




# Inductive Learning-Based Qualitative Fault Diagnosis in Distributed Systems

Márton Tarnay , András Földvári , Bertalan Zoltán Péter 

Budapest University of Technology and Economics

Department of Artificial Intelligence and Systems Engineering

Budapest, Hungary

Email: {tarnay.marton, andras.foldvari, bpeter}@edu.bme.hu

**Abstract**—The growing complexity of microservice systems poses significant challenges in diagnosing faulty systems. Traditional monitoring techniques often fall short due to the distributed and dynamic nature of these systems. This paper presents a novel model-based diagnostics framework that uses multimodal observability data for accurate fault detection and localization in microservice environments.

The diagnostic process uses Answer Set Programming (ASP), a declarative programming language that leverages logic reasoning over a qualitative multimodal data model to provide insights into the system’s state. The presented approach introduces an inductive learning solution for extracting the diagnostic rules, utilizing Inductive Learning of Answer Set Programs (ILASP) to derive explainable diagnostic rules from labeled historical datasets automatically.

The approach was evaluated on a benchmark microservice application dataset with promising results compared to existing fault detection and diagnostic solutions.

**Index Terms**—fault diagnosis, distributed systems, logic reasoning, microservices, qualitative modeling, distributed tracing, observability, answer set programming, inductive learning

## I. INTRODUCTION

Microservice architecture is a modern software development paradigm that has emerged as the prominent way to design rapidly evolving cloud-native applications. Microservices (Figure 1) are small, independently deployed services that communicate over lightweight APIs and are built around business domains [1].

While being highly scalable and flexible, microservice systems introduce significant challenges in observability and understanding system behavior due to their highly complex, decentralized, and modular design [2]. The heterogeneous nature of the architecture and constantly changing topology make traditional monitoring and fault localization techniques difficult. The reliability and availability requirements in these systems necessitate approaches that can tackle multimodal observability data and provide operators with explainable diagnostic results.

The Cloud Native Computing Foundation (CNCF) [3] describes observability as:

The work of Bertalan Z. Péter supported by the Doctoral Excellence Fellowship Programme (DCEP) is funded by the National Research Development and Innovation Fund of the Ministry of Culture and Innovation and the Budapest University of Technology and Economics under a grant agreement with the National Research, Development and Innovation Office.

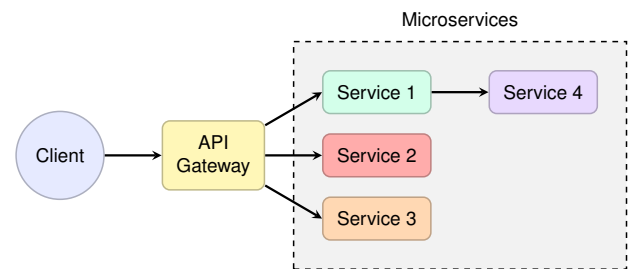


Figure 1. Simplified Logical Structure of Microservice Architecture

*“A system property that defines the degree to which the system can generate actionable insights. It allows users to understand a system’s state from these external outputs and take (corrective) action.”*

Observability is essential for understanding the health and performance of complex systems, particularly in microservice architectures. At its core, observability is built upon three main pillars [4]: logs, metrics, and traces. These data types provide a multi-aspect approach to monitoring and diagnosing system behavior. Each one offers unique insights that come together to give a complete picture of how a system works.

The goal of the paper is to address two main challenges concerning fault diagnosis and observability in microservice systems [5]: multimodal telemetry data representation and creating explainable diagnostic results.

The paper presents a novel model-based diagnostics framework for microservice systems, utilizing a joint metamodel representing multimodal telemetry data. The solution leverages Answer Set Programming (ASP) [6], a logic-based reasoning engine, and Inductive Learning of Answer Set Programs (ILASP) [7], an inductive logic programming (ILP) approach, to analyze this fused data, allowing for the identification of faults across interconnected microservice components. This approach tackles the challenges of fault localization in microservice systems by generating explainable diagnostic results and providing actionable insights for operations teams, ultimately improving system reliability and reducing operational workload.

## II. INDUCTIVE LEARNING

The goal of inductive learning in the solution is to extract general diagnostic rules from historical multimodal data. These interpretable rules can be applied in the diagnostic process to precisely identify the location and type of faults.

Inductive learning derives rules from observations that cover the patterns without overgeneralizing, ensuring that the rules are neither too loose – potentially covering irrelevant cases – nor too strict, which might exclude relevant ones.

In our approach, we used ILASP [7] to extract rules from the ASP representation of observations, ensuring that the rules accurately diagnose faults while minimizing the number of false positive results.

### A. Answer Set Programming

Answer Set Programming (ASP) is a form of declarative programming oriented towards difficult search problems [6]. The approach originates from the field of logic programming and non-monotonic reasoning. ASP utilizes the concept of stable model semantics to represent and solve problems. Problems are encoded as logic programs, and the solutions to the problems correspond to answer sets.

In ASP, we start with a finite set  $\mathcal{A}$  of propositional atoms. A *literal* is either an atom  $a \in \mathcal{A}$  or a default-negated atom  $\text{not } a$ . A *body* is a set of such literals, often split into the positively included atoms  $B^+$  and the default-negated atoms  $B^-$ . A *head* is a set of atoms  $\{a_1, \dots, a_k\} \subseteq \mathcal{A}$ . An ASP rule  $r$  has the form  $H \leftarrow (B^+, \text{not } B^-)$ , where  $H$  is the head and  $B^+ \cup B^-$  is the body. A *fact* is simply a rule with an empty body, indicating that the head holds unconditionally. Finally, an ASP program  $P = \{r_1, r_2, \dots, r_n\}$  is a finite set of such rules, encompassing facts and rules, and, optionally, *constraints* (i.e., rules with empty heads).

An *answer set* is a consistent set of literals that represent a possible solution that satisfies all rules and constraints of the program.  $AS(P)$  is the operator that, given a logic program  $P$ , defines the procedure for computing all of its answer sets.

### B. Inductive Logic Programming

The paradigms of inductive logic programming (ILP) and ILASP are used in this paper to support inductive diagnostic rule generation described in detail in section IV. The advantage of ILP systems is their ability to learn rules that can be explained and interpreted [8]. Compared to other state-of-the-art artificial intelligence and machine learning techniques, which function like black-boxes, ILP systems work transparently with algorithms that can be fully explained.

Formally, an ILP task is a tuple  $T = \langle B, S_M, \langle E^+, E^-, C \rangle \rangle$  where  $B$  is an ASP program,  $S_M$  is a set of ASP rules,  $E^+$  and  $E^-$  are the finite sets of positive and negative *examples*, and  $C$  is the optional context of the example in the form of an answer set program.  $H \subseteq S_M$  hypotheses is an inductive solution of  $T$  iff.:

$$\forall e^+ \in E^+ : \exists A \in AS(B \cup H \cup C) \quad : A \text{ extends } e^+ \quad (1)$$

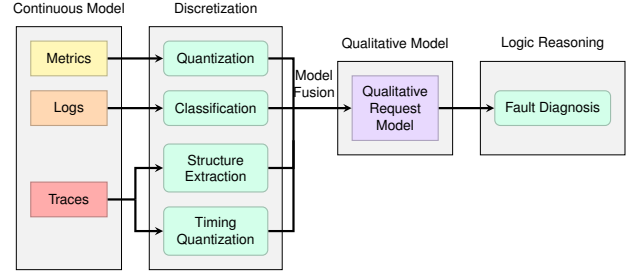


Figure 2. Model Extraction and Reasoning

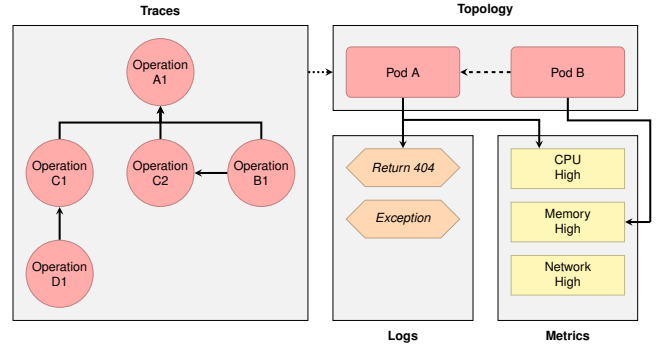


Figure 3. Aggregate Model of the Telemetry Data

and

$$\forall e^- \in E^- : \nexists A \in AS(B \cup H \cup C) \quad : A \text{ extends } e^- \quad (2)$$

(1) means that for all positive examples, there must be a rule that covers the example in the set of answer sets of the union of the background knowledge and the hypothesis. (2) means that for all negative examples, there must not be any rule in the answer sets that cover the negative example [7].

Inductive Learning of Answer Set Programs (ILASP) is a novel system for machine learning of ASP programs from examples [7]. ILASP works by creating a meta-level representation of the candidate hypotheses: rules to be learned in the form of mode declarations. Mode declarations specify the allowed structure of rules, which are used to generate all possible rule structures. Examples and context are translated into constraints. These constraints are used to test the candidate's hypotheses, as the solution must satisfy the positive examples and not violate the negative examples.

## III. SYSTEM DIAGNOSIS WITH MULTIMODAL DATA

The first step of the model extraction and diagnostic reasoning process (Figure 2) is to create a unified model representation from multimodal data. This unified representation enables the generation of a diagnostic code (in our case ASP), which subsequently facilitates the system diagnostic task through the use of logic reasoning.

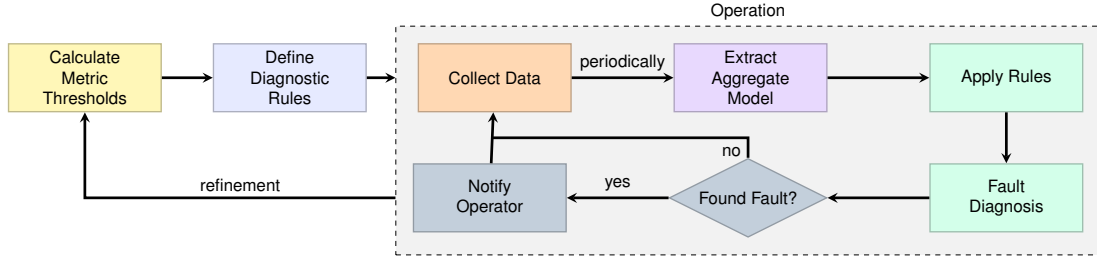


Figure 4. Diagnostic Workflow

### A. Heterogeneous Aggregated Metamodel

For the model created from multimodal data to be viable, an efficient, automatic process is required for the creation of models from the collected telemetry data. The properties are extracted from log, metric, and trace data. The current solution directly generates ASP code from the metamodel. However, there is a possibility for an intermediate representation allowing for the use of other toolchains.

The qualitative representation of data supports the creation of an aggregated model, enabling the compact representation of large datasets.

An example of the reduced and aggregated model can be seen in Figure 3, as a visual representation of the ASP program of the model. This model includes the operation graph. The aggregated logs and metrics are displayed, summarizing the significant anomalies in the time period.

This aggregated data model allows for faster inductive learning without sacrificing diagnostic performance. Using this method also allows for greater scalability, both in the rule extraction and operation phase.

### B. Diagnostics

Combining the ASP program of the aggregate model with the diagnostic rules yields a result that contains a set of fault literals representing the identified faults in the system. These fault literals capture the root cause of the diagnosed fault, while the relevant anomalies are also provided for an interpretable result.

The complete workflow of the created diagnostic approach can be seen in Figure 4. Metric thresholds are created from previously collected data, and the rules are defined before operation. During operation, the system periodically extracts the aggregate model based on the request models representing collected data. The diagnostic rules are evaluated over the aggregate model, and the resulting diagnosis is relayed to the operators in case a fault is detected. The diagnosis may also be used to execute automated corrective actions on the system.

## IV. RULE EXTRACTION WITH INDUCTIVE LEARNING

Manually defining diagnostics rules is time-consuming and complex, as it requires deep knowledge of the system architecture and behavior, potential failure modes, and even operational nuances. The application of inductive learning offers a solution to this challenge by automatically creating

diagnostic rules. With the analysis of labeled historical data, inductive algorithms are able to identify the correlations and patterns needed for fault diagnosis in the system.

### A. Task Definition

First, let us establish the following notations.

- Let  $\mathcal{O}$  denote the set of all operations within the system, and let us define  $\mathcal{OG} \subseteq \mathcal{O}^2$  where every  $(o, o_{\text{parent}}) \in \mathcal{OG}$  pair represents an execution path going from the parent to the child operation.
- Let  $\mathcal{S}$  denote the set of services in the system.
- Let  $\mathcal{M} \subseteq \mathcal{MT} \times \mathcal{MV} \times \mathcal{S}$  be the set of metric observations in the system where  $\mathcal{MT} \stackrel{\text{eg}}{=} \{\text{Memory, CPU, ...}\}$  is the set of observed metric types, and  $\mathcal{MV} \stackrel{\text{eg}}{=} \{\text{Low, High, ...}\}$  is the set of qualitative observation values.
- Let  $\mathcal{L} \subseteq \mathcal{LT} \times \mathcal{S}$  be the set of log observations extracted from the system where  $\mathcal{LT} \stackrel{\text{eg}}{=} \{\text{HTTP\_404, EXCEPTION, ...}\}$  is the set of log types.
- Let  $\mathcal{F} \subseteq \mathcal{FT} \times \mathcal{S}$  be the set of faults in the system where  $\mathcal{FT} \stackrel{\text{eg}}{=} \{\text{CPU\_CONTENTION, CPU\_CONSUMPTION, ...}\}$  is the set of known fault types.
- We partition  $\mathcal{F}$  into disjoint subsets  $\mathcal{F}^+$  and  $\mathcal{F}^-$ , comprising the active and inactive faults in the context of the evaluation, respectively.

To create inductive rules, we propose to encode this problem as *ILP tasks* as follows. Let the rule extraction problem be ILP task tuples of the form  $T = \langle B, S_M, \langle E^+, E^-, C \rangle \rangle$ , where  $S_M$  is the hypothesis space defined by the head and body mode declarations  $M = M_h \cup M_b$  that are used to generate the hypothesis space.

The heads are defined as  $M_h = \mathcal{F}$ . The bodies can be defined with the observation sets:  $M_b = \mathcal{M} \cup \mathcal{L} \cup \mathcal{OG}$ .

The context  $C$  contains all observations from the aggregation as  $C = \mathcal{L} \cup \mathcal{M} \cup \mathcal{OG}$ , in the form of ASP facts.

Finally,  $E^+ = \mathcal{F}^+$  and  $E^- = \mathcal{F}^-$ .

### B. Workflow

This section describes the workflow of inductive diagnostic rule extraction in detail, focusing on the creation of ILASP programs from a fault-labeled monitoring dataset. The complete workflow is shown in Figure 5.

The first step in the process is extracting all faults and their relevant data from a labeled dataset with faults in them. For

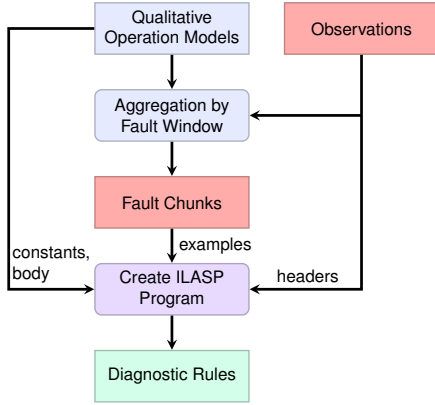


Figure 5. Rule Extraction Workflow

each active fault, the impacted timeslice, which encompasses the duration in which the fault occurred, must be processed; request models shall be created and aggregated into an aggregate model. Following data collection and aggregate model extraction, an ILASP program is created for each microservice.

Each ILASP program is tailored to a specific microservice in the system. The mode headers reflect only the faults injected into the particular microservice assigned to them, meaning that the results of the program will only create rules for that microservice relevant to the context of that service.

Each example in the program is a fault-affected timeslice; all timeslices have to be included across all ILASP programs. This is critical because it allows for the analysis of examples where the inspected microservice is fault-free, providing a more robust dataset for inductive learning. The results of the execution of these ILASP programs are the fault diagnosis rules. These rules encapsulate the learned relationships and patterns identified by inductive learning.

### C. Example

Listing 1 shows two examples from a simple ILASP program. The program contains a single fault, named `exampleFault`. There are two partial interpretations, `aggregate1` and `aggregate2` – the fault is active in the former and inactive in the latter. The only context given is a metric, named `someMetric`, which has a `high` anomaly in `aggregate1`. After running this simple program, the result is a rule that states that the example fault is active when `someMetric` is `high`. This demonstrates the basic principle of how partial interpretations work for diagnostic rules. The result of Listing 1 is:

```
fault(exampleFault) :- someMetric(high).
```

Listing 2 shows an excerpt from an ILASP program with two positive interpretations: `aggregate1` and `aggregate2`. `aggregate1` has a weight of 10 – this penalty value is calculated based on the relative relevance of the example in the diagnosis rule extraction. The value is higher if the example contains an actual fault in the service. It is also influenced by the amount of anomalies present in the context

```
#modeh(fault(exampleFault)).
#modeb(someMetric(const(metricLevel))).
metricLevel(high).
metricLevel(low).

#pos(aggregate1,
  { fault(cpuConsumed) },
  {},
  { someMetric(high) }
).
#pos(aggregate2,
  {},
  { fault(cpuConsumed) },
  {}
).
```

Listing 1: Example ILASP Program for Fault Rule Extraction

```
#pos(aggregate1@10,
  { fault(cpu_contention, frontend) },
  { fault(return, frontend) },
  { cpuMetric(high, frontend).
    memMetric(high, frontend). }
).
#pos(aggregate2@15,
  { fault(return, frontend) },
  { fault(cpu_contention, frontend) },
  {
    log(400, frontend).
    log(error, frontend).
    netMetric(high, frontend).
    has_parent(
      frontend_ErrorHandler,
      frontend_Open
    )
  }
).
```

Listing 2: Example ILASP Program with Penalties

aggregate model. The *positive example* part of the program is the `cpu_contention` fault localized in the `frontend` service. The *negative example* part is the `return` fault also localized in the `frontend` service. The *context* is the `high` CPU metric and `high` memory metric in the `frontend` service. `aggregate2` has a penalty of 15 – this value is higher because it contains more anomalies in the aggregate model. The *positive example* part is the `return` fault in the `frontend` service, and the *exclusions* part is the CPU contention fault also in the `frontend` service. The *context* describes two logs – a 400 HTTP response and an error log – and a high network latency metric. It also contains an operation relationship: the `Frontend_ErrorHandler` operation is called by the `Frontend_Open` operation.

This example shows how different penalties can be set for partial interpretation based on how heavily they should influence the rule learning process. This is essential, as real-world data usually contains noise, either from measurement errors or fault mislabeling. The example also shows how all kinds of multimodal observability data can be used to give context to the fault slices, allowing for the generation of more accurate diagnostic rules.

## V. EVALUATION

The Online Boutique dataset was used for the evaluation of the developed framework. This dataset was chosen for its use

of traces, metrics, and logs, as well as its previous usage for benchmarking contemporary diagnostic tools [9]. The dataset was extracted from a real, commonly employed benchmark system for microservice research and development.

Online Boutique [10] is a microservices demo application created by Google to demonstrate their cloud-native tools and services, focusing on Kubernetes and Google Cloud Operations. This application is built as an online store; the primary user journey in the application consists of the user selecting an item, adding it to their cart, and making the purchase.

#### A. Dataset

The dataset contains several kinds of faults injected into several different services. The following fault modes are used:

**CPU contention** occurs when multiple services are competing for CPU resources, leading to decreased performance because the CPU is overloaded with tasks.

**CPU consumption** means that a single service consumes an excessive amount of CPU resources, leading to performance degradation.

**Network delay** introduces latency in network communication between services. It causes delays in data transmission, leading to slower response times and timeouts.

**Exception** involves unexpected errors or exceptions occurring within a service’s execution. This causes the service to crash or behave unpredictably.

**Return** refers to a service returning with unexpected behavior, which can be due to logic errors, incorrect configuration, or data corruption, affecting the functionality of dependent services.

The first two faults represent node-level performance issues in a system. *Network delay* is intended to show faults in the communication network of the system. *Exception* and *Return* faults are application-level faults, which resemble configuration issues and software bugs. By using these five fault modes, the approach can be tested regarding its viability in diagnosing many areas of network, infrastructure, and application-level faults while minimizing the complexity of the results.

#### B. Results

Inductive rule extraction was performed on the training dataset from the Online Boutique application. This yielded 13 service-specific rules and 4 general rules. There was significant variety in the execution times of the ILASP programs for specific services, ranging from around 10 to 100 seconds, depending on the complexity of the service and the examples.

The diagnostic evaluation was done by taking each fault injection window (the timeslice where a specific fault is active) in the dataset as the input for fault diagnosis. Specific diagnostic rules were learned for each service in the system, while general diagnostic rules were aggregated from the results of multiple services.

The results were evaluated using the *Accuracy@1* ( $A@1$ ) – also known as Top-1 Accuracy – and *Accuracy@3* ( $A@3$ ) statistics, commonly used to evaluate the performance of

Table I  
RESULTS OF THE ONLINE BOUTIQUE BENCHMARK.  
COMPARISON DATA FROM NEZHA [9]

Approach	A@1	A@3
MicroScope [11]	12.5	41.07
MicroRCA [12]	16.07	62.5
SBLD [13]	19.64	23.21
LogFaultFlagger [14]	19.64	21.42
MicroRank [17]	41.07	48.21
TraceAnomaly [15]	30.35	33.92
PDiagnose [16]	41.07	73.21
Nezha [9]	92.86	96.43
<b>Presented Solution</b>	<b>37.5</b>	<b>50</b>

models in classification tasks, such as recommender systems and multi-class classification problems.

*Accuracy@1* measures the percentage of test cases where the correct answer was the top prediction. *Accuracy@3* measures the percentage of test cases where the top three predictions were correct. As the developed solution does not rank the possible diagnosis results but may present multiple answers, the correct  $A@1$  test cases are considered when the only given answer is correct.  $A@3$  test cases are considered correct if there are at most three possible diagnosis results, and one of them is correct.

After running the diagnostics framework on the benchmark dataset, the  $A@1$  result was 37.5%, while the  $A@3$  result was 50%. These results were compared to results [9] from other diagnostic and root cause analysis tools, using the same dataset, as seen in Table I. The developed solution is superior to MicroScope [11], MicroRCA [12], SBLD [13], LogFaultFlagger [14], and TraceAnomaly [15] in Top-1 accuracy. However, the results are slightly inferior to PDiagnose [16], a multimodal approach. Nezha [9], a novel multimodal approach, shows a significant accuracy difference between the solution presented in this paper, but also to other state-of-the-art approaches. It is clear that there is a great room for further development. However, compared to the imperative methodologies commonly used by other approaches, the declarative, logic-based approach presented here allows for enhanced scalability and flexibility. These results prove that the developed framework has potential, even in this early state.

#### C. Scalability

The time required to solve an ILP problem depends heavily on the size of the search space and the complexity of the background knowledge. For example, choice rules generate multiple models, each of which has to be evaluated. The number of examples also plays a smaller factor in the runtime; however, this was barely noticeable at the scale of the benchmark datasets used in this evaluation. The main concern in terms of scalability is increasing the complexity of the mode declarations. Using more types of variables and constants in these literals leads to an increase in the hypothesis space.

In the presented solution, the background knowledge does not affect the process since the program does not include choice rules. However, to decrease the number of constant types in the head and body declarations, we chose to decompose and extract rules based on services. This allowed for shorter execution times, facilitating rapid prototyping with different models and configurations necessary for research.

## VI. CONCLUSION

This paper aimed to address two main challenges concerning fault diagnosis and observability in microservice systems: multimodal telemetry data representation and creating explainable diagnostic results. The following contributions were made to tackle these challenges.

We presented a metamodel and aggregate model capable of handling heterogeneous observability data, integrating multiple types of observability data (metrics, logs, and traces) to facilitate a comprehensive view of system behavior and enable further analysis. Leveraging the power of the created metamodel, a modular diagnostic framework using ASP integrates the processing of telemetry data, qualitative model extraction, and the application of diagnostic rules, enabling precise and interpretable diagnostics in microservice environments.

To automate diagnostic rule extraction, a novel solution was presented that uses ILP to learn rules from historical data. This technique is able to systematically derive diagnostic rules by analyzing fault-containing datasets, enabling the discovery of complex relationships and causalities.

The developed framework was evaluated using a fault-injected dataset based on Google's Online Boutique system. The solution was able to outperform several contemporary diagnostic tools, showing the potential for accurate, interpretable diagnostics in microservice systems. The developed models and framework lays the groundwork for further research in the field of observability and fault diagnosis.

### Future Work

Other applications of the created metamodel, model extraction, and inductive reasoning framework should be explored. The developed approaches could have further uses ranging from auto-scaling, fulfilling extra-functional requirements, and even analyzing user behavior.

Improving the qualitative diagnostic process could yield more robust and accurate results. The use of clustering algorithms for metrics, qualitative knowledge discovery [18], and incorporating quantitative elements should be considered. Employing error propagation analysis (EPA) could enable the diagnosis of more complex faults [19].

Using a microservice test environment would allow for evaluating the solution and further improvements over a more controllable and diverse set of data.

## REFERENCES

[1] S. Wells, *Enabling Microservice Success*. O'Reilly, 2024, ISBN: 9781098130794.

[2] L. Giamattei, A. Guerriero, R. Pietrantuono, *et al.*, "Monitoring tools for devops and microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 208, p. 111906, 2024, ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2023.111906>.

[3] *Cloud native glossary: Observability*, <https://glossary.cncf.io/observability/>, Accessed: 2024-10-22.

[4] M. Usman, S. Ferlin, A. Brunstrom, and J. Taheri, "A survey on observability of distributed edge & container-based microservices," *IEEE Access*, vol. 10, pp. 86904-86919, 2022. DOI: [10.1109/ACCESS.2022.3193102](https://doi.org/10.1109/ACCESS.2022.3193102).

[5] S. Zhang, S. Xia, W. Fan, *et al.*, *Failure diagnosis in microservice systems: A comprehensive survey and analysis*, 2024. arXiv: [2407.01710](https://arxiv.org/abs/2407.01710). [Online]. Available: <https://arxiv.org/abs/2407.01710>.

[6] V. Lifschitz, "What is answer set programming?" In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, ser. AAAI'08, Chicago, Illinois: AAAI Press, 2008, pp. 1594-1597, ISBN: 9781577353683.

[7] M. Law, A. Russo, and K. Broda, "Inductive learning of answer set programs," in *Proceedings of the 14th European Conference on Logics in Artificial Intelligence - Volume 8761*, Berlin, Heidelberg: Springer-Verlag, 2014, pp. 311-325. DOI: [10.1007/978-3-319-11558-0\\_22](https://doi.org/10.1007/978-3-319-11558-0_22).

[8] S. Muggleton and L. de Raedt, "Inductive logic programming: Theory and methods," *The Journal of Logic Programming*, vol. 19-20, pp. 629-679, 1994, Special Issue: Ten Years of Logic Programming. DOI: [https://doi.org/10.1016/0743-1066\(94\)90035-3](https://doi.org/10.1016/0743-1066(94)90035-3).

[9] G. Yu, P. Chen, Y. Li, H. Chen, X. Li, and Z. Zheng, "Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data," in *ESEC/FSE 2023*, ACM, 2023.

[10] Google, *Online boutique application*, <https://github.com/GoogleCloudPlatform/microservices-demo>, Accessed: 2024-10-23.

[11] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *Service-Oriented Computing*, C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds., Cham: Springer International Publishing, 2018, pp. 3-20, ISBN: 978-3-030-03596-9.

[12] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1-9. DOI: [10.1109/NOMS47738.2020.9110353](https://doi.org/10.1109/NOMS47738.2020.9110353).

[13] C. M. Rosenberg and L. Moonen, "Spectrum-based log diagnosis," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '20, Bari, Italy: Association for Computing Machinery, 2020, ISBN: 9781450375801. DOI: [10.1145/3382494.3410684](https://doi.org/10.1145/3382494.3410684).

[14] A. Amar and P. C. Rigby, "Mining historical test logs to predict bugs and localize faults in the test logs," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 140-151. DOI: [10.1109/ICSE.2019.00031](https://doi.org/10.1109/ICSE.2019.00031).

[15] P. Liu, H. Xu, Q. Ouyang, *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 48-58. DOI: [10.1109/ISSRE5003.2020.00014](https://doi.org/10.1109/ISSRE5003.2020.00014).

[16] C. Hou, T. Jia, Y. Wu, Y. Li, and J. Han, "Diagnosing performance issues in microservices with heterogeneous data source," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 493-500. DOI: [10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00074](https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00074).

[17] G. Yu, P. Chen, H. Chen, *et al.*, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proceedings of the Web Conference 2021*, ser. WWW '21, Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 3087-3098, ISBN: 9781450383127. DOI: [10.1145/3442381.3449905](https://doi.org/10.1145/3442381.3449905).

[18] G. Kern-Isberner, M. Thimm, and M. Finthammer, "Qualitative knowledge discovery," in *Semantics in Data and Knowledge Bases*, K.-D. Schewe and B. Thalheim, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 77-102, ISBN: 978-3-540-88594-8.

[19] M. Tarnay, *Managing operational uncertainties in deployed systems using logic reasoning*, <https://tdk.bme.hu/conference/VIK/2023/sessions/distsys/paper/Mukodesi-Bizonnyaltalansagok-Kezelese-Elosztott>, 2023.