A Machine Learning-Based Approach to Support the Bottom-Up Design of Simple Emergent Behaviors in Systems-of-Systems

Valdemar Vicente Graciano Neto[1], Kanan Silva[2], Arlindo Rodrigues Galvão Filho[1], Aparna Kumari[3], Flávio Eduardo Aoki Horita[2], and Mohamad Kassab[4]

[1]Universidade Federal de Goias Instituto de Informatica
[2]Universidade Federal do ABC
[3]Nirma University
[4]The Pennsylvania State University

September 28, 2023

Abstract

Systems-of-Systems (SoS) are composed of multiple independent systems called constituents that, together, achieve a set of goals by means of emergent behaviors. Those behaviors can be deliberately planned as a combination of the individual functionalities (herein named as *features*) provided by the constituents. Currently, SoS engineers heavily rely on their own creativity and prior experience to combine the features and design the behaviors. However, the limitation of human perception in complex scenarios can lead to engineering sub-optimized SoS arrangements, potentially causing waste of the resources, sub-optimal services and reduction in quality. To handle the aforementioned issues, this article presents a machine learning-based mechanism for inferring/suggesting emergent behaviors that could be designed over a given set of constituents. An initial dataset was elaborated from a systematic mapping to feed the mechanism and a web-application was developed as a means for experts to evaluate this mechanism. Results revealed that the algorithm developed is capable of predicting feasible emergent behaviors for different sets of constituents and the system can be useful in the sense of aiding SoS engineers and experts in the bottom-up design of these behaviors.

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

A Machine Learning-Based Approach to Support the Bottom-Up Design of Simple Emergent Behaviors in Systems-of-Systems

Kanan Castro Silva*[1] | **Arlindo Rodrigues Galvão Filho**[2] | **Aparna Kumari**[3] | **Flávio Eduardo Aoki Horita**[1] | **Mohamad Kassab**[4] | **Valdemar Vicente Graciano Neto**[2]

[1]Federal University of ABC, São Paulo, Brazil
[2]Federal University of Goiás, Goiás, Brazil
[3]Nirma University, Gota, India
[4]Penn State University, Malvern, United States

Correspondence
*Corresponding author name, Corresponding address.
Email: kcs.kanan@gmail.com

Abstract

Systems-of-Systems (SoS) are composed of multiple independent systems called constituents that, together, achieve a set of goals by means of emergent behaviors. Those behaviors can be deliberately planned as a combination of the individual functionalities (herein named as *features*) provided by the constituents. Currently, SoS engineers heavily rely on their own creativity and prior experience to combine the features and design the behaviors. However, the limitation of human perception in complex scenarios can lead to engineering sub-optimized SoS arrangements, potentially causing waste of the resources, sub-optimal services and reduction in quality. To handle the aforementioned issues, this article presents a machine learning-based mechanism for inferring/suggesting emergent behaviors that could be designed over a given set of constituents. An initial dataset was elaborated from a systematic mapping to feed the mechanism and a web-application was developed as a means for experts to evaluate this mechanism. Results revealed that the algorithm developed is capable of predicting feasible emergent behaviors for different sets of constituents and the system can be useful in the sense of aiding SoS engineers and experts in the bottom-up design of these behaviors.

KEYWORDS
Systems-of-systems, emergent behavior, bottom-up design, machine learning, dataset, support vector regression, web-application

1 | INTRODUCTION

Software has been increasingly embedded into systems (e.g., autonomous cars, traffic control systems, power distribution systems) to increase the precision of their functionalities, deliver automation, and make them smarter[1,2]. At the same time, those systems have become progressively specialized in their duties, and the design of more complex behaviors have demanded them to be combined, forming what is known as Systems-of-Systems (SoS[†]). SoS are alliances of several systems that combine their individual functionalities to establish innovative behaviors, so-called emergent behaviors, i.e., complex behaviors that could not be offered by single constituents, but that can be achieved as the result of the combination, cooperation and/or interoperability among the constituents[3]. Smart cities are one of the most popular examples of SoS. Several innovative behaviors can be obtained from the set of systems involved in a smart city. For instance, to pursue a fluid traffic, it is necessary to combine the sensing abilities of several autonomous cars so that they avoid collisions and keep a safe distance while preserving an average speed to make the traffic flush[4]. Moreover, the cars should communicate with the traffic system so that they can regulate their own speed, stop and advance as citizen need to cross the street or the traffic light turns red.

Since SoS often support those domains, we consider them to be critical systems, i.e., systems in which failures can cause important threats, losses, damages or injuries to their users, to the surrounding environment or even to the public patrimony. Cars colisions, for instance, could cause injuries or losses to citizens or even to destroy light poles. Hence, SoS need to be reliable and accurate, since users' safety relies on their services; but the complexity involved is larger then the isolated systems

[†] Henceforth, SoS acronym will be interchangeably used to express both singular and plural forms: System-of-Systems and Systems-of-Systems

due to the number of systems and functionalities involved. In that direction, quality (i.e., the correspondence between what the system should offer in terms of functionalities and attributes (response time, ability to recover) and what it really delivers) is an imperative concern for SoS[5,6]. However, engineering SoS (and their emergent behaviors) still heavily relies on (and are limited to) the prior knowledge, experience and even creativity of the involved engineers to combine the capabilities (called herein as *features*) offered by the candidate constituent systems to design the intended emergent behaviors. For example, in a smart grid SoS, renewable resources such as photovoltaic cells or wind turbines might generate energy respectively from irradiation or wind velocity, while battery storages maintain this energy for later use, in case of a power outage or absence of renewable sources energy generation. There might be also transmission lines to flow this power to smart homes that use IoT applications, charging station for electric vehicles and so on. The interplay among these constituents could lead to a self-regulated balance between energy production and consumption, energy use optimization and grid stability. However, the combination of functionalities should be foreseen by the human engineers, which could lead to waste of resources and non-optimized architectures, impacting on the budget and results delivered.

Relying on human abilities to draw the SoS emergent behaviors can potentially lead to non-optimized services, jeopardizing the quality of the behaviors being delivered by the SoS. Moreover, (i) emergent behaviors can exhibit some level of uncertainty, sometimes being difficult to map or model them[7], then being essential to predict these behaviors and (ii) depending on which system is added or removed in a SoS, this could affect in different levels the global behavior being delivered[8]. In this context, machine learning (ML) algorithms might enhance the SoS Engineering process, as they are capable of dealing with a larger amount of data and understand patterns within collections of data in a way humans cannot manually perform due to the complexity. ML algorithms are also capable of processing a series of inputs in a dataset and proposing or inferring one or more outputs in a model. These algorithms can analyze the tacit knowledge from the engineers minds externalized in a dataset and replicate it not only to recognize possible emergent behaviors in a combination of constituents, but also to extend this process to predict feasible emergent behaviors that were not previously idealised.

The main contribution of this paper is presenting results of a research on the adoption of a ML mechanism to suggest possible emergent behaviors derived from a set of constituents. The contributions of this work are manifold and include:

- the preparation of a dataset that works as a repository of emergent behaviors specified in terms of multiple basic functionalities (features) offered by individual systems (candidate constituents);
- a ML algorithm that can learn how the features were combined to form the behaviors;
- a system implementation (called MacGyver Predictor); and
- the evaluation of the system with experts.

In our prior work[9], we only presented the idea and showed preliminary results of the ML algorithm. Herein, we extend our work by presenting the entire framework, including the dataset, the prototype of the conceived system and the evaluation of this prototype with experts. The results presented herein reveal that the algorithm developed is capable of predicting feasible emergent behaviors for different sets of constituents and the system can be useful in the sense of aiding SoS engineers and experts in the bottom-up design of these behaviors.

This article is organized as follows: Section 2 provides the background, Section 3 introduces the research proposal and Section 4 conducts an assessment with experts to evaluate the proposed solution accordingly. Section 5 concludes this paper with the study conclusions and future work.

2 | BACKGROUND

Systems-of-Systems. SoS are distinguished from other complex and large-scale systems by means of the following characteristics[3]: *Managerial independence of the constituents*, once the constituents are owned and managed by different companies and/or entities; *Operational independence of the constituents*, i.e., the constituents contribute to SoS goals, but they preserve their independence about their own operation; *Evolutionary development*: the SoS systemically evolves as a consequence of evolution of its constituents, missions/goals or architecture; *Distribution*, because the constituents are physically decoupled and communicate using some network technology; and *Emergent behavior*, once the purposes and functions performed by the SoS are not individually found in any of its constituents but, rather, they come from the interactions among these systems, which help on fulfilling the main goals of the SoS. These behaviors can be classified into four types, namely *Simple* (predicted in simplified models of the SoS, with only intentional behaviors emerging); *Weak* (reproducible consistently only in simulation models, partially predicted in advance, but undesired behaviors may also occur); *Strong* (consistent with known properties of SoS but not reproducible in any models); and *Spooky* (inconsistent with known properties of SoS, not reproducible or capable of being simulated in any model of the system).

SoS can be categorized into four classes[7]: (i) *Directed SoS*; (ii) *Acknowledged SoS*; (iii) *Collaborative SoS*; and (iv) *Virtual SoS*. This classification depends on properties regarding control and management within the SoS as well as the manner in which the interactions among constituents occur.

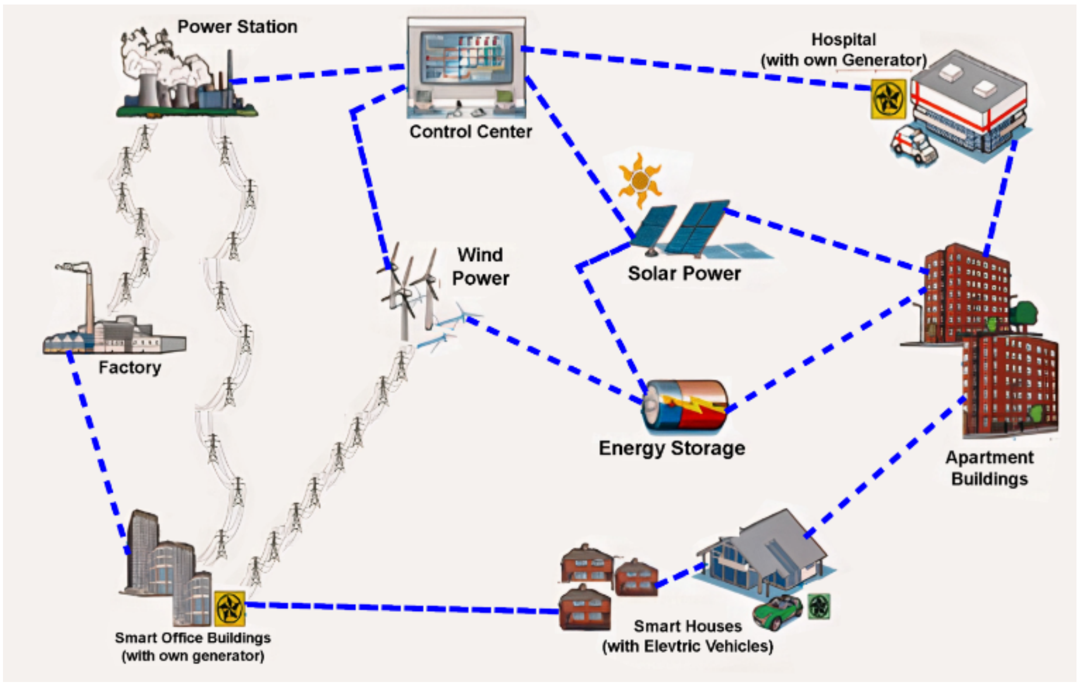

Power Station
Control Center
Hospital (with own Generator)
Wind Power
Solar Power
Factory
Energy Storage
Apartment Buildings
Smart Office Buildings (with own generator)
Smart Houses (with Elevtric Vehicles)

FIGURE 1 Smart Grid SoS (adapted from [10]).

An example of a smart grid SoS is depicted in Figure 1. In that SoS, energy can be provided by wind and solar generation or even from the generators available in smart buildings or hospitals. These buildings can also consume this energy available in the grid. The surplus of energy can be stored for later use when, for example, the generators are not able to provide this power. Each constituent in the SoS manages its own goals and a control center operates to guarantee that the SoS as a whole will achieve the global missions. The dashed lines represent the flow of energy, which might happen in a bidirectional way. The overall architecture is able to adapt itself to route this flow of energy around congestion in the grid and sense constituent changes, taking preventive actions. It has greater redundancy, resiliency and provides distributed generation and massive distributed storage, leading to an equilibrium between energy production and consumption[10].

Regarding the emergent behavior designing process, a SoS engineer often has two options[11]. In a *top-down* approach, given a set of established complex goals, the engineer refines each complex goal, breaking it down into small operational pieces until they reach a granularity that matches the granularity of the features offered by the available constituents. Then, s/he possibly combine the constituents in a data flow (or a business process) that interoperate them and enables the realization of the intended behavior as the result of the combination of their individual capabilities. In a *bottom-up* approach, the engineer observes the constituents available (and their respective features) to be used in the design of emergent behaviors and creatively combine their features by interoperating the constituents, creating data flows until the set of constituents can offer the desired results, i.e., the planned goals performed in emergent behaviors.

SoS, such as smart cities, are usually formed by a high number of constituents, which can range from hundreds to millions[1], leading to a deluge of data generated by so many instruments. In this sense, ML and DL techniques have the potential to aid in the design of emergent behaviors in SoS.

Machine Learning and Deep Learning. ML is defined as the ability of the computer to learn without explicitly being programmed[12]. Their conventional techniques are limited in capability of processing the data in their original form. Deep learning, a sub-field of machine learning, goes a step further by making computers automatically extract, analyse and understand the useful information from the raw data, making the decision based on this and possibly obtaining much improved results. It does not require manually extracted or handcrafted features as in the ML process[13].

These two techniques apply the following types of learning methodology: *supervised learning* and *unsupervised learning*. In supervised learning, the target values (labels) are assigned to train the data. They are used for solving classification (eg. Support Vector Machines, SVM) and regression (eg. Support Vector Regression, SVR) problems. In unsupervised learning, the labels are not provided, rather, this approach is used for making decision of association and clustering tasks (eg. Principal Component Analysis). There is a class of algorithms called Neural Networks, which can be applied both for supervised and unsupervised approaches[13].

Support Vector Regression (SVR) is a supervised learning technique and an optimization problem which defines a loss function (called ϵ-tube) to be minimized and finds the flattest tube that contains most of the training instances. A hyperplane is represented in terms of support vectors, which are training samples lying outside the boundary of the tube[14].

Neural networks (NN) compose the basis of human intelligence. The Multi-layer Perceptron is a form of deep learning. It consists of an Artificial Neural Network (ANN) that can mimic the human intelligence through a set of interconnected units, called nodes, which are grouped in three types of layers, namely, input layer (connected to the input data source), output layer (output of the whole NN) and middle hidden layer (all other intermediate layers)[15]. These nodes are connected by weights and the output signals are a function of the sum of the inputs to the node modified by a simple non-linear transfer, or activation, function. Each input signal is worked in an iterative process through the network so as to adjust the weights of the connections among the neurons. This process also permits that the output signals will present the smallest possible error between the predicted values and the true values of the variables, which can be aided by optimization techniques, such as *stochastic gradient descent, backpropagation and binary cross-entropy*[16 17].

Data pre-processing. Before the execution of the ML/DL algorithm, pre-processing of the data might be necessary. In this sense, encoding methods are useful mechanisms[18]. An example is one-hot encoding, which permits to binarize categorical inputs so that they can be considered as a vector from the Euclidean space, widely used to calculate similarities or distances between features in algorithms for classification. Another pre-processing procedure is the Principal Component Analysis (PCA), which is a mathematical algorithm that can reduce the dimensionality of the data (the number of variables) while keeping most of the information of the dataset. It serves as first step before clustering or classification tasks.

A big challenge when dealing with deep learning algorithms is imbalanced datasets, that is, datasets with a high class imbalance. Most techniques ignore the minority class, which is typically the most important one, resulting in a poor correctness on that class. In this context, another pre-processing technique is the SMOTE (Synthetic Minority Oversampling Technique)[19]. It works by selecting examples that are close in the feature space (the k nearest neighbours), drawing a line between these examples and, then, drawing a new sample at a point along that line. This procedure is useful for creating as many synthetic examples for the minority class as needed. It is an effective approach that creates plausible samples and also serves as a type of data augmentation for the minority class, aiding improve the algorithm predictions correctness, when the size of the dataset is small[20].

Another technique that might help managing imbalanced datasets is the use of *oversampling minority class* and *undersampling majority class*. In the first case, *random oversampling* is a non-heuristic algorithm whose objective is to balance class spreading through the repetition of minority target instances. Although it might bring the possibility of overfitting, it is well suited when there is not much data to work with. In the opposite direction, *random undersamplig* seeks to balance target distributions by randomly dropping majority class instances[21].

Algorithm Evaluation. Regarding the execution of the algorithm itself, two processes take place to fit and evaluate the model, namely, training and test, which involve a split of the data into different sets. *Generalization* is a term specified to describe the ability of a model when the latter is exposed to new or unknown data. This generalization is related to the concepts of *overfitting* and *underfitting*. There are some metrics that permit to measure the correctness of the model after the execution of the algorithm, such as accuracy, precision, recall and F1-score[22]. These metrics can be summarized in a *Confusion Matrix*, which brings a better comprehension of what the model is predicting correctly and what kind of errors it is making, as shown in Figures 5 and 9. The combined use of these evaluation metrics aid dealing with situations where there are imbalanced datasets.

Related Work. The related work is compared with the current study in Table 1. The first three criteria utilized for comparison regards to the main subjects with which this research is dealing and the last criterion is related to a feature that our method proposes to achieve within its respective implementation. Therefore, these criteria were considered relevant for a better comprehension of how our research is relationed to the state of the art.

TABLE 1 Comparison of related studies.

Study	Emergent behavior prediction	Emergent behavior design process	Deep learning mechanisms	Unrestricted number of constituents
12			✓	
23		✓		
24		✓		
25		✓		
15	✓		✓	
26	✓		✓	
27	✓	✓	✓	
This study	✓	✓	✓	✓

Some studies in the literature cover the adoption of Neural Networks (NN) when dealing with emergent behaviors in SoS. Raman and Jeppu adopted NN in the attempt to look for formal violations in the emergent behavior of a swarm of autonomous Unmanned Aerial Vehicles (UAVs) flying in formation, and dynamically changing the shape of the formation, to support varying mission scenarios[15]. Alzahrani et. al present a model of Microgrid using Feedforward Neural Networks. This NN characterizes the behaviors and outputs of the constituents of the SoS, which work collaboratively to achieve the goal of supplying power to the electric load[26].

The emergent behavior design process has also been covered in the literature. Garcés and Nakagawa describe a systematic process to establish, model and validate missions of SoS and also incorporate them into Reference Arquitectures (RAs) evaluated and applied in the healthcare domain. Here, the behaviors in the lower levels of the SoS are defined and, following the bottom-up approach, the higher levels can be modeled until the general missions are abstracted. As a result, a tree of emergent behaviors of the RA is defined[24].

Regarding the related work, as it can be inferred from Table 1, this study aims to advance the state of the art by combining different approaches covered separately in the prior work, such as the emergent behavior design process and deep learning mechanisms, plus adding the possibility of adding as many constituents as needed, a characteristic that was not available in those studies. Thus, the current research aims to advance at this comprehension of how a SoS evolves over time and the effects in the emergent behaviors derived from it.

3 | MACGYVER PREDICTOR SYSTEM

The main aim of this work is to support systems and software engineers with the design of emergent behaviors in SoS context. The inspiration for the name of the system is MacGyver, a 1990's TV show in which the main character had the ability of combining single devices or objects, such as bubble gum, powder and cables to form something more complex as a gun or bomb. We foresee it, in an exaggerated manner, as a desirable ability for systems and software engineers when we think about SoS[9,4]. In reality, engineering SoS is similar to that: we currently rely on (and we are limited to) the creativity of engineers to combine the capabilities (called herein as *features*) offered by the candidate constituent systems to draw (simple/weak) emergent behaviors. In that sense, Macgyver Predictor System adopts ML algorithms to support engineers with the design of those behaviors.

Regarding SoS design, both the approaches (*top-down* and *bottom-up*) can be really complex to be executed by human engineers. For example, in a (*bottom-up*) approach, the solution heavily relies on the engineer's creativity. S/he needs to conveniently combine the features available in the constituents set towards achieving the emergent behavior that executes the established goal, as MacGyver used to do to build his artifacts.

We envision the possibility of teaching a deep learning mechanism about how humans combine a set of features to design a desired behavior (a *bottom-up* approach). This mechanism could be iteratively enhanced as more data is added and validations are made, as to increase the correctness of the algorithm predictions. This solution was prototyped in a domain-specific context, i.e., the constituents available in the dataset that feeds the mechanism are from the Smart Grid and Smart Cities domains. However, given the non-linearities that the proposed algorithm can deal with, we suggest that it could be possible to virtually add and combine constituents from different domains in this proposed solution.

3.1 | Datasets Construction, Training and Pilot Test

Datasets are fundamental elements to teach a deep learning mechanism on how to perform any action. In our case, a dataset is a set of tuples that could be used by the deep learning mechanism to learn how engineers design emergent behaviors in a *bottom-up* approach. Each tuple should contain a label to designate which emergent behavior the tuple describes and the set of features used to compose the behavior.

We realized we could build a dataset with real emergent behaviors described in the specialized literature. Then, we conducted a Systematic Mapping Study (SMS) on simulation of emergent behaviors in Systems-of-Systems, as discussed in more depth in[9] to collect such information and build our dataset (this SMS is not in detail here since communicating its results is not the focus of this manuscript. Details can be seen in Chapter 3 of the Master's Dissertation [yet to be published] and in our prior work[9]).

From the set of included studies in that systematic mapping, we selected 14 studies that reported SoS in the domain of Smart/Power/Micro Grids and Smart Cities. They represented the majority of the studies (22,5%) and we could use them to extract (i) the constituents, (ii) the capabilities and (iii) the emergent behaviors, since they had some constituents in common. All the analyzed studies reported only Simple or Weak emergent behaviors. We manually extracted the information about the SoS constituents, their basic features and emergent behaviors to elaborate an initial dataset with which the deep learning algorithm was trained and evaluated. An excerpt of this extraction can be seen in Table 2.

A database was conceived to store the elaborated dataset. Four main entities were modeled, namely SoS, constituent, basic feature and emergent behavior. 13 SoS, 43 different constituents, 93 basic functionalities and 37 emergent behaviors were catalogued in that step. The database scheme, SQL scripts, as well as the datasets generated and pre-processed during this step are available here[‡].

The original dataset consisted of the tuples with SoS constituents and their respective emergent behaviors described in terms of the individual functionalities used to compose it, as described in Table 2, which represents an excerpt of the full dataset. In this case, one example of interaction among constituents through the mechanisms of their basic features and the resulting emergent behaviors can be derived from the expression 1, where c stands for constituents and f for features. For each constituent $c_i$there is one or more features f_j that, combined, form the emergent behavior eb_n.

$$c_1.f_1 + c_1.f_2 + c_3.f_4 + c_4.f_5 = eb_3 \tag{1}$$

The pre-processing of the dataset and implementation of the deep learning algorithm was written in Python, as it provides specialized libraries for deep learning, such as Keras and Tensorflow. The chosen algorithm was Multilayer Perceptron (MLP) for multiple label classification, as different sets of constituents might exhibit mutually non-exclusive emergent behaviors simultaneously. This was also the algorithm chosen due to the non-linearities that it is able to deal with, once the dataset has the potential to scale up as to gather a huge amount of information regarding SoS from different domains.

The NN used as learning algorithm in this step has 68 nodes in the input layer (number of labels of constituents in the dataset after the pre-processing) and 37 nodes in the output layer (one for each label of the possible emergent behaviors in the dataset). As there were only 15 samples in the original dataset, the SMOTE technique was applied as to extend the number of inputs up to 100, which was the number that presented the best results at this point of the research method. Two middle hidden layers were present in this NN and optimization techniques were utilized during the training process to deal with the imbalanced data from the original dataset as well as prevent overfitting of the model and enhance accuracy of the predictions. The split of the whole

[‡] https://github.com/kanancastroo/mac_gyver_predictor_extended_repository/

TABLE 2 Excerpt of the dataset

Constituent (c)	**Basic Feature (f)**	**Emergent Behavior (eb)**
(c_1) Battery storages	**(f_1) Power Supplier**: supplies the power to the electric grid in the absence of renewable sources generation **(f_2) Energy storage:** stores energy back from the main grid or renewable sources	(eb_3) Load-generation balance
(c_3) Photovoltaic cells (PCs)	**(f_4) Management of Energy generation:** generates energy from irradiance	
(c_4) Wind energy system	**(f_5) Management of Energy generation:** generates energy from wind velocity	

data was made with 80% for training and 20% for tests, with validation set as 30% of the training data. During the learning phase with the augmented dataset, the overall accuracy performed by the algorithm was of 82,4%. In other words, given a pre-determined set of features delivered as input to the algorithm, the algorithm was capable of correctly suggesting 82,4% of the emergent behaviors that could be feasible to be drawn as the combination of the input features.

However, accuracy is not a good metric of correctness for imbalanced datasets. Thus, in the context of the next step of the proposed method (the prototype of the web-application), this correctness evaluation was expanded, as to permit assess the results per emergent behavior present in the database, what occurred during the evaluation with experts.

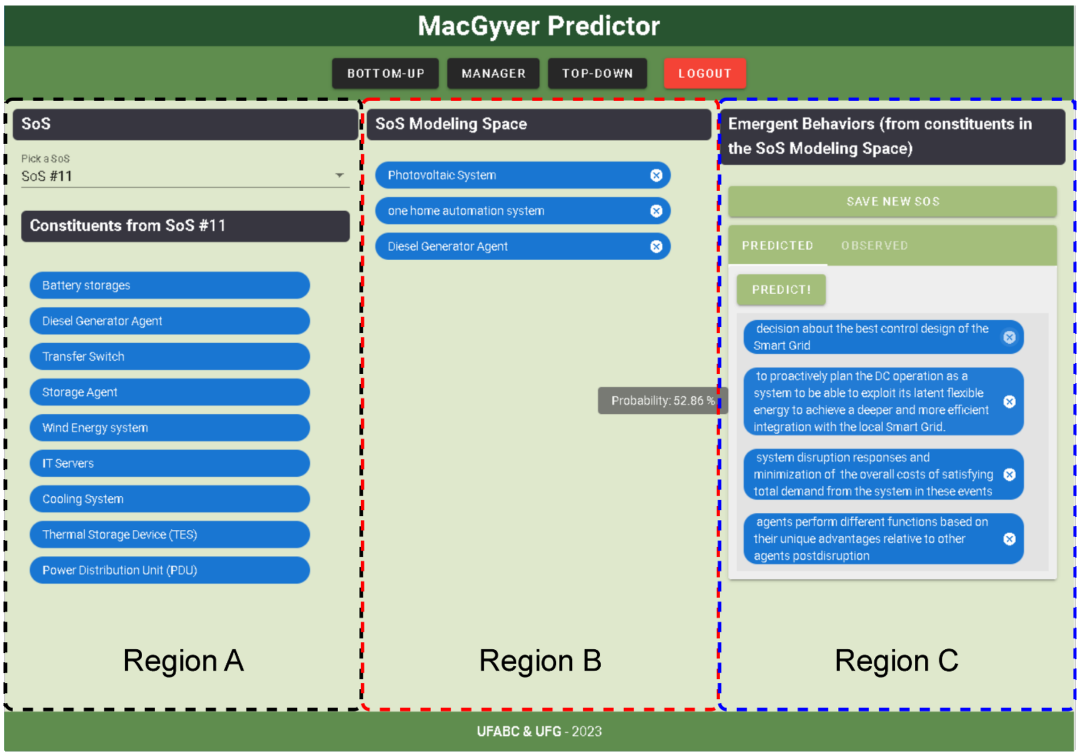
3.2 | Prototype of the Web-Application

Contextualizing the next step of the research method, a web application was developed and evaluated to permit engineers compose different SoS, using their actions as inputs to the deep learning algorithm to learn, enriching the dataset and enabling the prediction of possible emergent behaviors in other sets of constituents. The chosen front-end framework was Vue.js due to its light weight and componentization properties; the back-end was developed in Flask, as its core language (Python) was used in the main algorithm of the solution. The database was developed in Structured Query Language (SQL) using Postgres, as consistency in the data was required, that is, as constituents or behaviors are added or removed, the database should be updated accordingly. This choice was taken also due to the integration opportunities with the back-end algorithms. The communication between front-end and back-end is performed via HTTP requests since an API and the deep learning algorithms run on the back-end, which was deployed in a server along with the database, for performance purposes. The prototype is available for external tests in an external link[§] and the source-code of the system is available here[¶].

An example of the front-end interface is shown in Figure 2. The *Bottom-up* approach was idealized for the user's experience while dealing with SoS composition and behavior prediction. A set of existing SoS is available in the left panel (region A) and the user can choose the desired constituents to compose his/her own SoS in the SoS Modeling Space (region B). In an interactive manner, inferred emergent behaviors (and their probabilities), predicted by the algorithm, appear in the right panel (region C). These suggested possible emergent behaviors (which could be any of the existing in the database) derived from this association of constituents and combinations of their respective features. The user can validate these predictions and also add their own observed emergent behaviors for that SoS. This composed SoS can be saved into the database as soon as s/he is satisfied with designing process.

The web-application also has a manager mode, where the user can directly act over the database, executing CRUD operations on constituents, basic features, emergent behaviors and the SoS themselves. Relationships among these entities can also be managed. We believe that the recurrent use of both the bottom-up and manager mode can contribute to enhance the correctness of the deep learning algorithm predictions over the interactions.

[§] https://macgyver-predictor.up.railway.app/
[¶] https://github.com/kanancastroo/mac-gyver-predictor/



decision about the best control design of the Smart Grid

to proactively plan the DC operation as a system to be able to exploit its latent flexible energy to achieve a deeper and more efficient integration with the local Smart Grid.

system disruption responses and minimization of the overall costs of satisfying total demand from the system in these events

agents perform different functions based on their unique advantages relative to other agents postdisruption

FIGURE 2 Bottom-up approach

3.3 | Implications of the Prototype

Understading emergent behaviors in SoS is important as it allows the managers of such systems to act in a way that the latter fulfill the requirements of the former and respond appropriately according to their missions and purposes, given the critical environment they usually are and the level of reliability and accuracy to which they are required.

The results derived from this system could be of great assistance for SoS Engineers. Although normally the desired emergent behaviors are already known in advance by the engineers, the SoS might suffer from changes in its architecture during runtime, for example, with the joining or leaving of one or more constituents, or with the internal change in the architecture of a constituent. The new emergent behaviors derived from this new SoS architecture might be undesired or unpredicted by the engineers, or even worse, these behaviors might lead the SoS to a misbehavior, which can affect seriously its original goals and lead to failures or losses. Also, when the formed SoS has too many constituents or these constituents belong to different domains, this solution could aid obtaining more findings than a manual process, due to the human limitations in dealing with a combinatory explosion or the non-linearities derived from the different domains from which the constituents come.

Thus, we claim that DL mechanisms could overcome the human capacities to glimpse behaviors, as our solution proposes a mechanism to help SoS engineers to know in advance the possible emergent behaviors resulting from the combination of features of constituents in a given SoS.

4 | EVALUATION WITH EXPERTS

For receiving feedback about the support the system could offer to SoS experts and engineers during the design of emergent behaviors, we conducted a survey study. The evaluation involved SoS experts that used the system to design emergent behaviors in a bottom-up style so that the algorithm could continuously learn, enhancing the correctness of the predictions, and the dataset could be enriched with new samples obtained during the evaluation. The experts used the system according to a established set of tasks and answered a questionnaire elaborated to evaluate the solution. Next, the results obtained were analysed and an evaluation of the whole process was reported. Figure 3 depicts the overall process of this evaluation which was structured in four well defined steps (Planning, Execution, Analysis and Reporting) and further detailed in the next sections. The protocol was inspired by the guidelines proposed by Kasunic[28], Linåker et al.[29] and Seide et al.[30] and prior studies conducted by our group[31,32].

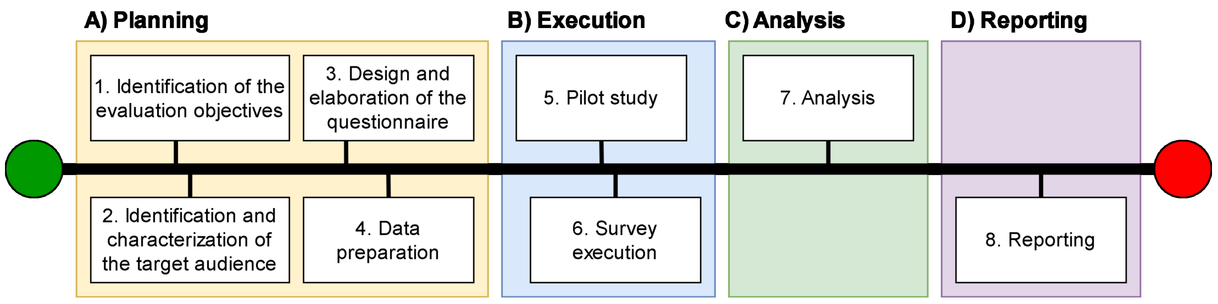

A) Planning
B) Execution
C) Analysis
D) Reporting
1. Identification of the evaluation objectives
3. Design and elaboration of the questionnaire
2. Identification and characterization of the target audience
4. Data preparation
5. Pilot study
6. Survey execution
7. Analysis
8. Reporting

FIGURE 3 Evaluation workflow (adapted from[32]).

4.1 | Planning

Scenario: The scenario comprises the use of the system Mac-Gyver Predictor to design emergent behaviors in a bottom-up style. The procedures involved in this step applied the Goal-Question-Metric (GQM) principles, following the guidelines presented in the model of Basili[33].

1. Identification of the evaluation objectives

Goal: The proposed solution (MacGyver Predictor) aims to aid experts and SoS engineers in the bottom-up design of emergent behaviors. In this context, the main objective of this evaluation was to assess the feasibility of the predicted behaviors, the correctness of the predictions delivered by the algorithm and the functional suitability of the system.

Rationale: As previously explained in Sections 1 and 3.3, the current manual process of bottom-up design of emergent behaviors might lead to waste of resources, sub-optimal architectures, non-linearities issues and late response in the redesign process. Thus, this evaluation seeks to verify if this proposed solution can help facing these issues.

The current evaluation combined *quantitative* and *qualitative analysis*[34], as described in the questions below:

RQ1: Are the emergent behaviors predicted feasible?

Rationale: We adopted the following definition for *feasible emergent behavior*, inspired by[35]:

Feasible emergent behavior: occurs when there is, in the constituent set, all the features necessary to execute the subgoals that compose the behavior predicted for the SoS. A feasible emergent behavior is one suggested by the system whose set of features, when combined, can deliver that emergent behavior. For example, the combination of features presented in Table 2 and the respective emergent behaviors achieved.

Metric: Reported feasibility by the experts. This is a quantitative and qualitative metric that was evaluated according to the answers in the questionnaire by the experts, with respect to the correctness and completeness of the behaviors predicted.

RQ2: How effective (correct) are the algorithms predictions?

Rationale: The algorithm is evaluated by means of training, validation and test sets. Thus, the more predictions delivered that are equal to the behaviors expected, the more effective (correct) the algorithm is.

Metric: The results obtained from the confusion matrices (accuracy, precision, recall and F1-Score), as the experts use the system. This is a quantitative metric to evaluate the correctness of the algorithm.

RQ3: Does the system fulfill its purpose with respect to the established objective (aiding in the bottom-up design of emergent behaviors)?

Rationale: Here, the feedbacks will permit to see if the proposed method, materialized through this system is a good starting point in the automation of the design bottom-up of emergent behaviors by SoS experts.

Metric: Feedbacks from the experts in the questionnaire. This is a quantitative and qualitative metric, where the experts answered whether the system fulfilled its purpose or not.

2. Identification and characterization of the target audience

The target audience included researchers and professionals with experience of at least one year in SoS.

3. Design and elaboration of the questionnaire

The questionnaire was organized into (i) demographic questions [DQ] (ommited here for sake of space restrictions) as to characterize the profile of the participants and (ii) study questions [SQ] (detailed in Table 3), to obtain the feedbacks from the experts during the use of the system. SQ contained affirmations which were evaluated by the experts using a five Likert-Scale [LS] ranging from ”totally disagree” to ”totally agree”. SQ also contained OEQ as to permit the experts elaborate more over each answer given in the Likert-scale questions, giving examples in each case. OEQ also contained a question asking for suggestions of improvements, corrections, implementations, among others, for the system.

TABLE 3 Study Liker-Scale (LS) and open-ended questions (OEQ)

ID	Study Questions (SQ)
LS1	The predicted behaviors were feasible.
LS2	Many of the behaviors that were initially predicted by the algorithm needed to be excluded before saving the SoS generated.
LS3	Many behaviors were not predicted by the algorithm and needed to be inserted manually through the option “Observed”.
LS4	The system fulfills its purpose in aiding SoS engineers and experts in the bottom-up design of emergent behaviors.

4. Data preparation and Material

The web system was delivered to the experts with an initial database, from where the evaluation was initiated. This data was prepared from the findings obtained in the previous step of the proposed research method. Thus, 13 SoS, 43 different constituents, 93 basic functionalities and 37 emergent behaviors conceived the initial database over which the experts could compose their own SoS, add new ones and evaluate the whole solution.

4.2 | Execution

5. Pilot Study

The evaluation would be performed as follows: the participants were asked to watch a brief tutorial#, explaining how the MacGyver Predictor worked, to guide these participants about how to operate the tool so that they could succeed in their endeavor. Next, they would use the system and answer the questionnaire. One expert was invited to run the pilot study, after which we concluded that no modifications were needed for the survey execution.

6. Survey Execution

The survey execution was performed through the direct invitation of professionals who had prior experience with SoS. 17 people were invited, 7 of which attended to the evaluation execution. Besides the collecting of their answers in the questionnaire,

https://youtu.be/XWrlONevgQk

statistics were also obtained through the usage of system to permit evaluate the correctness of the algorithm predictions in a behavior basis.

As these participants utilized the system, we noticed that the correctness metrics were not bringing the expected results. Thus, after reviewing and evaluating possible alternatives, the core algorithm of the system (Multilayer Perceptron) was replaced by a Support Vector Regression for a new round of the evaluation. Here, people who were not able to attend in the first invitation were asked to participate in this second round. Then, three more experts performed the evaluation, totalizing 10 people in the survey execution. The overall process was conducted between February 15th, 2023 and March 17th, 2023, with March 9th being the separation date between rounds one and two of the evaluation process.

4.3 | Analysis

The data obtained from the evaluation was analysed quantitatively and qualitatively. Correctness metrics were collected from the system to evaluate the output of the predictions in a behavior basis and Grounded Theory was applied through open coding and axial coding over the LS and OEQ answered by the participants in the questionnaire.

4.4 | Reporting

Going back to the evaluation RQ's and summarizing the results:

RQ1: Are the emergent behaviors predicted feasible?

In the context of the quantitative analysis, three affirmations were evaluated by the experts through a Likert-Scale (LS1, LS2 and LS3). According to these evaluations, the feedbacks from the experts were mostly positive, with a major of votes considering the emergent behaviors feasible. Not many interventions were needed from them as to exclude or insert an emergent behavior. These findings reinforce the promising approach that the system can become as an initial effort towards the automation of the bottom-up design of emergent behaviors.

With respect to the coding process, three OEQ were asked as to permit them elaborate more on the LS evaluations and their answers were subject to the coding process. One category directly related to the evaluation RQ's was identified, namely, *emergent behavior*. It was split into subcategories containing perceptions, requirements, improvements aspects, among others, regarding this category, as can be seen in Figure 4.

In the context of perceptions (EB1), there were some codes that highlighted that the judgement of the feasibility of the emergent behaviors depends on the prior experience of the engineers, as we commented in Section 3. They argued that the same applies to the quality of the modeling and identification of the features that are necessary to compose a certain emergent behavior (EB1.1). The simplicity of predictions (EB1.2) was another topic discussed in this subcategory, where it was mentioned that there are some emergent behaviors that might be never predicted, thus the complete list of features might be never reached.

Included in the influenting factors (EB2), it was discussed how the relations of the emergent behaviors with the SoS global goal impacts on whether the former is feasible or not in that SoS (EB2.1). The quality of the relations among the features of the constituents and the emergent behaviors was also mentioned in this context (EB2.3).

An expert pointed that emergent behaviors are strongly affected by attributes of quality, performance and availability of constituents (EB2.2). He also highlighted the importance that the stakeholders of the SoS be open to put their constituents to operate in a way that the latter helps to enable the emergent behavior (EB4.1). This was a requirement identified and included in the respective subcategory (EB4).

Improvements (EB3) were also suggested as some experts brought the need for evaluation in real scenarios to compare the outputs with the ones from the system (EB3.1). They also mentioned future simulations of emergent behaviors of type ”weak” (EB3.2) (they considered the ones of the system as type ”simple” (EB1.2)), deepening in description of those behaviors as to permit evaluate better whether are feasible or not (EB3.3). Finally, one expert asked for the possibility of modeling relations among the emergent behaviors themselves, perhaps permitting to know which ones generate positive/negative feedback loops in the SoS (EB3.4). The following excerpt exemplify these findings:
(EB3.1) - [Participant 8]

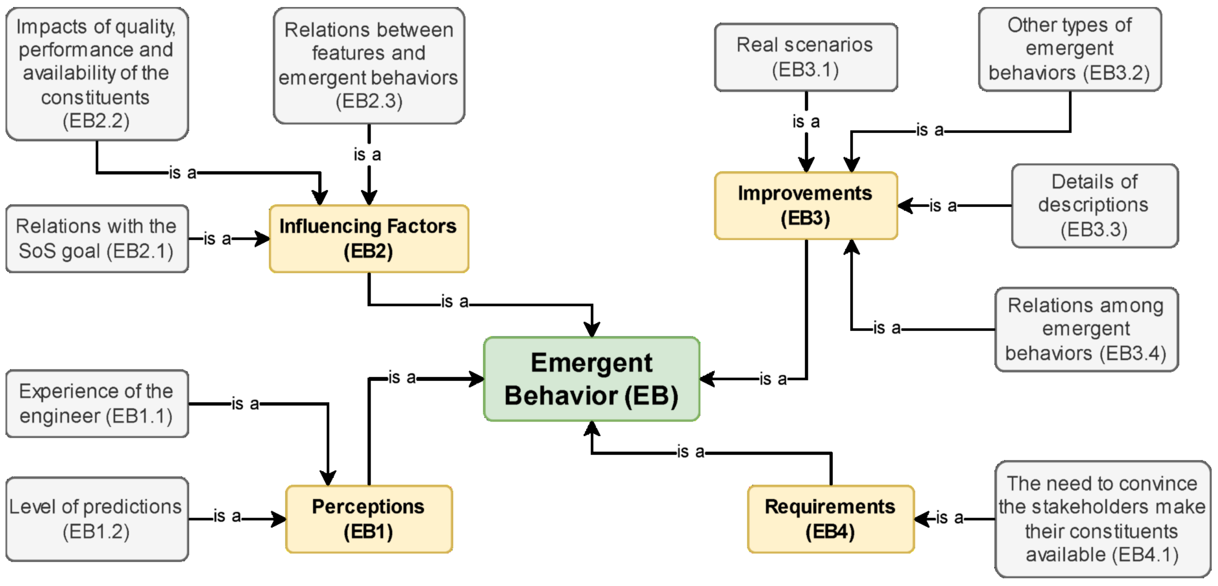

Impacts of quality, performance and availability of the constituents (EB2.2)
Relations between features and emergent behaviors (EB2.3)
Relations with the SoS goal (EB2.1)
Influencing Factors (EB2)
Real scenarios (EB3.1)
Other types of emergent behaviors (EB3.2)
Improvements (EB3)
Details of descriptions (EB3.3)
Relations among emergent behaviors (EB3.4)
Emergent Behavior (EB)
Experience of the engineer (EB1.1)
Level of predictions (EB1.2)
Perceptions (EB1)
Requirements (EB4)
The need to convince the stakeholders make their constituents available (EB4.1)
is a

FIGURE 4 Feedbacks related to the category emergent behavior (RQ 1)

"I imagine that it would facilitate the discovery of these new behaviors further investigations through simulation of the SoS or application in real scenarios, where the behaviors could be observed by experts in the application domain."

In summary, according to the Liker-Scale evaluations and the coding process over the OEQ, in general, the emergent behaviors predicted are feasible indeed. However, we highlight that those predictions are still simplistic and cover simple cases, as there are a series of influencing factors that need to be taken into account, relations among emergent behaviors, applications in real scenarios and conversations with stakeholders to permit a better integration among constituents. Thus, it might be possible, in the future, to predict more complex emergent behaviors.

RQ2: How effective (correct) are the algorithms predictions?

With respect to the correctness metrics, as previously mentioned, we noticed that the results were not evolving as expected. More specifically, accuracy, recall and F1-score did not seem to improve along the time when we checked the results per behavior predicted by the system. This result was generalised over the outputs of the algorithm and is exemplified in Figure 5. As pointed there, the metrics did not seem to converge, rather, they randomly changed as the experts used the system. An example of confusion matrix is shown in Figure 6. The algorithm roughly predicted the non-occurrences of the behavior (as there are many more samples for this class [Class 0]), exhibiting a certain accuracy, precision, recall and F1-score for these cases, while we wish, instead, to reach optimized correctness metrics in the other case, that is, the occurrences of the emergent behaviors (class 1 in the images). In practice, this means that the algorithm would not perform well for new data, that is, it would not generalise well and the predictions would be mostly incorrect.

We argue that these results were caused by the following factors (this list is not exhaustive): (i) The Multilabel SMOTE did not work as expected when we were trying to rebalance the dataset the fed the deep learning algorithm. We were taking the outputs (emergent behaviors) altogether and the number of zeroes and ones did not get close to each other; (ii) There were some cases in which there was an only example of combination of constituents that raised a certain emergent behavior. Thus, the correctness of the algorithm predictions was harmed due to the absence of more cases for that emergent behavior; (iii) The MLP needs a large amount of data to generalise well[13], thus leading us to conclude that the quantity originally present in the dataset was not enough yet; (iv) The number of variables (columns) was greater than the number of samples (rows) in the dataset the fed the algorithm.

We realized that the correctness of the algorithm predictions was not adequate. Furthermore, it would not aid SoS experts design emergent behaviors in a bottom-up style. Thus, having these preliminar results, we decided to interrupt the execution of the evaluation and ask the opinion of another expert, who did not and would not participate in the evaluation. Inspired by his

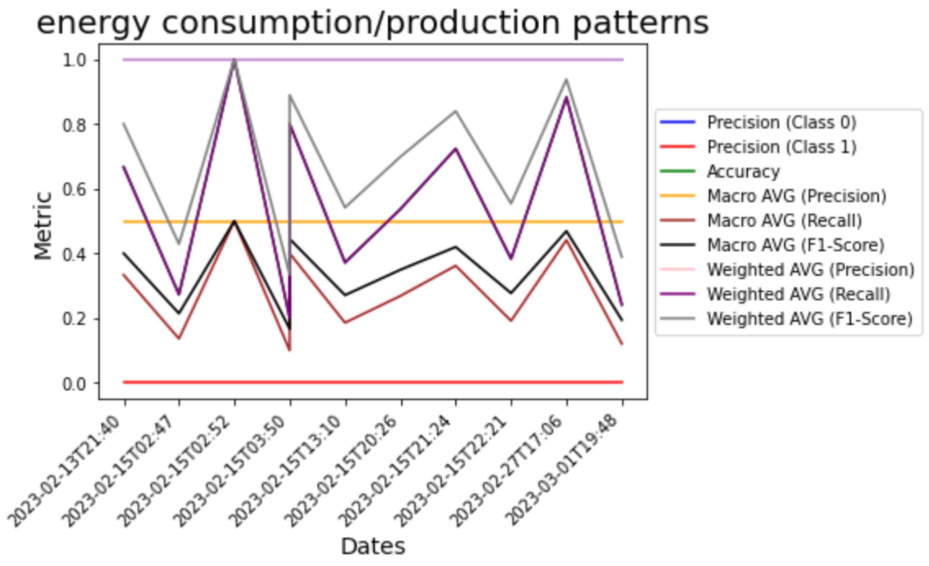


FIGURE 5 Example of correctness metrics in the first part of the evaluation execution (MLP)

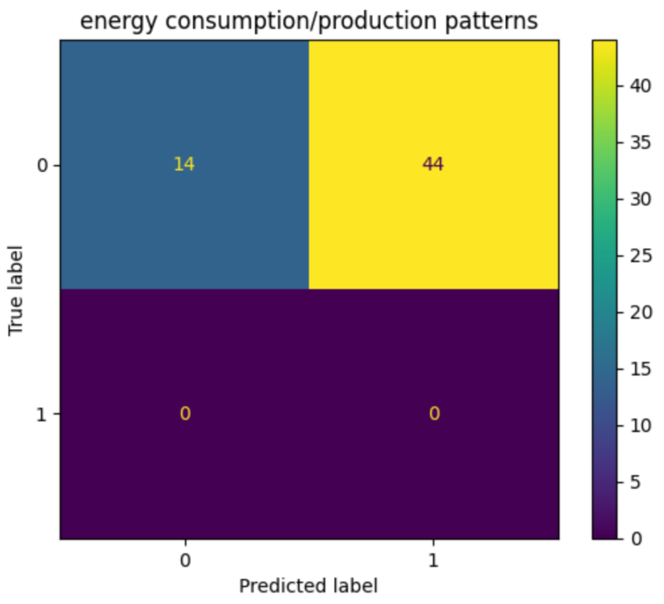


FIGURE 6 Example of confusion matrix in the first part of the evaluation execution (MLP)

advice, we took a copy of the current version of the database as of the interruption and executed a PCA analysis over the data. At the time, the dataset that would feed the algorithm had 109 samples, 44 input variables (possible constituents for composition) and 38 outputs (possible emergent behaviors predicted). The explained variance of the data is shown in Figure 7. As can be seen, with 10 components we can describe around 90% of the information of the data. This dimension reduction was then applied over the input data to use the latter in the new algorithm that would make the predictions.

To decide which algorithm to use for the next part of the evaluation execution we performed a series of tests with potential candidates and evaluated them through the classification report in a behavior basis. The algorithms utilized were linear and logistic regressions, support vector classification and regression (both with kernel ”linear”, decision tree, random forest, K-nearest neighbours and Gaussian Naive Bayes). The results varied depending on the behavior, where some algorithms performed

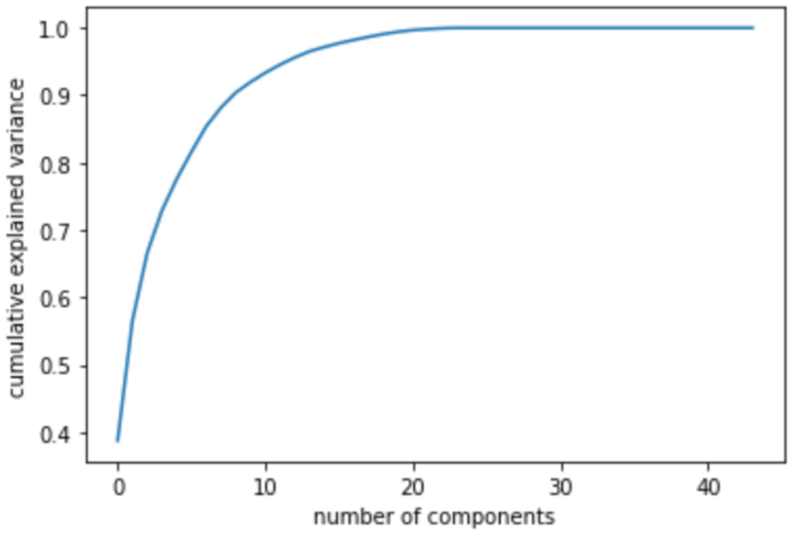


FIGURE 7 The explained information regarding the input data

better for some behaviors and worse for others. However, overall, support vector regression with kernel ”linear” presented the best results and was then chosen as the replacing algorithm for the core of the system predictions mechanism.

That choice can also be explained with an example depicted in Figure 8. It is was originated from a PCA of the input data with three dimensions as thus it can be plotted in a tridimensional space. It is a representation of the reduced dimension input dataset with respect to the predictions for a certain behavior. The blue dots represent scenarios where that emergent behavior does not occurr, while the yellow dot describes a context where that behavior happens. This reduced representation can be converted back to the original representation of the dataset as to know which combination of constituents is responsible for the referred emergent behavior. In this image, a hyperplane can be used to separate both classes (blue and yellow dots) and, hence, this is a scenario where SVR could work well in future predictions.

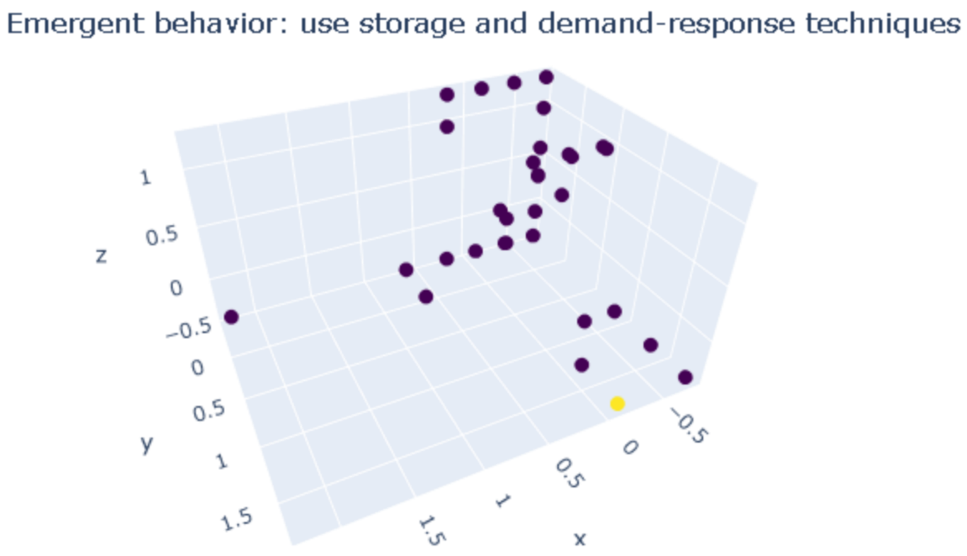


FIGURE 8 Example of correlation between the reduced input space and an emergent behavior

After choosing the new algorithm, we adopted the strategy to perform the predictions individually per behavior in the outputs. In other words, we would not take all the targets together as it had been done with the MLP, due to difficulties in the resampling strategy. We divided the dataset into train and test splits. The resampling strategy this time was not SMOTE, rather, we applied minority random oversampling and majority random undersampling, as to equalize the samples of each class in each split.

Although predicting each behavior individually, we adopted a regressor chain during the fitting of the algorithm, so each prediction in the training phase would take the previous predictions of the algorithm into account. This was done due to the fact that, given the nature of the SoS, there might be some correlation among emergent behaviors, that is, they are not strictly independent of each other. During the fitting process, from the 38 existing emergent behaviors in the database, 18 were not able to be fitted as there were not enough examples of each class. This reinforces the need for even more data as to enhance the algorithm.

With the replacement of the core algorithm completed, we re-invited the people who could not participate in the first part of the evaluation execution. Three experts attended to the process. Figure 9 shows an example of correctness metrics after finishing the second part of the evaluation execution. For this emergent behavior, the accuracy during the training phase was of 71,2% and the correctness metrics for the predictions were overall above 80%. The confusion matrix is depicted in Figure 10 Finally, the resampling strategy this time took effect and we were able to rebalance the dataset accordingly.

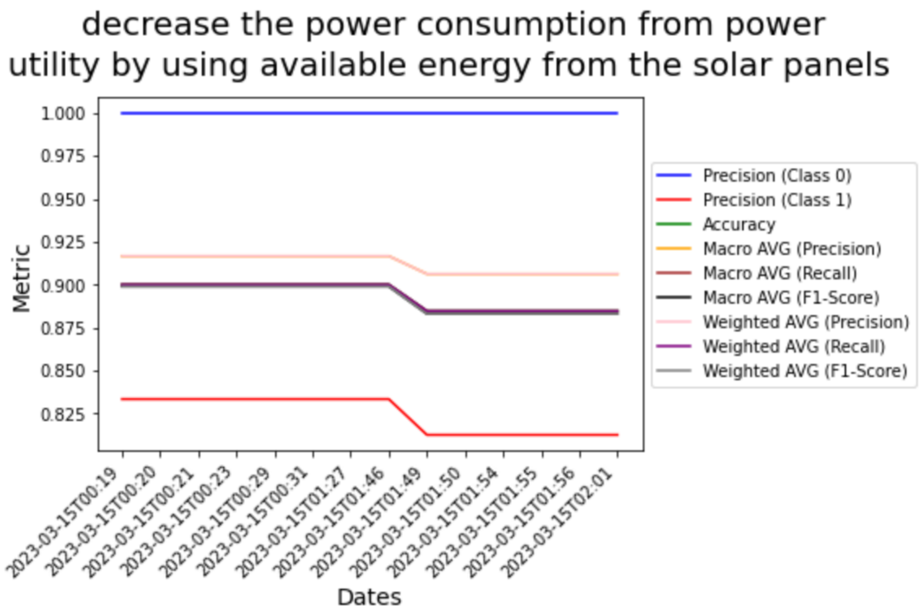


FIGURE 9 Example of correctness metrics in the second part of the evaluation execution (SVR)

Despite these promising results, unfortunately they were minority among the existing emergent behaviors in the database. Only 9 out of 38 (23,6%) exhibited metrics in the same fashion of the images previously presented. Although it is a low score, this implementation demonstrated to be a promising technique and it can be improved as more data and more balanced samples are collected, arising possible opportunities for future works as explained next.

We applied the same algorithm (SVR) to each behavior prediction. Figure 8 presents a scenario where one could draw an hyperplane to separate the two classes of the outputs (blue and yellow dots). But as the data grows, there could be more yellow dots among the blue ones in that figure, so that perhaps it would be difficult to manage separation hyperplanes. Thus, in this initial moment where we performed evaluations of potential algorithms, one could establish a procedure to evaluate which algorithm is adequate to which emergent behavior, depending on the structure of the data. In the limit, with the correct amount and quality of data and the correct algorithm for each behavior prediction, the correctness can increase significantly. Moreover, this might be a context where the MLP originally proposed could take place, as it would have a more ideal environment to run and could deal with potentially multiple non-linearities in the data, learning from it as to make the predictions. All the artifacts produced during this quantitative analysis are available in the Appendix Section.

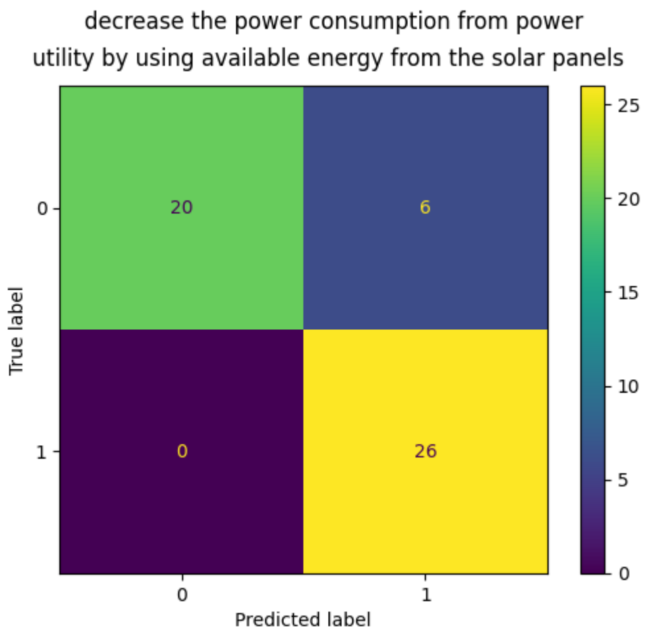


FIGURE 10 Example of confusion matrix in the second part of the evaluation execution (SVR)

RQ3: Does the system fulfill its purpose with respect to the established objective (aiding in the bottom-up design of emergent behaviors)?

In the context of the quantitative analysis, one affirmation was evaluated by the experts through a Likert-Scale (LS4). The experts reported that the system fulfils indeed its purpose of aiding SoS engineers and experts in the bottom-up design of emergent behaviors.

Regarding the coding process, two OEQ were asked as to permit them elaborate more on the LS evaluation and their answers were subject to the coding process. One category directly related to the evaluation RQ's was identified, namely, *system*. It was split into subcategories containing perceptions, requirements, improvements aspects, among others, regarding this category, as can be seen in Figure 11.

In regards to the system category, the evaluation brought some perceptions (S1) by the experts. There were positive and negative feedbacks (S1.1) due to the simplistic cases that the system can solve, the current limitations considering the initial stage that the mechanism is and the potential it has to evolve. These feedbacks were reflected in their expectations reported in the questionnaire (S1.2). But they recognized that the algorithm tends to be enhanced as more people use the system (S1.3).

A series of necessary corrections (S2) were noted from the feedbacks of the experts, as critical bugs during the use of the system were reported. The interactions between the backend and the database need to be improved (S2.1) as some predictions of the system ended up being deadlocked inside a loop (S2.2), some operations in the Manager mode were not possible to be finished and some dangerous results could be obtained from the system, as the prediction over a SoS with no constituents (S2.3) or the possibility of saving a SoS with only emergent behaviors and nothing else (S2.4). They also pointed, as a requirement (S3), the need for a better granularization of the features of the SoS (S3.1), which might imply into a re-modeling of the database architecture. These are valuable observations that certainly will be taken into account when continuing this project.

Remarkable were also the topics collected for improvements (S4), where the experts claimed the need for a better descriptions of the SoS goals in the system as to permit a better comprehension of how the constituents and emergent behaviors are related to them (S4.1). They also asked for better quality of predictions (S4.2), which shows the need for improvements in the core algorithm of the solution. Besides this, there were ideas for advances in the database (S4.3), in order to avoid inconsistencies that might cause the errors previously mentioned; upgrades in the frontend (S4.4), with highlighted colors in some elements; UX (S4.5), where some unfriendly usability flows were discussed as well as their solutions.

Finally, some suggestions (S5) were also listed by the respondents, such as new scenarios (S5.1) and other domains (S5.4) for the simulations, applications in the predictions of environmental disasters (S5.2), as this seems to be a trending topic in SoS.

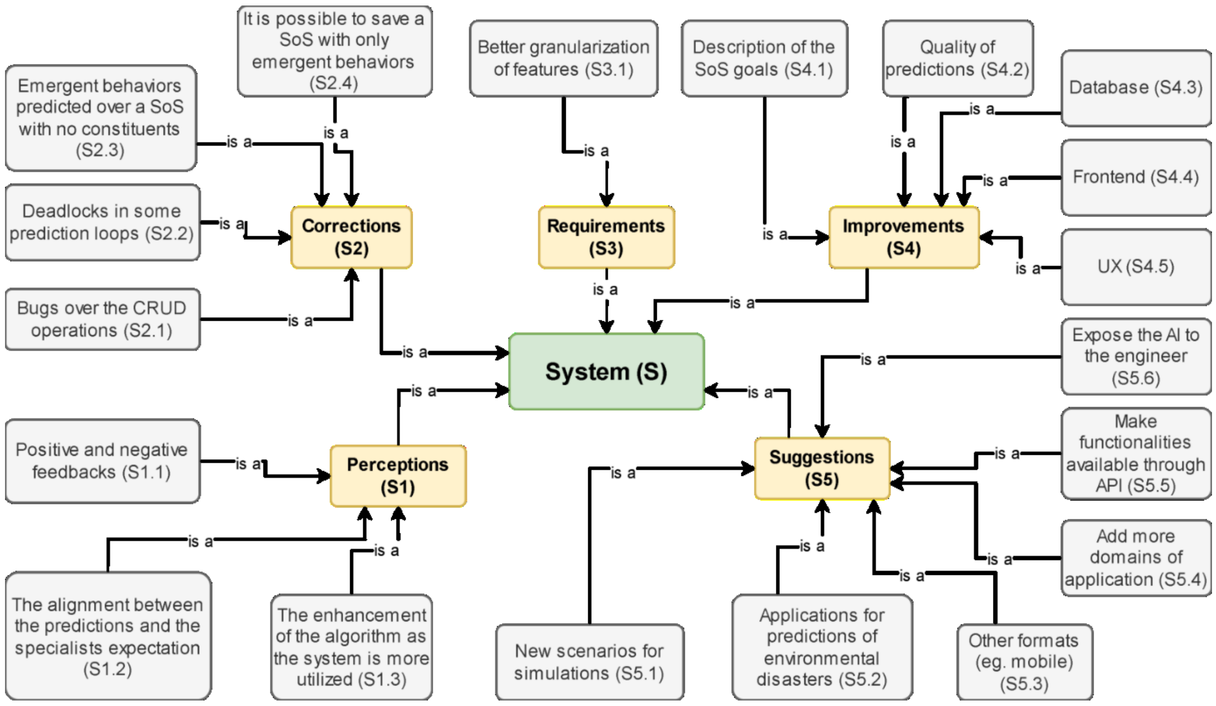

It is possible to save a SoS with only emergent behaviors (S2.4)
Better granularization of features (S3.1)
Description of the SoS goals (S4.1)
Quality of predictions (S4.2)
Database (S4.3)
Emergent behaviors predicted over a SoS with no constituents (S2.3)
Frontend (S4.4)
Deadlocks in some prediction loops (S2.2)
Corrections (S2)
Requirements (S3)
Improvements (S4)
UX (S4.5)
Bugs over the CRUD operations (S2.1)
System (S)
Expose the AI to the engineer (S5.6)
Make functionalities available through API (S5.5)
Positive and negative feedbacks (S1.1)
Perceptions (S1)
Suggestions (S5)
Add more domains of application (S5.4)
The alignment between the predictions and the specialists expectation (S1.2)
The enhancement of the algorithm as the system is more utilized (S1.3)
New scenarios for simulations (S5.1)
Applications for predictions of environmental disasters (S5.2)
Other formats (eg. mobile) (S5.3)
is a

FIGURE 11 Feedbacks related to the category system (RQ 3)

Another expert suggested to make the functionalities of the system available through an API (S5.5), so that other applications can consume this intelligence. He also commented about expanding the solution to other formats, such as the mobile (S5.3). Lastly, one feedback discussed on exposing the machine learning algorithm to the public (S5.6), so that the developers could comprehend how this intelligence worked and propose improvements. We argue that it is already done, as the source code is available in the Appendix Section. Furthermore, the questionnaire and respective mischaracterized answers are available in this same place. The following excerpt exemplify these findings:
(S5.2) - [Participant 10]

"I am impressed with the system. It fulfills its purpose indeed. I even foresee that it has a lot of potential in helping predicting environmental disasters, something that has shown to be a trend."

In summary, the feedbacks from the experts bring us the conclusion that, yes, the system does fulfil its purpose. Despite the mixed reviews, they recognize the capability of the system in performing simplistic predictions, given the initial state in which the project is. Some bugs and requirements were reported as well as remarkable topics for improvement and suggestions for future applications.

4.5 | Threats to Validity and Limitations

Prototype. In our approach, the scope covers only Simple emergent behaviors. Another issue regards the size of the dataset, which is relatively small. In these cases, learning algorithms may present bias, thus negatively affecting the results, that is, the correctness is high during training phase but the algorithm is incapable of generalizing well in test cases. That means, for example, that the predictions might vary intensively from one execution of the algorithm to the other.

Regarding the MLP, as more SoS are added to the dataset and, hence, to the network, the latter increases in size and needs more parameters and samples of each data to work properly. In an optimal scenario, all works harmonically, but in real cases, reality is different, the network will achieve a better performance as more, representative and balanced samples are provided. This is a challenge that guide further strategies in the process and it is something that needs to be further explored in the context of this research.

Evaluation. We identified some limitations in regards to this evaluation. The first of them is the size of the population, as there were only eleven participants. This is a difficulty reported in other studies in software engineering[32], as there are not many SoS specialists in the world and this topic (SoS) is continuously maturing and rising. Therefore, it represents a threat to validity in the proposed method. Moreover, other threats regard the manual process of extraction of data the fed the dataset elaborated and the coding process, as it was performed by only one researcher. Thus, we encourage future studies to reproduce, extend and enhance the evaluation that was executed in this method.

We also recognize threats regarding a certain mismatch between the target audience and the profile of the participants of the evaluation. The former included SoS engineers and experts, but the evaluation was performed only by experts, not SoS engineers. This could be subject of future work, as to study the impacts on the evaluation process. Furthermore, all the evaluations were made in a design-time context. Thus, runtime scenarios could be prepared and assessed, possibly with adaptations. Finally, we mention the assessment of the algorithm that replaced the MLP. Although we evaluated some possible alternatives, this process could be further investigated, testing different hyperparameters, resampling the input data in different ways, and so on. Outliers could also be removed, but only when the dataset was big enough, which was not the case in this evaluation.

This section detailed the evaluation performed with experts. Many valuable and rich findings were obtained along the process. Although the proposed application is in its infancy and has a series of limitations, we were able to evaluate its effectiveness and state that it is a promising approach that has potential to evolve.

5 | FINAL REMARKS

This paper presented Macgyver Predictor, a framework composed of (i) a dataset of emergent behaviors, (ii) a machine learning (ML) mechanism to support prediction of emergent behaviors given a set of constituents, and (iii) a system to support the design of emergent behaviors in the context of SoS. The results communicated herein are the final product of a master's project.

We were capable of inferring possible emergent behaviors derived from different sets and combinations of constituents. This solution proved to be a promising technique towards aiding SoS engineers in a quick design or redesign of emergent behaviors in SoS and this approach has the potential to become enhanced as more data is added into the mechanism, improving the algorithm inside it.

We argue that both techniques have the potential to aid correctly inferring possible emergent behaviors in a given set of constituents, prevented some conditions are satisfied. This can be achieved with the development of a big dataset, with good quality in the existing data, representative and balanced samples and iterative refinement of the algorithm, for example.

The **contributions** are manifold and include:

- i the initial dataset containing emergent behaviors expressed in terms of the basic features of the constituents. Obtaining a dataset of such type was a very hard task during our proposed method. So hard that we decided to build our own. This contribution could be seen as an initial effort which could be used in other researches or could be enhanced by the community as to be used in other works, acting as a common repository for emergent behaviors in SoS;
- ii the algorithms for predicting the emergent behaviors (obtained from the research method). In a similar rationale as the with the proposed initial dataset, this machine learning solution could not only aid SoS engineers but could also be enhanced and took further by the community, as to extend this solution beyond, towards improving dealing with emergent behaviors in SoS;
- iii the system developed in the context of this research, which could be established as an initial step towards the automation of the bottom-up design of emergent behaviors in SoS; and
- iv the evaluation with experts, which brought valuable feedbacks that could aid the community in advancing the overall process of dealing with emergent behaviors in SoS.

Future work includes (i) increasing the dataset with more information about other SoS from other domains and emergent behaviors of other types (eg. *Weak*); (ii) evaluating different machine and deep learning mechanisms with the collected data, varying the hyperparameters as to obtain the most suitable algorithm in each case; (iii) implementing the reported corrections of bugs in the system; (iv) adding an AI algorithm could also be implemented for the *Top-down* approach in that sense; and (v) enhancing the evaluation by recruiting more SoS engineers and more experts to participate and making collaborative work with other researchers at the time of analysing the results.

REFERENCES

1. Neto VVG, Manzano W, Kassab M, Nakagawa EY. Model-based engineering & simulation of software-intensive systems-of-systems: experience report and lessons learned. In: *Proc. of the 12th ECSA (Companion)* ACM 2018; Madrid, Spain:27:1–27:7.
2. Neto VVG. *A simulation-driven model-based approach for designing softwareintensive systems-of-systems architectures. (Une approche dirigée par les simulations à base de modèles pour concevoir les architectures de systèmes-des-systèmes à logiciel prépondérant)*. PhD thesis. University of Southern Brittany, Vannes, Morbihan, France, 2018.
3. Maier MW. Architecting principles for systems-of-systems. *Systems Engineering*. 1998;1(4):267–284.
4. Graciano Neto VV, Kassab M. *What Every Engineer Should Know About Smart Cities*. Taylor & Francis, 2023.
5. Santos DS, Oliveira BRN, Duran A, Nakagawa EY. Reporting an Experience on the Establishment of a Quality Model for Systems-of-Systems. In: Xu H., ed. *The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015* KSI Research Inc. and Knowledge Systems Institute Graduate School 2015:304–309.
6. Bianchi T, Santos DS, Felizardo KR. Quality Attributes of Systems-of-Systems: A Systematic Literature Review. In: Avgeriou P, Cuesta CE, Drira K, et al., eds. *3rd IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems, SESoS 2015, Florence, Italy, May 17, 2015* IEEE Computer Society 2015:23–30.
7. Mittal S, Cane SA. Contextualizing Emergent Behavior in System of Systems Engineering using Gap Analysis. In: *Proc of MSCIAAS 2016 and SPACE 2016* SCS.
8. Manzano W, Neto VVG, Nakagawa EY. Dynamic-SoS: An Approach for the Simulation of Systems-of-Systems Dynamic Architectures. *Comput. J.*. 2020;63(5):709–731.
9. Silva K, Horita FEA, Neto VVG. Bring Us MacGyver Predictor: Towards a Deep Learning-Based Mechanism to Design Emergent Behaviors in Systems-of-Systems. In: Almeida Maia dM, Dorça FA, Araújo RD, Flach vC, Nakagawa EY, Canedo ED., eds. *Proc. of SBES 2022* ACM 2022:299–304.
10. Anderson R, Boulanger A, Prosser R. C2SOS: A Military Cyber-Secure & Interoperable System of Systems for the Smart Grid Situational Awareness, Modeling, and Simulation to Power Command and Control and Automated Contingency Management Boeing Columbia U and Con Edison. 2009.
11. Teixeira PG, Lebtag BGA, Santos dRP, et al. Constituent System Design: A Software Architecture Approach. In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)* IEEE 2020.
12. LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature*. 2015;521(7553):436–444.
13. Chauhan NK, Singh K. A Review on Conventional Machine Learning vs Deep Learning. In: *Proc. of GUCON* IEEE 2018.
14. Awad M, Khanna R. Support Vector Regression. In: *Efficient Learning Machines* , , Apress, 2015:67–80.
15. Raman R, Jeppu Y. Does The Complex SoS Have Negative Emergent Behavior? Looking For Violations Formally. In: *2021 IEEE International Systems Conference (SysCon)* IEEE 2021.
16. Gardner M, Dorling S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*. 1998;32(14-15):2627–2636.
17. Ruby U, Yendapalli V. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng.* 2020;9(10).
18. Yu L, Zhou R, Chen R, Lai KK. Missing Data Preprocessing in Credit Classification: One-Hot Encoding or Imputation?. *Emerging Markets Finance and Trade*. 2020;58(2):472–482.
19. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*. 2002;16:321–357.
20. Ma Y, He H. *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley IEEE Press, 2013.
21. Mohammed R, Rawashdeh J, Abdullah M. Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. In: *Proc. of 11th ICICS* 2020:243-248.
22. Faceli K, Lorena AC, Gama J, Almeida TAd, Carva ACPLFd. *Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina*. LTC, 2021.
23. Sende M, Schranz M, Prato G, Brosse E, Morando O, Umlauft M. Engineering Swarms of Cyber-Physical Systems with the CPSwarm Workbench. *Journal of Intelligent &amp Robotic Systems*. 2021;102(4).
24. Garcés L, Nakagawa EY. A process to establish, model and validate missions of systems-of-systems in reference architectures. In: *Proceedings of the Symposium on Applied Computing* ACM 2017.
25. Al-Amin M, Dagli CH. A Tool for Architecting Socio-Technical Problems: SoS Explorer. In: *Proc. of ISSE* 2019:1-7.
26. Alzahrani A, Shamsi P, Ferdowsi M, Dagli C. Modeling and simulation of a microgrid using feedforward neural networks. In: *2017 IEEE 6th (ICRERA)* IEEE 2017.
27. Guariniello C, Mockus L, Raz AK, DeLaurentis DA. Towards Intelligent Architecting of Aerospace System-of-Systems: Part II. In: *2020 IEEE Aerospace Conference* IEEE 2020.
28. Kasunic M. Designing an Effective Survey. Tech. Rep. CMU/SEI-2005-HB-004, Carnegie Mellon Software Engineering Institute; Pittsburg, USA: 2005.
29. Linåker J, Sulaman S, Host M, Mello dR. Guidelines for Conducting Surveys in Software Engineering. tech. rep., Lund University; Sweden: 2015.
30. Molléri JS, Petersen K, Mendes E. Survey Guidelines in Software Engineering: An Annotated Review. In: *10th ESEM* Association for Computing Machinery 2016; Ciudad Real, Spain.
31. Lebtag BGA, Teixeira PG, Santos dRP, Viana D, Neto VVG. Evaluating the Understandability and Expressiveness of Simulation Executable Models with Professionals: Obtaining perceptions from researchers and practitioners for improving quality of models. In: Viana D, Schots M., eds. *19th SBQS 2020, São Luís, Brazil, December, 2020* ACM 2020:12.
32. Lebtag BGA, Teixeira PG, Santos RPD, Viana D, Neto VVG. Strategies to Evolve ExM Notations Extracted from a Survey with Software Engineering Professionals Perspective. *JSERD*. 2022;10.
33. V.R. Basili GC, Rombach H. The goal question metric approach. *Encyclopedia of software engineering*. 1994:528–532.
34. Viana D. Análise Qualitativa com Grounded Theory em Sistemas de Informação. 2021.
35. Guessi M, Oquendo F, Nakagawa EY. Checking the architectural feasibility of Systems-of-Systems using formal descriptions. In: *Proc. of 11th SoSE* IEEE 2016.