POLITECNICO DI TORINO
Repository ISTITUZIONALE

Real-time latency prediction for cloud gaming applications

Original
Real-time latency prediction for cloud gaming applications / Monaco, D., Sacco, A., Spina, D., Strada, F., Bottino, A., Cerquitelli, T., Marchetto, G.. - In: COMPUTER NETWORKS. - ISSN 1389-1286. - 264:(2025). [10.1016/j.comnet.2025.111235]

Availability:
This version is available at: 11583/3001203 since: 2025-06-22T18:24:53Z

Publisher:
Elsevier

Published
DOI:10.1016/j.comnet.2025.111235

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

26 November 2026

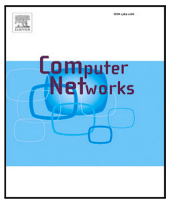
Computer Networks 264 (2025) 111235



Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet





Real-time latency prediction for cloud gaming applications

Doriana Monaco *, Alessio Sacco, Daniele Spina, Francesco Strada, Andrea Bottino, Tania Cerquitelli, Guido Marchetto

Department of Control and Computer Engineering, Politecnico di Torino, Italy

ARTICLE INFO

Keywords:
Latency prediction
Cloud gaming
Concept drift

ABSTRACT

Cloud gaming represents a rapidly growing segment in the entertainment industry, allowing users to stream and interact with high-quality games over the Internet. However, the problem of maintaining a seamless gaming experience is inherent to minimizing user-perceived latency. In this paper, we present CLoud Application lAtency Prediction (CLAAP), a novel solution that, to tolerate challenged network conditions in gaming, predicts such latency via a Machine Learning (ML) model and forecasts future network evolution. The model, trained over diverse network conditions and gaming scenarios, can then update its parameters via a concept drift detection algorithm that suggests a re-training action, reducing the prediction error up to 21% with minimal overhead. We then integrate this network metrics predictor into a game state prediction to further tolerate network latency spikes even from the user perspective, who can continue playing even in adversarial conditions without session interruptions. The results suggest the potential of advanced predictive analytics in mitigating latency issues, thereby setting the stage for more responsive and immersive cloud gaming services.

1. Introduction

The advent of cloud gaming has revolutionized the gaming industry by allowing users to stream and play high-quality video games on virtually any device with internet connectivity. This shift has removed the need for expensive hardware and has democratized access to cutting-edge gaming experiences. However, cloud gaming comes with higher bandwidth requirements and stricter constraints compared to traditional gaming. Unlike conventional console games that require 100–200 Kbps, cloud gaming demands at least 10–20 Mbps. More importantly, latency must remain below 100 ms to guarantee an immersive and seamless experience [1,2], as the delay between a user's input and the corresponding action in the game, can severely impact the gameplay experience, leading to frustration and reduced engagement [3–5].

Recent approaches attempted to minimize latency for maintaining the responsiveness and fluidity required for an enjoyable gaming experience [6–8]. Traditional approaches involve optimizing network infrastructure and server placement [9–12], while recent latency compensation techniques include server-side assistance for the user [13] and methodologies to adjust in-game objects [14]. However, these methods often fall short due to the dynamic nature of network conditions and the varying demands of different games and users. As a result, there is a growing need for more sophisticated solutions that can predict and adapt to latency in real-time. Some studies suggest to employing speculative solutions to hide the network delay, such as in [15], which proposes to send future frames of the game in advance to the client, where they are locally pre-fetched, reducing the perceived network delay to be zero. Based on this action-based video pre-fetching paradigm, new research proposals enhance the prediction of user actions to later send the most probable frame [6,16]. Nevertheless, continuously applying such mechanisms results in significant resource consumption and, since accurately predicting future frames is challenging and incorrect predictions can lead to visual artifacts and inconsistencies, in the long run it may harm user experience.

In this paper we advance these techniques by proposing CLoud Application lAtency Prediction (CLAAP). The solution is based on the idea of predicting the future network latency per user, such that whenever it becomes unbearable, an action prediction mechanism is activated to predict future game states (and frames) that are sent to the user in advance, until the network performance is re-established. Two main challenges arise related to the adoption of such a mechanism. The first challenge relies on the design of the user latency predictor, because of the varying cloud-to-user delays [17]. While traditional forecasting methods perform statistical analysis on the available data to predict future values, often requiring manual tuning of the parameters and employing a non-trivial amount of time for the inference, learning methods adapt well to a real-time scenario thanks to their fast execution. We model cloud gaming latency as a time series and analyze its

* Corresponding author.
E-mail address: doriana.monaco@polito.it (D. Monaco).

https://doi.org/10.1016/j.comnet.2025.111235
Received 17 December 2024; Received in revised form 14 March 2025; Accepted 16 March 2025
Available online 10 April 2025
1389-1286/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

characteristics. Our analysis reveals that these characteristics include locality and non-linearity, which are crucial for effective prediction. We then propose a model for prediction based on the Radial Basis Function Neural Network (RBFNN), which is particularly well-suited for this task due to the localized responses to input. By employing a Gaussian Radial Basis Function, these networks can act as universal approximators, enabling them to learn and predict complex non-linear patterns inherent in the latency data. The second challenge is related to the inability of machine learning (ML) models to adapt to concept drift [18,19], i.e., changes in the data distribution at testing time. While some detectors and mitigators have already been proposed leveraging KSWIN, DDM, and Page Hinkley, we model concept drift as change points, employ the Bayesian Online Changepoint Detection to estimate the occurrence of drift points, and later online fit the model on the most recent data for adaptation.

These components form CLAAP, which we extensively evaluate under multiple metrics and environments, including a real-world testbed. We test both Offline-CLAAP a plain predictor without adaptation module, and Online-CLAAP that can adapt to per user concept drift. Experiments demonstrate that Offline-CLAAP outperforms state-of-the-art solutions for time series forecasting in terms of accuracy and inference time, and with the little extra computation and time required by Online-CLAAP we can obtain an error reduction of 21% on average. Finally, we integrate it with a real game scenario [20], where the latency prediction is used in conjunction with a game state prediction to further address challenges in network conditions. Results validate that this system significantly enhances user performance, eliminating rendering drops (in terms of frames per second) entirely.

The rest of the paper is structured as follows. We first review state-of-the-art methods for latency forecasting and concept drift detection in Section 2. Subsequently, we model cloud gaming latency as a time series, study its properties, and motivate the use of RBFNN for forecasting in Section 3. In Section 4, we define concept drift in this scenario and describe in detail our detection algorithm. Next, we evaluate CLAAP prediction performance (Section 5) and its effect in a real-world cloud gaming testbed (Section 6). Lastly, we draw the conclusions of our work in Section 7.

2. Related work

In this section, we discuss and compare against existing similar works, highlighting the differences. We present the existing techniques used for time series forecasting, particularly for the latency prediction task. Later, we introduce the problem of concept drift and available methods to detect and mitigate it.

2.1. Latency forecasting

Monitoring QoS measurements such as throughput, jitter, and delay is essential to satisfy the requirements of delay-sensitive applications. In particular, an accurate estimator of future delays is essential to perform proactive actions that mitigate the impact of the latency on such applications. Numerous solutions have been proposed in the literature, broadly categorized into statistical methods and machine learning (ML) models.

Traditionally, forecasting methods were based on the statistical analysis of past data to estimate the future. Popular examples are Exponential smoothing [21], particularly efficient for online and short-term forecasting, and ARIMA [22], a simple yet robust method suited for series that exhibit trends and seasonality. These approaches assume data to be stationary and that the future linearly depends on the past, so they require pre-processing for non-stationary data and fail to capture non-linear patterns. On the other hand, non-linear methods, e.g., ARCH [23], typically model a specific behavior and, therefore, do not generalize to different patterns [24].

Recently, learning-based solutions have become popular for their ability to capture complex patterns from raw data. A comparison between statistical and ML approaches is conducted in [25], where LSTM proves superior w.r.t. ARIMA and GRU models. [26] compares Radial Basis Function neural networks (RBFNN) and Elman neural networks to forecast latency in vehicular networks, establishing a superior performance of RBFNN, while [27] leverages LSTM to predict latency violations in a B5G network, to accordingly steer or migrate the traffic between cloud and edge devices.

Latency prediction is not always modeled as a time series forecasting problem. A different approach is adopted in [28,29], where the latency prediction problem is modeled as a classification task, and ML classifiers predict whether the next value belongs to a specific range or exceeds a threshold. DeepQueueNet [30] models the prediction of per-packet latency as a sequence-to-sequence problem and adopts a Transformer, with the final goal of estimating the end-to-end performance of the network. Reparo [31] introduces a predictive approach for latency-sensitive applications that uses adversarial autoencoders to restore lost application-layer data. Furthermore, PERT-GNN [32] is a GNN model designed to forecast the end-to-end latency of microservice-oriented applications by analyzing API calls and resource usage.

We model the user latency prediction problem as a time series forecasting problem and use RBFNN to estimate the next step value based on a historical input for its ability in capturing locality and non-linear patterns. Unlike related works, our model is enriched with a concept drift detection module that adapts to the observed time series online.

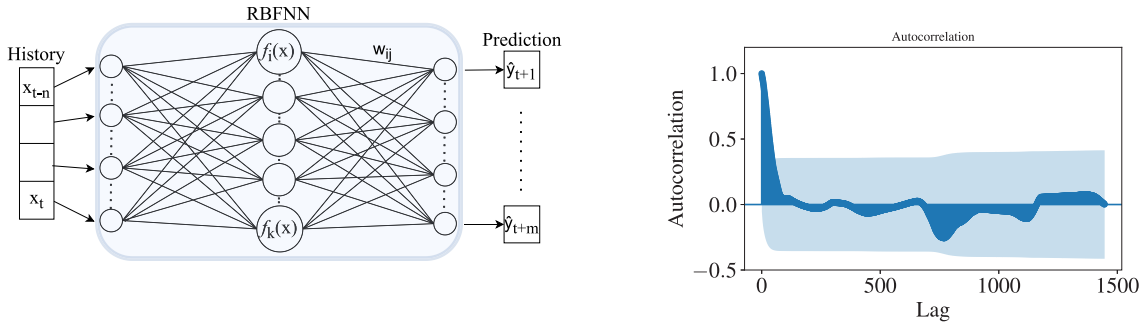
2.2. Concept drift detection and adaptation

As discussed in the previous paragraph, Machine Learning has been widely explored in many networking problems, including network forecasting. However, the performance of ML models is stable when the test data follows the same rules of the data seen during training, which is not always the case, especially in real-world production environments [18]. A change in the relationship between output and input features is referred to as concept drift, and we now discuss the existing methods to address this issue.

The first step concerns the detection of the drift, which is explored in [33], that adopts an ensemble method to detect changes in the distribution of Key Performance Indicators (KPIs), in Drift Lens [34] that studies per-label features distribution to detect concept drift in the labels of a classifier, and in [35], which investigates three detection methods, i.e., ADWIN, DDM and Page Hinkley, to detect drift in sensor loads. None of these papers, however, investigate adaptation methods once the drift is detected. On the contrary, LEAF [36] leverages KSWIN for the detection and later combines *forgetting* and *oversampling* for the re-training, while authors in [37] monitor the prediction error distribution of their Transformer to assign a drift level and consequently adjust the learning rate based on the assessed level. Both studies evaluate the performance of the model under concept drift but do not provide insights on the impact that this additional module has in terms of computing processing. We propose a Bayesian online changepoint detection approach to assess the presence of concept drift. In case of a positive outcome, we incorporate the newly distributed data with "on-the-fly" training over the last batch of data. We demonstrate that our model performance effectively improves with minimal extra computation.

3. Predicting user latency via RBF model

Radial Basis Function neural network has lately emerged as a powerful tool in a variety of tasks, i.e., image recognition, data forecasting, medical diagnosis [38–41], thanks to its ability in modeling non-linear interactions. This neural network comprises three layers (Fig. 1(a)): an input layer, a hidden layer with a non-linear activation function, and an output layer. Our input is constituted by a vector representing the



(a) Architecture of RBFNN.

(b) Autocorrelation of cloud gaming user latency.

Fig. 1. RBFNN hidden neurons are *centers* that produce a localized response through the Gaussian radial basis function, making them ideal for learning about the locality and non-linearity of cloud gaming latency.

Table 1
Summary of key notation.

Symbol	Description
x	Historical input
n	Input length
k	Number of hidden neurons
$f(x)$	Activation function
c	Center of radial basis function
σ	width of radial basis function
$\hat{y}$	Prediction output
m	Output length
r	Run length
$p(r_t \mid x_{1:t})$	Run length posterior distribution
π_t^r	Predictive distribution of x_t for each r
$p(r_t \mid r_{t-1})$	Changepoint prior
$H(\cdot)$	Hazard function

last n measured latency values. The hidden layer comprises neurons that produce a localized response $f_i(x)$ through radial basis functions centered at specific data points. We use the Gaussian radial basis function, which is defined for each neuron in the following way:

$$f_i(x) = \phi(\|x - c_i\|) = e^{-\frac{\|x-c_i\|^2}{2\sigma_i^2}}, 1 < i < k \tag{1}$$

In this formulation, ϕ is the standard normal probability density function of the Euclidean norm $\|\cdot\|$ of the difference between the input x and the centers of the functions c_i, σ is the width of the function and k identifies the number of hidden neurons. A final linear layer combines the responses of the hidden layer nodes to produce the prediction for the next m values in this way:

$$\hat{y}_j = \sum_i w_{ij} f_i(x), 1 < i < k, 1 < j < m \tag{2}$$

where w_{ij} are the learnable parameters and m identifies the output length. See Table 1 for the complete list of symbols.

The idea is to transform the input data into a higher dimensional layer where the data becomes linearly separable and use the distance between the input and the centers of hidden functions as a form of similarity, such that data points closer to each other in the input space produce similar values in the output space. Centers and widths of the activation functions are determined a-priori and kept fixed during the training, while the backpropagation updates the weights between the hidden and output layers. We first select the centers through K-Means clustering. Then, the K-nearest neighbors model computes the standard deviation of the distances between each neuron and its neighbors, determining the widths. Thanks to the localized activation of a subset of hidden units during the backpropagation process and the simple derivative of the Gaussian function, this neural network is very fast to train and use for inferring.

Additionally, this type of network is a powerful tool for function approximation, capable of modeling any continuous function when an appropriate RBF is selected. Precisely, if the RBF is continuous almost everywhere, locally essentially bounded, and non-polynomial, the network can approximate any continuous function with respect to the uniform norm [42,43]. The Gaussian RBF satisfies these conditions, enabling the network to act as a universal approximator.

To further motivate the use of RBFNN, we analyze the properties of latency in cloud gaming sessions. When a user connects to the cloud gaming application, at first, some flows are established between the user and the platform in order to administrate the account, perform the login, and manage related services (*platform administration flows*). After the game is selected, new flows are initiated to conduct network diagnostics to determine the optimal cloud server for the session (*platform management flows*). Once the gameplay begins, three key flows are established: *gaming management flows* to monitor the latency, *input flows* to transmit user commands, and *media streaming flows* to handle video and audio transmission. After the session ends, *platform administration flows* restart as the user returns to the main interface. While the flows related to platform administration and management are negligible in volume, the flows related to the gameplay session are predominant and largely influence the user gaming experience [2]. We now examine the autocorrelation of a latency time series collected in a Cloud Gaming Study [44].

Autocorrelation measures the similarity between values at different time lags, and as shown in Fig. 1(b), recent values have a more substantial influence on future values than those further in the past. This reflects the locality property of latency. Additionally, latency fluctuates over time in a non-linear manner, suggesting that simple linear models may not fully capture its behavior. As a result, cloud gaming latency exhibits both non-linearity and locality.

Given these characteristics, an RBF network is particularly well-suited for modeling cloud gaming latency. Its hidden layer consists of neurons that produce localized responses, making it highly effective at capturing short-range dependencies (locality). At the same time, the non-linear Gaussian activation function allows it to learn complex patterns (non-linearity). These theoretical considerations are later validated through our experiments.

4. Concept drift

ML models work well when the input state distribution is consistent over time [18,19]. However, it is not rare to face unpredictable changes in the distribution of time series data caused by the dynamic nature of real-world systems. A *change point* occurs when the underlying process generating the time series transitions to a different state [45]. In the context of time series forecasting, such shifts lead to *concept drift* because the relationship between input features and target variables changes over time [19,46], causing models trained on historical data to degrade in predictive performance. Formally, the concept drift mathematical definition is: $\exists X : p_{t_0}(X, y) \neq p_{t1}(X, y)$, where $p_t(X, y)$ is the

joint probability of the input X and the target variable y at time $t0$ and is referred to as *concept* [19,47].

This challenge is particularly relevant in cloud networks, where studies have highlighted significant latency variability [17,48–50]. These findings underscore the need for robust drift detection and adaptation mechanisms in online latency forecasting frameworks. However, this is challenging in an online setting like ours, where data are collected one value at a time, and properties cannot be known a priori. To solve this issue, we use an online Bayesian changepoint detection approach, which uses past data to estimate the properties of incoming data, thus fitting our application scenario.

4.1. Bayesian online concept drift detection

Our online concept drift detection works as follows. Given a sequence of data $x_{1:t} = x_1, \dots, x_t$, the sequence can be partitioned such that each partition contains i.i.d. samples from a distribution. A *change point* is identified as the point that separates two partitions such that the statistical properties governing these partitions are different [51].

The *run length* identifies the time elapsed from the last change point, such that:

$$r_t = \begin{cases} 0, \text{if change point at time t} \\ r_{t-1} + 1, \text{otherwise} \end{cases} \tag{3}$$

The basic objective of Bayesian online change point detection [51] is to infer the *run length posterior* distribution $p(r_t|x_{1:t})$, i.e., estimate the future values of the run length to compute the probability that a new change point occurs. Once we estimate the occurrence of concept drift, we update the model to reflect these changes. For Bayes theorem, the *run length posterior* is given by:

$$p(r_t|x_{1:t}) = \frac{p(r_t, x_{1:t})}{p(x_{1:t})} \tag{4}$$

where the probability of the sequence $x_{1:t}$ is obtained by summing the joint probabilities of the run length and the sequence, over all possible values of the run length at time t:

$$p(x_{1:t}) = \sum_{r_t=0}^{t} p(r_t, x_{1:t}) \tag{5}$$

The joint probability can be computed recursively:

$$\begin{aligned} p(r_t, x_{1:t}) &= \sum_{r_{t-1}} p(r_t, r_{t-1}, x_{1:t}) = \\ &= \sum_{r_{t-1}} p(x_t|r_{t-1}, x_t^r) p(r_t|r_{t-1}) p(r_{t-1}, x_{1:t-1}) \end{aligned} \tag{6}$$

The first term is the *predictive distribution* of the new sample and represents the probability of observing x_t given the data collected up to time $t-1$ and assuming that the current run length is r. The second factor is the *changepoint prior* and encodes our prior knowledge about where change points are likely to occur. It is parametrized through the *Hazard function* in the following way:

$$p(r_t|r_{t-1}) = \begin{cases} H(r_{t-1}+1), \text{if } r_t = 0 \\ 1 - H(r_{t-1}+1), \text{if } r_t = r_{t-1}+1 \\ 0, \text{otherwise} \end{cases} \tag{7}$$

where $H(r_{t-1})$ is a constant function $\in (0,1)$ that we set $H(r_{t-1}) = 1/250$ (see Section 5.6 for more details).

Since the joint probability at time t depends on time $t-1$, we can run a recursive algorithm to compute the joint probability over run-length and observed data based on message-passing of the term $p(r_{t-1}, x_{1:t-1})$. We initialize the *predictive distribution* with a normal-gamma distribution, denoted by four parameters, and since it depends on the most recent data only, we keep track of its parameters at time $t-1$ to compute the probability at time t.

The pseudocode of the whole agent prediction routine, comprehensive of the concept drift detection routine, is presented in Algorithm 1. Whenever a client observers a new latency value, it sends it to the agent, along with its client id *cid*, to predict the next value. The agent stores the value in two circular buffers for each client, the *batch* buffer contains the latest samples used for drift detection, the *history* buffer is the input of the model. At first, if the *batch* buffer is full, it evaluates the presence of concept drift and eventually online fits the model to this latest batch of data, and the buffer is emptied. Then, if enough historical values are collected, it predicts the next value to be sent to the client, otherwise, a symbolic value of -1 is returned.

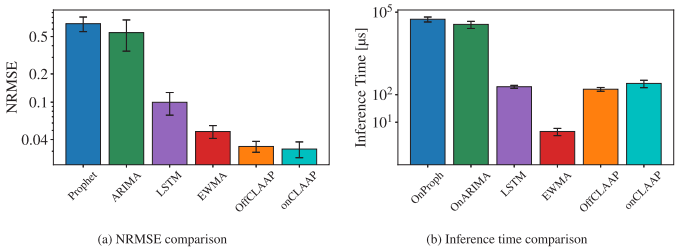
We now explain how the *concept_drift_detector* routine works. For each new sample x_t, it evaluates the *predictive distribution* under each of the parameters and for each possible value of run length; therefore, π_{r_t} contains t probabilities. Once it evaluates the hazard function on r_{t-1}, it computes the *growth probabilities* (the probability that the run length increases) for each run length value using (6), except for $r_t = 0$, which is the *changepoint probability* and is computed similarly in the successive step. Combining this information, it can determine the *run length posterior* following (4). Finally, the *predictive distribution* parameters are updated and the learned knowledge is passed to the following sample x_{t+1}. Once we know the *run length posterior*, we can establish the occurrence of concept drift at timestep t if the probability exceeds a threshold.

Algorithm 1: CLAAP prediction routine

```
cid // client id
new_value // last value observed by client
batch = {} // per client batch buffers
history = {} // per client history buffers
model = {} // per client neural network
μ_0^0 = mean(trainSet), λ_0^0 = variance(trainSet)
α_0^0 = 0.1, β_0^0 = 0.01
π ~ NormalGamma(μ, λ, α, β)
H = 1/250
predict(cid,new_value):
    add new_value to batch[cid]
    add new_value to history[cid]
    if batch[cid] is full then
        retrain =concept_drift_detector(batch[cid]);
        if retrain then
            fit model[cid] over batch[cid];
        reset batch[cid]
    if history[cid] is full then
        model[cid] infers ŷ from history[cid]
    else
        ŷ = −1
    return ŷ
concept_drift_detector(batch):
    for t, x_t ∈ enumerate(batch) do
        π_t^r = p(x_t|μ_t^r, λ_t^r, α_t^r, β_t^r)
        h = H(r_{t−1})
        p(r_t = r_{t−1} + 1|x_{1:t}) = (1 − h) · p(r_{t−1}|x_{1:t−1})π_t^r
        p(r_t = 0|x_{1:t}) = Σ_{r_{t−1}} h · p(r_{t−1}|x_{1:t−1})π_t^r
        p(r_t|x_{1:t}) = p(r_t,x_{1:t}) / Σ_{r_t} p(r_t,x_{1:t})
        update μ_{t+1}^r, λ_{t+1}^r, α_{t+1}^r, β_{t+1}^r
    return p(r_t|x_{1:t})
```

5. Latency predictor evaluation

In this section, we test our proposed approach in multiple settings. We first evaluate Offline-CLAAP, which is not equipped with the concept drift adaptation module, against state-of-the-art time series



(a) NRMSE comparison (b) Inference time comparison

Fig. 2. SOTA comparison. CLAAP exhibits a low inference time while being the most accurate solution.

forecasting methods in terms of prediction error and learning time. Later on, we demonstrate the efficacy of Online-CLAAP to adapt to drifts in the data distribution in a production setting. Finally, we report the sensitivity studies of multiple parameters.

5.1. Experimental settings

We used the data collected by a Cloud Gaming Study [44] for the evaluation. In particular, we extracted the delay data from QoS/QoE measured while playing "Dirt 4" (available on Google Stadia cloud gaming platform), a fast-paced racing game that demands rapid, continuous user inputs, much like first-person shooters. Such intense gameplay imposes stressful networking conditions and serves as a challenging benchmark for our latency-prediction framework. The delay captures were made through WebRTC under different emulated 4G network conditions, and measures were taken every 1 ms. We leveraged this data to simulate an interactive application with a refresh rate of 60 Hz, and built a training dataset picking values every 16 ms. The resulting training set is composed of 11283 values. The tests were run on a Ubuntu machine equipped with an Intel(R) Core(TM) i7-4770 processor, featuring 8 cores and 16 threads, running at a CPU clock speed of 3.40 GHz. The system had 32 GB RAM. CLAAP was evaluated with parameters $k = 20$, $n = 6$, and $m = 1$, whose default values are motivated by the sensitivity analysis conducted in Section 5.6. The error bars plotted in the graph represent the 95% confidence intervals.

5.2. Benchmark

- ARIMA: this model assumes that the time series adheres to a statistical distribution, such that future values are correlated to past ones. It consists of three components: the Auto Regressive (AR) module, which defines the variable as a linear combination of past ones; the Integrated (I) part, which differentiates the time series to make it stationary in case it is not; the Moving Average (MA) component, which determines the error as a linear combination of past errors. The ARIMA(p, d, q) forecasting process, where p represents the order of autocorrelation, d denotes the order of differencing, and q indicates the size of the moving average window, is mathematically expressed as: $\hat{y} = \rho_0 + \sum_{i=1}^{p} \rho_i y_{t-i} + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$, where ρ_i are the autoregressive coefficients, ρ_0 is a constant, ϵ_t is white noise, θ_i are moving average parameters and y_{t-i} is the data point at step $t - i$.
- LSTM: a special Recurrent Neural Network (RNN) equipped with a memory c_t at time t. The activation of an LSTM cell is given by $a_t = o_t tanh(c_t)$, where o_t is the *output gate* that modulates the content transferred to the predictor. The output gate is computed with a linear sum of the existing state and the new state: $o_t = \sigma(W_o [x_t, a_{t-1}]) + b_0$, where σ is the sigmoid function, W_0 and b_0 are weights and biases, x_t is the input at time step t, a_{t-1} is the previous cell output. The state of the current unit is given by including new data and partially forgetting the existing memory: $c_t = g_t c_{t-1} + i_t \tilde{c}_t$, where g_t is the forget gate and i_t the input gate.
- Prophet [52]: created by Facebook in 2017, is a publicly available forecasting tool designed to predict future outcomes automatically. Its focus lies in analyzing nonlinear time series data that exhibit patterns such as yearly, weekly, or daily cycles, as well as holiday impacts. Additionally, it can manage missing data points and identify outliers within the dataset. It is a decomposable time series model that combines trend, seasonality, and holidays in the following way, respectively: $y(t) = w(t) + s(t) + z(t) + \epsilon_t$, where ϵ_t is an error term.
- EWMA: Exponentially Weighted Moving Average is a statistical technique that assigns greater weight to recent observations, with weights decreasing exponentially for older data. $y_t = (1 - \alpha) y_{t-1} + \alpha x_t$, with $0 < \alpha \leq 1$ being a smoothing factor.

5.3. Evaluation metrics

Throughout the experiments, we analyze the inference time, the training time, the resource usage percentage but, mainly, the normalized root mean squared error (NRMSE) to assess the prediction accuracy. The RMSE is a statistical metric that measures the standard deviation of residuals. Residuals identify the distance between the observations and the predictions, i.e., the prediction errors. Computing the standard deviation, RMSE provides insights about the concentration of data points around the regression line.

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

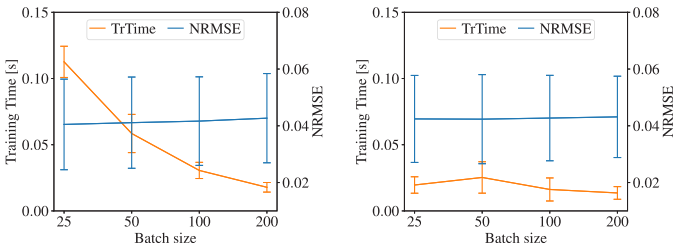
where N is the number of observations, y_i is the ith observation, $\hat{y}_i$ is the ith prediction. Since we compare the performance across different test sets, we normalize the RMSE w.r.t. the range of values of the dataset used.

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}} \tag{9}$$

5.4. SOTA comparison

Fig. 2 shows the performance comparison of CLAAP w.r.t to other state of the art solutions. We test two versions of our solution: *Offline-CLAAP* and *Online-CLAAP*. The former is not equipped with the concept drift detection module described in Section 5.5, the latter is re-trained "on-the-fly" over new data if concept drift is detected, as described in Algorithm 1.

We perform 5 tests with 5 different time series constructed from [44]. ARIMA and Prophet are fit on these sequences and predict the last 100 values. On the contrary, LSTM, Offline-CLAAP and Online-CLAAP are trained on the previously mentioned training dataset, and we test their generalization capability. EWMA is applied to the entire sequence with a smoothing factor $\alpha = 0.5$. Both Prophet and ARIMA typically guess the trend of the future data but are not accurate with their estimations. At the same time, the pre-trained models learned from a diversified training set and thus perform better. EWMA shows good enough performance despite CLAAP being better. Note that Prophet and ARIMA can be implemented with a rolling window such that they are fit on the last samples to predict the next one. This online version is more accurate but also very slow, as demonstrated in Fig. 2(b), which reports the infer time required by the models to predict the next value. Due to the significant amount of time spent on the online fitting, which delays the prediction, these approaches are impractical for our scenario. On the contrary, LSTM, CLAAP, and EWMA are fast and accurate, employing less than 300 µs for each prediction.

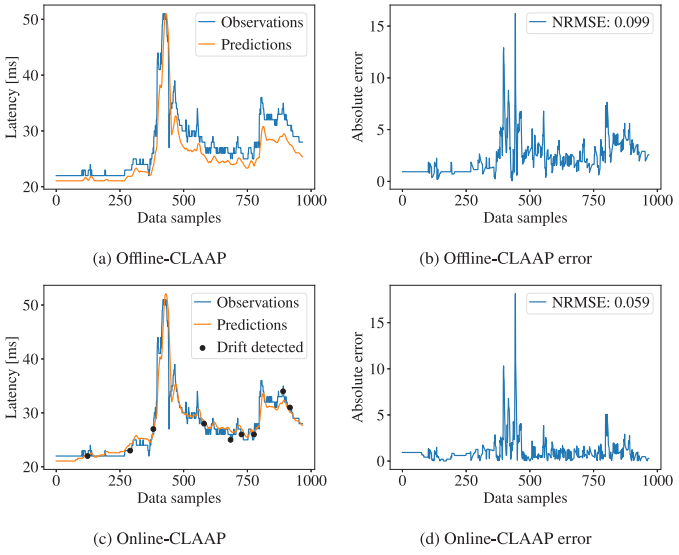


(a) Periodic updates (b) Online-CLAAP

Fig. 3. Comparison between periodic training and concept drift based re-training. Online-CLAAP exhibits comparable error employing much less time.

Table 2
Offline-CLAAPvs. Online-CLAAP. When the model detects concept drift, it re-trains over a batch of data decreasing the error.

Run #	NRMSE		Re-training overhead [ms]
	Offline-CLAAP	Online-CLAAP	
1	0.06103	0.05965	2
2	0.05594	0.03463	33
3	0.06702	0.05520	36
4	0.09866	0.05883	16
5	0.10935	0.06814	18
6	0.07069	0.04799	14
7	0.08287	0.08029	10
8	0.05611	0.04687	33
9	0.08036	0.07428	5
10	0.05619	0.05747	4
Avg.	0.074	0.058	17
Conf. Int.	0.013	0.01	9



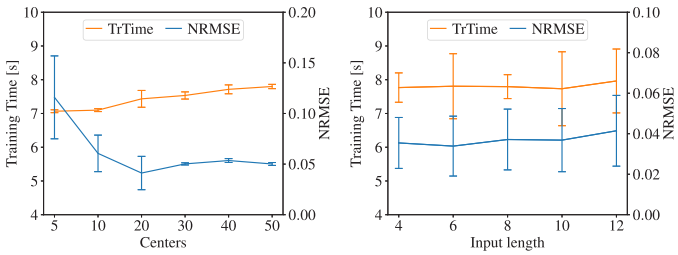
(a) Offline-CLAAP (b) Offline-CLAAP error

(c) Online-CLAAP (d) Online-CLAAP error

Fig. 4. Offline-CLAAP vs. Online-CLAAP. When concept drift is detected (black dots), re-training is performed and the prediction error is reduced.

5.5. Drift correction

In this set of experiments, we further study the impact of concept drift correction on the performance of our forecasting model. We first compare two approaches: periodically re-training the model every batch of data to adapt to the current time series characteristics, and our proposal Online-CLAAP, equipped with a Bayesian online approach to detect concept drift and re-train only when necessary, reducing the time spent on learning. Since the online re-training impacts the inferring time, we study both solutions' prediction error and training time, shown in Fig. 3. We expect to obtain a more accurate predictor



(a) Centers (b) Input length

Fig. 5. Sensitivity analysis of centers and input length. A growing number of centers lowers the error to a certain extent, but it takes more time to train. A higher input length requires more time to train, but the optimal value in terms of error depends on the data.

with frequent and regular re-training, which, however, requires more time for training. This first approach performs slightly better than our solution; nevertheless, Online-CLAAP displays a very similar error but takes considerably less time. This result demonstrates that our approach can detect concept drift and let us re-train the model on batches of data that significantly impact the performance while saving time.

In the next experiment, we simulate a realistic scenario with multiple clients: each concurrent client contacts the server to communicate the observed current delay and receives a prediction for the next latency value. The server collects and stores the received measurements in separated data structures, keeping a rolling buffer of size 6 as input for the model and a rolling buffer of size 50 for concept drift detection. During the inferring phase, once the concept drift buffer is full, the algorithm analyzes the distribution of this batch of data w.r.t. to previous data, warm-started with the training data. If at least one changepoint is detected, the model is re-trained "on-the-fly" over this last batch of data, and the buffer is reset. We sum the time spent in re-training due to the changepoint detection and report it as *re-training overhead*.

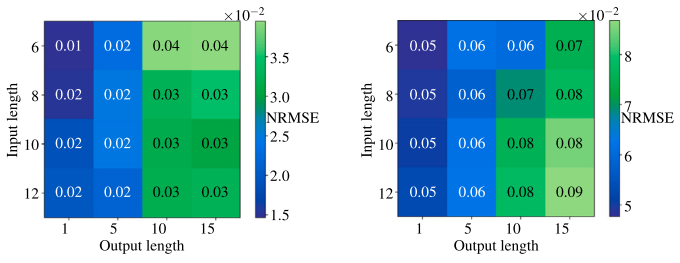
In Table 2 we compare the offline and online solutions deploying 10 clients with different datasets, and we report the average error and the overhead of Online-CLAAP, along with their confidence intervals. As it can be observed, this approach decreases the prediction error by 21% on average. The overhead introduced by this approach is 17 ms on average.

Then, for a specific run, i.e., the run 4, we also show the prediction error of Offline-CLAAP and Online-CLAAP in Fig. 4. This figure shows the behavior of CLAAP in the presence of frequent small variations and also sudden spikes. In the presence of small jitters, the error is kept lower than 5 ms. In the case of sudden variations, it is unable to predict the spike (as is the case with any other predictor), but CLAAP quickly adapts to the new data and reduces the error in future correlated predictions. It is clear that re-training "on-the-fly" is needed to further adapt to the new data and reduce the error. Online-CLAAP detects concept drift (the black dots) over batches of 50 data points. It takes 13 ms on average per window to detect changes and 2 ms on average to train online over the last collected batch.

5.6. Sensitivity analysis

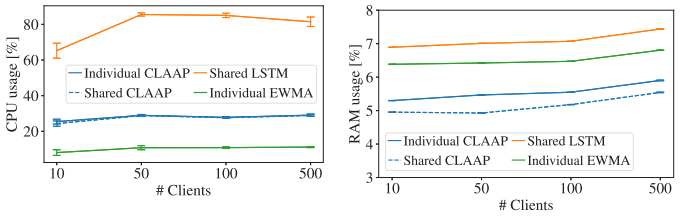
In this section, we first perform a sensitivity analysis over the number of hidden neurons k, i.e., centers, and n, the input size. Later, we give some insights into the Hazard function values and finally, we report the CPU and RAM usage for different solutions when scaling up the number of connected clients.

We now study the impact of centers and input size on the prediction error and training time, which is performed for the same number of epochs all over the experiments. Fig. 5(a) shows that increasing centers typically leads to higher training time as the neural network grows.



(a) Sampling every 1 ms (b) Sampling every 16 ms

Fig. 6. NRMSE varying input length and output length with different sampling intervals. The accuracy lowers as the prediction becomes far in time. A longer input history can capture intricate patterns or overshoot the trend in the sequence.



(a) CPU usage (b) RAM usage

Fig. 7. CPU and RAM usage comparison. The impact on resource usage when using one model per client is limited w.r.t. the number of clients.

Nonetheless, higher values lower the error; specifically, we select $k = 20$ for further experiments.

Figs. 5(b) and 6 depict the impact of input length and forward prediction on performance. The first experiment is conducted with 5 sequences, while the heatmaps are produced based on a singular test sequence. Predicting many steps in the future becomes more complex and less accurate, while the input length has different impacts for different datasets. When sampling every 1 ms, a more extended input history can help improve accuracy, while when sampling every 16 ms, considering many past values seems to degrade the performance. This depends on the fact that similarity between values shades over time due to the dynamic environment. However, the optimal input length heavily depends on the underlying trends in the data, but this range of values does not have an excessive impact on the training time and the prediction error; we adopt an input length (n) of 6 for our evaluation settings in which we predict one step in the future ($m = 1$).

Regarding the Hazard function H, a high value indicates that change points are frequent and can lead to excessive re-training. A very low value, on the contrary, could miss the occurrence of change points. We conducted a sensitivity analysis for this parameter, testing values ranging from 1/10 to 1/500. Since we did not observe any relevant patterns in the results, we decided not to include the complete sensitivity analysis graph in the article. Instead, we selected a value that represents a reasonable balance between frequent and sparse re-training with robust performance over our data $H(r_{t-1}) = 1/250$.

Finally, we report the CPU and RAM usage when running different solutions, as shown in Fig. 7, with a varying number of clients served in a realistic simulation. We compare (i) a predictor that uses a shared LSTM model for all clients, (ii) an equivalent version that adopts a shared CLAAP, (iii) our individual CLAAP solution that maintains a separate model for each client to detect concept drift and adapt to it, and (iv) an EWMA model per client. The graphs indicate that shared CLAAP is more efficient in terms of resource usage than individual CLAAP. However, it cannot adapt to the data of each client, whereas our individual CLAAP demonstrates greater accuracy, as shown in previous experiments, with only a minor increase in resource consumption. The complex operations performed by LSTM require significant CPU resources and more RAM compared to CLAAP. It is important to note that our concept drift adaptation module can also be applied to LSTM, but this may further increase the already high resource utilization. In addition, we can observe that EWMA demands less CPU usage, but it requires more RAM to maintain and update the weights associated with historical values continuously. These results, along with the accuracy comparison of Fig. 2, motivate our design choice to leverage a lightweight yet trainable neural network that learns from real traffic data.

6. Game evaluation scenario

Building on the results presented in the previous Sections, we investigate the application of CLAAP in a real-world cloud gaming scenario. In particular, we integrate CLAAP into the multiplayer car racing game described in [20]. This game is based on a server-authoritative architecture, where the server holds the certified version of the game state (GS), ensuring consistency and synchronization between all connected clients. Each client controls a car (Fig. 8, left) that competes against other players on a track consisting of elementary tiles (Fig. 8, right).

At each time frame t, all clients send to the server their client data (CD), which contain the 3D position, speed, and orientation of the vehicle they are controlling. The server then processes these data to update the GS and sends it back to the clients. This guarantees that the server has complete control over the game's progress, maintains a consistent state that is shared with all clients, and also ensures fairness by potentially preventing cheating. However, this architecture also makes the system very sensitive to network conditions, as any delay in client–server communication can lead to a desynchronization of the GS or a degradation of the player experience. To solve this issue, the server architecture described in [20] includes a GS prediction module (SPG), i.e., a dedicated software module that uses a neural network to predict the GS on the server side compensating for missing or delayed client data due to packet loss or latency.

In short, SPG operates as an external software module that interacts directly with the server's game logic (Fig. 9). Each time the game logic is updated (at time t), the server processes the data received from all connected clients (CD_{t-1}), to calculate the next game state, GS_t. At the same time, the current GS_{t-1} is sent to the SPG module, which predicts the next GS, denoted as GS_t^p, based on the CD received in the previous frames and sends it back to the server. The predicted GS is then sent back to the server, which integrates the calculated (GS_t) and predicted (GS_t^p) data according to the timely received and missing CDs. This updated GS is then sent to all clients to ensure a seamless gaming experience for all clients.

While SPG is effective in mitigating network issues, it also has limitations, especially in terms of efficiency and scalability [20]. SPG operates continuously and makes predictions for all clients regardless of their network conditions, which leads to unnecessary computational overhead for clients with stable connections. In addition, as the number of clients increases, so does SPG's workload, which can cause it to exceed the strict time constraints of real-time gaming applications, such as the 16 ms refresh cycle required for fast-paced games.

CLAAP can help address these issues by acting as a proactive latency predictor and working with SPG to optimize operations. By continuously monitoring network conditions, CLAAP can predict potential latency issues for each client in real time. This allows SPG to take targeted action for only those clients that are likely to experience network issues, significantly reducing its computational load. In this way, CLAAP increases SPG's efficiency and scalability. By filtering out unnecessary computation and focusing SPG resources on critical cases, CLAAP can help ensure that the system remains responsive and synchronized even when the number of connected clients increases, ensuring a smooth gaming experience under variable network conditions.

As for the overall architecture, both SPG and CLAAP are designed as external modules that interact with the server (Fig. 9). CLAAP

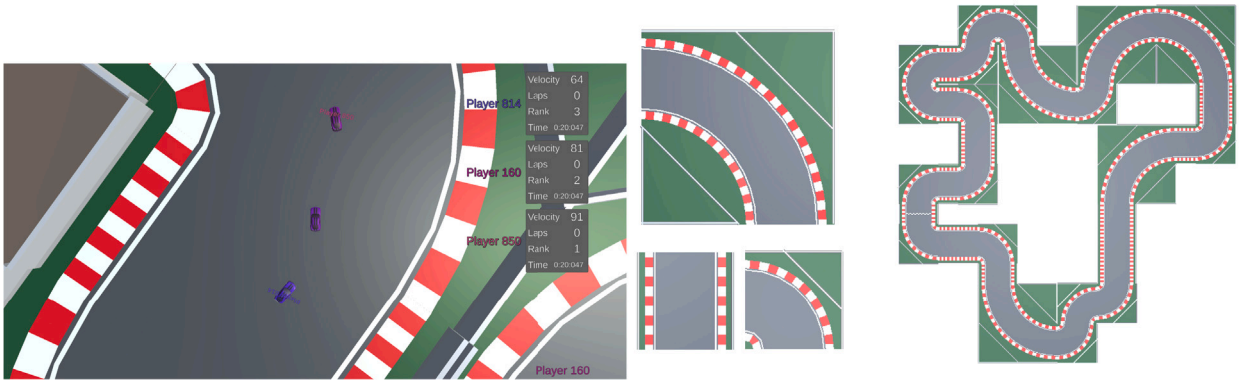


Fig. 8. Left: A screenshot of the multiplayer car racing game interface, showing three AI-controlled vehicles and their status panels (laps, velocity, rank, and time). Right: The racetrack, built by assembling modular corner and straight tiles to form different layouts.

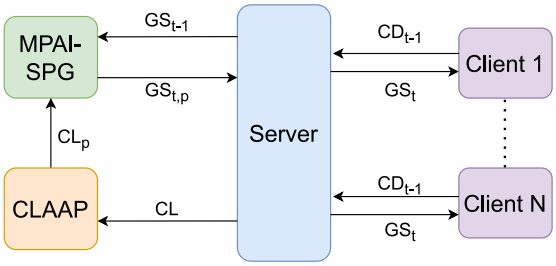


Fig. 9. SPG and CLAAP integration architecture.

communicates with both the server and SPG. At each time step t, the server sends a list of client latency values (CL) to CLAAP. These values, which are computed as the round-trip time (RTT) of dedicated latency messages (LM) exchanged between the server and each client, are processed by CLAAP to predict future latency values (CL^p). CLAAP uses these predictions to inform SPG about potential latency issues with specific clients, e.g., when a client's latency exceeds a predefined threshold, such as the 50 ms commonly used as a tolerance for fast-paced games. In this way, SPG can use its computing resources to calculate predictions for these clients only.

In terms of technical details, the server and the SPG module have been implemented using the Unity engine. Networking in the server is managed via the Mirror library, a robust and high-performance solution for real-time communication in multiplayer games. SPG utilizes Unity's Barracuda package for neural network inference, enabling efficient predictions on both the GPU and CPU and ensuring compatibility with Unity's runtime environment.

CLAAP was implemented as an independent Python process that runs concurrently with the game server. Communication between CLAAP and the server and between CLAAP and SPG is done via ZeroMQ, a lightweight messaging protocol chosen for its low latency. This separation of processes makes it possible to free the server from additional processing loads that could otherwise affect its real-time performance. In addition, CLAAP's independence facilitates scalability, as it can be deployed on separate machines or distributed systems to adapt to larger player bases without impacting the server's performance.

6.1. Settings

We conducted a series of experiments on the multiplayer racing game to investigate how CLAAP enhances performance relative to other approaches and to confirm specific hypotheses about scalability under different network conditions. We tested three configurations:

- **SPG**: The server continuously runs SPG for every client on each update cycle without any latency prediction. Our hypothesis is that this configuration does not scale well as the number of clients grows because SPG runs for all clients, irrespective of their actual network status.
- **SPG-CLAAP**: CLAAP forecasts each client's latency, and SPG only executes if a client's latency is predicted to exceed a threshold. We hypothesize that this selective activation lowers computational overhead compared to SPG, leading to better scalability.
- **SPG-LSTM**: Similar to SPG-CLAAP, but with an LSTM model for latency prediction. We hypothesize that although LSTM can match CLAAP's accuracy, it may come with higher computational overhead, potentially impacting performance under heavier loads.
- **SPG-EWMA**: In this configuration, an EWMA-based predictor replaces CLAAP. We expect a simpler predictor to achieve performance comparable to CLAAP from a system overhead perspective, although its forecasts are likely to be less accurate.

We ran four-minute test sessions with $1, 5, 10, 20$, or 80 concurrent clients, each controlled by an AI driving a car on the same track. To introduce realistic network latency for these configurations, we injected measured time series delay traces at the client side (application layer). These traces, extracted from an online database, reflect authentic network fluctuations (i.e., bursts, spikes, stable intervals). We defined three severity levels (low, medium, and high) based on whether $5\%, 20\%$, or 50% of the clients, respectively, were randomly assigned these delay traces at the beginning of each session. In contrast, in the SPG-only configuration, SPG runs continuously for all clients regardless of actual delay. Consequently, injecting latencies there would not affect performance metrics, so we did not apply latency injection in that scenario. Fig. 10 summarizes the distribution of clients facing latency spikes in the different configurations of the experiments (i.e., client number and severity levels).

All experiments were conducted on a high-performance workstation (Intel i7-14700KF CPU at 3.40 GHz, 64 GB of RAM, NVIDIA GeForce RTX 4080 SUPER GPU), with the clients distributed across six separate machines on an isolated local network. During each run, we collected frame rate (FPS), SPG prediction times, CPU usage, and RAM usage to compare system behavior under the different latency severities and client loads, thereby testing our hypotheses about each configuration's scalability. We further compared the prediction accuracy of the different latency predictors by measuring a *Misdetections* metric, counting false positives and false negatives. The error bars plotted in the graph represent the 95% confidence intervals.

6.2. Results

To show how prediction errors impact the SPG module, we measure false positive and false negative occurrences for each latency predictor

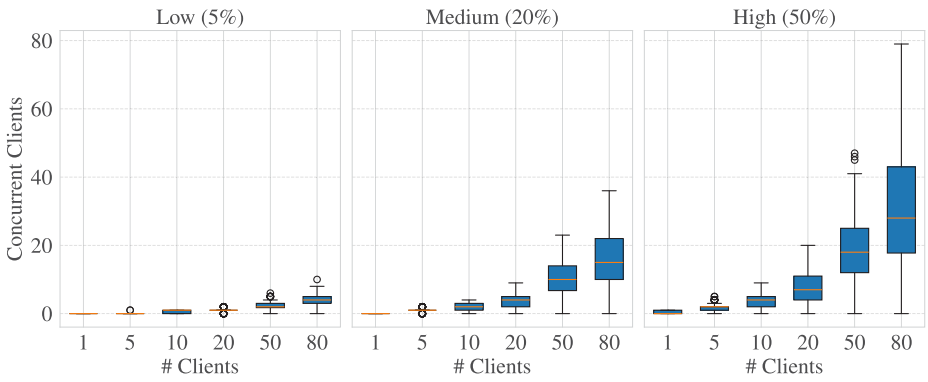
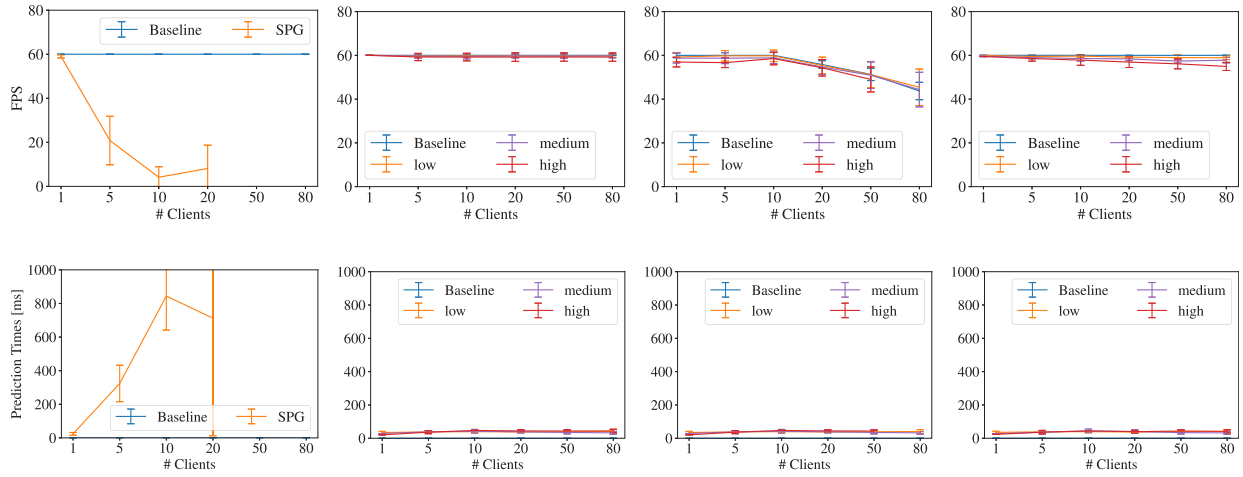


Fig. 10. Box plots of how many clients experienced a latency spike (above threshold) within the same one-second interval, across the three severity levels (low, medium, high) defined for the experiments. The *x*-axis indicates the total client scenarios simulated in the experiments, and each box summarizes the distribution of concurrency values over the four-minute game sessions.



(a) SPG (b) SPG-CLAAP (c) SPG-LSTM (d) SPG-EWMA

Fig. 11. In-game performance metrics for the four experimental configurations: FPS (top) and SPG prediction times (bottom). The left subplot (SPG) has no severity levels because SPG runs continuously, whereas the remaining subplots (CLAAP, LSTM, EWMA) each include baseline (no SPG) and three severity levels (low, medium, high).

Table 3
Sum of Miss and Overflow events. A more accurate predictor translates in fewer misdetections. Online-CLAAP is the most effective method.

# Clients	Misdetections (%)			
	Online-CLAAP	Offline-CLAAP	LSTM	EWMA
1	11.7	11.7	13.3	18.3
10	9.2	10.8	12.4	12.0
20	10.5	10.6	11.2	11.5
50	10.3	10.6	11.1	11.6
80	11.0	11.1	11.5	12.1

and report the average results in Table 3. As congestion is detected when client latency is predicted to exceed an acceptable threshold, *false negatives* occur when a predictor fails to recognize that the threshold has been exceeded, resulting in lost opportunities for compensation ("miss event"). Conversely, *false positives* happen when we incorrectly predict that the threshold has been exceeded, leading to unnecessary activation of the SPG and wasted resources ("Overflow event"). These experiments demonstrate that a more accurate predictor like Online-CLAAP leads to fewer misdetections, thus the game can more properly react to latency spikes without unnecessary operations.

We now focus on in-game performance metrics and report results in Fig. 11. We compare each configuration against a *baseline* condition to better isolate the overhead of the tested approaches. In particular, for SPG, the baseline is simply the multiplayer game without the SPG compensation (i.e., game state predictions). For SPG-CLAAP, SPG-LSTM, and SPG-EWMA, the baseline represents the version of the multiplayer game equipped with the respective latency predictor but whose functions are never invoked by SPG. Since there are no on-demand computations in these baselines, the network latency has no impact on the measured performance.

From the figure, we can observe significant performance issues in the SPG configuration as the number of clients increases (Fig. 11, (a)-top). The system only maintained a playable FPS of approximately 60 when predicting the GS for a single client. However, with as few as 5 clients, FPS dropped below 20, making the game unplayable. As the number of clients increased to 10 and 20, FPS dropped even further. This deterioration in performance is directly related to the exponential increase in SPG prediction times (Fig. 11, (a)-bottom). While the average prediction times were manageable under the 1 client condition, they became unsustainable with additional clients, reaching almost 1 second. Such long prediction times overwhelmed the server's game loop, which was busy with lengthy calculations, resulting in a dramatic FPS drop. In addition, this blocking effect affected the server's ability to process network messages in real time, causing clients to lose connections. As a result, the system became unstable, and running experiments with 50 or 80 clients was not feasible under these conditions.

In contrast, the SPG-CLAAP configuration shows exceptional stability and scalability for in-game performance metrics. Regardless of the number of clients or the severity of network issues (low, medium or

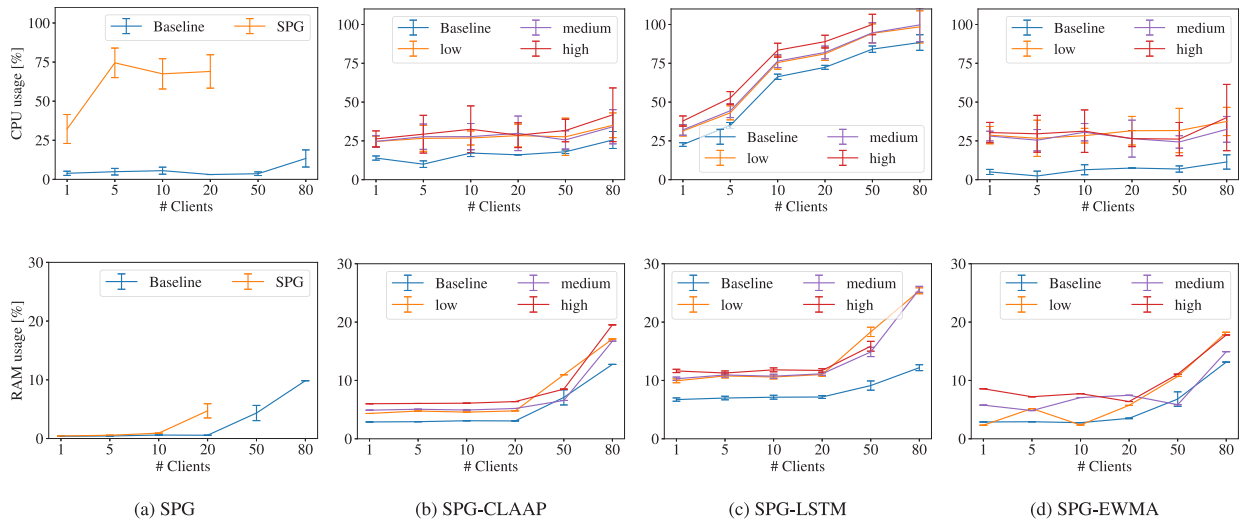


Fig. 12. In-game performance metrics for the four experimental configurations: CPU (top) and RAM (bottom). The left subplot (SPG) has no severity levels because SPG runs continuously, whereas the remaining subplots (CLAAP, LSTM, EWMA) each include baseline (no SPG) and three severity levels (low, medium, high).

high), the system consistently maintained a stable FPS of approximately 60 (Fig. 11, (b)-top). This is due to CLAAP's ability to selectively activate SPG only for clients experiencing latency, keeping average prediction times low (Fig. 11, (b)-bottom). As a result, the server's game loop did not stall, allowing for smooth and responsive gameplay even with 80 clients. On the other hand, SPG-LSTM configuration maintained near 60 FPS under moderate loads, but in high severity with 80 clients, the additional computational cost of running the LSTM blocked the server loop to the point of causing disconnections, similar to SPG configuration failing beyond 20 clients (Fig. 11, (c)-top). The SPG-EWMA version, by contrast, is similar to SPG-CLAAP and adds negligible overhead due to the simplicity of the EWMA predictor (Fig. 11, (d)-top). Its in-game performance usage remains comparable to CLAAP even at 80 clients under high severity, albeit with more misdetections that could undermine user experience if latency spikes are missed more often.

A similar trend can be observed in the system performance metrics: CPU and RAM usage. In the SPG configuration, CPU usage sharply increased as the number of clients grew, reflecting the intensive computation required to predict each client's GS (Fig. 12, (a)-top). With more than five clients, the variability in CPU usage also became pronounced, indicating instability in the server's ability to manage the workload. RAM usage increased steadily but remained within acceptable limits until the system crashed at higher client counts.

However, in the SPG-CLAAP configuration, CPU and RAM usage grows in a controlled and predictable manner, even with high client loads and varying network conditions (Fig. 12, (b)). CLAAP's proactive approach to latency prediction reduced unnecessary computations, ensuring that SPG was only activated when needed. This stabilized resource usage and minimized variability, as evidenced by the smaller error bars across all measurements. The combination of efficient prediction management and resource optimization enabled the system to process up to 80 clients without compromising performance or stability, highlighting the robustness of the CLAAP-integrated approach. SPG-EWMA similarly remains low in overhead, confirming that both CLAAP and EWMA leverage selective activation effectively (Fig. 12, (d)). SPG-LSTM scales well up to moderate conditions but falters at the highest loads, underscoring how its inference cost can undermine scalability (Fig. 12, (c)).

Overall, these results confirm the ability of CLAAP to improve the efficiency and scalability of real-time multiplayer systems, especially in scenarios with high client numbers and variable network conditions. While a simpler predictor (EWMA) achieves similar overhead, its lower accuracy may increase false negatives and compromise the user's experience. Meanwhile, a more complex predictor (LSTM) offers good accuracy yet risks severe overhead under extreme conditions. CLAAP strikes a balance, providing strong predictive performance alongside scalability and minimal system impact.

7. Conclusions

This study introduced CLAAP, a novel solution for cloud gaming applications that aims to manage high latency events by predicting future occurrences in real-time latency. By leveraging a Radial Basis Function Neural Network, CLAAP provides accurate latency prediction by capturing non-linear patterns and processing data efficiently in a stream fashion. Additionally, the integration of Bayesian Online Changepoint Detection enables CLAAP to adapt to changes in data distribution by detecting concept drift and re-training the model on the fly, ensuring sustained predictive accuracy in dynamic network conditions. We then integrated this mechanism into a multiplayer racing game with a server-authoritative architecture that includes a game state prediction module (SPG) designed to compensate for delayed client data. When the latency is predicted to be excessive, the solution triggers the prediction of the game state for the specific user.

Trace-driven experiments were conducted to evaluate CLAAP's effectiveness, comparing its performance against state-of-the-art latency prediction methods. These tests assessed prediction accuracy, inference time, and error rates across various simulated network conditions. CLAAP demonstrated superior accuracy and efficiency, excelling in forecasting latency. Notably, integrating concept drift correction enabled CLAAP to adapt dynamically to changes in data distribution, achieving an average error reduction of 21% compared to non-adaptive approaches, with minimal computational overhead. Moreover, CLAAP enhances the SPG by enabling selective activation, limiting predictions to only clients experiencing latency issues. This optimization reduced computational overhead and ensured stable in-game performance, maintaining consistent frame rates even under high client loads.

These findings demonstrate the potential of CLAAP to address critical latency challenges in cloud gaming, offering precision and adaptability. Future work will focus on testing the application with varying game genres and QoE metrics to further assess the validity of our approach and to assess how prediction errors (e.g., missing a real spike) impact playability and user-perceived immersion. Beyond cloud gaming, CLAAP's framework can be extended to other latency-sensitive applications such as video streaming, virtual reality, and industrial IoT systems, where latency forecasts could further optimize performance.

CRediT authorship contribution statement

Doriana Monaco: Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Alessio Sacco:** Writing – review & editing, Validation, Methodology, Conceptualization. **Daniele Spina:** Writing – original draft, Visualization, Software. **Francesco Strada:** Writing – review & editing, Validation, Methodology, Conceptualization. **Andrea Bottino:** Writing – review & editing. **Tania Cerquitelli:** Writing – review & editing. **Guido Marchetto:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the European Union - Next Generation EU under the Italian National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.3, CUP E13C2200187 0001, partnership on “Telecommunications of the Future” (PE00000001 - program “RESTART”).

Data availability

Data will be made available on request.

References

[1] M. Lyu, Y. Wang, V. Sivaraman, Do cloud games adapt to client settings and network conditions? in: 2024 IFIP Networking Conference, IEEE, 2024, pp. 482–488.
[2] M. Lyu, S.C. Madanapalli, A. Vishwanath, V. Sivaraman, Network anatomy and real-time measurement of Nvidia GeForce NOW cloud gaming, in: International Conference on Passive and Active Network Measurement, Springer, 2024, pp. 61–91.
[3] R. Eg, K. Raaen, M. Claypool, Playing with delay: With poor timing comes poor performance, and experience follows suit, in: 2018 Tenth International Conference on Quality of Multimedia Experience, QoMEX, IEEE, 2018, pp. 1–6.
[4] S. Vlahovic, M. Suznjevic, L. Skorin-Kapov, The impact of network latency on gaming QoE for an FPS VR game, in: 2019 Eleventh International Conference on Quality of Multimedia Experience, QoMEX, IEEE, 2019, pp. 1–3.
[5] L. Yang, Q. Jiaxing, W. Hongyi, L. Zhenhua, Q. Feng, Y. Jing, L. Hao, X. Bo, Q. Xiaokang, T. Xu, Dissecting and streamlining the interactive loop of mobile cloud gaming, in: The 22nd USENIX Symposium on Networked Systems Design and Implementation, NSDI, 2025.
[6] T. Ishioka, T. Fukui, R. Tsugami, T. Fujiwara, S. Narikawa, T. Fujihashi, S. Saruwatari, T. Watanabe, Improving quality of experience in cloud gaming using speculative execution, in: 2023 Fourteenth International Conference on Mobile Computing and Ubiquitous Network, ICMU, IEEE, 2023, pp. 1–4.
[7] J. Wu, Y. Guan, Q. Mao, Y. Cui, Z. Guo, X. Zhang, ZGaming: Zero-latency 3D cloud gaming by image prediction, in: Proceedings of the ACM SIGCOMM 2023 Conference, 2023, pp. 710–723.
[8] Z. Wang, G. Sun, D. Xu, Online optimization algorithms for edge computing-based cloud gaming user experience, in: 2023 3rd International Conference on Frontiers of Electronics, Information and Computation Technologies, ICFEICT, IEEE, 2023, pp. 70–75.
[9] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, C.-H. Hsu, QoE-aware virtual machine placement for cloud games, in: 2013 12th Annual Workshop on Network and Systems Support for Games, NetGames, IEEE, 2013, pp. 1–2.
[10] Y. Li, X. Tang, W. Cai, Play request dispatching for efficient virtual machine usage in cloud gaming, IEEE Trans. Circuits Syst. Video Technol. 25 (12) (2015) 2052–2063.
[11] Y. Lin, H. Shen, Cloud fog: Towards high quality of experience in cloud gaming, in: 2015 44th International Conference on Parallel Processing, IEEE, 2015, pp. 500–509.
[12] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, D.O. Wu, Improving cloud gaming experience through mobile edge computing, IEEE Wirel. Commun. 26 (4) (2019) 178–183.
[13] S.S. Sabet, S. Schmidt, S. Zadtootaghaj, B. Naderi, C. Griwodz, S. Möller, A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience, in: Proceedings of the 11th ACM Multimedia Systems Conference, 2020, pp. 15–25.
[14] R. Salay, M. Claypool, A comparison of automatic versus manual world alteration for network game latency compensation, in: Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play, 2020, pp. 355–359.
[15] B. Anand, P. Wenren, CloudHide: Towards latency hiding techniques for thin-client cloud gaming, in: Proceedings of the on Thematic Workshops of ACM Multimedia 2017, 2017, pp. 144–152.
[16] K. Lee, D. Chu, E. Cuervo, J. Kopf, Y. Degtyarev, S. Grizan, A. Wolman, J. Flinn, Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming, in: Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, 2015, pp. 151–165.
[17] F. Palumbo, G. Aceto, A. Botta, D. Ciuonzo, V. Persico, A. Pescapé, Characterizing cloud-to-user latency as perceived by AWS and azure users spread over the globe, in: 2019 IEEE Global Communications Conference, GLOBECOM, IEEE, 2019, pp. 1–6.
[18] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: A review, IEEE Trans. Knowl. Data Eng. 31 (12) (2018) 2346–2363.
[19] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) 1–37.
[20] D. Spina, A. Bottino, D. Cavagnino, L. Chiariglione, M. Lucenteforte, M. Mazzaglia, F. Strada, AI server-side prediction for latency mitigation and cheating detection: The MPAI-SPG approach, in: 2024 IEEE Gaming, Entertainment, and Media Conference, GEM, IEEE, 2024, pp. 1–6.
[21] R. Hyndman, A.B. Koehler, J.K. Ord, R.D. Snyder, Forecasting with Exponential Smoothing: the State Space Approach, Springer Science & Business Media, 2008.
[22] S. Siami-Namini, N. Tavakoli, A.S. Namin, A comparison of ARIMA and LSTM in forecasting time series, in: 2018 17th IEEE International Conference on Machine Learning and Applications, ICMLA, IEEE, 2018, pp. 1394–1401.
[23] A.K. Bera, M.L. Higgins, ARCH models: properties, estimation and testing, J. Econ. Surv. 7 (4) (1993) 305–366.
[24] Z. Liu, Z. Zhu, J. Gao, C. Xu, Forecast methods for time series data: a survey, Ieee Access 9 (2021) 91896–91912.
[25] G. White, A. Palade, S. Clarke, Forecasting QoS attributes using LSTM networks, in: 2018 International Joint Conference on Neural Networks, IJCNN, IEEE, 2018, pp. 1–8.
[26] Y. Fu, D. Guo, Q. Li, L. Liu, S. Qu, W. Xiang, Digital twin based network latency prediction in vehicular networks, Electronics 11 (14) (2022) 2217.
[27] D. Scano, F. Paolucci, K. Kondepu, A. Sgambelluri, L. Valcarenghi, F. Cugini, Extending P4 in-band telemetry to user equipment for latency-and localization-aware autonomous networking with AI forecasting, J. Opt. Commun. Netw. 13 (9) (2021) D103–D114.
[28] A.S. Khatouni, F. Soro, D. Giordano, A machine learning application for latency prediction in operational 4g networks, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM, IEEE, 2019, pp. 71–74.
[29] A.H. Ahmed, S. Hicks, M.A. Riegler, A. Elmokashfi, Predicting high delays in mobile broadband networks, IEEE Access 9 (2021) 168999–169013.
[30] Q. Yang, X. Peng, L. Chen, L. Liu, J. Zhang, H. Xu, B. Li, G. Zhang, Deepqueuenet: Towards scalable and generalized network performance estimation with packet-level visibility, in: Proceedings of the ACM SIGCOMM 2022 Conference, 2022, pp. 441–457.
[31] A. Sacco, F. Esposito, G. Marchetto, Restoring application traffic of latency-sensitive networked systems using adversarial autoencoders, IEEE Trans. Netw. Serv. Manag. 19 (3) (2022) 2521–2535.
[32] D.S.H. Tam, Y. Liu, H. Xu, S. Xie, W.C. Lau, Pert-gnn: Latency prediction for microservice-based cloud-native applications via graph neural networks, in: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2023, pp. 2155–2165.
[33] G.F. Ciocarlie, U. Lindqvist, S. Nováczki, H. Sanneck, Detecting anomalies in cellular networks using an ensemble method, in: Proceedings of the 9th International Conference on Network and Service Management, CNSM 2013, IEEE, 2013, pp. 171–174.
[34] S. Greco, T. Cerquitelli, Drift lens: Real-time unsupervised concept drift detection by evaluating per-label embedding distributions, in: 2021 International Conference on Data Mining Workshops, ICDMW, IEEE, 2021, pp. 341–349.
[35] R.K. Jagait, M.N. Fekri, K. Grolinger, S. Mir, Load forecasting under concept drift: Online ensemble learning with recurrent neural network and ARIMA, IEEE Access 9 (2021) 98992–99008.
[36] S. Liu, F. Bronzino, P. Schmitt, A.N. Bhagoji, N. Feamster, H.G. Crespo, T. Coyle, B. Ward, LEAF: Navigating concept drift in cellular networks, Proc. the ACM Netw. 1 (CoNEXT2) (2023) 1–24.
[37] C. Ding, J. Zhao, S. Sun, Concept drift adaptation for time series anomaly detection via transformer, Neural Process. Lett. 55 (3) (2023) 2081–2101.
[38] M.J. Er, S. Wu, J. Lu, H.L. Toh, Face recognition with radial basis function (RBF) neural networks, IEEE Trans. Neural Netw. 13 (3) (2002) 697–710.
[39] Z. Yang, M. Mourshed, K. Liu, X. Xu, S. Feng, A novel competitive swarm optimized RBF neural network model for short-term solar power generation forecasting, Neurocomputing 397 (2020) 415–421.

[40] A. Addeh, M. Iri, Brain tumor type classification using deep features of MRI images and optimized RBFNN, ENG Trans. 2 (2021) 1–7.
[41] M.J. Samonte, Y. Lan, Q. Cao, H. Zhu, A zero-shot super-resolution image reconstruction technique based on radial basis function neural networks, in: Proceedings of the 2023 11th International Conference on Computer and Communications Management, 2023, pp. 58–64.
[42] A. Ismayilova, M. Ismayilov, On the universal approximation property of radial basis function neural networks, Ann. Math. Artif. Intell. 92 (3) (2024) 691–701.
[43] Y. Liao, S.-C. Fang, H.L. Nuttle, Relaxed conditions for radial-basis function networks to be universal approximators, Neural Netw. 16 (7) (2003) 1019–1028.
[44] P. Graff, X. Marchal, T. Cholez, S. Tuffin, B. Mathieu, O. Festor, An analysis of cloud gaming platforms behavior under different network constraints, in: 17th International Conference on Network and Service Management, CNSM, IEEE, 2021, pp. 551–557.
[45] S. Aminikhanghahi, D.J. Cook, A survey of methods for time series change point detection, Knowl. Inf. Syst. 51 (2) (2017) 339–367.
[46] L. Yuan, H. Li, B. Xia, C. Gao, M. Liu, W. Yuan, X. You, Recent advances in concept drift adaptation methods for deep learning, in: IJCAI, 2022, pp. 5654–5661.
[47] G.I. Webb, R. Hyde, H. Cao, H.L. Nguyen, F. Petitjean, Characterizing concept drift, Data Min. Knowl. Discov. 30 (4) (2016) 964–994.
[48] O. Hilyard, B. Cui, M. Webster, A.B. Muralikrishna, A. Charapko, Cloudy forecast: How predictable is communication latency in the cloud?, 2023, arXiv preprint arXiv:2309.13169.
[49] O. Tomanek, P. Mulinka, L. Kencl, Multidimensional cloud latency monitoring and evaluation, Comput. Netw. 107 (2016) 104–120.
[50] P. Mulinka, L. Kencl, Learning from cloud latency measurements, in: 2015 IEEE International Conference on Communication Workshop, ICCW, IEEE, 2015, pp. 1895–1901.
[51] R.P. Adams, D.J. MacKay, Bayesian online changepoint detection, 2007, arXiv preprint arXiv:0710.3742.
[52] S.J. Taylor, B. Letham, Forecasting at scale, Amer. Statist. 72 (1) (2018) 37–45.



Doriana Monaco received the M.Sc degree in Computer Engineering from Politecnico di Torino, in 2022, where she is currently pursuing a Ph.D degree. Her research interests cover computer networks monitoring and management; distributed learning applications for Software-Defined Networks (SDN); and cloud-computing solutions.



Alessio Sacco is an Assistant Professor at Politecnico di Torino. He received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Italy, in 2018 and 2022, respectively. His research interests include architecture and protocols for network management; implementation and design of cloud computing applications; and algorithms and protocols for service-based architecture, such as Software Defined Networks (SDN), used in conjunction with Machine Learning algorithms.



Daniele Spina received his M.Sc. in Computer Engineering from Politecnico di Torino in 2023, where he currently holds a research grant. His research focuses on multiplayer gaming and the application of Virtual and Augmented Reality technologies for cultural heritage dissemination.



Francesco Strada is currently an Assistant Professor at the Department of Control and Computer Engineering of Politecnico di Torino, where he conducts research under the Computer Graphics and Vision research group of the same university. His current research interests include Virtual and Augmented Reality for learning and training, Serious Games, Human–Computer Interaction, and the use of immersive technologies in collaborative scenarios.



Andrea Bottino is currently an Associate Professor at the Department of Control and Computer Engineering of the Politecnico di Torino, where he heads the Computer Graphics and Vision research group of the same university. His current research interests include Computer Vision, Machine Learning, Human–Computer Interaction, Serious Games, and Virtual and Augmented Reality. He is a member of the IEEE.



Tania Cerquitelli is a full professor at the Department of Control and Computer Engineering of the Politecnico di Torino, Italy. She is responsible for aggregate functions to the Deputy Rector for Society, Community, and Program Delivery, supporting activities for the Polytechnic University Community. Her research interests include techniques for explaining black-box models, algorithms to democratize data science, and algorithms to detect concept drift early. She is responsible for different research projects and has obtained research funding from the EU, the Piedmont Region, the Ministry of University and Research, and private companies.



Guido Marchetto received the Ph.D. degree in computer engineering from the Politecnico di Torino, in 2008, where he is currently Full Professor with the Department of Control and Computer Engineering. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures.