Quantum Machine Intelligence (2026) 8:35
https://doi.org/10.1007/s42484-026-00374-9

RESEARCH ARTICLE

QURIOSO: Quantum gate parameter predictIon through quantum-enhanced long-short term memory

Corrado Loglisci[1] · Vito N. Losavio[1] · Saverio Pascazio[2] · Donato Malerba[1]

Received: 9 August 2025 / Accepted: 19 February 2026

Abstract
In Variational Quantum Algorithms (VQAs), circuit parameters are typically re-optimized from scratch for each new dataset, an approach that becomes inefficient in continuous computation settings where data arrive incrementally and statistical properties evolve over time. This limitation is exacerbated on noisy intermediate-scale quantum (NISQ) devices, where optimization is costly and prone to barren plateaus. To overcome these challenges, we introduce QURIOSO (QUantum gate parameteR predictIOn through quantum-enhanced long-Short term memOry), a machine learning framework that replaces repeated optimization with parameter prediction. QURIOSO learns trajectories of PQC parameters from past data and forecasts parameters for new data using either classical or quantum-enhanced LSTM architectures. We redesign quantum LSTM cells by employing Y-axis controlled rotations for amplitude modulation and full entanglement generation, enabling richer transformations than conventional phase-based designs. The framework operates in two modes: direct parameter transfer to unseen data and prediction-driven initialization to accelerate subsequent optimization. Validation on real-world binary classification tasks with hybrid classical–quantum models shows that quantum-enhanced LSTMs frequently outperform classical counterparts, highlighting their ability to preserve non-linearity through quantum encoding and measurement, and demonstrating the effectiveness of QURIOSO in generalizing across parameter trajectories within continuous quantum learning scenarios.

Keywords Variational quantum algorithms · Parameterized quantum circuits · Parameter estimation · Sequence modeling

1 Introduction

In the rapidly advancing field of quantum technologies and quantum information processing, Noisy Intermediate-Scale Quantum (NISQ) devices have emerged as a key area of focus, demonstrating quantum supremacy in a limited number of applications (Madsen et al. 2022). Although NISQ devices are not yet capable of performing fully error-corrected quantum computations, they represent a significant milestone in quantum technology development. However, their limitations–such as short coherence times and high error rates–have led to the creation of specialized algorithms, including Variational Quantum Algorithms (VQAs), designed to be resilient against NISQ hardware imperfections.

The interplay between the physical constraints of NISQ devices and innovative design of VQAs is a crucial area of research in quantum computing. VQAs are hybrid quantum-classical algorithms that integrate parameterized quantum

Special issue on AIQ*QIA 2024 2nd International Workshop on AI for Quantum and Quantum for AI.

✉ Corrado Loglisci
corrado.loglisci@uniba.it

Vito N. Losavio
v.losavio5@phd.uniba.it

Saverio Pascazio
saverio.pascazio@ba.infn.it

Donato Malerba
donato.malerba@uniba.it

[1] Department of Computer Science, Universita' degli Studi di Bari 'Aldo Moro', Bari, Italy

[2] Department of Physics, Universita' degli Studi di Bari 'Aldo Moro', Bari, Italy

Published online: 18 March 2026

circuits (PQCs) with optimization techniques. PQCs consist of single- and multi-qubit gates characterized by tunable parameters, which are adjusted by classical or quantum optimizers, similarly to the construction of weights for neural networks. Optimization techniques typically follow a multi-step process that begins with initialized parameters and iteratively tunes them based on input data, minimizing (or maximizing) an objective function until refinement is achieved.

The effectiveness of VQAs depends heavily on how the parameters are initialized and optimized. Indeed, PQCs are susceptible to the barren plateau (McClean et al. 2018), which is the effect of exponentially vanishing gradients, with flattening of the optimization landscape. This makes the refinement of the circuit parameters extremely difficult and training process significantly hindered. The problem impacts also on the design of quantum circuits. The access to real quantum computers, which could guarantee the speeding-up of the VQA executions, is limited, often handled by large corporations which provide only an usage controlled by job queuing mechanisms, resulting in long wait times and high costs (Liang et al. 2024). Therefore, the development of VQAs very often resorts to simulated quantum machines, which lack the full computational capabilities of real quantum hardware. As a result, the optimization process typically dominates the computational cost of the execution of VQAs. Given these challenges, effective strategies for refining parameters and and driving to convergence become crucial. Starting the optimization from an informed initial point–rather than from a random or uniform distribution–can significantly improve VQA performance. Indeed, one could steer the process toward promising regions of the parameter space, while increasing the chances of convergence. To achieve that, computational approaches must be capable of making non-intuitive decisions, exploring complex high-dimensional spaces, and uncovering patterns or correlations that are not immediately obvious. They should also be robust to noise and errors while reducing the reliance on empirical trial-and-error methods (Falla et al. 2024). This is where machine learning (ML) algorithms can play a decisive role. ML models can systematically estimate parameters, efficiently search for optimal values, and significantly improve the execution of quantum algorithms.

Only recently this topic has gained significant attention, with most efforts focusing on the use of models to initialize quantum-classical optimizers (Meng and Zhou 2024; Liang et al. 2024). To cite someone, Sauvage et al. (2021) introduce a meta-learning strategy that trains neural networks on families of PQCs to enable effective parameter initialization for previously unseen, structurally diverse circuits. An agnostic encoding–decoding scheme supports generalization across variations in circuit depth and parameter count. While Moussa et al. (2022) adopt an unsupervised approach to parameter initialization, deriving starting values from clusters built on QAOA (Quantum Approximate Optimization Algorithm) execution-related data. Leveraging the concentration property of QAOA parameters, this method identifies suitable initializations by using centroids derived from various representations of QAOA problems.

However, these approaches operate within the classical paradigm of batch processing, where all data are available at once. This is the typical scenario where most research in quantum computing for accelerating computational solutions has been focused. In these cases, VQAs are executed on the entire dataset at once, and PQC parameters remain fixed once estimated. In contrast, many real-world applications involve data that are not accessible all at once but arrive incrementally over time. This necessitates continuous processing rather than batch-mode execution. Furthermore, the system or domain generating the data may evolve, meaning that the statistical properties of the data can change over time. As a result, data samples may exhibit time-varying dependencies or correlations with previously observed data samples (Homayouni et al. 2020; Parisi et al. 2019). This raises an important challenge: PQC parameters optimized on data available at one point in time may become suboptimal or even obsolete as new data–potentially with different characteristics–arrive. This issue has significant implications for various applications of VQAs. For instance, in real-world applications such as continuous biomedical monitoring or sensor-based rehabilitation, patient-specific patterns evolve gradually over weeks; in personalized recommendation systems, user preferences drift over time; and in financial time-series analysis, market regimes may change abruptly. In all these settings, PQC parameters cannot be regarded as static, but must be continually adapted to track the underlying time-varying data distribution; otherwise, the predictive performance of the VQA will inevitably degrade. More generally, in quantum machine learning tasks, PQC parameters tuned to the data acquired at a given moment may later lead to degraded performance and reduced model accuracy as data properties evolve. Thus, it becomes necessary to rethink the parameter optimization process when data are not fully available at once but are accessed progressively over time. Indeed, the choice of repeatedly executing the costly operation of parameter tuning is impractical and counterproductive, while, we have to update the parameters, in order to account for the properties of newly incoming data. Consequently, ML algorithms, which could support this process, must also be designed from scratch, as those available in the literature–originally tailored for batch processing–would not be adequate for the challenges at hand. It is also reasonable to expect that studies on this kind of problem could broaden the range of application domains for QC approaches.

In this paper, we investigate the problem of tuning the parameters of PQCs operating within a Continual learning paradigm (Parisi et al. 2019; Van De Ven and Tolias 2019). Here, incoming data must be processed continuously, and optimized parameter configurations–based on data acquired up to that point–must be returned simultaneously. In such a scenario, PQCs could underpin the application of quantum circuits, for instance, in quantum machine learning tasks within *lifelong* learning paradigms (Homayouni et al. 2020). We propose a ML-based method, named as **QURIOSO** (QUantum gate parameteR predictIOn through quantum-enhanced long-Short term memOry), to estimate parameters and support the optimization process both to refine parameters and reach convergence. The high-level purpose is to reduce the workload of the iterative procedure of the optimizer. To do that, our method learns the behavior of the optimization function when operates near the convergence, enabling it to predict promising parameters based on previously observed configurations. Thus, instead to explore the parameter space, the method uses learned predictions of the parameters, obtained through a supervised learning approach, to guide the search more efficiently.

The method relies on Sequence modeling techniques (Xie et al. 2023). because parameter data are inherently sequential. Indeed, at each execution the PQC proceeds through a series of iterations returning parameter configurations. Also, PQC operates in a continuous computations and therefore may be repeatedly executed over time. Therefore, Sequence modeling is well-suited for this problem. By guiding the optimizer with learned insights, the method aims to accelerate convergence, reducing the number of iterations required. This enhances the practicality and effectiveness of VQAs and opens new possibilities for NISQ devices, pushing the boundaries of what can be achieved in this emerging field.

The remaining sections of this paper are organized as follows: In Section 2, we discuss the motivation behind QURIOSO and draw the main contributions, both technical and methodological. In Section 3, through the workflow of QURIOSO, we explain the methodological decisions and algorithmic details. Specifically, in Section 3.1 we report the notions and notations adopted and introduce the scientific problem at hand. Section 3.2 provides a detailed description of three algorithms that define the workflow. Section 3.2.1 discusses the QURIOSO procedure. Section 3.2.2 provides details on the procedure of generation of the input data of the LSTM architectures. Section 3.2.3 presents the LSTM architectures which build the forecasting model utilized. Section 4 proposes experimental evaluation of QURIOSO in a real-world scenario and provides discussions on the performances. Section 5 offers an overview of the related literature, and finally, Section 6 concludes the paper.

2 Overview and contribution

In this paper, we study the design and application of VQAs within a Continual learning, a topic that, to the best of our knowledge, has not yet been explored in the literature. In Continual learning, the task remains fixed (in the sense that the output space, loss function, and input–output mapping are invariant over time), while the input distribution changes, requiring the model to adapt to successive distribution shifts without forgetting what previously learned ones (Van De Ven and Tolias 2019). In this context, VQAs are required to operate continuously, minimizing a cost function over a flow of data that arrives without a predefined endpoint. These data are inherently ordered due to their constant generation, making the computational setting intrinsically sequential.

The central challenge lies in continuously tuning the parameters of the PQCs underlying the VQAs, regardless of the specific optimization method used (e.g., gradient-based or gradient-free). In a continuous computation setting, this tuning process should be performed unceasingly in order to adapt the model to the new data. As a result, a refined PQC configuration may never be ready for deployment, since it is constantly subject to refinement, potentially limiting the usefulness of the VQA. Another issue arises from the rate of data generation, which may surpass the ability of the VQA to process previously acquired data samples.

In response to these challenges, we introduce a paradigm shift: a machine learning (ML)-assisted framework to support VQAs operating on sequential data. Specifically, we propose QURIOSO, an ML model designed to simulate the behavior of the optimization process as it approaches convergence. Near convergence, the optimizer identifies promising PQC parameter values and local minima of the objective function. These parameters represent a valuable source of information, which we use to train a predictive model capable of estimating future parameter values. This predictive model inherently acts as a forecasting tool, estimating future PQC parameters based on historical data.

Without loss of generality, in QURIOSO (Fig. 1) we assume that data samples can be processed in consecutive data blocks. Data samples can either have binary labels or be unlabelled. The VQA is embedded within a binary classification model trained on the sole labeled samples in each data block. The binary classifier addresses the domain-specific ML task. After processing a data block, the VQA produces a sequence of refined PQC parameters that serve as training data to learn the forecasting model. This is subsequently used to estimate the PQC parameters for the next data block. Specifically, QURIOSO supports

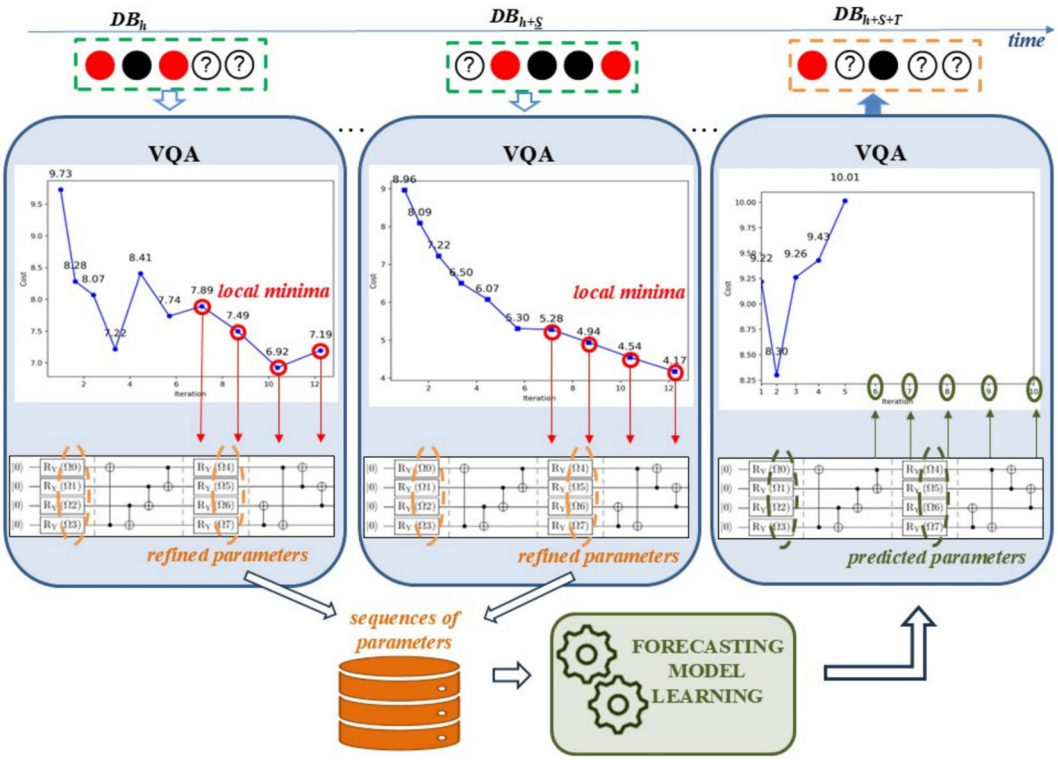


Fig. 1 A high level representation of QURIOSO. The workflow begins with continuously arriving data, partitioned into consecutive data blocks. Each block is processed by a VQA, embedded within a binary quantum classifier, to produce a sequence of optimized PQC parameter sub-sequences. These sub-sequences are fed into a forecasting model learning component which exploits temporal correlation to forecast the parameter configuration of the next block. As a result, we have a binary classifier adapted to the current data block thanks to the predicted VQA parameters produced for the same block

two modes of operation: *(i) forecast deployment*, where parameters are first refined through optimization and then passed to the forecasting model, which provides new parameter estimations; and *(ii) forecast refinement*, where parameters are refined through an initial optimization process, input to the forecasting model to generate new estimations, and subsequently refined again through a final optimization step. While mode (i) ensures rapid processing and immediate utilization for the binary classifier, mode (ii) enables a tighter adaptation to the characteristics of the next data block. In both cases, the goal is to assist the VQA in handling the labeled samples by providing informed parameter estimates, thereby reducing the number of optimization iterations required for convergence on the current data block. This becomes especially valuable when designing PQCs for continuously incoming data, where performing full optimization from scratch on every data block would be prohibitively time-consuming.

It should be clarified that the forecasting model also relies on optimization techniques. However, unlike the repeated execution of an optimizer for every data block, the forecasting model can be potentially trained only once– from the training data blocks of the binary classifier–and is then queried to provide predictions on subsequent data blocks. This one-time training approach offers a distinct efficiency advantage over re-running the optimizer for each new block.

To build the forecasting model in QURIOSO, we resort to sequence learning task, where the input becomes the time series of optimized parameter sub-sequences. Each sub-sequence encodes the local data distribution within each data block, and, considered all together, trace the the trajectory of the optimizer toward convergence. These sub-sequences capture the temporal evolution of the optimization trajectory, reflecting how the PQC parameters adapt to changes in the input data distribution across consecutive data blocks.

Sequence modeling techniques, such as *LSTM* (Long short term memory) (Hochreiter and Schmidhuber 1997a) are well-suited to learn temporal evolution across such ordered inputs. By leveraging historical sequences, they are specifically able to accommodate two statistical properties of the time-series we find in parameter

sub-sequences, *temporal proximity* and *temporal recurrence* (Rossi 2018). In temporal proximity, nearby parameter configurations tend to be more strongly correlated than distant ones because the optimization landscape shifts incrementally as new data arrive. The memory units of the LSTMs intrinsically considers these short-term dependencies, allowing it to capture gradual changes in PQC parameters from one data block to the next. In temporal recurrence, Similar patterns of parameter evolution can reappear whenever the data distribution exhibits repeated behaviors. The ability of the LSTM to retain long-term memory enables it to recognize and leverage these replicated segments, even if they occur many data blocks apart. For experimental purposes, we propose two alternative implementations, one based on purely classical-computing layers, the other one enhanced by quantum layers, where the canonical gates of forget, input and output are determined by quantum measurement outcomes instead of classical nonlinear activations.

To sum up, the methodological and technical contribution which this manuscript aims to provide:

- Machine learning supporting Quantum Computing. We present a concrete application of Artificial intelligence in support of Quantum computing by designing an LSTM-based forecasting model. This model predicts VQA parameters, accelerating the optimization process within VQAs by reducing the number of iterations needed for convergence.
- VQA in lifelong/continual learning. We explore the deployment of VQAs within a lifelong/continual learning (a machine learning paradigm withing continuous computation)– an underexplored scenario in which data arrive in an unbounded flow. We highlight the specific challenges posed by this setting, including incremental parameter tuning and real-time model adaptation.
- A multi-approach quantum-classical architecture. QURIOSO integrates a sequence modeling approach to perform domain-specific binary classification. Moreover, the strategy of constructing PQCs through predictive forecasting is quite general and can be adapted to other machine learning tasks that leverage VQAs.
- Comparative evaluation of alternative forecasting models. We evaluate both conventional LSTM and a quantum-enhanced counterpart. While the classical LSTM captures intrinsically non-linearity via activation functions, the quantum-enhanced version achieves similar abilities by using measurement outcomes of quantum circuits–which naturally encode non-linear transformation.

3 Quantum gate parameter predictIon through sequence modeling

In this section, we introduce basic concepts and notations and then we provide the algorithmic details of the workflow of QURIOSO represented in Fig. 1.

3.1 Concepts and notations

QURIOSO faces a binary classification problem in the setting of domain-incremental learning (Loglisci et al. 2024). Specifically, we have data samples described by $\mathrm{X} \cup y$, X are values of the set X of descriptive features or attributes, and $y \in \{+1, -1\}$ denotes the class label. Domain-incremental learning is a machine learning paradigm within the broader field of lifelong learning. It refers to a scenario where the distribution of the class label y remains unchanged, while the domains of X shift over time. This justifies the adaptation of a PQC-based classifier to new parameter settings that mirror new data samples, while using the same objective function for the VQA.

The data samples arrive individually, one after another, without a predefined order between labeled and unlabeled samples. They are collected into equally-sized data blocks, which include both labeled data (where y is provided) and unlabeled data (where the to-be-estimated label is denoted as $\hat{y}$). This aligns with the realistic assumption that the distribution of labeled and unlabeled data samples is not known a priori.

We denote the h-th data block as DB_h, the portion of labeled samples as L_h, the portion of unlabeled samples as U_h. Each data block DB_h is used to run the VQA process for a fixed number of iterations I, effectively training the binary classifier on that block. We consider S consecutive data blocks to train the classifier. The execution of the VQA solves an optimization problem formulated as:

$$\min_{\boldsymbol{\theta_h}} \mathbb{E}_{(X,y)\sim DB_h} L\left(y, f(X;\boldsymbol{\theta}_h)\right)$$

where:

- $DB_h = \left\{\left(\mathrm{X}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{|DB_h|}$, where $\mathrm{X}^{(i)}$ is a classical input encoded into a quantum state $|\phi(\mathrm{X}^{(i)})\rangle$, and $y^{(i)}$ is the corresponding label.
- $\boldsymbol{\theta}_h \in \mathbb{R}^d$: trainable parameters of the quantum circuit at the data block DB_h
- $L(y, \hat{y})$: loss function defined as cross-entropy
- $f(X;\boldsymbol{\theta}_h)$: prediction function defined as an expectation value

$$f(X;\boldsymbol{\theta}_h) = \langle \phi(X)|U^\dagger(\boldsymbol{\theta}_h)\hat{O}U(\boldsymbol{\theta}_h)|\phi(X)\rangle$$

- $\hat{O}$: observable defined as Pauli-Z operator measured at the end of the circuit $U(\boldsymbol{\theta}_h)$

For each data block DB_h (and specifically for the portion L_h), the VQA produces I consecutive parameterizations $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_I}]$[1] (each resulting from one iteration) and the corresponding sequential values of $L(y, \hat{y})$. From these, we select a subsequence of k values of local minima $[l_{h_1}, \ldots, l_{h_k}]$ (h=1,...,S), and we trace back the corresponding parameters $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}]$.

At the end of the binary classifier training, we will have S subsequences, each of dimension d and length k. These subsequences are concatenated to construct multivariate time series of length M, which become inputs for the forecasting model.

We define them as the set W:

$$W = \left\{ [\boldsymbol{\theta}_{h_1}, \boldsymbol{\theta}_{h_2}, \ldots, \boldsymbol{\theta}_{h_M}] \;\middle|\; 1 \leq j \leq (S \times k) - M + 1 \right\}.$$

each $\boldsymbol{\theta}_{h_j}$ denotes the j-th parameter in the time-series of length formed by concatenating the subsequences of length k (of consecutive data blocks) with a slide of one step.

For illustrative purposes, we consider S=2, k=4, M=5. Given the subsequences:

$[\boldsymbol{\theta}_{1_1}, \boldsymbol{\theta}_{1_2}, \boldsymbol{\theta}_{1_3}, \boldsymbol{\theta}_{1_4}]$,
$[\boldsymbol{\theta}_{2_1}, \boldsymbol{\theta}_{2_2}, \boldsymbol{\theta}_{2_3}, \boldsymbol{\theta}_{2_4}]$,
$[\boldsymbol{\theta}_{3_1}, \boldsymbol{\theta}_{3_2}, \boldsymbol{\theta}_{3_3}, \boldsymbol{\theta}_{3_4}]$,
$[\boldsymbol{\theta}_{4_1}, \boldsymbol{\theta}_{4_2}, \boldsymbol{\theta}_{4_3}, \boldsymbol{\theta}_{4_4}]$.

the set W becomes

$[\boldsymbol{\theta}_{1_1}, \boldsymbol{\theta}_{1_2}, \boldsymbol{\theta}_{1_3}, \boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}]$
$[\boldsymbol{\theta}_{1_2}, \boldsymbol{\theta}_{1_3}, \boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}, \boldsymbol{\theta}_{2_2}]$
$[\boldsymbol{\theta}_{1_3}, \boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}, \boldsymbol{\theta}_{2_2}, \boldsymbol{\theta}_{2_3}]$
$[\boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}, \boldsymbol{\theta}_{2_2}, \boldsymbol{\theta}_{2_3}, \boldsymbol{\theta}_{2_4}]$

When $M < k$ the forecasting model learns the progression of the parameters within the same data block. When $M > k$, it learns what happens across consecutive data blocks.

In light of these concepts, the problem investigated by QURIOSO can be stated as follows:

[1] The notation $[i_1, \ldots, i_n]$ refers to a sequence of n values resulting from n consecutive processes.

Given

$W = \left\{ [\boldsymbol{\theta}_{h_j}, \boldsymbol{\theta}_{h_{j+1}}, \ldots, \boldsymbol{\theta}_{h_{j+M-1}}] \;\middle|\; 1 \leq j \leq (S \times k) - M + 1 \right\}.$

generated by running a VQA over S consecutive data blocks $\{DB_1, \ldots, DB_S\}$ for I iterations each,

Find

a forecasting model F to estimate the values of $[\boldsymbol{\theta}_{(S+T)_1}, \ldots, \boldsymbol{\theta}_{(S+T)_k}]$ by enabling the VQA working on the data block DB_{S+T} ($T \in \mathbb{Z}$) for J iterations, with $J < I$.
The model F has to be trained on the set W to learn to provide the estimation $\hat{\boldsymbol{\theta}}_{h_{j+M-1}}$ of the value $\boldsymbol{\theta}_{h_{j+M-1}}$. Thus, F has to be queried on the subsequence $[\boldsymbol{\theta}_{i_1}, \ldots, \boldsymbol{\theta}_{i_{M-1}}]$ of parameters refined by the VQA, in order to provide the estimation $\hat{\boldsymbol{\theta}}_{i_M}$ of the value $\boldsymbol{\theta}_{i_M}$.

For illustrative purposes, the model F is trained on the time series of length M=5

$[\boldsymbol{\theta}_{1_1}, \boldsymbol{\theta}_{1_2}, \boldsymbol{\theta}_{1_3}, \boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}]$
$[\boldsymbol{\theta}_{1_2}, \boldsymbol{\theta}_{1_3}, \boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}, \boldsymbol{\theta}_{2_2}]$
$[\boldsymbol{\theta}_{1_3}, \boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}, \boldsymbol{\theta}_{2_2}, \boldsymbol{\theta}_{2_3}]$
$[\boldsymbol{\theta}_{1_4}, \boldsymbol{\theta}_{2_1}, \boldsymbol{\theta}_{2_2}, \boldsymbol{\theta}_{2_3}, \boldsymbol{\theta}_{2_4}]$

to be queried on the time-series $[\boldsymbol{\theta}_{i_1}, \boldsymbol{\theta}_{i_2}, \boldsymbol{\theta}_{i_3}, \boldsymbol{\theta}_{i_4}]$ of length $M-1$ to provide the estimation $F([\boldsymbol{\theta}_{i_1}, \boldsymbol{\theta}_{i_2}, \boldsymbol{\theta}_{i_3}, , \boldsymbol{\theta}_{i_4}])=\hat{\boldsymbol{\theta}}_{i_M}$

3.2 Algorithms

QURIOSO has been designed as a hybrid framework that combines classical and quantum techniques. The control of the whole workflow and execution of the VQA over data blocks are handled by classical computing. The VQA itself is hybrid as well, since it relies on a classical iterative processing aimed at refining the parameters. Hybrid is also the design of a crucial component of QURIOSO, namely the learning of the forecasting model. As we will explain later, the forecasting model resorts to a LSTM architecture enhanced with two quantum mechanisms: quantum measurements, used to inject non-linearity produced by quantum circuit evolution into the neural architecture, and entanglement-based gates, used to accommodate correlations between time steps.

In this section, we report the algorithms that define QURIOSO (Fig. 1) and provide the pseudo-code representation and procedural details. Algorithm 1 enacts the workflow illustrated in Fig. 1. It handles the generation of the sequential data from each execution of the VQA and the learning process of forecasting model, respectively. For clarity, all symbols and indices used in the pseudo-code are summarized in Table 1.

3.2.1 The procedure of QURIOSO

Algorithm 1 processes data blocks individually by solving a binary classification problem on each of them. The classifier is designed in the form of the PQC embodied in the VQA. Specifically, Algorithm 1 operates in Domain-incremental learning (a sub-task of Continual learning Van De Ven and Tolias 2019) such that the classifier is trained on the portions of labeled samples L_h of S of data blocks. That is, it is trained from scratch on L_1 and subsequently updated on the labeled samples L_h as they arrive. PQCs offer model adaptability and therefore align well with domain-incremental learning. Indeed, the parameter values are tuned through a refinement process that gradually incorporates the properties of incoming data. Thus, when new labeled samples arrive, existing quantum models can be adapted by updating their parameters, ensuring adaptability and extensibility without requiring retraining from scratch. Moreover, the unitary nature of quantum circuits allows quantum states to evolve reversibly, meaning that updates do not erase previously acquired knowledge unless a measurement is performed.

Table 1 Notation used in QURIOSO (Algorithm 1)

Symbol	Description
L_h	Labeled samples in the h-th data block
U_h	Unlabeled samples in the h-th data block
S	Number of initial data blocks used for QURIOSO setup
T	Number of subsequent data blocks used for evaluation
$\boldsymbol{\theta}^{init}$	Uniform initial PQC parameters
$\boldsymbol{\theta}_h$	PQC parameters at block h
I	Number of full optimisation iterations
JP	Number of partial optimization iterations
JS	Number of refinement iterations
k	Number of extracted PQC parameterizations
$\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}$	Extracted parameters from PQC
$\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_{M-1}}$	Sequence of previously extracted PQC parameters used as input to the forecasting model
$\hat{\boldsymbol{\theta}}_{h_{JP+1}}$	Parameters predicted by the forecasting model F after JP partial optimization iterations
W	Training set of parameter sequences for the forecasting model
M	Multivariate time series length
F	Forecasting model
X	Fixed feature representation shared across data blocks
CS	Criterion for extracting relevant local minima
L	Sequence of cost function values (loss history) used to extract local minima
$\hat{Y}$	Predicted labels for the unlabeled samples U_h
h	Index of the current data block

Preliminarily, under the assumption that the descriptive features remains the same in domain-incremental learning, Algorithm 1 selects the subset X from the descriptive features to be used for all the subsequent data blocks (line 1). To this end, it considers the mutual information between the class labels (Ross 2014; Loglisci and Malerba 2023), which measures the degree of relatedness between pairwise variables. Each execution of the VQA (lines 3, 8) solves an optimization problem (as formulated in Section 3.1) and returns a sequence of parameterizations $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_I}]$. Except for the first execution, corresponding to L_1 (line 3), where the parameters are initialized with uniform values ($\boldsymbol{\theta}^{init}$), in the subsequent calls the parameters are set (line 8) as the last parameterization $\boldsymbol{\theta}_{h_I}$ produced by the previous call on the samples L_{h-1}. This design ensures adaptability and extensibility without requiring retraining from scratch. Then, Algorithm 1 selects a set of k parameterizations (lines 4,9) based on the corresponding local minima, through the function $extract\_points()$ which we will explain later. Once a set of S labeled samples has been processed, the algorithm has accumulated into W as many subsequences of k-parameterizations $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}]$ (lines 5, 10). Next, the forecasting model is trained on the sequences of length M created on W, as described at Section 3.1, using the quantum/classical LSTM-based architectures which we will explain later.

So far, after a phase of construction of the binary classifier (underlying the VQA) and forecasting model F, Algorithm 1 uses only labeled samples. In the data blocks subsequent to DB_S, however, it processes both labeled samples and unlabeled samples. The former are used as an adaptation step to the new data blocks. In this phase, the VQA is launched for a number of iterations (JP and JS) smaller than I (lines 15,19). The unlabeled samples, instead, are the target of the binary labeling, once the parameters of the PQC have been tuned through the optimizing iterations (JP and JS) and estimations provided by the model F. This reflects the core idea of this manuscript: the few VQA iterations of the VQA on L_h ($h > S$) aim to inject the properties of newly incoming samples into the parameters of PQC, while the predictions of F inject the capability to converge learned by F on past samples with similar properties. The model F is built over sequences of parameterizations because it has to learn the behavior of the optimization function from "instances of converging processes". Conversely, F is queried on a single

parameterization (produced after a few iterations) in order to guide it toward convergence. In particular, F is queried to both refine the parameters generated by the VQA (after JP iterations) and guide to convergence (after JS iterations). This behavior corresponds to the lines 14–25, where F is used to handle the two modes introduced in Section 2– i) forecast deployment and ii) forecast refinement. The number of optimizer iterations predefined in the mode i) is constrained by the input threshold JP. Thus, for each pair (L_h, U_h), the VQA is executed for JP iterations ($JP < I$) to obtain an initial sequence of configurations compatible with the M-long sequences learned by F (lines 15,16). Indeed, the line 17 returns the estimation $\hat{\boldsymbol{\theta}}_{h_M}$ corresponding to the predicted parameterization following $\hat{\boldsymbol{\theta}}_{h_{M-1}}$ which is appended to the last generated one, namely $\hat{\boldsymbol{\theta}}_{h_{JP}}$. The mode i) is completed with the deployment of $\hat{\boldsymbol{\theta}}_{h_{JP+1}}$ to the binary classifier, which therefore can provide the binary labels $\hat{Y}$ for U_h (line 22).

While the mode i) is predefined in the workflow of QURIOSO, the mode ii) is optional. It can be activated by setting a non-zero value for the threshold JS, which determines the number of optimizer iterations performed after querying F. In this case, the VQA further refines the estimation $\hat{\boldsymbol{\theta}}_{h_{JP+1}}$ while still operating on L_h. The mode ii) is completed with the deployment of $\hat{\boldsymbol{\theta}}_{h_{JS+1}}$ to the binary classifier, which therefore can provide the binary labels $\hat{Y}$ for U_h (line 20).

The design of QURIOSO is intentionally general with respect to the underlying VQA, and the implementation of Algorithm 1 is independent on the specific PQC used, making the approach broadly applicable across different PQCs. The VQA presents the usual structure with two main constituents, that is, *i)* encoding circuit designed with quantum gates at fixed parameters and responsible for mapping classical data into quantum states; *ii)* composite gate-block consisting of parameterized gates, whose parameters are iteratively refined by an optimization algorithm. Numerous alternatives can be considered for each of these components, however, for the purpose of this work, it suffices to employ standard circuit patterns from the literature. The sole choice we make is to replicate the same circuit pattern along the PQC. Consequently, the total number of the parameters amounts to d (Section 3.1).

Although this choice can exacerbate decoherence issues on real quantum machines, it provides a suitable foundation for the forecasting model to learn from the optimizing function when tuning related parameters. Indeed, if the same parametric gate $U(\theta)$ is applied at every layer, the full sequence

$$U(\theta^{(L)})\,U(\theta^{(L-1)})\,\ldots\,U(\theta^{(1)})$$

inherits symmetries. For example, swapping $\theta^{(1)} \leftrightarrow \theta^{(L)}$ might not change the value of the objective function. This can lead to the existence of equivalence classes, where the parameters at the first and last layers behave similarly (Cerezo et al. 2021). Furthermore, during the optimization process, for instance based on gradient descent, the gradient at layer 1 indirectly influences the parameters of the subsequent layers. Specifically, updating $\theta_i^{(1)}$ alters the state for layers $2, 3, \ldots, L$, meaning that the refinements at layer L are dependent on the earlier modifications (Schuld et al. 2020; Pesah et al. 2021).

Algorithm 1 QURIOSO

Require: $[(L_1, U_1), \ldots, (L_h, U_h), \ldots, (L_S, U_S), \ldots, (L_{S+T}, U_{S+T})]$, M, JP, JS, CS
1: $X \leftarrow select\_features(L_1)$
2: $h = 1$; $W \leftarrow \oslash$
3: $L, \boldsymbol{\theta}_h \leftarrow execute(VQA(\boldsymbol{\theta}^{init}), I, L_h, X)$
4: $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}] \leftarrow extract\_points(L, \boldsymbol{\theta}_h, k, CS)$
5: $W \leftarrow [\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}] \cup W$
6: $h++$
7: **while** $(h < S)$ **do**
8: $L, \boldsymbol{\theta}_h \leftarrow execute(VQA(\boldsymbol{\theta}_{h_I}), I, L_h, X)$
9: $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}] \leftarrow extract\_points(L, \boldsymbol{\theta}_h, k, CS)$
10: $W \leftarrow [\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}] \cup W$
11: $h++$
12: **end while**
13: $F \leftarrow train\_model(W, M)$
14: **while** $(h < T)$ **do**
15: $L, \boldsymbol{\theta}_h \leftarrow execute(VQA(\boldsymbol{\theta}^{init}), JP, L_h, X)$
16: $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_{M-1}}] \leftarrow extract\_points(L, \boldsymbol{\theta}_h, M-1, CS)$
17: $\hat{\boldsymbol{\theta}}_{h_{JP+1}} \leftarrow forecast(F, [\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_{M-1}}])$
18: **if** $(JS > 0)$ **then**
19: $L, \boldsymbol{\theta}_h \leftarrow execute(VQA(\hat{\boldsymbol{\theta}}_{h_{JP+1}}), JS, L_h, X)$
20: $\hat{Y} \leftarrow classify(VQA(\hat{\boldsymbol{\theta}}_{h_{JS+1}}), U_h)$
21: **else**
22: $\hat{Y} \leftarrow classify(VQA(\hat{\boldsymbol{\theta}}_{h_{JP+1}}), U_h)$
23: **end if**
24: $h++$
25: **end while**

3.2.2 Generation of training data from parameterizations

QURIOSO constructs the training data for the forecasting model by using the sequences of parameterizations $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_I}]$ produced by the VQA while building the binary classifier on the training data blocks. For each sequence $\boldsymbol{\theta}_h$, it selects a subset of k d-dimensional parameterizations, on the basis of the characteristics of the associated loss values L (Algorithm 1, lines 4, 9, and 16). This selection is performed using the algorithm `extract_points()` (Algorithm 2), specifically designed to identify "instances of optimization convergent processes." As known, the completion of the optimization step does not necessarily imply the determination of the local minima. Indeed, these could be identified

even during the process, at the intermediate iterations and not necessarily at final iterations. To cope with these eventualities, the Algorithm 2 implements two alternative modes, each activated with the option *CS* . When it is being chosen (line 1), the algorithm constructs a set of k parameterizations whose associated losses better represents a converging process. Specifically, it first generates all subsequences of L of length k, and, for each subsequence, it computes the sum of absolute gradients (differences) between consecutive iterates. Then, it selects the subsequence with the smallest total gradient, indicating the most stable region near convergence, and return the corresponding k values of $\boldsymbol{\theta}_h$. When *CS* is not activated, the algorithm works under the hypothesis that the losses have a decreasing tendency and therefore extracts the last k points from the initial set of m losses to construct a sequence denoting a converging process.

Algorithm 2 extract_points()

Require: $L = [\ell_1, \ell_2, \ldots, \ell_m]$, $\boldsymbol{\theta}_h = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_m]$, k, CS
1: **if** CS **then**
2: $L \leftarrow \left\{ [\ell_j, \ldots, \ell_{j+k-1}] \;\middle|\; j = 1, \ldots, m-k+1 \right\}$
3: $G \leftarrow \left\{ \sum_{t=1}^{k-1} |\ell_{j+t} - \ell_{j+t-1}| \;\middle|\; [\ell_j, \ldots, \ell_{j+k-1}] \in S \right\}$
4: $r \leftarrow \arg\min(G)$
5: **return** $[\boldsymbol{\theta}_{h_1}, \ldots, \boldsymbol{\theta}_{h_k}] \leftarrow sort([\boldsymbol{\theta}_r, \ldots, \boldsymbol{\theta}_{r+k-1}], descending)$
6: **else**
7: **return** $[\boldsymbol{\theta}_{m-k+1}, \ldots, \boldsymbol{\theta}_m]$
8: **end if**

3.2.3 Sequence modeling through LSTM architectures

The training data produced by Algorithm 2 feeds the learning process of the forecasting model F (line 13). The forecasting model consists of d LSTM-based submodels, each assigned to one parameter and trained to estimate its value in the range $[0, 2\pi]$. Each submodel receives, as input, a time-series of length $M-1$ and predicts the M-th value, as also formulated in Section 3.1:

$$[\theta_{h_1}, \theta_{h_2}, \ldots, \theta_{h_{M-1}}] \mapsto \theta_{h_M},$$

where θ_{h_t} is a scalar representing the parameter value at time step t.

We implement F with two variants inspired by LSTM networks, a class of recurrent neural networks specifically designed to capture long-range dependencies in sequential data while mitigating the vanishing gradient problem (Hochreiter and Schmidhuber 1997a). The first variant is inspired to the design of the LSTM in classical computing. An LSTM network processes a sequence over time and consists of a chain of LSTM cells. At each time step t, an LSTM cell receives the current input θ_{h_t} and the previous hidden and cell states, applies gating operations that regulate information flow, and outputs updated versions of the hidden state and cell state. This enables the model to maintain both short and long term memory.

Each LSTM cell maintains a hidden state n_t and a cell state c_t, which are propagated along the sequence. Three gates control how the information is handled, that is, *forget*, which controls the information to be discarded, *input*, which controls the information to be stored, and *output*, which controls the information to be returned. The input gate works in tandem with a candidate memory vector $\tilde{c}_t$, which encodes new information potentially useful for the cell state. The cell state acts as long-term memory, while the hidden state reflects short-term memory for prediction. The gating operations implement the following equations:

$$f_t = \sigma(W_f \theta_{h_t} + U_f n_{t-1} + b_f)\,, \tag{1}$$

$$i_t = \sigma(W_i \theta_{h_t} + U_i n_{t-1} + b_i)\,, \tag{2}$$

$$\tilde{c}_t = \tanh(W_c \theta_{h_t} + U_c n_{t-1} + b_c)\,, \tag{3}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{4}$$

$$o_t = \sigma(W_o \theta_{h_t} + U_o n_{t-1} + b_o)\,, \tag{5}$$

$$n_t = o_t \odot \tanh(c_t), \tag{6}$$

where $\odot$ denotes element-wise multiplication.

where,

- δ is the hidden dimension of the LSTM cell, that is, dimensions of the hidden state and cell state;
- $\theta_{h_t} \in [0, 2\pi]$ is the input at time t ;
- $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid activation function;
- $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ is the hyperbolic tangent activation function;
- $f_t, i_t, o_t \in (0,1)^\delta$; $\tilde{c}_t, n_t \in (-1,1)^\delta$; $c_t \in \mathbb{R}^\delta$;
- $W_f, W_i, W_o, W_c \in \mathbb{R}^{\delta \times 1}$ are input weight matrices;
- $U_f, U_i, U_o, U_c \in \mathbb{R}^{\delta \times \delta}$ are recurrent weight matrices;
- $b_f, b_i, b_o, b_c \in \mathbb{R}^\delta$ are bias vectors.

The submodel for one parameter is queried on the previous $M-1$ values of that parameter to predict the M-th value $\hat{\theta}_{h_M} \in [0, 2\pi]$ by applying a linear transformation to the final hidden state n_{M-1}:

$$\hat{\theta}_{h_M} = W_y n_{M-1} + b_y, \tag{7}$$

where $W_y \in \mathbb{R}^{1 \times \delta}$ and $b_y \in \mathbb{R}$ are the output weight and bias, respectively.

The second variant explores a quantum-enhancement of the LSTM, that retains the gating operations-based structure and anticipates each gate with equally-sized quantum circuits (Chen et al. 2020). These circuits are arranged within the LSTM structure to ensure the transmission of a classical long- and short-term memory state between consecutive cells. The goal is to leverage quantum properties, such as superposition and entanglement, to represent and process information more efficiently, In fact, empirical studies have already proven improvement in terms of accuracy over the classical LSTMs, also thanks to representational power of the Hilbert space.

However, in the original design, the size of the qubit register dictates the dimensions of the input vector and hidden state, in fact the total dimensionality of both has to equal the number of qubits. Thus, to match the dimensions of the hidden layer and the cell state, only a subset of the qubits is measured, resulting in a significant quantum information loss. Such a "partial" read-out is performed at each optimization iteration and therefore limits the expressive capacity of the whole qubit register. Several strategies can deal with issue. One is the use of the amplitude encoding, which eliminates the forced one-to-one association between classical features and qubits. This could require re-sizing the qubits or preliminary operations for state preparation. An alternative involves the insertion of entanglement between the qubits encoding the hidden states and those encoding input vector, and then, keeping the latter, since would convey the information of the former. This could necessitate re-designing the data encoding and using a larger qubit register.

A particularly promising solution is to detach the dimension of the LSTM states from the number of the qubits, leaving the four circuits anticipating the three gates and candidate memory (Cao et al. 2023). This allows to set one independently from each other and opens to the possibility to design better the quantum circuits (avoiding large number of qubits) and devise the gating mechanisms closer to the forecasting problem and input sequences.

To implement that, we introduce intermediate representational mappings which i) encode input sequences and LSTM states into subspaces of Hilbert space defined by the quantum circuits and ii) decode measurement outcomes into the LSTM vector space. In particular, as to the *encode*, since the input and hidden state are given with the same representation to the three gates and candidate memory (namely, a concatenating vector), we propose using a unified embedding block which compresses the concatenation " $\oplus$ " of the two vectors n_{t-1} and θ_{h_t} $(n_{t-1} \oplus \theta_{h_t})$ to a lower-dimensional space, sized as the number of the qubits of the four circuits. The advantage of this technique is two-fold, one representational , the other computational. First, it allows the model to build a unified dense feature representation that captures joint temporal patterns between the input and the past hidden state, providing informative elements to all gates. Second, a unique embedding block guarantees fewer weights than four distinct embeddings, resulting in gains in terms of efficiency and in terms of overfitting, each avoid embedding specialized for each of the the tree gates. As to the *decode*, the intermediate mapping serves to project the measurement outcomes (determined on the 2^n basis states, n as number of qubits of the four circuits) into the real-valued space of the activation functions. However, considering the peculiarities of the different gating mechanisms, the choice of using a unifying mapping could not be appropriate, so we argue in favor of gate-specific mappings that would facilitate the transformation of the outcomes in the form and scale needed for each gate of the LSTM cell. The advantage is that a dedicated embedding can learn the precise mapping from quantum outputs to the LSTM vectors, while a single embedding could to be too generic, incurring in the risk of degrading the performances. The gating operations can be adapted in order to integrate parameterized quantum circuits and embedding classical blocks (Cao et al. 2023). Thus, we can reformulate the equations (1)–(6). At each time step *t*, we have

$$z_t = E(n_{t-1} \oplus \theta_{h_t}) \tag{8}$$

$$f_t = \sigma\left(E\left(\mathrm{QG}(z_t)\right)\right) \tag{9}$$

$$i_t = \sigma\left(E\left(\mathrm{QG}(z_t)\right)\right) \tag{10}$$

$$\tilde{c}_t = \tanh\left(E\left(\mathrm{QG}(z_t)\right)\right) \tag{11}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{12}$$

$$o_t = \sigma\left(E\left(\mathrm{QG}(z_t)\right)\right) \tag{13}$$

$$n_t = o_t \odot \tanh(c_t) \tag{14}$$

It can be see that the same design of quantum circuit (QG) is used to generate the arguments of the activation functions $\sigma()$ and $\tanh()$. The same mapping (E) is used for the forget, input and output gates, and candidate memory. The intermediate mapping E compresses classical data (n_{t-1}, θ_{h_t}) which are encoded into quantum states prepared for the QG. The resulting basis states probabilities feed the same block E.

The formulation of the QG is reported below:

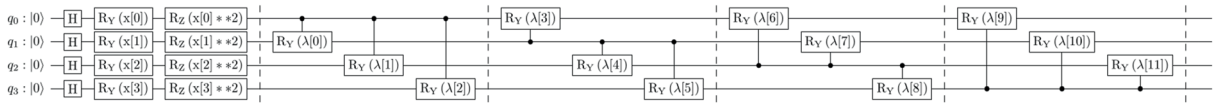


Fig. 2 QG: Encoding block $H^{\otimes 4}R_y^{\otimes 4}(\arctan(\mathbf{\Lambda}))R_z^{\otimes 4}(\arctan(\mathbf{\Lambda}^2))$ followed by parametrized controlled $CRY^{c,t}(\lambda)$ rotations, where $\mathbf{\Omega}$ denotes the trainable parameters of the circuit

$$\begin{aligned}\mathrm{QG}(\mathbf{\Omega},\mathbf{\Lambda}) = &\left[H^{\otimes^4} {R_y}^{\otimes^4}(arctan(\mathbf{\Lambda})){R_z}^{\otimes^4}(arctan(\mathbf{\Lambda}^2))|0^{\otimes^4}\rangle\right]\\ &\left[CRY^{1,2}(\Omega_0)\otimes I^{\otimes^2} CRY^{1,3}(\Omega_1)\otimes I^{\otimes^2} CRY^{1,4}(\Omega_2)\otimes I^{\otimes^2}\right]\\ &\left[CRY^{2,3}(\Omega_3)\otimes I^{\otimes^2} CRY^{2,4}(\Omega_4)\otimes I^{\otimes^2} CRY^{2,1}(\Omega_5)\otimes I^{\otimes^2}\right]\\ &\left[CRY^{3,4}(\Omega_6)\otimes I^{\otimes^2} CRY^{3,2}(\Omega_7)\otimes I^{\otimes^2} CRY^{3,1}(\Omega_8)\otimes I^{\otimes^2}\right]\\ &\left[CRY^{4,3}(\Omega_9)\otimes I^{\otimes^2} CRY^{4,2}(\Omega_10)\otimes I^{\otimes^2} CRY^{4,1}(\Omega_11)\otimes I^{\otimes^2}\right]\end{aligned} \tag{15}$$

with $CRY^{control,target}$[2] denoting the two-qubit controlled gate with a rotation gate $R_y()$ operating on the control qubit, $\mathbf{\Omega}$ denotes the parameter set of the circuit (one parameter for each controlled rotation-x gate *CRY*()). The block $H^{\otimes^4} {R_y}^{\otimes^4}(arctan(\mathbf{\Lambda})){R_z}^{\otimes^4}(arctan(\mathbf{\Lambda}^2))$ represents the encoding of the vector $\mathbf{\Lambda}$ denoting the projection of z_t through *E*. The proposed QG is shown in Fig. 2.

As proposed in Cao et al. (2023), we introduce two innovations compared to Chen et al. (2020), *i)* the explicit implementation of all the connections between qubits explicit, *ii)* the use of these connections to enable a wider exploration of the Hilbert space. The first point is addressed by introducing two qubit gates on pairs of qubits, specifically controlled-gates. Indeed, they establish relationships between the base state $|1\rangle$ of the control and the quantum state of the target. The second point is addressed with controlled-rotational gates, which better tune the dependence between target and control through a condition.

Differently to what presented in Cao et al. (2023), we conjecture the use of controlled-rotational gate on the Y-axis, with respect to the Z-axis. The rotation gate $R_Y()$ combines the amplitudes of $|0\rangle$ and $|1\rangle$ states using $\cos()$ and $\sin()$, therefore the controlled-rotational gate on Y-axis performs this operation upon a condition, that is, only when the control is in the state $|1\rangle$. This allows both dependence and amplitude modulation be injected into a single gate, resulting in a richer transformation than the rotation gate $R_Z()$. In fact, the controlled-rotational gate on Z-axis modifies only the phase between the basis states. While this is sufficient for entanglement, it often requires additional gates (e.g., Hadamard) to convert phase information into measurable amplitude changes. In contrast, the controlled-rotatonal gate on Y-axis directly changes the magnitude and consequently the measurement probabilities, enabling the exploration within a wider space.

It is noteworthy for the purposes of training and forecasting that the capacity of unearthing complex relationships of the classical LSTMs is preserved and even emphasized along "distributed contributions" in the quantum-enhanced variant formulated in the equations (8)–(14), although the evolution of the quantum state along the circuit (until the measurement) is linear in itself, operating always in Hilbert space. In fact, the work of "squashing" that the two activation functions between their unbounded arguments and bounded ranges (that is, [0,1] for $\sigma()$, [-1,1] for $\tanh()$]) is unrolled by the quantum data encoding, parameter-tunable gates of QG and measurements. Indeed, the parameterized gates R_y and R_z introduce a sort of non-linear dependence between the real-valued classical data and magnitude and phase resulting from the rotations along the Y and Z axis of the Bloch sphere. Clearly, this holds on the relationship between the quantum states produced through the refinements of the parameters of the tunable gates and classical data. Additionally, at the end of the circuit, the measurement of the the expectation value of the observable of the quantum state produces a non-linear function on the parameters due to the quantum amplitudes and interference effects.

4 Experiments

We conducted an evaluation of QURIOSO using two classical datasets. The first dataset is the *Spambase*[3] dataset (Hopkins et al. 1999), which consists of approximately 4600 samples described by 57 real-valued features derived from timestamped emails. These features include word and character frequencies, accompanied by binary labels that indicate whether the messages are classified as spam or non-spam. The second dataset is *Ozone*[4] (Zhang et al. 2008), notable for its temporal structure and class imbalance. To ensure a fair evaluation in this sequential context, we implemented a balancing procedure to achieve an approximately equal distribution of classes within each fold prior to training and evaluation.

[2] The notation $CRY^{c,t}$ indicates that the superscript *c* denotes the index of the control qubit, while *t* denotes the index of target qubit of the controlled-not gate.

[3] https://archive.ics.uci.edu/ml/datasets/Spambase

[4] https://archive.ics.uci.edu/dataset/172/ozone+level+detection

QURIOSO has been implemented utilizing Python 3.10 in conjunction with the IBM Qiskit SDK version 1.0.2 (Anis et al. 2021), an extension designed for quantum machine learning applications within the Qiskit framework (Sahin et al. 2025). The implementation was evaluated using the *Aer* simulator. The following libraries were employed: Scikit-learn version 1.4.2 (Pedregosa et al. 2011), as well as Torch version 2.3.0, TensorFlow version 2.16.1 and PennyLane version 0.35.1.

4.1 Evaluated algorithms

We have evaluated several variants of QURIOSO, each designed on different architectural and methodological decisions introduced in Section 3.2. Specifically,

- type of forecasting model, either the classical neural network architecture of Section 3.2.3, named as *LS*, or the quantum-enhanced version of Section 3.2.3, named as *QY*
- selection of the *k* parameterizations used to train the forecasting model, either the generation of the subsequence of *k* losses with the smallest total gradient (*CS*–lines 4,9–Algorithm 1), here named as *ks*, or the generation of the subsequence with the last generated *k* losses, here named as *kn*.

Hence, four alternative variants have been drawn. We name LS_{ks} and QY_{ks} the variants that combine the forecasting model with *CS*, while LS_{kn} and QY_{kn} are the variants with *CS* inactivated, respectively. The implementation of the option *CS* when training of the forecasting model opens up an additional category of variants, the one which uses *CS* to query the model *F* (Algorithm 1–lines 16,17). In fact, once *F* has been trained (line 13), it is being consulted on a sequence of $M-1$ points extracted from those of *JP* iterations. We refer to these variants with the superscripts *sJ* and *nJ*, respectively.

The several variants of QURIOSO have been compared against the baseline designed by a conventional VQA architecture with COBYLA as optimization technique and the parameterized quantum circuit here formulated:

$$\begin{aligned}\mathrm{VQA}(\boldsymbol{\Omega},\boldsymbol{\Theta}) = \\ &[R_y^{\otimes^8}(\boldsymbol{\Omega})] \\ &[CNOT^{7,8}\otimes I^{\otimes^6}][CNOT^{6,7}\otimes I^{\otimes^6}] \\ &[CNOT^{5,6}\otimes I^{\otimes^6}][CNOT^{4,5}\otimes I^{\otimes^6}] \\ &[CNOT^{3,4}\otimes I^{\otimes^6}][CNOT^{2,3}\otimes I^{\otimes^6}] \\ &[CNOT^{1,2}\otimes I^{\otimes^6}][R_y^{\otimes^8}(\boldsymbol{\Theta})]|0^{\otimes^8}\rangle\end{aligned} \tag{16}$$

This formulation represents the circuit structure used for the VQA, developed over a depth of 3. It is composed by multiple layers of parameterized single-qubit rotations around the Y-axis, interspersed with entangling operations. The entanglement follows a reverse order with CNOT gates applied from the last qubit to the first, thereby creating a directed chain of entanglement in reverse order. It presents two set of parameters, $\boldsymbol{\Omega}$ and $\boldsymbol{\Theta}$, both having 8 parameters each. Thus, the VQA presents a total number of 32 parameters, considering that the set $\boldsymbol{\Theta}$ is replicated in the circuit depth (Kandala et al. 2017). The proposed VQA is shown in Fig. 3.

Although QURIOSO offers generality with respect to the relative variational circuit, for experimental purposes, we have adopted the same choices done for the baseline VQA. As to the forecasting model, the Adam optimizer has been used, featuring a learning rate of 0.001, over a number of 200 epochs, for the LSTM model (Hochreiter and Schmidhuber 1997b) we use Tensorflow (Singh and Manure 2019) for the building, proposing a very simple model the LSTM cell and a dense layer for the prediction, instead, regarding the quantum-enhanced version, the model used is described in the Section 3.2.3, and is implemented using Torch (Imambi et al. 2021) and Pennylane (Bergholm et al. 2018).

4.2 Experimental setting and performance measures

The performances have been evaluated following a scheme adequate for the lifelong learning scenario. By the principle of the sequential data, where labeled and unlabeled data are not available as a batch, but become available incrementally and without a predefined order, we considered the Prequential Evaluation (Gama et al. 2013), which is an approach very often used for the evaluation of classical computing

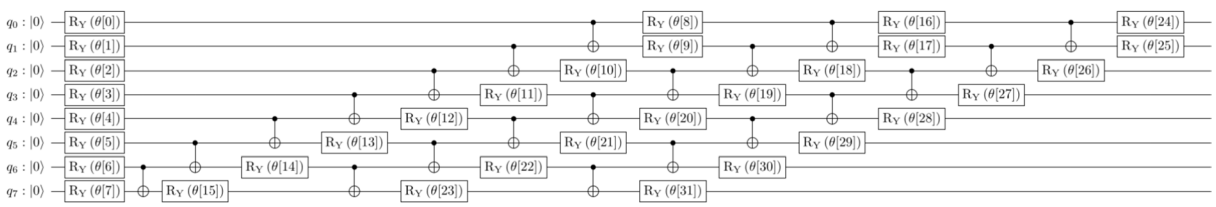


Fig. 3 VQA circuit with depth 3, consisting of alternating blocks of parametrized single-qubit R_Y rotations and reverse-ordered CNOT entangling operations. The architecture uses two parameter sets, $\boldsymbol{\Omega}$ and $\boldsymbol{\Theta}$, for a total of 32 trainable parameters, with $\boldsymbol{\Theta}$ replicated across the three layers

learner performances. Prequential evaluation sequentially combines training data (labeled data used during training sessions) and testing data (labeled data with hidden labels during prediction sessions). This approach allows us to assess the reliability and generality of the classifier on unseen data that has not yet been acquired. Applied to this manuscript, the binary classifier is trained on S labeled data portions. Then, when the portions U_h of unlabeled samples are ready, the predictions commence. The classifier provides estimations on binary labeling, while the ground-truth labels are hidden and retained for evaluation. In essence, for experimental purposes, unlabeled data samples are generated from labeled data with label being hidden. Once processed, the information on the hidden labels is discarded and not used for updating the classifier or subsequent evaluations. This evaluation approach enables us to assess the classifier performance as it encounters new unlabeled data, and ensures that the model performance is evaluated on data that closely reflects its operational environment. The overall collection of data samples has been split into 60 equally sized data blocks, following the chronological order of the samples. The first 12 data blocks have been used to built the S data blocks (Algorithm 1, lines 1–13). The remaining 48 have been used to query the forecasting model (Algorithm 1, lines 14–25) and provide the binary labeling. More precisely, for each of such data blocks, a portion of 70% has been used to execute the VQA (lines 15,19) and query the forecasting model (line 17), while the remaining portion of 30% is the target of the binary classifier (lines 20,22), once the parameters have been estimated (line 17).

The experimental setting also concerns the determination of user-defined thresholds of the baseline VQA and QURIOSO (Algorithm 1). For VQA, the number of the iterations I is fixed to 100. While, for QURIOSO, in order to fully leverage the design of QURIOSO compared to VQA in terms of number of optimizing iterations, JP is set to 40, which is less than half of I. The remaining thresholds are configured as follows:

- k, number of the extracted parameterizations, set to 5, 10, 15
- M, length of the input sequences to train the model F, takes 4 distinct values dependently on the values of k
- JS, number of the refinement iterations once F has been queried, set to 0, $\frac{I}{2} - JP$, $I - JP$. The value 0 implies that the parameters are estimated by sole contribution of F. The value $\frac{I}{2} - JP$ implies that the parameters are estimated by combined effect of F and a few iterations of the variational circuit of QURIOSO (still lower than the half of I). The value $I - JP$ has been considered to evaluate how QURIOSO perform by working on I iterations, the same as the baseline VQA.

Regarding performance measures, we collect the following:

- *moving average accuracy*, which explains how the standard accuracy runs over the unlabelled data portions. It is re-calculated once a data block has been processed.

$$\mathrm{MAA}(h) = \frac{1}{h - S + 1} \sum_{j=S}^{h} \mathrm{Acc}(U_j), \qquad h = S, \ldots, S + T, \tag{17}$$

where $\mathrm{Acc}(U_h)$ is the accuracy on an unlabeled portion U_h.

- *mean accuracy*, which is the accuracy averaged over the set of all unlabeled data portions. This metric tends to approximate the final moving average closely, given that the incremental mean is derived from a comparatively limited number of data portions

$$\mathrm{MeanAcc} \; = \; \frac{1}{T + 1} \sum_{h=S}^{S+T} \mathrm{Acc}(U_h). \tag{18}$$

- *running time* required to complete the overall process illustrated in Algorithm 1, comprising the operation of training of the forecasting model (over labelled data portions) and operation of querying (over unlabelled data portions).

4.3 Results

The experimental results have been collected and presented in order to give empirical evidence to the following research questions:

RQ1: Does QURIOSO achieve an accuracy comparable to or higher than that of the baseline VQA?
RQ2: How does the moving average accuracy evolve across the data blocks, and how it changes in comparison to the baseline VQA?
RQ3: How are the execution times of QURIOSO across data blocks, and how does it compare to the baseline?
RQ4: What is the relationship between PQC depth and parameterisation?

4.3.1 RQ1: Does QURIOSO achieve an accuracy comparable to or higher than that of the baseline VQA?

The first research question (**RQ1**) examines whether the QURIOSO framework can achieve a mean accuracy that is comparable to or exceeds that of the baseline model (VQA). Given that the datasets under consideration exhibit distinct characteristics, we present and discuss the results separately for each dataset.

RQ1 – Spambase. To address **RQ1** for the Spambase dataset, we assess whether the proposed QURIOSO framework can achieve aggregate classification accuracy that is comparable to or exceeds that of the VQA. The findings for the Spambase dataset are summarized in Tables 2, 3, and 4, which correspond to increasing values of the extracted context size k.

For $k = 5$ (Table 2), QURIOSO variants based on quantum-enhanced forecasting (QY) consistently outperform the baseline across all tested values of M. Notably, configurations that utilize slope-based extraction (QY_{ks}^{sJ}) achieve the highest accuracies for small to moderate values of M, indicating that quantum-enhanced forecasting is particularly effective in short-context scenarios. In contrast, classical forecasting variants (LS) occasionally reach baseline-level performance but display higher variability and demonstrate greater sensitivity to the choice of refinement strategy.

As the context size increases to $k = 10$ (Table 3), the advantages of QY models become more evident for larger values of M. While several LS configurations experience a noticeable decline as the temporal window expands, various QY variants maintain or surpass baseline performance, with optimal results arising in settings where forecasting is paired with moderate refinement. These observations imply that quantum-enhanced forecasting offers enhanced robustness as the optimization landscape becomes more complex.

For the largest context size, $k = 15$ (Table 4), the performance differences among models become more subtle. Although the baseline remains competitive, certain QY configurations continue to achieve accuracies that meet or exceed the baseline for intermediate and large values of M. In contrast, most LS variants show a gradual decline in performance, confirming their limited capacity to sustain effectiveness as both context size and temporal aggregation increase.

RQ1 – Ozone. We now direct our focus to the findings derived from the Ozone dataset. As outlined in the respective results tables (Tables 5, 6, and 7), the baseline VQA model consistently attains a mean accuracy of 0.808 across all sequential folds, thereby establishing a robust reference point for the evaluation of the proposed QURIOSO variants within this dataset.

The performance of QURIOSO on the Ozone dataset shows a narrower margin compared to the baseline, in contrast to the broader margin seen in the Spambase dataset. Such behavior is anticipated, given the heightened variability introduced by artificial class balancing, in conjunction with the more heterogeneous nature of the data distribution. Nonetheless, several configurations of QURIOSO demonstrate the capability to match or even exceed baseline accuracy across varying values of k and M, thus affirming the competitiveness of the proposed methodology, even within this more challenging context.

In particular, for smaller context sizes (k=5), both classical (LS) and quantum-enhanced (QY) forecasting variants

Table 2 Mean accuracy values obtained on the Spambase dataset for the different evaluated algorithms ($M \in \left(\frac{k}{2}, k, \frac{2k}{3}, 2k\right)$), with $k = 5$

Algorithms		M=3	M=5	M=8	M=10
VQA		0.730	0.730	0.730	0.730
LS_{ks}^{sJ}	$JS = 0$	0.700	0.698	0.613	0.572
	$JS = \frac{I}{2} - JP$	0.655	0.668	0.700	0.717
	$JS = I - JP$	0.721	0.725	0.710	0.715
LS_{kn}^{sJ}	$JS = 0$	**0.745**	0.710	0.686	0.712
	$JS = \frac{I}{2} - JP$	0.730	0.714	0.698	0.695
	$JS = I - JP$	0.723	0.719	0.715	0.717
LS_{kn}^{nJ}	$JS = 0$	0.702	0.721	0.696	0.720
	$JS = \frac{I}{2} - JP$	0.725	0.728	0.718	0.718
	$JS = I - JP$	0.722	0.740	**0.732**	0.706
QY_{ks}^{sJ}	$JS = 0$	**0.774**	**0.737**	**0.742**	**0.743**
	$JS = \frac{I}{2} - JP$	0.698	0.719	**0.731**	**0.731**
	$JS = I - JP$	0.700	0.722	0.704	0.722
QY_{kn}^{sJ}	$JS = 0$	0.721	0.724	0.728	0.717
	$JS = \frac{I}{2} - JP$	0.711	0.722	0.723	0.716
	$JS = I - JP$	0.707	0.711	**0.739**	0.725
QY_{kn}^{nJ}	$JS = 0$	0.709	**0.736**	0.715	0.725
	$JS = \frac{I}{2} - JP$	0.730	0.699	0.721	**0.732**
	$JS = I - JP$	**0.735**	**0.737**	0.724	0.713

Table 3 Mean accuracy values obtained on the Spambase dataset for the different evaluated algorithms ($M \in \left\{\frac{k}{2}, k, \frac{2\times k}{3}, 2\times k\right\}$) with $k = 10$

Algorithms		M=5	M=10	M=15	M=20
VQA		0.730	0.730	**0.730**	0.730
LS_{ks}^{sJ}	$JS = 0$	0.708	0.683	0.499	0.570
	$JS = \frac{I}{2} - JP$	0.690	0.671	0.632	0.658
	$JS = I - JP$	0.709	**0.738**	0.704	0.715
LS_{kn}^{sJ}	$JS = 0$	**0.734**	**0.735**	0.662	0.726
	$JS = \frac{I}{2} - JP$	0.727	0.716	0.702	0.714
	$JS = I - JP$	0.732	0.727	0.716	0.712
LS_{kn}^{nJ}	$JS = 0$	0.730	0.725	0.650	0.688
	$JS = \frac{I}{2} - JP$	0.730	0.705	0.667	0.691
	$JS = I - JP$	0.710	0.717	0.730	0.709
QY_{ks}^{sJ}	$JS = 0$	0.646	0.660	0.681	**0.751**
	$JS = \frac{I}{2} - JP$	0.675	0.698	0.711	**0.735**
	J+(I - J)	0.712	0.701	0.716	0.706
QY_{kn}^{sJ}	$JS = 0$	0.710	0.683	0.701	**0.736**
	$JS = \frac{I}{2} - JP$	0.694	0.723	0.718	0.715
	$JS = I - JP$	0.726	0.709	0.717	0.717
QY_{kn}^{nJ}	$JS = 0$	0.719	0.710	0.697	0.733
	$JS = \frac{I}{2} - JP$	0.714	0.709	**0.730**	0.723
	$JS = I - JP$	0.721	0.727	0.723	0.730

Table 4 Mean accuracy values obtained on the Spambase dataset for the different evaluated algorithms ($M \in \left\{\frac{k}{2}, k, \frac{2\times k}{3}, 2\times k\right\}$) with $k = 15$

Algorithms		M=8	M=15	M=23	M=30
VQA		0.730	0.730	0.730	0.730
LS_{ks}^{sJ}	$JS = 0$	0.666	0.715	0.567	0.657
	$JS = \frac{I}{2} - JP$	0.687	0.714	0.725	0.679
	$JS = I - JP$	0.717	0.720	0.721	0.699
LS_{kn}^{sJ}	$JS = 0$	0.703	0.648	0.710	0.704
	$JS = \frac{I}{2} - JP$	0.721	0.675	0.705	0.701
	$JS = I - JP$	0.730	0.714	0.719	0.704
LS_{kn}^{nJ}	$JS = 0$	0.714	0.721	0.730	0.673
	$JS = \frac{I}{2} - JP$	0.722	0.709	0.714	0.712
	$JS = I - JP$	0.728	0.730	0.720	0.722
QY_{ks}^{sJ}	$JS = 0$	0.727	0.685	**0.738**	**0.731**
	$JS = \frac{I}{2} - JP$	0.712	**0.733**	0.712	0.722
	$JS = I - JP$	0.713	0.721	0.703	0.721
QY_{kn}^{sJ}	$JS = 0$	0.699	0.699	**0.734**	0.712
	$JS = \frac{I}{2} - JP$	**0.750**	0.702	**0.734**	0.724
	$JS = I - JP$	0.708	0.729	0.726	0.706
QY_{kn}^{nJ}	$JS = 0$	0.721	0.701	0.705	**0.731**
	$JS = \frac{I}{2} - JP$	0.694	0.716	0.722	0.712
	$JS = I - JP$	0.719	**0.732**	**0.734**	0.725

Table 5 Mean accuracy values obtained on the Ozone dataset for the different evaluated algorithms ($M \in \left(\frac{k}{2}, k, \frac{2k}{3}, 2k\right)$), with $k = 5$

Algorithms		M=3	M=5	M=8	M=10
VQA		0.808	0.808	0.808	0.808
LS_{ks}^{sJ}	$JS = 0$	0.792	0.806	0.633	**0.811**
	$JS = \frac{I}{2} - JP$	0.799	**0.810**	0.503	0.806
	$JS = I - JP$	0.803	0.807	0.713	0.718
LS_{kn}^{sJ}	$JS = 0$	0.803	0.806	0.807	0.807
	$JS = \frac{I}{2} - JP$	0.796	0.803	**0.810**	**0.811**
	$JS = I - JP$	0.806	0.803	0.804	0.806
LS_{kn}^{nJ}	$JS = 0$	0.797	0.806	0.804	0.807
	$JS = \frac{I}{2} - JP$	0.796	0.804	0.804	0.806
	$JS = I - JP$	0.807	**0.808**	0.804	0.804
QY_{ks}^{sJ}	$JS = 0$	0.804	0.696	0.742	**0.810**
	$JS = \frac{I}{2} - JP$	0.792	0.721	**0.808**	0.804
	$JS = I - JP$	0.807	0.808	0.804	0.807
QY_{kn}^{sJ}	$JS = 0$	**0.808**	0.806	0.806	0.767
	$JS = \frac{I}{2} - JP$	0.807	**0.808**	0.807	0.800
	$JS = I - JP$	0.808	0.807	0.803	0.807
QY_{kn}^{nJ}	$JS = 0$	0.793	0.804	**0.810**	**0.810**
	$JS = \frac{I}{2} - JP$	0.806	**0.808**	0.803	**0.810**
	$JS = I - JP$	0.800	0.804	0.804	0.808

Table 6 Mean accuracy values obtained on the Ozone dataset for the different evaluated algorithms ($M \in \left\{\frac{k}{2}, k, \frac{2\times k}{3}, 2 \times k\right\}$) with $k = 10$

Algorithms		M=5	M=10	M=15	M=20
VQA		0.808	0.808	0.808	0.808
LS_{ks}^{sJ}	$JS = 0$	0.801	0.542	0.487	0.726
	$JS = \frac{I}{2} - JP$	0.804	0.525	0.717	0.696
	$JS = I - JP$	0.808	0.767	0.772	0.799
LS_{kn}^{sJ}	$JS = 0$	0.804	**0.810**	0.804	0.800
	$JS = \frac{I}{2} - JP$	0.801	0.806	0.807	0.800
	$JS = I - JP$	0.804	0.807	**0.811**	0.799
LS_{kn}^{nJ}	$JS = 0$	0.803	0.799	0.803	0.806
	$JS = \frac{I}{2} - JP$	0.792	**0.808**	0.806	0.806
	$JS = I - JP$	0.803	0.808	0.801	0.804
QY_{ks}^{sJ}	$JS = 0$	0.743	0.807	0.797	0.790
	$JS = \frac{I}{2} - JP$	0.771	**0.810**	**0.811**	0.797
	$J + (I - J)$	0.804	0.807	0.806	0.806
QY_{kn}^{sJ}	$JS = 0$	**0.810**	0.792	0.797	0.794
	$JS = \frac{I}{2} - JP$	0.804	0.786	0.790	0.803
	$JS = I - JP$	**0.812**	0.807	0.807	0.807
QY_{kn}^{nJ}	$JS = 0$	0.801	**0.814**	0.806	0.807
	$JS = \frac{I}{2} - JP$	0.797	0.808	**0.815**	**0.810**
	$JS = I - JP$	0.808	**0.812**	0.803	**0.811**

Table 7 Mean accuracy values obtained on the Ozone dataset for the different evaluated algorithms ($M \in \left\{\frac{k}{2}, k, \frac{2\times k}{3}, 2\times k\right\}$) with $k = 15$

Algorithms		M=8	M=15	M=23	M=30
VQA		0.808	**0.808**	0.808	0.808
LS_{ks}^{sJ}	$JS = 0$	0.799	0.596	0.782	0.751
	$JS = \frac{I}{2} - JP$	0.803	0.782	0.782	0.753
	$JS = I - JP$	**0.810**	0.808	0.794	0.797
LS_{kn}^{sJ}	$JS = 0$	0.803	0.806	0.803	0.799
	$JS = \frac{I}{2} - JP$	0.801	0.803	0.803	0.804
	$JS = I - JP$	0.800	0.803	**0.812**	0.800
LS_{kn}^{nJ}	$JS = 0$	0.800	0.806	0.804	**0.808**
	$JS = \frac{I}{2} - JP$	0.800	0.806	0.803	0.806
	$JS = I - JP$	0.808	0.804	0.804	0.807
QY_{ks}^{sJ}	$JS = 0$	0.740	0.714	0.740	0.804
	$JS = \frac{I}{2} - JP$	0.771	0.733	0.792	0.768
	$JS = I - JP$	**0.810**	0.804	0.808	0.803
QY_{kn}^{sJ}	$JS = 0$	0.796	0.776	0.782	0.793
	$JS = \frac{I}{2} - JP$	**0.810**	0.793	0.771	0.794
	$JS = I - JP$	**0.810**	0.806	0.808	0.803
QY_{kn}^{nJ}	$JS = 0$	**0.808**	0.801	0.799	0.804
	$JS = \frac{I}{2} - JP$	0.807	0.804	0.803	0.807
	$JS = I - JP$	**0.810**	0.806	0.803	0.806

sporadically surpass the baseline; importantly, no single family consistently outperforms across all configurations. As the context size increases (k=10 and k=15), the performance differences exhibit a more structured pattern: select QY variants maintain accuracies that are comparable to, or slightly exceed, the baseline, especially when refinement strategies are carefully combined with forecasting techniques.

Contrasting with the Spambase results, where quantum-enhanced forecasting manifested a distinct advantage, the outcomes on the Ozone dataset underscore a more balanced scenario wherein refinement assumes a pivotal stabilizing role. In this context, QURIOSO benefits less from relying solely on forecasting. Instead, it excels through a careful balance of forecasting and parameter refinement. This approach allows the model to adapt effectively while minimizing the noise caused by changes in the data distribution.

RQ1 – Discussion The results associated with **RQ1** provide a comprehensive overview of the static performance of QURIOSO across datasets with distinctly different characteristics. On the Spambase dataset, the proposed framework, especially when integrated with quantum-enhanced forecasting, consistently outperforms the static VQA baseline. This underscores the effectiveness of forecasting-driven parameter updates within relatively stable and structured data distributions.

In contrast, the Ozone dataset poses a more challenging scenario due to artificial class balancing and heightened variability across folds. In this context, QURIOSO generally achieves performance levels that are comparable to the baseline, with noticeable improvements arising only in specific configurations. These findings suggest that while forecasting remains advantageous, its impact is inherently limited by the noisier optimization landscape created by heterogeneous data distributions.

A consistent trend emerges across both datasets: *QY* tend to demonstrate greater robustness compared to *LS*, particularly as M increases. However, the extent of the improvement is heavily influenced by the interaction between forecasting and refinement. While minimal refinement is typically sufficient for Spambase, Ozone benefits from more carefully balanced update strategies to avoid performance degradation.

Overall, the findings of **RQ1** indicate that QURIOSO is capable of matching or surpassing the static baseline across diverse settings, with its benefits becoming more evident in scenarios where the optimization process can effectively leverage structured temporal signals.

4.3.2 RQ2: How does the moving average accuracy evolve across the data blocks, and how does it change in comparison to the baseline VQA?

To address **RQ2**, we examine the progression of moving average accuracy (*MAA*) throughout the sequential data portions U_h with the aim to experimentally verify whether

the proposed framework sustains or enhances the predictive performance during sequential learning, and to compare the behavior against the static baseline (VQA), specifically focusing on a representative configuration with $k = 5$ and $M = 5$. Detailed findings for additional configurations can be found in the Appendix A. This configuration provides a balanced setting for evaluating both short-term adaptability and constrained temporal aggregation.

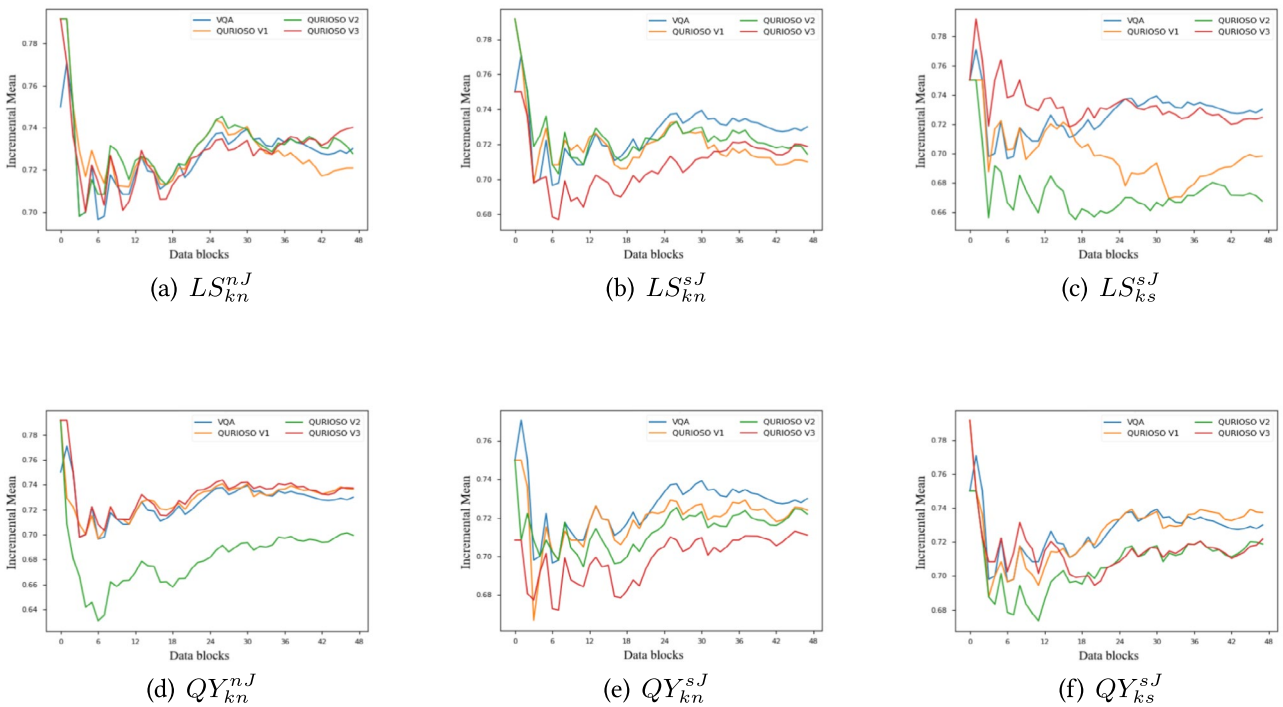
In our analysis of the Spambase dataset (Fig. 4 is accompanied by the legend provided in Table 8), we observe that QURIOSO variants utilizing QY consistently exhibit greater stability and enhanced average performance when contrasted with both the VQA and classical forecasting methods LS. Notably, the QY configurations are capable of exceeding baseline performance without necessitating extensive refinements, thereby indicating an effective leveraging of the forecasting signal to guide parameter adaptation throughout the sequential learning process.

Conversely, on the Ozone dataset (Fig. 5 is accompanied by the legend provided in Table 8), the moving average accuracy tends to remain relatively close to the baseline across most configurations, reflecting the increased variability introduced by artificial class balancing. Nevertheless, several QURIOSO variants still manage to meet or surpass baseline performance in specific regions of the sequence, which corroborates the competitive nature of the proposed framework even under more challenging conditions. Within this context, the significance of refinement is accentuated: whilst certain QY configurations maintain stable performance in the absence of refinement, others demonstrate improved outcomes with the implementation of suitable refinement strategies, thereby mitigating performance degradation across successive folds.

A particularly noteworthy case arises with the QY_{ks}^{sJ} configuration on the Ozone dataset, where both the absence of refinement ($JS = 0$) and partial refinement ($JS = \frac{I}{2} - JP$) result in a gradual decline in performance. However, the application of full refinement ($JS = I - JP$) allows this configuration to regain

Table 8 Summary of the QURIOSO variants used across all moving–average–accuracy plots. Each variant corresponds to a different choice of the sequential coupling term *JS*, while the baseline is given by the static VQA. Colors match those used consistently throughout all figures in the paper

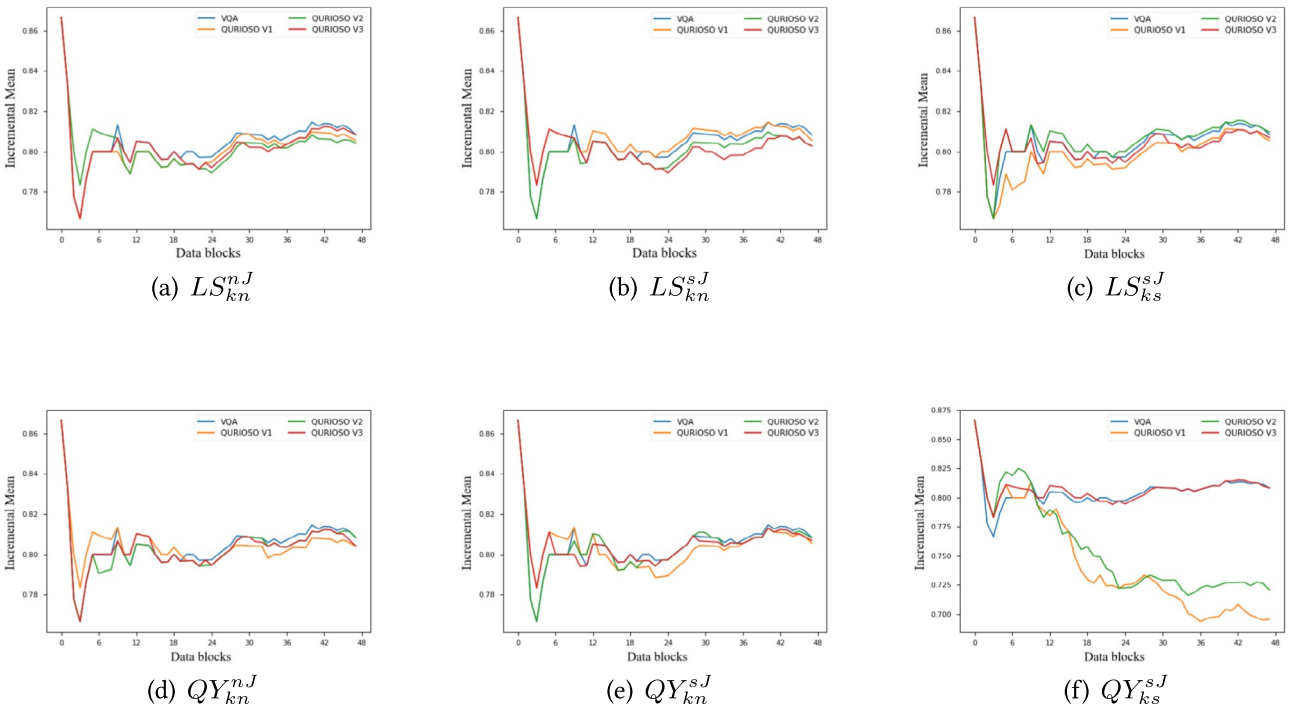
Variant	Color	Description
VQA	Blue	Baseline
QURIOSO V1	Orange	$JS = 0$
QURIOSO V2	Green	$JS = \frac{I}{2} - JP$
QURIOSO V3	Red	$JS = I - JP$



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 4 Moving average accuracy for $k = 5$ and $M = 5$ for spambase dataset. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 5 Moving average accuracy for $k = 5$ and $M = 5$ for the ozone dataset. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy

performance levels commensurate with the baseline, underscoring the stabilizing influence of refinement amid heterogeneous data distributions.

In summary, this comprehensive analysis reveals that QURIOSO effectively maintains predictive performance throughout Continual Learning across a variety of data characteristics. While the Spambase dataset underscores the intrinsic benefits of quantum-enhanced forecasting, the Ozone dataset highlights the critical need to balance forecasting efforts with appropriate refinement strategies to uphold stability. Collectively, these findings affirm the robustness and adaptability of the proposed framework in diverse Continual Learning scenarios.

4.3.3 RQ3: How does the execution time of QURIOSO evolve across data blocks, and how does it compare to the baseline?

To address **RQ3**, we conduct an analysis of the execution time associated with the QURIOSO framework in comparison to the baseline VQA model, with particular focus on the evolution of runtime across sequential data blocks. The baseline VQA model performs a comprehensive adaptation at each data block, resulting in consistently elevated execution times throughout the sequence.

In contrast, the QURIOSO framework employs the forecasting model F, which is trained exclusively during the initial phase. This trained model is subsequently utilized to direct the parameter updates of the VQA as new data blocks are introduced. This architectural design significantly alleviates the computational burden that typically arises from repeated full optimizations. Following an initial peak in training time, all QURIOSO variants are observed to exhibit systematically reduced execution times relative to the baseline, thereby underscoring the effectiveness of the proposed methodology in mitigating unnecessary computations during the adaptation process.

Notably, an exception is seen in configurations where $JS = I - JP$; in these cases, the inclusion of an additional refinement phase escalates the computational demands, resulting in execution times that may occasionally match or exceed those of the baseline. This behavior is anticipated due to the extra optimization steps necessitated by the extended refinement of the VQA.

When comparing the various QURIOSO variants, it is evident that models leveraging classical LSTM architectures consistently achieve lower execution times than their quantum counterparts. This disparity is indicative of the additional overhead entailed in simulating quantum circuits on classical hardware, encompassing state-vector

manipulations and unitary transformations. Nevertheless, it is important to note that classical models also impose substantial computational resource requirements, including GPU acceleration for both LSTM training and inference, rendering them not entirely cost-free.

Furthermore, we observe minor fluctuations in execution times across data blocks within the Ozone dataset, particularly where artificial class balancing is implemented. The resultant heterogeneity across these blocks can induce variability in the optimization process, even with a fixed number of optimization iterations. Such variability is expected in sequential contexts and does not undermine the overarching trend, which consistently favors QURIOSO in terms of computational efficiency.

In summary, the QURIOSO framework demonstrates a more efficient runtime profile than the baseline VQA, particularly when JS is selected closely to the predictive index. Classical forecasting models provide lightweight and stable performance; however, quantum-enhanced variants incur a moderate additional cost that is offset by increased modeling flexibility and expressiveness.

All execution running times plots are presented in Appendix B.

4.3.4 RQ4: What is the relationship between PQC depth and parameterization?

To address RQ4, we analyze how varying the depth of the PQC impacts performance while keeping the circuit architecture fixed. In QURIOSO, the PQC consists of a parametric block with a consistent encoding and entangling pattern, repeated along the depth dimension. The repetition factor, denoted as "reps," controls the circuit depth and thus the number of trainable parameters, without changing the underlying architecture.

For this analysis, we focus on a representative configuration with $k = 5$ and $M = 3$, progressively reducing the circuit depth from three to two and one repetition. This controlled approach allows us to isolate the effects of parameterization while maintaining consistency across the other components of the pipeline.

The results for the Spambase and Ozone datasets are presented in Tables 9 and 10, respectively. Across both datasets, increasing the number of repetitions generally leads to improved performance for the baseline VQA and several QURIOSO variants. This suggests that a higher expressive capacity can be advantageous. The trend is particularly noticeable when moving from one to two repetitions, whereas the performance gains from additional depth tend to be more moderate.

Table 9 Accuracy obtained by varying the number of repetitions (reps) of the PQC, which controls circuit depth and the number of trainable parameters while preserving the same PQC architecture on the Spambase dataset. The notation of LS and QY variants is consistent with that used in the moving average accuracy plots

Configuration		1 rep	2 reps	3 reps
Number of parameters		16	24	32
VQA		0.706	0.727	0.730
LS_{ks}^{sJ}	$JS = 0$	0.578	0.710	0.700
	$JS = \frac{I}{2} - JP$	0.607	0.695	0.655
	$JS = I - JP$	0.659	0.727	0.721
LS_{kn}^{sJ}	$JS = 0$	0.686	**0.728**	**0.745**
	$JS = \frac{I}{2} - JP$	**0.714**	0.725	0.730
	$JS = I - JP$	0.691	0.726	0.723
LS_{kn}^{nJ}	$JS = 0$	**0.710**	0.726	0.702
	$JS = \frac{I}{2} - JP$	**0.719**	0.713	0.725
	$JS = I - JP$	**0.716**	0.722	0.722
QY_{ks}^{sJ}	$JS = 0$	0.368	0.717	**0.774**
	$JS = \frac{I}{2} - JP$	0.700	0.693	0.698
	$JS = I - JP$	**0.707**	0.717	0.700
QY_{kn}^{sJ}	$JS = 0$	0.649	0.692	0.721
	$JS = \frac{I}{2} - JP$	0.699	0.707	0.711
	$JS = I - JP$	**0.718**	0.724	0.707
QY_{kn}^{nJ}	$JS = 0$	**0.718**	0.725	0.709
	$JS = \frac{I}{2} - JP$	**0.733**	0.725	0.730
	$JS = I - JP$	**0.715**	0.725	**0.735**

Table 10 Accuracy obtained by varying the number of repetitions (reps) of the PQC, which controls circuit depth and the number of trainable parameters while preserving the same PQC architecture on the Ozone dataset. The notation of *LS* and *QY* variants is consistent with that used in the moving average accuracy plots

Configuration		1 rep	2 reps	3 reps
Number of parameters		16	24	32
VQA		0.554	0.806	0.808
LS_{ks}^{sJ}	$JS = 0$	0.512	0.788	0.792
	$JS = \frac{I}{2} - JP$	0.531	0.783	0.799
	$JS = I - JP$	**0.561**	0.806	0.803
LS_{kn}^{sJ}	$JS = 0$	0.532	**0.808**	0.803
	$JS = \frac{I}{2} - JP$	0.537	0.806	0.796
	$JS = I - JP$	**0.567**	0.803	0.806
LS_{kn}^{nJ}	$JS = 0$	**0.565**	**0.807**	0.797
	$JS = \frac{I}{2} - JP$	0.531	**0.807**	0.796
	$JS = I - JP$	0.547	0.806	0.807
QY_{ks}^{sJ}	$JS = 0$	0.492	**0.810**	0.804
	$JS = \frac{I}{2} - JP$	0.504	0.806	0.792
	$JS = I - JP$	0.550	**0.807**	0.807
QY_{kn}^{sJ}	$JS = 0$	0.472	0.803	**0.808**
	$JS = \frac{I}{2} - JP$	0.512	**0.811**	0.807
	$JS = I - JP$	0.531	0.806	0.808
QY_{kn}^{nJ}	$JS = 0$	0.517	**0.811**	0.793
	$JS = \frac{I}{2} - JP$	0.526	0.794	0.806
	$JS = I - JP$	0.535	**0.808**	0.800

Moreover, increasing the circuit depth expands the dimensionality of the parameter space explored by the optimizer within a fixed optimization budget. This leads to richer and more diverse parameter trajectories. Since QURIOSO trains its forecasting component on sequences derived from these trajectories, deeper circuits typically provide more parameters to predict, resulting in a greater number of forecasting models. In contrast, shallower circuits reduce the number of trainable parameters and may produce less diverse trajectories, limiting the availability of informative sequential data for training the prediction phase. This effect may lead to more stable optimization behavior, even as the overall diversity of the trajectories diminishes.

The observed variability between the *LS* and *QY* variants and their sensitivity to the chosen refinement strategy further reflects these dynamics. Overall, our results confirm that performance differences arise from changes in parameterization and optimization behavior induced by circuit depth, rather than modifications to the PQC architecture itself. This analysis supports the design decision in QURIOSO to consider circuit depth as a tunable parameter that influences expressivity and the richness of the forecasting signal while maintaining a fixed architectural template.

5 Related Work

The widespread use of the VQAs, especially on NISQ devices, passes also through the reduction of the computational cost of the refinement of the parameters. Many efforts have been done through the integration of classical computing techniques in the VQA loop. Ravi et al. (2022) proposes a classical simulation approach to bootstrap the parameters. The estimations are determined through an efficient classical search over the Clifford gate parameter space. Although Clifford gates are not universal and cannot fully explore quantum state space, they can still be used to generate high-quality, noise-free initial states, which is especially valuable in the NISQ era. Numerous studies in the literature have instead explored the applicability of machine learning algorithms. These algorithms demonstrate the ability to estimate parameters and uncover latent correlations within the data. A distinguishing factor of such methods is their inherent robustness to noise and error. For example, in Sauvage et al. (2021), the authors introduce a meta-learning approach that utilizes a neural network-based model trained on families of PQC problems. This strategy is designed to generalize across related PQC tasks, enabling the initialization of parameters for new PQC instances. Its flexibility lies in the

ability to make predictions for PQCs that differ from the training set in terms of qubit count, circuit depth, and number of parameters. This is made possible through a model-agnostic PQC encoding-decoding scheme that can represent parameter values, circuit structure, and optimization objectives. The PQC families are generated either by varying circuit sizes with a fixed cost function, or by modifying the cost function while keeping the ansatz fixed. The goal is to identify and exploit recurring patterns in the parameter space, particularly in the context of QAOA applied to max-cut problems.

Similarly, in Moussa et al. (2022), the authors exploit the concentration effect, that is, the phenomenon where optimal parameters from one QAOA instance have good generalization properties. They adopt an unsupervised learning approach involving clustering, using as input the angle values from training QAOA problems, their graph-based features, and representations derived from a graph autoencoder. Clusters are defined with fixed cardinality, and their centroids are used as initialization indications for solving new QAOA instances. The results show that machine learning techniques can effectively predict suitable QAOA parameters, significantly reducing the number of required optimizer evaluations while incurring only minor degradation in the approximation ratio. These outcomes are comparable to those obtained via exhaustive angle optimization, offering a substantial reduction in the overall number of circuit evaluations. Jain et al. (2022) proposes Graph Neural Networks (GNNs) as a warm-starting method, showing that combining GNNs with QAOA leads to better performance than using either approach alone. Notably, GNNs enable generalization across different graph instances and sizes, offering an advantage over traditional warm-start techniques.

Another perspective of the problem is provided by methods designed to determine model parameters from instances of models, which is the view of meta-learning (Vanschoren 2019). Meta-learning frameworks operate at a higher level of abstraction and returns learners trained by leveraging the experience from multiple learning tasks. Meta-learners have been studied also in the paradigm of lifelong learning. Son et al. (2025) present a comprehensive work that categorizes the relationship between meta-learning and continual learning along several learning frameworks. The meta-learner sets initial parameters to enhance the optimization process, applicable to both meta-online and meta-continual learning. Additionally, sequential Bayesian updates use the Bayesian rule to update beliefs based on new data and prior knowledge. Sequence modeling uses recurrent or autoregressive models to learn from a continuous flow of information, encoding the learning trajectory in evolving hidden representations. Additionally, meta-learning is being used to refine the optimization process itself, employing architectures like LSTMs to create adaptive optimizers that utilize past optimization trajectories. These models are more resilient to noise and improve generalization to new tasks (Wilson et al. 2021; Wang et al. 2021). While there is a consolidated research of meta-learning in classical computing, we could not register the same on quantum solutions. Sauvage et al. (2021) is probably the notable attempt, which however cannot work on the lifelong learning that has been studied in classical computing, but there are not quantum counterparts yet.

6 Conclusions

We investigated the problem of efficient parameter tuning for VQAs operating in a continuous computation (lifelong learning) setting, where data arrive incrementally and data distributions may evolve over time. In such non-stationary scenarios, repeatedly re-optimizing PQC parameters by means of batch methods is computationally expensive and may show inefficiency and sub-optimality as the scenario changes.

To address this challenge we proposed QURIOSO, a hybrid machine-learning framework that leverages sequential modeling to forecast PQC parameter trajectories and thus reduce the cost of iterative optimization procedure. QURIOSO supports two practical modes of operating. First, it applies predicted parameters directly to incoming data, and second, it refines predictions with a short optimization step. The forecasting model is trained on sequences of optimized parameter sub-trajectories (extracted from the data blocks previously processed) and can be implemented with classical LSTMs or a quantum-enhanced QLSTM variant that uses VQCs to generate gate activations. This design detaches long-term forecasting from frequently executed and costly optimization, enabling prompt processing of new data blocks and reducing iterative optimizer workload.

Experimental evaluation on a binary classification task provides arguments in favor of QURIOSO across three research questions. First, QURIOSO matches or surpasses the baseline VQA accuracy across a variety of configurations. Quantum-enhanced variants often outperform the baseline. The results also highlight the role of sequence input length. Second, the design of the forecasting model is decisive. In fact, configurations using a QLSTM forecasting model consistently capture both short and long-range dependencies, while preserving stable mean average accuracy. Third, QURIOSO generally exhibits lower execution times than the baseline after the initial training peak, since the forecasting model is trained once and VQA parameters are incrementally updated. The main runtime exception occurs in the refinement step where complex internal-state updates can increase post-update cost. While classical LSTM models train faster than quantum-enhanced variants under quantum simulation, this observation reflects the overhead of simulated quantum execution rather than a definitive comparison on real quantum hardware. Nevertheless, the expressivity and adaptability of quantum-enhanced

models frequently compensate in terms of predictive performance. A thorough assessment of runtime behavior on real quantum hardware is left to future work, as it depends on device-specific characteristics and noise properties.

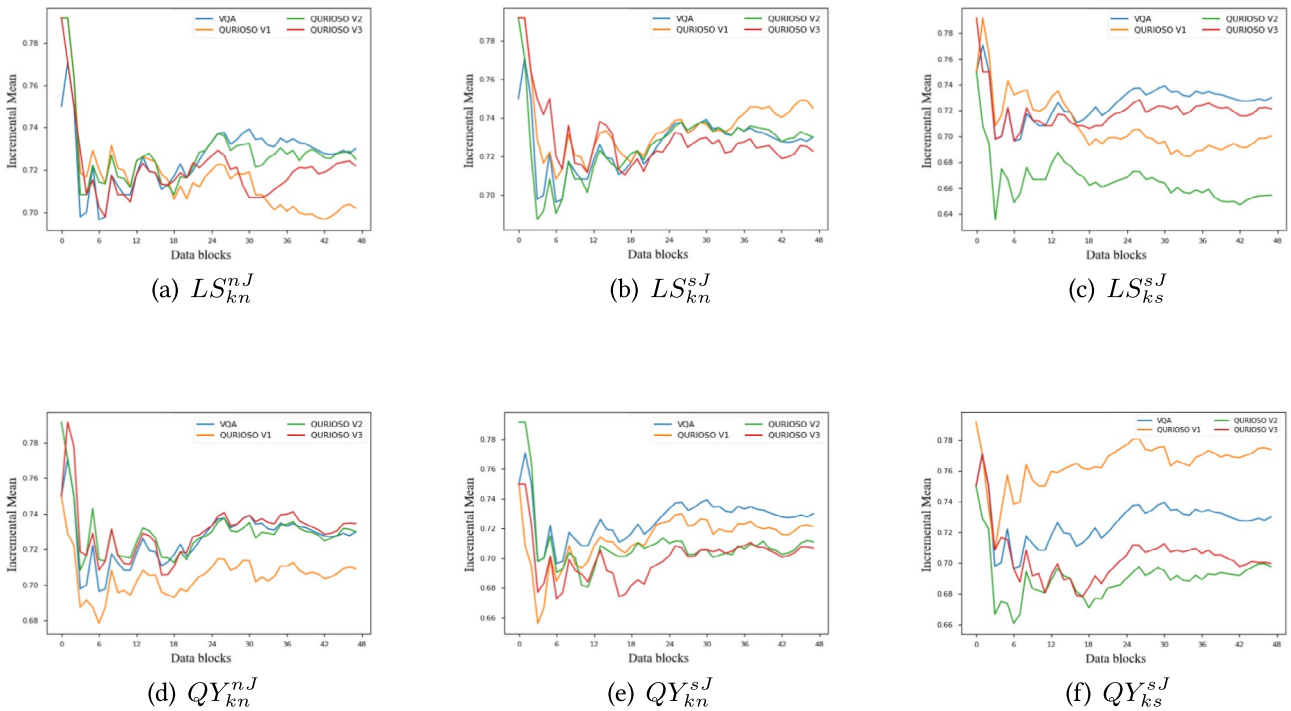
An additional conceptual contribution of this work is the demonstration of a bi-directional Artificial intelligence–Quantum computing loop. On the one hand, sequence modeling supports quantum computing by forecasting PQC parameters and thus accelerating VQA optimization. On the other hand, quantum-enhanced components, both quantum LSTM versions and quantum data embedding, feed back into the machine learning pipeline, improving the task performance. This mutual reinforcement highlights a productive synergy: AI can make quantum algorithms more practical in non-stationary environments, while quantum-enhanced modules can enrich AI models with novel and expressive representations.

We acknowledge several limitations. Not all configurations of QURIOSO guarantee improvements, motivating a study of multivariate parameter interactions and robustness. Moreover, the computational cost of the proposed approach is sensitive to the number of forecasting models required: if each PQC parameter trajectory is learned by an independent forecasting model, overall training and inference cost may grow linearly (or worse) with the number of considered parameters. This scaling can increase memory usage and runtime, particularly for large PQCs or high-dimensional parameter spaces. Finally, the runtime measurements reported in this work were obtained on simulators, noise was not modeled. Therefore, the observed execution-time gains should be interpreted as indicative of orders of magnitude rather than exact time differences on real hardware, where compilation, parallel execution, and noise can alter performances.

To mitigate this issue, future work will investigate strategies such as forecasting multiple parameters jointly or multi-output forecasting algorithms, and model compression techniques. Scaling to larger PQC designs, eventually with HPC infrastructures, and deployment on real NISQ devices remain our objectives.

Appendix A Detailed Moving Average Accuracy Analysis

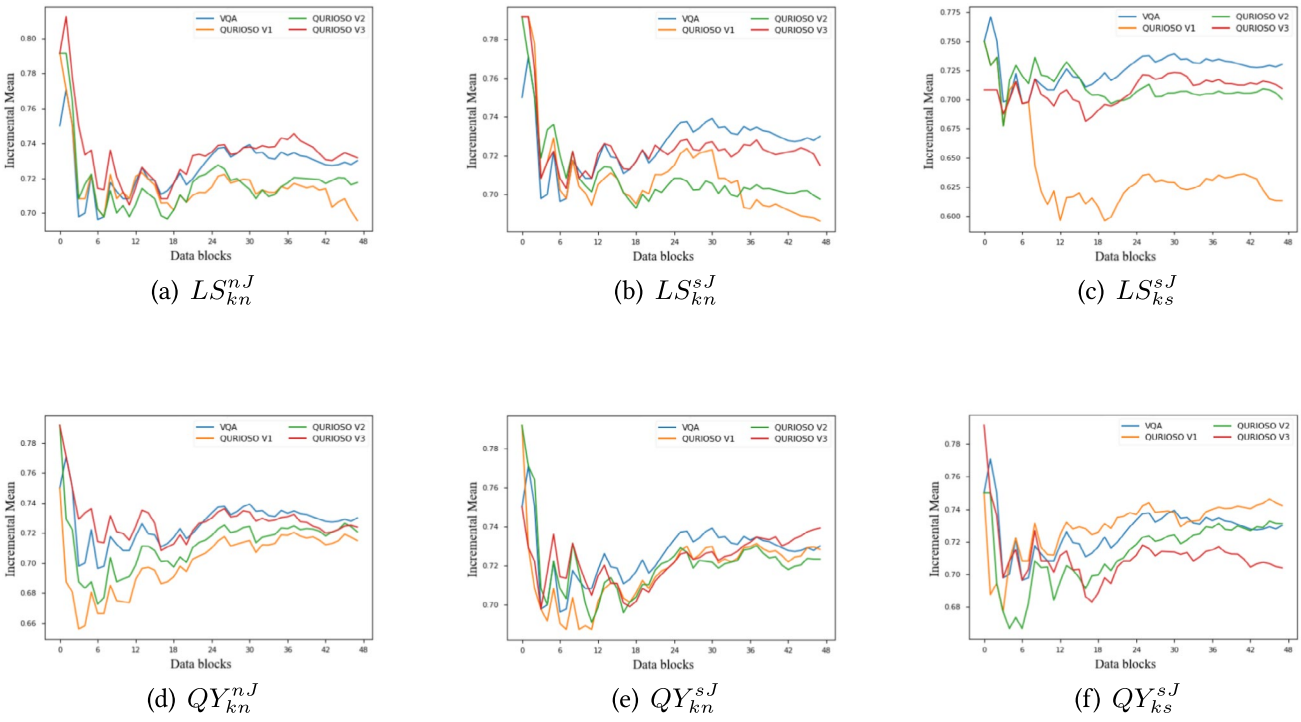
A.1. Spambase



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

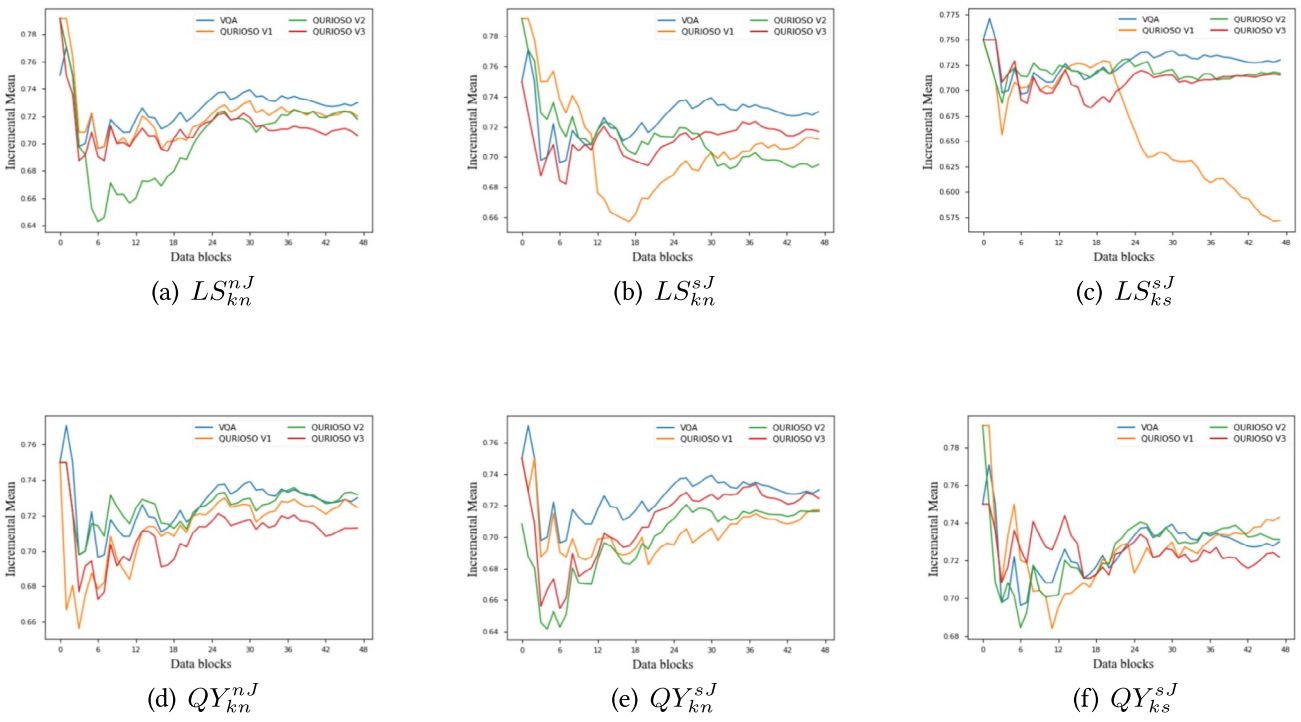
Fig. 6 Moving average accuracy for $k = 5$ and $M = 3$. MAA for confiThe blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

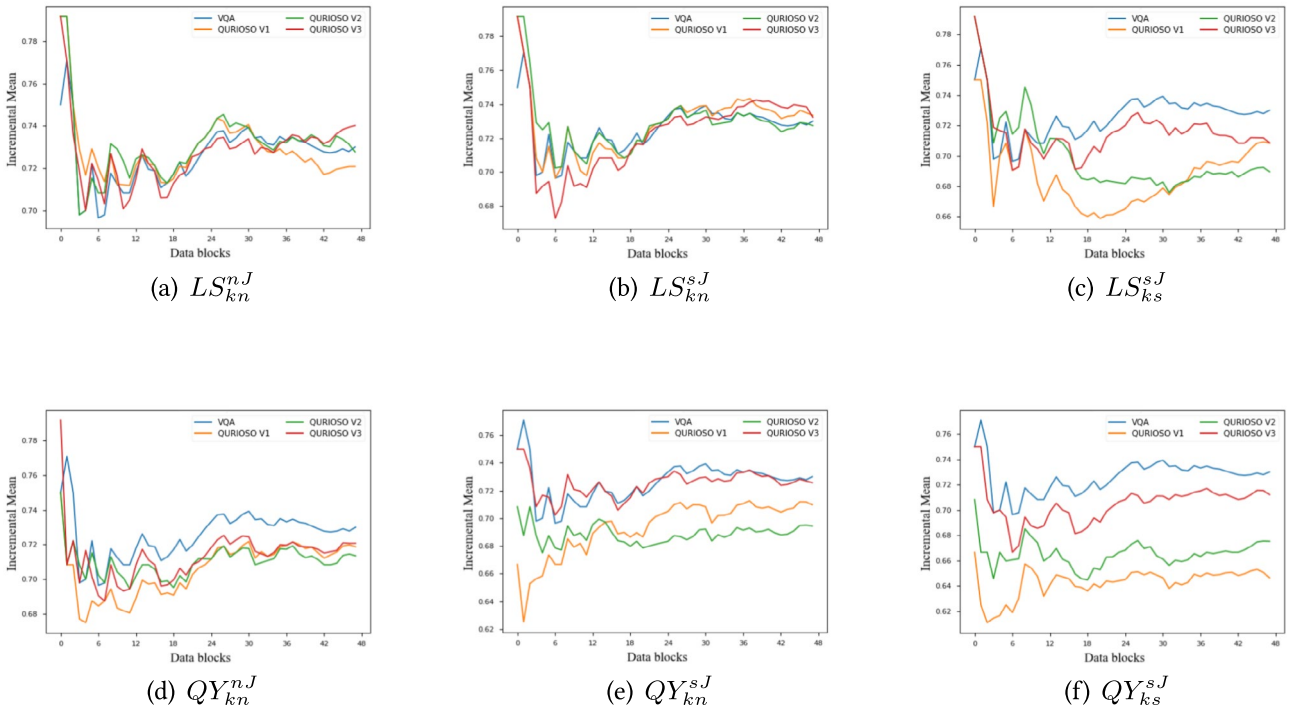
Fig. 7 Moving average accuracy for $k = 5$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

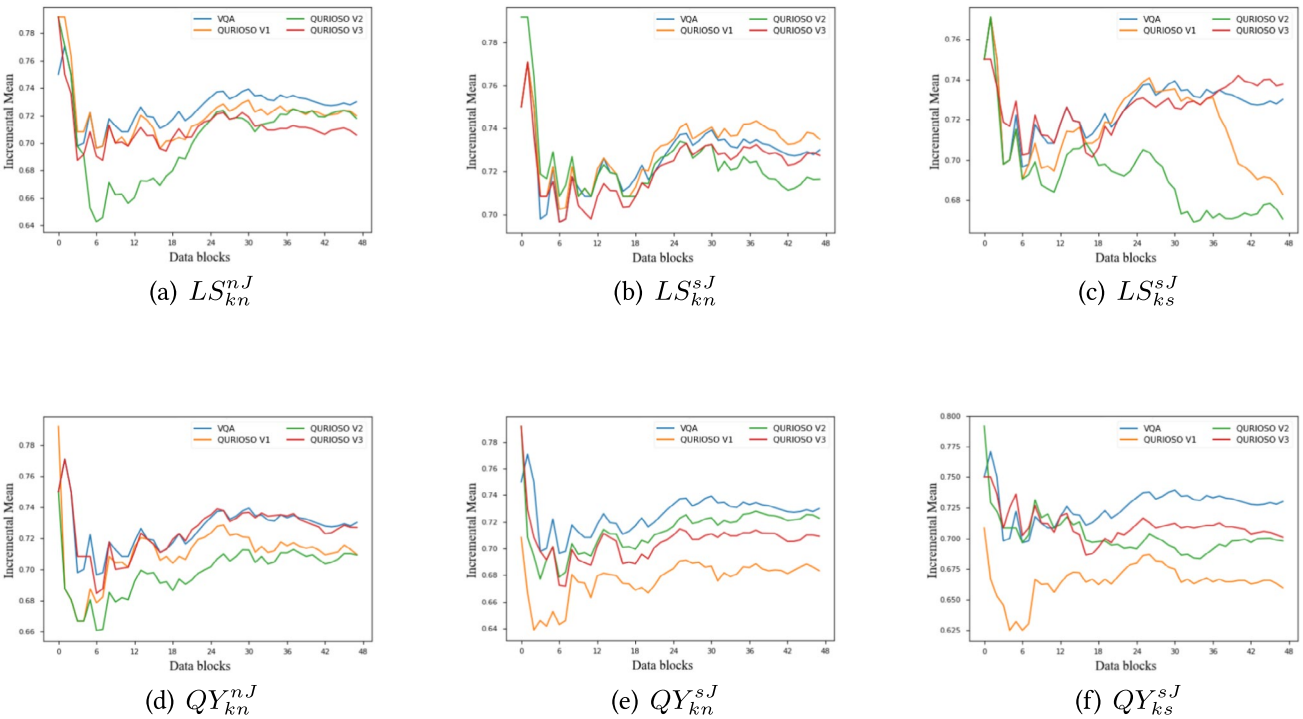
Fig. 8 Moving average accuracy for $k = 5$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

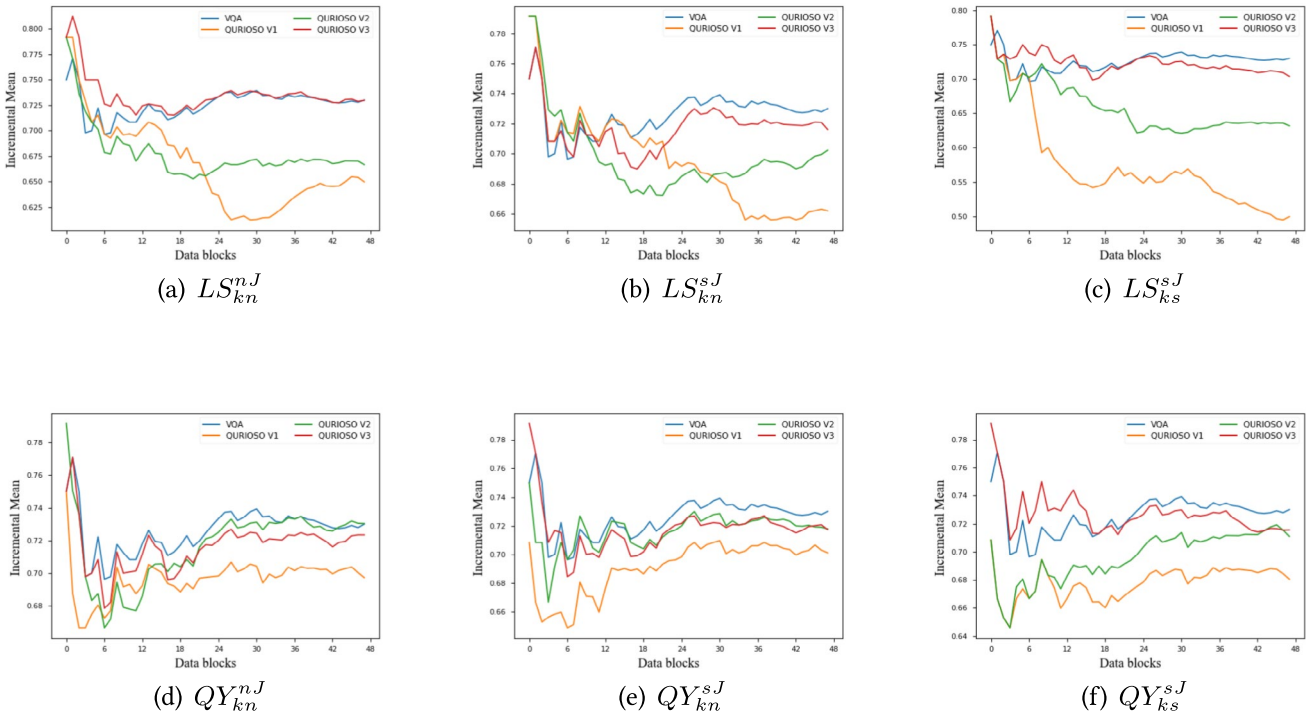
Fig. 9 Moving average accuracy for $k = 10$ and $M = 5$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

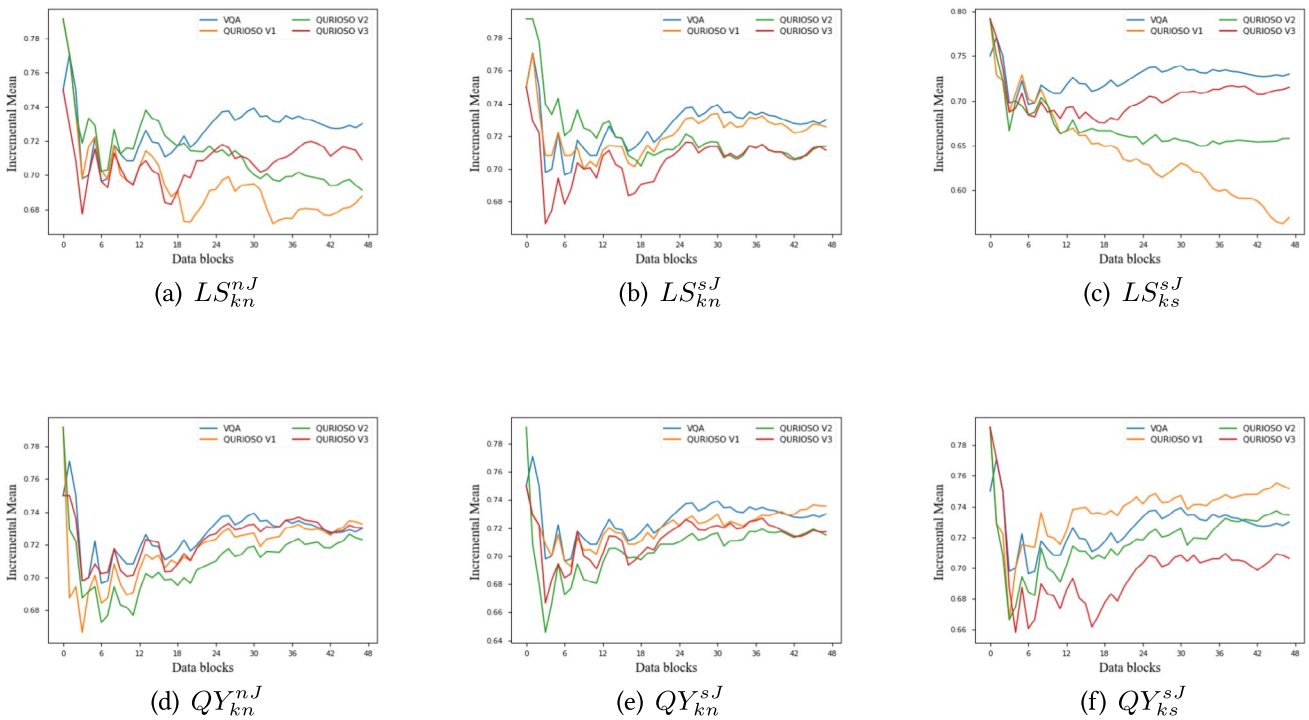
Fig. 10 Moving average accuracy for $k = 10$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

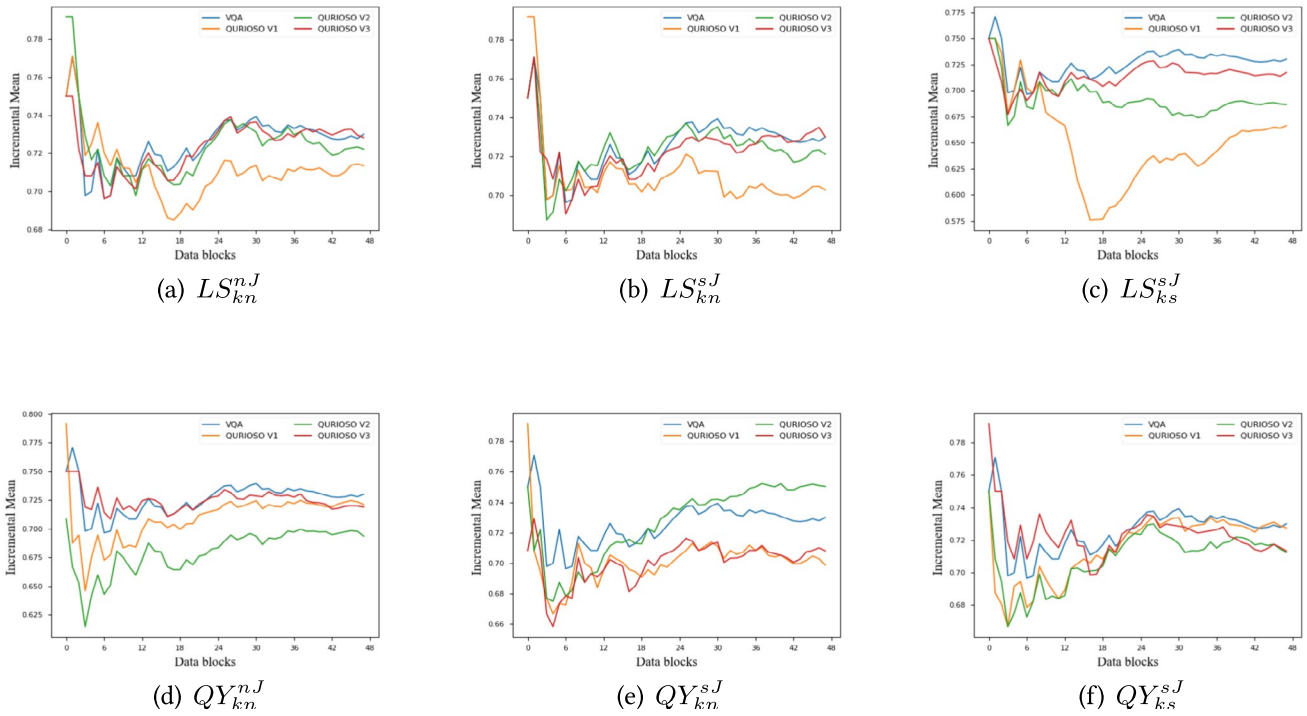
Fig. 11 Moving average accuracy for $k = 10$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

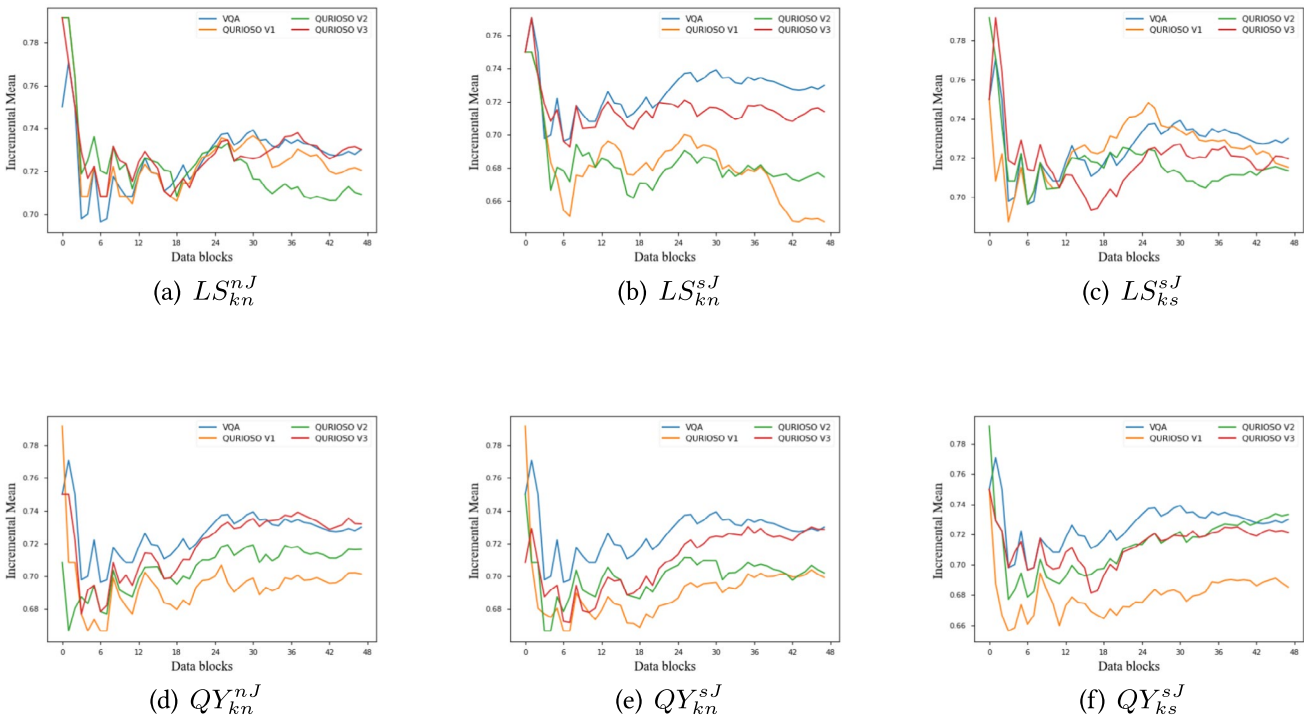
Fig. 12 Moving average accuracy for $k = 10$ and $M = 20$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

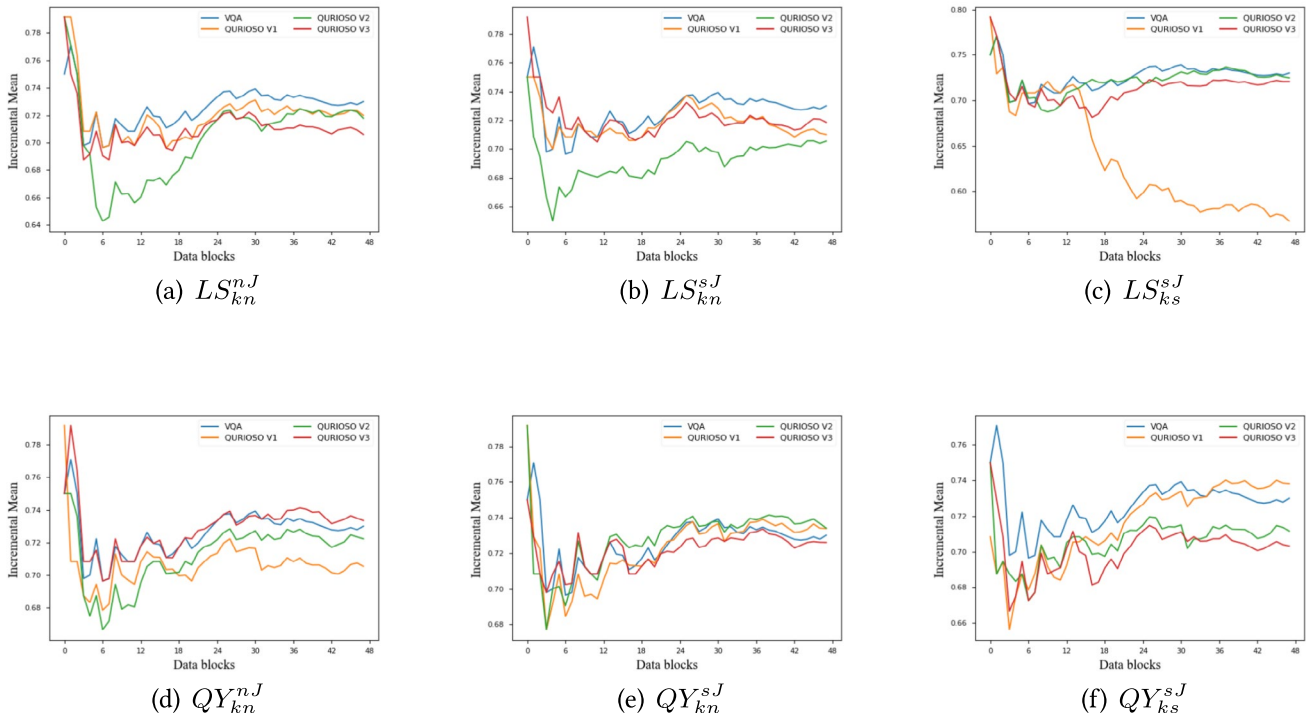
Fig. 13 Moving average accuracy for $k = 15$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

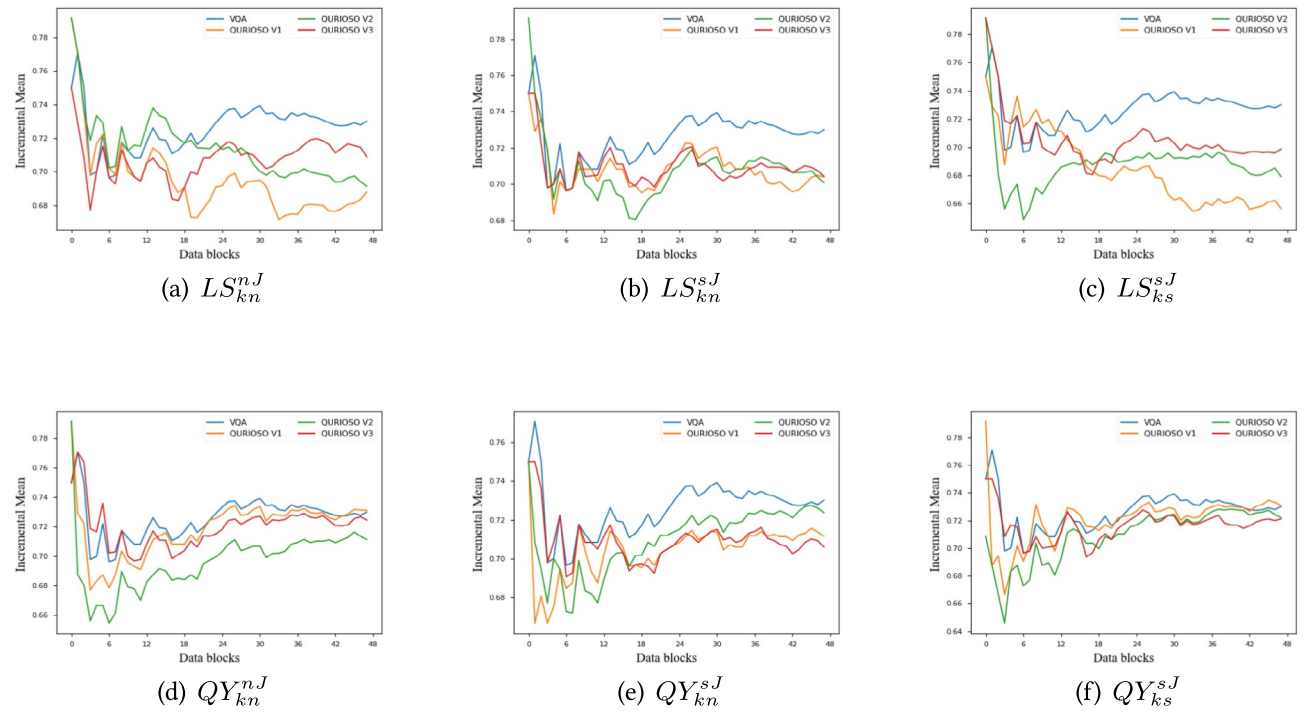
Fig. 14 Moving average accuracy for $k = 15$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 15 Moving average accuracy for $k = 15$ and $M = 23$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy

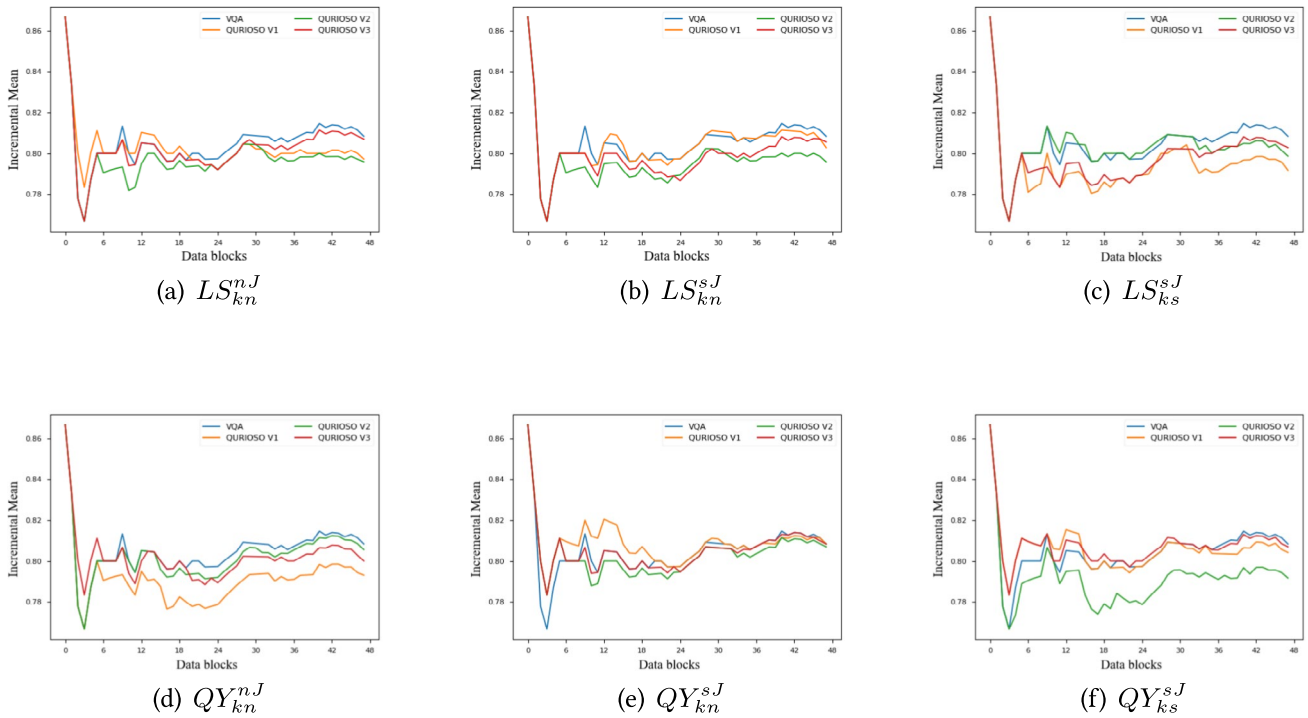


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 16 Moving average accuracy for $k = 15$ and $M = 30$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy

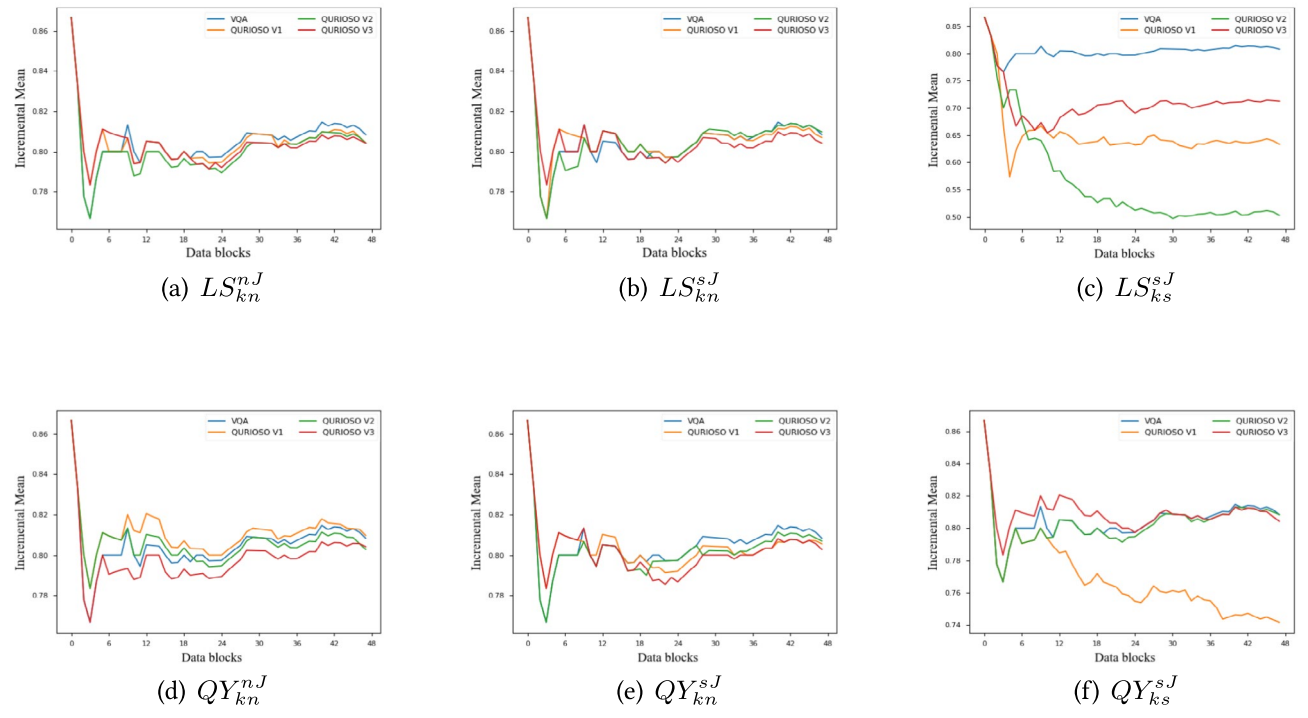
A.2. Ozone



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

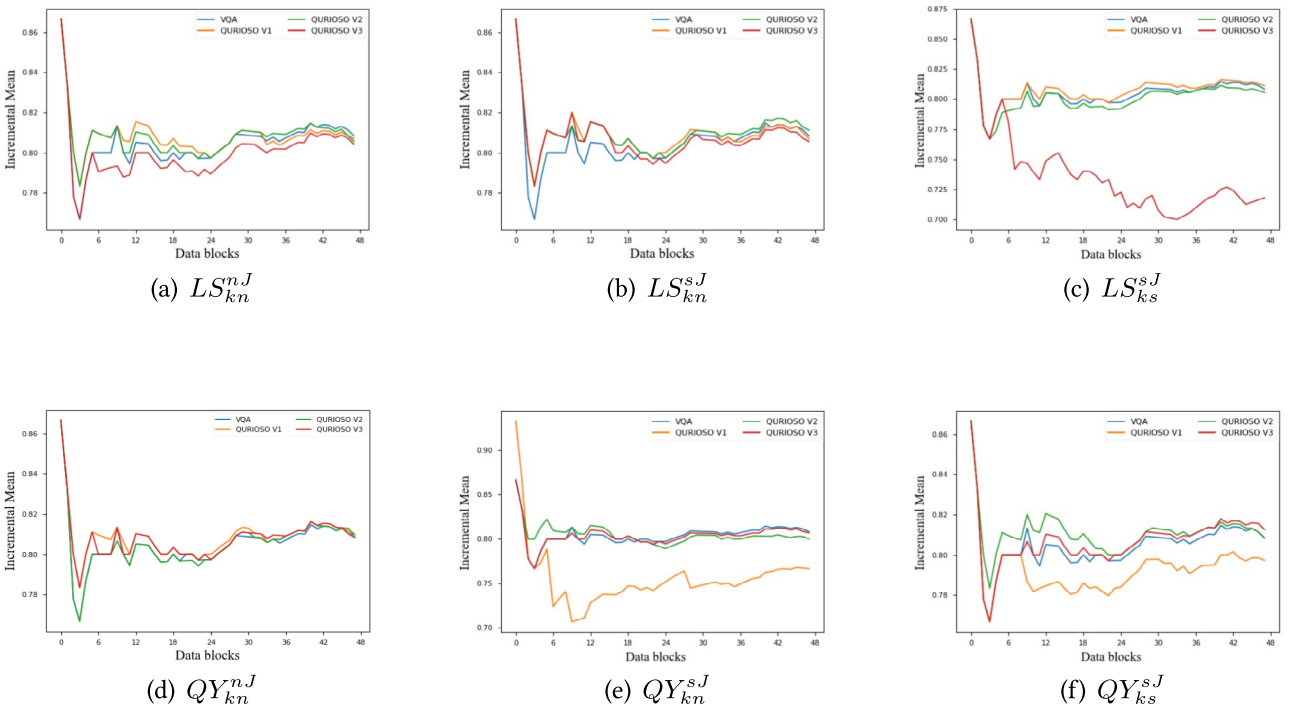
Fig. 17 Moving average accuracy for $k = 5$ and $M = 3$. MAA for confiThe blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

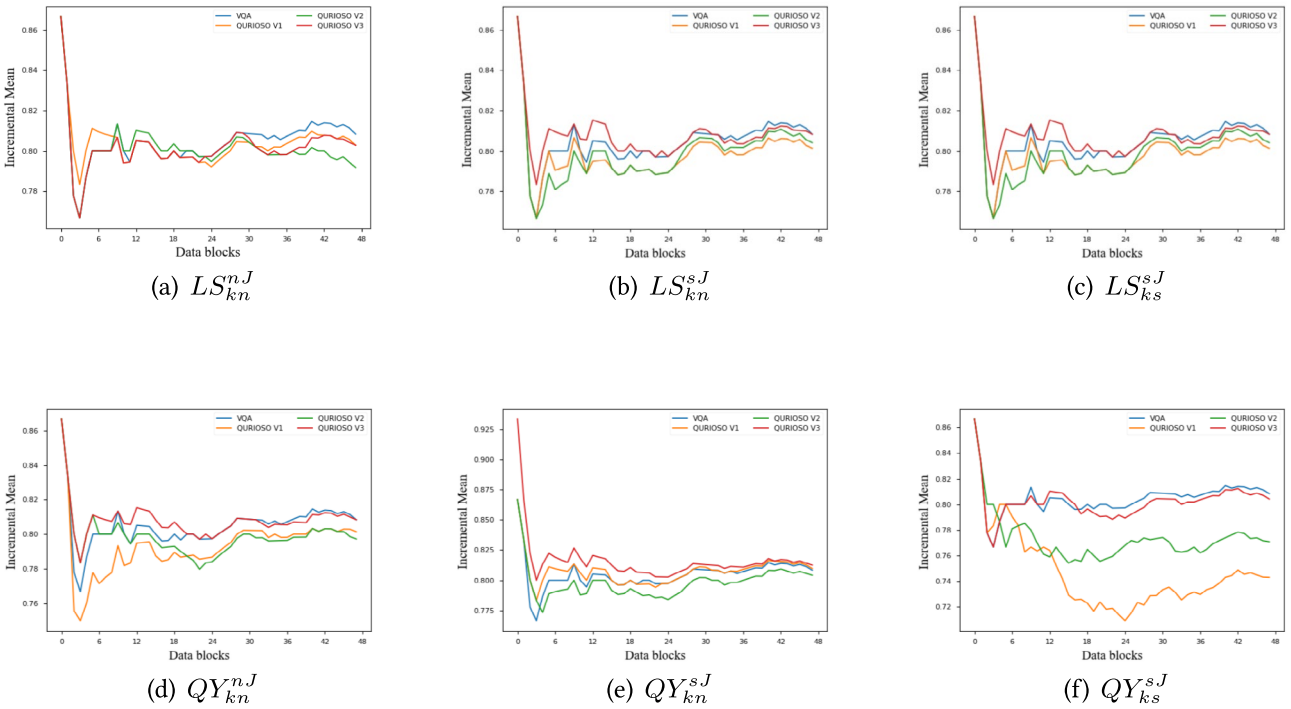
Fig. 18 Moving average accuracy for $k = 5$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

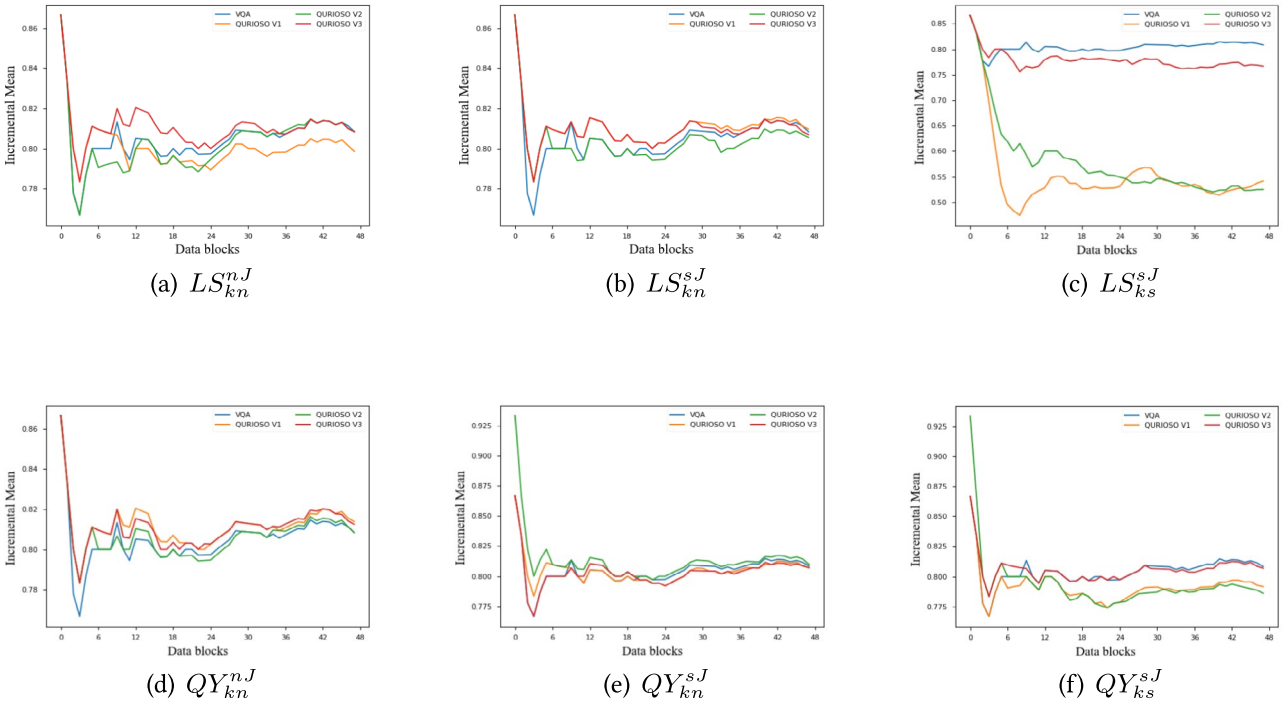
Fig. 19 Moving average accuracy for $k = 5$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

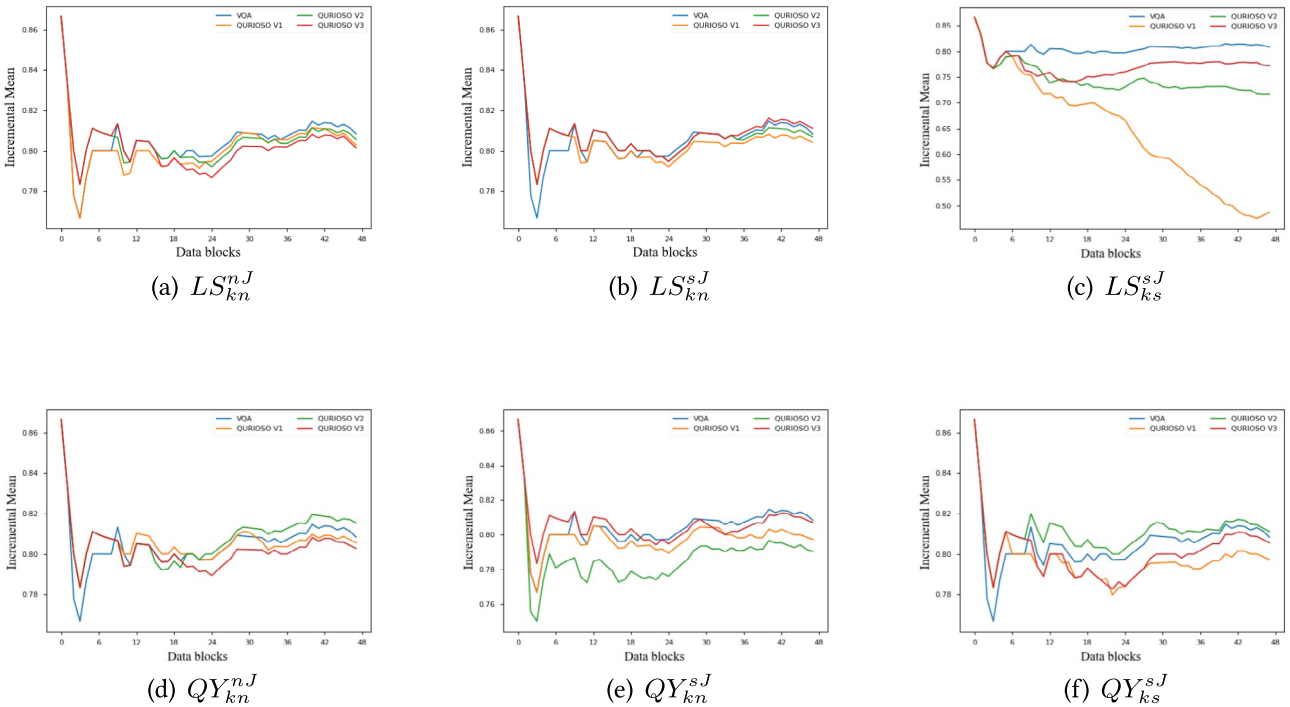
Fig. 20 Moving average accuracy for $k = 10$ and $M = 5$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

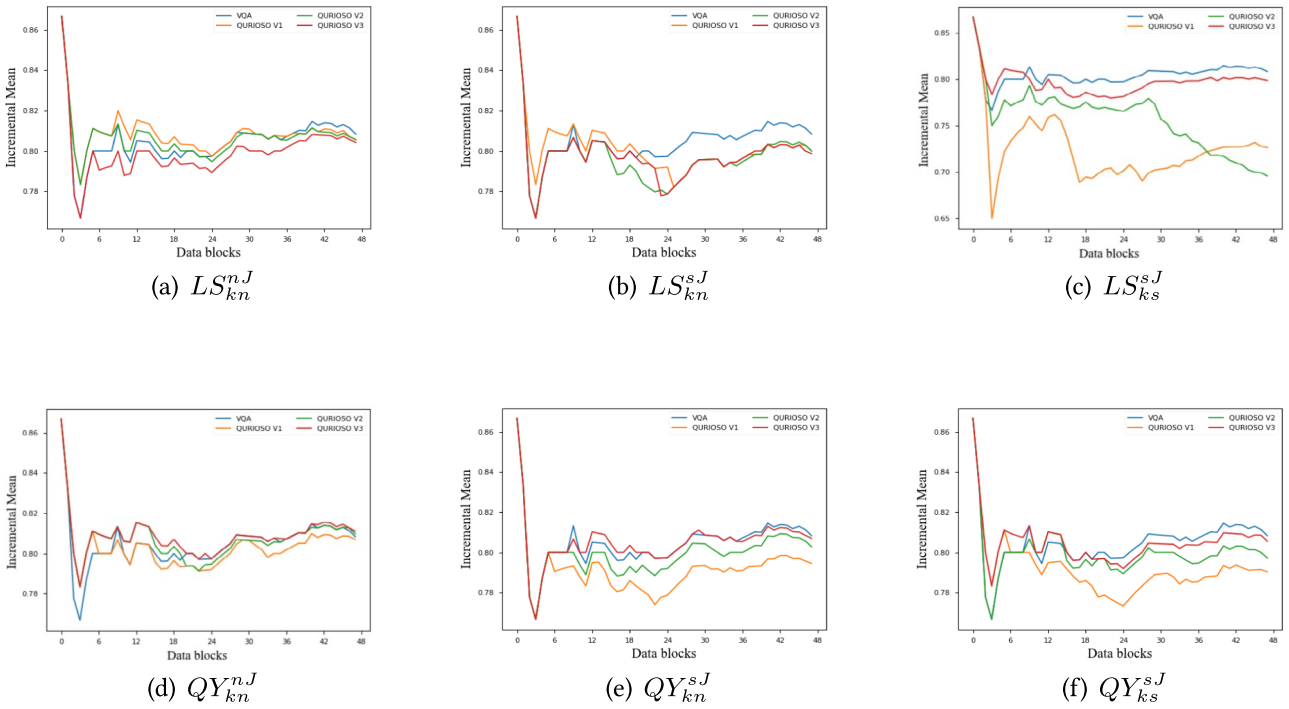
Fig. 21 Moving average accuracy for $k = 10$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

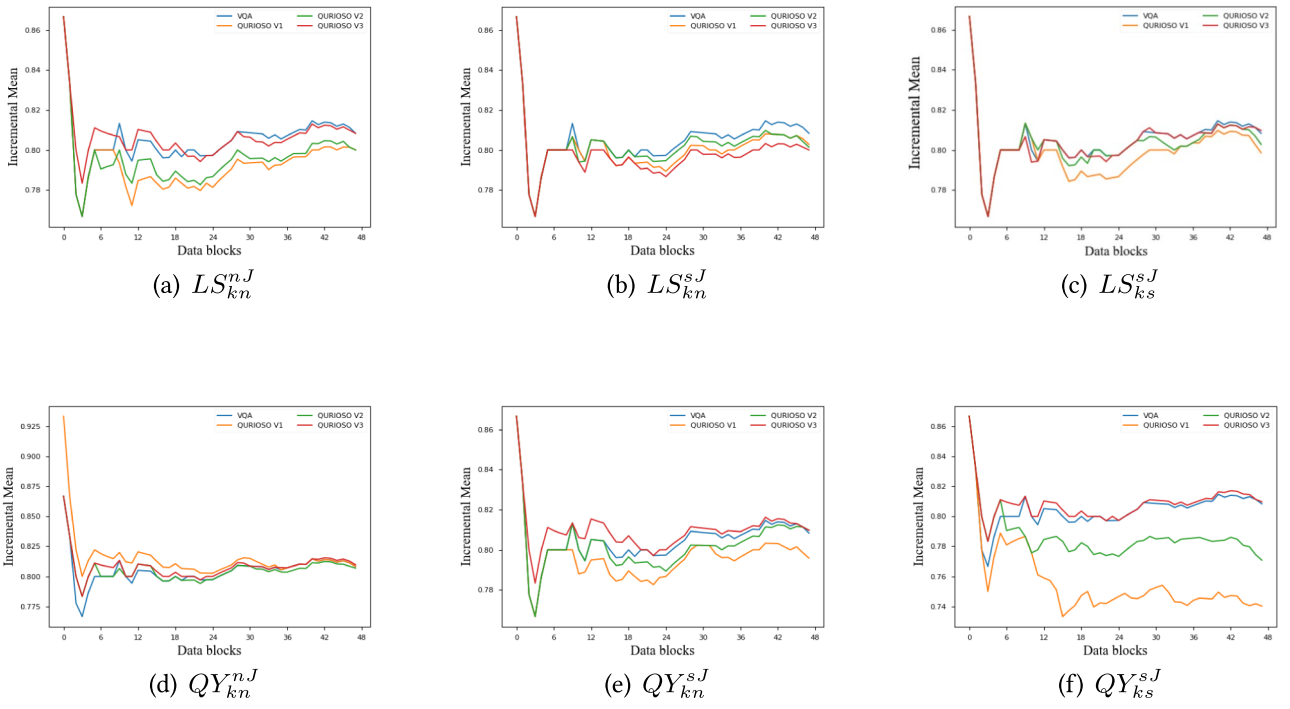
Fig. 22 Moving average accuracy for $k = 10$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

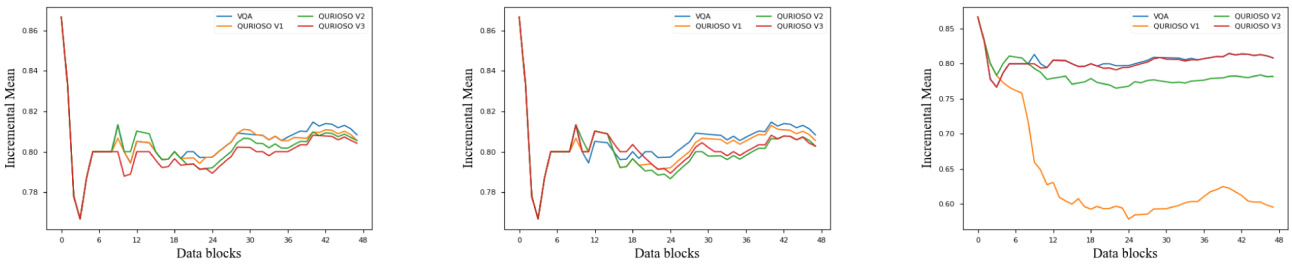
Fig. 23 Moving average accuracy for $k = 10$ and $M = 20$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



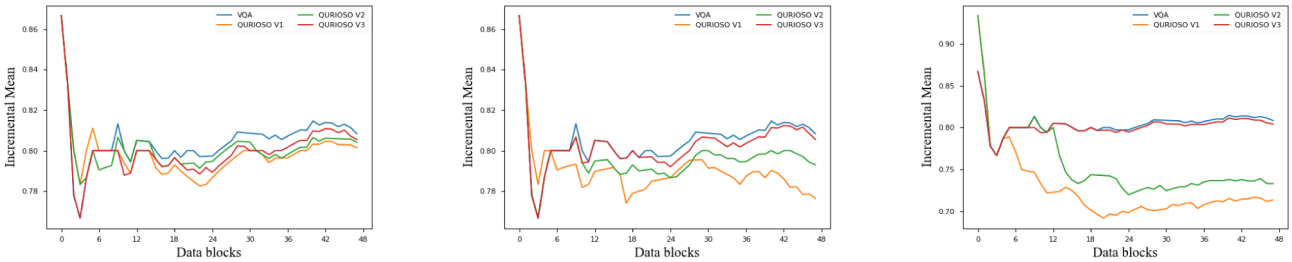
(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 24 Moving average accuracy for $k = 15$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy

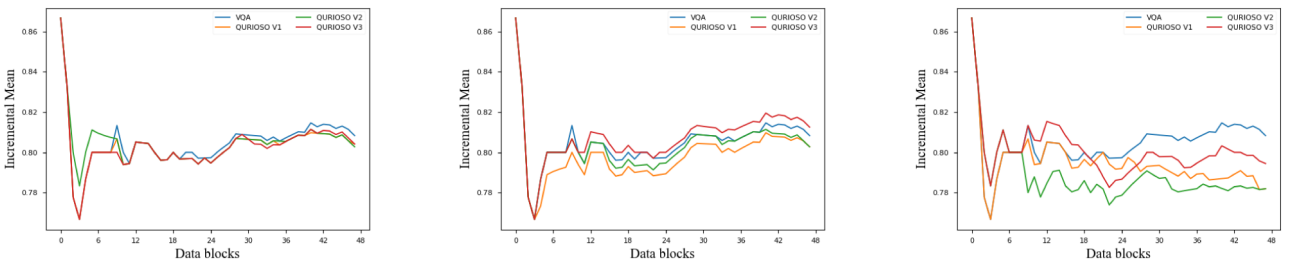


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

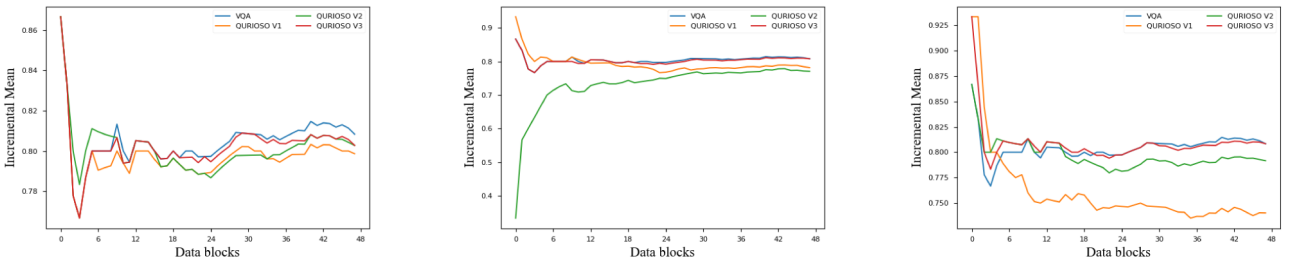


(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 25 Moving average accuracy for $k = 15$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy

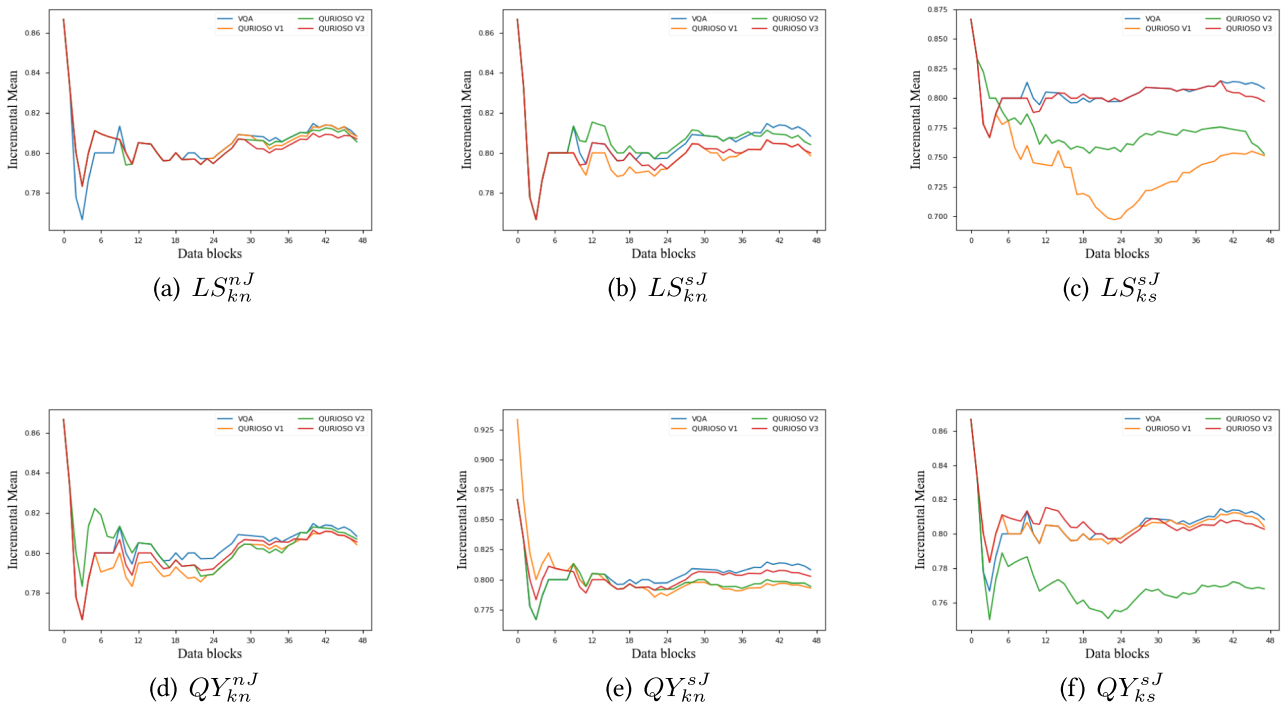


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}



(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 26 Moving average accuracy for $k = 15$ and $M = 23$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy



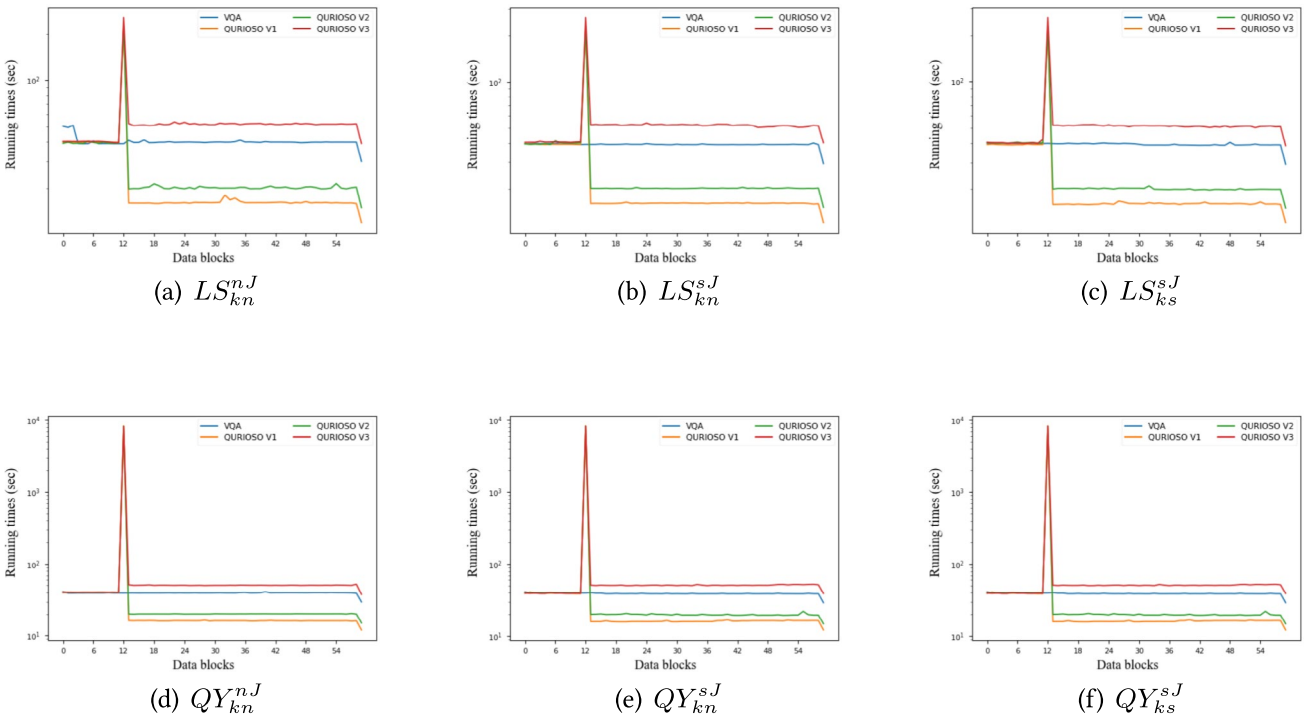
(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 27 Moving average accuracy for $k = 15$ and $M = 30$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The x-axis reports the sequence of incoming data blocks and the y-axis the moving average accuracy

Appendix B Running Times

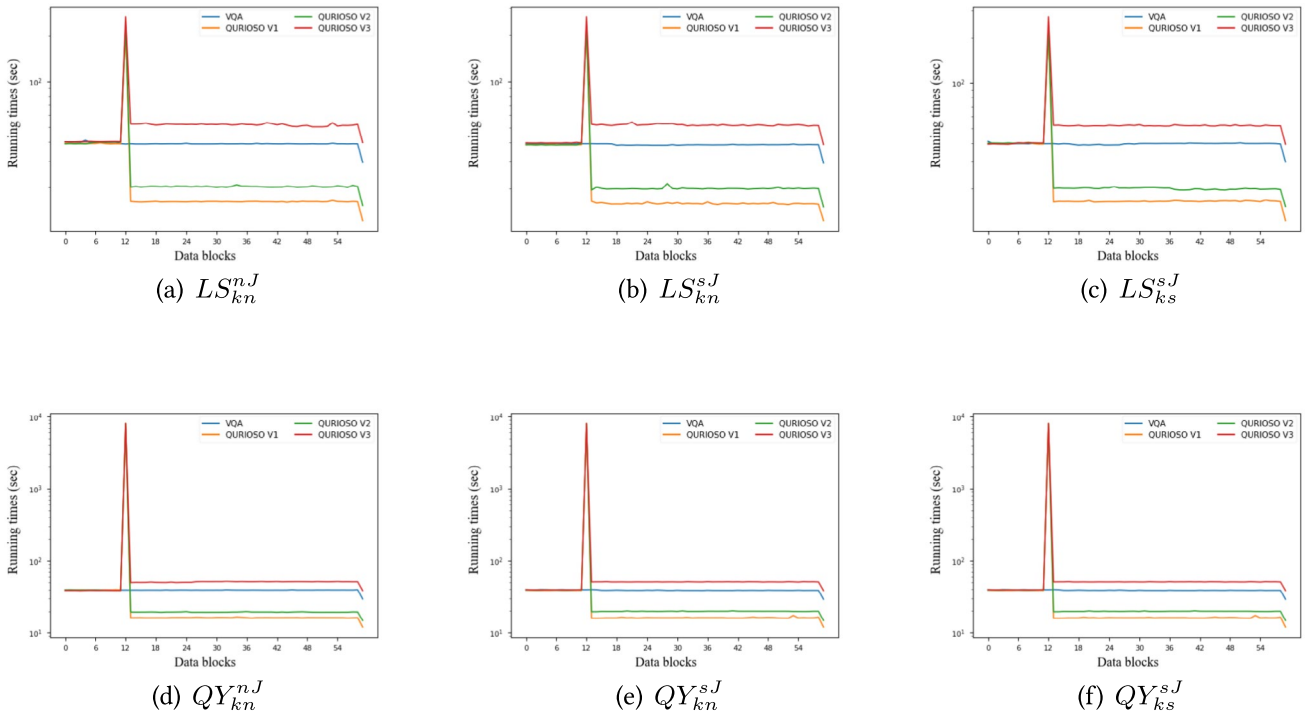
B.1. Spambase



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

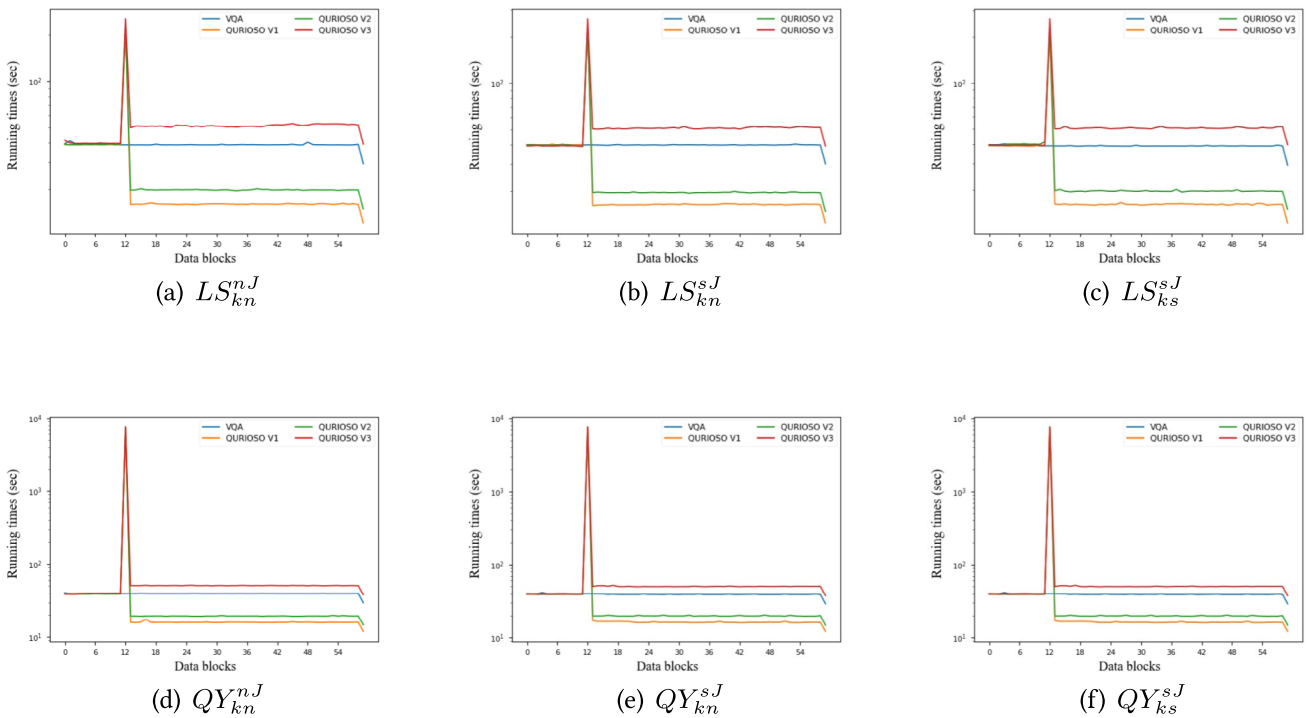
Fig. 28 Running times for $k = 5$ and $M = 3$. The blue line corresponds to the baseline (VQA), the orange line to $JS = 0$, the green line to $JS = \frac{I}{2} - JP$, and the red line to $JS = I - JP$. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

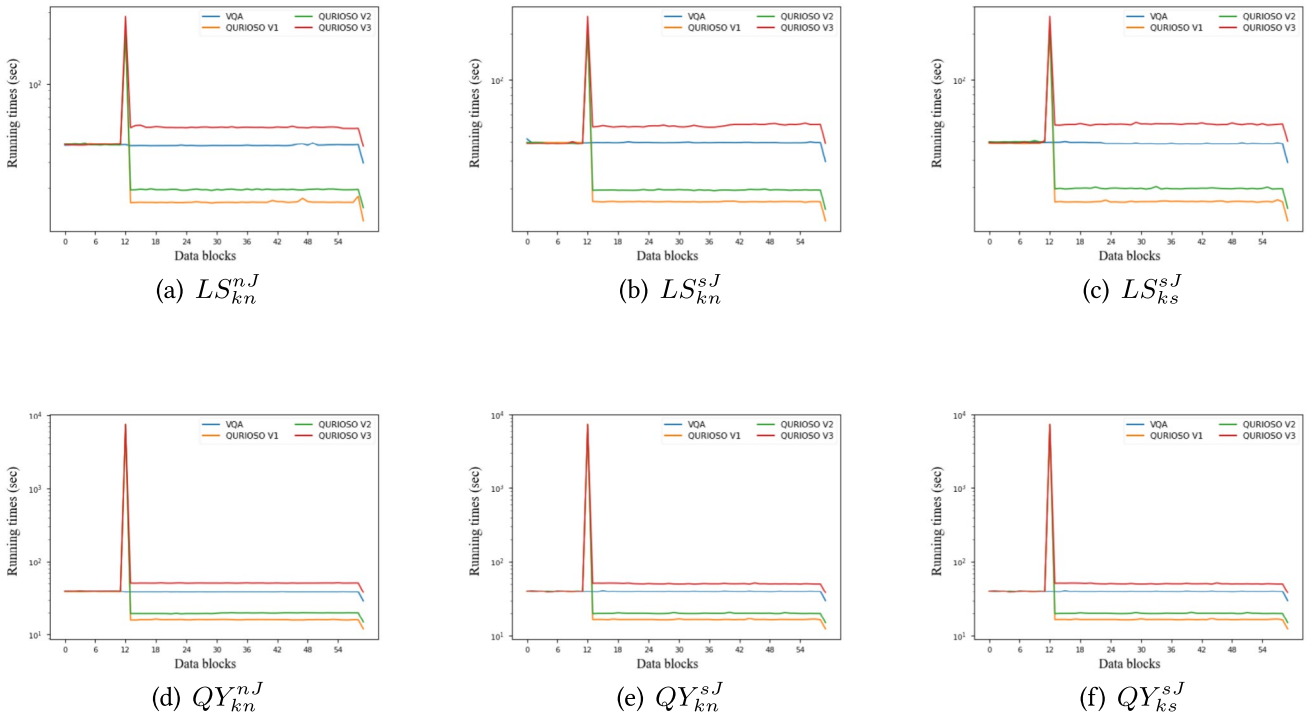
Fig. 29 Running times for $k = 5$ and $M = 5$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

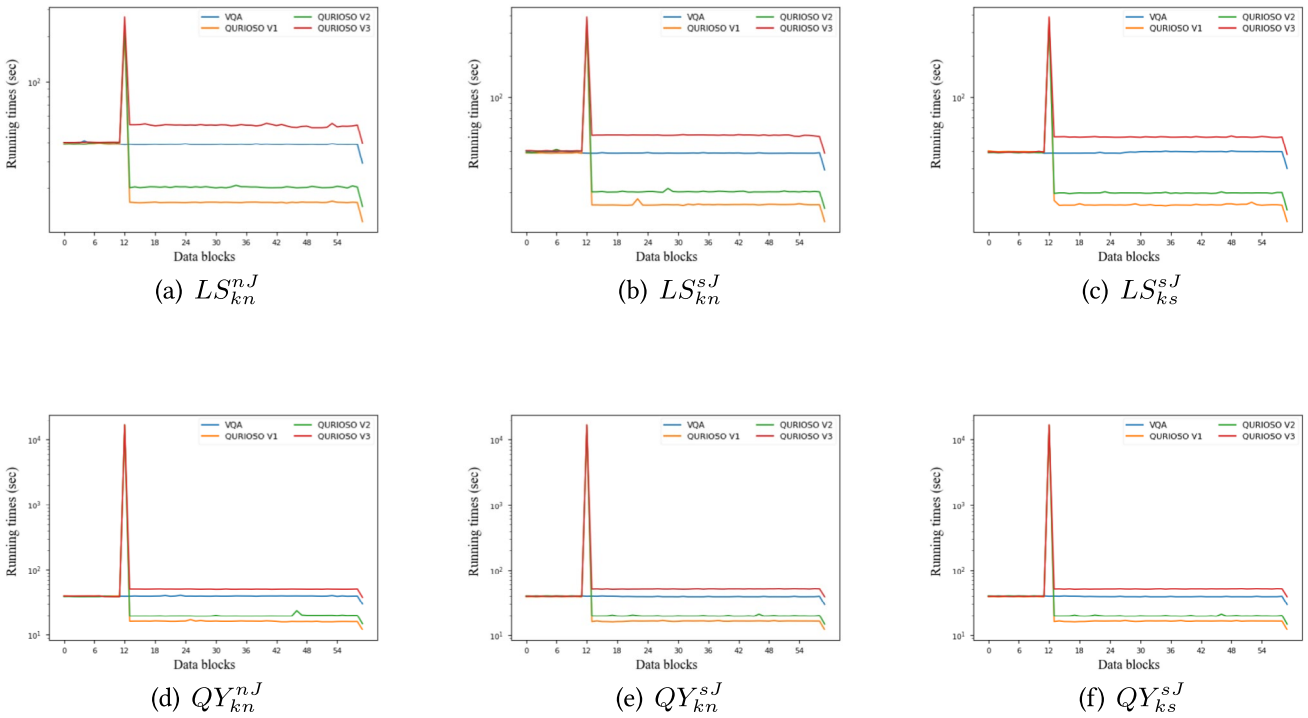
Fig. 30 Running times for $k = 5$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

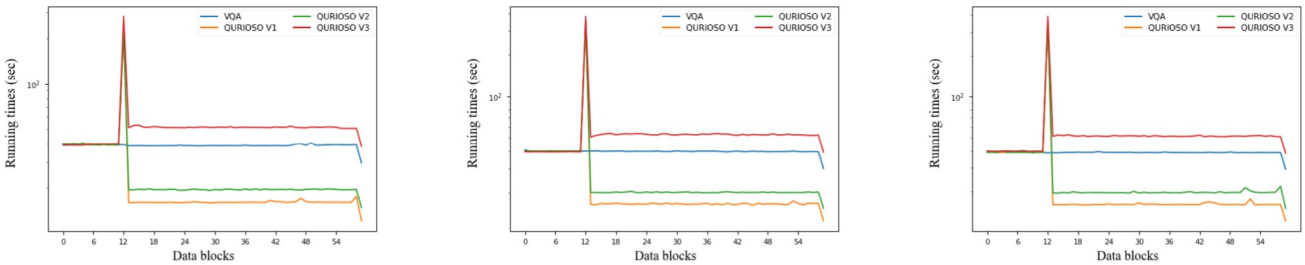
Fig. 31 Running times for $k = 5$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



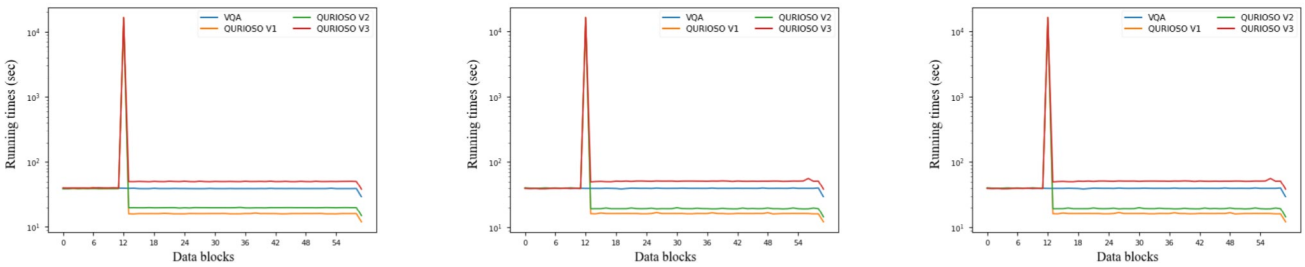
(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 32 Running times for $k = 10$ and $M = 5$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale

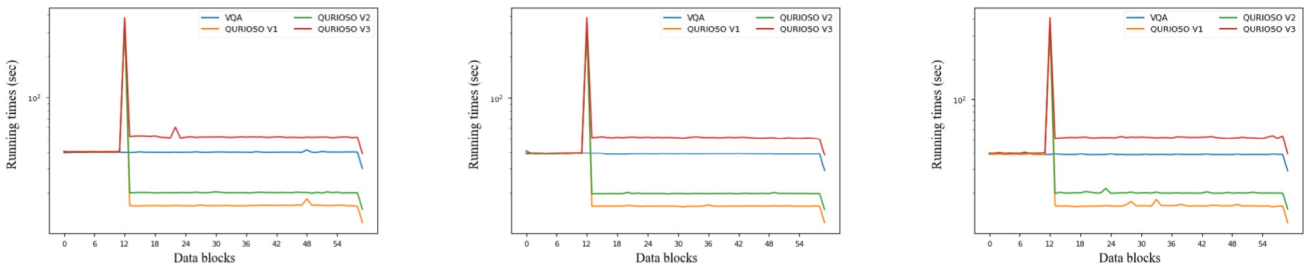


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

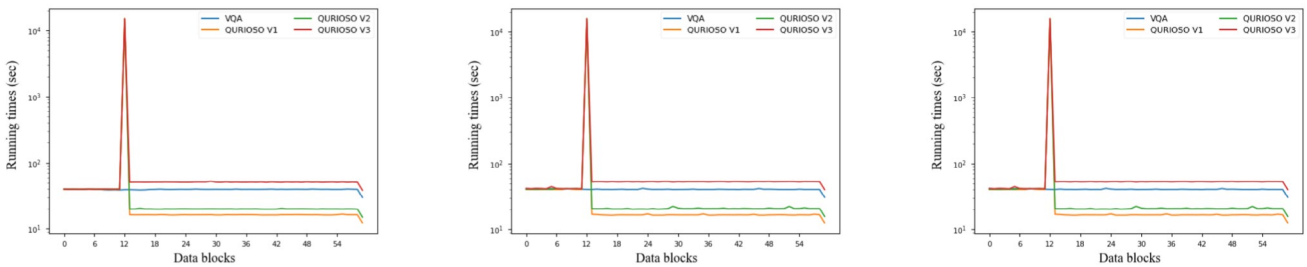


(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 33 Running times for $k = 10$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale

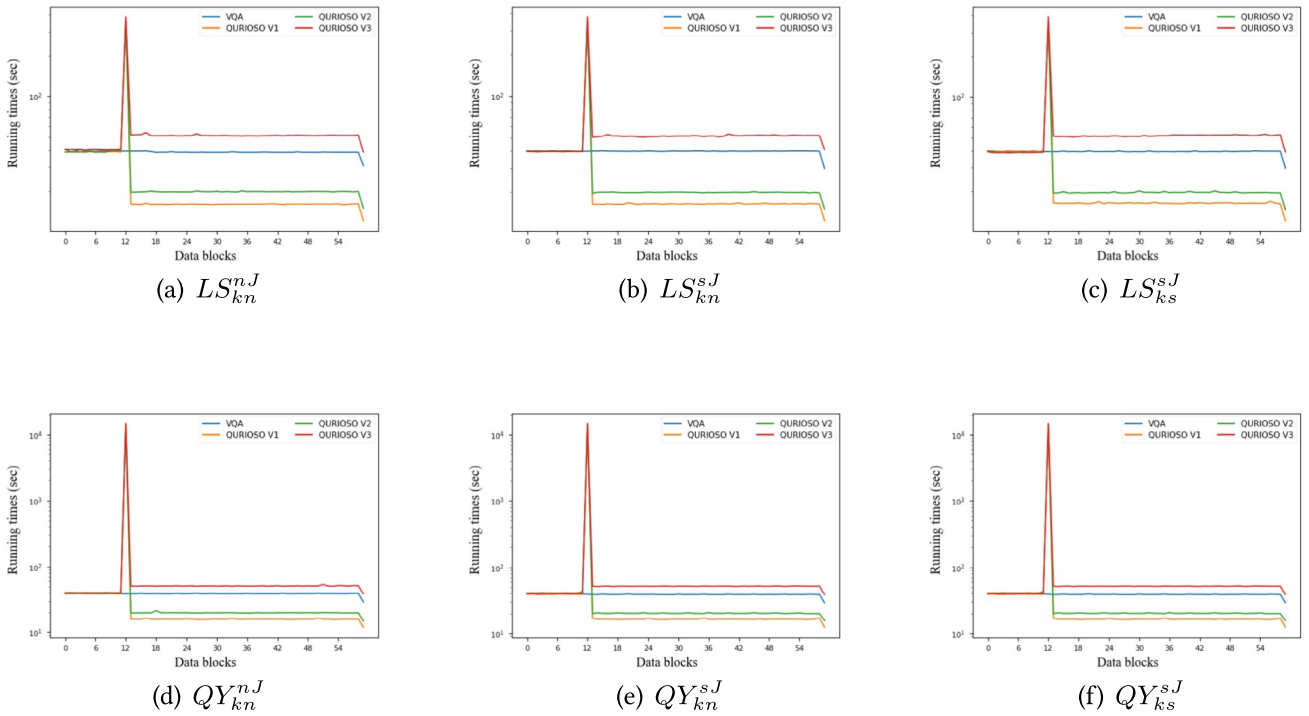


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}



(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

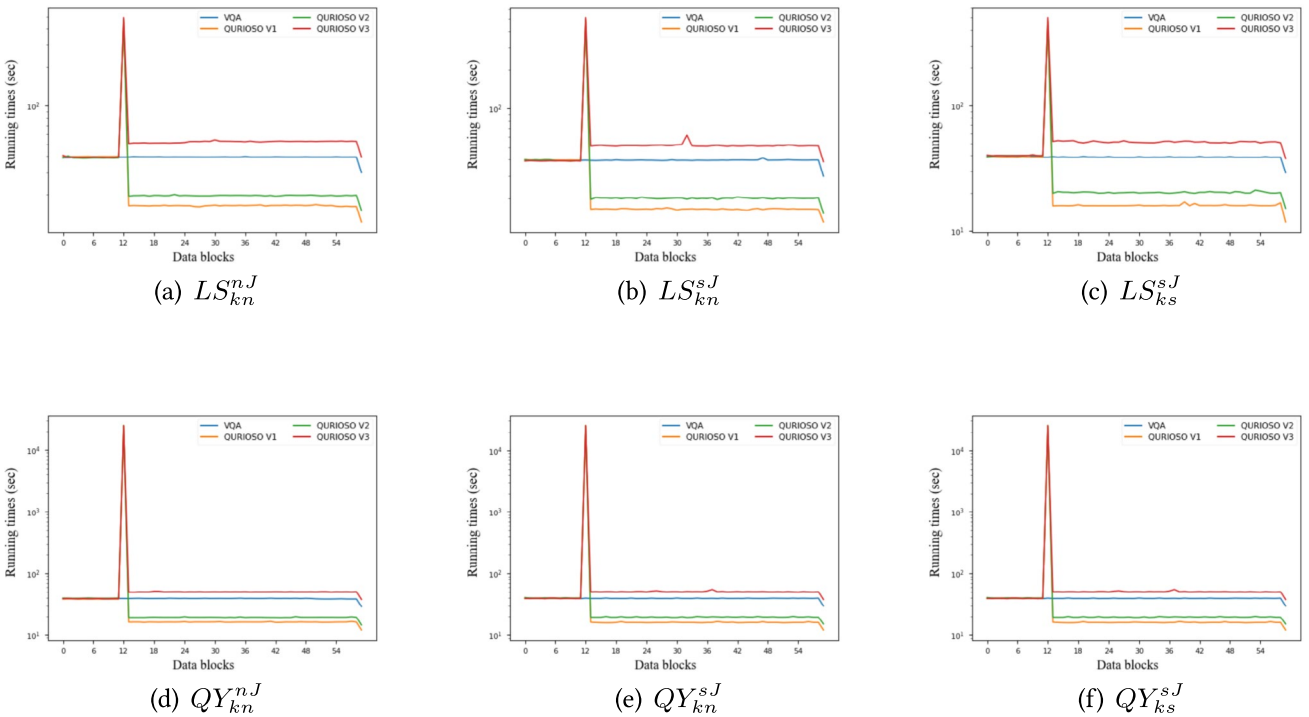
Fig. 34 Running times for $k = 10$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

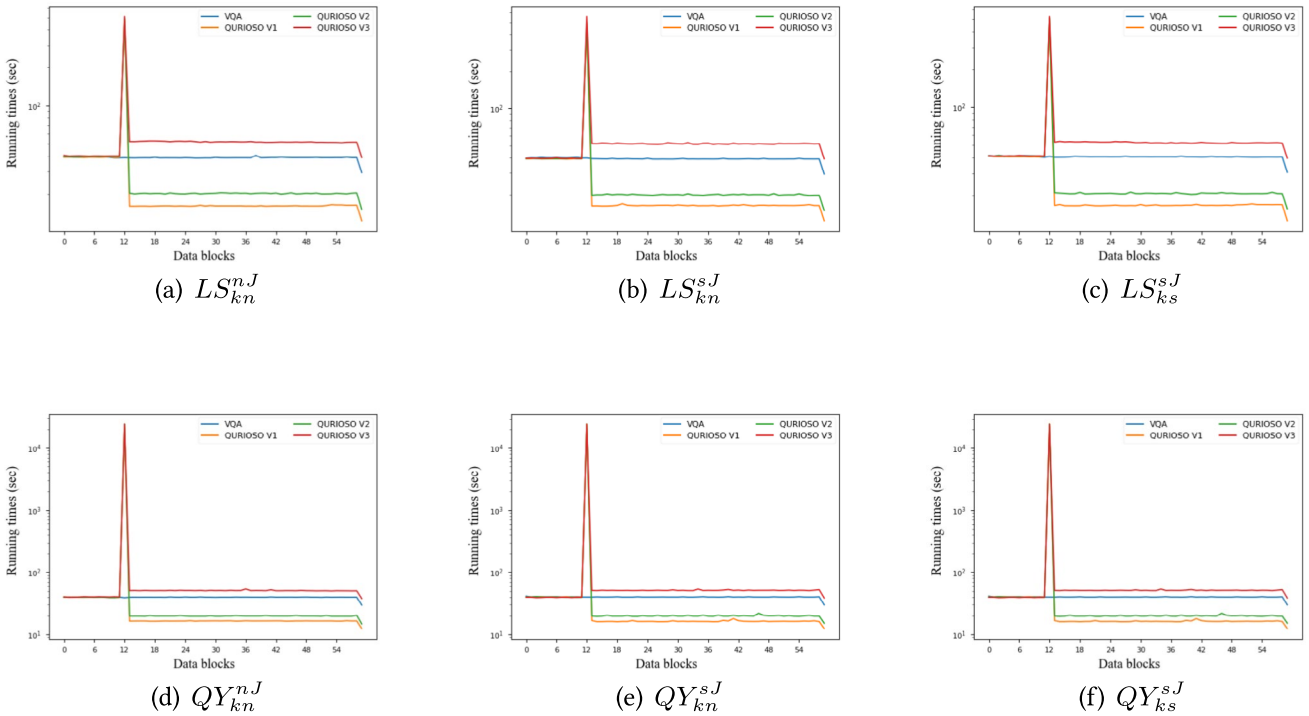
Fig. 35 Running times for $k = 10$ and $M = 20$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

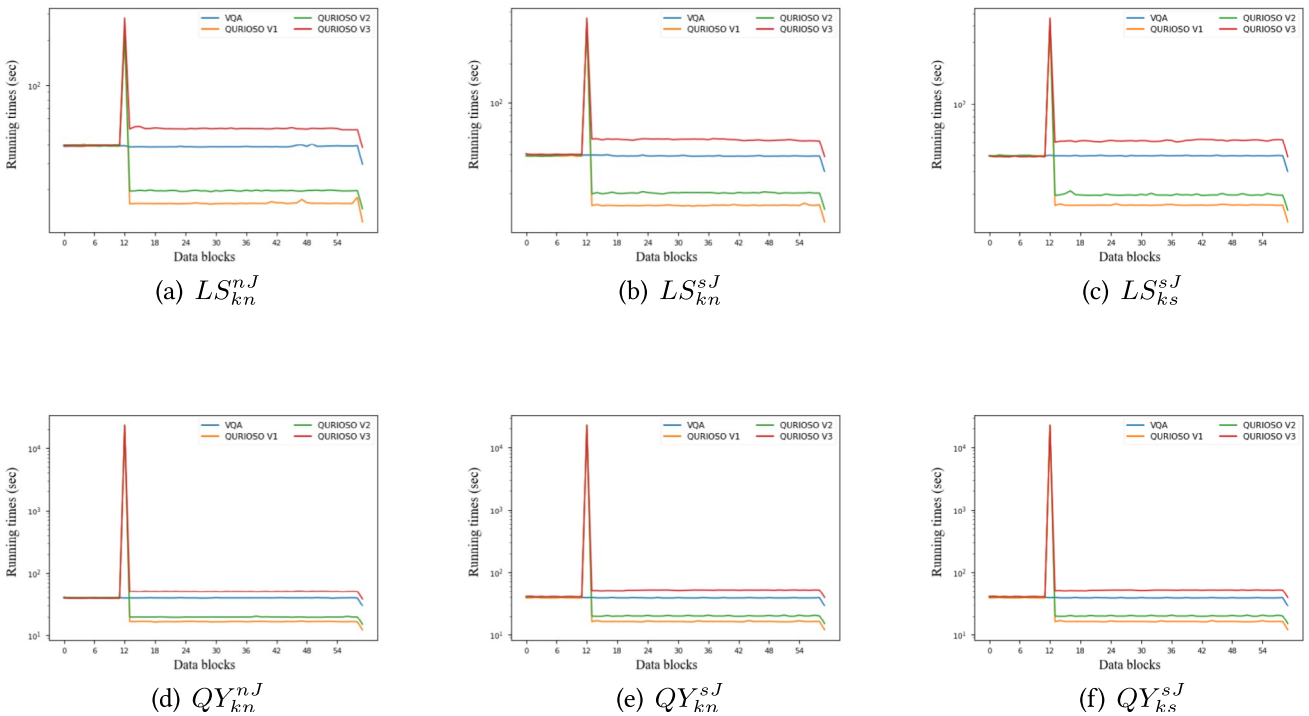
Fig. 36 Running times for $k = 15$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

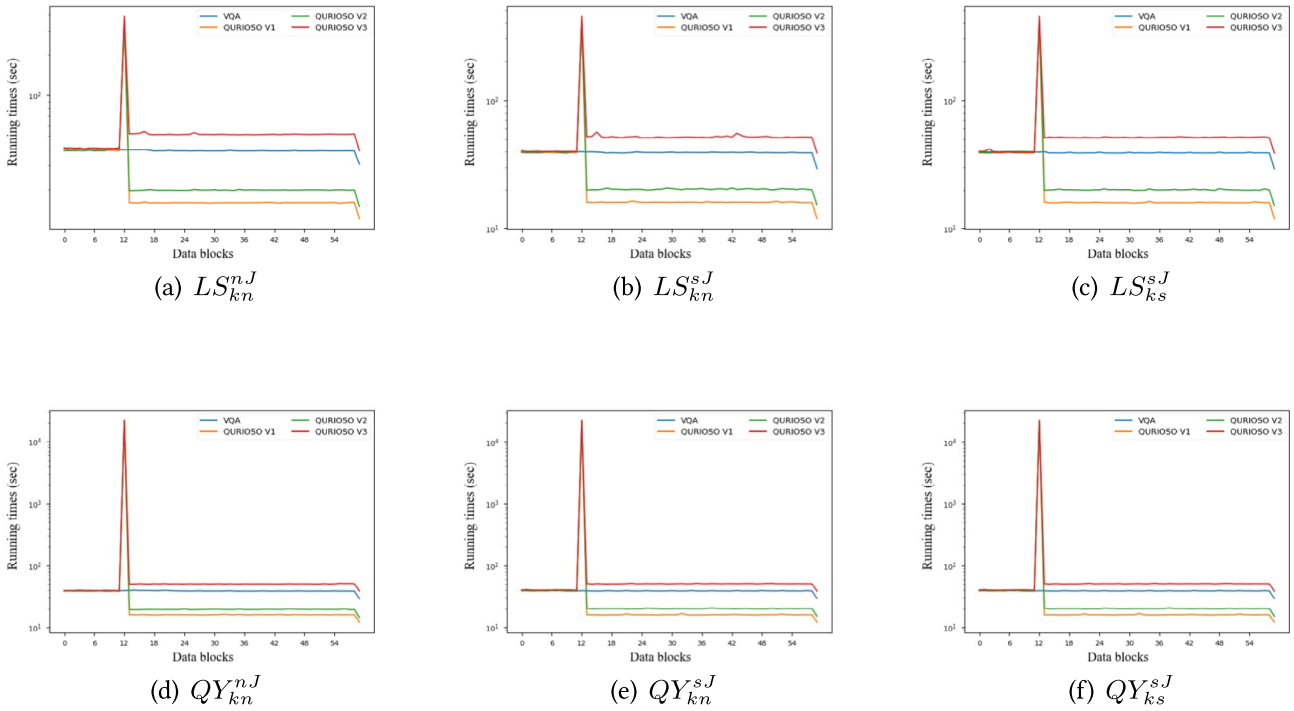
Fig. 37 Running times for $k = 15$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 38 Running times for $k = 15$ and $M = 23$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale

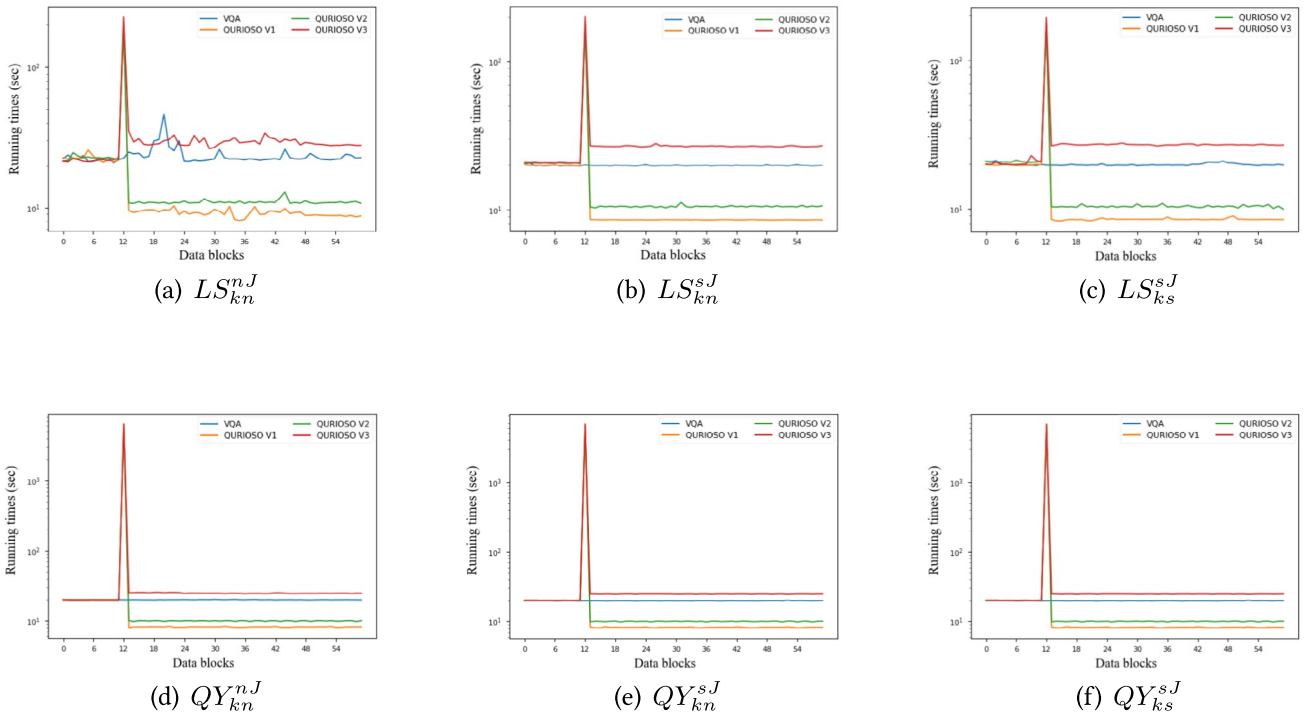


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 39 Running times for $k = 15$ and $M = 30$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale

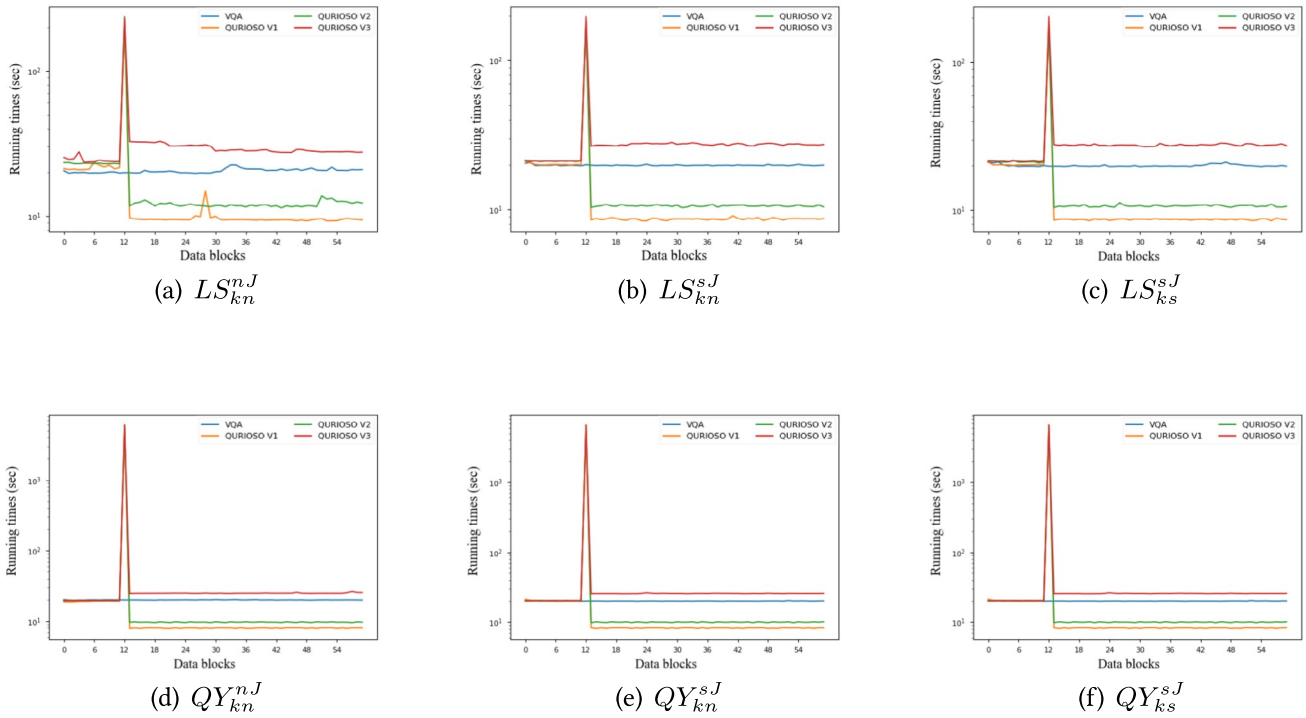
B.2. Ozone



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

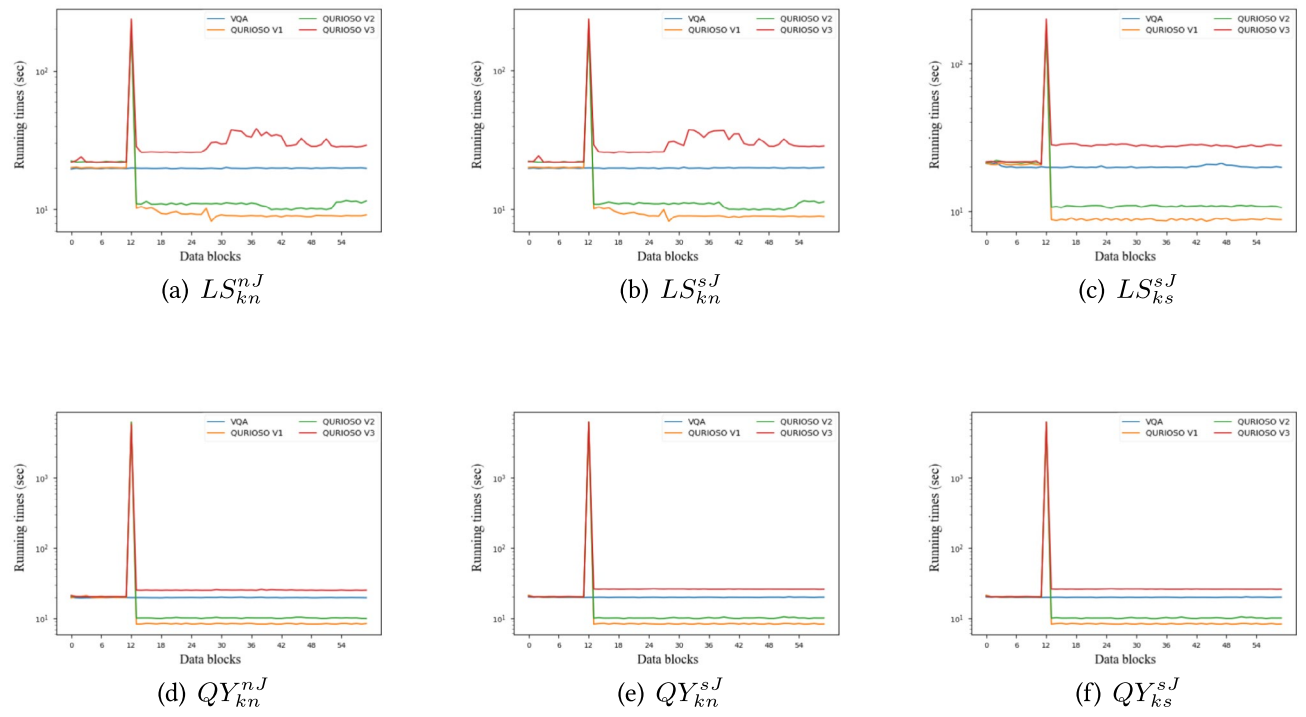
Fig. 40 Running times for $k = 5$ and $M = 3$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

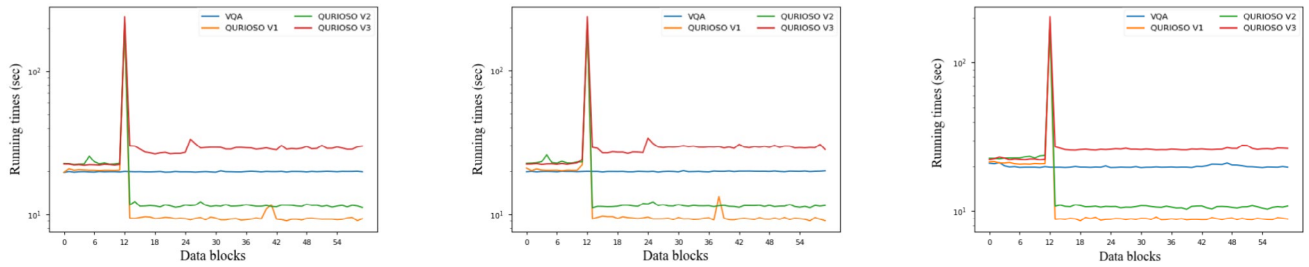
Fig. 41 Running times for $k = 5$ and $M = 5$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



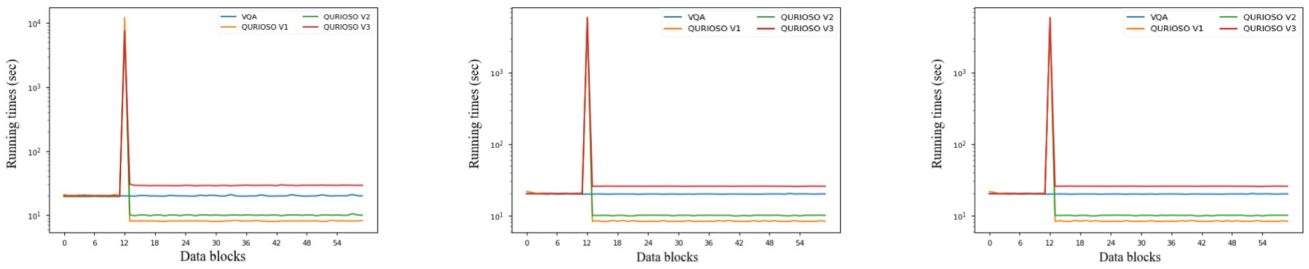
(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 42 Running times for $k = 5$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale

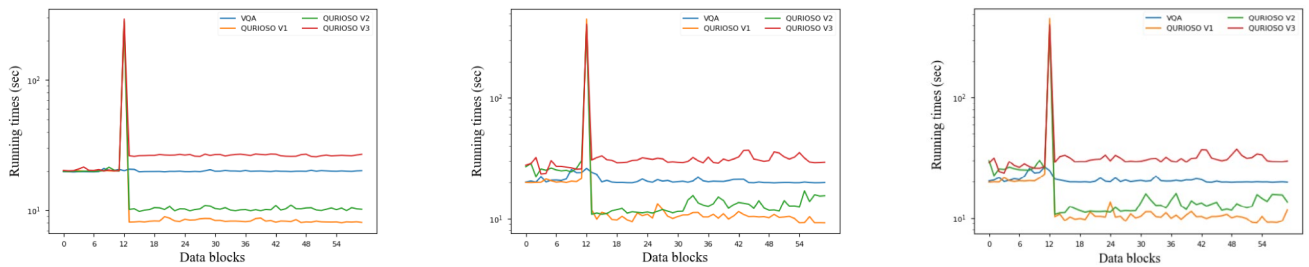


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

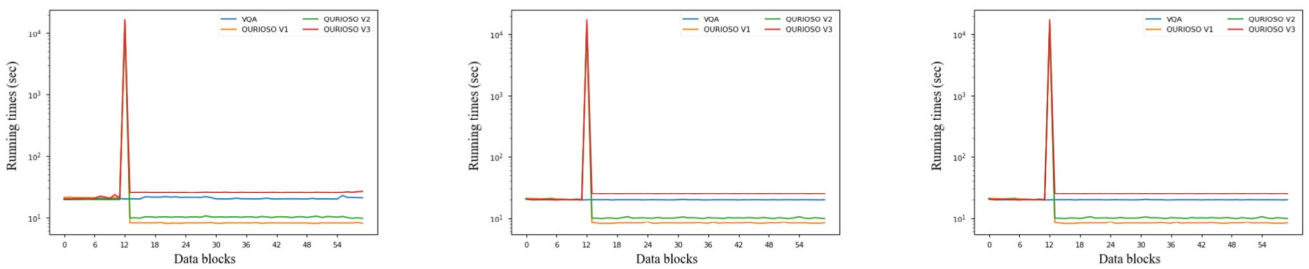


(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 43 Running times for $k = 5$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale

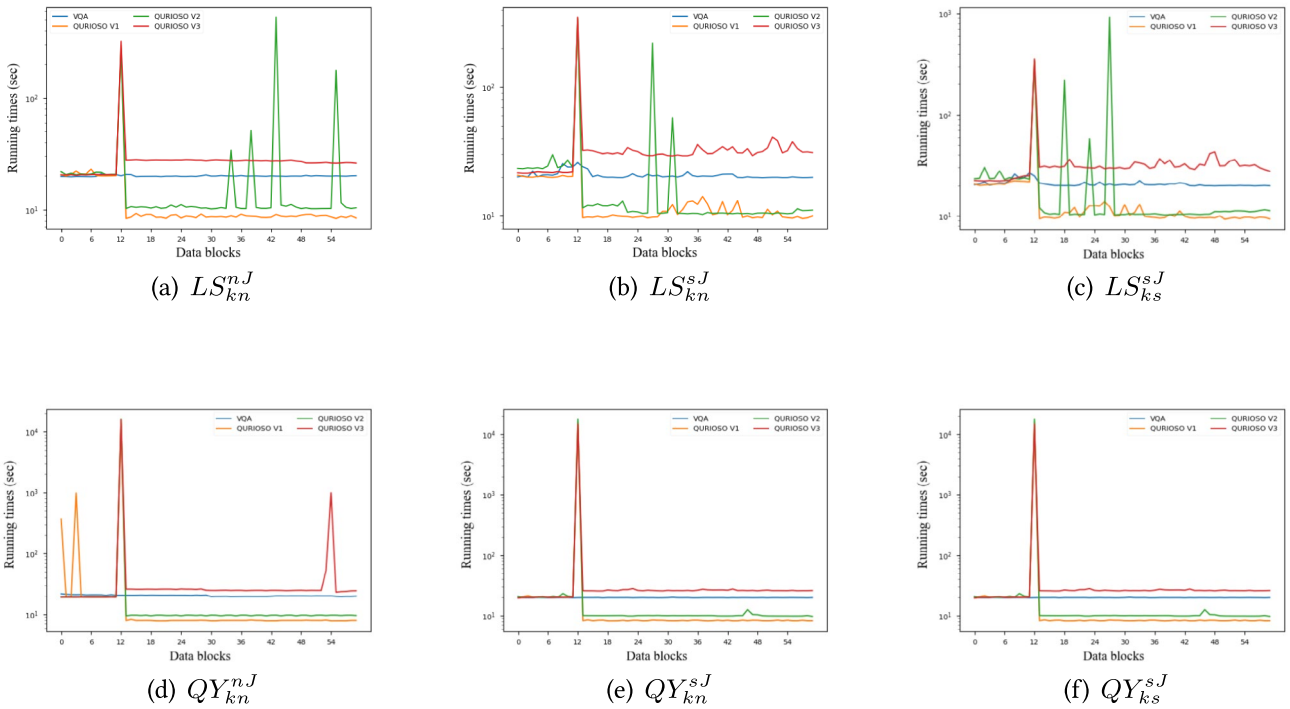


(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}



(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

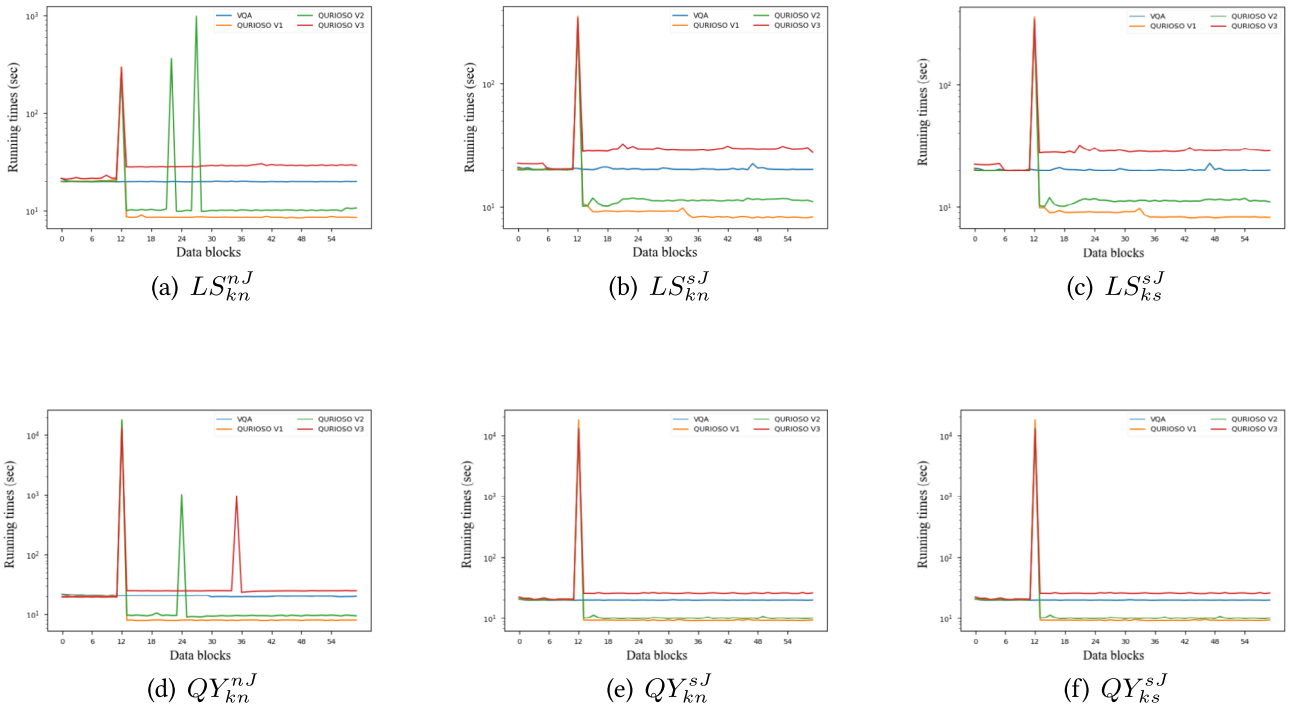
Fig. 44 Running times for $k = 10$ and $M = 5$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

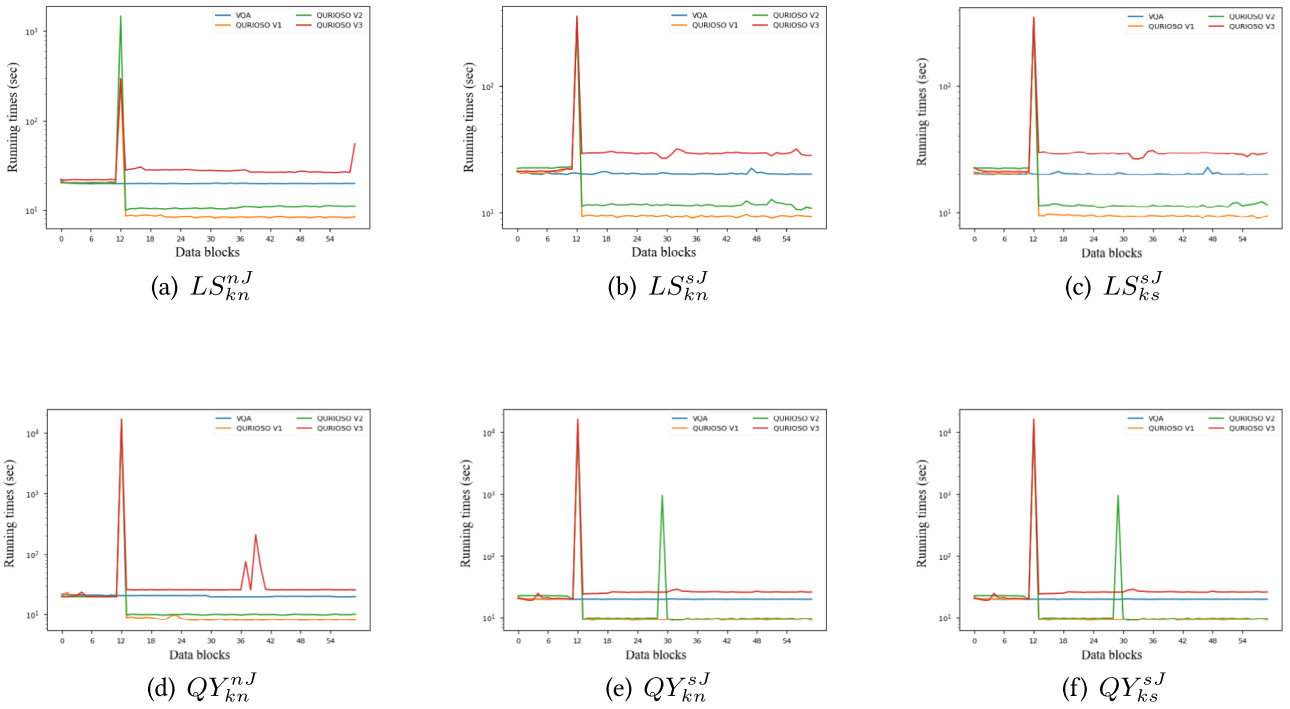
Fig. 45 Running times for $k = 10$ and $M = 10$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

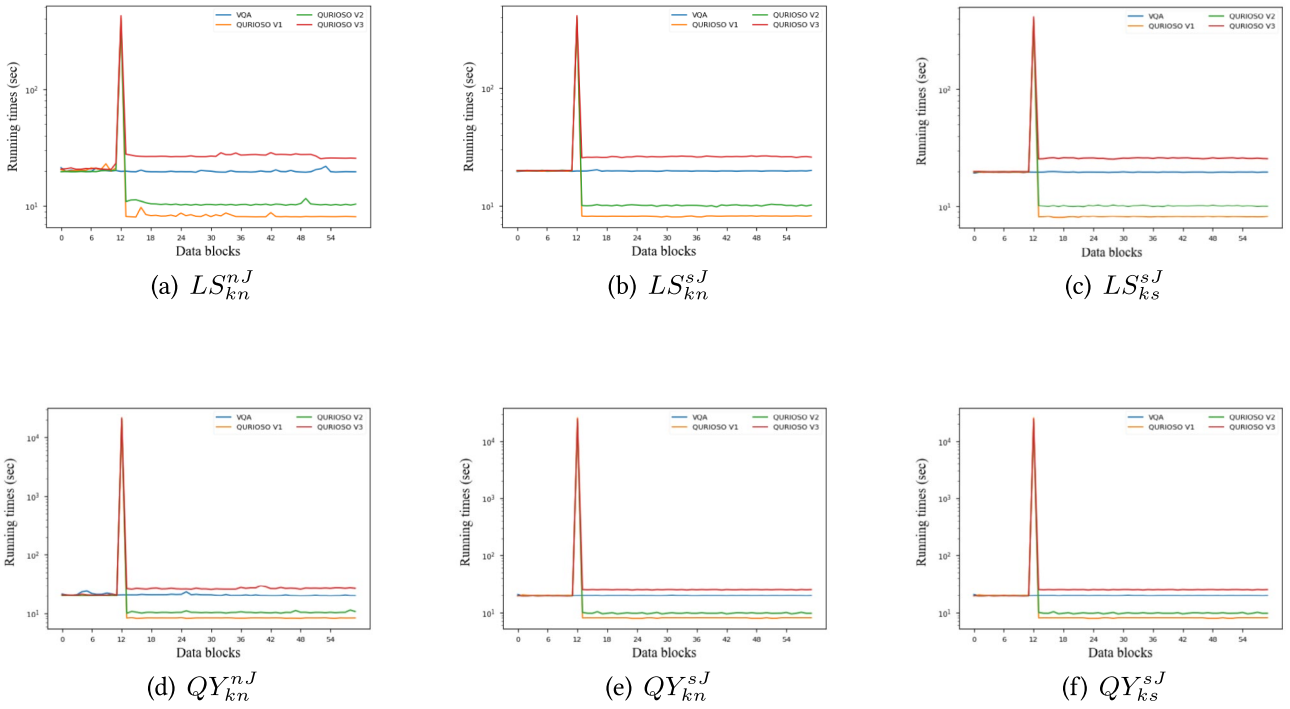
Fig. 46 Running times for $k = 10$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

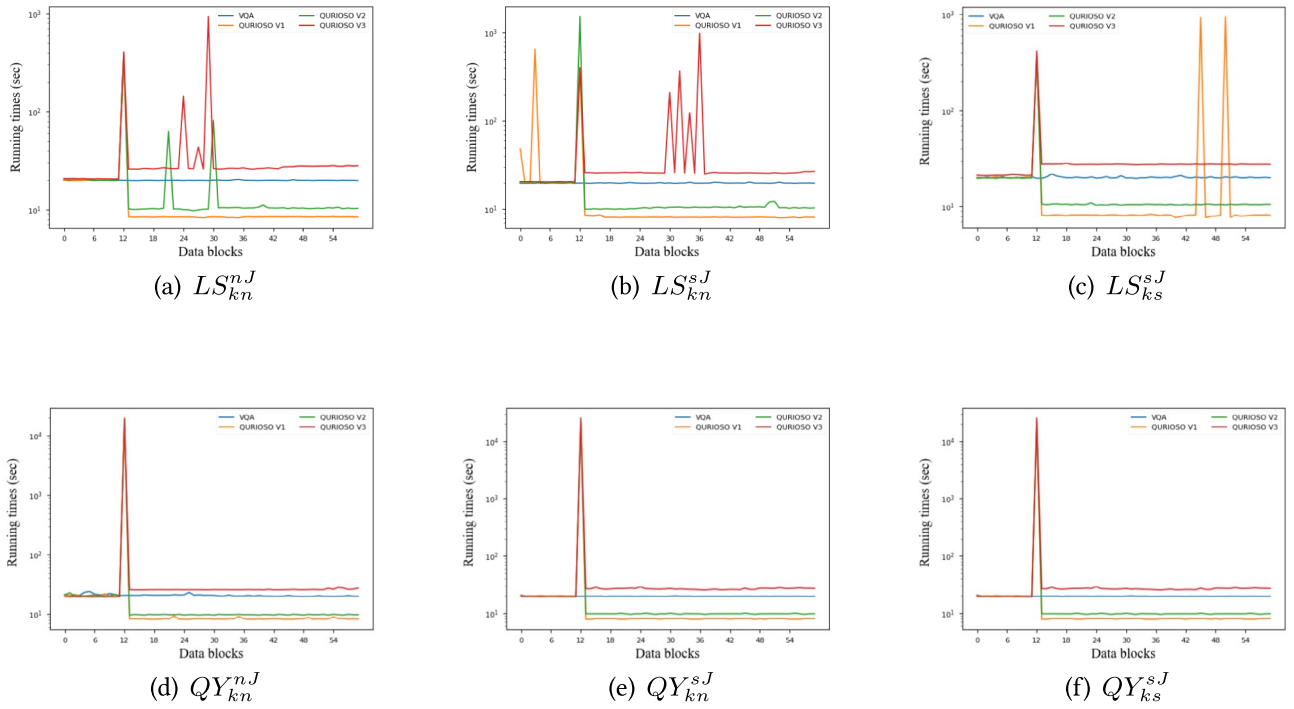
Fig. 47 Running times for $k = 10$ and $M = 20$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

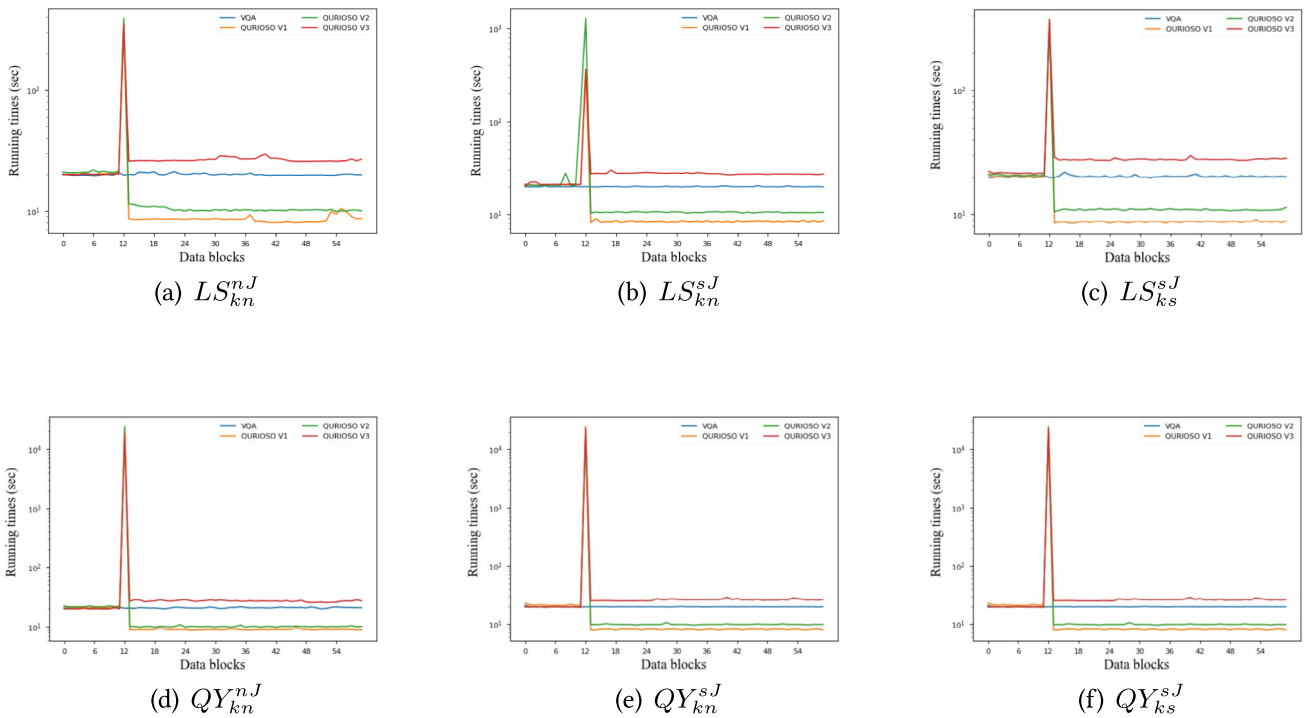
Fig. 48 Running times for $k = 15$ and $M = 8$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

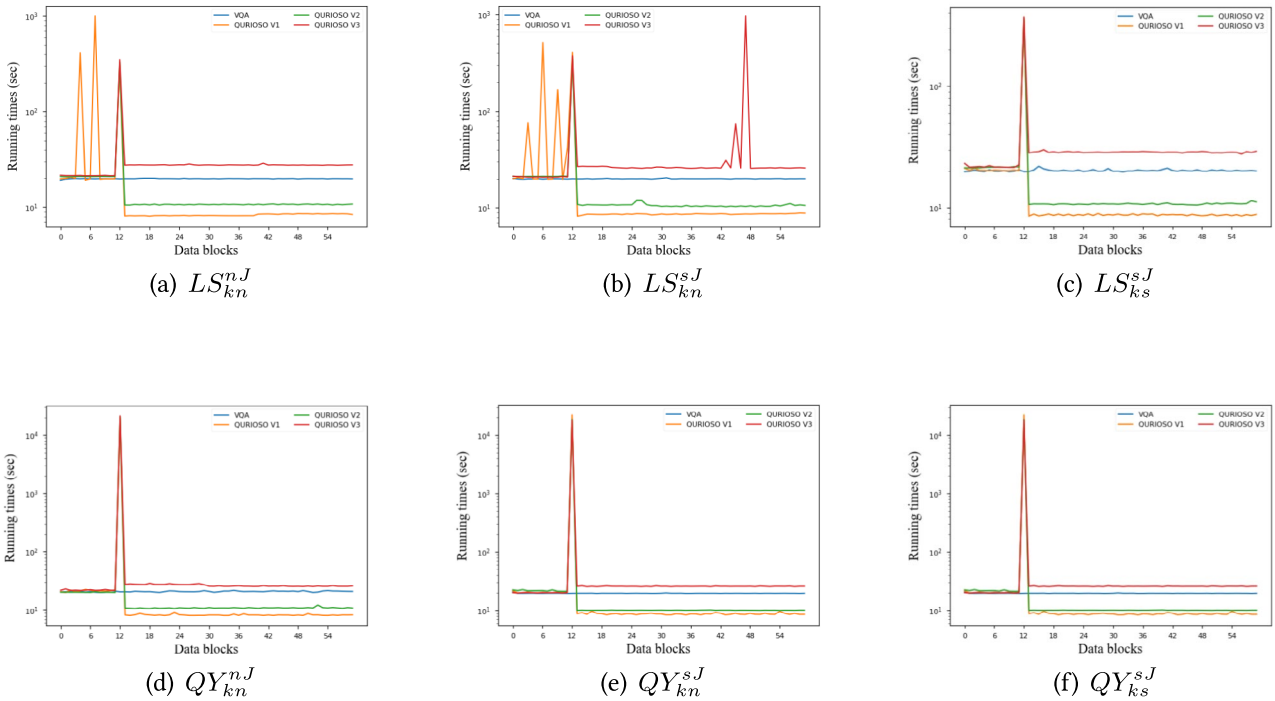
Fig. 49 Running times for $k = 15$ and $M = 15$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 50 Running times for $k = 15$ and $M = 23$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale



(a) LS_{kn}^{nJ} (b) LS_{kn}^{sJ} (c) LS_{ks}^{sJ}

(d) QY_{kn}^{nJ} (e) QY_{kn}^{sJ} (f) QY_{ks}^{sJ}

Fig. 51 Running times for $k = 15$ and $M = 30$. The blue line denotes the VQA baseline, while the colored lines correspond to the QURIOSO variants defined in Table 8. The X-axis represents the sequence of incoming data blocks, while the Y-axis reports the running times on a logarithmic scale

Acknowledgments Corrado Loglisci acknowledges the support from the PNRR MUR Project No. PE0000023-NQSTI.

Author Contributions C.L. contributed to methodology, validation, and writing. V.L. contributed to methodology, software, and validation. S.P. contributed to supervision and funding acquisition D.M. contributed to methodology and supervision. During the preparation of this work, the authors used ChatGPT-4 for grammar and spelling checks. All outputs were reviewed and edited by the authors, who take full responsibility for the final content of the manuscript.

Funding Open access funding provided by Università degli Studi di Bari Aldo Moro within the CRUI-CARE Agreement.

Data Availability The experimental datasets are available at the links https://archive.ics.uci.edu/ml/datasets/Spambase, https://archive.ics.uci.edu/dataset/172/ozone+level+detection

Declarations

Competing interests The authors declare no competing interests.

References

Anis MS et al (2021) Qiskit: An open-source framework for quantum computing. https://doi.org/10.5281/zenodo.2573505

Bergholm V, Izaac J, Schuld M, Gogolin C, Ahmed S, Ajith V, Alam MS, Alonso-Linaje G, AkashNarayanan B, Asadi A et al (2018) Pennylane: Automatic differentiation of hybrid quantum-classical computations. arXiv:1811.04968

Cao Y, Zhou X, Fei X, Zhao H, Liu W, Zhao J (2023) Linear-layer-enhanced quantum long short-term memory for carbon price forecasting. Quantum Mach Intell 5:1–12

Cerezo M, Arrasmith A, Babbush R, Benjamin SC, Endo S, Fujii K, McClean JR, Mitarai K, Yuan X, Cincio L, Coles PJ (2021) Variational quantum algorithms. Nat Rev Phys 3:625–644. https://doi.org/10.1038/s42254-021-00348-9

Chen SY-C, Yoo S, Fang Y-LL (2020) Quantum long short-term memory. arXiv:2009.01783

Falla J, Langfitt Q, Alexeev Y, Safro I (2024) Graph representation learning for parameter transferability in quantum approximate optimization algorithm. Quantum Mach Intell 6:46

Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. Mach Learn 90:317–346. https://doi.org/10.1007/S10994-012-5320-9

Hochreiter S, Schmidhuber J (1997a) Long short-term memory. Neural Comput 9:1735–1780

Hochreiter S, Schmidhuber J (1997b) Long short-term memory. Neural Comput 9:1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Homayouni H, Ghosh S, Ray I, Gondalia S, Duggan J, Kahn MG (2020) An autocorrelation-based lstm-autoencoder for anomaly detection on time-series data, in. IEEE Int Conf Big Data (Big Data) 2020:5068–5077. https://doi.org/10.1109/BigData50022.2020.9378192

Hopkins M, Reeber E, Forman G, Suermondt J (1999) Spambase data set, UCI machine learning repository. https://doi.org/10.24432/C53G6X

Imambi S, Prakash KB, Kanagachidambaresan G (2021) Pytorch. In: Programming with TensorFlow: solution for edge computing applications, Springer, pp 87–104

Jain N, Coyle B, Kashefi E, Kumar N (2022) Graph neural network initialisation of quantum approximate optimisation. Quantum 6:861. https://doi.org/10.22331/q-2022-11-17-861

Kandala A, Mezzacapo A, Temme K, Takita M, Brink M, Chow JM, Gambetta JM (2017) Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature 549:242–246. https://doi.org/10.1038/nature23879

Liang Z, Liu G, Liu Z, Cheng J, Hao T, Liu K, Ren H, Song Z, Liu J, Ye F et al (2024) Graph learning for parameter prediction of quantum approximate optimization algorithm. arXiv:2403.03310

Loglisci C, Malerba D (2023) Coupling quantum classification and quantum distance estimation in continual learning. In: AIQxQIA@AI*IA, volume 3586 of CEUR Workshop Proceedings, CEUR-WS.org

Loglisci C, Malerba D, Pascazio S (2024) Quarta: quantum supervised and unsupervised learning for binary classification in domain-incremental learning. Quantum Mach Intell 6:68. https://doi.org/10.1007/s42484-024-00196-7

Madsen L, Laudenbach F, Askarani M, Rortais F, Vincent T, Bulmer J, Miatto F, Neuhaus L, Helt L, Collins M, Lita A, Gerrits T, Nam SW, Vaidya V, Menotti M, Dhand I, Vernon Z, Quesada N, Lavoie J (2022). Quantum computational advantage with a programmable photonic processor. https://doi.org/10.1038/s41586-022-04725-x

McClean JR, Boixo S, Smelyanskiy VN, Babbush R, Neven H (2018) Barren plateaus in quantum neural network training landscapes. Nat Commun 9:4812. https://doi.org/10.1038/s41467-018-07090-4

Meng F, Zhou X (2024) Parameter generation of quantum approximate optimization algorithm with diffusion model. arXiv:2407.12242

Moussa C, Wang H, Bäck T, Dunjko V (2022) Unsupervised strategies for identifying optimal parameters in quantum approximate optimization algorithm. EPJ Quantum Technol 9:11

Parisi GI, Kemker R, Part JL, Kanan C, Wermter S (2019) Continual lifelong learning with neural networks: A review. Neural Netw 113:54–71. https://doi.org/10.1016/j.neunet.2019.01.012

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. J Mach Learn Res 12:2825–2830

Pesah A, Cerezo M, Wang S, Volkoff T, Sornborger AT, Coles PJ (2021) Absence of barren plateaus in quantum convolutional neural networks. Phys. Rev. X 11. https://doi.org/10.1103/physrevx.11.041011

Ravi GS, Gokhale P, Ding Y, Kirby W, Smith K, Baker JM, Love PJ, Hoffmann H, Brown KR, Chong FT (2022) Cafqa: A classical simulation bootstrap for variational quantum algorithms, ASPLOS 2023, Association for Computing Machinery, New York, NY, USA, p 15–29. https://doi.org/10.1145/3567955.3567958

Ross BC (2014) Mutual information between discrete and continuous data sets. PLoS ONE 9:e87357. https://doi.org/10.1371/journal.pone.0087357

Rossi RA (2018) Relational time series forecasting. Knowl Eng Rev 33:e1. https://doi.org/10.1017/S0269888918000024

Sahin ME, Altamura E, Wallis O, Wood SP, Dekusar A, Millar DA, Imamichi T, Matsuo A, Mensa S (2025) Qiskit machine learning: an open-source library for quantum machine learning tasks at scale on quantum hardware and classical simulators. arXiv:2505.17756

Sauvage F, Sim S, Kunitsa AA, Simon WA, Mauri M, Perdomo-Ortiz A (2021) Flip: A flexible initializer for arbitrarily-sized parametrized quantum circuits. arXiv:2103.08572

Schuld M, Bocharov A, Svore KM, Wiebe N (2020) Circuit-centric quantum classifiers. Phys Rev A 101:032308. https://doi.org/10.1103/PhysRevA.101.032308

Singh P, Manure A (2019) Introduction to tensorflow 2.0, in: Learn TensorFlow 2.0: Implement machine learning and deep learning models with python, Springer, pp 1–24

Son J, Lee S, Kim G (2025) When meta-learning meets online and continual learning: A survey. IEEE Trans Pattern Anal Mach Intell 47:413–432. https://doi.org/10.1109/TPAMI.2024.3463709

van de Ven GM, Tolias AS (2019) Three scenarios for continual learning. arXiv:1904.07734

Vanschoren J (2019) Meta-learning, in: Automated machine learning: methods, systems, challenges, Springer International Publishing Cham, pp 35–61

Wang H, Zhao J, Wang B, Tong L (2021) A quantum approximate optimization algorithm with metalearning for maxcut problem and its simulation via tensorflow quantum. Math Probl Eng 2021:6655455

Wilson M, Stromswold R, Wudarski F, Hadfield S, Tubman NM, Rieffel EG (2021) Optimizing quantum heuristics with meta-learning. Quantum Mach Intell 3:13

Xie Z, Yang Y, Zhang Y, Wang J, Du S (2023) Deep learning on multi-view sequential data: a survey. Artif Intell Rev 56:6661–6704

Zhang K, Fan W, Yuan X (2008) Ozone level detection data set. UCI Mach Learn Reposit. https://doi.org/10.24432/C5NG6W