Instituto Superior Técnico

Computação Inteligente / Applied Computational Intelligence

MEEC

Project 1 - Fuzzy Systems and Neural Networks, 1st Semester 2023/2024

Students:
Afonso Brito Caiado Correia Alemão | 96135 | afonso.alemao@tecnico.ulisboa.pt
Rui Pedro Canário Daniel | 96317 | ruipcdaniel@tecnico.ulisboa.pt

Group **9**

Teachers:
Nuno Horta, Joao Carvalho, Rui Neves, Vasco Pearson

October 9, 2023

0 Introduction

The explosive growth and increasing computing power of Internet of Things (IoT) devices have resulted in unprecedented volumes of data. Edge devices are computing devices that can process data on a local level and can also transmit data to the local network and to the cloud to be processed.

In order to exploit this amount of data it is necessary to optimally manage the deployment of edge computing workloads, by balancing the pros and cons of local processing (increase of `CLP`) and of sending to other devices (decrease of `CLP`). This work aims to build an intelligent system that manages the computing load (`CLP`) assigned for edge computing tasks on an edge device without compromising aspects such as network security risks, management complexities, and limitations of latency and bandwidth. We consider each device only processes data from lower levels, and when it is operating as a pure edge device (`CLP` = 0), only networking duties (including routing data to cloud) are performed.

It is possible to represent an edge device at a given time by a set of features including memory usage [%] (`MemoryUsage`), processor load [%] (`ProcessorLoad`), input network throughput (`InpNetThroughput`), output network throughput (`OutNetThroughput`), available output bandwidth (`OutBandwidth`), latency [ms] (`Latency`), and by the respective variation rates.

Initially, there is no data available to train a system, so the goal is to build a Fuzzy Inference System (FIS) to predict the output (`CLPVariation`). Then, using the developed FIS we generate a dataset with considerable size in order to enable the implementation of a supervised approach (MLP Regression Neural Network) to address this problem.

The developed code is in three `.py` files and it is delivered together with four `.csv` files.

- `GenerateDatasetFromFIS.py`: Contains the FIS implementation including a plot of the Fuzzy Sets (Membership Functions) used for each variable. Then, it uses this system to generate a dataset.
- `NN_model.py`: Trains an NN model with the previously generated dataset and saves it. It also tests this model on a holdout set from a train/test split of the generated data.
- `TestMe.py`: Tests both FIS and NN models on a dataset from `Proj1_TestS.csv`. The NN model is loaded from the previously saved model trained on the generated dataset.
- `Proj1_TestS.csv`: **Initial provided dataset**. File that contains 12 columns of system input normalized data (between 0 and 1) with the previously described features. It also contains 1 column with the true value for `CLPVariation`.
- `TestResult.csv`: Initially provided dataset with the addition of the output by FIS (`CLPVariation_FIS`) and by NN (`CLPVariation_NN`), for each instance.
- `Proj1_TestS_GeneratedData.csv`: Generated dataset by using the developed FIS described in section 1 to predict `CLPVariation` for each instance. Same structure as in the initial provided dataset. The process of generating features is described in section 2.
- `MLP_Results_in_Fuzzy_Generated_Dataset.csv`: Generated dataset with the addition of the output of NN (`CLPVariation_pred`) for each instance.

1 Fuzzy System

To develop an intelligent system that efficiently manages the computational load (`CLP`) designated for edge computing tasks on an edge device despite certain constraints, we initially employed a Fuzzy Inference System (FIS). This approach is particularly suited for systems with inherent uncertainties, as is the case with our system.

Fuzzy sets are used to represent some precise gradual entity consisting of a collection of items (sets), with gradualness indicated through membership. Fuzzy inference is a process of mapping from a given input to an output, composed of fuzzy rules (Rule Base) and fuzzy reasoning processes using fuzzy sets.

Fuzzy systems complete rule base with n inputs of m term sets has m^n rules. So it becomes exponentially harder to implement them with the increase of the number of inputs because the number of rules increases exponentially (combinatorial rule explosion).

To mitigate the challenges of combinatorial rule explosion and enhance interpretability, we have reduced the number of inputs in each rule base. This was achieved through a hierarchical approach where we combined features in small groups.

Fig. 1 represents the FIS architecture by delineating the structure of these hierarchical relationships among variables within our fuzzy system.

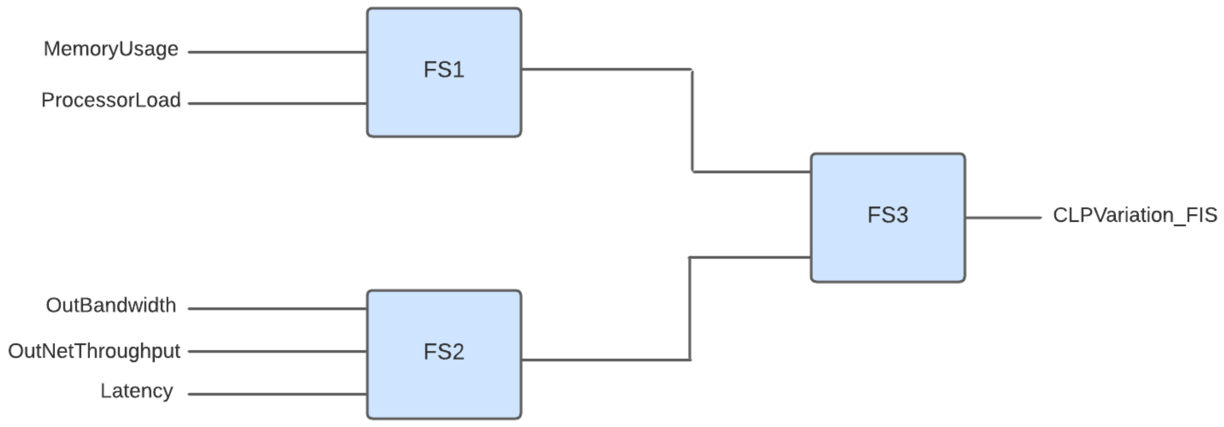


Figure 1: Hierarchical Fuzzy Inference System Architecture.

Our system integrates various fuzzy inference systems, and we have specifically selected the Takagi-Sugeno (TS) Fuzzy System over the Mamdani approach for each of these.

We favored the TS model over the Mamdani due to its analytical nature and concise output representation. Its capacity to generate linear output functions through weighted averages offers a precise and efficient solution, making it a better fit for managing our variable relationships. TS models can approximate nonlinear functions more effectively and better capture complex relationships between inputs and outputs. This can come in handy due to the fact that our system is composed of several features contributing in different ways to the final output.

Moreover, using a TS Fuzzy System can be very beneficial considering the fact the outputs are more scattered. By providing a consistent and straightforward method for defuzzification, the output is always a crisp value, which can be beneficial in control systems where precise output values are required, as in regression problems like the one we are approaching.

In addition, we opted not to utilize all the available features. Specifically, the `InpNetThroughput` was found to be of limited relevance to the specific problem at hand: predicting the `CLPVariation`. This can be attributed to the system's decision-making approach. Initially, the system evaluates whether it possesses adequate resources to locally process more data, both in terms of `ProcessorLoad` and `MemoryUsage`. If resources are sufficient, it then assesses the `OutNetThroughput`, `OutBandwidth`, and `Latency` to determine if local processing on the edge system is advantageous, or if data transfer to the cloud is more beneficial. Given this methodology, the `InpNetThroughput` does not play a fundamental role in the decision process. We also noted a high correlation between `InpNetThroughput` and other used features, which indicates its redundancy. We also do not consider the features relative to variation rates for the same reason. By not using features that are not as important for the problem we reduce its complexity.

We will provide a more detailed description of each of the fuzzy inference systems in the following subsections.

1.1 FS1 Description

The `FS1` inference system governs the relationship between memory usage and processor load, in order to predict their effect in the `CLPVariation`.

The system aims to sustain both the processor load and available memory above average levels, ensuring optimal utilization without squandering processing potential. However, it is crucial to note that when either memory usage or processor load exceeds 85%, the system might struggle to execute its fundamental tasks efficiently.

Both features, `MemoryUsage` and `ProcessorLoad` operate within a range of [0, 1].

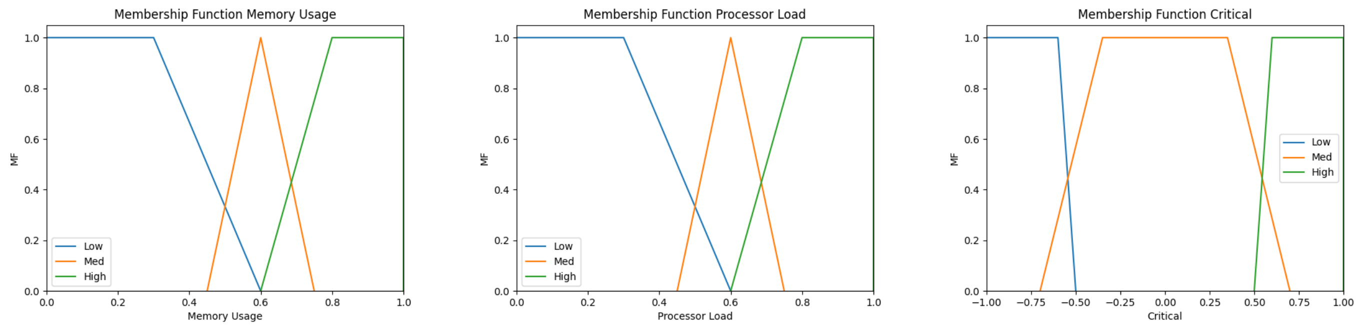
We've categorized `MemoryUsage` into three linguistic terms: `Low`, `Med`, and `High`. Both the `Low` and

`High` terms are represented by a trapezoidal membership function (MF). Specifically, for the `Low` MF, `MemoryUsage` in [0, 0.3] is uniformly viewed as `Low`, receiving a consistent membership value of 1, and between the domain [0.3, 0.6], the `Low` MF decreases. Conversely, for the `High` MF, any `MemoryUsage` in [0.8, 1] is consistently considered `High`, given a uniform membership value of 1, and in the segment [0.6, 0.8] the `High` MF increases (reflecting a gradual transition towards values of `MemoryUsage` seen as higher). The `Med` MF is a triangular MF spanning the domain [0.45, 0.75], increasing in [0.45, 0.6] and decreasing in [0.6, 0.75]. This signifies that values within this range are gradually viewed as moderate `MemoryUsage`, with peak membership (1) at the midpoint of this domain. The `MemoryUsage` MF is represented in Fig. 2(a).

We've similarly categorized `ProcessorLoad` using the three linguistic terms: `Low`, `Med`, and `High`, akin to `MemoryUsage`. The fuzzy sets for `ProcessorLoad` are equal to the ones for `MemoryUsage` and can be found in Fig. 2(b).

The outcome of `FS1` fuzzy inference rules is termed as `Critical`. In future applications, it will serve to predict the impacts of `MemoryUsage` and `ProcessorLoad` on `CLPVariation`. `Critical` spans from -1 to 1 and is characterized by three linguistic terms: `Low`, `Med`, and `High`. The fuzzy sets for `Critical` can be found in Fig. 2(c).

With respect to `Critical`, the `High` MF is a trapezoidal MF, where any `Critical` in [0.6, 1] is consistently considered `High`, given a uniform membership value of 1, and in the segment [0.5, 0.6] the `High` MF increases. Elevated `Critical` values imply diminished `CLPVariation`, as the system potentially struggles with primary task execution. Conversely, `Low` value for `Critical` imply a greater `CLPVariation`, as they indicate under-utilization of processing capacity due to lower values of `MemoryUsage` and `ProcessorLoad`. The `Low` MF is a trapezoidal MF, where any `Critical` in $[-1, -0.6]$ is consistently considered `Low`, given a uniform membership value of 1, and in the segment $[-0.6, -0.5]$ the `Low` MF decreases. For states of `Critical` that aren't distinctly `High` or `Low`, `Critical`'s value doesn't significantly affect the final `CLPVariation` prediction. The `Med` term, characterizing this region, has a trapezoidal MF, where any `Critical` in $[-0.35, 0.35]$ is consistently considered `Med` given a uniform membership value of 1. In the segment $[-0.7, -0.35]$ the `Low` MF increases, and in the segment $[0.35, 0.7]$ the `Low` MF decreases.



(a): `MemoryUsage` MF. (b): `ProcessorLoad` MF. (c): `Critical` MF.

Figure 2: Membership functions introduced in `FS1`.

The rule base of `FS1` is outlined by the rules presented in Table 1.

Table 1: Rule Base for `Critical` Inference.

Critical		ProcessorLoad		
		Low	Med	High
MemoryUsage	Low	Low_Critical	Low_Critical	High_Critical
	Med	Low_Critical	Regular	High_Critical
	High	High_Critical	High_Critical	High_Critical

With the TS Fuzzy System at play, we represent in the `FS1` rule base a designation for each of the three possible equations (equations that give individual rule outputs): High_Critical, Low_Critical and Regular.

High_Critical equation is given by (1) and it is relevant (has a high degree of fulfillment) when either MemoryUsage or ProcessorLoad is High and the system becomes critically stressed, prompting a need to minimize the CLPVariation.

$$\texttt{Critical} = \max(\texttt{MemoryUsage}, \texttt{ProcessorLoad}) \times 2 - 1 \tag{1}$$

Low_Critical equation is given by (2) and it has high degree of fulfillment when either MemoryUsage or ProcessorLoad is Low and none of them is High, indicating the system isn't exploiting its full potential. This signals a need to increase the CLP.

$$\texttt{Critical} = \left(\frac{\texttt{MemoryUsage} + \texttt{ProcessorLoad}}{8}\right) \times 2 - 1 \tag{2}$$

Regular equation is given by (3) and has relevance when both metrics align with the Med term. Here, the system operates efficiently, and CLPVariation hinges more on aspects like OutNetThroughput, OutBandwidth, and Latency.

$$\texttt{Critical} = \left(\frac{\texttt{MemoryUsage} + \texttt{ProcessorLoad}}{2}\right) \times 2 - 1 \tag{3}$$

In the previous equations, values are normalized from the range $[0, 1]$ to $[-1, 1]$ ($x_{\text{normalized}} = 2x - 1$). These output functions (rule outputs) were optimized and tuned based on the initial dataset in order to attain good performance and prevent overfitting.

1.2 FS2 Description

The FS2 inference mechanism defines the interplay among OutNetThroughput, OutBandwidth, and Latency. Its purpose is to predict their collective influence on CLPVariation.

The OutNetThroughput denotes the speed at which messages are transmitted over communication channels, like Ethernet or packet radio, originating from edge devices.

We have segmented OutNetThroughput into three linguistic descriptors: Low, Med, and High. Each is represented by a triangular MF. Spanning the domain [0, 0.5], the Low MF gradually decreases. The High MF within [0.5, 1] gradually increases. The Med MF, defined within the domain [0.3, 0.7], peaks at the midpoint (Med MF is equal to 1 at 0.5), increases in [0.3, 0.5], and decreases in [0.5, 0.7]. The OutNetThroughput MF is represented in Fig. 3(a).

The available output bandwidth is the amount of data that the network can carry out over time. Although it is possible to increase network bandwidth to accommodate more devices and data, the cost can be significant and there are still other problems associated.

We've similarly categorized OutBandwidth using the three linguistic terms: Low, Med, and High, akin to OutNetThroughput. The fuzzy sets for OutNetThroughput are equal to the ones for OutNetThroughput and can be found in Fig. 3(b). When OutBandwidth is Low, the expense of relaying information to the cloud for processing is higher. On the other hand, when OutBandwidth is High it reflects enhanced capabilities to send data to the cloud.

Latency is the time needed to send data between two points on a network. Although communication ideally takes place at the speed of light, large physical distances coupled with network congestion or outages can delay data movement across the network. This slows down any analytics and decision-making processes and reduces the ability of a system to respond in real-time.

Latency is classified into three distinct linguistic categories: Low, Med, and High. The Low and High categories are mapped using trapezoidal MF. Specifically, the Low MF in [0, 0.3] is uniformly perceived as Low yielding a stable membership value of 1, and within [0.3, 0.5] the Low MF decreases. In contrast, the High MF in the range [0.7, 1] is uniformly perceived as High (membership value of 1), and in [0.5, 0.7] the High MF increases. The Med MF is a triangular MF across the span [0.3, 0.7], increasing in [0.3, 0.5], decreasing in [0.5, 0.7], and with maximum membership (1) observed at the midpoint (0.5). The Latency MF is represented in Fig. 3(c).

When Latency is Low, there is a minimal cost for transmitting data to the cloud for processing. On the other hand, when Latency is High it reflects that local data processing is more advantageous than cloud transmission.

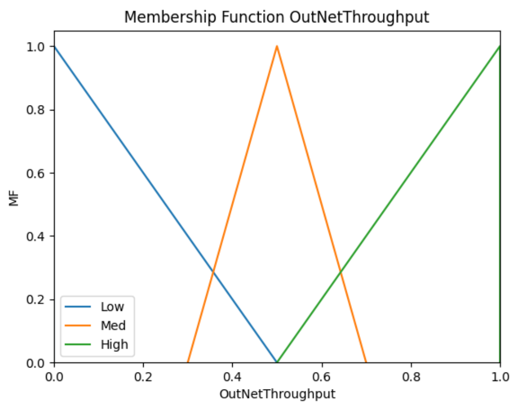
The outcome of `FS2` fuzzy inference rules is denoted as `FinalOut`. This variable represents the contribution of `OutNetThroughput`, `OutBandwidth`, and `Latency` on `CLPVariation`. The range for `FinalOut` extends from -1 to 1, encapsulated by three linguistic descriptors: `Low`, `Med`, and `High`.

A `High` value for `FinalOut` implies that the system either contends with elevated `Latency` or exhibits diminished `OutNetThroughput`/`OutBandwidth`. This translates to a higher cost of transferring data for cloud processing, thereby inclining the system to prefer local data processing. Consequently, this implies a higher `CLPVariation`. So high `FinalOut` values suggest higher `CLPVariation`. The `High` MF is represented by a triangular MF that increases in the range [0, 1].

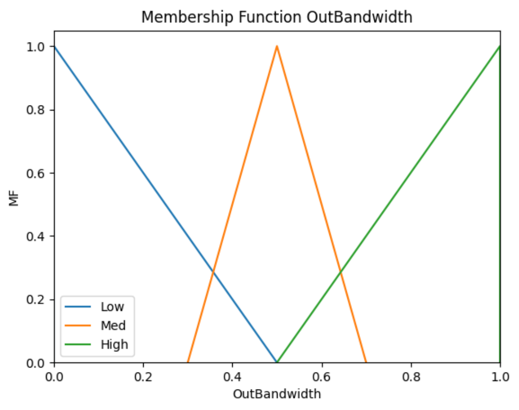
In contrast, a `Low` value for `FinalOut` values indicates low `CLPVariation`. Such values represent a system optimized for swift data transfer to the cloud, attributed to minimal latency or heightened OutNetThroughput/OutBandwidth (in cases where there is neither maximum latency nor minimal OutNetThroughput/OutBandwidth). This minimizes the dependency on local data processing. The `Low` MF is represented by a triangular MF that decreases in the range $[-1, 0]$.

For the remaining scenarios, `FinalOut` is `Med`. The `Med` MF is a triangular MF across the span $[-0.4, 0.4]$, increasing in $[-0.4, 0]$, decreasing in [0, 0.4], and with maximum membership (1) observed at the midpoint (0).

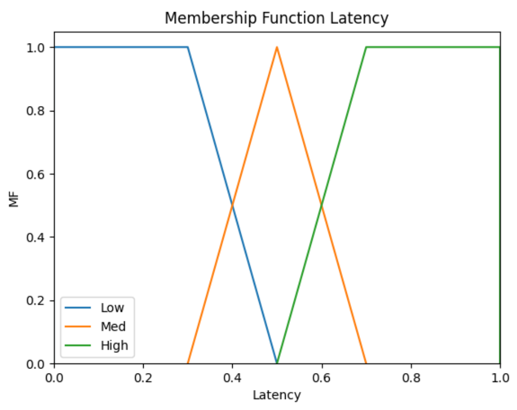
The `FinalOut` MF is represented in Fig. 3(d).



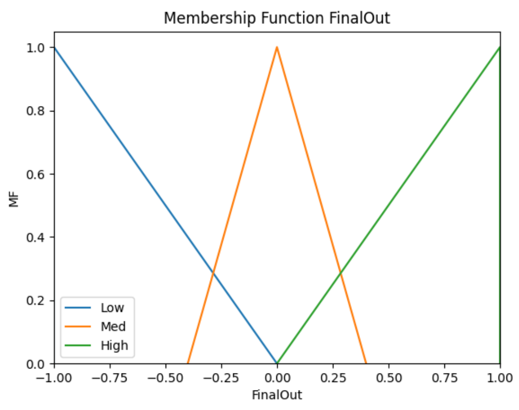
(a): OutNetThroughput MF.



(b): OutBandwidth MF.



(c): Latency MF.



(d): FinalOut MF.

Figure 3: Membership functions introduced in `FS2`.

The rule base of `FS2` is outlined by the rules presented in Table 2.

Utilizing the TS Fuzzy System, we've delineated four output functions denominated `HIGH_LAT`, `LOW_ONT`, `LOW_OBW` and `OTHER`.

Table 2: Rule Base for `FinalOut` Inference.

`OutNetThroughput`	`OutBandwidth`	`Latency`	`FinalOut`
High	Low	Low	LOW_OBW
High	Low	Med	LOW_OBW
High	Low	High	HIGH_LAT
High	Med	Low	OTHER
High	Med	Med	OTHER
High	Med	High	HIGH_LAT
High	High	Low	OTHER
High	High	Med	OTHER
High	High	High	HIGH_LAT
Med	Low	Low	LOW_OBW
Med	Low	Med	LOW_OBW
Med	Low	High	HIGH_LAT
Med	Med	Low	OTHER
Med	Med	Med	OTHER
Med	Med	High	HIGH_LAT
Med	High	Low	OTHER
Med	High	Med	OTHER
Med	High	High	HIGH_LAT
Low	Low	Low	LOW_OBW
Low	Low	Med	LOW_OBW
Low	Low	High	HIGH_LAT
Low	Med	Low	LOW_ONT
Low	Med	Med	LOW_ONT
Low	Med	High	HIGH_LAT
Low	High	Low	LOW_ONT
Low	High	Med	LOW_ONT
Low	High	High	HIGH_LAT

These functions facilitate the articulation of the relationship between these features and the relative significance of each feature under varied conditions. In addition, both `OutNetThroughput` and `OutBandwidth` have a decreasing effect in the final value of `FinalOut`, meaning the lower the value of these features, the higher the `FinalOut` and consequently the higher will be the `CLPVariation`. So in all the equations, we use as terms $(1 - \texttt{OutBandwidth})$ and $(1 - \texttt{OutNetThroughput})$.

`HIGH_LAT` equation is determined by (4) and it has high degree of fulfillment when `Latency` is `High`. Under these circumstances, `Latency` emerges as the key feature, overshadowing the contributions of the other two features and increasing the `CLPVariation`. In this case, the impact of `OutBandwidth` and of `OutNetThroughput` is negligible.

$$\begin{aligned}\texttt{FinalOut} = \max\Bigg(\min\Bigg(&(0.0 \times (1 - \texttt{OutNetThroughput}) + 0.05 \times (1 - \texttt{OutBandwidth})\\ &+ 1 \times \texttt{Latency} + 0.1) \times 2 - 1, 1\Bigg), -1\Bigg)\end{aligned} \tag{4}$$

`LOW_OBW` equation is formulated as (5) and it is relevant in instances where `Latency` is not categorized as `High` and `OutBandwidth` is deemed `Low`. In this case, `OutBandwidth` emerges as the main factor in the equation, with the highest coefficient. A decrease in `OutBandwidth` directly amplifies the value of `FinalOut`, thereby increasing the `CLPVariation`. While `Latency` retains some significance in this scenario, its influence is less than `OutBandwidth` influence. Furthermore, `OutNetThroughput` holds a comparable level of influence to that of `Latency`.

$$\begin{aligned}\texttt{FinalOut} = \max\Bigg(\min\Bigg(&(0.4 \times (1 - \texttt{OutNetThroughput})\\ &+ 0.9 \times (1 - \texttt{OutBandwidth})\\ &+ 0.5 \times \texttt{Latency}) \times 2 - 1, 1\Bigg), -1\Bigg)\end{aligned} \tag{5}$$

`LOW_ONT` equation is given by (6) and it has high degree of fulfillment when `Latency` are not `High`, `OutBandwidth` isn't identified as `Low`, and `OutNetThroughput` is `Low`. In this case, `Latency` is the predominant influencer in the equation, followed by `OutNetThroughput` also with high significance. On the other hand, the influence of `OutBandwidth` is low.

$$\begin{aligned}\texttt{FinalOut} = \max\Bigg(\min\Bigg(&(0.6 \times (1 - \texttt{OutNetThroughput})\\ &+ 0.2 \times (1 - \texttt{OutBandwidth})\\ &+ 0.9 \times \texttt{Latency}) \times 2 - 1, 1\Bigg), -1\Bigg)\end{aligned} \tag{6}$$

`OTHER` equation is formulated as (7) and it is relevant in the absence of the previously described conditions (in cases where rules governed by (4), (5), and (6) are not relevant). Here, even though `OutNetThroughput` and `Latency` stand out as the primary determinants, `OutBandwidth` retains a marked influence on the computation of the `FinalOut` value.

$$\begin{aligned}\text{FinalOut} = \max\Bigg(\min\Bigg(&(0.6 \times (1 - \text{OutNetThroughput})\\ &+ 0.3 \times (1 - \text{OutBandwidth})\\ &+ 0.63 \times \text{Latency}) \times 2 - 1, 1\Bigg), -1\Bigg)\end{aligned} \tag{7}$$

In the preceding equations, we not only normalize the values from the range of $[0, 1]$ to $[-1, 1]$, but we also enforce the restriction that the output of each equation remains within the interval $[-1, 1]$ by employing the technique $x_{out} = \max(\min(x, 1), -1)$. These output functions were optimized (tuning of the equation's coefficients) to attain good performance and prevent overfitting, based on the initial dataset.

1.3 FS3 Description

The culminating Fuzzy Inference System `FS3` predicts the `CLPVariation`, building upon the earlier deduced values of `FinalOut` and `Critical`.

The `CLPVariation` is categorized into three linguistic descriptors: `Positive`, `Null`, and `Negative`. Each descriptor is characterized by a triangular membership function (MF). The `Negative` MF, defined over the range $[-1, 0]$, shows a diminishing trend. Conversely, the `Positive` MF, spanning the interval [0, 1], exhibits a rising trend. The `Null` MF, set within the domain $[-0.35, 0.35]$, peaks at its midpoint (`Null` MF is equal to 1 at 0.5), ascending in the interval $[-0.35, 0]$ and descending in $[0, 0.35]$. The `CLPVariation` MF is represented in Fig. 4.

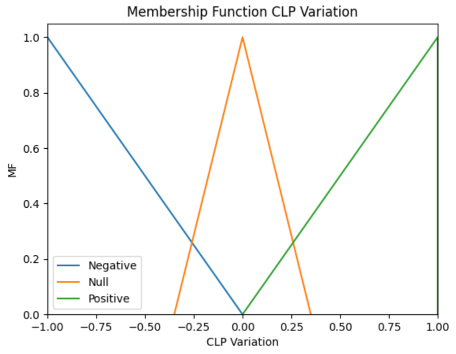


Figure 4: Membership functions of CLPVariation.

This system is designed to receive two primary variables: `Critical` and `FinalOut`. Specifically, for the variable `Critical`, if its value is `High` (indicating that the system may encounter challenges in performing its core functions effectively due to factors like excessive processor load or memory constraints), or `Low` (indicative of unused processing capability and available memory), it signifies that the system's state is critical (`VERY_CRITICAL`). In such scenarios, the variation in `CLPVariation` is significantly influenced by the `Critical` value.

On the other hand, when `Critical` has intermediate values (`NOT_CRITICAL`), its significance diminishes, rendering the `CLPVariation` more reliant on the value of `FinalOut`.

The rule base of `FS3` is outlined by the rules presented in Table 3.

Table 3: Rule Base for `CLPVariation` Inference.

CLPVariation		FinalOut		
		Low	Med	High
Critical	Low	VERY_CRITICAL	VERY_CRITICAL	VERY_CRITICAL
	Med	NOT_CRITICAL	NOT_CRITICAL	NOT_CRITICAL
	High	VERY_CRITICAL	VERY_CRITICAL	VERY_CRITICAL

Through the implementation of the TS Fuzzy System, we have defined two output functions consistent with the above observations, denominated `VERY_CRITICAL` and `NOT_CRITICAL`. It's worth noting that since `Critical` has a negative impact on the final value of `CLPVariation`, all our equations attach negative coefficients to the `Critical` term. This implies that a lower value of `Critical` corresponds to a higher `CLPVariation`. On the other hand, an increase in the `FinalOut` value directly corresponds to a rise in the `CLPVariation`.

`VERY_CRITICAL` equation is given by (8) and it is relevant in instances where `Critical` is either `Low` or `High`. In this case, `Critical` heavily influences the equation, diminishing the role of `FinalOut`.

$$\texttt{CLPVariation} = \max\left(\min\left(-0.91 \times \texttt{Critical} + 0.09 \times \texttt{FinalOut}, 1\right), -1\right) \tag{8}$$

`NOT_CRITICAL` equation is formulated as (9) and has greater significance when the system operates under non-critical conditions (`Critical` is neither `Low` nor `High`). In this case, `FinalOut` emerges as the predominant influencer in the equation to obtain `CLPVariation`.

$$\texttt{CLPVariation} = \max\left(\min\left(-0.15 \times \texttt{Critical} + 0.85 \times \texttt{FinalOut}, 1\right), -1\right) \tag{9}$$

In the preceding equations, we not only normalize the values from the range of $[0, 1]$ to $[-1, 1]$, but we also enforce the restriction that the output of each equation remains within $[-1, 1]$. These output functions were optimized (tuning of the equation's coefficients) to attain good performance and prevent overfitting, based on the initial dataset.

1.4 Results

Finally, we use the FIS model to compute the `CLPVariation` for each instance in the initially provided dataset (results are in column `CLPVariation_FIS` of `TestResult.csv`), and we obtain as performance metrics:

- Mean Squared Error = 0.0053
- Root Mean Squared Error = 0.0729
- Mean Absolute Error = 0.0660

These results are discussed in section 4.

2 Generated Dataset

Using the developed Fuzzy System we generate a dataset in order to implement a supervised approach to address this regression problem. In section 3, we describe this approach by using a model based on a Multilayer Neural Network (NN) able to replicate the results of the FIS.

The generated dataset has 1024 instances with a high diversity of values for its features. The 6 columns with the variation rates are obtained by the differences between the respective features in the current and previous instances.

The `InpNetThroughput` column is obtained by random values between 0 and 1. For the remaining 5 columns we establish a set $\{0.2, 0.4, 0.6, 0.8\}$ and we consider all possible combinations of these values by features to create 4^5 instances (1024). Subsequently, some random noise (between -0.2 and 0.2) was added to each of these five features, in each instance.

Finally, for each instance, we use the developed FIS in section 1 to obtain the respective `CLPVariation` prediction.

3 Neural Networks

Using the generated dataset, we implement a supervised approach (a model based on a Multilayer Neural Network) to address the problem of managing CLP.

Overfitting occurs when a model learns the training data too well, capturing noise and minor fluctuations instead of the underlying patterns, which leads to excellent performance on the training set but poor generalization to unseen data (new data). In order to avoid overfitting (and underfitting) the model to the generated dataset, we use a train/test split and then a train/validation split within a cross-validation strategy described below.

This train/test split ensures that the model can generalize to unseen data. The training set is used to train the model and the testing set (holdout set) evaluates its performance: the error on the test set provides a better (unbiased) estimate of the true generalization error. The split ratio considered was 70% training set and 30% testing set.

3.1 Training

Neural networks can be used to extract patterns and detect trends in the dataset in order to map the system. For conceiving a model we use `MLPRegressor` (Multi-layer Perceptron Regressor) from scikit-learn [3]. This is a type of artificial neural network used for regression tasks (prediction of a continuous numeric value), capable of modeling complex and non-linear relationships in data. However, it may require a substantial amount of training data which we have due to the considerable generated dataset size. It also requires careful hyperparameter tuning to perform effectively.

In the input layer, the model has one neuron for each feature in the input data. We only consider as relevant features `MemoryUsage`, `ProcessorLoad`, `OutNetThroughput`, `OutBandwidth` and `Latency`, so input layer has 5 neurons.

The model typically has one or more hidden layers between the input and output layers, consisting of multiple neurons. Each neuron in a hidden layer is connected to every neuron in the previous layer through weighted connections (weights are parameters learned during training) and may apply a non-linear activation function to the weighted sum of its inputs in order to introduce non-linearity into the model.

The output layer of the regression model consists of a single neuron, usually with a linear activation function, aiming to obtain a prediction for `CLPVariation`.

To train the model we use a cross-validation strategy using scikit-learn model `GridSearchCV`, where the parameters of the model are optimized (tuned) by cross-validated grid-search over a parameter grid.

Cross-validation allows the evaluation of the performance and generalizability of a predictive model. The training set is split into $k = 5$ folds T_k. One fold is used for validation and the others for training. The validation fold rotates k times, to ensure a robust assessment. Then, the performance metrics are obtained by the average of the performance during the k rounds. This allows to detect configurations of model parameters that lead to overfitting/underfitting. So it provides robust model performance evaluation that can be used for tuning parameters.

We used `GridSearchCV` [4] for hyperparameter tuning (it could also be used for model selection), with the goal of finding the best set of hyperparameters that optimize model performance. It explores all combinations of hyperparameters specified in the `parameter_space` and selects the combination that produces the best performance score. We chose as scoring parameter the mean squared error (MSE). The convention is that higher score values are better than lower score values, so in this case, we use `scoring='neg_mean_squared_error'`. [5]

Now, we describe the parameters chosen for tuning, i.e, the `parameter_space`.

- Activation function for the hidden layer: {Tanh, ReLU, Logistic}. Chosen since they are non-linear activation functions that introduce non-linearity into the network.

- Solver for weight optimization: {SGD, Adam}.
- Alpha (α) (strength of the L2 regularization term): {0, 0.001, 0.01, 0.05}. L2 regularization imposes a penalty on the size of the weights of the nodes, which prevents overfitting.
- Learning rate (η): {constant, adaptive}. Only used with SGD in order to govern the pace at which the weights are updated. Constant means η given by η_0 (constant). Adaptive causes variations in η according to the behavior of training loss.
  - Learning rate init (η_0): {0.01, 0.05, 0.1}.
- Hidden Layer Sizes: {(12,12,12),(10,10,10), (8,6,3), (6,4,2), (4,5,4),(4,3,3),(8),(9,6),(8,7,6)}.
  - We consider as candidates diverse situations for the number of hidden layers:
    - 1 hidden layer can approximate any function that contains a continuous mapping from one finite space to another. However, it may struggle with complex data patterns.
    - 2 hidden layers can represent an arbitrary decision boundary and any smooth mapping.
    - 3 hidden layers allows the learning of more complex representations.
  - We also test cases with different number of neurons per layer because too few neurons may cause underfitting and too much neurons may cause overfitting and high temporal complexity.

We also created an Early stopping monitor that will stop training when the validation loss is not improving in each round. This criterion was used to avoid overfitting, given that the training of the network stopped when the score of the validation data did not improve, over the best configuration, in 10 consecutive epochs (`early_stopping=True, n_iter_no_change=10`).

Using the described cross-validated grid-search strategy, the obtained tuned best parameters (used in the trained model) are:

- Activation function: Logistic
- α: 0
- Hidden layer sizes: (12, 12, 12)
- Learning rate: Constant (0.05)
- Solver: Adam

Finally, using the python library `pickle` [6] we save the trained model as `NN_model.sav` for future use.

3.2 Testing

Firstly, we test the performance of the model in the testing set (holdout set) from the FIS generated data (results are in column `CLPVariation_pred` of `MLP_Results_in_Fuzzy_Generated_Dataset.csv`), and we obtain as performance metrics:

- Mean Squared Error = 0.0321
- Root Mean Squared Error = 0.1792
- Mean Absolute Error = 0.1179

Finally, we use the NN model to compute the `CLPVariation` for each instance in the initial provided dataset (results are in column `CLPVariation_NN` of `TestResult.csv`), and we obtain as performance metrics:

- Mean Squared Error = 0.0284
- Root Mean Squared Error = 0.1686
- Mean Absolute Error = 0.1440

4 Discussion

In this section, we compare the performance of both FIS and NN models on the initial provided dataset (which is labeled). In other words, we are comparing the performance of `CLPVariation_FIS` and `CLPVariation_NN` with the true label `CLPVariation`, from `TestResult.csv`. As we can verify in sections 1 and 3, all the considered metrics (MSE, RMSE and MAE) obtain better results for the FIS model.

This was expected, since for the FIS model this dataset was used as a reference in the model development (especially in tuning), so the model is overfitted to this dataset (excellent performance in this). On the other hand, for the NN model this dataset is a holdout set not used in the model training process, providing a better (unbiased) estimate of the true generalization error.

NN models require a large amount of labeled data for effective training, so it was required to generate an annotated dataset since real annotations or labels for a dataset were not given (because they are scarce or expensive to obtain). Instead of relying on an oracle (human annotator), we use pseudo-annotation by using a model (FIS) to generate labels for the data. Pseudo-annotations come with several disadvantages compared to using real annotations as lack of ground truth by experts, and that they are generated by the FIS model, which means they inherit any errors or biases present in the FIS model, in addition to the inherent uncertainty associated with the NN model. This is another reason why the performance metrics are worse for the NN model.

5 Conclusion

In this work, we build an intelligent system that predicts the variation in the computing load (`CLPVariation`) assigned for edge computing tasks on an edge device without compromising aspects such as network security risks, management complexities, and limitations of latency and bandwidth.

Initially, there was no data available to train a system, so we built a Fuzzy Inference System (FIS) to predict the output (`CLPVariation`). Using this system, it was possible to generate a larger dataset of pseudo-annotations that allowed training an MLP Regressor Neural Network model of the system.

As we discussed in section 4, in the initial provided dataset (TestS) the considered performance metrics (MSE, RMSE and MAE) obtain better results for the FIS model compared with the NN model. This is due to the overfitting of FIS model to the TestS dataset and to the use of pseudo-annotations in the training of the NN model, with inherent errors and biases from the pseudo-annotation model (in this case the FIS model).

References

[1] Applied Computational Intelligence (CInte) Slides - Slides of Theoretical Classes 2023/2024, 1st Semester (MEEC), by Nuno Horta and Joao Paulo Carvalho;

[2] CInte 2022/2023, Project 1 - Fuzzy Systems and Neural Networks – Final Version;

[3] Sklearn, Neural Network, MLPRegressor. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html`;

[4] Sklearn, Model Selection, GridSearchCV. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`;

[5] Metrics and scoring: quantifying the quality of predictions. Available: `https://scikit-learn.org/stable/modules/model_evaluation.html`.

[6] Save and Load Machine Learning Models in Python with scikit-learn: Save your model with pickle. Available: `https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/`.