# Graph Neural Networks for Metrics Prediction in Microservice Architecture

Anna Golovkina(✉), Dmitry Mogilnikov, and Vladimir Ruzhnikov

Saint Petersburg State University, Saint Petersburg, Russia
a.golovkina@spbu.ru

**Abstract.** The article examines the issues of monitoring performance in microservice architectures. We explore the problem of forecasting performance indicators as well as fault propagation in such systems, which are distributed and have independent service deployments. The paper addresses these issues by proposing a novel approach that uses multimetric time series data to establish causal relationships between microservices and build graph neural networks based on revealed system dependencies. The method's goal is to proactively forecast performance indicators and fault propagation in order to assure the resilience and reliability of microservices. Various graph neural network architectures are discussed. The best one DCRNN uses a diffusion convolutional recurrent neural network in a basis and is able to predict well both on data with and without anomalies.

**Keywords:** Graph neural networks · Microservice · Metrics prediction · microservice preformance

## 1 Introduction

The shift towards microservice architectures in software development has become widespread, dividing monolithic architectural parts into smaller interconnected elements. This approach is used for various software solutions, including cloud-based applications. Microservices-based applications can be implemented by various service providers using various technologies or programming languages in multiparty and open network environments [5,24].

However, in contrast to a monolithic application, there is a need to handle service-to-service communication resulting from the dispersion of the system's different components. The reason for this is that every service in a microservice architecture is deployed independently and requires intercommunication with one another. This could lead to problems including latency, inconsistent data, and network errors. At their turn, failures may threaten the reliability of the system due to the unclear responsibility of multiple internal microservices for the fault, thus introducing challenges in monitoring and maintaining system performance.

Supported by Saint Petersburg State University, project ID: 95438429.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14821, pp. 343–357, 2024.
https://doi.org/10.1007/978-3-031-65308-7_24

The key difficulties concerning microservices performance monitoring are recognized and thoroughly covered in the literature. They include isolating the root cause of system faults [6,20,23], detecting anomalies [4,10,14], and predicting performance [13,25]. Performance is typically measured using a number of metrics (throughput, response time, error rate, resource usage, etc.) that hint at various potential system issues. For instance, throughput refers to the number of requests processed by a microservice within a given time frame (usually a second: QPS). It indicates the system's capacity to handle a certain load and is essential for determining the overall system efficiency. Resource utilization metrics, such as CPU utilization, memory usage, and network bandwidth, provide insights into how efficiently resources are being utilized by a microservice and help to optimize resource allocation. The error rate measures the frequency of errors encountered by a microservice during its operation. Monitoring error rates helps identify issues impacting service reliability and the user experience.

The dynamic behavior of microservices, with their distributed and interconnected nature, makes traditional monitoring approaches insufficient for handling the intricacies of these systems. Predicting performance metrics and fault propagation, detecting anomalies, and localizing their root causes require more complicated methodologies that can adapt to the constantly changing nature of microservice architecture.

In this paper, we focus on the tasks of performance metrics prediction and fault propagation analysis in microservice architectures, assuming that a call graph and dependencies between microservices in the system are unknown.

To address these challenges, we develop our approach at the intersection of causality identification and neural networks. By establishing causal relationships within the microservice architecture based on multimetric time series data and constructing graph neural networks based on the reconstructed call graph, we seek to proactively anticipate performance issues, swiftly identify anomalies, and predict their propagation, which may disrupt system functionality.

To sum up, our contribution includes several points. First, we propose a way to construct microservice impact graphs using multiple types of metrics to discover causal relationships between services. The algorithm is based on the stable Peter-Clark (PC) algorithm applied to the Uniform Manifold Approximation and Projection for Dimension Reduced (UMAP) multimetric time series data. Second, we propose to apply the diffusion convolutional recurrent neural network originally developed for road traffic forecasting in order to predict performance metrics (QPS and response time 95% quantile) for the given time horizon. Third, we investigate the proposed framework trained on anomaly-free data for the network fault propagation problem.

The rest of the paper is organized as follows: we summarize the related work in Sect. 2. Section 3 presents the description of the combined model. Section 4 introduces the details of the experiment and the experimental results, Sect. 5 explains the conclusions and future work.

## 2 Related Work

### 2.1 Time Series Causal Discovery

Time series causal discovery involves identifying causal relationships and dependencies between variables in time series data. There are several classes of methods that can be employed to uncover causal links and understand the temporal dynamics of systems. We briefly describe the most commonly used ones in the literature applied to time series data generated by microservices metrics.

*Granger Causality Test Based Methods.* The most widely used causality measure is Granger causality (GC) [7,16]. It can be considered one of the basic approaches to infer causal relations from time series data, and serves as foundation for many other methods. It assesses whether past values of one variable help predict the future values of another variable. One of its potentially useful modifications is variable-lag GC [2,16], that addresses the problem of a lag between a cause and an effect being considered at a fixed time point, rather then changing in time. Methods that use transfer entropy (generalized GC) and full conditional independence [18] are based on mutual information calculation and are rather computationally expensive and suffer from the curse of dimensionality.

*Causal Network Learning Algorithms.* There are two basic methods for learning a directed acyclic graph (DAG): score-based and constraint-based approaches. Constrained-based approaches, such as Peter-Clark (PC) [8,22] and Fast Causal Inference (FCI) [21], learn the causal graph's skeleton via conditional independence tests before orienting the edges according to established orientation constraints. Score-based approaches, such as Greedy Equivalence Search (GES) [9], assign scores to various causal graphs using a specified score function and then search the space of DAGs for the best one.

*Structural Equation Models.* These methods aim to construct a model based on linear and non-Gaussian assumptions and solve it to get graph [9]. Among structural equation-based methods, Linear Non-Gaussian Acyclic Model (LiNGAM) [19] can be considered to be a baseline approach, however it is linear, assumes that there are no unobserved confounders and may not converge to a correct solution if the initial state is badly chosen. To improve LiNGAM estimation, for example, Fast Adjacency Skewness (FASK) [17] method can be used.

### 2.2 Time Series Forecast

*Traditional Time Series Forecasting Methods.* Time series prediction models, like ARMA and ARIMA, use previous values of the variables in order to predict their future values. However, accurate forecasting can be challenging when there's insufficient information about the indicator or there are complicated nonlinear dependencies between the variables [11]. In microservices domains, the multitude of service indicators and their evolving interrelationships pose such problems, so more enhanced methods should be considered for metrics prediction.

*Time Series Forecasting Based on Machine Learning.* Machine learning-based enhanced time series forecasting methods mostly use gradient boosting machine algorithms such as XGBoost and LightGBM, as well as recurrent neural network models such as LSTM and GRU. These models perform well with tabular data, including time series data, and are capable of capturing nonlinear relationships and interactions between variables, as well as patterns in time series data.

*Spatio-Temporal Time Series Forecasting* Spatial-temporal time series forecasting is a class of methods that predict future variable values based on their position and interconnection with each other. It is made up of components that learn spatial and temporal dependencies, such as graph neural networks (GNNs) for spatial relationships and recurrent neural networks (RNNs) for temporal correlations. Thus, both temporal and spatial information is stored, eliminating the need for extra temporal networks and allowing for irregular time series, varying sample frequencies, or missing data without data imputation. Based on these characteristics, they are an ideal candidate for performance metric forecasting in microservice architectures, where interconnection is critical for accurate forecasting.

## 3 Model

We propose a multi-variable time series prediction framework based on stable PC algorithm and graph neural network. As shown in Fig. 1, the whole algorithm is divided into the following parts:

- UMAP based metrics reduction,
- Stable PC-based causality graph construction,
- Graph neural network time series prediction.

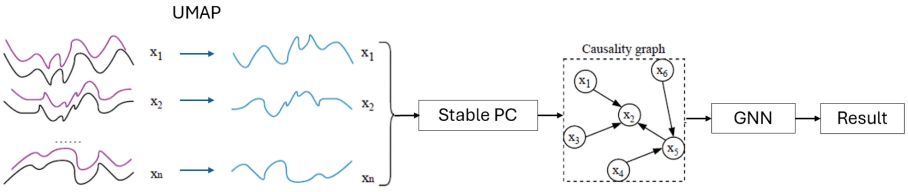


**Fig. 1.** The diagram of the proposed framework

The data set consists of $N \cdot m$ time series indicators collected for $N$ microservices $(x_1, ..., x_n)$ with $m$ performance metrics for each of them. Section 4 provides a full discussion of the data collection methods and tested description.

### 3.1 UMAP-Based Metrics Reduction

UMAP is a manifold learning technique for dimension reduction [1,15]. It consists of two phases: constructing a fuzzy topological representation and optimizing the low-dimensional representation to achieve the closest fuzzy topological representation as measured by cross entropy. UMAP is a k-neighbor based graph learning algorithm, with the first phase constructing a weighted k-neighbor graph and the second phase optimizing the low-dimensional representation to have the closest fuzzy topological representation. This method can create a low-dimensional representation of high-dimensional nonlinear data that retains meaningful structure and relationships, which is why it was selected for metrics reduction for the causal graph construction problem.

### 3.2 Stable PC Causal Graph Construction

Several algorithms were considered for causal graph construction phase: Peter Clark algorithm PC and its stable modification, LiNGAM, FCI algorithm, time-aware PC modifications (tigramite, time-aware PC) [3,18]. Stable PC algorithms [9] produce the best results in terms of the number of accurately detected connections and their directions. This is the main reason it was chosen for the framework's causality identification step.

The PC algorithm, a well-known constraint-based causal discovery method [8,22], is based on independent and identically distributed sampling with no latent confounder assumptions. It identifies conditional independencies using a search strategy and statistical methodologies. It starts with a complete undirected graph and iteratively reduces edges between conditionally independent variables to improve efficiency in sparse graphs. Edge orientations are found by detecting $V$-structures and using orientation propagation.

The PC-algorithm was designed for learning DAGs under the assumption of causal sufficiency, i.e., no unmeasured common causes and no selection variables. It learns a Markov equivalence class of DAGs that can be uniquely described by a so-called completed partially directed acyclic graph (CPDAG). PC algorithm starts by forming the complete undirected graph, then "thins" that graph by removing edges with zero order conditional independence relations, thins again with first order conditional independence relations, and so on. The set of variables conditioned on need only be a subset of the set of variables adjacent to one or the other of the variables conditioned.

The PC-algorithm is widely used in high-dimensional settings, since it is computationally feasible for sparse graphs with up to thousands of variables. PC-algorithm to be consistent for high-dimensional sparse graphs. Flexibility of utilizing a wide and growing class of conditional independence tests, from linear partial correlation (ParCorr) and non-parametric residual-based approaches to Kernel measures, tests based on conditional mutual information and neural networks. However, if there are unmeasured common causes or unmeasured selection variables then causal inference based on the PC-algorithm may be incorrect.

### 3.3 Graph Neural Networks

As we mentioned earlier, microservice systems operate in a dynamic environment that makes the problem of forecasting to require taking into account not only temporal factors in metrics data but also spatial ones related to the interconnected nature of microservices.

Moreover, the microservice metrics are influenced more by the downstream nodes than the upstream ones. This means that the spatial structure in metrics is nonEuclidean and directional.

In this research, we consider approaches based on graph neural networks applied to road traffic prediction problems, as in these tasks we encounter the same aspects as those related to microservice systems.

We investigated three recently proposed architectures: TGCN [27], STGCN [26], and DCRNN [12], which outperform the state-of-the-art approaches, and assessed their ability to make consistent predictions even when data contains faults. To the best of our knowledge, this is the first time these architectures have been used to solve a metric prediction problem in microservice architecture.

*TGCN: Temporal Graph Convolutional Network.* The authors integrate the graph convolutional network with the gated recurrent unit. The graph convolutional network captures the network's topological structure and models spatial dependence. The gated recurrent unit captures the dynamic changing of data to model temporal dependence.

*STGCN: Spatio-Temporal Graph Convolutional Networks.* To fully exploit spatial information, the authors use a whole graph representing the network structure rather than handling edges individually. For temporal dependencies prediction, they propose to use convolutional structure on the time axis rather than recurrent networks. Thus, this architecture is made up of many spatio-temporal convolutional blocks that combine graph convolutional layers and convolutional sequence learning layers to simulate spatial and temporal connections.

*DCRNN: Diffusion Convolutional Recurrent Neural Network.* The authors create a weighted directed network by calculating the pair-wise spatial correlations of node metrics. They describe the dynamics of interconnected node metrics as a diffusion process and suggest a diffusion convolution technique to capture spatial dependency. DCRNN combines diffusion convolution, sequence-to-sequence architecture, and scheduled sampling techniques.

## 4 Experiment

### 4.1 Experimental Setup

To locally simulate and run microservice system and collect metrics we used OpenTelemetry Demo[1]. It is an open source microservice project composed of

[1] https://opentelemetry.io/docs/demo/.

11 microservices which simulates the process of buying in on-line shop. The microservices in demo are written in different programming languages that communicate to each other over gRPC and HTTP; and a load generator which uses Locust[2] to fake user traffic. The microservice architecture is shown in Fig. 2.

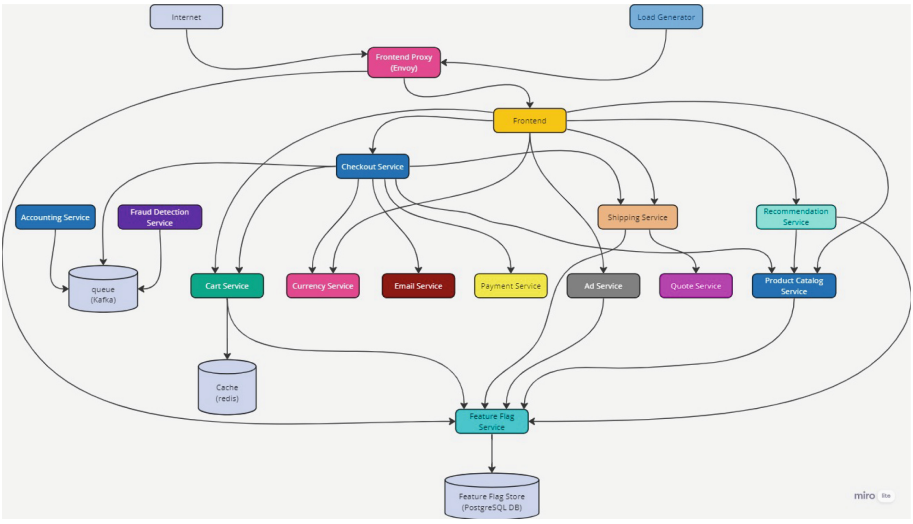


**Fig. 2.** Open telemetry demo microservice architecture

The traffic was injected using Locust framework. We have analyzed different shapes of users' activity on correctness of the microservice impact graph reconstruction: graduate increase of users number, graduate decrease, step-wise change of activity and '/\'-shaped profile (see Fig. 3). We had a hypothesis that stronger signal variations hesitate more microservices interconnections and it should influence the collected time-series metrics. Those ones in their turn should be better for revealing calls dependencies. '/\'-shaped profile (Fig. 3) demonstrated the best results with respect to all causality identification algorithms. That is the reason why it was used in all data acquisition scenarios.

We deploy OpenTelemetry demo application into local Kubernetes cluster with full support for a span metrics collection. These spans are processed by the Jaeger[3] microservice. Then a Prometheus microservice collects the span statics with a 15 s discreteness for QPS and response time (95% quantile) metrics. Moreover, the Prometheus microservice also collects the CPU and Memory usage statistics for each microservice directly from a Kubernetes with 60 s discreteness. To process raw collected metrics we used Grafana[4] tool.

[2] https://locust.io/.
[3] https://www.jaegertracing.io/.
[4] https://grafana.com/.

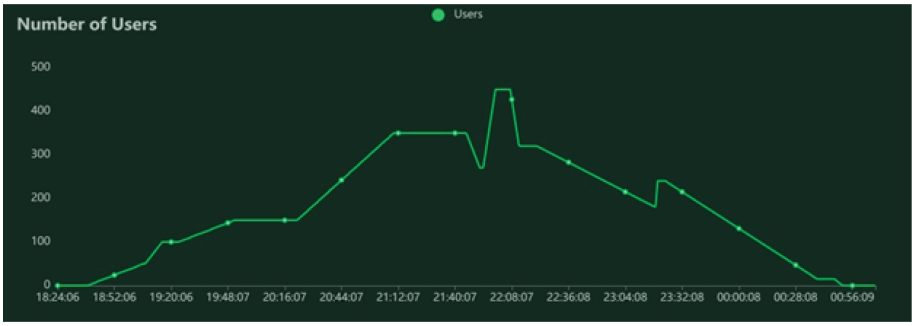


**Fig. 3.** Traffic injection in Open Telemetry

The profile depicted in the Fig. 3 was ran in the faultless system operation scenario. Besides it we simulated response latency faults in selected microservices in the system.

The errors were introduced into the cart, currency, and product catalog services independently and three data sets corresponding to each of the faults type were collected. Latency anomalies propagate from downstream to upstream, which is why these services were chosen for anomaly injection (see Fig. 2): they have the largest number of upstream connections, hence the number of services affected by the fault is higher.

### 4.2 Datasets

To assess the effectiveness of the suggested in Sect. 3 methodology, we applied the framework to a data set collected during operation of microservice system according to the described above scenario. We aimed to answer several research questions:

**R1** Can graph-based neural networks predict different metrics for each of the microservices in the system, and for what time horizon? Analyze different GNNs with different time prediction parts: RNN-based, CNN-based, and diffusion-CRNN-based (described in Sect. 3).

**R2** Can GNN predict faults in metrics?

**R3** GNN requires system architecture in the form of an adjacency matrix. Due to the dynamic nature of the microservices, the impact graph can change with time with respect to both the nodes and edge numbers, but we can identify it with causal discovery algorithms applied to the metrics time-series data. In this case, obviously, we are able to reconstruct the graph with a certain degree of accuracy. The question is how this inaccuracy affects the quality of the prediction.

To answer the questions above, we collected four datasets in total: 1) corresponds to the normal system operation under increasing load, followed by decreasing load (the '/\'-shaped profile shown in Fig. 3); 2) response latency is

injected to the cart service (after 1800 requests to empty a cart the next 200 requests take 3 s more to execute); 3) response latency is injected to the recommendation service (after 4500 requests the next 500 requests take 1 s more to execute); 4) response latency is injected to product catalog service (after 3700 requests the next 300 requests would take 1 s more to execute).

It should be noted that time discretness of the acquired metrics is different. In order, to analyze them jointly, we considered all of them with 60 s discreteness and reduced to a single time-series for each of the microservices with UMAP method.

Thus, the training data consist of the following: for the causality identification subtask, we used a composite time series with discreteness 60 s according to the reduction procedure described above. For the prediction subtask, we used first 80% of time-series data collected during the normal operation with the user profile shown in Fig. 3 for QPS and 95% quantile time response metrics (15 s discreteness). We used 20% of the remaining normal time series data for testing purposes, along with a fault injection to the cart service, recommendation service, and product catalog service.

### 4.3 Results

We made several experiments with the collected data.

1. The impact graph is used from the system architecture (see Fig. 2).
2. The impact graph is constructed based on the described in Sect. 3 algorithm and UMAP algorithm.
3. The neural network is trained on the faultless data (QPS and response duration) independently and was tested on both a normal part of data and data with anomalies.

Figure 4 depicts an impact graph created using a stable PC algorithm with reduced UMAP metrics (QPS, CPU utilization, memory consumption, and response duration). The edge statistics compared to the ground truth are as follows (Fig. 2): 11 false negative edges, 8 false positive edges, and 9 true positive edges were discovered.

The Tables 1 and 2 summarize the results obtained during the first two experiments for QPS and response time 95% quantile time series data collected with 15 s discreteness.

Table 1 compares averaged metrics (MAE: mean absolute error, RMSE: root mean squared error, MAPE: mean absolute percentage error, $R^2$: coefficient of determination) with respect to all microservices for QPS test data. Table 2 summarizes the same values but for response time 95% quantile z-score normalized test data. The metrics were computed for three GNN-based prediction models with the assumptions that the call graph is accurately known from the system architecture and that the call graph is reconstructed using causality identification algorithms (Fig. 4). We evaluated different prediction time horizons for both cases: 3, 6, and 12 data points ahead, which correspond to 45, 90, and 180 s time horizons.

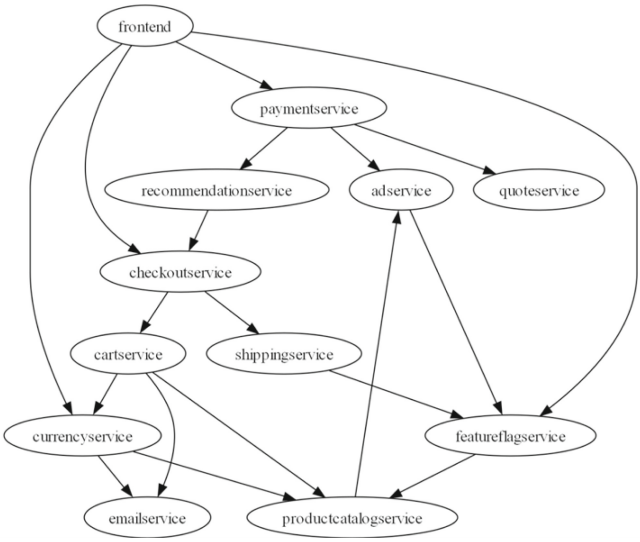


**Fig. 4.** Reconstructed microservices impact graph according to the model described in Sect. 3

Table 1 indicates clearly that, in both scenarios with accurate and inaccurate knowledge about the system architecture, the best results for QPS metric prediction are obtained using either TGCN or DCRNN model. However, according to the results in Table 2, TGCN failed for time response data set predictions, while DCRNN demonstrated the best metrics irrelevant to the preliminary knowledge about the system structure.

**Table 1.** Comparison of the results of QPS prediction using different graph neural networks architectures

| Adj. | **Model** | 45 sec horizon | | | | 90 sec horizon | | | | 180 sec horizon | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MAE | RMSE | MAPE | $R^2$ | MAE | RMSE | MAPE | $R^2$ | MAE | RMSE | MAPE | $R^2$ |
| True | **TGCN** | 30.51 | 75.79 | **43.15** | 0.93 | 28.94 | 66.98 | 50.4 | 0.95 | 32.10 | 75.28 | **45.27** | 0.93 |
| | STGCN | 39.17 | 83.35 | 61.91 | 0.92 | 45.21 | 98.98 | 66.56 | 0.89 | 37.44 | 83.69 | 49.83 | 0.92 |
| | **DCRNN** | **23.63** | **54.41** | 45.41 | **0.97** | **21.49** | **52.19** | **31.80** | **0.97** | **26.73** | **63.56** | 46.41 | **0.95** |
| Rec. | **TGCN** | 27.73 | 63.23 | **41.93** | 0.95 | **20.28** | **49.71** | **29.07** | **0.97** | 28.67 | 72.35 | **37.78** | 0.94 |
| | STGCN | 33.99 | 73.83 | 53.55 | 0.94 | 32.82 | 71.74 | 47.55 | 0.94 | 43.34 | 91.94 | 79.27 | 0.90 |
| | **DCRNN** | **26.78** | **58.76** | 54.05 | **0.96** | 27.93 | 60.41 | 57.71 | 0.96 | **27.53** | **60.71** | 54.4 | **0.96** |

Figure 5 shows the prediction results for the frontend microservice across a 180-second time horizon using GNN-based models. The vertical line refers to the data partition between the train and test sets. These plots justify the conclusion made according to the metrics analysis. The form of QPS and time response

**Table 2.** Comparison of the results of response time 95% quantile prediction using different graph neural networks architectures

| Adj. | Model | 45 sec horizon | | | | 90 sec horizon | | | | 180 sec horizon | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MAE | RMSE | MAPE | $R^2$ | MAE | RMSE | MAPE | $R^2$ | MAE | RMSE | MAPE | $R^2$ |
| True | TGCN | 0.99 | 3.13 | 8.92 | 0.94 | 1.10 | 3.60 | 8.90 | 0.93 | 1.49 | 4.51 | 9.06 | 0.88 |
| | STGCN | 2.19 | 6.16 | 10.9 | 0.78 | 2.19 | 6.28 | 11.44 | 0.78 | 2.24 | 6.52 | 11.51 | 0.76 |
| | **DCRNN** | **0.81** | **2.41** | **7.46** | **0.97** | **0.84** | **2.83** | **7.36** | **0.95** | **0.95** | **3.33** | **7.86** | **0.94** |
| Rec. | TGCN | 1.14 | 3.54 | 10.02 | 0.93 | 2.21 | 6.19 | 9.53 | 0.78 | 1.64 | 4.82 | 10.14 | 0.87 |
| | STGCN | 2.17 | 6.36 | 10.36 | 0.77 | 2.35 | 6.83 | 11.26 | 0.73 | 2.51 | 7.23 | 11.87 | 0.70 |
| | **DCRNN** | **0.67** | **2.31** | **6.39** | **0.97** | **0.83** | **2.81** | **7.45** | **0.95** | **0.96** | **3.32** | **8.04** | **0.94** |

profiles varies across microservices; we chose the frontend for illustration because it is the most representative.

The third experiment is related to the testing abilities of GNN-based models trained with faultless data to predict on data with latency anomalies in the response time and analyzing their ability to predict the metrics whose behavior is different a lot compared to the profiles during normal operation.

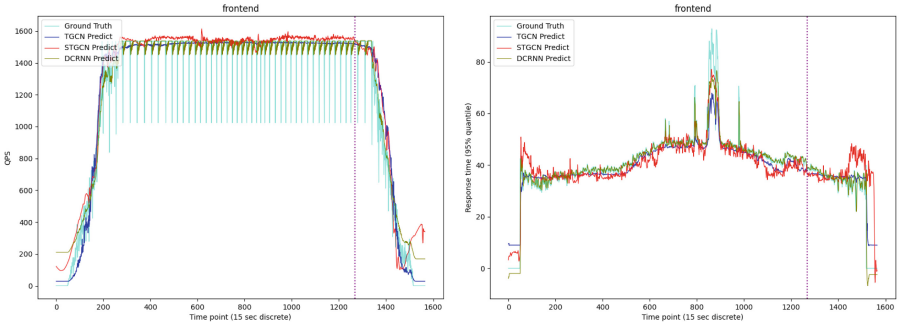


**Fig. 5.** Prediction for frontend over 180 s time horizon compared to the ground truth data (cyan color): left – QPS data, right – time response data. The vertical line represents the divide of data between training and testing. (Color figure online)

Figures 6 and 7 demonstrate the results of the third experiment when latency anomalies were injected into currency and product catalog service. Figure 6 depicts the RMSE values computed for all microservices QPS (left) and time response (right) test data using three evaluated GNN models with latency in the currency microservice. The color in Fig. 6 shows the spread of RMSE values within different prediction horizons (45–180 s); the straight line corresponds to the middle value (90 s). The error spread for the time response prediction is larger than for QPS, and it is natural if you take into account the time series profile shapes. The call graph was reconstructed according to the algorithm described in Sect. 3. If we compare the plots with the corresponding values from Table 1 and

2, we can conclude that DCRNN outperforms the other models with negligible degradation of error values. It is interesting because the model was trained on the data without anomalies but is still able to give good results on the faulty data.

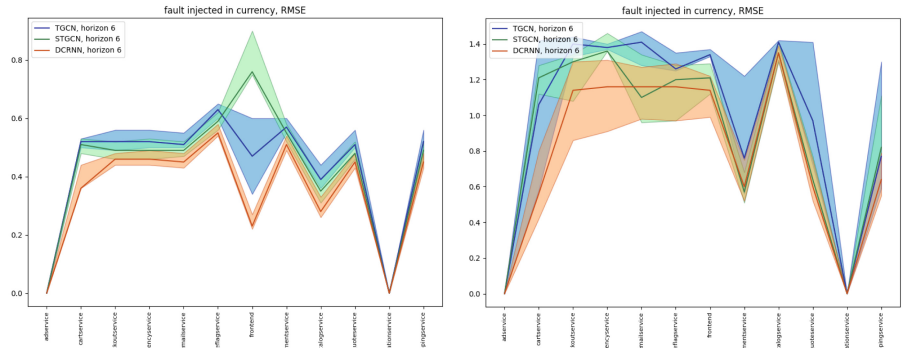


**Fig. 6.** RMSE values across the microservices for QPS (left) and time response (right) data with fault injected into the currency microservice. Prediction horizon is 90 s

Figure 7 compares prediction results obtained with the considered models for a randomly selected moment of time (not falling into the episode of fault occurrence) across all microservices with a 180-s prediction horizon. The charts show that the microservice where an anomaly was injected has high discrepancies between predicted and ground truth data, as do the ones that have an upstream link to the root cause microservice, for instance, frontend and checkout services (cf. Fig. 4). Thus, this observed behavior can additionally be helpful for a fault localization problem, by revealing possible root cause of the faults.

## 5 Conclusion

In this paper, we present our framework that integrates the stable PC method with GNN-based time series prediction model and a causality detection algorithm based on UMAP-reduced microservices metrics. As a result, it is feasible to construct links between microservices and represent them as an adjacency matrix that is fed into a GNN model. We assessed three cutting-edge systems originally developed for the road traffic prediction problem using an open telemetry demo application. Even in cases where the causation graph was reconstructed incorrectly, DCRNN produced the best results for all experiments that were examined. Including the cases when the model was trained with faultless data and tested on the data with the injected faults.

In the future, this approach will be applied to bigger data sets with an emphasis on anomaly detection and root cause investigation.

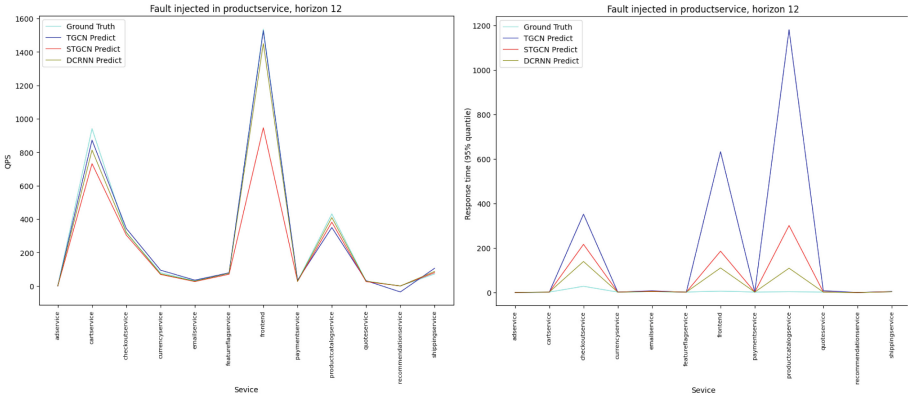


**Fig. 7.** Prediction for one selected moment of time across all the microservices. QPS (left), time response (right) data with fault injected into productcatalog service. Prediction horizon is 180 s

## References


1. Journal of Open Source Software: UMAP: Uniform Manifold Approximation and Projection. https://joss.theoj.org/papers/10.21105/joss.00861
2. Amornbunchornvej, C., Zheleva, E., Berger-Wolf, T.Y.: Variable-lag granger causality for time series analysis. In: 2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 21–30. IEEE, Washington, DC, USA, October 2019. https://doi.org/10.1109/DSAA.2019.00016
3. Biswas, R., Shlizerman, E.: Statistical perspective on functional and causal neural connectomics: a comparative study. Front. Syst. Neurosci. **16**, 817962 (2022). https://doi.org/10.3389/fnsys.2022.817962
4. Chang, Y.Y., Li, P., Sosic, R., Afifi, M.H., Schweighauser, M., Leskovec, J.: F-FADE: frequency factorization for anomaly detection in edge streams. In: Proceedings of the 14th ACM International Conference on Web Search and Data Mining, pp. 589–597. ACM, Virtual Event Israel, March 2021. https://doi.org/10.1145/3437963.3441806
5. Chen, L.: Microservices: architecting for continuous delivery and DevOps. In: 2018 IEEE International Conference on Software Architecture (ICSA), pp. 39–397. IEEE, Seattle, WA, April 2018. https://doi.org/10.1109/ICSA.2018.00013
6. Chen, P., Qi, Y., Zheng, P., Hou, D.: CauseInfer: automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In: IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, pp. 1887–1895. IEEE, Toronto, ON, Canada, April 2014. https://doi.org/10.1109/INFOCOM.2014.6848128
7. Cliff, O.M., Novelli, L., Fulcher, B.D., Shine, J.M., Lizier, J.T.: Assessing the significance of directed and multivariate measures of linear dependence between time series. Phys. Rev. Res. **3**(1), 013145 (2021). https://doi.org/10.1103/PhysRevResearch.3.013145
8. Colombo, D., Maathuis, M.H.: Order-independent constraint-based causal structure learning. J. Mach. Learn. Res. **15**(1), 3741–3782 (2014)

9. Glymour, C., Zhang, K., Spirtes, P.: Review of causal discovery methods based on graphical models. Front. Genetics **10**, June 2019. https://doi.org/10.3389/fgene.2019.00524
10. Jing, N., Li, H., Zhao, Z.: A microservice fault identification method based on LightGBM. In: 2022 IEEE 8th International Conference on Cloud Computing and Intelligent Systems (CCIS), pp. 709–713, November 2022. https://doi.org/10.1109/CCIS57298.2022.10016384
11. Knyazev, N., Pershin, A., Golovkina, A., Kozynchenko, V.: Forecasting the state of complex network systems using machine learning methods. Cybern. Phys. **12**(2), 129–135 (2023). https://doi.org/10.35470/2226-4116-2023-12-2-129-135
12. Li, Y., Yu, R., Shahabi, C., Liu, Y.: Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting, February 2018. https://doi.org/10.48550/arXiv.1707.01926
13. Liu, H., Zhang, W., Zhang, X., Cao, Z., Tian, R.: Context-aware and qos prediction-based cross-domain microservice instance discovery. In: 2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS), pp. 30–34, October 2022. https://doi.org/10.1109/ICSESS54813.2022.9930241
14. Liu, P., 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), pp. 48–58. IEEE, Coimbra, Portugal, October 2020. https://doi.org/10.1109/ISSRE5003.2020.00014
15. McInnes, L., Healy, J., Melville, J.: UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, September 2020. https://doi.org/10.48550/arXiv.1802.03426
16. Moraffah, R., et al.: Causal inference for time series analysis: problems, methods and evaluation. Knowl. Inf. Syst. **63**(12), 3041–3085 (2021). https://doi.org/10.1007/s10115-021-01621-0
17. Ramsey, J., Andrews, B.: FASK with Interventional Knowledge Recovers Edges from the Sachs Model, May 2018. https://doi.org/10.48550/arXiv.1805.03108
18. Runge, J.: Causal network reconstruction from time series: From theoretical assumptions to practical estimation. Chaos: An Interdisciplinary J. Nonlinear Sci. **28**(7), 075310 (2018). https://doi.org/10.1063/1.5025050
19. Shimizu, S., Hoyer, P.O., Hyvärinen, A., Kerminen, A.: A linear non-gaussian acyclic model for causal discovery. J. Mach. Learn. Res. **7**, 2003–2030 (2006)
20. Soldani, J., Brogi, A.: Anomaly Detection and Failure Root Cause Analysis in. Micro) Service-Based Cloud Applications, A Survey (2021)
21. Spirtes, P.: An anytime algorithm for causal inference. In: International Workshop on Artificial Intelligence and Statistics, pp. 278–285. PMLR, January 2001
22. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search, Lecture Notes in Statistics, vol. 81. Springer, New York (1993). https://doi.org/10.1007/978-1-4612-2748-9
23. Wang, P., et al.: CloudRanger: root cause identification for cloud native systems. In: 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 492–502. IEEE, Washington, DC, USA, May 2018. https://doi.org/10.1109/CCGRID.2018.00076
24. Wolff, E.: Microservices: flexible software architecture. Addison-Wesley, Boston (2017)
25. Yang, Y., Jiang, Y.: Microservice indicator prediction method based on STE and CNN-BiLSTM. In: 2023 IEEE 9th International Conference on Cloud Computing and Intelligent Systems (CCIS), pp. 511–515, August 2023. https://doi.org/10.1109/CCIS59572.2023.10262956

26. Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, pp. 3634–3640, July 2018. https://doi.org/10.24963/ijcai.2018/505, arXiv:1709.04875 [cs, stat]
27. Zhao, L., et al.: T-GCN: a temporal graph convolutional network for traffic prediction. IEEE Trans. Intell. Transp. Syst. **21**(9), 3848–3858 (2020). https://doi.org/10.1109/TITS.2019.2935152