish the initial benchmark, the Transformer Encoder provides a modern comparator, and the fixed-backbone BiLSTM encoder–decoder ablation isolates the contribution of attention and residual pathways.

E. Decision Label Construction

Decision supervision is derived by mapping continuous future utilization into three autoscaling actions scale_down, hold, and scale_up. The thresholds are estimated from the training targets only using a quantile-based rule with $Q_{LOW} = 0.35$and $Q_{HIGH} = 0.80$, so that the labeling policy remains temporally valid and does not leak future distributional information into model development [20], [21]. The resulting train-only thresholds are $\tau_{low}^{cpu} = 0.085627086461$, $\tau_{high}^{cpu} = 0.266939997673$, $\tau_{low}^{mem} = 0.529880583286$, and $\tau_{high}^{mem} = 0.718845546246$. Oracle labels are computed from the true next-step CPU and memory targets, rather than from predicted values. When CPU and memory imply conflicting actions, the priority rule is scale_up > scale_down > hold, reflecting the operational preference to avoid under-provisioning before conservative de-escalation is considered.

The oracle decision rule is expressed as

$$\ell(c_{t+1}, m_{t+1}) = \begin{cases} \text{scale\_up}, & c_{t+1} \geq \tau_{high}^{cpu} \text{ or } m_{t+1} \geq \tau_{high}^{mem}, \\ \text{scale\_down}, & c_{t+1} \leq \tau_{low}^{cpu} \text{ and } m_{t+1} \leq \tau_{low}^{mem}, \\ \text{hold}, & \text{otherwise.} \end{cases} \quad (7)$$

In (7), $c_{t+1}$and m_{t+1}denote the true next-step CPU and memory utilization, while all four thresholds are estimated exclusively from the training partition.

To quantify the decision value of forecasting signals explicitly, three feature-set settings are defined. NoPred uses only observed engineered features and represents a reactive decision formulation. PredOnly uses only predicted_cpu and predicted_mem, thereby isolating the information content of the forecasting outputs alone. WithPred combines the observed engineered features with the two forecast-derived signals generated by the frozen BiLSTM encoder–decoder with Bahdanau attention and residual connection. In the final exported matrices, the WithPred files contain 31 columns including audit and label fields, corresponding to 24 classifier predictors. The NoPred files contain 29 columns including audit and label fields, corresponding to 22 classifier predictors under the same split protocol. This design makes it possible to evaluate prediction-to-decision transfer under controlled attribution rather than inferring it indirectly from end-to-end policy behavior [8], [9].

F. Classification Models

The downstream decision task is benchmarked using multinomial logistic regression, decision tree, random forest, and XGBoost under a shared preprocessing pipeline [33]–[36]. Logistic regression is retained as a regularized multiclass linear-probabilistic baseline [35], decision trees provide a direct nonlinear partitioning baseline [36], and random forests extend that baseline through bagging-based averaging that improves predictive accuracy while controlling overfitting relative to a single tree [34]. Related applied classification studies have also used C4.5 or Decision Tree variants and SVM-style comparators to evaluate structured classification tasks, reinforcing the value of retaining tree-based models as interpretable nonlinear baselines before selecting a stronger ensemble classifier [23], [24], [37]. XGBoost is included because gradient-boosted tree ensembles remain highly competitive on structured tabular problems, particularly when decision boundaries depend on nonlinear feature interactions and mixed-scale predictors [33], [38]. This justification is important in the present setting because the autoscaling decision stage consumes engineered tabular summaries rather than raw sequential inputs.

For all classification baselines, preprocessing uses median imputation followed by standardization, both fitted on the training partition only to prevent leakage [20], [21]. Model selection is performed exclusively on the validation split, using macro F1 Score as the primary criterion; Log Loss, Balanced Accuracy, Accuracy, and AUC OvR are used as tie-breakers when needed. In the untuned baseline comparison, XGBoost is selected as the strongest validation model and therefore becomes the basis for champion improvement. The tuning stage then evaluates WithPred, NoPred, and PredOnly separately using a common XGBoost configuration family with 20 manual trials, n_estimators = 2000, tree_method = hist, a multiclass objective, and early stopping after 50 rounds on an internal 12% stratified split drawn from the training partition [16], [33].

Table III summarizes the classification models and optimization settings. The baseline stage evaluates whether the engineered decision features are better handled by linear, single-tree, ensemble-tree, or gradient-boosted models, while the improvement stage uses XGBoost because validation results identify it as the strongest baseline for the structured tabular representation.

TABLE III
CLASSIFICATION MODELS AND OPTIMIZATION SETTINGS

Model Group	Model	Feature Set	Key Setting
Baseline	Logistic Regression	WithPred	multinomial; max_iter 2000
Baseline	Decision Tree	WithPred	nonlinear single-tree baseline
Baseline	Random Forest	WithPred	400 trees
Baseline	XGBoost	WithPred	fixed baseline parameters
Improvement	Tuned XGBoost	NoPred, PredOnly, WithPred	20 trials; early stopping; validation selection

All classification models use median imputation followed by standardization fitted on the training partition only. XGBoost is not selected arbitrarily; rather, it is retained for champion improvement because the decision stage consumes structured tabular summaries, and the validation comparison shows that tree-based nonlinear models outperform the linear baseline. The tuned XGBoost stage uses an internal 12% stratified training holdout for early stopping, while external validation remains the basis for model selection.

G. Evaluation

Forecasting performance is reported per target using RMSE, MAE, SMAPE, and R^2 for both CPU and memory utilization, so that forecasting quality is not interpreted from a single error criterion alone [20], [22]. Let $y_i$denote the true target value, $\hat{y}_i$denote the predicted value, $\bar{y}$denote the empirical mean of the true target values, and $n$denote the number of evaluated samples. The forecasting error metrics are defined in (8)–(11).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{8}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i | \tag{9}$$

$$SMAPE = \frac{100}{n}\sum_{i=1}^{n} \frac{2|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i| + \epsilon} \tag{10}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{11}$$

In (8)–(11), RMSE gives stronger penalty to large errors, MAE reports average absolute deviation, SMAPE provides a symmetric percentage-style error measure with ϵused only to avoid division by zero, and R^2 reports the proportion of target variance explained by the forecast [20], [22]. Because the forecasting task predicts two targets, namely CPU and memory utilization, macro summaries are computed as the arithmetic mean of the two target-specific scores. For a given metric M, the macro score is defined as

$$M_{macro} = \frac{M_{CPU} + M_{MEM}}{2}. \tag{12}$$

In (12), $M_{CPU}$and M_{MEM}denote the metric values computed separately for CPU and memory, respectively. This macro formulation keeps multi-target comparison compact while preserving the target-specific results in the main reporting tables.

Classification performance is reported using Accuracy, Precision, Recall, F1 Score, one-versus-rest AUC, and Log Loss. Accuracy is computed as the overall proportion of correctly classified samples, whereas Precision, Recall, and F1 Score are reported as macro-averaged scores across the three autoscaling actions, namely scale_down, hold, and scale_up, unless otherwise stated [22]. For a class k, precision, recall, and F1 score are defined in (13)–(15), where TP_k, FP_k, and FN_kdenote the true positive, false positive, and false negative counts for class k, respectively [22].

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \tag{13}$$

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \tag{14}$$

$$F1_k = \frac{2 \cdot Precision_k \cdot Recall_k}{Precision_k + Recall_k} \tag{15}$$

The reported Precision, Recall, and F1 Score values are obtained by averaging the corresponding class-wise scores across the three autoscaling actions, namely scale_down, hold, and scale_up [22]. Across both forecasting and classification stages, validation performance is used for model selection, while test performance is reserved exclusively for final reporting.

H. Downstream Validation

A supporting downstream validation is conducted as a read-only trace-driven simulation, meaning that no forecasting or classification model is retrained during this stage. Instead, the previously selected forecasting champion, namely the BiLSTM encoder–decoder with Bahdanau attention and residual connection, and the final tuned WithPred classifier are frozen and then evaluated against autoscaling baselines aligned with Kubernetes-style horizontal and vertical policies [1], [3]. The compared systems are HPA, VPA, and the proposed policy, implemented as a prediction-aware rule that uses max(pred, obs) together with in-place adjustment logic. Two demand conditions are examined: Normal and Stress×3. The Normal setting represents the trace regime as observed after preprocessing, whereas Stress×3 magnifies the workload signal to probe policy behavior under substantially heavier demand.

This downstream validation is intended to contextualize the prediction-to-decision pipeline in operational terms rather than to redefine the study as a full systems deployment paper.

Consequently, the main comparison is reported using SLO violation rate and downtime rate, together with provision-related summaries, so that forecasting and classification improvements can be interpreted in terms of decision consequences under trace-driven stress [2], [4], [18]. The downstream stage should therefore be read as supporting evidence for practical relevance, not as the primary optimization objective of the study.

III. Results And Discussion

This section reports the empirical findings across forecasting, burst behavior, decision classification, computational overhead, downstream validation, and generalization boundaries. The discussion is organized to connect model-level accuracy with decision-level relevance, rather than treating forecasting, classification, and autoscaling simulation as isolated results.

A. Experimental Setup and Reporting Protocol

All experiments were conducted under a consistent time-respecting protocol to ensure that the reported results reflected generalization to later workload segments rather than artifacts of random reshuffling or information leakage. Each service stream was divided chronologically into training, validation, and test partitions with proportions of 0.70, 0.15, and 0.15. The forecasting task used a fixed input length of 30 timesteps and predicted the next-step CPU and memory utilization jointly. Under this setting, the resulting forecasting tensors consisted of 127,179 training windows, 26,103 validation windows, and 26,155 test windows, each containing 22 input features and two target variables.

Preprocessing was deliberately constrained to the training partition. Feature standardization, imputation in the classification stage, and oracle-threshold estimation were fitted or derived only from training data before being applied unchanged to validation and test partitions. Model development also followed a validation-first selection rule: validation results were used to select forecasting and classification champions, while the test partition was reserved exclusively for final reporting. This separation is particularly important in the present pipeline because forecasting outputs are later transferred into the decision classifier; any leakage in the forecasting stage would therefore propagate into the downstream autoscaling decision stage.

The additional Transformer, burst, and overhead analyses follow the same discipline: none of them uses the test split for model selection, and all are interpreted as supporting evidence rather than as separate optimization targets. Consequently, the following results are read not only as leaderboard scores, but also as evidence of how the pipeline behaves under temporal separation, decision attribution, burst conditions, and implementation-oriented cost constraints.

B. Forecasting Results

The forecasting results are discussed in two layers. The first layer evaluates the original baseline models and the added Transformer Encoder comparator, thereby addressing whether the proposed model remains competitive when it is not compared only against recurrent architectures. The second layer examines the BiLSTM encoder–decoder improvement and attention ablation, with particular attention to whether the observed error reduction is merely numerical or has practical meaning for autoscaling decisions near utilization thresholds. Because the target series include values close to zero, the discussion emphasizes RMSE, MAE, SMAPE, and R^2 rather than relying on MAPE as the main percentage-style error measure.

Table IV condenses the original CPU table, memory table, and macro comparison into a single forecasting summary. The results show that GRU is the strongest recurrent baseline on CPU RMSE, whereas BiLSTM remains stronger when CPU and memory are considered jointly. The added Transformer Encoder provides a more demanding modern comparator: it improves the macro RMSE of the BiLSTM baseline from 0.020649 to 0.019534 and substantially improves memory forecasting, but it does not surpass the final BiLSTM encoder–decoder with Bahdanau attention and residual connection. The final model achieves the lowest CPU RMSE, memory RMSE, and macro RMSE, with a macro RMSE of 0.017188.

TABLE IV
Forecasting Baseline, Transformer Comparator, And Champion Summary

Model	CPU RMSE	CPU MAE	Memory RMSE	Memory MAE
SVR	0.058745	0.017081	0.021290	0.007385
LSTM	0.036644	0.015438	0.015705	0.010027
GRU	0.030161	0.013068	0.012906	0.008006
BiLSTM baseline	0.030977	0.011862	0.010322	0.004862
Transformer Encoder	0.031850	0.011961	0.007218	0.003653
BiLSTM-ED + Bahdanau, residual ON	**0.028924**	**0.010409**	**0.005452**	**0.001886**

Metrics not displayed in Table IV are still important for interpretation. The BiLSTM baseline records a macro MAE of 0.008362, macro SMAPE of 6.046201%, and macro R^2 of 0.982699. The Transformer Encoder records a macro MAE of 0.007807, macro SMAPE of 5.677608%, and macro R^2 of 0.982347. The final Bahdanau residual model improves these values to a macro MAE of 0.006147, macro SMAPE of 3.997143%, and macro R^2 of 0.985581. Thus, the final model reduces macro RMSE by approximately 16.76% relative to the BiLSTM baseline and 12.01% relative to the Transformer Encoder. This comparison is deliberately bounded: it shows superiority over the evaluated vanilla Transformer Encoder,

not over the entire family of Transformer-based forecasting models.

Table V reports the complete attention ablation while keeping the table compact enough for the JAIC layout. The table focuses on RMSE because it is the primary error magnitude metric used to compare forecasting variants across CPU, memory, and joint macro performance. The results show that the BiLSTM encoder–decoder without attention does not consistently improve the BiLSTM baseline: it slightly improves CPU error, but it worsens memory error. This finding is important because it indicates that the gain does not arise from converting the model into an encoder–decoder structure alone.

TABLE V
COMPLETE ATTENTION ABLATION RESULTS

Model	Residual	CPU RMSE	Memory RMSE	Macro RMSE
BiLSTM baseline	—	0.030977	0.010322	0.020649
BiLSTM-ED, no attention	—	0.030115	0.011234	0.020675
BiLSTM-ED + Luong	ON	0.030073	0.007578	0.018826
BiLSTM-ED + Bahdanau	ON	**0.028924**	**0.005452**	**0.017188**
BiLSTM-ED + Multi-head	ON	0.030525	0.006448	0.018487
BiLSTM-ED + Luong	OFF	0.029857	0.007778	0.018818
BiLSTM-ED + Bahdanau	OFF	0.030062	0.007203	0.018633
BiLSTM-ED + Multi-head	OFF	0.029805	0.008094	0.018949

Once attention is introduced, the joint forecasting error decreases more consistently. The BiLSTM encoder–decoder with Bahdanau attention and residual connection produces the lowest CPU RMSE, memory RMSE, and macro RMSE among all evaluated ablations. Its CPU RMSE of 0.028924 corresponds to a 6.63% reduction relative to the BiLSTM baseline, while its memory RMSE of 0.005452 corresponds to a 47.18% reduction. The target-specific MAE, SMAPE, and R^2 values support the same interpretation: the final model obtains CPU MAE 0.010409, CPU SMAPE 6.694905%, CPU R^2 0.971788, memory MAE 0.001886, memory SMAPE 1.299381%, and memory R^2 0.999374. Residual-off variants remain competitive on selected CPU-side metrics, but they do not provide the strongest joint CPU–memory result. Therefore, the final selection is based on balanced multi-target forecasting performance rather than on a single isolated metric.

Several points follow from the ablation. First, attention is not merely an architectural decoration in this experiment. The BiLSTM encoder–decoder without attention improves CPU only modestly and degrades memory, whereas the attention-equipped variants consistently produce stronger memory forecasts and generally lower CPU errors. Second, the residual pathway should be interpreted as a stabilizing design choice rather than a universal winner for every single metric. In several rows, residual-off variants remain competitive, particularly on selected CPU metrics; however, the final BiLSTM encoder–decoder with Bahdanau attention and residual connection is the most balanced model because it achieves the lowest RMSE on both CPU and memory while also maintaining the strongest overall macro performance. Third, the Transformer Encoder result sharpens the interpretation of the contribution. The proposed attention-enhanced BiLSTM encoder–decoder does not win simply because it is compared against weak recurrent baselines; it remains superior after the addition of a modern Transformer Encoder comparator, especially when CPU and memory errors are evaluated jointly. The practical meaning of these improvements is best understood in relation to autoscaling decisions rather than isolated regression scores. A small CPU error reduction can still matter when utilization is near an action threshold, because a slight underestimation may delay a scale_up decision, whereas a slight overestimation may trigger unnecessary provisioning. Memory forecasting, meanwhile, affects the stability of the decision signal because memory changes more gradually in the retained subset and can act as a complementary indicator of sustained pressure. Therefore, the combined reduction of CPU and memory error supports the later classification stage: the classifier receives predicted demand signals that are not only accurate on average, but also less noisy around regions where the autoscaling action is sensitive to threshold boundaries.

The numerical forecasting results are complemented by visual inspection of the held-out test trajectory. Figure 2 shows the CPU forecast against the actual CPU signal, where the main concern is whether rapid increases are followed without excessive lag. This visual evidence is not used for model selection, but it helps interpret why even moderate CPU RMSE reductions may matter near the hold and scale_up boundary.

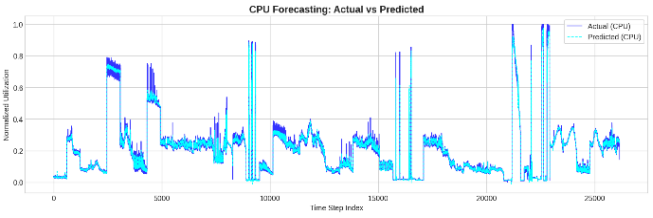

Figure 2. Forecast vs. Actual CPU

Figure 3 presents the corresponding memory forecast. In contrast to CPU, memory utilization is smoother in the retained subset, and the visual alignment supports the quantitative finding that attention-based temporal aggregation produces a larger improvement for memory than for CPU. Together, Fig. 2 and Fig. 3 show that the selected forecasting model improves both volatile and smoother resource signals, although the magnitude of improvement differs by target.

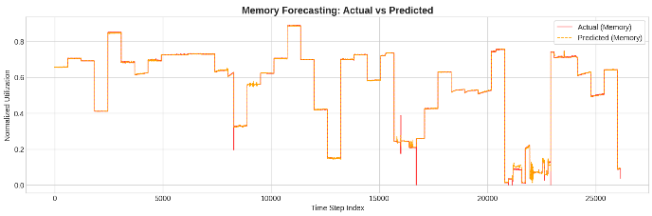

Figure 3. Forecast vs. Actual Memory

C. *Error Behavior under Burst Workload*

Average forecasting metrics are useful for comparing model performance over the entire test split, but they may hide localized errors that are operationally important for autoscaling. In particular, an autoscaling decision becomes more sensitive to prediction error when CPU demand rises abruptly, because an underestimated or delayed forecast may shift a future state from scale_up to hold. For that reason, an additional burst-oriented error analysis was conducted on the held-out test split. This analysis was not used for model selection; instead, it was designed to examine whether the forecasting advantage observed in the previous subsection remained meaningful under volatile CPU behavior.

Burst segments are selected from the held-out test split using the 95th percentile of actual CPU utilization as the spike threshold. This procedure yields a CPU spike threshold of 0.536159, detects 67 local CPU peaks, and retains 25 non-overlapping segments after applying a three-timestep local neighborhood, a 15-timestep minimum peak gap, a 15-timestep segment radius, and a maximum of four segments per service. Table VI combines the burst error, peak-alignment, and segment-win results in one compact table.

TABLE VI
BURST WORKLOAD ERROR AND PEAK BEHAVIOR

Model	Spike RMSE	Spike MAE	Peak Lag	Wins
BiLSTM baseline	0.116831	0.078936	4.84	3/25
Transformer Encoder	0.113253	0.068521	**4.20**	6/25
BiLSTM-ED + Bahdanau, residual ON	**0.098787**	**0.060335**	5.12	**16/25**

The corresponding mean peak-underestimation values are 0.239126 for the BiLSTM baseline, 0.194555 for the Transformer Encoder, and 0.190659 for the Bahdanau residual model. CPU SMAPE on the selected spike segments follows the same order, with 19.6487% for BiLSTM, 16.3581% for Transformer Encoder, and 15.9141% for the Bahdanau residual model. Although all models achieve zero missed high-threshold rate and full crossing coverage, their mean absolute crossing delays differ slightly, namely 0.32, 0.28, and 0.24 timesteps, respectively.

The final Bahdanau residual model provides the strongest magnitude control during burst segments, with the lowest spike RMSE, spike MAE, and mean peak underestimation. It also wins 16 of the 25 selected burst segments, whereas the Transformer Encoder wins six and the BiLSTM baseline wins three. However, the Transformer Encoder obtains the lowest mean absolute peak lag and the highest peak capture rate, which indicates that the final model should not be described as superior on every spike-related indicator. The proper interpretation is more specific: the proposed forecasting champion better controls error magnitude during selected CPU spikes, while the Transformer Encoder remains competitive in peak-timing alignment. All three models record zero missed high-threshold segment rate and full crossing coverage, so those two indicators do not separate the models in this spike subset. Because all models achieve zero missed high-threshold rate and full crossing coverage, the more informative burst indicators are spike RMSE, spike MAE, peak underestimation, peak lag, and segment-level wins.

These findings clarify the practical meaning of the moderate CPU-side RMSE reduction reported on the full test split. The improvement is not confined to smoother regions: during selected CPU spikes, the Bahdanau residual model reduces squared and absolute error relative to both BiLSTM and Transformer Encoder, which is relevant because burst periods are precisely where the hold–scale_up boundary becomes time-sensitive. Nevertheless, all models still tend to underestimate abrupt CPU peaks, so the proposed model should be interpreted as improving burst-region error control rather than eliminating the inherent difficulty of sudden workload transitions.

Figure 4 presents representative CPU burst segments from the held-out test split, complementing the aggregate spike metrics in Table VI by showing how the BiLSTM baseline, Transformer Encoder, and Bahdanau residual model respond to abrupt CPU increases. This visualization supports the interpretation that the proposed model better controls burst-region error magnitude, while the Transformer Encoder remains competitive in peak-timing alignment.

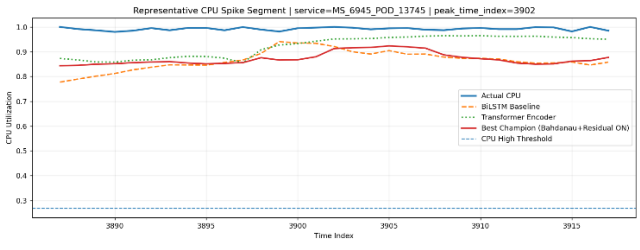

Figure 4. Representative CPU Burst Forecasting Segments

D. *Decision Classification and Feature-Set Attribution*

After the forecasting stage was evaluated, the next question was whether the predicted CPU and memory signals contributed measurable value to autoscaling decisions. The classification task was therefore formulated as a three-class decision problem with oracle labels scale_down, hold, and scale_up. These labels were not treated as manually observed

production actions; rather, they were derived from the train-only CPU and memory thresholds described in the Method section. This distinction is important because the classification stage evaluates whether the forecasting outputs help reproduce a controlled scaling rule, not whether the model has already been validated as a live production controller.

Table VII reports the oracle-label distribution used in the classification stage. The hold class is the largest class in all splits, but the test partition also contains a substantial scale_up proportion of 39.59%, which makes the decision task sensitive to errors around the upper utilization boundary. Because the classes are not perfectly balanced, macro F1 and macro recall provide a more informative view than accuracy alone. The labels are derived from train-only CPU and memory thresholds, not from human-annotated or production-executed autoscaling actions.

TABLE VII
ORACLE LABEL DISTRIBUTION ACROSS DATA SPLITS

Split	Scale Down	Hold	Scale Up	Total
Training	24,638 (19.37%)	59,634 (46.89%)	42,907 (33.74%)	127,179
Validation	5,446 (20.86%)	12,271 (47.01%)	8,386 (32.13%)	26,103
Test	4,442 (16.98%)	11,359 (43.43%)	10,354 (39.59%)	26,155

The train-only thresholds used to construct these labels are CPU_LOW = 0.0856270865, CPU_HIGH = 0.2669399977, MEM_LOW = 0.5298805833, and MEM_HIGH = 0.7188455462. The decision rule prioritizes scale_up when either target exceeds its upper threshold, assigns scale_down only when both CPU and memory are below their lower thresholds, and assigns hold otherwise.

Table VIII combines the classification baseline comparison and the feature-set attribution results. The baseline rows show why XGBoost is an appropriate model for the improvement stage: the decision task is represented as structured tabular features, and the tree-based models clearly outperform logistic regression. Random Forest remains a strong comparator and even records slightly higher test accuracy than the untuned XGBoost baseline, but XGBoost is selected by the predefined validation macro F1 criterion and gives the strongest baseline macro F1 on the test split.

TABLE VIII
CLASSIFICATION BASELINE AND FEATURE-SET ATTRIBUTION RESULTS

Model / Setting	Accuracy	Macro Precision	Macro Recall	Macro F1
Logistic Regression, WithPred	0.788262	0.824631	0.779113	0.797620
Decision Tree, WithPred	0.920321	0.924870	0.913140	0.918661
Random Forest, WithPred	0.947811	0.950146	0.945039	0.947526
XGBoost baseline, WithPred	0.947505	0.947670	0.947555	0.947609
Tuned XGBoost, NoPred	0.946052	0.945166	0.947945	0.946507
Tuned XGBoost, PredOnly	0.945861	0.945559	0.947781	0.946657
Tuned XGBoost, WithPred	**0.950602**	**0.951422**	**0.950683**	**0.951026**

For the final WithPred classifier, the additional diagnostic metrics are also strong: Cohen Kappa is 0.921035, MCC is 0.921077, Log Loss is 0.118289, and macro AUC OvR is 0.995157. Per-class F1 scores are 0.951566 for scale_down, 0.944094 for hold, and 0.957418 for scale_up. The remaining errors concentrate mainly between hold and scale_up, which is expected because these classes meet around the upper CPU or memory thresholds.

These baseline results justify XGBoost for the improvement stage because the classifier consumes structured tabular summaries rather than raw time series. The strong performance of Random Forest and XGBoost relative to logistic regression indicates that the decision boundary is nonlinear, especially around combinations of rolling memory statistics, lagged utilization, and forecast-derived CPU or memory values.

The attribution rows show that forecast-derived signals are decision-relevant but not sufficient as a replacement for observed temporal summaries. PredOnly remains competitive despite using only predicted_cpu and predicted_mem, indicating that the forecasting outputs contain substantial decision information. However, WithPred is consistently strongest, reaching 0.950602 accuracy and 0.951026 macro F1, with a test macro F1 gain of 0.004519 over NoPred and 0.004369 over PredOnly. The improvement is modest rather than dramatic, yet it supports the central attribution claim: the forecasting stage contributes measurable decision value when its predicted CPU and memory signals are integrated with recent engineered context under the same XGBoost tuning protocol.

Figure 5 shows the confusion matrix of the final tuned XGBoost WithPred model on the test split. The classifier performs strongly across all three actions, with per-class F1 scores of 0.951566 for scale_down, 0.944094 for hold, and 0.957418 for scale_up. The largest remaining errors occur between hold and scale_up: 364 hold samples are predicted as scale_up, while 498 scale_up samples are predicted as hold. This pattern is operationally meaningful because the hold–scale_up boundary is precisely where small differences in predicted CPU or memory can change the final autoscaling action. Direct confusion between scale_down and scale_up

remains very small, with only 6 scale_down samples predicted as scale_up and 8 scale_up samples predicted as scale_down.

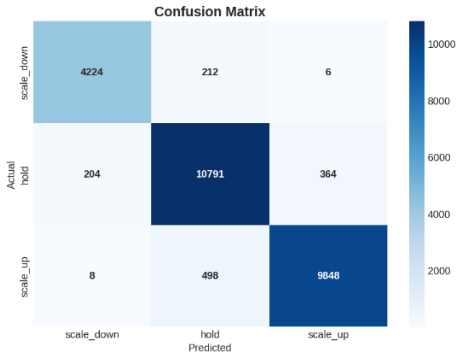

Figure 5. Confusion Matrix of Tuned XGBoost WithPred

This confusion pattern explains why the forecasting-to-decision link matters: the largest errors occur at the adjacent hold–scale_up boundary, where small differences in predicted CPU or memory may change the assigned action. Thus, the WithPred gain should be read as a boundary-sensitive improvement rather than as a complete removal of classification ambiguity.

Overall, the classification results support the prediction-to-decision attribution design of this study. XGBoost is justified empirically because it is selected by validation F1 Score and remains competitive on the held-out test split, while Random Forest serves as a strong near-tie reference rather than a weak baseline. More importantly, the NoPred, PredOnly, and WithPred comparison shows that forecast-derived CPU and memory signals contain decision-relevant information. The best performance is achieved when those signals are combined with observed engineered features, confirming that the forecasting stage contributes to the downstream classifier without requiring the paper to claim that predicted signals alone are sufficient or that the oracle labels represent live production scaling decisions.

E. Computational Overhead and Downstream Validation

The previous subsections show that the final forecasting and classification components improve predictive and decision-level metrics under the controlled evaluation protocol. However, autoscaling models should not be assessed only by accuracy, because practical deployment also depends on computational cost, inference latency, and the behavior of the resulting policy under operational constraints. For this reason, the analysis is extended in two directions. First, the main forecasting models are compared in terms of parameter count, artifact size, and inference latency. Second, the selected forecasting–classification pipeline is evaluated in a read-only trace-driven downstream simulation against horizontal and vertical autoscaling baselines. These analyses are intended to contextualize practical relevance; they should not be interpreted as evidence from a live Kubernetes deployment.

Table IX reports the accuracy–overhead comparison among the BiLSTM baseline, the Transformer Encoder comparator, and the final Bahdanau residual model. The final model gives the lowest macro RMSE, but it also has the largest parameter count and artifact size. This confirms that the forecasting champion should be interpreted as an accuracy-oriented architecture, not as the lightest model. The Transformer Encoder is the fastest model in the inference benchmark, whereas the BiLSTM baseline is the smallest in parameter count and artifact size.

TABLE IX
FORECASTING ACCURACY AND COMPUTATIONAL OVERHEAD

Model	Macro RMSE	Params (M)	Size (MB)	Inference (ms/sample)
BiLSTM baseline	0.020649	0.052930	0.648	0.026007
Transformer Encoder	0.019534	0.072706	0.969	0.018523
BiLSTM-ED + Bahdanau, residual ON	0.017188	0.240210	2.844	0.021754

Inference is reported as median milliseconds per sample. In throughput terms, the Transformer Encoder reaches approximately 53,986.77 samples per second, the final Bahdanau residual model reaches 45,967.63 samples per second, and the BiLSTM baseline reaches 38,451.39 samples per second. The final model uses about 3.30 times the parameters of the Transformer Encoder and 4.54 times the parameters of the BiLSTM baseline. Nevertheless, it remains faster than the BiLSTM baseline in this benchmark, although it is slower than the Transformer Encoder. These values should be read as an offline relative benchmark, not as production serving latency.

The overhead results indicate a trade-off rather than a one-sided advantage. The Transformer Encoder is the fastest model, whereas the final Bahdanau residual model is the most accurate but also the largest in parameter count and artifact size. Even so, its latency remains within the same practical range as the alternatives and is lower than the BiLSTM baseline in this benchmark. These latency values should be interpreted as an offline relative comparison, not as universal production-serving latency, because live performance would depend on the serving stack, hardware allocation, monitoring interval, batching strategy, and autoscaling-control integration.

The downstream validation then evaluates whether the selected forecasting–classification pipeline produces practically meaningful behavior under autoscaling-inspired policies. This validation is conducted as a read-only trace-driven simulation. No forecasting or classification model is retrained at this stage; instead, the previously selected forecasting champion and the final tuned WithPred classifier

are frozen. The compared systems are an HPA-style horizontal policy, a VPA-style vertical policy, and the proposed prediction-aware policy. Two demand conditions are considered: the Normal scenario, which follows the retained test trace, and the Stress×3 scenario, which magnifies utilization signals to examine policy behavior under heavier demand.

Table X combines the Normal and Stress×3 downstream simulations. In the Normal scenario, the proposed policy reduces SLO violation from 0.432040% under the HPA-style policy to 0.221755%, corresponding to a 48.67% relative reduction, while maintaining zero simulated downtime. Under Stress×3, the proposed policy again lowers SLO violation relative to the HPA-style policy, from 1.888740% to 1.449054%, corresponding to a 23.28% reduction. These results support the practical relevance of forecast-augmented decision making, but they also reveal a trade-off: the proposed policy has a higher provision-change rate than the HPA-style baseline and, under Stress×3, a slightly higher mean provision.

TABLE X
DOWNSTREAM VALIDATION SUMMARY

Scenario and Policy	SLO (%)	Downtime (%)	Mean Provision	Change Rate (%)
Normal — HPA-style	0.432040	0.000000	1.828484	0.202638
Normal — VPA-style	5.348882	6.610591	1.128929	6.610591
Normal — Proposed	**0.221755**	0.000000	1.511342	2.091378
Stress×3 — HPA-style	1.888740	0.000000	3.703957	1.426114
Stress×3 — VPA-style	6.484420	11.730071	3.056217	11.730071
Stress×3 — Proposed	**1.449054**	0.000000	3.755718	3.605429

The downstream results should therefore be read as an SLO-oriented trade-off under trace-driven simulation. Relative to HPA, the proposed policy reduces SLO violation by 48.67% in the Normal scenario and 23.28% under Stress×3, while also reducing underutilization from 37.9469% to 32.5597% and from 17.5683% to 7.5970%, respectively. However, it increases provision-change activity and, under stress, slightly increases mean provision relative to HPA; its P95 provisioning is also higher, rising from 3.0 to 3.5283 in the Normal scenario and from 7.0 to 9.6255 under Stress×3. Compared with VPA, the proposed policy substantially lowers SLO violation and avoids simulated downtime, but VPA remains lower on underutilization. The zero-downtime result is a consequence of the simulated in-place update assumption and should not be generalized as a live Kubernetes deployment guarantee, because real deployments would include scheduler delays, pod lifecycle behavior, admission control, monitoring latency, and platform-specific update constraints.

Figure 6 visualizes the central downstream outcome: the proposed policy lowers SLO violation relative to the HPA-style policy in both Normal and Stress×3 scenarios while maintaining zero simulated downtime under the in-place update assumption. The provisioning and change-rate trade-offs are therefore discussed from Table X rather than shown in a separate figure, so that the visual emphasis remains on the main downstream claim while the text still acknowledges the cost side of the policy behavior.

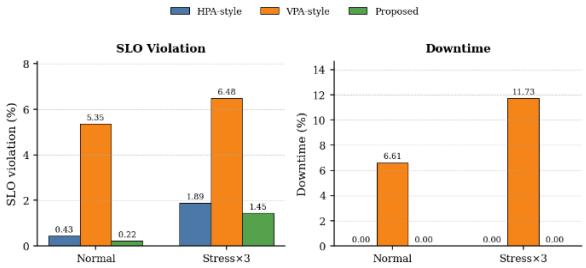

Figure 6. SLO Violation and Downtime under Normal and Stress×3 Scenarios

Taken together, the overhead and downstream analyses provide a more realistic interpretation of the proposed pipeline. The forecasting champion is more accurate but not the lightest model; its use is justified when lower forecasting error is valued more than minimal parameter count or artifact size. At the decision-policy level, the proposed trace-driven simulation reduces SLO violation relative to the HPA-style baseline in both Normal and Stress×3 scenarios and avoids simulated downtime under the in-place update assumption. However, these gains are accompanied by trade-offs in provision-change activity and, under stress, mean provision. The practical conclusion is therefore deliberately bounded: the proposed pipeline shows stronger simulation-based autoscaling behavior under the evaluated trace conditions, but its real deployment value would still require validation in a live cluster with measured control-loop latency, actuation delay, and workload diversity.

F. Generalization and Limitations

The results above show that the proposed forecasting-to-decision pipeline improves forecasting accuracy, provides measurable feature-attribution gains in the decision classifier, and yields favorable behavior in the trace-driven downstream simulation. Nevertheless, the scope of these findings is bounded by one public microservice trace family, a retained subset of 48 service instances and 47 nodes, one-step-ahead forecasting, and a preprocessing pipeline that uses ordered per-service sequence indices rather than wall-clock calendar semantics. Workloads with stronger seasonality, longer dependency structures, heavier multi-tenant interference, or

different burst characteristics may change the relative behavior of recurrent, Transformer-based, and attention-enhanced encoder–decoder models.

Two additional boundaries are important. First, the node-capacity and service-metadata variables are synthetic auxiliary covariates, not fully observed production metadata. They support controlled experimentation, but they do not prove that the same contextual effects would appear unchanged in a live cluster with authentic service priorities, replica constraints, node configurations, and application topology. Second, the decision labels are oracle labels derived from train-only CPU and memory thresholds, not human-validated or production-executed scaling actions. Thus, the NoPred, PredOnly, and WithPred comparison should be interpreted as methodological attribution: it shows that forecast-derived signals add complementary decision information under a transparent rule-based labeling scheme.

Finally, the downstream validation is a read-only trace-driven simulation that freezes the selected forecasting and classification models before evaluating HPA-style, VPA-style, and proposed policies. It connects model-level improvements to SLO violation, downtime, provisioning, and change-rate behavior, but it does not reproduce the full complexity of a live Kubernetes environment. Future work should therefore evaluate additional workload traces, authentic operational metadata, longer prediction horizons, stronger Transformer-family baselines, uncertainty-aware forecasting, sequential autoscaling decisions, and live or high-fidelity Kubernetes experiments that directly measure control-loop latency, pod readiness, actuation delay, and application-level SLO effects.

IV. CONCLUSION

This study examined whether a controlled forecasting-to-decision pipeline can improve proactive autoscaling decisions for microservice workloads while making the source of improvement attributable at each stage. The revised evaluation shows that the forecasting stage benefits from an attention-enhanced BiLSTM encoder–decoder design, particularly when Bahdanau attention is combined with a residual connection. Compared with the BiLSTM baseline, the final forecasting model reduces test RMSE from 0.020649 to 0.017188 when CPU and memory are summarized jointly, and it remains stronger than the added vanilla Transformer Encoder comparator under the same trace subset, windowing protocol, and train-only preprocessing discipline. Target-specific results further show that the improvement is asymmetric but meaningful: CPU RMSE decreases moderately to 0.028924, while memory RMSE decreases more substantially to 0.005452, which is important because both signals contribute to the downstream autoscaling decision rule.

The burst-workload analysis strengthens the practical interpretation of these forecasting results. Although the final BiLSTM encoder–decoder with Bahdanau attention and residual connection does not dominate every spike-related indicator, it achieves the lowest spike-segment RMSE and MAE and wins most selected CPU burst segments. At the same time, the Transformer Encoder shows stronger behavior on selected peak-alignment indicators, indicating that burst forecasting remains a nuanced problem involving both magnitude estimation and timing alignment. This finding prevents an overly broad claim of model superiority and clarifies that the main advantage of the selected forecasting model lies in stronger error control, especially in volatile regions where autoscaling decisions may become sensitive to prediction error.

The decision-stage results confirm that predicted CPU and memory provide complementary value when transferred into the classifier. XGBoost is justified as the final classifier because it provides the strongest validation-based selection result among the tested tabular baselines, while Random Forest remains a near-tie comparator. Under the feature-set attribution design, the WithPred setting achieves the strongest test performance, with accuracy of 0.950602 and macro F1 Score of 0.951026. The comparison among NoPred, PredOnly, and WithPred shows that forecast-derived signals are informative on their own, but their greatest value appears when they are combined with observed engineered features. The remaining classification errors are concentrated mainly between hold and scale_up, which is consistent with the threshold-sensitive nature of autoscaling decisions near upper resource boundaries.

The practical evaluation further indicates that the proposed pipeline has operational potential, but only within the limits of a read-only trace-driven simulation. The final forecasting model is more accurate than the BiLSTM baseline and Transformer Encoder comparator, although it requires a larger parameter budget and artifact size. Its measured inference latency remains low in the shared benchmark, suggesting that the accuracy gain does not introduce an excessive inference-time burden in the evaluated environment. In downstream simulation, the proposed policy reduces SLO violation relative to the HPA-style baseline under both Normal and Stress×3 scenarios and avoids the simulated downtime associated with the VPA-style baseline. However, these gains are accompanied by higher provision-change activity and, under stress, slightly higher mean provision than the HPA-style policy. Therefore, the downstream result should be interpreted as a favorable SLO-oriented trade-off under simulation assumptions, not as proof of live deployment performance.

Overall, the contribution of this study lies not merely in applying attention or using a two-stage pipeline, but in combining strict temporal evaluation, fixed-backbone attention ablation, explicit feature-set attribution, burst-oriented error analysis, and trace-driven downstream validation. The findings suggest that improving multivariate forecasting can strengthen autoscaling decision quality when forecast-derived signals are integrated carefully into a supervised decision model. Nevertheless, the study remains bounded by one public trace family, a retained subset of 48

service instances and 47 nodes, synthetic auxiliary metadata, one-step-ahead forecasting, rule-based oracle labels, and simulation-based downstream validation. Future work should extend this pipeline to additional workload traces, real operational metadata, longer prediction horizons, sequential autoscaling decisions, stronger Transformer-family baselines, uncertainty-aware forecasting, and live or high-fidelity Kubernetes experiments that measure control-loop latency, actuation delay, pod readiness behavior, and application-level SLO effects directly.

REFERENCES

[1] Kubernetes Authors, "Horizontal Pod Autoscaling," *Kubernetes Documentation*. 2025. Accessed: Dec. 08, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/

[2] H. Ahmad, C. Treude, M. Wagner, and C. Szabo, "Towards resource-efficient reactive and proactive auto-scaling for microservice architectures," *Journal of Systems and Software*, vol. 225, p. 112390, 2025, doi: https://doi.org/10.1016/j.jss.2025.112390.

[3] Kubernetes Authors, "Vertical Pod Autoscaling," *Kubernetes Documentation*. 2025. Accessed: Dec. 08, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/autoscaling/vertical-pod-autoscale/

[4] Z. Wang, S. Zhu, J. Li, W. Jiang, K. K. Ramakrishnan, Y. Zheng, M. Yan, X. Zhang, and A. X. Liu, "DeepScaling: microservices autoscaling for stable CPU utilization in large scale cloud systems," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 16–30. doi: 10.1145/3542929.3563469.

[5] P. B. Guruge and Y. H. P. P. Priyadarshana, "Time series forecasting-based Kubernetes autoscaling using Facebook Prophet and Long Short-Term Memory," *Frontiers in Computer Science*, vol. Volume 7-2025, 2025, doi: 10.3389/fcomp.2025.1509165.

[6] J. Yoo and U. Kang, "Attention-Based Autoregression for Accurate and Efficient Multivariate Time Series Forecasting," in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 531–539. doi: 10.1137/1.9781611976700.60.

[7] Y. Li and D. C. Anastasiu, "Multivariate Segment Expandable Encoder-Decoder Model for Time Series Forecasting," *IEEE Access*, vol. 12, pp. 185012–185026, 2024, doi: 10.1109/ACCESS.2024.3513256.

[8] T. Vanderschueren, T. Verdonck, B. Baesens, and W. Verbeke, "Predict-then-optimize or predict-and-optimize? An empirical evaluation of cost-sensitive learning strategies," *Information Sciences*, vol. 594, pp. 400–415, 2022, doi: https://doi.org/10.1016/j.ins.2022.02.021.

[9] J. Mandi, J. Kotary, S. Berden, M. Mulamba, V. Bucarey, T. Guns, and F. Fioretto, "Decision-Focused Learning: Foundations, State of the Art, Benchmark and Future Opportunities," *J. Artif. Int. Res.*, vol. 80, Sep. 2024, doi: 10.1613/jair.1.15320.

[10] Alibaba Open Source, "Alibaba Cluster Trace: Microservices v2021," *GitHub*, 2021. Accessed: Dec. 08, 2025. [Online]. Available: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2021

[11] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 412–426. doi: 10.1145/3472883.3487003.

[12] D. Huye, L. Liu, and R. R. Sambasivan, "Systemizing and Mitigating Topological Inconsistencies in Alibaba's Microservice Call-graph Datasets," in *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, 2024, pp. 276–285. doi: 10.1145/3629526.3645043.

[13] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11106–11115, May 2021, doi: 10.1609/aaai.v35i12.17325.

[14] S. Almaghrabi, M. Rana, M. Hamilton, and M. S. Rahaman, "Multidimensional dynamic attention for multivariate time series forecasting," *Applied Soft Computing*, vol. 167, p. 112350, 2024, doi: https://doi.org/10.1016/j.asoc.2024.112350.

[15] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, and M. Lindauer, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *WIREs Data Mining and Knowledge Discovery*, vol. 13, no. 2, p. e1484, 2023, doi: https://doi.org/10.1002/widm.1484.

[16] Optuna Contributors, "Optuna: A hyperparameter optimization framework," *Optuna Documentation*, 2025. Accessed: Dec. 25, 2025. [Online]. Available: https://optuna.readthedocs.io/en/stable/

[17] A. Bali, Y. El Houm, A. Gherbi, and M. Cheriet, "Automatic data featurization for enhanced proactive service auto-scaling: Boosting forecasting accuracy and mitigating oscillation," *Journal of King Saud University - Computer and Information Sciences*, vol. 36, no. 2, p. 101924, 2024, doi: https://doi.org/10.1016/j.jksuci.2024.101924.

[18] Z. Pan, Y. Wang, Y. Zhang, S. Bin Yang, Y. Cheng, P. Chen, C. Guo, Q. Wen, X. Tian, Y. Dou, Z. Zhou, C. Yang, A. Zhou, and B. Yang, "MagicScaler: Uncertainty-Aware, Predictive Autoscaling," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3808–3821, Aug. 2023, doi: 10.14778/3611540.3611566.

[19] M. Mekki, B. Brik, A. Ksentini, and C. Verikoukis, "XAI-Enabled Fine Granular Vertical Resources Autoscaler," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, Jun. 2023, pp. 161–169. doi: 10.1109/NetSoft57336.2023.10175438.

[20] H. Hewamalage, K. Ackermann, and C. Bergmeir, "Forecast evaluation for data scientists: common pitfalls and best practices," *Data Mining and Knowledge Discovery*, vol. 37, no. 2, pp. 788–832, 2023, doi: 10.1007/s10618-022-00894-5.

[21] scikit-learn developers, "Common pitfalls and recommended practices," *scikit-learn Documentation*, 2025. Accessed: Dec. 25, 2025. [Online]. Available: https://scikit-learn.org/stable/common_pitfalls.html

[22] scikit-learn developers, "Metrics and scoring: quantifying the quality of predictions," *scikit-learn Documentation*, 2025. Accessed: Dec. 08, 2025. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html

[23] M. A. Barata, D. Irnawati, I. Wisma, D. Prastya, and D. I. Hastuti, "Hydrogen Sulfide Leak Detection Using The C4.5 Algorithm: Optimizing Feature Extraction For Enhanced Accuracy," *PROCEEDING Al Ghazali Internasional Conference*, vol. 1, pp. 348–358, 2025, doi: https://doi.org/10.52802/aicp.v1i1.1352.

[24] M. A. Barata, E. Noersasongko, Purwanto, and M. A. Soeleman, "Improving the Accuracy of C4.5 Algorithm with Chi-Square Method on Pure Tea Classification Using Electronic Nose," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 7, no. 2 SE-Computer Science Applications, Mar. 2023, doi: 10.29207/resti.v7i2.4687.

[25] M. J. Vikri, I. Wisma, D. Prastya, U. P. Sanjaya, and M. A. Barata, "Rice Quality Identification For Indonesian Food Standards Based On Electronic Nose Berdasarkan Standar Pangan Indonesia Berbasis," *INOVTEK Polbeng - Seri Informatika*, vol. 10, no. 1, 2025, doi: https://doi.org/10.35314/0y0xct32.

[26] scikit-learn developers, "SVR," *scikit-learn Documentation*, 2025. Accessed: Jan. 05, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[27] scikit-learn developers, "MultiOutputRegressor," *scikit-learn Documentation*, 2025. Accessed: Jan. 05, 2026. [Online]. Available: https://scikit-

learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html
[28] Keras Developers, "LSTM layer," *Keras Documentation*, 2025. Accessed: Jan. 05, 2026. [Online]. Available: https://keras.io/api/layers/recurrent_layers/lstm/
[29] Keras Developers, "GRU layer," *Keras Documentation*, 2025. Accessed: Jan. 05, 2026. [Online]. Available: https://keras.io/api/layers/recurrent_layers/gru/
[30] Keras Developers, "Bidirectional layer," *Keras Documentation*, 2025. Accessed: Jan. 05, 2026. [Online]. Available: https://keras.io/api/layers/recurrent_layers/bidirectional/
[31] A. C. R. Klaar, S. F. Stefenon, L. O. Seman, V. C. Mariani, and L. dos S. Coelho, "Optimized EWT-Seq2Seq-LSTM with Attention Mechanism to Insulators Fault Prediction," *Sensors*, vol. 23, no. 6, 2023, doi: 10.3390/s23063202.
[32] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A Transformer-based Framework for Multivariate Time Series Representation Learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery \& Data Mining*, 2021, pp. 2114–2124. doi: 10.1145/3447548.3467401.
[33] xgboost developers, "XGBoost Parameters," *XGBoost Documentation*, 2025. Accessed: Dec. 25, 2025. [Online]. Available: https://xgboost.readthedocs.io/en/stable/parameter.html
[34] scikit-learn developers, "RandomForestClassifier," *scikit-learn Documentation*, 2025. Accessed: Feb. 15, 2026. [Online]. Available: https://sklearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[35] scikit-learn developers, "LogisticRegression," *scikit-learn Documentation*, 2026. Accessed: Feb. 15, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[36] scikit-learn developers, "DecisionTreeClassifier," *scikit-learn Documentation*, 2026. Accessed: Apr. 25, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[37] E. F. Laili, Z. Alawi, R. Rohmah, and M. A. Barata, "Komparasi Algoritma Decision Tree Dan Support Vector Machine (SVM) Dalam Klasifikasi Serangan Jantung," *Jurnal Sistem Informasi dan Informatika*, vol. 8, no. 1 SE-Articles, pp. 67–76, doi: 10.47080/simika.v8i1.3683.
[38] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022, doi: https://doi.org/10.1016/j.inffus.2021.11.011.