LUNAR: A Runtime Verification Tool for Anomaly Detection in Gas Networks

Demonstration Track

Julius Gasson
Imperial College
London, United Kingdom
julius.gasson2023@alumni.imperial.ac.uk

Francesco Belardinelli
Imperial College
London, United Kingdom
francesco.belardinelli@imperial.ac.uk

ABSTRACT

We introduce LUNAR, a framework to detect and classify network anomalies. The tool is designed to (1) synthesize safety constraints expressed in Signal Temporal Logic (STL) based on network data; (2) detect anomalies in new samples wrt the STL constraints; (3) learn to classify anomaly types based on user labels of prior anomalies.

KEYWORDS

Anomaly Detection; Runtime Verification; Utility Networks.

ACM Reference Format:
Julius Gasson and Francesco Belardinelli. 2025. LUNAR: A Runtime Verification Tool for Anomaly Detection in Gas Networks: Demonstration Track. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 3 pages.

1 INTRODUCTION

Utility networks are often modeled as distributed multi-agent systems [8, 15, 16], where individual sensors measure key variables in their respective sections of the network [14]. To ensure the correct behavior of the overall network, it is crucial to perform *anomaly detection* (AD) and *anomaly classification* (AC) to identify any unexpected sensor reading. For this purpose, we introduce LUNAR (*Logic-based Utility Network Anomaly Recognition*). The tool was developed in collaboration with Terranova[1], a software house, which required a solution for monitoring gas networks, but it is applicable to any utility network that outputs regular time series data. The code is publicly available on GitHub[2], as is a video demonstration[3].

2 TOOL DESIGN

Overview. The tool architecture is shown in Figure 1. This is grouped into 4 main components: *data collection* (1-3), *sensor prediction* (4-5), *anomaly detection* (6-7) and *anomaly classification* (8-10).

From its initialization to its full execution, the tool has 4 phases, during which these components are incorporated into the execution process one by one, as the remainder of this section will explain.

[1]https://www.terranovasoftware.eu/en
[2]https://github.com/lunar-rv/lunar-rv
[3]https://github.com/lunar-rv/lunar-rv/raw/refs/heads/main/lunar_demo_video.mp4

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

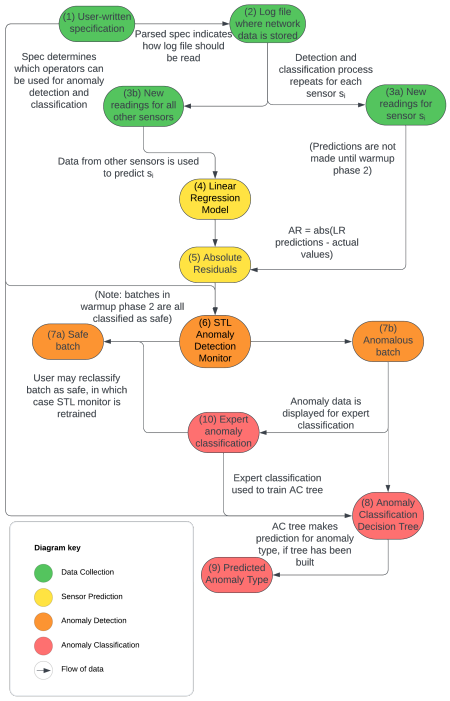

Figure 1: The LUNAR monitoring process.

Warm-up Phase w_1. To detect anomalies, the tool must collect data from the system which it can use to generate rules that model normal behavior. Therefore, in this initial phase, only the *data collection* component is used. Before initialization, the user must write a specification file (1), which specifies the length of w_1 and w_2 and other key monitor attributes. Once this has been parsed, the sensor data can be read. The tool processes data one batch at a time, where each batch consists of a certain number of readings (e.g. a day's worth) from each sensor in the network. When a new batch of readings is written into the log file (2), for each sensor s_i, the data is partitioned into two groups, one with readings from s_i (3a), the other containing the rest (3b). These sets of data are used in the next phase to train the prediction models for each sensor.

Warm-up Phase w_2. In this phase, the *sensor prediction* component is incorporated. For each new batch collected during w_2, the tool carries out the same data collection steps from w_1, but also makes predictions for the value of each sensor's readings at each time step using the values of other sensors. Here, the batches

(a) Detection Accuracy Comparison

Component Variant	*AD Accuracy*	σ
LUNAR	0.9091	0.000278
Ordinary Linear Regression	0.8694	0.000371
Binary Classifier	0.8556	0.000631

(b) Classification Accuracy Comparison

Component Variant	*AC Accuracy*	σ
LUNAR	0.9223	0.00452
Alternative algorithm from [5]	0.9186	0.00420

collected in w_1 are used as training data, along with any batches seen so far in w_2. This step uses *linear regression*, a common technique for AD in time series data [1–3, 9]. We use *non-negative least squares* [13] and *feature selection* [18] in order to avoid overfitting; we also use *rolling window regression* [19] to improve speed. From the predictions, the monitor can obtain an array for each sensor that contains the absolute residuals (prediction error sizes) for the predictions at each time step (5).

Execution Period. At the end of w_2, the expected data patterns in these absolute residuals are represented for each sensor s_i by an STL formula. We refer to [7, 11] for a presentation of STL syntax, robustness semantics and notation conventions. Assuming that sensor readings are recorded regularly, the batch of residuals acquired for each sensor from the prediction phase is modeled as a discrete signal, where the interval between each reading is treated as 1 time step. The process of learning an STL formula requires a set of templates with parameters that can be adjusted to make the formula more or less strict [4]. Here, this set is chosen by the user from the following options (displayed in simplified form), where N is the batch size, $r[t]$ is the value of residuals in the batch at time t, and b_k, μ_k are formula parameters:

$G_{[0,N)}F_{[0,b_1)}(\,r[t] \leq \mu_1)$: in every possible period of b_1 consecutive time steps within the batch, $r[t]$ is below μ_1 at least once.

$G_{[0,N)}(\bar{G}_{[0,b_2)}r[t] \leq \mu_2)$: in every possible period of b_2 consecutive time steps within the batch, $r[t]$ is on average below μ_2.

$G_{[0,N)}\ (r[t] \leq \mu_3)$: $r[t]$ is always below μ_3.

To learn the parameters b and μ, we use simulated annealing [12], using the batches processed during the warm-up phases as training data. The objective function is the *tightness* of each resulting formula, explained as follows. If a new batch of residuals contains values larger than the absolute residuals in the training batches by more than a small amount, the new batch should be flagged as anomalous. Therefore, μ_k and b_k are chosen for each template so that the robustness value of the resulting formula is small and positive in relation to each trace. This approach is comparable to a one-class Support Vector Machine (SVM) [17], where the model learns a decision boundary that *tightly* encloses the majority of the training data, typically representing the normal class. The three templates are combined using the conjunction $\wedge$ to form φ_i, the actual AD formula for sensor s_i. Now, when a new batch arrives, all the *data collection* and *sensor prediction* steps are carried out as in w_2, creating a batch of absolute residuals r_i for each sensor. Then, the *anomaly detection* component of the tool becomes part of the process. The monitor applies φ_i to r_i (6), and classifies r_i as safe if φ_i is satisfied (7a), and anomalous if it is violated (7b).

Anomaly Classification. If r_i is classified as anomalous, the tool aims to predict what type of anomaly might have occurred (9). This is done using an STL decision tree [5, 6] (8), in which the splitting rule at each node is an STL formula whose parameters are learned to maximize the information gain. Each sensor has its own AC tree. To build this tree, the tool relies on previously detected anomalies that have been labeled by the user. Each time an anomaly is detected, the user is prompted to supply an anomaly type (10), based on information about the batch displayed by the tool. This input is used to create and update the decision tree. Alternatively, if the user enters the word 'safe', the AC tree is not updated, and the AD formula is relearned to accommodate this new batch.

The AC algorithm is similar to the online algorithm for STL decision trees laid out in [5]. In our algorithm, however, the tree is only rebuilt downwards from its leaves, and a new leaf node is only created if a set of criteria is met. Here, since few anomalies are expected for any sensor, it is useful to be able to update the tree using a sample of limited size. Hence, LUNAR has no such restrictions for growing the tree: whenever a new anomaly is classified, all relevant nodes are updated to ensure that the tree's splitting rules are always optimal for all seen data.

Implementation. The framework is implemented in Python. The specification uses a custom language; from this, a parser builds a Python data structure, which is then passed as an argument into the monitor function. This function contains a main loop, which runs until no more data remains in the source file, or until it is ended by the user. The LinearRegression model from scikit-learn is used in the linear regression wrapper, and STL robustness is calculated using numpy ufuncs for fast vectorized operations.

3 EXPERIMENTAL EVALUATION

We assessed the performance of LUNAR's individual components using a dataset provided by Terranova, which contained some minor synthetic anomalies [10]. For this, we built 3 copies of the tool; in each of these, a different component of the tool was replaced by an alternative algorithm for the same task. These alternatives were: an ordinary linear regression model for sensor prediction in place of stage (4) from Figure 1, a binary classifier for AD instead of (6) and the STL decision tree algorithm from [5] instead of (8). The results are not indicative of LUNAR's ability to detect real-life anomalies, but rather show how it performs compared to these alternatives. Tables 1a and 1b present the results, showing the average of 3 experiments; the standard deviation σ is also recorded.

The results show that each of LUNAR's components performs better than their alternatives. Despite the inferior accuracy, the algorithm from [5] was noticeably faster for AC. However, this is no great concern as few anomalies are expected for each sensor.

4 ACKNOWLEDGEMENTS

The research described in this paper is supported by the EPSRC grant number EP/X015823/1. The authors acknowledge the contribution of L. Machetti, A. Mella, and G. Lorenzini at Terranova.

REFERENCES

[1] Hermine N. Akouemo and Richard J. Povinelli. 2016. Probabilistic anomaly detection in natural gas time series data. *International Journal of Forecasting* 32, 3 (2016), 948–956. https://doi.org/10.1016/j.ijforecast.2015.06.001

[2] Mustafa Akpinar and Nejat Yumuşak. 2017. Naive forecasting of household natural gas consumption with sliding window approach. *Turkish Journal of Electrical Engineering and Computer Sciences* 25, 1 (2017), 30–45. https://doi.org/10.3906/elk-1404-378

[3] Lorenzo Baldacci, Matteo Golfarelli, Davide Lombardi, and Franco Sami. 2016. Natural gas consumption forecasting for anomaly detection. *Expert Systems with Applications* 62 (2016), 190–201. https://doi.org/10.1016/j.eswa.2016.06.013

[4] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Nickovic. 2022. Survey on mining signal temporal logic specifications. *Information and Computation* 289 (2022), 104957. https://doi.org/10.1016/j.ic.2022.104957

[5] Giuseppe Bombara and Calin Belta. 2021. Offline and Online Learning of Signal Temporal Logic Formulae Using Decision Trees. *ACM Trans. Cyber-Phys. Syst.* 5, 3, Article 22 (3 2021), 23 pages. https://doi.org/10.1145/3433994

[6] Giuseppe Bombara, Cristian-Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. 2016. A Decision Tree Approach to Data Classification using Signal Temporal Logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control* (Vienna, Austria) *(HSCC '16)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/2883817.2883843

[7] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. 2017. Robust online monitoring of signal temporal logic. *Formal Methods in System Design* 51, 1 (8 2017), 5–30. https://doi.org/10.1007/s10703-017-0286-7

[8] Manuel Herrera, Ajith Kumar Parlikad, Joaquín Izquierdo, and Marco Perez Hernandez. 2020. Multi-Agent Systems and Complex Networks: Review and Applications in Systems Engineering. *Processes* 3, 8 (03 2020). https://doi.org/10.3390/pr8030312

[9] Tao Hong, Jason Wilson, and Jingrui Xie. 2014. Long Term Probabilistic Load Forecasting and Normalization With Hourly Information. *IEEE Transactions on Smart Grid* 5, 1 (2014), 456–462. https://doi.org/10.1109/TSG.2013.2274373

[10] Jie Hu, Yawen Huang, Yilin Lu, Guoyang Xie, Guannan Jiang, Yefeng Zheng, and Zhichao Lu. 2024. AnomalyXFusion: Multi-modal Anomaly Synthesis with Diffusion. arXiv:2404.19444 [cs.CV] https://arxiv.org/abs/2404.19444

[11] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. 2017. TeLEx: Passive STL Learning Using Only Positive Examples. In *Runtime Verification*, Shuvendu Lahiri and Giles Reger (Eds.). Springer International Publishing, Cham, 208–224.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671 arXiv:https://www.science.org/doi/pdf/10.1126/science.220.4598.671

[13] C. Lawson and R. Hanson. 1995. 23. Linear Least Squares with Linear Inequality Constraints. , 158-173 pages. https://doi.org/10.1137/1.9781611971217.ch23

[14] Om Mahela, Mahdi Khosravy, Neeraj Gupta, Baseem Khan, Hassan Haes Alhelou, Rajendra Mahla, Nilesh Patel, and Pierluigi Siano. 2020. Comprehensive Overview of Multi-Agent Systems for Controlling Smart Grids. https://doi.org/10.17775/CSEEJPES.2020.03390

[15] R.R. Negenborn, B. De Schutter, and J. Hellendoorn. 2008. Multi-agent model predictive control for transportation networks: Serial versus parallel schemes. *Engineering Applications of Artificial Intelligence* 21, 3 (2008), 353–366. https://doi.org/10.1016/j.engappai.2007.08.005

[16] Gaurav Singh Negi, Anupama Mishra, Mukul Kumar Gupta, Nitin Kumar Saxena, DilipKumar Jang Bahadur Saini, and Kapil Joshi. 2024. Microgrid digital twins: concepts and their controlling through multi-agent systems. *Int. J. High Perform. Syst. Archit.* 11, 4 (7 2024), 225–236. https://doi.org/10.1504/ijhpsa.2023.139898

[17] Zineb Noumir, Paul Honeine, and Cédue Richard. 2012. On simple one-class classification methods. In *2012 IEEE International Symposium on Information Theory Proceedings*. IEEE, Cambridge, MA, 2022–2026. https://doi.org/10.1109/ISIT.2012.6283685

[18] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. 2007. Filter methods for feature selection–a comparative study. , 178–187 pages.

[19] Eric Zidot and Jiahui Wang. 2006. *Rolling Analysis of Time Series*. Springer New York, New York, NY, 313–360. https://doi.org/10.1007/978-0-387-32348-0_9