ASP Chef for Water Waste Monitoring

Mario Alviano, Luis Angel Rodriguez Reiners*

DeMaCS, University of Calabria, 87036 Rende (CS), Italy

Abstract

Water quality monitoring is a critical task in environmental protection and climate change adaptation. In this paper, we present the use of ASP Chef, a lightweight web-based environment for exploring and transforming ASP answer sets, in the context of the Tech4You project. ASP Chef enables intuitive pipelines over arrays of interpretations, supporting data-driven analysis without requiring low-level programming or complex tooling. Inspired by CyberChef, ASP Chef has recently been extended with a novel mechanism for content generation based on Mustache templates. This feature allows answer sets to be transformed into JSON and other structured formats, enabling seamless integration with JavaScript-based visualization frameworks such as @vis.js/Network, Tabulator, and ApexCharts. We demonstrate how ASP Chef is used to visualize and analyze water quality data collected by multisensory buoys, which monitor a wide range of chemical and physical parameters. Raw data is preprocessed using Python libraries such as NumPy, Pandas, and TensorFlow for cleaning and neural network-based modeling. Selected portions of the cleaned data are then explored via ASP Chef recipes, launched directly through dumbo-asp, a tool that opens the browser and executes the recipe with the given input. Our results show that ASP Chef enables effective visual exploration of parameter trends, detection of critical values through logic queries, and the creation of interactive dashboards for domain experts. This work illustrates how declarative logic programming can be combined with modern front-end technologies to build practical tools for environmental monitoring and decision support.

Keywords

Answer Set Programming, ASP Chef, visualization, data analysis

1. Introduction

Answer Set Programming (ASP) is a declarative programming paradigm rooted in nonmonotonic logic and stable model semantics [1, 2, 3, 4, 5], and it has been successfully applied in a wide range of knowledge-intensive domains [6, 7, 8, 9]. ASP is particularly suitable for problems involving combinatorial search, reasoning with incomplete or uncertain information, and constraint satisfaction. Despite its strong theoretical foundations and expressiveness, the practical integration of ASP into real-world applications has traditionally posed nontrivial challenges, especially for those applications requiring interaction with modern web technologies or rich data visualizations [10, 11, 12, 13, 14, 15]. To address this gap, ASP Chef [16] has emerged as a lightweight framework designed to facilitate the generation of structured output from ASP computations. ASP Chef offers a unique approach to problem-solving through the concept of ASP recipes. These recipes consist of chains of ingredients or operations combining computational tasks typical of ASP addressed by the CLINGO ASP solver [17] with other operations like data manipulation and visualization.

Originally inspired by CyberChef (https://gchq.github.io/CyberChef/), ASP Chef is a simple and intuitive web application designed to facilitate the analysis and manipulation of answer sets produced by ASP programs [18]. Unlike traditional ASP development environments, ASP Chef does not aim to be an IDE or code editor. Instead, it focuses on providing a user-friendly interface for building pipelines of operations over collections of answer sets, enabling users to inspect, transform, filter, and visualize interpretations without the need to integrate complex external tools or write auxiliary code in general-purpose programming languages. Recently, ASP Chef has been extended with support

CILC 2025: 40th Italian Conference on Computational Logic, June 25–27, 2025, Alghero, Italy

*Corresponding author.

mario.alviano@unical.it (M. Alviano); luis.reiners@unical.it (L. A. Rodriguez Reiners)

https://alviano.net/ (M. Alviano)

0000-0002-2052-2063 (M. Alviano); 0009-0000-1808-9910 (L. A. Rodriguez Reiners)

for template-driven content generation [19], allowing users to define output formats using Mustache templates (a popular web templating systems; [20]). These templates are dynamically expanded using the atoms projected via `#show` directives, enabling answer sets to be transformed into structured data formats such as JSON, CSV, or Markdown. This makes it particularly effective for integrating ASP reasoning with modern web-based visualization libraries, without requiring users to write glue code or deal with low-level data wrangling. Among the recently integrated front-end frameworks there are the followings:

- *@vis.js/Network* (https://visjs.org/) for visualizing graphs and network structures, useful for understanding relationships and dependencies;
- *Tabulator* (https://tabulator.info/), an interactive table library that supports sorting, filtering, pagination, and inline editing—ideal for inspecting structured datasets;
- *ApexCharts* (https://apexcharts.com/), a comprehensive charting library capable of producing time series graphs, scatter plots, radar charts, box plots, heatmaps, and more.

In this paper, we illustrate the application of ASP Chef within the context of the Tech4You project (Technologies for climate change adaptation and quality of life improvement; https://iia.cnr.it/project/tech4you/), where one of the goals is to develop intelligent tools for analyzing environmental data, particularly focusing on water quality monitoring. As climate change and pollution increasingly affect natural water sources, real-time monitoring and intelligent analysis of water parameters have become essential tools for environmental protection, public health, and sustainable resource management.

The data considered in our study was collected using multisensory buoys, which continuously monitor a diverse set of chemical and physical water quality parameters. These include:

- chemical indicators such as ammonia (NH_3), nitrate ions (NO_3^-), ammonium ions (NH_4^+), and fluorescent dissolved organic matter (fDOM),
- physical and electrochemical properties such as specific conductance (SpCond), dissolved oxygen (DO), redox potential (ORP), salinity (Sal), total dissolved solids (TDS), pH, temperature (T), and turbidity.

In addition to the in-water measurements, meteorological and hydrological observations were recorded over the same time period, including rainfall, atmospheric pressure, hydrometric level, and humidity. This multi-source dataset allows for rich, multi-dimensional analysis, uncovering relationships between environmental conditions and water quality metrics.

The complete sensor and meteorological dataset collected by the multisensory buoys, comprising both water quality and meteorological parameters, is initially processed using standard Python data science libraries, including NumPy and Pandas for data wrangling, cleaning, and transformation. These tools are employed to handle missing values, normalize sensor readings, and align time series across multiple sources. For predictive analysis and pattern recognition, selected features are fed into neural network models implemented with TensorFlow, allowing the identification of anomalies and trends through supervised and unsupervised learning techniques. Once this initial processing is complete, selections of the cleaned and structured data are passed to ASP workflows using ASP Chef recipes. This integration is made seamless through the use of dumbo-asp (https://github.com/alviano/dumbo-asp), a lightweight Python module that provides a convenient interface to launch the ASP Chef environment. With a single command, dumbo-asp opens the browser and loads a specified recipe along with the relevant input data, enabling users to interactively explore results, apply logic-based filters, and visualize findings without manual setup or scripting.

Using ASP Chef, we were able to construct a suite of interactive visualizations that aid experts in interpreting the data, identifying anomalies, and formulating hypotheses. ASP logic is used not only to preprocess and filter the data, but also to declaratively express conditions of interest, such as threshold exceedance, joint parameter deviations, or domain-specific constraints. These logic-based selections are then rendered visually via the integrated frameworks. Among the visualizations, here we present the followings:

- Time series charts that compare the evolution of multiple parameters over time, helping identify trends and correlations;
- Scatter plots with customizable axes to explore relationships between any pair of parameters, including meteorological vs. chemical indicators;
- Box plots and radar charts that summarize the distribution and spread of parameters, highlighting outliers or unusually high variability;
- Network diagrams (via @vis.js/Network) used to visualize correlation graphs or logical dependencies encoded in ASP;
- Interactive tables (via Tabulator) for sorting, filtering, and browsing subsets of the dataset in a user-friendly format.

A particularly notable feature of ASP Chef is that the logic and the presentation remain decoupled but aligned: the reasoning layer determines what information should be displayed, and the templating system handles how it is rendered. This modularity encourages the reuse of logic components across different visualizations, supports rapid prototyping, and empowers domain experts to explore the data through different lenses without changing the underlying data processing pipeline.

In summary, ASP Chef provides a novel and practical bridge between declarative logic programming and web-based data visualization, offering a powerful toolset for environmental monitoring and decision support. The case study in this paper demonstrates its effectiveness in the water quality domain, but its architecture is general and can be readily applied to other fields where structured reasoning and interactive exploration of complex datasets are needed.

2. Background

This section provides an overview of essential Python libraries and methodologies commonly employed in data science, machine learning, and artificial intelligence, along with Answer Set Programming (ASP).

2.1. Time-Series Forecasting

Let us fix a set $P = \{p_1, \ldots, p_m\}$ of parameters sampled at regular intervals ΔT for n time steps. The resulting n-length time-series associated with parameter p_i $(1 \leq i \leq m)$ is denoted

$$S_i = (\langle p_{i,1}, T_1\rangle, \ldots, \langle p_{i,n}, T_n\rangle)\,, \text{ with } \Delta T = T_j - T_{j-1} \text{ for every } 1 < j \leq n, \tag{1}$$

where $p_{i,j}$ is the value of parameter p_i at observation time T_j $(1 \leq j \leq n)$.

In time-series forecasting, the goal is to predict the future value(s) of a variable based on its past observations and, optionally, the historical data from other related variables. Formally, let p_i be the variable we want to predict. The prediction at the next time step $n + 1$ can be defined as

$$\hat{p}_{i,n+1} = f(p_{i,1}, \ldots, p_{i,n},\ p_{i_1,1}, \ldots, p_{i_1,n},\ \ldots,\ p_{i_k,1}, p_{i_k,n}) \tag{2}$$

where $\hat{p}_{i,n+1}$ is the predicted value of the target parameter p_i at time step $n+1$; $p_{i,j}, \ldots, p_{i,n}$ are the past n observations of the target parameter p_i; the remaining arguments are the past n observations of the k auxiliary variables $\{p_{i_1}, \ldots, p_{i_k}\} \subset P$. This formulation applies to both univariate and multivariate time series forecasting: in the univariate case, only the history of the target variable is used; in the multivariate case, the model also uses data from other related variables.

Long Short-Term Memory (LSTM) networks [21] are a specialized type of recurrent neural network (RNN) designed to handle sequential data and capture long-term dependencies. They achieve this through the use of memory cells and gating mechanisms; specifically, the input gate, output gate, and forget gate (as shown in Figure 1). These components allow LSTMs to selectively retain or discard information over time, making them well-suited for tasks that involve patterns spread across long sequences. Thanks to these capabilities, LSTM networks are widely used in time-series forecasting applications such as speech recognition, language modeling, and environmental data analysis.

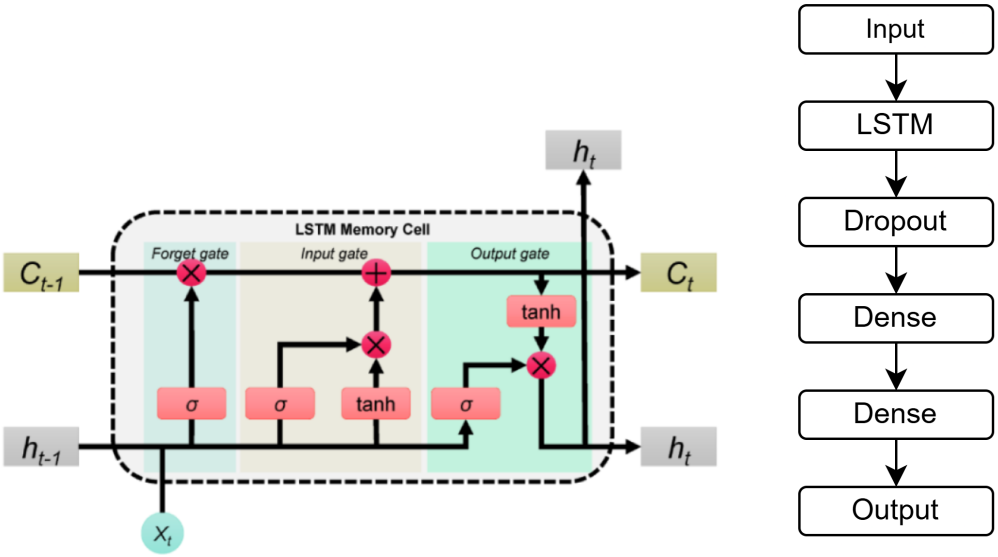


Figure 1: (Left) LSTM cell structure depicting the internal components, including input, forget, and output gates. (Right) Model architecture consisting of an Input layer, LSTM layer, Dropout layer for regularization, followed by two Dense layers for prediction.

2.2. Python Libraries

Pandas (https://pandas.pydata.org/) is a Python library for data manipulation and analysis. Its core data structures, *DataFrame* and *Series*, provide efficient handling of tabular and labeled one-dimensional data, respectively. Pandas supports powerful indexing, reshaping, and aggregation operations, and integrates with MATPLOTLIB for data visualization [22]. TensorFlow (https://www.tensorflow.org/) is an open-source framework for developing and deploying machine learning models. It abstracts low-level computation through high-level APIs and represents data as multidimensional arrays called *tensors*. TensorFlow supports training and inference on various hardware accelerators, making it suitable for both research and production [23].

2.3. Answer Set Programming

An ASP program is a finite set of rules. Each rule typically has a head, representing a conclusion (which may be atomic or a choice), and a body, representing a set of conditions that must hold (a conjunction of literals, aggregates and inequalities). Formally, an ASP program Π induces a collection (zero or more) of answer sets (also known as stable models), which are interpretations that satisfy all the rules in Π while also fulfilling the stability condition (i.e., the models must be supported and minimal in a specific formal sense [24]). The intended output of a program can be specified using `#show` directives of the form

`#show` $p(\bar{t})$ `:` *conjunctive_query* `.`

Here, p denotes an optional predicate symbol, $\bar{t}$ is a (possibly empty) sequence of terms, and *conjunctive_query* is a conjunction of literals serving as a condition for displaying instances of $p(\bar{t})$. Answer sets are then projected accordingly. For a detailed specification of syntax and semantics, including `#show` and other directives, we refer to the ASP-Core-2 standard format [25].

Example 1. Consider a scenario where we want to choose a subset of water monitoring stations that jointly cover a required set of chemical parameters. The following ASP program models this selection problem:

```
r1 :  1 <= {selected(S) : station(S)} <= N :- limit(N).
r2 :  :- required(P), #count{S : selected(S), covers(S,P)} = 0.
r3 :  #show S : selected(S).
```

Rule r_1 is a *choice rule* that selects between 1 and N stations (as determined by the fact `limit(N)`). Rule r_2 is a *constraint* that enforces full coverage: every required parameter P must be covered by at least one selected station. The `#show` directive in r_3 ensures that only the selected stations appear in the output (projecting the answer sets accordingly). Suppose the following input facts define the stations, the parameters they monitor, and the desired coverage:

```
parameter(ph).  parameter(nitrate).   parameter(lead).     parameter(arsenic).
required (ph).  required (nitrate).   required (lead).     limit(2).
station(s1).    covers(s1, ph).       covers(s1, nitrate).
station(s2).    covers(s2, lead).     covers(s2, arsenic).
station(s3).    covers(s3, nitrate).  covers(s3, lead).
```

The program has the projected answer set `s1 s2`, indicating that selecting stations `s1` and `s2` ensures full coverage of all required water quality parameters using at most two stations. ■

2.4. ASP Chef

An *operation* O is a function receiving in input a sequence of interpretations and producing in output a sequence of interpretations. Operations may produce side outputs (e.g., a graph visualization) and accept parameters to influence their behavior. An *ingredient* is an instantiation of a parameterized operation with side output. A *recipe* is a tuple of the form $(encode, Ingredients, decode)$, where $Ingredients$ is a (finite) sequence $O_1\langle P_1\rangle, \ldots, O_n\langle P_n\rangle$ of ingredients, and $encode$ and $decode$ are Boolean values. If $encode$ is true, the input of the recipe is mapped to **[**[`__base64__("`s`")`]**]**, where $s = Base64(s_{in})$ (i.e., the Base64–encoding of the input string s_{in}). After that, the ingredients are applied one after another. Finally, if $decode$ is true, every occurrence of `__base64__(`s`)` is replaced with (the ASCII string associated with) $Base64^{-1}(s)$. Among the operations supported by ASP Chef there are *Encode*$\langle p, s\rangle$ to extend every interpretation in input with the atom p`("`t`")`, where $t = Base64(s)$; *Search Models*$\langle \Pi, n\rangle$ to replace every interpretation I in input with up to n answer sets of $\Pi \cup \{p(\bar{t}).\mid p(\bar{t}) \in I\}$; *Show*$\langle \Pi\rangle$ to replace every interpretation I in input with the projected answer set $\Pi \cup \{p(\bar{t}).\mid p(\bar{t}) \in I\}$ (where Π comprises only `#show` directives.

Example 2. The problem from Example 1 can be addressed in ASP Chef by a recipe comprising a single *Search Models*$\langle\{r_1, r_2, r_3\}, 1\rangle$. Alternatively, a recipe separating computational and presentational aspects would comprise two ingredients, namely *Search Models*$\langle\{r_1, r_2\}, 1\rangle$ and *Show*$\langle\{r_3\}\rangle$. ■

Several operations in ASP Chef support expansion of *Mustache templates* [19]; among them, there are *Expand Mustache Queries*, *@vis.js/Network* (to visualize graphs), *Tabulator* (to arrange data in interactive tables), and *ApexCharts* (to produce different kinds of charts). A Mustache template comprises queries of the form `{{ `Π` }}`, where Π is an ASP program with `#show` directives—alternatively, `{{= `$p(\bar{t})$` : `*conjunctive_query*` }}` for `{{ #show `$p(\bar{t})$` : `*conjunctive_query*`. }}`. Intuitively, queries are expanded using one projected answer set of $\Pi \cup \{p(\bar{t}).\mid p(\bar{t}) \in I\}$, where I is the interpretation on which the template is applied on. Separators can be specified using the predicates `separator/1` (for tuples of terms), and `term_separator/1` (for terms within a tuple). The varadic predicate `show/*` extends a shown tuple of terms (its first argument) with additional arguments that enable repeating tuples in output and can be used as sorting keys (using predicate `sort/1`). Moreover, Mustache queries can use `@string_format(format, ...)` to format a string using the given format string and arguments, and floating-point numbers are supported with the format `real("NUMBER")`. Format strings can also be written as (multiline) *f-strings* of the form `{{f"..."}}`, using data interpolation `${`*expression*`:`*format*`}` to render *expression* according to the given *format*.

Example 3. Recipes from Example 2 can be further extended by including ingredients to visualize input and computed solution graphically. To this aim, the following Mustache template can be combined with *@vis.js/Network* to obtain the graph shown in Figure 2:

```
{ data: {
    nodes: [
```

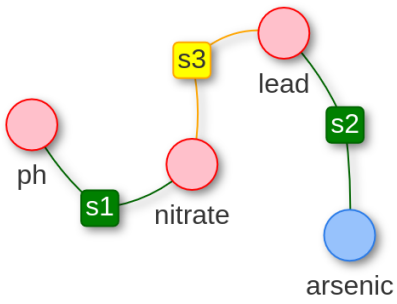


Figure 2: Graphical representation of input and selected stations for the problem presented in Example 1

```
      {{= {{f"{ id: "${S}", label: "${S}",
                group: "selected_station" }"}} : station(S), selected(S) }}
      {{= {{f"{ id: "${S}", label: "${S}",
                group: "other_station" }"}} : station(S), not selected(S) }}
      {{= {{f"{ id: "${P}", label: "${P}",
                group: "required_parameter" }"}} : parameter(P), required(P) }}
      {{= {{f"{ id: "${P}", label: "${P}",
                group: "other_parameter" }"}} : parameter(P), not required(P) }} ],
    edges: [ {{= {{f"{ from: "${S}", to: "${P}" }"}} : covers(S,P) }} ],  },
  options: {
    nodes: {shape: "dot", size:30, font:{size:32}, borderWidth:2, shadow: true },
    edges: { width: 2, shadow: true },
    groups: {
      selected_station: { shape: "box", font: { color: "white" },
                          color: { background: "green", border: "darkgreen" } },
      other_station: { shape:"box", color:{background:"yellow", border:"orange"} },
      required_parameter: { color: { background: "pink", border: "red" } },
    },
  },
}
```

Mustache queries define nodes and links in the graph starting from facts in the computed answer set. A recipe addressing the selection problem and producing the visualization shown in Figure 2 is available at https://asp-chef.alviano.net/s/CILC2025/station-selection. ■

3. The Fitterizzi Dataset

Several physicochemical parameters were recorded continuously by a multiparametric probe located in Fitterizzi (in the province of Cosenza, Southern Italy), among them ammonia (NH_3), specific conductance (SpCond), fluorescent dissolved organic matter ($fDOM$), nitrate ions (NO_{3-}), dissolved oxygen (DO), ammonium ions (NH_{4+}), redox potential (ORP), salinity (Sal), total dissolved solids (TDS), pH, temperature (T), and turbidity. Moreover, meteo-hydrological observations were available for the same period of time, such as rainfall, atmospheric pressure, hydrometric level and humidity.

Given their nature, water quality and meteorological data should be represented as time-series, however the provided dataset consists of two files (not necessarily aligned): a CSV file containing data from the sensors; an Excel (XLSX) file containing meteorological and hydrological data. In fact, once we loaded them into Pandas dataframes using the functions `read_csv()` and `read_excel()`, the summary statistics reported in Tables 1–2 and obtained with the `describe()` method revel that not all sensor variables have the same number of observations. As shown by the plots in Figure 3, the dataset misses several values. We also observed that some parameters exhibit extreme or unexpected negative values, indicating calibration problems or malfunctions within the instruments (e.g., NO_{3-} and $fDOM$). Moreover, the time periods covered by the two datasets do not match perfectly, and the

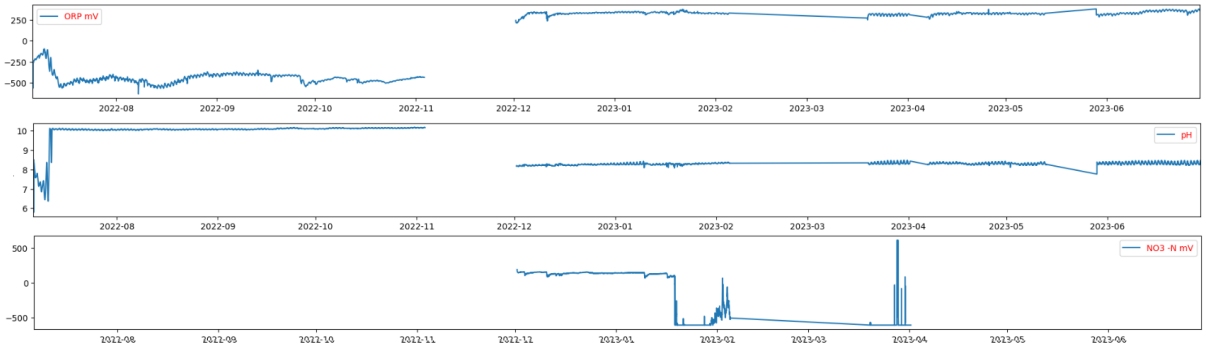


Figure 3: Trend of some variables over time, with missing values represented as gaps in the data

`infer_freq()` function reveals that while weather data are collected regularly at hourly frequency ($\Delta T = 1h$), sensor data has irregular intervals (no fixed ΔT), meaning that the time between readings is not consistent (i.e., the time-series do not conform to the format of equation 1). This is a problem for time-series analysis, which usually assumes that data points are collected at regular intervals. To fix this, the sensor data is resampled to a fixed 1-hour frequency using the `resample()` method. After that the datasets are filtered to include only the overlapping time period, and merged together. Finally, missing values are handled to ensure the final dataset is clean and usable.

Given the presence of sensors failures and missing values for some variables, we employed time-series forecasting to predict the missing values and possibly correct extreme and unexpected values, so that the data remains complete and reliable. We therefore use the readings from other sensors that are still working to predict missing values. Here we report the case of the target variable pH, for which we implemented a prediction model using a LSTM network and the architecture shown in Figure 1. In our model, the LSTM layer captures long-term dependencies and patterns in our sequential data. A dropout

Table 1
Summary of descriptive statistics for sensors parameters.

Parameter	count	mean	std	min	25%	50%	75%	max
NH_3 mg/L	21100	0.009	0.005	0	0.01	0.01	0.01	0.03
Cond µS/cm	41285	412.28	50.90	3.90	369.20	418.90	453.70	530.20
fDOM QSU	41285	20.25	8.55	-6.83	14.31	19.52	23.98	117.86
NH_{4+} -N mV	21100	-57.98	9.95	-74.90	-66.60	-57.20	-52.20	-23.20
NO_{3-} -N mV	11277	-127.08	348.83	-607.70	-607.70	127.70	140.60	607.80
ORP mV	38395	-18.35	389.66	-636.0	-447.10	310.10	333.70	383.10
Sal psu	41285	0.24	0.03	0.0	0.23	0.25	0.26	0.33
TDS mg/L	41285	330.03	39.70	2.0	312.0	332.0	353.0	442.0
Turbidity FNU	41285	55.36	308.68	-0.20	2.95	5.25	10.70	6176.52
TSS mg/L	41285	0	0	0	0	0	0	0
pH	38395	9.04	0.91	5.80	8.28	8.37	10.07	10.17
Temp °C	41285	15.37	4.54	5.95	11.68	14.58	19.02	26.79

Table 2
Summary of descriptive statistics for meteo-hydrological parameters.

Parameter	count	mean	std	min	25%	50%	75%	max
Rain mm	8736	0.14	0	0	0	0	31.0	0.85
Water Level m	8736	0.39	0.26	0.34	0.38	0.42	1.06	0.07
Pressure hPa	8736	995.33	976.2	992.1	995.33	998.4	1013.7	6.17
Solar Radiation W/m^2	8736	-188.78	0	0	0	323	1117	284.48
Humidity %	8736	73.38	14	56.0	75.0	96.0	100.0	22.07
Wind Speed m/s	8736	3.46	0	1.6	2.3	4.4	22.7	2.85

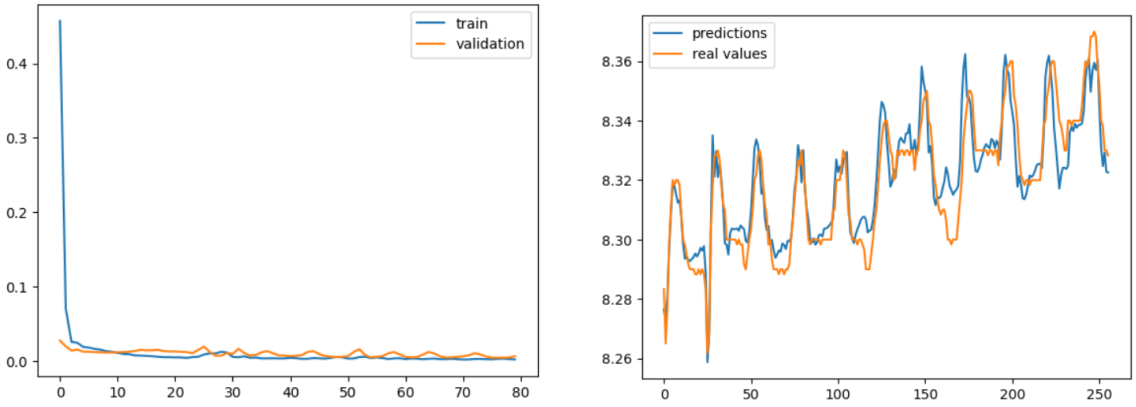


Figure 4: The training and validation loss curves (MSE) during model training (left), and the performance of the model (RMSE) on the test dataset (predicted vs. actual values; right).

layer is inserted after the LSTM layers to prevent overfitting by randomly dropping a fraction of the connections during training, which encourages the model to improve generalization. Finally, two dense layers are used to produce the final prediction. We used the following hyperparameters: 64 neurons for the LSTM layer; dropout rate of 0.5; 10 neurons for the first Dense layer.

The definition, training, and evaluation of the model were all implemented using TensorFlow. In training the LSTM model, we set the number of epochs to 80, with the objective of predicting the next pH value based on the observations from the previous 12 hours. The Mean Square Error (MSE) was selected as the loss function. For model optimization, we employed Adaptive Moment Estimation (Adam) with a learning rate of 0.01, a well established algorithm that has proven effective in practical applications. Figure 4 reports MSE values across each epoch iteration, and the performance of the trained model on the test dataset (RMSE). We observe that RMSE achieved a value of 0.014.

4. Filtering and Visualizing Data with ASP Chef

This section presents how we performed further data analysis using ASP Chef. In particular, we showcase how the Fitterizzi dataset is loaded in ASP Chef and a few recipes to obtain interactive visualization. An interactive recipe is available at https://asp-chef.alviano.net/s/CILC2025/fitterizzi.

4.1. Data Loading

In ASP Chef, CSV data files can be easily loaded using the ingredient *Parse CSV*. The result is a set of `__cell__` predicate instances, which in our case represent parameter values. After that, the parameters can be grouped by data collection time using a *Search Models* ingredient.

Example 4. In our case, for every data collection time (DT), collected parameters are conductivity (C), fDOM (FD), oxidation-reduction potential (ORP), salinity (SAL), total dissolved solids (TDS), turbidity (TUR), pH (PH) and temperature (TEMP). Once these parameters are represented as instances of `__cell__/3`, the grouping by DT is achieved by *Search Models* using the following program:

```
row(DT,C,FD,ORP,SAL,TDS,TUR,PH,TEMP) :- __cell__(R,2,DT),  __cell__(R,3,C),
  __cell__(R,4,FD),  __cell__(R,5,ORP), __cell__(R,6,SAL), __cell__(R,7,TDS),
  __cell__(R,8,TUR), __cell__(R,9,PH),  __cell__(R,10,TEMP), R > 1.
```

■

After loading the data, we can use the *Tabulator* operation to represent the dataset in a table, where each row corresponds to a specific time point or observation, providing a clear and organized view of the information for further analysis.

Example 5. Continuing with the set of facts representing the raw data obtained in Example 4, we create a table representation of the dataset. Using the *Tabulator* ingredient we encode a configuration

Datetime	Cond	fDOM Q...	ORP mV	Sal psu	TDS mg/L	Turbidity	pH	Temp °C
2023-01-17 00:10:00...	377.5	20.3	333.3	0.25	341	556.91	8.29	10.369
2023-01-17 00:20:00...	369	26.39	333.1	0.25	333	398.02	8.29	10.335
2023-01-17 00:30:00...	371	25.89	332.1	0.25	335	398.5	8.3	10.345
2023-01-17 00:40:00...	379.3	19.87	332.3	0.26	342	549.96	8.3	10.366
2023-01-17 00:50:00...	384.8	13.14	333	0.26	347	885.06	8.29	10.389
2023-01-17 01:00:00...	381.5	14.74	333.8	0.26	344	802.13	8.29	10.402
2023-01-17 01:10:00...	379.9	17.15	334	0.26	342	649.36	8.29	10.398
2023-01-17 01:20:00...	377.4	16.97	334.2	0.25	340	682.28	8.29	10.389
2023-01-17 01:30:00...	375.4	18.5	334.6	0.25	338	572.62	8.28	10.397

Figure 5: Table representation with *Tabulator*

JSON object within a `__tab__` predicate. This configuration specifies the input data and presentation options. The following Mustache template renders the table shown in Figure 5:

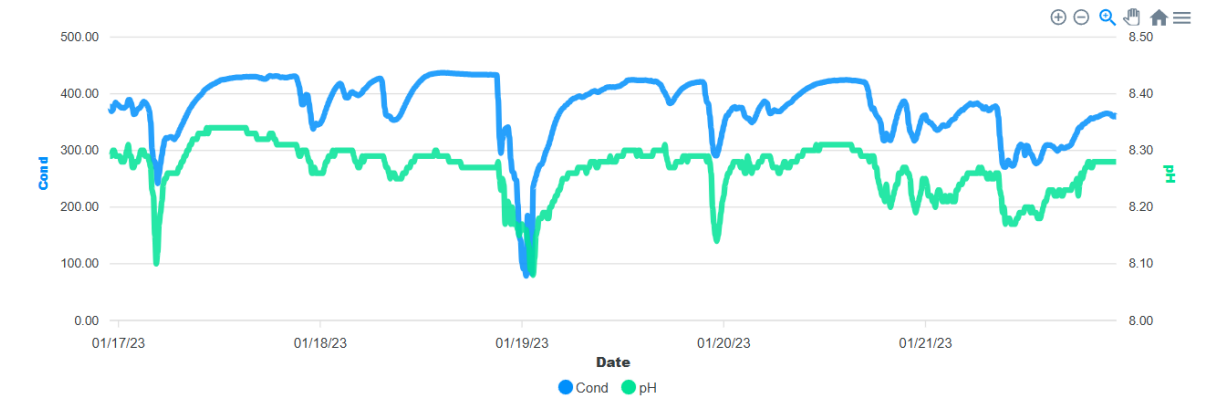
```
{  data: [{{= {{f"{ DateTime: "${DT}",
                    Cond:     ${C},
                    fDOM_QSU: ${FD},
                    ORP:      ${ORP},
                    Sal_psu:  ${SAL},
                    TDS:      ${TDS},
                    Turbidity: ${TUR},
                    pH:       ${PH},
                    Temp:     ${TEMP},
    }"}} : row(DT,C,FD,ORP,SAL,TDS,TUR,PH,TEMP) }} ],
  layout: "fitColumns",
  pagination: "local",
  paginationSize: 10,
  columns: [ {title: "Datetime",  field: "DateTime" },
             {title: "Cond",      field: "Cond"     },
             {title: "fDOM QSU",  field: "fDOM_QSU" },
             {title: "ORP mV",    field: "ORP"      },
             {title: "Sal psu",   field: "Sal_psu"  },
             {title: "TDS mg/L",  field: "TDS"      },
             {title: "Turbidity", field: "Turbidity"},
             {title: "pH",        field: "pH"       },
             {title: "Temp °C",   field: "Temp"     } ]
}
```

It is worth noting how the data field is obtained using a Mustache query. When evaluated inside *Tabulator*, the Mustache query expands to produce the following result:

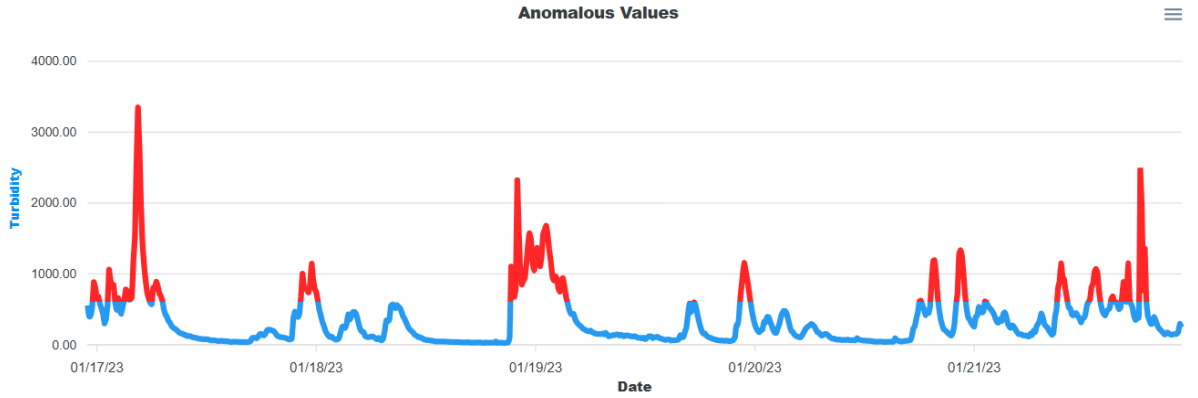
```
[ { DateTime: "2023-01-17 00:10:00+01:00", Cond: 377.5, fDOM_QSU: 20.3, ORP:
    333.3, Sal_psu: 0.25, TDS: 341, Turbidity: 556.91, pH: 8.29, Temp: 10.369 }
  { DateTime: "2023-01-17 00:20:00+01:00",  Cond: 369, fDOM_QSU: 26.39, ORP:
    333.1, Sal_psu: 0.25, TDS: 333, Turbidity: 398.02, pH: 8.29, Temp: 10.335 }
  { DateTime: "2023-01-17 00:30:00+01:00", Cond: 371, fDOM_QSU: 25.89, ORP: 332.1,
    Sal_psu: 0.25, TDS: 335, Turbidity: 398.5, pH: 8.3, Temp: 10.345 } ... ]
```

4.2. Exploratory Data Analysis

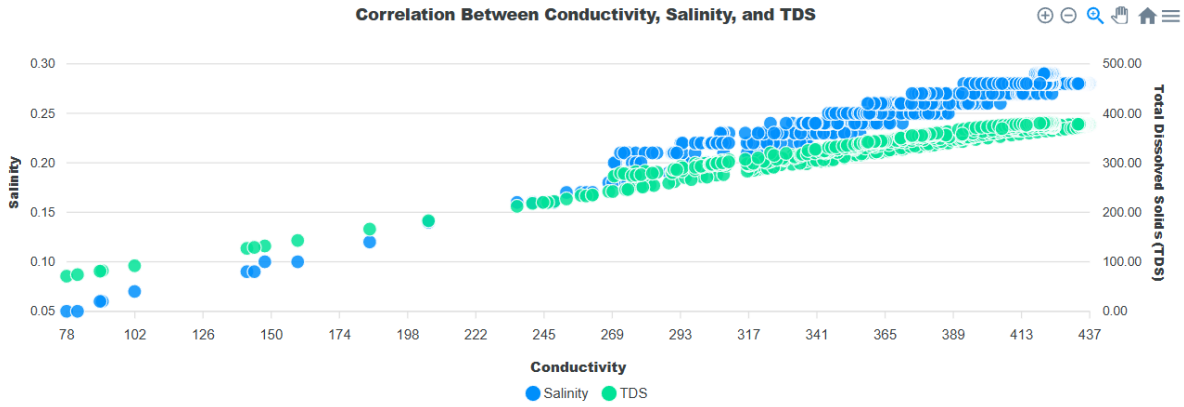
Raw data, in isolation, offers limited insight, and the sheer volume can be overwhelming to process. Visual description of the data or turning information into graphs, charts, and other visuals helps people understand the meaning of the values in the data set, revealing trends and patterns. We took advantage of *ApexChart* to obtain several interactive charts, and used *Tabulator* to compare the quartile and mean



(a) Time Series of Water Quality Parameters (Conductivity and pH).



(b) Anomalous Values in Turbidity Measurements Over Time.



(c) Scatter Plot of Conductivity, Salinity, and Total Dissolved Solids (TDS).

Figure 6: Exploratory analysis of water quality metrics: temporal trends, anomalies, and correlations. Plot (a) highlights a common pattern among conductivity and pH over time. Plot (b) signals turbidity values above the attention threshold defined by the domain experts. Plot (c) correlates measured values of conductivity, salinity and total dissolved solids, regardless of the time of relevation.

values collected in Fitterizzi with those originated from a different location.

Example 6. We used the data from Example 4 to obtain the charts reported in Figure 6 and investigate water quality metrics. Figure 6a focuses on selected variables (conductivity and pH levels) to narrow the focus of statistical analysis, and is obtained using the following Mustache template for the *ApexChart* ingredient:

```
{ series: [{ name: 'Cond',
             data: [{{= show({{f"${C}"}}, DT) : row(DT,C,_,_,_,_,_,PH,_) }}] },
```

	Place	Cond	fDOM Q...	ORP mV	Sal psu	TDS mg...	Turbidity	pH	Temp °C
min (2 items)									
min	San Nicola	225.2	-1.44	279.9	0.16	220	21.09	7.66	6.38
min	Fitterizzi	78.8	0.9	313	0.05	71	26.94	8.08	5.96
25% (2 items)									
25%	San Nicola	242.65	-1.21	302.8	0.17	227	64.57	7.68	8.63
25%	Fitterizzi	350.75	15.34	329.4	0.24	323	89.08	8.25	8.99
50% (2 items)									
50%	San Nicola	261.2	-1.1	326.7	0.17	234	66.07	7.83	10.28
50%	Fitterizzi	382.9	27.06	341.3	0.26	350	200.91	8.28	10.51
75% (2 items)									
75%	San Nicola	345.9	9.25	341.95	0.23	306	90.11	8.39	10.95
75%	Fitterizzi	420.1	35.61	354.1	0.28	373	494.21	8.29	11.28
max (2 items)									
max	San Nicola	364.7	14.49	372	0.25	331	4406.08	8.43	11.84
max	Fitterizzi	437	45.97	377.8	0.29	382	3348.47	8.34	12.28

(a) Comparison of quartile values for measures from two locations.

Place	Week Mean Values							
	Cond	fDOM QSU	ORP mV	Sal psu	TDS mg/L	Turbidity	pH	Temp °C
Fitterizzi	374.93	25.69	342.45	0.25	339.64	354.52	8.27	10.15
San Nicola	284.79	3.16	324.38	0.19	259.58	226.43	8.01	9.84

(b) Comparison of weekly mean values between two locations.

Figure 7: Exploratory Analysis of Water Quality Metrics: Temporal Trends, Anomalies, and Correlations.

```
          { name: 'pH',
            data: [{{= show({{f"${PH}"}}, DT) : row(DT,_,_,_,_,_,_,PH,_) }}] } ],
  xaxis: { type: 'datetime',
           categories: [{{= {{f""${DT}""}} : row(DT,_,_,_,_,_,_,_,_) }}],
           title: { text: 'Date' },  labels: { format: 'MM/d/yy' } },
  yaxis: [ { title: { text: "Cond", } },
           { opposite: true, title: { text: "pH", } } ],
}
```

The chart is useful to spot trends and variations over specific dates (in our case, November 2023). Figures 6b and 6c are also created using Mustache templates and *ApexChart*: Figure 6b identifies anomalous turbidity values and time-delayed measurements, suggesting potential data collection issues or environmental fluctuations; Figure 6c illustrates the correlation between conductivity, salinity, and TDS, revealing a linear relationship that aligns with expected physicochemical behavior in aquatic systems. Together, these visualizations provide foundational insights into data quality, temporal patterns, and parameter interdependencies, guiding further statistical or domain-specific analysis. ■

We compared the Fitterizzi dataset with similar data obtained in a different location in Cosenza, namely San Nicola Arcella, with the aim of identifying common trends and patterns. Such common trends are useful for detecting deviations from expected values, so that authorities can be informed and take action. Here we report a comparison of aggregated data using groups in *Tabulator*, and specialized charts in *ApexChart*. Interactive recipes are available at https://asp-chef.alviano.net/s/CILC2025/fitterizzi-vs-sannicola-1 and https://asp-chef.alviano.net/s/CILC2025/fitterizzi-vs-sannicola-2.

Example 7. Let us consider the mean and quartile values for the data of the two locations to compare. Aggregated data can be loaded as presented in Example 4, and *Tabulator* can be configured to obtain the tabular representation shown in Figure 7. With respect to the table presented in Example 5, the main difference is how values are grouped by place. We used the following Mustache template:

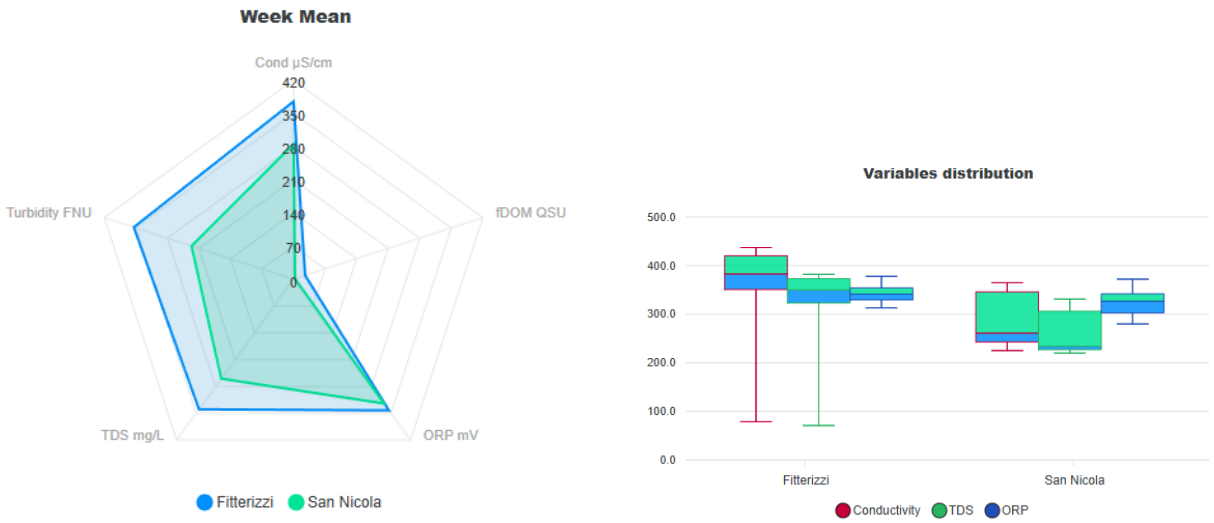


Figure 8: Comparison of mean weekly values and distribution: A Radar Chart (Spider Graph) for mean values across two locations and a Boxplot for distribution of selected variables. In the Radar Chart, we observe higher mean values for Fitterizzi, while the Boxplot reveals extremely low minimum values for conductivity and total dissolved solids, which may suggest mulfunctionalities on some sensors.

```
{ data:[{{= {{f"{ Quart: "${Q}", Place: "${PLA}", Cond: ${COND},
                  fDOM_QSU: ${FDOM}, ORP: ${ORP}, Sal_psu: ${SAL},
                  TDS: ${TDS}, Turbidity: ${TUR}, pH: ${PH}, Temp: ${TEMP}
                }"}} : q(Q, R,PLA,COND,FDOM,ORP,SAL,TDS,TUR,PH,TEMP) }} ],
  groupBy:"Quart", groupValues:[["min", "25%", "50%", "75%", "max"]],
  ⋮
```

Additionally, we can create a graphical representation of the aggregated data, allowing quicker insights and comparison between the two locations. *ApexCharts* can be configured to obtain boxplots and radar charts as shown in Figure 8. ■

5. Related Work

Efforts to improve the accessibility and interpretability of ASP have led to the development of various visualization tools that assist users understand the content and structure of answer sets. Early tools such as ASPViz [10], IDPD3 [12], and Kara [11] introduced the idea of using ASP facts to describe visual layouts, effectively allowing logic programs to define how their output should be displayed graphically. These tools typically rely on an auxiliary logic program that encodes visual elements, such as shapes, colors, and positions, as logical atoms, which are then interpreted by an external visualization engine. This approach makes it possible to generate customized visualizations directly from ASP output, without the need for post-processing or manual design. More recent tools, including clingraph [14] and ASPECT [15], build on this idea by offering more expressive and user-friendly declarative languages for visualization. These systems aim to make visualization a first-class citizen in the ASP ecosystem, enabling users to specify graphical representations in a concise and intuitive manner, using logic rules to define visual features. The common underlying principle of all these approaches is to treat visualization as a logic-based task, where a dedicated logic program produces a set of special atoms that encode graphical primitives. These atoms are then interpreted by a rendering component to produce visual outputs, bridging the gap between symbolic reasoning and human-understandable representations.

ASP Chef [16] introduced its first visualization capability in [18], through the inclusion of the *Graph* ingredient, a dedicated component designed to generate graph visualizations as a side output of logic-based recipes. This approach allows users to embed visual instructions directly within ASP programs, producing interactive graph representations that reflect the structure or behavior of the underlying

answer sets, following the same idea of ASPViz, IDPD3 and Kara. Building upon this initial effort, recent versions of ASP Chef have significantly expanded their visualization support by integrating modern JavaScript libraries with a different idea: adapt Mustache templates to ASP in order to configure third-party libraries by querying the processed answer set. These include *@vis.js/Network*, which enables the rendering of dynamic, interactive network diagrams; *Tabulator*, a flexible and highly customizable table library for displaying structured data; and *ApexCharts*, a powerful library for rendering a wide range of charts such as line, bar, and area plots. These enhancements aim to provide users with a richer, more versatile visualization experience, enabling the creation of tailored graphical outputs that align with the structure of ASP solutions. By supporting these libraries, ASP Chef not only improves its usability for teaching and demonstration purposes, but also empowers domain experts to interpret complex answer sets through visually appealing and interactive dashboards.

Traditional machine learning techniques such as Random Forest (RF), Decision Trees (DT), Support Vector Regression (SVR), Linear Regression (LR), and Gradient Boosting Regression (GBR) have been widely employed for water quality prediction tasks, offering interpretable and relatively efficient solutions for estimating key water quality indicators [26, 27, 28]. These models typically rely on handcrafted features derived from environmental sensors and historical records, often requiring careful preprocessing and domain knowledge to achieve accurate results. In parallel, the field has seen growing interest in deep learning approaches, which have demonstrated strong performance in capturing non-linear patterns and temporal dependencies inherent in water quality data. Notably, architectures based on Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks have been applied successfully to model sequential water-related variables [29, 30, 31]. For example, in [29], an LSTM-based model was developed to forecast pH and water temperature in mariculture environments, where water quality is strongly influenced by natural ecological conditions. More specialized models have also emerged. In [27], the authors introduce a prediction framework based on Principal Component Regression (PCR) to estimate the Water Quality Index (WQI) by reducing dimensionality and mitigating multicollinearity among input variables. Another notable contribution is presented in [32], where the authors leverage the strengths of multi-task learning [33] in combination with deep architectures. Their model integrates Convolutional Neural Networks (CNNs) with LSTM layers in a hybrid structure, enabling simultaneous prediction of multiple water quality indicators. These works demonstrate the effectiveness of machine learning and deep learning in environmental monitoring.

6. Conclusion

In this work, we demonstrated how ASP Chef can be effectively used to support the exploration and interpretation of environmental time-series data through a suite of interactive and logic-driven visualizations. By combining ASP with modern JavaScript visualization libraries, we enabled domain experts to go beyond static data inspection and engage in dynamic hypothesis generation, anomaly detection, and multivariate analysis. ASP logic played a central role not only in preprocessing and filtering the data, but also in declaratively expressing domain-specific conditions, such as threshold violations, co-occurring deviations across parameters, or user-defined constraints. These conditions are translated into structured visual outputs, offering a clear and customizable interface to inspect the reasoning results.

A particularly compelling aspect of our approach is the decoupling of logic and presentation: the ASP rules define *what* to show, while the visualization templates control *how* to show it. This modularity promotes the reuse of logic components across different visual formats, facilitates rapid prototyping, and empowers domain experts to explore their data from multiple perspectives without modifying the core data pipeline. Overall, the integration of declarative logic programming with interactive visualization offers a powerful paradigm for data-driven decision-making, especially in domains like environmental monitoring, where expert knowledge, interpretability, and adaptability are key.

Acknowledgments

This work was supported by the Italian Ministry of University and Research (MUR) under PRIN project PRODE "Probabilistic declarative process mining", CUP H53D23003420006, under PNRR project FAIR "Future AI Research", CUP H23C22000860006, under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006, and under PNRR project SERICS "SEcurity and RIghts in the CyberSpace", CUP H73C22000880001; by the Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by the Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); under PN RIC project ASVIN "Assistente Virtuale Intelligente di Negozio" (CUP B29J24000200005); and by the LAIA lab (part of the SILA labs). Mario Alviano is member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-4o for grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[2] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68.

[3] V. Lifschitz, Answer Set Programming, Springer, 2019.

[4] R. Kaminski, J. Romero, T. Schaub, P. Wanko, How to build your own asp-based system?!, Theory Pract. Log. Program. 23 (2023) 299–361.

[5] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, ASP and subset minimality: Enumeration, cautious reasoning and muses, Artif. Intell. 320 (2023) 103931.

[6] P. Cappanera, M. Gavanelli, M. Nonato, M. Roma, Logic-based benders decomposition in answer set programming for chronic outpatients scheduling, TPLP 23 (2023) 848–864. doi:10.1017/S147106842300025X.

[7] M. Cardellini, P. D. Nardi, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, I. Porro, Solving rehabilitation scheduling problems via a two-phase ASP approach, TPLP 24 (2024) 344–367. doi:10.1017/S1471068423000030.

[8] F. Wotawa, On the use of answer set programming for model-based diagnosis, in: H. Fujita, P. Fournier-Viger, M. Ali, J. Sasaki (Eds.), IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings, volume 12144 of *LNCS*, Springer, 2020, pp. 518–529. doi:10.1007/978-3-030-55789-8_45.

[9] R. Taupe, G. Friedrich, K. Schekotihin, A. Weinzierl, Solving configuration problems with ASP and declarative domain specific heuristics, in: M. Aldanondo, A. A. Falkner, A. Felfernig, M. Stettinger (Eds.), Proceedings of the 23rd International Configuration Workshop (CWS/ConfWS 2021), Vienna, Austria, 16-17 September, 2021, volume 2945 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 13–20. URL: https://ceur-ws.org/Vol-2945/21-RT-ConfWS21_paper_4.pdf.

[10] O. Cliffe, M. D. Vos, M. Brain, J. A. Padget, ASPVIZ: declarative visualisation and animation using answer set programming, in: ICLP, volume 5366 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 724–728.

[11] C. Kloimüllner, J. Oetsch, J. Pührer, H. Tompits, Kara: A system for visualising and visual editing of interpretations for answer-set programs, in: INAP/WLP, volume 7773 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 325–344.

[12] R. Lapauw, I. Dasseville, M. Denecker, Visualising interactive inferences with IDPD3, CoRR abs/1511.00928 (2015).
[13] L. Bourneuf, An answer set programming environment for high-level specification and visualization of FCA, in: S. O. Kuznetsov, A. Napoli, S. Rudolph (Eds.), FCA4AI 2018, Stockholm, Sweden, July 13, 2018, volume 2149 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 9–20. URL: https://ceur-ws.org/Vol-2149/paper2.pdf.
[14] S. Hahn, O. Sabuncu, T. Schaub, T. Stolzmann, *Clingraph*: A system for asp-based visualization, Theory Pract. Log. Program. 24 (2024) 533–559. URL: https://doi.org/10.1017/s147106842400005x. doi:10.1017/S147106842400005X.
[15] A. Bertagnon, M. Gavanelli, ASPECT: answer set representation as vector graphics in latex, J. Log. Comput. 34 (2024) 1580–1607. URL: https://doi.org/10.1093/logcom/exae042. doi:10.1093/LOGCOM/EXAE042.
[16] M. Alviano, D. Cirimele, L. A. Rodriguez Reiners, Introducing ASP recipes and ASP Chef, in: ICLP Workshops, volume 3437 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.
[17] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory Pract. Log. Program. 19 (2019) 27–82. doi:10.1017/S1471068418000054.
[18] M. Alviano, L. A. Rodriguez Reiners, ASP chef: Draw and expand, in: P. Marquis, M. Ortiz, M. Pagnucco (Eds.), Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024. URL: https://doi.org/10.24963/kr.2024/68. doi:10.24963/KR.2024/68.
[19] M. Alviano, W. Faber, L. A. Rodriguez Reiners, ASP Chef grows Mustache to look better, 2025. URL: https://arxiv.org/abs/2505.24537. arXiv:2505.24537.
[20] J. S. Mittapalli, M. P. Arthur, Survey on template engines in Java, in: ITM Web of Conferences, volume 37, EDP Sciences, 2021, p. 01007.
[21] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.
[22] S. Molin, Hands-On Data Analysis with Pandas: A Python data science handbook for data collection, wrangling, analysis, and visualization, Packt Publishing Ltd, 2021.
[23] N. Shukla, K. Fricklas, Machine learning with TensorFlow, volume 7, Manning Greenwich, 2018.
[24] M. Gelfond, V. Lifschitz, Logic programs with classical negation, in: D. Warren, P. Szeredi (Eds.), Logic Programming: Proc. of the Seventh International Conference, 1990, pp. 579–597.
[25] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, ASP-Core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. URL: https://doi.org/10.1017/S1471068419000450. doi:10.1017/S1471068419000450.
[26] M. Y. Shams, A. M. Elshewey, E.-S. M. El-Kenawy, A. Ibrahim, F. M. Talaat, Z. Tarek, Water quality prediction using machine learning models based on grid search method, Multimedia Tools and Applications 83 (2024) 35307–35334.
[27] M. S. I. Khan, N. Islam, J. Uddin, S. Islam, M. K. Nasir, Water quality prediction and classification based on principal component regression and gradient boosting classifier approach, Journal of King Saud University-Computer and Information Sciences 34 (2022) 4773–4781.
[28] A. H. Haghiabi, A. H. Nasrolahi, A. Parsaie, Water quality prediction using machine learning methods, Water Quality Research Journal 53 (2018) 3–13.
[29] Z. Hu, Y. Zhang, Y. Zhao, M. Xie, J. Zhong, Z. Tu, J. Liu, A water quality prediction method based on the deep lstm network considering correlation in smart mariculture, Sensors 19 (2019) 1420.
[30] S. Aslan, F. Zennaro, E. Furlan, A. Critto, Recurrent neural networks for water quality assessment in complex coastal lagoon environments: a case study on the Venice Lagoon, Environmental Modelling & Software 154 (2022) 105403.
[31] L. Li, P. Jiang, H. Xu, G. Lin, D. Guo, H. Wu, Water quality prediction based on recurrent neural network and improved evidence theory: a case study of Qiantang River, China, Environmental Science and Pollution Research 26 (2019) 19879–19896.
[32] X. Wu, Q. Zhang, F. Wen, Y. Qi, A water quality prediction model based on multi-task deep learning: a case study of the Yellow River, China, Water 14 (2022) 3408.
[33] R. Caruana, Multitask learning, Machine Learning 28 (1997) 41–75.