Towards Automated Monitor Configuration for Cloud Services: A Data-Driven AIOps Framework Informed by Industrial Practice

Anson Bastos
Microsoft
ansonbastos@microsoft.com

Anjaly Parayil
Microsoft
aparayil@microsoft.com

Ayush Choure
Microsoft
aychoure@microsoft.com

Chetan Bansal
Microsoft
chetanb@microsoft.com

Rujia Wang
Microsoft
rujiawang@microsoft.com

ABSTRACT

Reliability of large-scale cloud services is critical for user satisfaction and business continuity. Despite significant investments in reliability engineering, production incidents remain inevitable, often leading to customer impact and operational overhead. In large cloud companies, multiple services are deployed across regions necessitating robust health monitoring systems. However, the current monitor configuration process is manual, largely reactive and ad hoc, resulting in gaps in coverage and redundant alerts. In this paper, we present a comprehensive study of monitor creation in Microsoft, identifying key components in the existing process. We further design a modular recommendation framework that processes the graph structured service entities to suggest optimal monitor configurations. Through extensive experimentation on historical data and user study of recommendations for production services at Microsoft, we demonstrate the efficacy of our approach in providing relevant recommendations for monitor configurations. The code is is available on this link.

1 INTRODUCTION

Large scale cloud service providers such as Amazon, Microsoft, Google etc. run thousands of services over complex environments. At Microsoft, the hyper-scale cloud infrastructure supports over 5,000 services deployed across more than 60 regions, serving hundreds of millions of users globally. Ensuring the continuous availability of these services is critical to sustaining customer satisfaction and preserving business revenue [18]. Despite extensive investments in reliability engineering, production incidents and failures remain unavoidable, often resulting in customer impact, financial losses and requiring substantial engineering effort for detection, diagnosis and mitigation. Consequently, early detection and resolution of such incidents are vital to minimizing user disruption and reducing operational costs. To this end, service providers engage in continuous health monitoring of services, aiming to proactively identify and address issues before they affect end users.

The lifecycle of an incident generally begins with detection, followed by triaging to the concerned stakeholders for resolution who then diagnose the incident and apply some mitigation so that the service becomes functional. This is generally followed by a detailed root causing of the incident and working towards a longer term solution for example by making necessary code changes. Incidents could either be detected by automated watchdogs called *monitors* or reported by customers. The latter is undesirable as it results in loss of revenue and affects the reputation of the company. Further manual incident reporting delays detection which increases service downtime. In this paper we focus on the detection phase of an incident's lifecycle, specifically by monitors.

The current methodology for configuring monitors primarily follows a trial-and-error paradigm. Service owners define monitors based on their understanding of the service architecture and service-level objectives but do not consider the intricate relation between existing monitor configurations and the past incident detection performance. Additionally, monitors are often revised or newly introduced in response to production incidents. However, this approach is inherently reactive, which means critical monitors may be absent until an incident reveals the gap. Second, it frequently leads to the creation of redundant monitors, resulting in excessive alert noise and unnecessary operational overhead.

Related Works: In recent years, numerous empirical studies have explored the challenges associated with monitoring and incident resolution in cloud services [2, 3, 6–9, 12, 16, 17, 20, 25], as well as the practical difficulties in defining and adhering to Service Level Objectives (SLOs) [4, 5, 13]. These prior works generally assume that the appropriate monitors are already identified or only study an aspect on the monitor configuration problem.

This work addresses a critical and underexplored problem in Cloud Intelligence / AIOps: the *automated configuration*

of service monitors in large-scale cloud systems. We identify the following key AIOps challenges addressed by our work: 1) **Monitor Configuration Complexity**: Service owners must manually define monitors without systematic guidance, often leading to gaps in coverage, redundant alerts, and delayed incident detection. This problem is exacerbated by the scale and heterogeneity of cloud services. 2) **Lack of Intelligent Support for Monitor Design**: Existing tools offer limited support for recommending configurations based on historical data, service context, or domain-specific patterns. There is a need for an *intelligent, data-driven framework* that can assist engineers in *configuring effective monitors*. We make the following **Contributions:** 1) We present an empirical study (of ~ 60k monitors) and analysis of the individual components of the monitor creation pipeline. 2) We build models to recommend component-wise configurations. 3) We unify the results of individual modules into a single framework for service owners to create monitors. 4) We conduct extensive evaluation and production user study of our method to understand the performance efficacy.

2 PROBLEM SETTING

We define the below terms in the context of micro-services:

A *metric* is defined as a time-series entity that is generated through the utilization of a resource and is therefore inherently linked to that resource. These metrics could emitted across various attributes called *dimensions*, which can be thought of as the granularity along which the data has to be aggregated. *Alerting logic* refers to a set of anomaly detection rules that operate over metrics and serve as triggers for alert creation. *Expressions* are mathematical forms that are tied to certain metrics as defined by the user. A *monitor* is formally defined as a collection of tuples, each comprising a metric, the dimensions to aggregate across for each metric and an associated alerting logic with corresponding expressions. We now define the graph network formed by the interconnection of these components.

Definition 1. *(Monitor Entity Graph): We represent the data as a heterogeneous graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *where* $\mathcal{V} = \{\mathcal{V}_s \cup \mathcal{V}_m \cup \mathcal{V}_d \cup \mathcal{V}_k \cup \mathcal{V}_{exp}\}$ *represents the set of nodes with* $\mathcal{V}_s, \mathcal{V}_m, \mathcal{V}_d, \mathcal{V}_k, \mathcal{V}_{exp}$ *denoting services, monitors, dimensions, metrics and expressions respectively and* $\mathcal{E} = \{\mathcal{E}_{sm} \cup \mathcal{E}_{md} \cup \mathcal{E}_{kd} \cup \mathcal{E}_{mk} \cup \mathcal{E}_{mexp} \cup \mathcal{E}_{kexp} \cup \mathcal{E}_{dexp}\}$ *represents the set of edges, capturing following types of relationships: 1)* $\mathcal{E}_{sm}$*: "service creates monitor", 2)* $\mathcal{E}_{md}$*: "monitor associated with dimension", 3)* $\mathcal{E}_{kd}$*: "metric has dimension", 4)* $\mathcal{E}_{mk}$*: "monitor emits metric" 5)* $\mathcal{E}_{mexp}$*: "monitor evaluates expression". 6)* $\mathcal{E}_{kexp}$*: "metric used in expression" 7)* $\mathcal{E}_{dexp}$*: "dimension used in expression"*

Figure 1 gives a broad overview of the monitor entity graph along with the service health monitoring system. Since we deal with heterogeneous graph structured data, in order

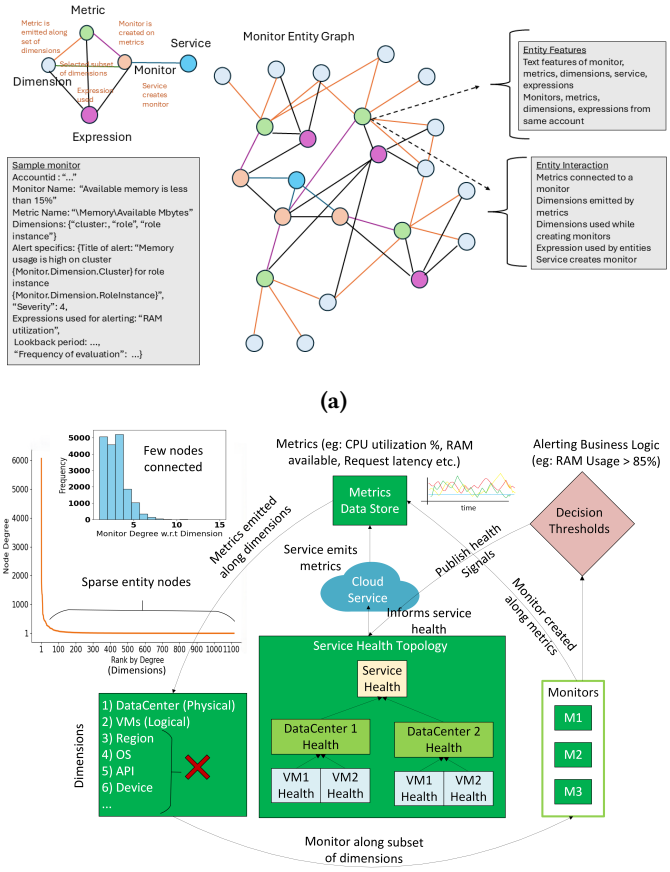


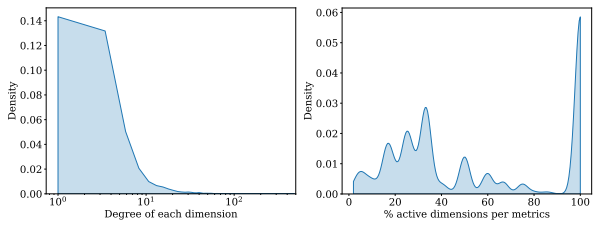
Figure 1: a) Monitor entity graph: Nodes represent the monitors, metrics, and dimensions in a cloud setting. Each node contains text features and interacts with their neighboring nodes b) The cloud service emits metrics along many dimensions and only few are being used for monitoring health. For eg. in this case the service health depends on health of each datacenter using the service which in turn depends on the health of the individual VMs in the datacenter.

to recommend the entities we use a graph based framework. We pose the recommendation as an entity ranking problem.

3 EMPIRICAL STUDY

In this section, we study the different aspects of the monitor entity graph and discuss the findings. The graph is constructed using the service data from ~ 3600 accounts of Microsoft consisting of ~ 60k historical monitors, ~ 14K metrics, ~ 7K dimensions, ~ 27K expressions and ~ 2M alerting conditions. Based on the analysis and derived observations in this section, we formulate the monitor configuration recommendation framework.

Sparsity in the nodes / entities in the monitor entity graph: Figure 2 illustrates the sparsity characteristics of the monitor entity graph. From Figure 2a we can see that, the majority of monitors (94%) do not need to aggregate the metrics along all dimensions along which they are emitted. This necessitates intelligently selecting the subset of dimensions to monitor. Thus the problem of selecting an optimal subset of dimensions on which to monitor is relevant.



(a) (b)

Figure 2: Characteristics of the Monitor Entity Graph

Observation 1: The "monitor entity" graph exhibits activity sparsity. Although many dimensions are associated with metrics, only a subset of them is used to aggregate the metric.

Mathematical forms used in the expressions: In this study we shall explore the categories of expressions used in the alerting logic. These expressions assume some mathematical forms such as summing, averaging and so on over the metrics' timeseries data. This study informs us what are the major types of expressions that are being evaluated by existing monitors and how to select the expression for a new monitor. From Figure 3, we observe the following major types of expressions in current use by the monitors: Count, Sum, Average, Percentile, Rate, QoS, Max, Min. These cover more than 95% of the expressions.

Observation 2: As most of the expressions use similar mathematical forms as their counterpart monitors, we treat the problem of recommending expressions as finding similar expressions from existing monitors after learning over the monitor entity graph.

After deducing the expression form, the actual evaluation would be performed over the monitor's metrics.

Monitoring status and alerts from timeseries: We study whether time-series features alone can predict (i) whether a metric should be monitored and (ii) its alert threshold, using statistical descriptors with a Random Forest classifier and a linear regression model, respectively. The results in Figure 4 show that time-series features predict monitoring necessity with high precision and statistically significantly outperform textual features ($p < 0.05$), with frequency-based

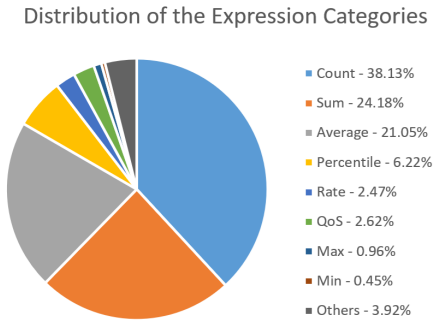
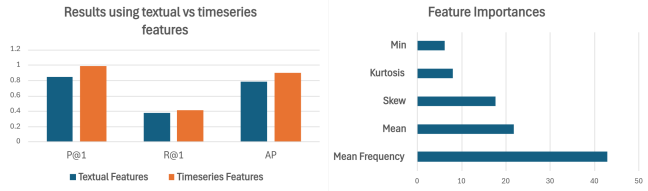


Figure 3: Most expressions (83%) use either of count, sum or average as the mathematical form to aggregate the metrics data.

features being the most informative. In contrast, predicting alert thresholds yields poor performance ($R^2 < 0.1$), likely because the analysed time ranges lack anomalies or raw values do not directly encode business rules.

Observation 3: The timeseries data can help to select the metrics for monitoring with high precision but is not a good indicator of the alert thresholds. Thus the timeseries data can be used to select the metrics but not to decide the alert thresholds.



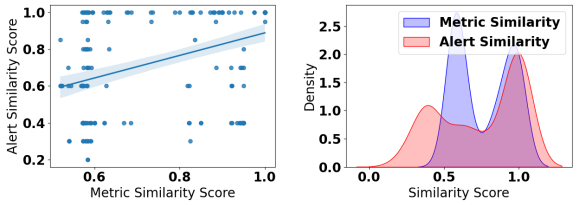
(a) (b)

Figure 4: Analysis of the importance of time series features on predicting the metric monitoring status.

Figure 5 analyses the relationship between metric feature similarity and alert condition similarity by computing correlations between the two. Using a combination of textual and time-series–based metric similarity and LLM-evaluated alert condition similarity, we observe a moderate positive correlation of 0.41, indicating that metrics with similar features tend to have similar alert conditions.

Observation 4: We find a high correlation between the similarities of the metrics and alerts conditions indicating that finding similar metrics can help with alerting conditions.

These findings motivate the alert generation approach in §4.



(a) (b)

Figure 5: Study showing the relation between alert conditions and the feature based similarity.

4 A UNIFIED FRAMEWORK FOR MONITOR CONFIGURATION

In this section, we briefly describe the individual modules composing the recommendation system and see how we unify the individual recommendations into a single coherent interface providing feedback to the system. The entire framework for the automated monitor configuration recommendation system is provided in figure 6. Figure 7 depicts an overview of the recommendation modules. We look at each module in detail below.

Selecting subset of Metrics for monitoring: We design the metric selection module as a weakly supervised classification model that jointly captures global and local monitoring dynamics. Given service metadata (name, description, dependencies), available metrics, and associated dimensions, we encode all textual features using sentence embeddings (e.g., E5 [19]) and aggregate variable-sized sets via mean pooling in a DeepSets-style framework [23]. The resulting set embeddings are concatenated and passed through a two-layer MLP with ReLU activation [15], followed by a binary classification head that predicts whether a metric should be monitored, optimized using the binary cross-entropy loss. While this global objective assumes a shared decision boundary across services, such an assumption may not hold in practice; therefore, we additionally model service-specific local dynamics by enforcing neighbourhood consistency in the latent embedding space. Specifically, we perform k-nearest neighbour voting at inference time and introduce a contrastive loss $\mathcal{L}_{contrast}(e_k) = \sum_{k^+,k^-} \left(\|e_k^+ - e_k\| - \|e_k^- - e_k\| \right) + [\gamma + \|e_{cent}^+ - e_{cent}^-\|]_+$. In this equation, the loss is computed for the given metric embedding (e_k), k^+ denotes the positive metrics (i.e. of similar monitoring status) and k^- are the negative (dissimilar monitoring status) pairs with e_k, e_{cent} represents the euclidean centroid of a set of embeddings, γ is a positive margin (set to 1 empirically) and [.] denotes the positive value of the term (else 0). The loss pulls embeddings of metrics with similar monitoring status closer while separating dissimilar ones within a margin. To further support downstream alert recommendation, we retrieve metrics similar to a target metric by combining ontology-based textual similarity (computed via embedding proximity) and LLM-based semantic scoring (with time-series similarity derived from shapelet extraction and comparison [11, 14, 22]), and use the alert conditions of these similar metrics to inform final alert construction.

Graph based Recommendation: Once we select which metrics to monitor, we next recommend (i) the dimensions along which each metric should be emitted/aggregated and (ii) the expressions used to evaluate those metric streams, by learning over the monitor entity graph $\mathcal{G}$ (Section 2), which encodes both entity attributes and their relational context. Inspired by [10], our graph module uses a heterogeneous, attention-based message passing mechanism: multi-head attention computes neighbour importance via dot-product query–key scores, edge-aware transformations propagate weighted value messages, and node states are updated by concatenating the current representation with the aggregated message and applying a learned linear map with ReLU. To capture longer-range/global dependencies beyond local homophily [1], we additionally incorporate meta-path style random walks [21, 24] from target nodes. For optimization, we employ a composite objective that combines binary cross-entropy for prediction accuracy with TOP1-max ranking loss to improve top-k recommendation ordering, similar to [10].

Generation of Alert Conditions We integrate the outputs of the metric, dimension, expression, and similar-metric modules into a single coherent alert condition for monitor creation, noting that reliable time-series data and thresholds are often unavailable or uninformative at early stages. Consequently, we employ an LLM to infer baseline alert conditions from semantically similar monitors and documented best practices, and to consolidate the recommended components into the monitor-specific alert format, as illustrated in Fig. 8.

Computational complexity: The overall complexity is linear in the monitor entity graph size.

5 RESULTS

In this section, we evaluate the effectiveness of the unified framework using real-world datasets collected from Microsoft, a large-scale cloud system provider. In particular, we aim to answer the following RQs: 1) RQ1: What is the effectiveness of the individual modules? 2) RQ2: How effective is the proposed unified framework compared to baselines, for monitor configuration recommendation? 3) RQ3: What is the importance of each module in the framework? 4) RQ4: Are the generated alerts from the system, meaningful for services in production?

Results of individual modules (RQ1): 1) Metric Selection: Table 1 shows the results on the metric selection problem. We report similar metrics as [17] (the only baseline

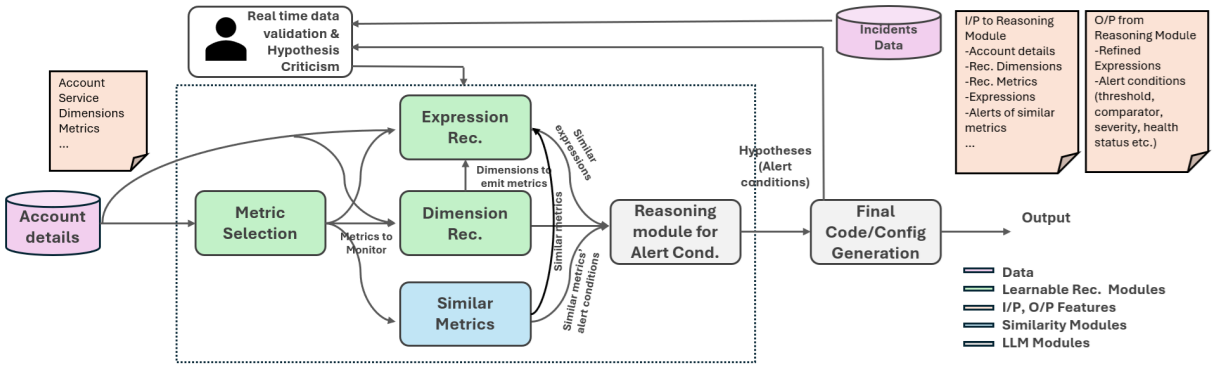
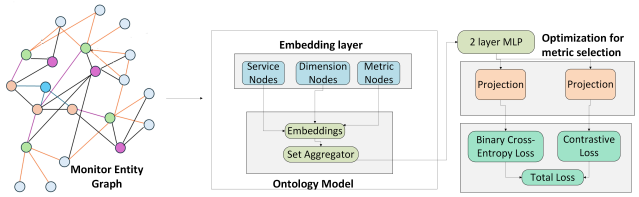
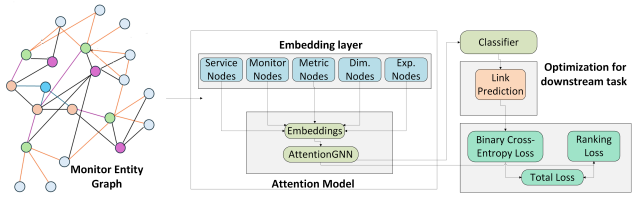


Figure 6: The framework takes service or account metadata together with emitted metrics and their associated dimensions as input, and sequentially applies specialised modules to recommend which metrics to monitor, the most informative dimensions, and appropriate expressions, while leveraging historical metric similarity to enrich these recommendations. A reasoning module based on an LLM then integrates all intermediate outputs to generate final alert conditions and executable monitor configuration code.



(a)



(b)

Figure 7: The overall architecture of the recommendation modules. Figure a shows the metric recommendation module which uses a 2 layer MLP over the set aggregated service features. Figure b shows the architecture used for the dimension/expression recommendation modules. The framework applies an enhanced transformer-based graph convolution.

for metric selection available) for fair comparison namely: accuracy, precision, recall, f1-score (micro), macro f1-score and the hamming loss. We can see that our learning based approach outperforms the SVD approach of [17] ($p<0.05$, Wilcoxon signed rank test). Moreover the baseline is computationally expensive requiring cubic complexity to compute the SVD. However our method requires no such preprocessing and only involves a forward pass through the network over the metrics requiring linear complexity.

> You are an expert service engineer. Your task is to design the configuration of the alert expressions and thresholds for the given service...
> You will be given the service details such as the name of the service, the metric (time-series) name, the time series values (if available)...
> You will also be given similar alert expressions, thresholds and best practices from similar metrics.
> Glossary of the operators and definitions with examples: ...
> Given service information: {Account:..., Metrics:..., Dimensions:..., Sampling Types:..., Raw Timeseries:..., Percentile data:..., best practices:...}
> Below are the alert conditions of similar metrics: ...
> Please generate the alert conditions in the following format: ...

Figure 8: The structure of the LLM prompt used to obtain the final alert configurations from the recommended values.

Table 1: Metric selection results wrt the baseline [17]. We outperform the baseline while having lower computational complexity ($O(N)$ vs $O(N^3)$, N=No. of metrics).

Metric	Acc.(↑)	Prec.(↑)	Rec.(↑)	F1-score(↑)	Macro-F1(↑)	Hamming(↓)
SVD	0.841	0.754	0.840	0.795	0.431	0.159
Ours	0.866	0.773	0.863	0.816	0.453	0.134

2) Dimension/Expression Recommendation: We evaluate dimension and expression recommendation using standard ranking metrics (HR@k, MRR, NDCG@k, Recall@k) against collaborative filtering, text-only MLPs, and a range of homogeneous and heterogeneous GNN baselines. Results (2) show that text features already outperform collaborative filtering, graph-based models further improve performance,

and our attention-based model with ranking loss and induced metapaths achieves the best results, with statistically significant gains ($p < 0.05$), trends consistent for expression recommendation. Thus overall the metric, dimension and expression recommendation modules are able to outperform the baseline and provide competitive results answering RQ1.

Table 2: Our method outperforms baseline graph and non-graph methods. Basline results show only dimension rec (we see similar results for expression rec).

Metric	HR@1	HR@3	HR@5	MRR	N@k	R@1	R@3	R@5
CF	0.230	0.149	0.116	0.391	0.217	0.117	0.310	0.438
MLP	0.312	0.186	0.128	0.464	0.307	0.184	0.392	0.492
SAGE(v1)	0.383	0.186	0.127	0.499	0.328	0.218	0.379	0.474
SAGE(v2)	0.291	0.154	0.111	0.414	0.262	0.165	0.323	0.398
GAT	0.355	0.185	0.18	0.487	0.323	0.213	0.397	0.493
HGT	0.396	0.185	0.131	0.510	0.356	0.228	0.402	0.507
HAN	0.348	0.194	0.131	0.485	0.315	0.202	0.407	0.500
HetGNN	0.375	0.173	0.121	0.492	0.321	0.224	0.400	0.502
T (dim. rec.)	0.331	0.188	0.134	0.481	0.306	0.178	0.399	0.523
T (exp. rec.)	0.241	0.386	0.466	0.349	0.246	0.188	0.292	0.362
T + RL (dim. rec.)	0.573	0.246	0.159	0.672	0.525	0.342	0.592	0.675
+ Paths (dim. rec.)	**0.597**	**0.265**	**0.173**	**0.714**	**0.555**	**0.355**	**0.649**	**0.748**
+ Paths (exp. rec.)	**0.535**	**0.701**	**0.750**	**0.631**	**0.519**	**0.380**	**0.596**	**0.662**

Results of Unified Framework (RQ2, RQ3): We evaluate the end-to-end framework that integrates metric, dimension, expression, and alert-condition generation modules, comparing three metric selection variants (BCE, contrastive/KNN, and their ensemble) and incrementally adding dimension and expression recommendation to the pipeline. Using Jaccard similarity, precision, recall, and LLM-based alert quality scoring, we find that the ensemble metric method performs best and that adding dimension and expression modules consistently improves both intermediate and final alert quality, with all gains statistically significant ($p < 0.05$), demonstrating the necessity of each module (RQ2, RQ3).

> You are evaluating configs for a cloud monitoring system.
> Account Context: {Service: ..., Metrics: ..., ...} Alert Conditions to evaluate: ... Actual Alert Conditions Set: ... 1st percentile of the metrics' timeseries data: ... 99th percentile of the metrics' timeseries data: ...
> Evaluation Criteria: Rate each criteria from [0,1]:
> 1. Threshold Appropriateness: ... 2. Condition Validity: ... 3. Detecting Customer Incidents: ... 4. Noise reduction: ... 5. Specificity: ... 6. Completeness of alerts: ... 7. Checks for required fields: ...
> Response Format: {JSON response format}

Figure 9: LLM prompt used to evaluate the expected and generated alert conditions.

Human Evaluation and User Study (RQ4): We validate the LLM-based alert evaluation with human judgement by asking three SMEs to score ~100 predicted alert conditions on a $[0, 1]$ scale, observing a Cohen's $\kappa = 0.62$ agreement between LLM scores and the human majority vote. We further find a strong, statistically significant correlation of 0.673 ($p < 10^{-4}$) between LLM and average human scores, demonstrating that the LLM evaluation reliably reflects human assessment at scale, answering RQ4.

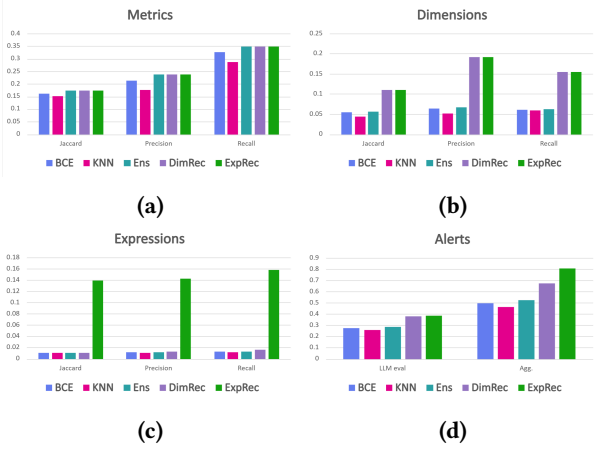


Figure 10: Results of the Unified Framework. The colors represent the model configuration used for system evaluation. For metric selection (a), we observe best performance by the ensemble (Ens) model. Further graph based recommendation models (DimRec:Ens+DimRec, ExpRec:Ens+Dimrec+ExpRec) improve the individual and overall results (b,c,d).

User Study and lessons learnt for production services: We conducted a user study with approximately 30 service owners to understand challenges in monitor creation and assess the usefulness of our recommendations, finding that dimension selection and threshold setting (typically updated on a weekly cadence) are the primary pain points. Users reported that recommendations are most actionable when accompanied by explanations grounded in similar services, emphasized on the UX and agreed on the need for end-to-end automation. The users rated the overall relevance of the recommendations highly (average 4.5/5).

6 CONCLUSION

In this paper, we propose a holistic and end-to-end framework for recommending monitor configurations in a large scale cloud environment. We study the monitoring process for production services in Microsoft and design recommendation systems to obtain predictions for individual components. We further build a system combining the individual components, to provide coherent configuration recommendations. The proposed evaluation framework shows that the method performs competitively in recommending monitor configurations corroborated by human evaluation. We hope this work fosters future research in intelligent monitoring and incident detection of cloud services.

REFERENCES

[1] Anson Bastos, Abhishek Nadgeri, Kuldeep Singh, Hiroki Kanezashi, Toyotaro Suzumura, and Isaiah Onando Mulang'. 2022. How Expressive are Transformers in Spectral Domain for Graphs? *Transactions on Machine Learning Research* (2022). https://openreview.net/forum?id=aRsLetumx1

[2] Ayush Bhardwaj, Zhenyu Zhou, and Theophilus A. Benson. 2021. A Comprehensive Study of Bugs in Software Defined Networks. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 101–115. doi:10.1109/DSN48987.2021.00026

[3] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, Yingnong Dang, Feng Gao, Pu Zhao, Bo Qiao, Qingwei Lin, Dongmei Zhang, and Michael R. Lyu. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1487–1497. doi:10.1145/3368089.3417055

[4] Jianru Ding, Ruiqi Cao, Indrajeet Saravanan, Nathaniel Morris, and Christopher Stewart. 2019. Characterizing Service Level Objectives for Cloud Services: Realities and Myths. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*. 200–206. doi:10.1109/ICAC.2019.00032

[5] Jianru Ding, Ruiqi Cao, Indrajeet Saravanan, Nathaniel Morris, and Christopher Stewart. 2019. Characterizing Service Level Objectives for Cloud Services: Realities and Myths. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*. 200–206. doi:10.1109/ICAC.2019.00032

[6] Yu Gao, Wensheng Dou, Feng Qin, Chushu Gao, Dong Wang, Jun Wei, Ruirui Huang, Li Zhou, and Yongming Wu. 2018. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 539–550. doi:10.1145/3236024.3236030

[7] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 126–141. doi:10.1145/3542929.3563482

[8] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. 2014. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SOCC '14)*. Association for Computing Machinery, New York, NY, USA, 1–14. doi:10.1145/2670979.2670986

[9] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. 2016. Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/2987550.2987583

[10] Fiza Hussain, Anson Bastos, A. Parayil, Ayush Choure, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. 2025. Attention Enhanced Entity Recommendation for Intelligent Monitoring in Cloud Systems. https://www.microsoft.com/en-us/research/publication/attention-enhanced-entity-recommendation-for-intelligent-monitoring-in-cloud-systems/

[11] Eamonn J. Keogh and Thanawin Rakthanmanon. 2013. Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets. In *Proceedings of the 13th SIAM International Conference on Data Mining, May 2-4, 2013. Austin, Texas, USA*. SIAM, 668–676. doi:10.1137/1.9781611972832.74

[12] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents?. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19)*. Association for Computing Machinery, New York, NY, USA, 155–162. doi:10.1145/3317550.3321438

[13] Jeffrey C. Mogul and John Wilkes. 2019. Nines are Not Enough: Meaningful Metrics for Clouds. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19)*. Association for Computing Machinery, New York, NY, USA, 136–141. doi:10.1145/3317550.3321432

[14] John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1855–1870. doi:10.1145/2723372.2737793

[15] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (01 Oct 1986), 533–536. doi:10.1038/323533a0

[16] Akanksha Singal, Divya Pathak, Kaustabha Ray, Felix George, Mudit Verma, and Pratibha Moogi. 2025. Metric Criticality Identification for Cloud Microservices. arXiv:2501.03547 [cs.DC] https://arxiv.org/abs/2501.03547

[17] Pooja Srinivas, Fiza Husain, Anjaly Parayil, Ayush Choure, Chetan Bansal, and Saravan Rajmohan. 2024. Intelligent Monitoring Framework for Cloud Services: A Data-Driven Approach. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (Lisbon, Portugal) *(ICSE-SEIP '24)*. Association for Computing Machinery, New York, NY, USA, 381–391. doi:10.1145/3639477.3639753

[18] Chellammal Surianarayanan, Pethuru Raj Chelliah, and Pethuru Raj Chelliah. 2019. *2019*. Cloud Monitoring. Essentials of Cloud Computing: A Holistic Perspective (2019). 241–254 pages.

[19] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533* (2022).

[20] Zibo Wang, Pinghe Li, Chieh-Jan Mike Liang, Feng Wu, and Francis Y. Yan. 2024. Autothrottle: a practical bi-level approach to resource management for SLO-targeted microservices. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'24)*. USENIX Association, USA, Article 9, 17 pages.

[21] Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. 2023. Simple and Efficient Heterogeneous Graph Neural Network. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 9 (Jun. 2023), 10816–10824. doi:10.1609/aaai.v37i9.26283

[22] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. 2024. MonitorAssistant: Simplifying Cloud Service Monitoring via Large Language Models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 38–49. doi:10.1145/3663529.3663826

[23] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017.

Deep Sets. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf

[24] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous Graph Neural Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 793–803. doi:10.1145/3292500.3330961

[25] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Understanding and Handling Alert Storm for Online Service Systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 162–171.