D2.2: MIRAI ML/AI Algorithms And Data-Mining Models Specification And Design

Work Package	WP2
Dissemination level	Public
Status	Final
Date	31/05/2022
Deliverable leader	Nicolás González-Deleito (Sirris)
Potential Contributors	Macq, Shayp, ISEP, UPORTO, Eliar, Enforma, BTH

Contributors

Name	Organization
Nicolás González-Deleito	Sirris
Sarah Klein	Sirris
Mustafa Çom	Eliar
Pedro Santos	ISEP
Ricardo Morla	U.Porto
Sreeraj Rajendran	Sirris
Sena Çağlar, Burak Ketmen	Enforma

Reviewers

Name	Organization
Joana Sousa	NOS
Sena Çağlar	Enforma

Document History

Date	Main changes	Name
08-03-2022	Initial draft	Nicolás González-Deleito
13-05-2022	Contributions to sections 3.2.3, 3.2.5, 3.4.1	Sarah Klein
17-05-2022	Document review	Nicolás González-Deleito
23-05-2022	Contribution to section 3.2.4	Sreeraj Rajendran
23-05-2022	Added conclusion	Nicolás González-Deleito
23-05-2022	Reviewed deliverable (first iteration)	Joana Sousa
30-05-2022	Reviewed deliverable (first iteration)	Sena Çağlar

Table of Contents

Abbreviations

ML	Machine Learning
UC	Use Case
UC1	Use case 1: Distributed renewable energy systems (UC owner: 3E)
UC2	Use case 2: Secure Internet provisioning (UC owner: NOS)
UC3	Use case 3: Traffic management (UC owner: Macq)
UC4	Use case 4: Water management (UC owner: Shayp)
UC5	Use case 5: Continuous auto configuration of industrial controllers at edge (UC owner: Eliar & Enforma)

1. Executive Summary

This deliverable describes the relevant techniques and models resulting from the research activities in T2.2 and T2.3. The results presented in this first iteration of this deliverable include the work done up to M18. A second (and final) iteration is planned at M24.

2. Introduction

The present deliverable provides a detailed description of the different machine learning and data mining techniques considered in MIRAI, and of the different distributed/composable decision making models for scaling computations horizontally and vertically in edge computing environments. While the former techniques are studied in T2.2, the latter are examined in T2.3. As such, this deliverable provides a detailed description of the techniques and models resulting from both tasks.

D2.2 builds upon D2.1, which provided an overview of the state-of-the-art techniques relevant to the 5 different use cases considered in the project. Based on a study of the requirements at the level of distributed intelligence for each of these 5 use cases, D2.1 identified 11 relevant state-of-the-art areas, organized in 4 different domains. These are summarized in the following table:

Domain	**State-of-the-art area**
Data preprocessing	Distributed and multi-modal data fusion at the edge
Computations at the edge	Prediction at the edge
	(Distributed) feature extraction at the edge
	Anomaly detection at the edge
	Context understanding at the edge
	Compression of time series data at the edge
Distributed AI	Federated learning
	Transfer learning
	Distributing computations among edge nodes
Security and privacy	Privacy-preserving learning techniques
	Secure data sharing

D2.2 is organized around these state-of-the-art areas and domains. For each area, the relevant techniques and models resulting from T2.2 and T2.3 and described in the corresponding subsection of section 3.

This first version of D2.2 describes the results up to M18. A second version of this deliverable will be issued at M24. The actual implementations of the described techniques and models are provided in D2.3.

3. Detailed descriptions of techniques and models

This section describes the relevant techniques and models resulting from T2.2 and T2.3, up to M18 and following the state-of-the-art domains and areas identified in D2.1, through the following template:

Technique/model name	*A one-line name for the technique/model*
Partner(s)	*The list of project partners working on this technique/model*
Context	*A brief description of the relevant context in which this technique/model is situated/applies*
Description	*A detailed description of the technique/model*
Status	*A short description of the current status in the context of MIRAI and of any current relevant limitation/constraint and future extension until M24*

3.1. Data preprocessing

3.1.1. Distributed and multi-modal data fusion at the edge

No relevant techniques/models are reported at M18 for this state-of-the-art area. This area will be covered at the second iteration of this deliverable, at M24.

3.2. Computations at the edge

3.2.1. Prediction at the edge

Technique/model name	The Holt-Winters seasonal method
Partner(s)	Eliar, Enforma
Context	Time series prediction for steam data
Description	The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations — one for the level **ℓ_t**, one for the trend **b_t**, and one for the seasonal component **s_t**, with corresponding smoothing parameters **α**, **$\beta*$** and **γ**. In our use case, the frequency of steam data is one second. Higher frequency time series often exhibit more complicated seasonal patterns. Data that are observed every second have minute seasonality. We see the frequency of seasonality in 40-minutes for steam data. The seasonal period is $m=2400$, and the appropriate unit of time for h is in seconds. There are two variations to this method that differ in the nature of the seasonal component. The additive method is preferred when the seasonal variations are roughly constant through the series, while the multiplicative method is preferred when the seasonal variations are changing proportional to the level of the series. The multiplicative method model was more effective at predicting steam data. The component form for the additive method is: $y_{t+h\|t}=\ell_t+hb_t+s_{t+h-m(k+1)}$ $\ell_t=\alpha(y_t-s_{t-m})+(1-\alpha)(\ell_{t-1}+b_{t-1})$ $b_t=\beta*(\ell_t-\ell_{t-1})+(1-\beta*)b_{t-1}$ $s_t=\gamma(y_t-\ell_{t-1}-b_{t-1})+(1-\gamma)s_{t-m}$

Status	Implemented in Python using statsmodels module, tested on historical data.

Technique/model name	**Random Forest regressor & sliding window for time series data**
Partner(s)	Eliar, Enforma
Context	Time series prediction for steam data
Description	Machine learning methods like deep learning can be used for time series forecasting. Given a sequence of numbers for a time series dataset, we can restructure the data to look like a supervised learning problem. We can do this by using previous time steps as input variables and use the next time step as the output variable. This method is called a sliding window, as the window of inputs and expected outputs is shifted forward through time to create new "samples" for a supervised learning model. We use the *shift()* function in Python Pandas to automatically create new framings of time series problems given the desired length of input and output sequences. All variates in the time series can be shifted forward or backward to create multivariate input and output sequences. This permits not only classical X -> y prediction, but also X -> Y where both input and output can be sequences. The function that transforms time series data takes three important arguments: • *data:* Sequence of observations as a list or 2D NumPy array. • *n_in:* Number of lag observations as input (X). Values may be between [1..len(data)]. • *n_out:* Number of observations as output (y). Values may be between [0..len(data)-1]. After transforming our historical time series data, we achieved successful results in the Random Forest Regressor model for prediction. However, other regression algorithms can be used after the sliding window process.
Status	Implemented in Python using scikit-learn library, tested on historical data.

3.2.2. (Distributed) feature extraction at the edge

Technique/model name	Least-squares-based parameter identification
Partner(s)	Eliar, Enforma
Context	The least-squares can be used for parameter identification of corresponding models by using real process data.
Description	To be able to identify the parameters of a pre-specified process model such as first order plus time delay (FOPTD), the least-squares can be utilized because it is easy to implement and ensures satisfactory identifications. Discrete time domain representation of the aforementioned process model is given as: $$y[k+1] = y[k] + T\,(-by[k] + au[k])$$ where $k,\ a,\ b,\ u,\ T$ and y represent the discrete time index, the model gain, the model time constant, the control input, the sampling period and the process output, respectively. a and b are identified by using the least-squares equations that are given as:

	$$A = T\begin{bmatrix}1 & -y_1\\ \vdots & \vdots \\ 1 & -y_{n-1}\end{bmatrix} \qquad c = \begin{bmatrix} y_2 - y_1 \\ \vdots \\ y_n - y_{n-1}\end{bmatrix}$$ $$\hat{x} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}c = \begin{bmatrix}a & b\end{bmatrix}^{\mathrm{T}}$$ These parameters will represent heating performances for UC5 and will be used for PID controller parameter tunings.
Status	Implemented in Python, tested and verified. Currently running in the local cloud.

Technique/model name	Controller performance assessments
Partner(s)	Eliar, Enforma
Context	Some statistical metrics and an exclusively designed metric are used to assess controller performance.
Description	To be able to assess controller performance, the use of some metrics is necessary. These metrics can be not only already defined statistical metrics but also exclusively designed metrics. Firstly, the manipulated variable of the control system is collected in an array as follows: $$y[k], k = 1,2,\dots,n$$ where k is the discrete time index and n is the total control time. Afterwards, corresponding metrics are calculated as follows: $$OV = \max(y[k] - SP[k])$$ $$MAE = \frac{\sum_{k=1}^{n}\lvert y[k] - SP[k]\rvert}{n}$$ $$MSE = \frac{\sum_{k=1}^{n}\lvert y[k] - SP[k]\rvert^2}{n}$$ $$S = 100\left(1 - \frac{\sum_{k=1}^{n}\lvert y[k] - SP[k]\rvert}{10n}\right)\%$$ where $SP[k], OV, MAE, MSE$ and S represent the set point at instant k, the overshoot, the mean absolute error, the mean squared error and the exclusively designed success percentage metric, respectively. These metrics will be used for controller performance assessments and PID controller parameter tunings.
Status	Implemented in Python, tested and verified. Currently running in the local cloud.

3.2.3. Anomaly detection at the edge

Technique/model name	Compression-based anomaly detection
Partner(s)	Shayp, Sirris
Context	The compression-based anomaly detection can be used for time series data with specific patterns that indicate the anomaly.
Description	The general approach identifies anomalies represented by specific patterns in univariate time series data by comparing two different compression results across a rolling time window. For this, one of the compression methods should lead to size inflation in case of an anomaly. For the leakage detection (UC4), the string lengths of a purely Fibonacci-encoded and of a combined run-length and Fibonacci-encoded water consumption pulse pattern are used (cf. section 3.2.5). In case of a

	leakage, the compression based on run-length encoding leads to a length inflation. By using a simple thresholding, a binary variable indicates a critical inflation and by that a leakage. This threshold can be building-type specific.
Status	Implemented in Python, tested on historical data. To be implemented in C for validation in edge device.

Technique/model name	Ensemble anomaly detection method
Partner(s)	U.Porto/ISEP, NOS
Context	Identify IoT and anomalous network traffic.
Description	An ensemble method (a collection of various 'weak classifiers') is used to identify anomalies in network traffic. The ensemble strategy is bagging (as opposed to boosting): the output of the weak classifiers is combined to produce a final prediction. The 'weak classifier' methods are: • Local Outlier Filter (LOF): Unsupervised anomaly detection method that computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors. • Support Vector Machine (SVM): Different linear classifiers can produce a multitude of boundaries to discriminate two classes. SVMs try to select a decision boundary for which the margin between data points of different classes is maximized. In our case, we are using a One-Class SVM. • Elliptic envelope: Creates an elliptical area around a given dataset. Values inside the envelope are considered normal data and anything outside the envelope is returned as outliers. This algorithm works best if data has a Gaussian distribution. The combination method is the majority rule.
Status	Implemented in Python, tested on publicly available datasets and datasets shared by NOS. Refinement continues.

3.2.4. Context understanding at the edge

Technique/model name	Re-identification of vehicles at the edge
Partner(s)	Macq, Sirris
Context	Identifying same vehicle across different cameras is an integral part of context understanding.
Description	Reidentification of vehicles across cameras is performed in two steps. Advanced deep learning models are used in both steps to achieve state-of-the-art performance. First, vehicles are detected from the camera feeds using a YOLO v4 model. At the second step, abstract features of the detected vehicles are generated using a Squeeze-and-Excitation Network with 101 layers. The model is pretrained on ImageNet for classification and further finetuned on the Cityflow dataset. The model is further adapted with early exits, at three different depths of the model, to improve its inference performance at the edge. The extracted features (512 features) from images are ranked after computing a dot product with the query image vectors. As only these abstract features

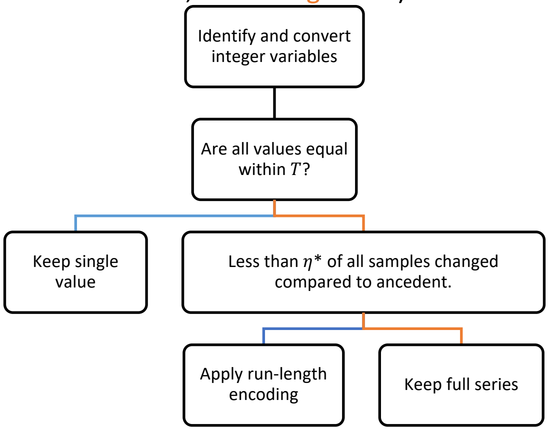
	leave the cameras, the reidentification model provides a basic guarantee to protect privacy of vehicle users.
Status	Implemented in Python using Pytorch, tested on both Cityflow and two-camera datasets from Macq. The YOLO and the reidentification models should be ported to Macq's edge device for further validation.

3.2.5. Compression of time series data at the edge

Technique/model name	Run-length encoding
Partner(s)	Sirris
Context	Encoding of univariate time series data.
Description	Compression method that is helpful for integer-based univariate time series data with fixed time step and long sequences without value changes. Instead of reporting a value v_i at each time t_i, the number n of sequential occurrences of the same value v is reported, such that a sequence $[v_1,\ v_1,\ v_1,\ v_1,\ v_1,\ v_2,\ v_2,\ v_2]$ becomes $[(5, v_1), (3, v_2)]$.
Status	Implemented in Python.

Technique/model name	Fibonacci encoding
Partner(s)	Sirris
Context	Encoding of univariate time series data.
Description	Compression method that is helpful for integer-based, non-negative, non-monotonous univariate time series data with fixed time step. Instead of reporting a value v_i at time t_i, the Fibonacci sequence is used to encode the values, similarly to a binary encoding with base 2. Zeckendorf's theorem says that non-zero integer values can be written uniquely with the "stopword" 11 at the end[1]. We define that for a value $n \geq 0$ the encoding $E(n+1) = E(f)$ with $f \geq 1$, in order to be able to encode the value "0" as well. Given the Fibonacci series 1, 1, 2, 3, 5, 8, 13, …, and respecting the end "stopword", the following sequence would be encoded as follows: [1, 0, 0, 6, 3, 0, 0, 10] = [1, 0, 0, (5 + 1), 3, 0, 0, (8 + 2)] (1) = ['011', '11', '11', '10011', '0011', '11', '11', '00111'] (2) = 00111111100110011111100111 (3) In step (1), all numbers that are not Fibonacci numbers, hence 6 and 10, are replaced by the sum of Fibonacci numbers, starting from the next smaller one. In step (2), for each Fibonacci number (e.g. 3) or combination of Fibonacci numbers (e.g. 5+1), use the binary encoding including the "stopword". Finally, in (3), concatenate all binary representations into one list of bits.
Status	Implemented in Python.

[1] J. Spiegel, P. Wira and G. Hermann, "A Comparative Experimental Study of Lossless Compression Algorithms for Enhancing Energy Efficiency in Smart Meters," 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), 2018, pp. 447-452, doi: 10.1109/INDIN.2018.8471921

Technique/model name	Automated time series compression selection
Partner(s)	3E, Sirris
Context	Compression of multi-variate time series data.
Description	For high-dimensional data with high temporal resolution (δt), there is no one-size-fits-all solution for compressing data. In our approach, the data is sent after a fixed time step $T \gg \delta t$. During T, for each dimension, we perform the following operations (for conditional branching, blue means true, and orange false):  Identify and convert integer variables Are all values equal within T? Keep single value Less than η* of all samples changed compared to ancedent. Apply run-length encoding Keep full series * with $0 < \eta < 1$. With these remaining values per dimension, build a JSON file instead of CSV file, in order to only minimize the amount of duplicated information.
Status	Implemented in Python, tested on historical data. On the example data provided by 3E, we reach a compression factor of approximately 13.

3.3. Distributed AI

3.3.1. Federated learning

No relevant techniques/models are reported at M18 for this state-of-the-art area. This area will be covered at the second iteration of this deliverable, at M24.

3.3.2. Transfer learning

No relevant techniques/models are reported at M18 for this state-of-the-art area. This area will be covered at the second iteration of this deliverable, at M24.

3.3.3. Distributing computations among edge nodes

Technique/model name	**Rule-based methods testing on distributed simulation**
Partner(s)	Eliar, Enforma
Context	PID fine-tuning using steam availability and the batch/process priority.
Description	The distribution of steam to the machines depending on steam availability is an important issue in the control of the textile dyeing process. As a result, the initial parameters need to be fine-tuned based on the critical situations with steam and process priorities. Using rule-based AI, this can be accomplished. However, since the real system in the textile factory can't be used to test the rules, the distributed simulation running on Docker was developed. Rule-based PID tuning methods assume a certain process response to obtain easy

	mathematical formulas that enable the tuning of a PID controller. We will combine steam data prediction on the edge and rule-based PID fine-tuning. The initial parameters will be updated for each machine as needed, using the best rules tested in the simulation.
Status	Implemented in Python using Docker.

3.4. Security and privacy

3.4.1. Privacy-preserving learning techniques

Technique/model name	Messaging with risk-sensitive random timing
Partner(s)	Shayp, Sirris
Context	Determination of messaging times for privacy and security-sensitive data.
Description	Saving energy and bandwidth of messaging and still alerting users as soon as possible in case of an anomaly, can lead to privacy and security leakages as water consumption patterns (UC4) can be deducted from the messaging schema itself. A possibility to overcome this is a random time messaging schema. In order to still alert the users as quickly as possible, an additional risk-sensitive layer can be added to the random timing. First, in case of an observed anomaly, a message is sent as soon as possible. In that case, the heuristic distribution derived from the random sending times changes in case many leakages occur. Hence, a curious attacker could derive from the shift in the distribution whether the building has a high risk of leakages. Hence, as a second step, the average sending time within a given time window is calculated and the distribution from which the sending time is drawn, is adapted accordingly. By this, a temporary bias to the higher sending times is introduced which shifts the heuristic distribution of random sending times back towards the desired distribution, shadowing the outlier in time caused by the alert. Like this, the overall heuristic distribution derived from the random sending times follows the same distribution as the imposed one without early alerting and without leakage.
Status	Implemented in Python, tested on historical data. To be implemented in C for validation in edge device.

3.4.2. Secure data sharing

No relevant techniques/models are reported at M18 for this state-of-the-art area. This area will be covered at the second iteration of this deliverable, at M24.

4. Conclusion

D2.2 provides a detailed description of the different machine learning and data mining techniques considered in MIRAI (resulting from T2.2), and of the different distributed/composable decision making models for scaling computations horizontally and vertically in edge computing environments (resulting from T2.3).

The present iteration of this deliverable presents the techniques and models having been (to a large extent) fully explored from a research point of view up to M18, and which are currently being translated and further validated in the context of the different project's use cases. These techniques and models cover 5 out of the 11 state-of-the-art areas identified in D2.1, based on which D2.2 is structured. Several other techniques and models, covering these and the remaining 6 areas, are currently being explored by the project partners. The results will be included in the final version of this deliverable, planned at M24.