Reinforcement Learning Framework for Missing Data Imputation in IoT Environments

Ahmed M. Salama Salem, Sayed AbdelGaber A, Ahmed E. Yakoub
Faculty of Computers and Artificial Intelligence, Helwan University, Cairo, Egypt

***Abstract*—Continuous, accurate meteorological sensing underpins many Internet of Things (IoT) applications, from smart irrigation and urban heat-island monitoring to early weather warnings, but data from distributed stations are often disrupted by sensor faults, power loss, or communication noise, causing missing values that degrade analytics and decisions. Existing data imputation methods lose accuracy on small or irregular datasets and adapt poorly to dynamic IoT settings. This study proposes a reinforcement learning (RL)-based framework for missing-data imputation that treats each gap as a sequential decision problem. The authors develop and compare three RL architectures, two Q-table methods and one Deep Q-learning model, to learn temporal dependencies and optimize imputation via experience. A second objective is to assess the feasibility and performance of RL for imputation in domains related to robotics and autonomous systems, where RL remains less explored. A third objective is to validate the methods on real-world datasets and simulations, supported by a user-friendly graphical interface for visualization and performance monitoring. The proposed RL imputers outperform state-of-the-art methods in accuracy and robustness: the best RL configuration cuts MSE/MAE by 8.6%/5.9% vs. K-Nearest Neighbors' algorithm (KNN), 74.4%/75.6% vs. autoencoder, 79.6%/79.9% vs. clustering, 89.0%/83.7% vs. mean, 89.5%/83.3% vs. median, and 94.2%/89.3% vs. most-frequent, while raising the coefficient of determination (R^2) by +0.023, +0.532, +0.123, +0.407, +0.436, and +0.932, respectively. These findings highlight RL as an effective paradigm for intelligent data restoration in IoT-based sensing systems.**

***Keywords*—Data imputation; reinforcement learning; machine learning; deep learning; Internet of Things (IoT)**

I. Introduction

Industry and academia are both deeply interested in maximizing the potential usefulness of the Internet of Things (IoT), a system that links many millions of physical devices to the Internet and, in doing so, generates unprecedented volumes of data. Embedded sensors and actuators enable such devices to sense relevant properties of their surroundings, analyze and process information locally, and exchange data with peers or cloud services, enabling coordinated autonomous decisions [1]. Advances in smart-sensor design, wireless communication, and data-aggregation technologies have made it possible to collect heterogeneous data streams from numerous different sources such as environmental probes, industrial machinery, surveillance cameras, and mobile devices [2]. Such data drives predictive models that improve reliability, efficiency, profitability, and overall performance in smart cities, industrial automation, intelligent buildings, connected vehicles, and environmental monitoring systems [3].

Despite the benefits, real-world deployment suffers from missing or corrupted observations due to sensor faults, loss of power, communication failures, long-term monitor fatigue, battery depletion, and/or poorly defined boundaries in large-scale network studies. Missing values can introduce bias, complicate statistical analysis, and degrade model accuracy. This missing value can be imputed using machine learning (ML) and deep learning (DL) algorithms.

Data imputation is defined as representing missing values in a dataset or IoT stream, which presents a number of challenges for data analysis as such values can increase the chance of introducing errors, reduce analytic accuracy, and limit the reliability of confidence intervals. Data Imputation can be done via conventional machine learning (ML) algorithms, such as hot- and cold-deck substitution, last observation carried forward, mean, multiple or regression imputation, non-negative matrix factorization, and stochastic methods, frequently reject useful data or introduce bias [4]. IoT data will often include varied forms including binary, categorical, numerical, ordinal, spatial, and temporal, with each requiring its own specialized analysis. Thus, recent research has moved toward data-driven and hybrid approaches that are better able to adapt to heterogeneous data inputs. As a result, imputation techniques may now be differentiated into two broad groups which are machine learning-based, and deep learning-based approaches.

Deep learning (DL) approaches, such as autoencoders and convolutional neural networks (CNNs), show great potential for coping with missing data [5], but often need substantial and well-organized datasets to provide reliable performance. Autoencoders provide suggestions for missing data by learning compressed, noise-tolerant versions of the data and then rebuilding the original input, predicting absent values during the decoding process. Multimodal, stacked and variational autoencoders can enhance reconstruction accuracy for many data types. CNN-based models are most effective when the data can be reformed into multi-dimensional arrays such as matrices or images, enabling the network to capture temporal or spatial relationships. This makes CNNs suitable for imputing missing values in, for example, traffic-sensor grids, medical records, or spatiotemporal datasets which enable the study of the maintenance of patterns over time and are particularly useful for environmental data.

However, three major challenges remain. First, most traditional machine learning and statistical imputation methods provide limited accuracy when applied to real-world problems, while modern DL-based approaches often require large training datasets and their performance deteriorates when only small or incomplete datasets are available. Second, although RL has been

extensively explored in robotics and autonomous systems, relatively little attention has been given to its potential in the data domain for imputing missing data, despite its successful application to autonomous decision-making. Third, to the best of the authors' knowledge, there is still no standardized workflow for simulating and deploying advanced imputation models in realistic IoT environments or on a commercial scale.

To address such challenges, the authors propose a RL-based imputation framework that makes each missing value a sequential decision-making problem. This constitutes a first contribution by explicitly formulating missing-data imputation as a sequential decision-making task. This framework develops and is evaluated via three progressively enhanced variants: 1) a baseline Q-Table, 2) an improved Q-Table Version 2 (V2) with optimized hyperparameter tuning, and 3) Deep Q-Learning (V3) that utilizes neural networks for function approximation. This systematic development enables a direct and controlled comparison between tabular and deep RL architectures, representing a second contribution of this work. These RL variants are benchmarked against the widely used imputation techniques: clustering, mean, most-frequent-value as well as DL-based approaches such as autoencoders, and ML methods such as K-Nearest Neighbors (KNN). The evaluations employ multiple performance metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and the coefficient of determination (R^2) to provide a comprehensive assessment of accuracy and reliability. This proposed RL-based approach should enhance imputation accuracy with limited data availability to maintain accuracy of real-time analytics for continuous, dynamic IoT data streams by providing more robust and scalable data.

The remainder of this study is organized as follows: Literature Review discuss the state-of-the-art in this domain and introduces the theoretical foundations of RL. Then, Methodology discusses the design and implementation of the proposed RL algorithms, including data preprocessing, missing-data synthesis, model training, and simulation. The Results and Discussion section presents and interprets the experimental outcomes compared to relevant published works. Finally, the Conclusion summarizes the key findings and suggests directions for future research.

II. Literature Review

Accurate and reliable imputation remains fundamental to downstream analytics in IoT settings where streams are heterogeneous, non-stationary, and frequently sparse (see Fig. 1). Contemporary work spans two broad families: 1) classical baselines prized for simplicity and speed, 2) deep learning (DL) models that trade higher capacity for greater data and training demands. The review below positions our chosen approach within this landscape and explains why RL is a timely alternative.

Large comparative studies consistently show that traditional ML approaches such as KNN [6], clustering [7], mean [8],[9], median, most frequent [6], and tree-based methods, such as MissForest, outperform very simple imputers. However, while clustering, mean, and most-frequent are convenient they are, typically, weaker representations of patterns often found in IoT-style datasets [8],[9]. A 2024 cohort analysis identifies KNN and Random Forest among the strongest practical baselines [9]. Similarly, a recent benchmark analysis paper evaluate the performance of common methods of missing data imputation reports that KNN and MIDAS, which is a denoising-autoencoder approach, perform best at higher rates of random data and block omissions, with substantial runtime differences between methods [6].

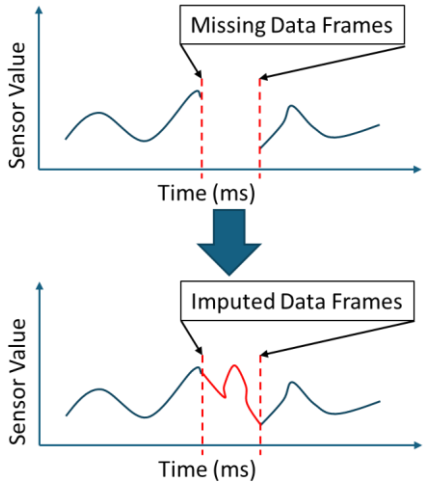

Fig. 1. Data imputation reconstructs missing segments in IoT sensor time series in real-time. The upper plot shows a typical sensor signal with a gap between two dashed lines representing missing data frames caused by sensor faults, packet loss, or power interruptions. The central block (arrow) denotes the imputation algorithm, which processes the incomplete sequence. The lower plot illustrates the imputed data frames (red curve) that reconstruct the missing segment while preserving the original temporal trend and signal continuity.

DL produces high accuracy rather than ML. One DL method, Generative Adversarial Networks (GAN)-based imputers, such as GAIN and its variants, continue to evolve to mitigate mode collapse and gradient instability [10]. A recent study introduces training refinements that improve both stability and accuracy. The clustering and classification-based generative adversarial imputation network (CC-GAIN) features enhanced couples clustering or classification with GANs to better handle multivariate settings which were successfully used in missing data imputation in an electricity consumption system [11]. There is also evidence that the generative adversarial imputation network (GAIN) in hybrid combination with convolutional neural networks (CNNs) can be used for exploiting tensorized representations to provide greater accuracy with sparse multivariate data, but with heavier computational cost and potential training instability compared to simpler baselines [12]. These works justify including DL baselines beyond autoencoders in comparisons while acknowledging their practical trade-offs.

Auto encode (AE) is also a DL method and recent research has refined denoising autoencoders (DAEs) specifically for tabular imputation. DAE for missing data imputation has been developed using a modified loss and a simple hyperparameter rule [13]. It use across multiple UCI datasets and missingness regimes has shown it to provide thorough ablations against standard baselines and rank among the top performers, making it a strong contender for use as an AE-style imputer for tables [5].

Beyond generic IoT benchmarks, DAEs have been evaluated in cases where the missingness was high (≥40%) heterogeneous biomedical datasets (with a mix of imaging, clinical, cognitive, and genetic variables), which demonstrated where DAEs excel and but also that they struggled under extreme sparsity typical of real-world IoT/health data sets [13]. Remasker, a recently developed method, extended masked autoencoding to tabular data by re-masking observed entries during training and predicting them as a self-supervised target [15]. Competitive or superior imputation fidelity/utilities across benchmarks were reported, with released code and slides clarifying design choices (simple masking strategy, strong performance as the missing ratio grew) [14]. These properties make Remasker an appropriate "modern DL" point of comparison alongside DAEs in our study.

In summary, classical machine-learning methods provide strong, efficient models that can operate at high speed. DL models offer higher-capacity imputation but come with stability and computational limitations, and they typically require substantial training data. When such data and careful training are available, DL methods achieve state-of-the-art accuracy. In contrast, RL-based imputation remains relatively under-explored, particularly for streaming IoT time-series data characterized by small sample sizes and block-missing patterns. To address this gap, we formulate missing-data imputation as a sequential decision-making problem, design an IoT-specific state-action-reward structure, and conduct a systematic comparison of tabular Q-learning and Deep Q-Learning within a unified framework. The proposed approach is evaluated using MSE, MAE, and R^2 to assess both accuracy and robustness under realistic IoT data conditions.

III. Proposed Methods

A. Overview

RL provides a learning framework through interaction: an agent repeatedly observes the current situation of the environment, chooses an action, receives a numerical response, and moves to a new situation. Over time, the agent defines and then refines a policy, a mapping from observed situations to actions taken, so that the long-term response is maximized. Two internal estimates direct this process. State-value functions provide a measure of how desirable a situation is, while action-value functions estimate the effectiveness of taking a specific action in a particular situation. Learning algorithms such as Q-learning adjust the estimates by comparing the predicted results with those observed after taking action. Modern RL combines these approaches with deep neural networks to provide deep RL, which can provide value functions and policies directly from high-dimensional data. The volatility that can arise when learning and decision-making take place simultaneously is substantially reduced by using stabilizing techniques such as experience replay, target networks, and the ε-greedy algorithm. Because RL learns directly from the interactions rather than static examples, it is well-suited to problems such as IoT data imputation, where the agent must cope online with varying streaming inputs, noisy feedback, and changing sensor conditions.

In this study, the authors develop a model that incorporates three hyperparameterized methods to enhance imputation accuracy. It integrates two variants of Q-Table-based RL and one Deep Q-RL approach. The Q-Table RL is a model-free algorithm that uses tabular representation to store and update action-state values (Q-values). These values guide the agent in selecting optimal actions within a discrete environment based on a defined reward function. Although having the benefits of being simple and interpretable, Q-Table methods are limited in scalability when the state-action space becomes large. To address this, the model includes a second stage with enhancements in exploration and value generalization. In contrast, DQL extends the traditional Q-learning paradigm by employing a deep neural network to approximate the Q-function. This enables the agent to cope with high-dimensional or continuous state spaces more effectively. Combining the three methods, this model expects to provide a more accurate and robust imputation strategy than traditional techniques.

B. Data Preprocessing and Missing Data Synthesizing

In this study, the authors divided the Air-Quality dataset (UCI ID 360) from the UCI Machine Learning Repository [15, 16] for training and testing the proposed algorithms and the baseline methods for data imputation. The selected data source is one of the most popular benchmark resources in the IoT weather domain. It contains recorded responses from a gas multisensory device that was deployed in a field in an Italian city. Hourly average sensor readings are provided together with reference gas-concentration measurements obtained from a certified analyzer. In total, the dataset comprises 9,357 hourly averaged instances collected by an array of five metal-oxide chemical sensors embedded in the air-quality multisensory device, as shown in Table I. The device was positioned at road level in a heavily polluted urban area. Data were gathered continuously for one full year, from March 2004 to February 2005, yielding the longest freely available recording of on-field air-quality chemical sensor responses. The sensing unit comprised five metal-oxide (MOX) chemical sensors, each nominally tuned to specific target gases, along with environmental measurements. The dataset includes both the raw sensor responses (PT08.S1–PT08.S5) and reference gas-concentration measurements obtained from a co-located certified analyzer. The measured and reference parameters are summarized in Table I.

The dataset was split into training and validation sets, comprising 80% of the data, and a testing set comprising the remaining 20%. The authors used this split to ensure a large amount of data for training and validation, enabling better generalization and reducing the risk of overfitting, while the remaining 20% contained representative features sufficient for reliable testing.

The data preprocessing pipeline begins by retrieving and reviewing the dataset's metadata, such as source, measurement period, and definitions of variables, followed by generating summary statistics and data-type profiles to verify completeness and to detect potential anomalies. The column providing the Date is deconstructed from the original MM/DD/YYYY string format into a suitable datetime object, enabling the earliest and latest sampling times to be recorded for temporal consistency checks. All other numerical attributes (i.e., columns after the first two identifier fields) are rescaled to the interval $[0, 1]$ using a Min-Max transformation to ensure comparable magnitudes to

accelerate model convergence. Finally, two comma-separated value (CSV) files are generated: one preserves the raw measurements as collected, and the other stores the normalized version to be supplied to the learning algorithm. This procedure ensures that downstream analysis uses a clean, temporally aligned, and scale-independent dataset but retains access to the original records for auditing.

TABLE I. DATASET DESCRIPTION [15, 16]

Attribute	Description	Unit
Date	Recording date	DD/MM/YYYY
Time	Recording time	HH.MM.SS
CO	True hourly averaged Carbon Monoxide (CO) concentration	mg/m³
PT08.S1 (Tin Oxide)	Sensor response nominally targeted to CO	arbitrary unit
NMHC	True hourly averaged Non-Methane Hydrocarbons (NMHC) concentration	µg/m³
C_6H_6	True hourly averaged Benzene (C_6H_6) concentration	µg/m³
PT08.S2 (Titania)	Sensor response nominally targeted to NMHC	arbitrary unit
NO_x	True hourly averaged Nitrogen Oxides (NO_x) concentration	ppb
PT08.S3 (Tungsten Oxide)	Sensor response nominally targeted to NO_x	arbitrary unit
NO_2	True hourly averaged Nitrogen Dioxide (NO_2) concentration	µg/m³
PT08.S4 (Tungsten Oxide)	Sensor response nominally targeted to NO_2	arbitrary unit
PT08.S5 (Indium Oxide)	Sensor response nominally targeted to Ozone (O_3)	arbitrary unit
T	Temperature (T)	°C
RH	Relative Humidity (RH)	%
AH	Absolute Humidity (AH)	g/m³

All experiments were conducted using a single missingness procedure: entries in the numeric columns were masked independently with a fixed probability of 0.2, applied separately to the training, validation, and test partitions. This masking of elements generated random gaps across the dataset, guaranteeing that missing positions varied across folds and splits in the data. Thus, model training and validation losses reflected the reconstruction of the missing values, and final test metrics quantified performance on a retained set subjected to the same masking process (see Algorithm 1). With all models evaluated consistently against this single, fixed masking strategy and under the given setting (0.2). Algorithm 1 describes the proposed procedure for synthesizing missing data at a fixed rate across training, validation, and test splits. Starting from a complete dataset, the data are first divided into training/validation and test sets. During the cross-validation phase, a fixed fraction of values is randomly masked as missing in the numeric columns of both training and validation sets using a uniform random process, ensuring consistent missingness across folds. The imputation agent is trained on the masked training data and validated against the original uncorrupted data. In the final phase, the same missingness strategy is applied to the combined training/validation set and the test set, after which the imputation model is trained and evaluated. This procedure ensures a controlled, reproducible, and unbiased evaluation of imputation performance under fixed-rate missing-data conditions.

Algorithm 1 shows the proposed missing data synthesizing via a fixed-rate missingness for train, validation and test splits.

Algorithm 1: Proposed Missing Data Synthesizing Method

```
Start Algorithm
Initialize complete dataset D
Define numeric column set C
Define missing fraction p = 0.20
Define number of CV folds k
Randomly split D into train/validation set (D_tv) and test set
(D_test) with 80 % / 20 % ratio
# Cross-validation phase
For each fold f in K-Fold(D_tv, k) do
|  Split D_tv into training set (D_train) and validation set (D_val)
for fold f
|  Create D_train_missing ← copy of D_train
|  Create D_val_missing ← copy of D_val
|  For each column c ∈ C do
|  |  For each row r in D_train_missing do
|  |  |  u ← Uniform(0, 1)
|  |  |  If u < p then
|  |  |  |  D_train_missing[r, c] ← NaN
|  |  |  End If
|  |  End For
|  |  Repeat the same masking process for D_val_missing
|  End For
|  Train imputation agent on (D_train_missing, D_train) and
validate on (D_val_missing, D_val)
End For
# Final test phase
Create D_tv_missing ← copy of D_tv
Create D_test_missing ← copy of D_test
For each column c ∈ C do
|  For each row r in D_tv_missing do
|  |  u ← Uniform(0, 1)
|  |  If u < p then
|  |  |  D_tv_missing[r, c] ← NaN
|  |  End If
|  End For
|  Repeat the same masking process for D_test_missing
End For
Train final imputation agent on (D_tv_missing, D_tv) and evaluate
on (D_test_missing, D_test)
End Algorithm
```

C. Proposed Models' Architecture

1) Reinforcement learning based on Q table: Fig. 2 contrasts the end-to-end workflows for the two generations of our reinforcement learning imputer. Version 1 (V1) begins by injecting a user-specified ratio of synthetic gaps into the training fold and then iterates column-by-column inside the dashed box the Column Imputation Agent V1 represents each decision state solely by the row index of the missing value; contextual cues such as neighboring readings remain implicit. Two baseline actions are always available to the ε-greedy

selector: the KNN fill and a feature-statistics fill (mean, median or standard-deviation offset), while the remaining actions correspond to previously learned Q-values for that index. After an action is chosen, the reward is calculated as the negative absolute error normalized by the column's standard deviation; this signal updates the Q(state, action) table and the per-epoch MSE log. Once all gaps in the train-and-validation partitions are imputed, fold-level metrics are recorded and the agent proceeds to the next column or cross-validation split, ultimately producing a single Q-table per feature that is reused to fill the held-out test set [see Fig. 2(a)].

Version 2 (V2) retains the outer train, validation, and test scaffold, but tackles three limitations observed in V1. First, it discretizes the numeric context: each previous value is binned, and the joint bin index becomes the state, dramatically shrinking and regularizing the state space. Second, it replaces the absolute-error reward with a squared-error signal and logs both training and validation MSE curves, enabling early stopping and hyper-parameter tuning. Third, ε in the ε-greedy policy decays across epochs, allowing aggressive exploration early on and more deterministic exploitation later. Internally, four Q-tables are maintained (one per context bin), each updated with the same loop of reward calculation and value iteration. During inference, the agent consults the appropriate Q-table for the discretized context of every missing cell, producing fills that are both context-aware and data-driven [see Fig. 2(b)].

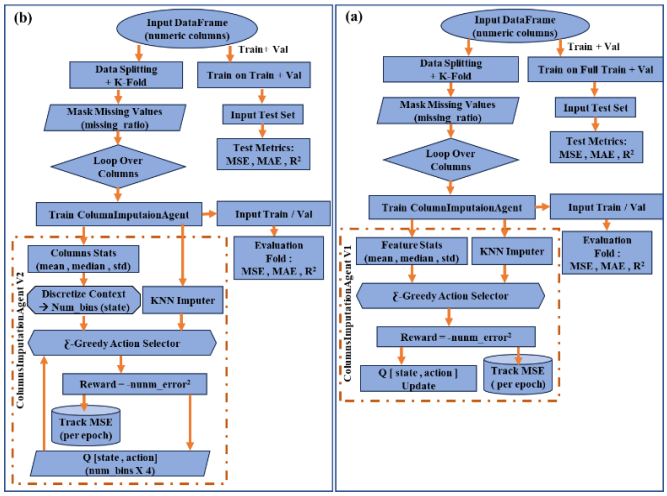

Fig. 2. The model structure of the proposed Q table methods.

2) Reinforcement learning based on deep Q: The proposed imputation framework employs a Deep Q-Learning (DQN) agent designed to learn optimal imputation strategies through interaction with an environment representing missing-value patterns. At each environment step, the agent observes a scalarized state s corresponding to the normalized position of the missing entry within a column and selects an action $a \in \{mean, median, random, knn\}$ using an ε-greedy policy. The environment then returns a reward r proportional to the negative normalized squared error between the imputed value and the ground truth. The next state s' reflects the subsequent missing entry index. This transition tuple (s, a, r, s') represents a single reinforcement learning experience used to update the DQN model. To enhance stability and sample efficiency, training is conducted using a batched replay mechanism. Transitions accumulated during interaction are stored in a replay buffer of finite capacity K. Instead of performing one update per sample, we repeatedly sample B mini-batches from the buffer and execute U stochastic gradient descent updates for each, thereby increasing the effective signal-to-noise ratio of the temporal-difference (TD) gradients. This multi-batch, multi-update schedule mitigates variance and reduces dependency on fresh data collection, an essential property when computational or data-access costs are constrained. For each sampled transition iii, a bootstrap target is computed as: $y_i = r_i + \gamma \max_{a'} Q_\theta - (s_i', a')$ where γ is the discount factor and θ^- denotes the parameters of a slowly updated target network. The online network parameters θ are optimized to minimize the mean-squared TD error: $L(\theta) = \frac{1}{B}. \sum_i (y_i - Q_\theta (s_i, a_i))^2$ using the Adam optimizer with learning rate α. Following each outer training epoch, both the exploration rate ϵ and the coefficient for target update τ decay according to defined schedules: $\theta^- \leftarrow (1-\tau)\theta^- + \tau\theta$ This ensures gradual policy refinement and stable convergence.

The Q-Network architecture (Fig. 2) is a compact three-layer feedforward model realized in PyTorch: an input normalization layer, a fully connected layer with 64 ReLU-activated neurons, and a linear output layer producing four Q-values corresponding to the available imputation actions. This lightweight design enables efficient on-device training and inference under limited memory and computation constraints. Collectively, the integration of replay memory, target-network stabilization, and multi-update training yields a robust Deep Q-Learning agent capable of acquiring context-adaptive imputation strategies for heterogeneous data distributions. The overall training pipeline, from dataset preparation and K-fold cross-validation to column-wise DQN training and test-time inference, is shown in Fig. 3.

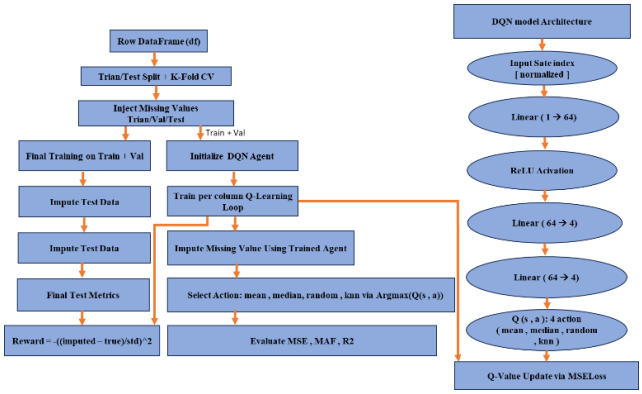

Fig. 3. The model structure of the proposed deep Q method.

3) Proposed simulation for RL environment: The authors developed an interactive grid-based simulation using the Pygame Library, to illustrate how a RL agent might traverse an air-quality dataset and impute missing values in real time (see Algorithm 2). After loading a CSV file that contains numerical features with artificially injected gaps, the data were

normalized to the [0,1] range and mapped onto a 2-D grid whose rows and columns correspond to samples and features, respectively (see Fig. 4). Each cell is 100 px square: filled observations appear in blue, missing entries in red, and their numeric contents are over-printed "NA" for clarity. The green agent starts at the upper-left corner [0,0] and can be moved by the user using the arrow keys. When the agent is positioned on a missing cell pressing the space bar triggers an "imputation action". In this demonstrator the action is a random draw from [0,1], the ground-truth value is likewise mocked, and the reward is the negative absolute error, these values are stored in a simple Q-table keyed by grid location. Although rudimentary, the framework visualizes the essential RL loop: state perception (color-coded grid), action selection (move or impute), reward feedback, and value-function update, all occurring at 30 FPS on an 800 × 600 canvas. This educational tool provides an accessible first step towards integrating and debugging more sophisticated, neural-based RL strategies for data-gap filling on edge-generated sensor streams. Algorithm 2 shows how the pseudocode outlines the full flow: loading and normalizing a dataset with missing values, drawing a color-coded grid, letting the user navigate a green "agent" with arrow keys, and pressing the space bar to impute and score missing entries. The Q-table records per-cell rewards, providing a scaffold for replacing random placeholders with a true reinforcement-learning policy at a later stage.

Fig. 4. RL Simulation for data imputation.

Algorithm 2 shows the pseudocode for Pygame RL-Style Imputation Simulator.

Algorithm 2: Pygame RL-style Imputation Simulator

```
Start Algorithm
If D contains no NaN then Raise error "dataset complete"
Normalize D to [0, 1]
rows ← number of samples in D
cols ← number of features in D
agent_pos ← [0, 0]          # row, col
Q_table ← ∅                 # (row,col) → reward
running ← True
While running do
|   Clear WINDOW with BACKGROUND_COLOUR
|   For i = 0 … rows−1 do
|   |   For j = 0 … cols−1 do
|   |   |   x ← j × CELL_SIZE; y ← i × CELL_SIZE
|   |   |   If D[i,j] is NaN then
|   |   |   |   color ← MISSING_COLOUR; label←"NA"
|   |   |   Else
|   |   |   |   color ← FILLED_COLOUR; label ←
|   |   |   |   round(D[i,j],2)
|   |   |   End If
|   |   |   Draw filled rectangle (x, y, CELL_SIZE, CELL_SIZE)
|   |   |   with color
|   |   |   Draw border rectangle with GRID_COLOUR
|   |   |   Render label in FONT_COLOUR at (x+10, y+10)
|   |   End For
|   End For
|   agent_x ← agent_pos.col × CELL_SIZE
|   agent_y ← agent_pos.row × CELL_SIZE
|   Draw rectangle (agent_x, agent_y, CELL_SIZE, CELL_SIZE)
|   in AGENT_COLOUR
|   For each EVENT in pygame.event.get() do
|   |   If EVENT.type = QUIT then
|   |   |   running ← False
|   |   Else If EVENT.type = KEYDOWN then
|   |   |   Case EVENT.key of
|   |   |   |   UP: If agent_pos.row > 0 then agent_pos.row −= 1
|   |   |   |   DOWN: If agent_pos.row < rows−1 then
|   |   |   |   agent_pos.row += 1
|   |   |   |   LEFT: If agent_pos.col > 0 then agent_pos.col −= 1
|   |   |   |   RIGHT: If agent_pos.col < cols−1 then
|   |   |   |   agent_pos.col += 1
|   |   |   |   SPACE:
|   |   |   |   |   i ← agent_pos.row; j ← agent_pos.col
|   |   |   |   |   If D[i,j] is NaN then
|   |   |   |   |   |   true_val←Uniform(0,1)   # placeholder
|   |   |   |   |   |   imput_val←Uniform(0,1)   # placeholder
|   |   |   |   |   |   D[i,j]←imput_val
|   |   |   |   |   |   reward← − |true_val − imput_val|
|   |   |   |   |   |   Q_table[(i,j)] ← reward
|   |   |   |   |   End If
|   |   |   |   End Case
|   |   End If
|   End For
|   Update WINDOW (pygame.display.flip())
|   Wait so frame-rate = 30 FPS (clock.tick(30))
End While
pygame.quit()
End Algorithm
```

IV. Results and Discussion

A. Tooling

The algorithms were developed in Python and used the Keras, Matplotlib, NumPy, Pandas, PyTorch, Scikit-learn, and TensorFlow data processing and ML libraries. Keras provides a user-friendly, high-level neural-network API compatible with both CPUs and GPUs. Matplotlib provides data visualization, allowing patterns to be displayed as charts and plots. NumPy offers a large collection of high-level mathematical functions able to support efficient operations on multi-dimensional arrays and matrices. Pandas provides data analysis and preprocessing, high-level data structures and a rich set of tools for filtering, combining, and grouping datasets before training. PyTorch is an

extensive open-source software useful for natural language processing (NLP) and computer vision. Scikit-learn provides a wide range of supervised and unsupervised ML algorithms for data mining and analysis. TensorFlow provides high-performance computation for training and operating deep-learning models.

The model was trained on the Google Colab cloud platform, which provides a shared high-performance computing (HPC) environment. The model training was carried out using two virtual CPUs, with specs Intel Xeon, family 6, model 79, clocked at 2.20 GHz with a 56,320 kB cache, alongside an NVIDIA T4 graphical processing unit (GPU) and 52 GB of random-access memory (RAM). The model inference and testing were executed on a local personal computer (PC) equipped with a 6th generation Intel Core i7 processor (2.6-2.8 GHz), an integrated Intel HD 520 GPU, 32 GB of RAM, and a 1 TB solid-state drive (SSD).

B. Data Preprocessing and Missing Data Synthesizing

In Fig. 5, the histogram of KDE shows panels depicting the empirical distributions of seven key features from the UCI Air-Quality dataset. All are gases ground truth (GT), CO(GT), NOx(GT), NO2(GT) and C6H6(GT), and all share a clear spike at -200 units that marks the dataset's built-in sentinel for missing or invalid sensor readings, and a pronounced positive skew caused by occasional pollution surges.

After removing the -200 flags, CO shows a very narrow peak while NOx displays a long right tail extending beyond 1 000 µg m⁻³, whereas NO2 peaks much more narrowly around 140 µg m⁻³. Benzene (C6H6) shows a compact distribution around 7 µg m⁻³ with a slight positive skew but virtually no extreme outliers, reflecting the lower variability of this volatile organic compound. In contrast, the meteorological variables exhibit near-Gaussian behavior once the sentinel values are discounted: temperature (T) clusters tightly between 15 °C and 30 °C, relative humidity (RH) is centered near 70 %, and absolute humidity (AH) forms an even tighter bell around 0.035 kg m⁻³. These contrasting shapes emphasize why a one-size-fits-all imputation rule is inadequate: heavy-tailed, zero-inflated gas measurements demand context-aware reconstruction, whereas smoother climatic variables can tolerate simpler statistical fillers.

Fig. 6 presents the pair-wise relationships among the five metal-oxide sensor channels embedded in the Air-Quality monitoring unit for CO, NMHC, NOx, NO2 and O3. Each panel shows a plot of one sensor's raw resistance (in ADC counts) against each of the others, revealing several instructive patterns. First, the tight, positively sloped clouds produced by linking PT08.S1(CO) with PT08.S2(NMHC) and PT08.S4(NO_2) indicate strong cross-sensitivities: when the CO sensor peaks, the NMHC and NO_2 channels rise almost proportionally, a hallmark of overlapping gas responses in metal-oxide arrays. Second, PT08.S3(NOx) behaves quite differently; its relationships with the other four channels produce a downward-curving trajectory, signifying an inverse, non-linear dependence due to its specific redox reaction mechanism. Third, every scatter plot contains a dense cluster at the origin, corresponding to the dataset's sentinel value (-200 ADC), which flags missing or invalid readings; because these artefacts sit far from the genuine signal manifold, any imputation model must avoid naively averaging across them. Finally, the diagonal panels showing lines remind us that each sensor reading is highly self-correlated, providing an upper bound for imputation accuracy.

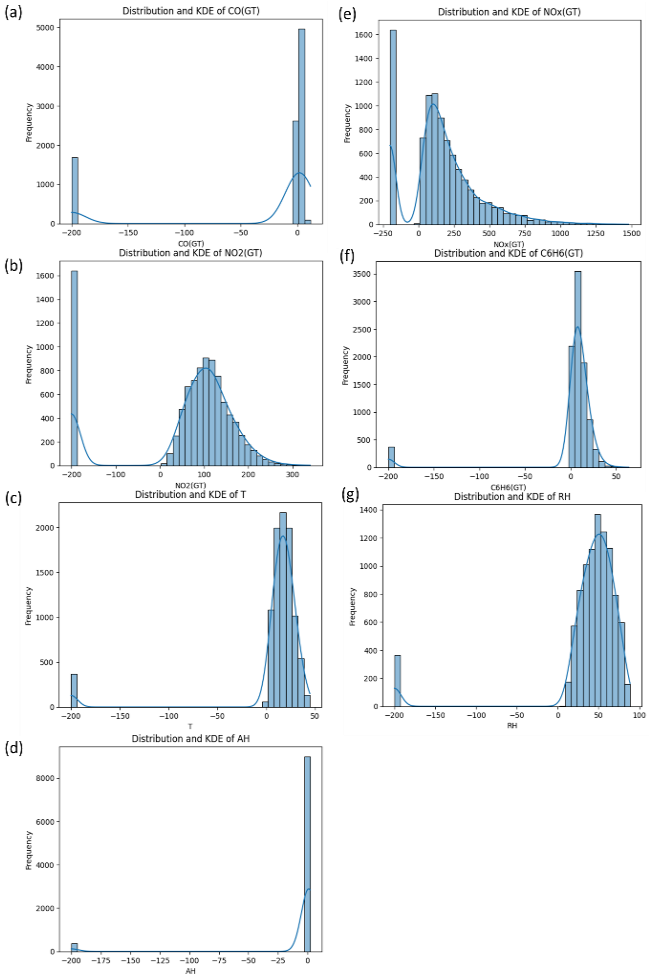

Fig. 5. Distributions and kernel density estimates of the selected Air Quality features: (a) CO(GT), (b) NOx(GT), (c) NO2(GT), (d) C6H6(GT), (e) temperature (T), (f) relative humidity (RH), and (g) absolute humidity (AH). The spike at -200 denotes the sensor-defined missing-value indicator, while the skewed shapes and long right tails, particularly for gaseous pollutants, reflect episodic concentration spikes and heterogeneous data characteristics.

PT08.S5 (Indium-oxide, nominally O_3) exhibits curvilinear, mostly inverse relationships with PT08.S3(NOx) and PT08.S4(NO_2), consistent with oxidizing reducing cross-sensitivities typical of metal-oxide sensor arrays. Its scatter plots with PT08.S1(CO) and PT08.S2(NMHC) appear weaker and more dispersed, indicating lower direct coupling and stronger modulation by environmental covariates such as temperature (T), relative humidity (RH), and absolute humidity (AH). The same -200 ADC sentinel cluster is also visible for PT08.S5 and must be masked prior to training or imputation to prevent distortion of the learned data manifold. For imputation tasks, PT08.S5 benefits particularly from non-linear models that can exploit its monotonic segments with NOx and NO_2 while conditioning on meteorological parameters, rather than relying on simple linear correlations.

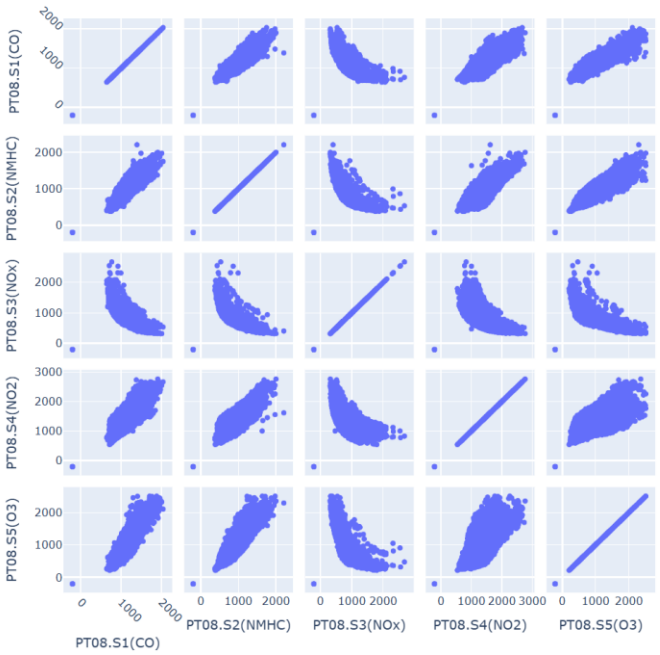

Fig. 6. Pairwise scatter-matrix of the five metal-oxide sensor channels (PT08.S1–S5). Blue dots represent valid observations.

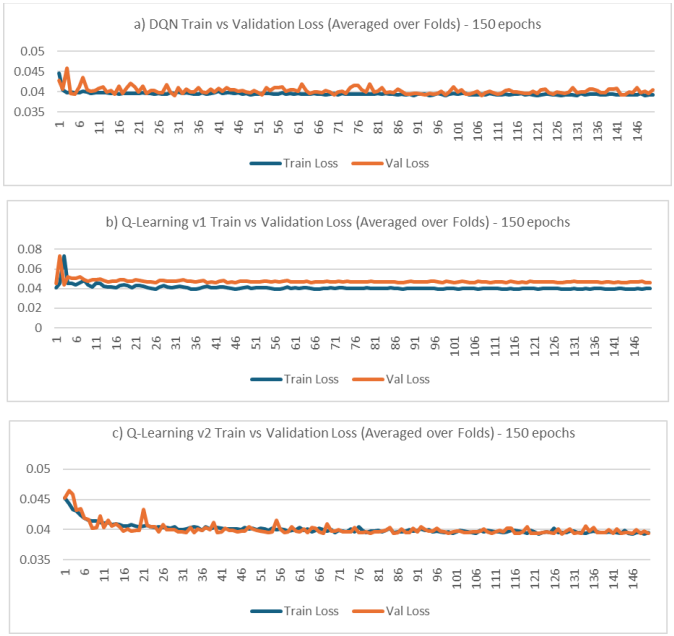

Fig. 7. Training vs. validation loss (averaged over folds) across 150 epochs for the proposed RL-based imputers: (a) Deep Q-Network (DQN), (b) Q-Learning v1, and (c) Q-Learning v2.

Third, every scatterplot contains a dense cluster at the origin, corresponding to the dataset's sentinel value (-200 ADC), which flags missing or invalid readings; because these artefacts sit far from the genuine signal manifold, any imputation model must avoid naively averaging across them. Finally, the diagonal panels showing lines remind us that each sensor reading is highly self-correlated, providing an upper bound for imputation accuracy.

C. Models Training and Validation

Fig. 7 presents the relative training–validation loss dynamics of the three RL approaches over 150 epochs (averaged across folds). In Fig. 7(a), the DQN model exhibits low overall loss values, with training and validation curves tracking closely after an initial period of fluctuations. The validation loss shows more pronounced oscillations than the training loss but without a continuing divergence, suggesting stable learning with only mild noise-induced variance on unseen data. Fig. 7(b) presents Q-Learning V1, where, after an initial period of relatively few epochs, the training loss steadily decreases towards an MSE value of 0.040 while the validation loss remains consistently higher at an MSE value of about 0.045. This persistent difference implies a degree of overfitting, where the learned policy better fits the training environment than unseen samples. In Fig. 7(c), Q-Learning V2 achieves very close alignment between training and validation losses after an initial decrease in value. Both curves stabilize at an MSE value between 0.039 and 0.040 with minimal oscillation, indicating strong generalization and absence of significant overfitting. Overall, these trends demonstrate that while all three models converge to low loss values, Q-Learning V2 maintaining the most consistent train–validation alignment, DQN follows closely with small validation variance, and Q-Learning V1 demonstrates a modest but persistent generalization gap.

Table II shows the comparison between the proposed algorithms and the state-of-the-art methods. In particular, Training length has different effects on the three RL approaches. For QL Version 1 (V1), extending training from 10 to 150 epochs steadily lowers the mean-squared error from 0.0098 to 0.0092, and raises R^2 from 0.7655 to 0.7772, with only a slight improvement in mean-absolute error from 0.0269 to 0.0204. After about 50 epochs, gains taper off, indicating a performance plateau. QL Version 2 (V2) behaves differently. At 10-epochs this model reaches its lowest MSE value (0.00785) and the highest R^2 (0.8041). Further training slightly worsens both metrics, suggesting mild over-fitting once the dominant patterns have been captured. DQL peaks at 50 epochs with an MSE of 0.0074, R^2 of 0.8053, and MAE of 0.0225, after which there is a modest decline at 150 epochs, indicating stability with some over-fitting. Comparing the RL models reveals a trade-off: V2 and DQN achieve lower MSE and stronger R^2, with better suppression of extreme deviations, but at the cost of a higher MAE. On the other hand, V1 delivers more uniform point-wise accuracy. Classical baselines can offer useful reference points and we see KNN provides a balanced but unremarkable performance (MSE = 0.0081, R^2 = 0.7822), whereas clustering and autoencoder underperform markedly, and the simplest imputations (mean, median, most-frequent) produce large errors with poor R^2, confirming their unsuitability for continuous environmental streams.

Considering all three metrics: QL V1 at 150 epochs offers the best balance, with the smallest MAE, a respectable MSE, and solid R^2; QL V2 at 10 epochs excels at limiting outliers shown in its superior MSE; and DQN at 50 epochs combines strong R^2 with low error and minimal over-fitting. Among non-RL methods, only KNN approaches RL-level performance but fails to surpass it. Consequently, the authors propose the use of RL agents, especially V1-150 for consistent accuracy, V2-10 for

outlier suppression, and DQN-50 for balanced generalization, as the most practical solutions for real-time IoT imputation, delivering strong accuracy without prohibitive training demands on edge hardware.

This early optimum of Q-Table V2 can be attributed to its internal architectural modifications compared with V1. As detailed in Fig. 2(b) (Section III), V2 discretizes the numeric context into bins, reducing the size of the state-action space and allowing the agent to learn stable Q-values within very few epochs. It also employs a squared-error reward function instead of the normalized absolute-error reward used in V1. This squared-error formulation amplifies large deviations during the initial learning phase, enabling rapid minimization of MSE but also increasing the likelihood of mild over-adjustment once the major error patterns have been corrected. Further, V2 produces a decay in the exploration rate (ε) within the ε-greedy policy with an increase in the number of epochs, which encourages broad exploration during the first few epochs but quickly transitions to exploitation. Once the policy stabilizes, subsequent updates contribute little additional improvement and can even marginally degrade test performance, reflecting controlled over-specialization rather than instability.

In contrast, V1 relies on a continuous state representation and a fixed ε, which promotes slower but steadier convergence and greater resilience to over-fitting. Consequently, V1 benefits from extended training up to 150 epochs, progressively refining its policy and reducing MAE without significant loss in generalization. The different convergence behaviors of V1 and V2 stem directly from their design philosophies; V2 prioritizes fast convergence and computational efficiency, making it suitable for lightweight IoT or edge deployments, whereas V1 favors gradual learning and robust generalization over longer training cycles.

These findings confirm that the observed peak performance of V2 at 10 epochs is a result of its architecture rather than an anomaly. The early convergence and subsequent plateau reflect the intended trade-off between rapid policy stabilization and marginal over-fitting within a highly compact state space.

The behavior of the Deep-Q Network (DQN) supports the interpretation that the number of training epochs interacts with model capacity. Unlike the tabular versions, DQN uses a neural function approximator to estimate Q-values, enabling smoother generalization across unseen states but introducing a higher sensitivity to over-training. At around 50 epochs, the network reaches an optimal balance between exploration and convergence, achieving the lowest MSE (0.0074) and highest R^2 (0.8053) while maintaining a relatively low MAE (0.0225). Beyond 50 epochs, continued training leads to a slight degradation in performance until, at 150 epochs, we have MSE = 0.0084 and R^2 = 0.7855, primarily due to over-fitting of the Q-value function and reduced stochastic exploration as the policy becomes deterministic. Thus, 50 epochs with this model represent the point of maximum generalization; the network has learned sufficient value structure to minimize prediction error without memorizing the training trajectories. This pattern is consistent with typical DQN behavior, where moderate training durations combined with early stopping yield the most stable convergence of the value function and the best trade-off between bias and variance.

TABLE II. COMPARISON BETWEEN THE PROPOSED AND STATE-OF-THE-ART METHODS

Method	Epochs	MSE	MAE	R^2
Proposed Q Table V1	10	0.0098	0.0207	0.7655
Proposed Q Table V1	50	0.0098	0.0209	0.7693
Proposed Q Table V1	150	**0.0092**	**0.0204**	**0.7772**
Proposed Q Table V2	10	**0.0078**	**0.0228**	**0.8041**
Proposed Q Table V2	50	0.0084	0.0227	0.7883
Proposed Q Table V2	150	0.0081	0.0231	0.8014
Proposed Deep Q	10	0.0077	0.0237	0.7805
Proposed Deep Q	50	**0.0074**	**0.0225**	**0.8053**
Proposed Deep Q	150	0.0084	0.0234	0.7855
Autoencoder [13]	-	0.0289	0.0923	0.2738
KNN [6]	-	0.0081	0.0239	0.7822
Clustering [7]	-	0.0362	0.1119	0.6828
Mean [9]	-	0.0675	0.1379	0.3987
Most Frequent [6]	-	0.1286	0.2095	-0.1271
Median [9]	-	0.0708	0.1350	0.3688

Across all methods, the Deep-Q model at 50 epochs delivers the best MSE (0.0074) and highest R^2 (0.8053), while the strongest classical baseline KNN has an MSE = 0.0081 and R^2 = 0.7822. This represents a ~9% reduction in MSE and a +2.31 percentage-point increase in R^2. For MAE, the best performer is Q-Table V1 at 150 epochs (0.0204), improving on KNN (0.0239) by ~14.5%. All RL variants clearly outperform the simple statistical baselines and the tested Autoencoder, which lags substantially on all metrics.

If the application prioritizes calibration and variance-sensitive fidelity, such as forecasting or anomaly scoring downstream, Deep-Q-50 epochs are the best choice (lowest MSE, highest R^2). If robustness to outliers and absolute deviation matter more, such as thresholding deviations for alerts, Q-Table V1 150 epochs is preferable (best MAE, ~9.3% lower than Deep-Q-50). Q-Table V2 offers a middle ground with near-Deep-Q R^2 but slightly weaker MAE, which might be attractive when a simpler function approximation is desired.

The loss curves in Fig. 7 show rapid early convergence followed by small, stable train-validation gaps for all three RL variants, except for V2, where the curve stabilizes early due to fast policy saturation within the discretized state space, indicating controlled capacity and limited over-fitting. Deep-Q shows optimum values for each of MAE, MSE, and R^2 at 50 epochs rather than 150, suggesting that light early stopping or entropy/exploration annealing could further stabilize performance. For the Q-Table models, longer horizons consistently help MAE, supporting the view that policy refinement through continued exploration benefits absolute-error minimization.

D. Simulation

Fig. 8 presents successive frames from the Pygame-based simulation visualizer, illustrating a single decision cycle of the RL imputation agent. In the first frame [Fig. 8(a)], the agent (green square) is positioned next to a red "NA" cell representing a missing value, while previously imputed entries appear in blue with normalized magnitudes. Upon user action, the agent imputes a value of 0.66 and receives a small negative reward [Fig. 8(b)], reflecting a near-match to the hidden ground truth (0.65). In the subsequent frame [Fig. 8(c)], the imputed cell turns blue, confirming state update and storage in the replay buffer. This visual transition highlights real-time gap resolution and online learning, as rewards are logged to refine future actions, with repeated cycles progressively filling the grid and quantifying imputation accuracy interactively.

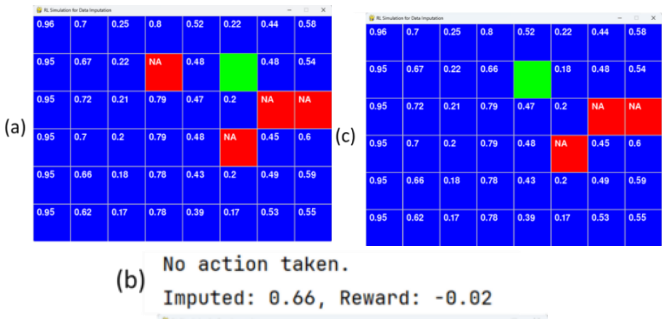

Fig. 8. Consecutive frames from the Pygame imputation simulator. The agent (green) selects an action on a red "NA" cell, where its ground truth was 0.65, fills it with the determined value (here 0.66), receives a small negative reward (-0.02), and updates the grid, which turns the cell blue to indicate imputation.

V. Conclusion and Future Work

This work addressed the continuing problem of missing values in edge-collected meteorological data streams in an IoT environment, caused by sensor faults, power interruptions, and communication noise, which degrade downstream analytics and time-critical decision support. To address this challenge, the authors have developed a lightweight reinforcement-learning imputation framework that formulates each gap as a sequential decision problem, learns directly from sparsely populated streams by utilizing cross-sensor temporal correlations, executes within the compute and energy envelopes of embedded IoT nodes, and is integrated with an interactive visual simulator implemented in Pygame. Beyond reporting performance gains, this work demonstrates that modeling missing-data imputation as a sequential decision-making process constitutes a general and effective paradigm for IoT data restoration, rather than a purely static prediction task. Empirically, the RL approach yields consistent gains under both block-missing and random-missing regimes: relative to non-RL baselines, the best configuration delivers 8.6 to 94.2% lower MSE and 5.9 to 89.3% lower MAE, while lifting R^2 by +0.023 to +0.932; even against the strongest baseline (KNN), it achieves 8.6%/5.9% error reductions with +0.023 R^2, underscoring robustness rather than overfitting. These results support the overarching contribution that sequential, policy-driven imputation can better adapt to dynamic and data-scarce IoT settings than other methods. This simulator allows real-time visualization of the agent's decision-making process, each state, action, and reward, facilitating rapid prototyping, policy debugging, and latency profiling during development. Future work will incorporate uncertainty-aware imputation, explore transfer and meta-learning for data-scarce stations, and optimize energy-efficient on-device adaptation, while hardening the edge stack through embedded profiling, model compression, and physics-informed simulator enhancements.

References

[1] I. Ficili, M. Giacobbe, G. Tricomi, and A. Puliafito, "From Sensors to Data Intelligence: Leveraging IoT, Cloud, and Edge Computing with AI," Sensors, vol. 25, no. 2025.

[2] A. Choudhary, "Internet of Things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions," Discover Internet of Things, vol. 4, no. 1, p. 31, 2024.

[3] A. Dauda, O. Flauzac, and F. Nolot, "A Survey on IoT Application Architectures," Sensors, vol. 24, no. 16, pp. 5320, 2024.

[4] S. Jäger, A. Allhorn, and F. Bießmann, "A Benchmark for Data Imputation Methods", Frontiers in Big Data, Original Research vol.4, 2021.

[5] M. Liu et al., "Handling missing values in healthcare data: A systematic review of deep learning-based imputation techniques," Artificial Intelligence in Medicine, vol. 142, p. 102587, 2023.

[6] P. Prakash, K. Street, S. Narayanan, B. A. Fernandez, Y. Shen, and C. Shu, "Benchmarking Machine Learning Missing Data Imputation Methods in Large-Scale Mental Health Survey Databases," medRxiv, 2024.

[7] C.-H. Cheng and S.-F. Huang, "A novel clustering-based purity and distance imputation for handling medical data with missing values," Soft Computing, vol. 25, no. 17, pp. 11781-11801, 2021.

[8] L. O. Joel, W. Doorsamy, and B. S. Paul, "A comparative study of imputation techniques for missing values in healthcare diagnostic datasets" Int Jou of Data Science and Analytics, vol.20, no.7, pp.6357-6373, 2025.

[9] J. Li et al., "Comparison of the effects of imputation methods for missing data in predictive modelling of cohort study datasets," (in eng), BMC Med Res Methodol, vol. 24, no. 1, p. 41, Feb 16 2024.

[10] X. Qin and et al. "Improved generative adversarial imputation networks for missing data," Applied Intelligence, vol. 54, no. 21, pp. 11068-11082, 2024.

[11] J. Hwang and D. Suh, "CC-GAIN: Clustering and classification-based generative adversarial imputation network for missing electricity consumption data imputation," Expert Systems with Applications, vol.255, p.124507, 2024.

[12] L. Zhang and et al. "Generative Adversarial Networks for Imputing Sparse Learning Performance," in Pattern Recognition, Cham, A. Antonacopoulos, S. Chaudhuri, R. Chellappa, C.-L. Liu, S. Bhattacharya, and U. Pal, Eds., 2025: Springer, pp. 381-396.

[13] N. T. Haridas, J. M. Sanchez-Bornot, P. L. McClean, and K. Wong-Lin, "Autoencoder imputation of missing heterogeneous data for Alzheimer's disease classification," (in eng), Healthc Technol Lett, vol. 11, no. 6, pp. 452-460, 2024.

[14] T. Du, L. Melis, and T. Wang, "Remasker: Imputing tabular data with masked autoencoding," presented at the International Conference on Learning Representations, Vienna, Austria, 2024, 2024

[15] S. Vito. Air quality, UCI Machine Learning Repository, 2008

[16] S. De Vito and et al., "On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario," Sensors and Actuators B: Chemical, vol. 129, no. 2, pp. 750-7