ADOps: An Anomaly Detection Pipeline in Structured Logs

Xintong Song
NetEase Fuxi AI Lab
Hangzhou, China
songxintong01@corp.netease.com

Yusen Zhu
NetEase Fuxi AI Lab
Hangzhou, China
hzzhuyusen@corp.netease.com

Jianfei Wu
NetEase Fuxi AI Lab
Hangzhou, China
hzwujianfei@corp.netease.com

Bai Liu
NetEase Fuxi AI Lab
Hangzhou, China
hzliubai@corp.netease.com

Hongkang Wei
NetEase Fuxi AI Lab
Hangzhou, China
weihongkang@corp.netease.com

ABSTRACT

Anomaly detection has been extensively implemented in industry. The reality is that an application may have numerous scenarios where anomalies need to be monitored. However, the complete process of anomaly detection will take much time, including data acquisition, data processing, model training, and model deployment. In particular, some simple scenarios do not require building complex anomaly detection models. This results in a waste of resources. To solve these problems, we build an anomaly detection pipeline(ADOps) to modularize each step. For simple anomaly detection scenarios, no programming is required and new anomaly detection tasks can be created by simply modifying the configuration file. In addition, it can also improve the development efficiency of complex anomaly detection models. We show how users create anomaly detection tasks on the anomaly detection pipeline and how engineers use it to develop anomaly detection models.

PVLDB Reference Format:
Xintong Song, Yusen Zhu, Jianfei Wu, Bai Liu, and Hongkang Wei. ADOps: An Anomaly Detection Pipeline in Structured Logs. PVLDB, 16(12): 4050 - 4053, 2023.
doi:10.14778/3611540.3611618

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/tensortkk/ADOps.

1 INTRODUCTION

Anomaly detection aims to identify behaviors that do not conform to expectations[8], and has been extensively implemented in domains such as online game monitoring, and financial fraud detection. Logs are one of the most valuable data sources for anomaly detection[14]. The structured logs record the massive behaviors of users in games or applications (trading, battles, advertising clicks ,etc.). The majority of abnormalities can be inferred based on log sequences because they usually violate regular rules. So far a wide variety of anomaly detection methods have been devised including supervised methods[4], semi-supervised methods[17], and unsupervised methods[3]. Nevertheless, they focus on the promotion of the accuracy of anomaly detection or the scarcity of available labeled anomalies. There is also a shortage of studies on how to establish an efficient, stable, and general anomaly detection pipeline, which is vital in the industry. In many scenarios, hundreds or thousands of items may need to be monitored for anomalies, so training an anomaly detection model for each one and deploying it online is difficult to achieve. It requires a lot of time and computing power.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611618

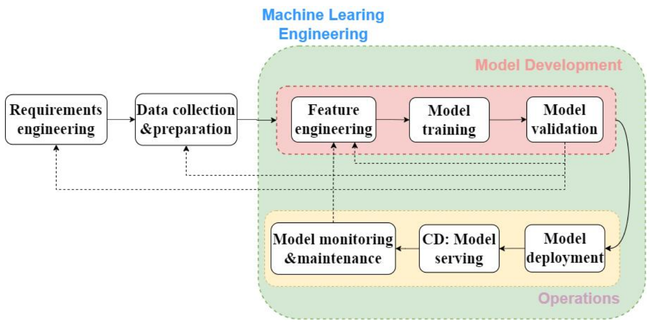

Figure 1: An pipeline of MLOps

Challenges remain in establishing an anomaly detection pipeline. Application developers may need to configure hundreds of detection items to monitor whether anomalies occur. Consequently, monitored items need to be easy to extend and modify in this pipeline. Moreover, each item possesses its own monitoring rule. For complex scenarios, e.g. real money trading detection, user behavior sequence modeling is momentous to detect outliers[16], while some simple scenarios may only need to execute a SQL command to locate anomaly log. It requires the pipeline to be able to implement a mass of anomaly detection methods. At last, the pipeline needs to have the ability to analyze continuous streams of data to meet the demand for real-time anomaly detection.

To overcome these challenges, we construct a pipeline of log anomaly detection named ADOps to provide efficient and diverse anomaly detection services. The system is derived from the concept of Machine Learning Operations (MLOps[1]). MLOps is a combination of machine learning and operation. It is a set of methods for automating the lifecycle management of machine learning algorithms in production. The core problem to be solved by MLOps is to

[1]https://ml-ops.org/

shorten the iterative cycle of model development and deployment. Figure 1 illustrates the components of MLOps. (1)Design, including requirements engineering, machine learning(ML) use-cases prionzation, and data availability check. (2)Model development, including data engineering, ML model engineering, model testing & validation. (3)Operations, including ML model deployment, CI/CD/CT Pipelines, monitoring & triggering.

The ADOps enables to (i) accept JSON format for establishing or modifying anomaly monitoring tasks; (ii) execute multiple anomaly detection methods(SQL command, xgboost[7], autoencoder[1], etc.) (iii) recommend appropriate anomaly detection thresholds for users and (iv) support offline analysis of logs but also anomaly detection on the data stream.

To demonstrate the benefits of the proposed system, we designed several interfaces to exhibit the different functions of ADOps. First, it allows users to specify thresholds to query the anomaly logs; second, it can recommend suitable thresholds for users; third, it enables to monitor the data stream in real-time according to user-configured detection rules, and finally, it helps engineers to develop and deploy personalised models for anomaly detection more efficiently.

2 PRELIMINARIES

In this section, we present the necessary background for the remainder of the paper.

[Datasets] The dataset we use consists of a public dataset and a private dataset, the public dataset is derived from the Kaggle competition to detect the presence of click fraud[2]. The private dataset is derived from structured logs of NetEase game[3]. Each entry records the player's behavior in the game or the player's state at that moment. The data is stored on the distributed file system HDFS.

[Time series and streaming data] A time series is a set of variables ordered by time, and timestamping is one of its key attributes. We could define it as $X = [x_1, x_2, ..., x_T]$, $X \in \mathbb{R}^T$. T is the length of X, and The appearance of x_i precedes that of x_j, $i < j$. Streaming data can be considered a dynamic collection of data that grows indefinitely with the continuation of time. In our dataset, streaming data is a generalization of the time series, i.e., $T \rightarrow \infty$.

[Anomaly Definition] Anomalies mainly contain two categories: point anomalies and contextual anomalies[5]. A point anomaly is an individual data instance that can be considered anomalous concerning the rest of the data (e.g., a player's level must be at least 50 when entering a specific game scenario). A contextual anomaly is a data instance anomalous in a specific context but not otherwise (For example, the player's acquisition of gold coins over a while exceeds a specified threshold). Since logs are refreshed daily at a fixed time, contextual anomalies are subdivided in this paper into anomalies with refresh time and anomalies without refresh time.

3 SYSTEM OVERVIEW

[Preparation] We built a log offline analysis tool(OATool) that explores anomalies under different thresholds. It requires the user

[2]https://www.kaggle.com/competitions/talkingdata-adtracking-fraud-detection/overview
[3]http://game.163.com/

Table 1: Similarity of manual-based and gaussian-based

Anomaly Type	Similarity
Point	95.6%
Contextual	89.8%
Contextual(refresh)	93%

to set anomaly detection thresholds. It parses the log ID, monitoring items, and other filter conditions the user provides into SQL statements to retrieve logs related to the monitoring items from plenty of logs. Log filtering is implemented through the Impala engine[11] because it communicates faster with HDFS. Log filtering can output anomalies directly for point anomalies, and for contextual exceptions, it will aggregate filtering results based on the user-defined time range. OATool assists in the proper functioning of the components of ADOps and offers candidate dimension sets for interpretative analysis of anomalies.

3.1 ADOps Framework

ADOps is an anomaly detection pipeline that can interact with the user. Given a set of user-defined monitoring tasks (including log id, monitoring items, etc.), output anomalies with explanatory exploration. Figure 2 shows an overview of ADOps. The lines in the figure represent the process of moving the data. ADOps comprises three main components: the Pre-computation Engine; the Feature Store Database and the Intelligent Detection Engine. ADOps is able to recommend appropriate anomaly classification thresholds for users and create services for real-time monitoring. When the recommended thresholds do not cover the user's needs, the engineer can develop and deploy anomaly detection models on ADOps to detect anomalies intelligently.

[Pre-computation Engine] Modeling each monitoring item is difficult to achieve when there are many monitoring items (hundreds or thousands), simple scenarios, and a lack of labeled data. Filtering anomalies by thresholds and then labeling them is an efficient way. The pre-computing engine aims to recommend anomaly thresholds to novice users who need help setting them. It can (i) handle massive amounts of data; (ii) personalise the recommended threshold for each monitored item in the log; (iii) provide a short response time. Taking the threshold based on expert knowledge(manually configured) as the baseline, we calculate the similarity between the threshold by Gaussian modeling[9, 12] and the baseline. The comparison result is shown in Table 1 and they have a significant similarity. Based on these, we recommend thresholds for users by pre-computation. We employ Apache Kylin[4] as the *Pre-computation Engine* to measure (e.g., COUNT, SUM, AVG) potential monitoring items using player id and time (minute level) as dimensions.

Kylin applies Apache Spark[18] for data computation and stores computation results as parquet tables. Since the pre-computed results are fine-grained (at the minute level), the actual monitoring items may reach the ten-minute or hourly level. The *Pre-computation Engine* first needs to aggregate the result table and then generate the recommended thresholds by the Gaussian model(3σ rule, box plot rule, etc.). The *Pre-computation Engine* then interacts

[4]https://kylin.apache.org/

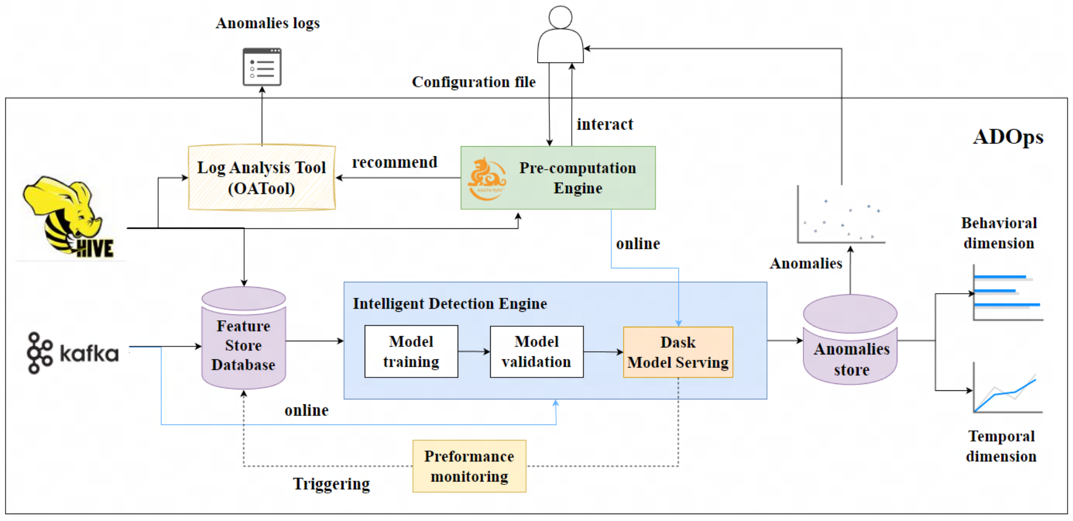

Figure 2: The ADOps architecture.

with the user. The user can select either a recommended threshold or a custom threshold. Then, the *Pre-computation Engine* passes the configuration information to the intelligent detection engine to create a real-time task. The *Pre-computation Engine* only provides initial thresholds for anomaly monitoring, and subsequently, engineers can update the thresholds based on actual anomalies.

[Feature Store Database] In machine learning tasks, the original data must undergo a series of pre-processing operations before being fed into the machine learning model, such as one-hot encoding and normalization. The data obtained after these transformations are called features, which are also the input to the machine learning model. Feature Store[5] Database (FsDB) is a data management tool for machine learning. It stores feature values and endorses sharing feature values across teams and domains. FsDB is the data provider for intelligent detection engines. It is built based on OpenMLDB[6]. FsDB provides the following aspects: (*i*) guarantees data consistency during training and inference of anomaly detection models; (*ii*) ensures low tail latency for feature construction in online scenarios. (*iii*) supports distributed storage and computation. It manages offline data(derived from Hive[2] or local files) separately from online data(from Apache Kafka[6]). In Fsdb, all feature processing relies on SQL statements. Engineers can use SQL statements to turn offline data into features for model training. Then it could switch the SQL feature processing statement from offline to online mode and deploy online. Finally, it could load the real-time data and perform online real-time feature computation for model inference.

[Intelligent Detection Engine] The *Intelligent Detection Engine* is responsible for managing online tasks and improving the efficiency of model development. It receives the monitoring configuration from the *Pre-computation Engine* and creates Dask[15] real-time monitoring tasks based on the configuration items. In some complex scenarios, threshold monitoring may not meet the user's demands, and the engineer must build a personalized model to detect the anomalies. The pipeline reduces model development time (from data processing to model validation). Engineers can simplify feature processing with Fsdb and model deployment with the *Intelligent Detection Engine*. They can load offline and online features from the FsDB for model training and validation.

Engineers can easily manage and maintain anomaly detection tasks on the *Intelligent Detection Engine*. When users need to add new monitoring items, they can add new tasks by simply updating the log ID and monitoring items entered by users into the configuration file. Moreover, the *Intelligent Detection Engine* can assist in model deployment. After the engineer completes the model training, he can provide data format, model storage location, and model type to the intelligent detection engine in the form of JSON, and the intelligent detection engine can deploy the model online to provide real-time anomaly detection services. All the anomaly data is saved in the MySQL database[7], when the engineer finds that the anomaly data is wrong, the model can be iteratively updated. The iterative process is shown in Figure 2, where the data is re-written to the feature store database and the anomaly detection pipeline is re-executed.

Intelligent Detection Engine also has a built-in shap library[13] that allows anomalies to be explored in both temporal and behavioral dimensions, helping engineers to attribute them. The temporal dimension refers to the change in the value of the monitored item over a while before and after the anomaly. The behavioral dimension refers to the contribution value of other features to determine an outlier. A bigger contribution value indicates that the feature may be wrong and needs to be investigated by the engineer. The outlier analyzer can display the feature contribution of a single outlier or the feature distribution of multiple outliers by clustering.

4 DEMONSTRATION

In our demo, users will use ADOps to create anomaly detection tasks. Users can get recommended anomaly thresholds from ADOps

[5]https://www.featurestore.org/
[6]https://kafka.apache.org/
[7]https://www.mysql.com/

Listing 1: An example of the monitoring settings for contextual anomaly

```
{
  "projectName": projectName,
  "logid": logid,
  "montior": "DELTA",
  "auxiliary_field": [ "SERVER", "ROLE_ID", "SRCTYPE",
      "MINUTE", "DS" ],
  "ts_thd": 10,
  "value_thd": 8000,
  "aggregation": "sum",
  "op": "greater",
  "filter": [ [ "srctype", "=", "14" ] ],
  "name": "stream_monitoring"
}
```

or customize them. We will also demonstrate how engineers can quickly develop anomaly detection models on ADOps using Jupyter notebook[10].

Anomaly Detection Task Creation. Initially, the user needs to specify the log ID, the monitoring item, the type of anomaly (point anomaly, contextual anomaly, contextual anomaly with refresh time), and other auxiliary fields. For contextual anomalies, the user also needs to specify a time range (10 minutes, 2 hours, 1 day, etc.) with log update times. These information form a monitoring setting. Each monitoring setting represents an anomaly detection task and is passed as JSON to the *Pre-computation Engine* interface. ADOps returns the corresponding recommended anomaly threshold. The user can either use the recommended thresholds directly or modify the thresholds and then pass the modified monitoring setting to the *Intelligent Detection Engine* interface, which will successfully create a new anomaly detection task.

Listing 1 is an example of the monitoring settings for contextual anomaly. $ts\_thd = 10$ represents the range of time is ten minutes; $value\_thd$ is anomaly threshold; $montior$ is the monitoring item; $aggregation$ is the aggregation methods for streaming data; and op is the comparison operator. This example means that a player who gains deltas greater than 8000 in 10 minutes is an outlier. All anomaly settings and anomalies are stored in the MYSQL database and can be viewed by the user.

Anomaly Detection Model Development. After connecting to FsDB, engineers can create and store features using SQL statements. After the model design is complete, engineers can use the offline features in FsDB to train the model. Subsequently, engineers can pass model deployment configuration to the intelligent detection engine and deploy it online. Engineers can also explore the anomaly detection model and anomaly data in terms of temporal and behavioral dimensions. When it is time to iterate the model, engineers can directly reuse the original feature-processing SQL statements without redeveloping them.

5 CONCLUSION

In this demo, we present ADOps, an anomaly detection pipeline that allows users to quickly build multiple real-time anomaly detection tasks without requiring programming. It also provides a recommended solution for users with no knowledge of anomaly detection and can improve the efficiency of anomaly detection model development. With two demo scenarios, we showed that ADOps make it convenient to detect anomalies and develop models.

ACKNOWLEDGMENTS

This work is supported by the Key Research and Development Program of Zhejiang Province (No. 2022C01011). We are also grateful to our colleagues of NetEase Games for providing the existing anomaly detection rules as a basis for ADOps construction.

REFERENCES

[1] Jinwon An and Sungzoon Cho. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE* 2, 1 (2015), 1–18.

[2] Jesús Camacho-Rodríguez, Ashutosh Chauhan, Alan Gates, Eugene Koifman, Owen O'Malley, Vineet Garg, Zoltan Haindrich, Sergey Shelukhin, Prasanth Jayachandran, Siddharth Seth, et al. 2019. Apache hive: From mapreduce to enterprise-grade big data warehousing. In *Proceedings of the 2019 International Conference on Management of Data*. 1773–1786.

[3] Lei Cao, Yizhou Yan, Caitlin Kuhlman, Qingyang Wang, Elke A Rundensteiner, and Mohamed Eltabakh. 2017. Multi-tactic distance-based outlier detection. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 959–970.

[4] Raghavendra Chalapathy, Ehsan Zare Borzeshi, and Massimo Piccardi. 2016. An investigation of recurrent neural architectures for drug name recognition. *arXiv preprint arXiv:1609.07585* (2016).

[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[6] Cheng Chen, Jun Yang, Mian Lu, Taize Wang, Zhao Zheng, Yuqiang Chen, Wenyuan Dai, Bingsheng He, Weng-Fai Wong, Guoan Wu, et al. 2021. Optimizing in-memory database engine for AI-powered on-line decision augmentation using persistent memory. *Proceedings of the VLDB Endowment* 14, 5 (2021), 799–812.

[7] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[8] Dennis Hofmann, Peter VanNostrand, Huayi Zhang, Yizhou Yan, Lei Cao, Samuel Madden, and Elke Rundensteiner. 2022. A demonstration of AutoOD: a self-tuning anomaly detection system. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3706–3709.

[9] Paul S Horn, Lan Feng, Yanmei Li, and Amadeo J Pesce. 2001. Effect of outliers and nonhealthy individuals on reference interval estimation. *Clinical chemistry* 47, 12 (2001), 2137–2145.

[10] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. *Jupyter Notebooks-a publishing format for reproducible computational workflows.* Vol. 2016.

[11] Marcel Kornacker, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Casey Ching, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, et al. 2015. Impala: A Modern, Open-Source SQL Engine for Hadoop.. In *Cidr*, Vol. 1. Asilomar, CA, 9.

[12] Jorma Laurikkala, Martti Juhola, Erna Kentala, N Lavrac, S Miksch, and B Kavsek. 2000. Informal identification of outliers in medical data. In *Fifth international workshop on intelligent data analysis in medicine and pharmacology*, Vol. 1. 20–24.

[13] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 4768–4777.

[14] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.

[15] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference 2015 (SciPy 2015), Austin, Texas, July 6 - 12, 2015.* scipy.org, 126–132.

[16] Jianrong Tao, Jianshi Lin, Shize Zhang, Sha Zhao, Runze Wu, Changjie Fan, and Peng Cui. 2019. Mvan: Multi-view attention networks for real money trading detection in online games. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2536–2546.

[17] Miryam Elizabeth Villa-Pérez, Miguel A Alvarez-Carmona, Octavio Loyola-Gonzalez, Miguel Angel Medina-Pérez, Juan Carlos Velazco-Rossell, and Kim-Kwang Raymond Choo. 2021. Semi-supervised anomaly detection algorithms: A comparative summary and future research directions. *Knowledge-Based Systems* 218 (2021), 106878.

[18] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.