Anomaly Detection from System Tracing Data using Multimodal Deep Learning

Sasho Nedelkoski*, Jorge Cardoso†, Odej Kao*
*Complex and Distributed IT-Systems Group, TU Berlin, Berlin, Germany
Email: {nedelkoski, odej.kao}@tu-berlin.de
†Huawei Munich Research Center, Huawei Technologies, Munich, Germany
Email: jorge.cardoso@huawei.com

***Abstract*—The concept of Artificial Intelligence for IT Operations (AIOps) combines big data and machine learning methods to replace a broad range of IT operations including availability and performance monitoring of services. Such platforms typically use separate models for each modality of monitoring data (e.g., textual properties and real-valued response time in logs and traces) to detect faults and upcoming anomalies in cloud services, which do not capture the existing correlation between the modalities. This paper extends the range of utilized data types for creation of a single model to improve the anomaly detection. We use a bimodal distributed tracing data from large cloud infrastructures in order to detect an anomaly in the execution of system components. We propose an anomaly detection method, which utilizes a single modality of the data with information about the trace structure. In the next step, we extend the single-modality neural architecture to a multimodal neural network with long short-term memory (LSTM) to enable the learning from the sequential nature of both modalities in the tracing data. Furthermore, we demonstrate an approach to detect dependent and concurrent events using the ability of the model to reconstruct the execution path. The implemented prototype is experimentally evaluated with data from a large-scale production cloud. The results demonstrate that the novel approaches outperform other deep-learning methods based on traditional architectures.**

***Index Terms*—AIOps; anomaly detection; multimodal deep learning; distributed tracing; service reliability; LSTM.**

I. Introduction

The increasing number of IoT applications with dynamically linked devices and their implementation in real-world environments drive the creation of large multi-layered systems. Consequently, the complexity of the systems is steadily increasing to a level, where it is impossible for human operators to oversee and holistically manage the entire systems without additional support and automation. However, as uninterrupted services with guaranteed latency, response times, and other Quality of Service parameters are required for many data-driven autonomous applications, loss of control is not allowed for any system or infrastructure. Large service providers are aware of the need for always-on dependable services and thus already deployed various measures by introducing additional human and artificial intelligence to the IT ecosystem. The next step is to rapidly decrease the reaction time in case an urgent administration activity is needed to prevent a system anomaly developing into a fault. Such anomalies are typically evolving from performance problems, component/system failures (e.g., outages, degraded performance), or security incidents. Therefore, an important part of AIOps platforms is to detect and recognize the anomaly, before it leads to a service or system failure.

The foundation for AIOps platforms is the availability of suitable and descriptive data. As shown in Figure 1, the observational data consists three components: tracing, logging, and monitoring information. The tracing component produces events (spans) containing bi-modal information reflecting the execution path in form of sequence of text labels and the real-valued response time describing the service performance. The log data represent interactions between data, files, or applications that are typically used to analyze trends or to record decisive events/actions for a later forensic. The widely collected monitoring data describes the current utilization and status of the infrastructure, typically as a cross-layer information regarding CPU, memory, disk, network throughput, and latency.

While the anomaly detection on system log and metric data has been already investigated [1]–[4], the use of tracing data is still limited as it is significantly more complex to collect and handle. The tracing data also can be used to know the underlying infrastructure, which previously has been obtained by topology inference approaches [5]. However, currently developed technologies for distributed services, as part of cloud-based operating systems, enable to also record tracing data information about all of the individual components involved in a particular user request (initiator) or remote procedure call [6].

The current state-of-the-art systems for anomaly detection using log data model the normal system behavior out of a single data type, which is either the textual log keys, or real-valued performance parameters. Commonly, they use separate models for both types of data and build an ensemble to generate the final prediction [3]. Other approaches operate solely on one data modality [1], [2] or apply separate modeling techniques for multiple modalities, which are later integrated into an unified indicator [7].

However, such additive models do not utilize the existing correlation between the data sources. They learn the normal system behavior from partial and limited data which might affect the overall performance. The learning from fused representations of multimodal data has provided good results in

various tasks, matching or outperforming other deep learning models [8]. Therefore, we investigate methods for unsupervised anomaly detection out of sequential, multimodal data.

The tracing data collected during execution of system actions consist of two modalities:

- *service response time* in the form of real-valued data
- *causal relationships* with other related services represented as a sequence of textual labels.

As shown in Figure 1, if user request (e.g., create virtual machine) involves the services $\{11, 21, 31, 32\}$, then a trace contains events representing the intra-service calls produced when each service is invoked. We believe that such data can improve the anomaly detection, root-cause analysis, and remediation in the system, since it contains very detailed information about the state of the system from a service perspective. Therefore, we transfer and compile the tracing data into an abstract structure, which is similar to structures for anomaly detection in processes, logs, natural languages, or any sequential data in order to exploit and improve sophisticated analysis methods developed in these domains.

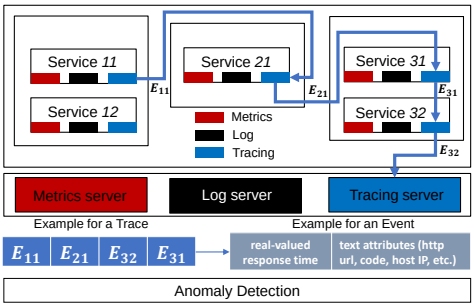

Fig. 1. Overall system architecture showing communication between services and the three system observability components. We combine two modalities of tracing data in a single model for anomaly detection in cloud infrastructures.

Contributions. This work addresses anomaly detection in large cloud infrastructures by using LSTM neural networks [9] with data from distributed tracing technologies. We present a deep learning model for sequence learning to model the causal relationship between the services in a trace using the single-modality, sequential text data. We show the importance of the response time as a second type of data in a trace and challenges for its modelling. We extend the single-modality architecture by introducing a model, which utilizes the multimodal tracing data as a combination of a text and real-valued sequence. We show that the multimodal approach can be used to model the normal system behavior and detect anomalies considering not only the causality of the services but also their response times within a trace. Furthermore, we detect dependent and parallel tasks using the model to reconstruct the execution path. Through an exhaustive experimentation on data from a real production cloud, we show that our approaches outperform the baselines with an accuracy larger than 90%, and that the multimodal LSTM achieves the best overall accuracy.

Outline. The rest of the paper is structured as follows. In section II we discuss the related work. The proposed methodology is presented in section III, while section IV shows the evaluation results. Section V presents the conclusions of this paper.

II. Related Work

The anomaly detection in large-scale systems is widely studied. We review past research of unsupervised methods, as the labelling by experts or injection of anomalies directly into the cloud platforms to obtain labeled data do not meet the requirements of real-word systems. This owes to the appearance of new patterns or possible harm to the running system by an intentional fault injection.

The recent advancement of deep learning has led to performance breakthroughs for various problems including sequence-to-sequence learning tasks [10], [11]. Specifically, LSTM Networks [9] are most commonly utilized in related applications. Malhotra et al. [12] used stacked recurrent hidden layers to enable learning of higher level temporal features. They presented a model of stacked LSTM networks for anomaly detection in time series. A network was trained on non-anomalous data and was used as a predictor over a number of time steps. Taylor et al. [13] proposed an anomaly detector based on an LSTM neural network to detect network attacks. The detector operates by learning to predict the next data word originating from each sender on the network. If there is a difference between the prediction and the actual data word, an anomaly is flagged.

Brown et al. [14] presented recurrent neural network (RNN) language models augmented with attention for anomaly detection in system logs. Du et al. [3] proposed the state-of-the-art DeepLog. It models the recent history of log events as a sequence, outperforming other traditional machine learning algorithms on various datasets. DeepLog splits the log key information from the values in the logs and models the data using two different models. The log keys are formulated as a multiclass classification task over the recent context. They use history h of recent log keys as input, and try to predict the next log key in the sequence. For a given sequence of log keys $\{k_1, k_2, \ldots, k_n\}$ and a history window of size h, the input and the output for training are: $\{k_1, k_2, k_3 \rightarrow k_4\}, \{k_2, k_3, k_4 \rightarrow k_5\} \ldots \{k_{n-3}, k_{n-2}, k_{n-1} \rightarrow k_n\}$.

In contrast to the above anomaly detection systems, we aim to develop a single model that utilizes the multimodal nature of system data. We explain our approaches through anomaly detection in tracing data. Nonetheless, they can be generally employed in many other applications involving anomaly detection for multimodal system data.

In multimodal deep learning, Ngiam et al. [15] proposed a novel application of deep networks to learn features over multiple modalities. They presented a series of tasks for multimodal learning and showed how to train deep networks that learn features to address these tasks. Srivastava et al. [8] proposed a probabilistic method for multimodal deep learning with Deep Boltzmann Machine (DBM). They showed how to extract a meaningful representation of multimodal data, later used for classification and information retrieval tasks.

These studies paved the way for further progress in the field. Park et al. proposed a LSTM-variational autoencoder to detect anomalies in robot-assisted feeding. They showed that learning from multimodal sensory signals can be helpful for detection of a wide range of anomalies, overcoming the challenges from the fusion of high-dimensional and heterogeneous modalities. These approaches have ultimately demonstrated that learning from multimodal data opens new perspectives, but were not yet investigated for anomaly detection in complex and large systems.

III. Multimodal Anomaly Detection from Tracing Data

This section explains the parsing of the tracing data, the single-modality LSTM, its extension to the multimodal LSTM neural network, and describes the method for reconstruction of the execution path in order to detect concurrent or dependent events providing characteristic insights from the tracing data that allow to perform better root-cause analysis. Both proposed approaches for anomaly detection model the normal system behavior and detect anomalies by flagging any deviations.

A. Trace Data and Parsing

Traces in microservice architectures are composed of events (spans) [16]. An event is a vector of key-value pairs (k_i, v_i) describing the state, performance, and further service characteristics at a given time t_i. Each time a user executes a command to request a record to be inserted, updated, deleted from a database, or when it calls an external server using, e.g., remote procedure calls (RPCs), one or more events within a trace are generated, as shown in Figure 1.

A trace $T = \{e_0, e_1, \ldots, e_i\}$ is represented as an enumerated collection of events sorted by the timestamps. The analogy to the natural languages as type of sequential data originates from this representation, where one can map the trace to a sequence of words, the events inside a trace to words, and the causal relationship between events to a language grammar. Each event in the trace contains at least the following attributes:

- trace ID (identifier that assigns an event to a trace), event ID (unique event identifier), parent ID (event ID of the parent service)
- protocol (can be either HTTP or function protocol), host IP, HTTP return code, HTTP URL
- response time (time difference between start and stop of the execution of the service) and timestamp (when the particular service was invoked)

Depending on the executed action, the traces can have different lengths and events representing various services. Two sample traces $T_p = \{e_0^p, e_1^p, e_2^p, \ldots, e_i^p\}$ and $T_q = \{e_0^q, e_2^q, e_1^q, \ldots, e_i^q\}$ are different in structure (e_1 and e_2 are swapped), but originate from the same system activity. This type of behavior occurs frequently in real systems, where the events e_1^p and e_2^q originate from concurrently invoked services. The order of events depends on the concurrent invocation of the recording component, so that both traces represent the same execution path.

The key-value pairs from an event are recorded as JSON objects. We parse the entries into a structured, vector representation, which then serves as an input into the LSTM neural network. For each event we extract two data modalities: textual label, characterizing the type of the event, and response time, describing the service performance. Before the computation of the label, we extract the service endpoint information from the HTTP URL by applying regular expression filter. For example, https://1.1.1.11/v2/a16d/servers/detail is transformed into v2/id/servers/detail. We denote the post-regex expression as HTTP pattern. The final label is then formed by concatenating the HTTP code, the HTTP pattern, and the host IP (e.g., 200_v2_id_servers_detail_126.75.191.253). The label is then added to a dictionary. In order to increase the robustness of the algorithm, we avoid labels that appear only few times by considering the top-M most frequent labels. Finally, an additional label ('!0') is reserved for padding and trace ending. This symbol maps to the zeroth index in the dictionary.

We denote the number of unique labels in the dictionary as N_l. The unique numerical indices in the dictionary are used to represent the event's attributes. To this end, the traces are represented by numerical vectors with different size. We then create vectors with a predefined fixed length by applying padding up to T_l, which represents the maximum allowed trace length. Traces, which are longer than T_l, are truncated. This makes the traces equally sized, but still they contain different number of non-zero elements. The vector representation is then converted into a one-hot categorical encoding [17] making the data format ready for training with shape $D_1 = (N_t, T_l, N_l)$, where N_t is the number of all recorded traces. D_1 describes the structure of the traces, i.e., contains information for the execution path of the events in the trace.

The response times of the events in all traces are grouped by label and min-max scaled between zero and one, so they can be viewed as a time-series. These real-valued numbers provide an additional dataset with a shape of $D_2 = (N_t, T_l, 1)$. For each event we have one float value representing the its response time.

In following, we propose a single-modality architecture using only the labels, motivate the use of the response time as a second data modality, and describe the design changes needed to train models with this type of data.

B. Single-Modality LSTM network

We denote the set of all traces recorded within a period as $T_{set} = \{T_0, T_1, \ldots, T_{N_t}\}$, while $L = \{l_0, l_1, \ldots, l_{N_l}\}$ is the set of unique labels in the data. Furthermore, e_i denotes the one-hot encoding value of the label $l_j \in L$ positioned at index i in the trace T_k for $k \in \{0, 1, \ldots, N_t\}$. The value of e_i depends on the trace structure prior to e_i, as the events in a trace originate from the execution of services upon a user request, where the events have a parent-child causal relationship [16]. We model the structural anomaly detection in traces as a sequence-to-sequence, multi-class classification

problem, where each distinct label represents a class. Figure 2 shows a single-modality architecture for both types of data D_1 (Structural Anomaly Detection, SAD) and D_2 (Response Time Anomaly Detection, RTAD). The detection of anomalies from a sequence of labels enables to capture the errors during the execution as well as to detect unexpected execution paths.

The input to the model are the event labels from a trace $T_k = \{e_0, e_1, \ldots, e_{T_l}\}$. Each e_i is fed as input in the corresponding timestep $time = i$. The output at $time = i$, for the current inputs $\{e_0, e_1, \ldots, e_{i-1}\}$, is a probability distribution over the N_l unique labels from L, representing the probability for the next label e_i in the sequence. The detection phase uses this model to make a prediction and compares the predicted output against the observed label value. The LSTM network is trained to maximize the probability of each e_i ($i \in \{1, 2..T_l\}$) to appear as a next label. Every LSTM block in Figure 2 at $time = i$ is composed of h LSTM cells. It has a memory state that encodes all of the information from the previous timesteps together with the input fed at the same timestep. LSTMs use different types of trainable gates [9], which together with the the input label at $time = i$ and the output from the previous block H_{i1} are used to decide:

- how much of the previous cell state C_{i1} should retain in its own state,
- how to use the current input and the previous output H_{i-1} to influence the state, and
- how to construct the output H_i.

In this manner, the possible extracted non-linear and temporal information is passed between adjacent LSTM blocks.

The stacking of layers as shown in Figure 2 is a common practice in order to achieve better results by extracting highly-abstract features. We formulate each input-output pair from D_1 as

$$T_{input} = \{e_0, e_1, \ldots e_{T_l}\} \rightarrow T_{output} = \{e_1, e_2, \ldots e_{T_l}, \text{'!0'}\}$$

where the output is shifted by one event and concluded with the '!0' label. These pairs are used to incrementally update the network's weights through categorical cross-entropy loss minimization via gradient descent.

Detection. In order to evaluate, if a trace T_{test} represents an anomaly and to discover, which events support the decision, $T_{test} = \{e_0, e_1, \ldots, e_{T_l}\}$ is clamped to the network's input. In each timestep, the network calculates a probability distribution:

$$P = \{l_0 : p_0, l_1 : p_1, \ldots, l_{N_l} : p_{N_l}\}$$

describing the probability for each label from L to appear as the next label value in the trace, given the previous values. The output layer of the network is composed of a soft-max function. It distributes the probability over the labels and ensures that $\sum_i^{N_l} p_i = 1$.

Previously, we described that two or more events can be a result of multiple concurrent actions; therefore few of the possible labels can appear as the next label in the sequence. Comparison of the input label only to the most probable

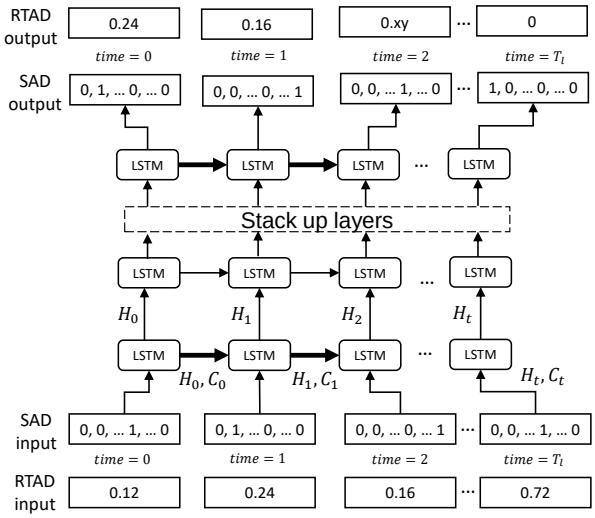

Fig. 2. Single-modality LSTM network architecture

label can be a measure for the performance of the model in terms of the ability to capture the normal execution paths. For robustness, we compare the observed label to the k predicted most probable labels: if the input label observed in the next timestep from the original sequence is not one of the top-k labels with the maximum probability, then an anomaly is reported, i.e. the trained network has not observed a normal trace with the same or similar structure. Along with the information that the trace contains an anomalous execution path, we also provide the user with information which events contributed to the decision. This enables a better root-cause analysis.

1) Response Time Anomaly Detection (RTAD): The response time characterizing intra-service calls is decisive for the anomaly detection, as a sudden increase may indicate a problem with the involved service or with the underlying distributed system. Unfortunately, the response time values grouped as a time-series exhibit a low signal-to-noise ratio and typically include multiple frequencies, distributions, and concept drifts. The low signal-to-noise ratio is a result of the different components affecting the response time of microservices such as switches, routers, memory capacity, CPU performance, programming languages, thread/process concurrency, bugs, and volume of user requests. Moreover, the dependencies between the events in a trace affect the response times. Assume a service A calls other service B, collects the result, and proceeds with the execution (parent-child relationship). In this case, the second service is a service-child. Therefore, an increase or decrease in the child's response time will correspondingly lead to a change in the parent's response time. The ability to detect specific events that are anomalous in such perspective provides more insights and extends the range of the anomaly detection from the tracing data.

The training of neural networks requires normalized values. When grouped by labels, the response times from different

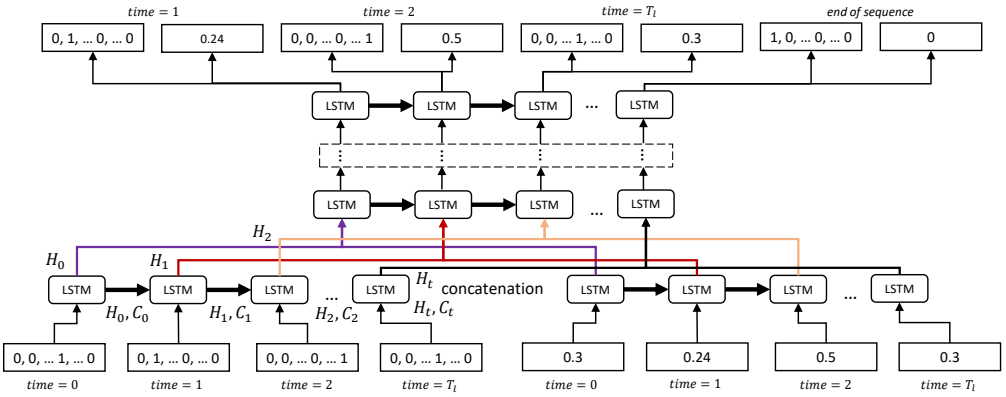

Fig. 3. Multimodal LSTM neural network architecture for anomaly detection from complete tracing data

services will be even more similar in terms of values. Therefore, learning a model only from the response times without coupling with the labels of the services is a hard task and leads to a poor performance. However, in order to extend the single-modality architecture to the multimodal LSTM, we need to describe how to model the response time independently of the labels. Therefore, we reuse the neural network architecture presented in Figure 2 and model the inter-event, response time dependencies in a trace using RTAD (D_2) data. The difference is that instead of the multi-class classification, the task is regression where the input and the output are real-valued numbers. In each timestep $time = i$, with $\{rt_0, rt_1, \ldots, rt_{i-1}\}$, the network predicts the response time rt_i for the next event. In contrast to SAD, the weight updates are obtained through minimization of the mean squared loss via gradient descent.

The detection is carried out by computing the squared error distance $error_i = (rt_i - rt_i^p)^2$, where rt_i^p is the predicted value at timestep $time = i$. Furthermore, we separately model the error values for each label by fitting the Gaussian distribution and produce a trace T_{test} predicted by the model. If the squared error between the prediction and the input at $time = i$ is out of the 95% confidence interval obtained from the Gaussian, the event and the trace are flagged as anomalous.

C. Multimodal LSTM

The response time together with the event's label completely characterizes a single event. The correlation between these two types of data motivates the need to use both modalities of the data in a single model aiming to extend the anomaly detection for tracing data and to achieve a better overall accuracy.

The proposed multimodal LSTM architecture is assembled as a horizontal concatenation of both single-modality architectures, as shown in Figure 3. The model contains two data modalities as inputs: D_1 and D_2. From the bottom-up perspective of the architecture, we have layer with LSTM blocks for each input. We perform the concatenation in the second hidden layer. The concatenation can be carried out in any hidden layer chosen by cross-validation. The merging of the two modalities is done in the following way: LSTM outputs from the first layer in $time = i$ are joined and forwarded into the same timestep in the next LSTM layer. In Figure 3, the color-coding scheme represents the concatenated pairs.

Starting from the concatenated layer, the information is jointly encoded and flows between modalities. From this joint representation, the many-to-many neural network learns the mapping to the outputs. The input-output pairs can be formalized as:

$$input = [\{e_0, e_1, \ldots, e_{T_l}\}, \{rt_0, rt_1, \ldots, rt_{T_l}\}]$$

$$\rightarrow output = [\{e_1, e_2, \ldots, e_{T_l}, \text{'!0'}\}, \{rt_1, rt_2, \ldots, rt_{T_l}, 0\}]$$

In the training phase, we use different cost functions for the modalities. The input-output pairs are used to learn the weight updates through minimization of a joint loss which is the sum of the mean squared error for the response time and the categorical cross-entropy for the labels.

Detection. The detection in the multimodal setting is performed by comparing the output element-wise with the input for both modalities using the strategy developed in the single-modality architectures. An advantage is that with a single model we can detect anomalies in both modalities, overcoming the limitations of not having a single-modality response time anomaly detection. For example, when there is an increased response time in one of the events, and the computed error between the prediction and the input is out of the confidence interval, the method reports an anomaly. Similarly to the SAD, if the observed input labels of the sequence are not in the top-k element-wise predictions, an anomaly is reported. Considering the both modalities, the anomaly type can be either: response time anomaly, structural anomaly, or anomaly in both modalities. It is worth noting that the anomaly in the response time can be independent of the structure and vice versa.

D. Detection of Dependent and Concurrent Events

The output of structured anomaly detection model encodes the underlying execution path. Every label predicted as the label for the next event in the trace describes a probability distribution of all possible labels. As described before, the

events have a parent-child relationship, which can be extracted from the recorded data. However, the challenge of recognition of concurrent and dependent events in the execution path is not solved yet and has to be addressed.

Considering that the neural network learns the underlying execution paths (i.e. the causal relationship between the events), we can reconstruct them using the predicted probability distribution over the labels, similar to the approach presented in Figure 4. Assume we aim to predict the event in the trace next to the vertical, dashed line. The input at timestep $time = 3$ is $\{l_1, l_3, l_8, l_{12}\}$. Let the top-2 predictions for the next label in the sequence be $\{p_6 = 0.55, p_{11} = 0.45\}$, while for the rest of them, the probability is zero. In order to determine if these two events are produced as a result from concurrently invoked services we analyze two different inputs: $\{l_1, l_3, l_8, l_{12}, l_6\}$ and $\{l_1, l_3, l_8, l_{12}, l_{11}\}$. If for both sequences the probability for the next label is $p_2 = 1.0$, the events (services) are concurrent as both of them lead to the same event.

We also propose a mechanism to detect dependent events. In this case, if the probability to observe l_k as a label for the next event in the sequence e_{i+1} is one, considering $\{e_0, e_1, \ldots, ei\}$ as input, then e_i and e_{i+1} are dependent.

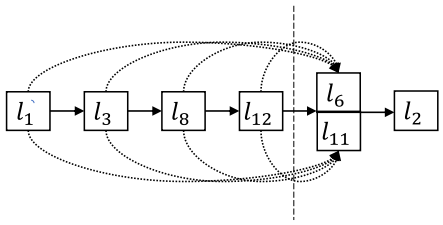

Fig. 4. Trace execution path with concurrent events, where the numbers represent the indices of the labels $l_i \in L$

In summary, the core approach from the described methodology is the multimodal LSTM network. As mentioned, it uses different data modalities, forms a joint representation, and utilizes the possible highly non-linear relationships between the data modalities in addition to the features extracted from the long-term dependencies using the LSTMs. This helps to achieve a higher accuracy than that for the single-modality architecture.

On the other hand, the detection of dependent and concurrent events helps to perform a better root-cause analysis and provides more insights from the observed data. In general, the multimodal approach is generic and can be used for anomaly detection in sequential data when multiple data modalities are available.

IV. Evaluation

The deep learning methods were implemented in Python using Keras [18]. The experiments on the collected dataset were carried out on regular personal computer using GPU-NVIDIA GTX 1060. For all models we used batch size of 512, learning rate of 0.001, and 400 epochs.

A. Experimental Setup

The data was collected from a production cloud platform which runs Openstack [19] with Zipkin [16] as a tracing technology. With more than 1000 micro services, the underlying system enables an exhaustive and realistic evaluation of our approach. To the best of our knowledge, there is no publicly available data set containing distributed traces, whereas Openstack log data can be easily generated using CloudLab [20]. The collected traces are recorded over a period of 50 days, yielding over 4.5 million events distributed in more than one million traces with different lengths.

The JSON objects representing the events, are parsed and the two different data modalities D_1 and D_2 are compiled. In order to avoid outliers, we select labels that appear more than 1000 times in the data, making a total of 105 unique labels. The distribution of the trace lengths in our dataset is imbalanced; more than 90% of the traces have lengths smaller than 10 events. This is compensated by selecting only 1000 samples of each trace length, where the trace lengths with less samples are completely included into the dataset. In this regard, our approach requires approximately less than 1% of all the recorded data, which makes it efficient and fast for training. For robustness, we also select the traces with lengths between 4 and 20. Traces with larger lengths appear only few times in the given time-period. We consider them as outliers and do not insert into the final training data set.

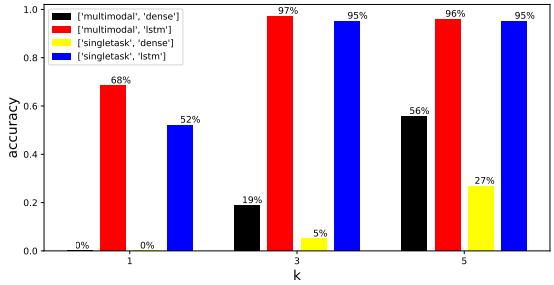

Fig. 5. Comparison of the overall accuracies of the models evaluated for three values of $k \in \{1, 3, 5\}$.

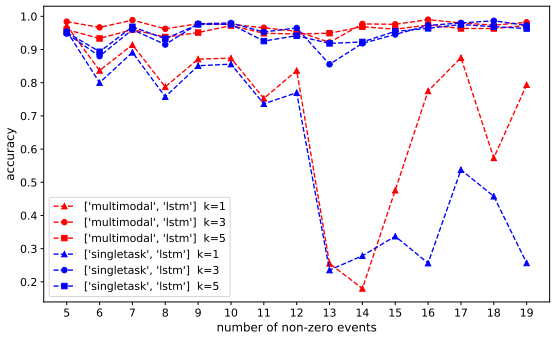

Fig. 6. Comparison of the accuracies of two best models, evaluated for each trace length $4 - 20$ and $k \in \{1, 3, 5\}$.

Anomaly Injections and Measurement of Accuracy for SAD. The accuracy evaluation for detected anomalies is carried

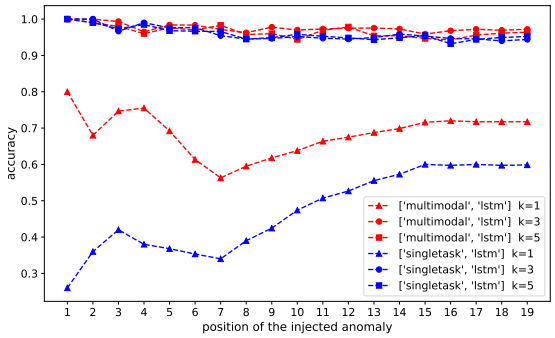

Fig. 7. Comparison of the accuracy of the two best models, evaluated for different positions $0, 20$ of the injected anomaly and $k \in \{1, 3, 5\}$.

out by artificially injecting anomalies in the data. For SAD, we performed anomaly injections as follows: given a trace $T_{test} = \{e_0, e_1, \ldots, e_{T_l}\}$, with N_z non-zero elements and top-k predictions for each position i in the trace, we select a label, which is not in the top-k predictions, and inject it at the position of the normal observed label. We then run the prediction with the model and determine a decision anomaly/normal by comparing the non-corrupted sample with the prediction. In case of output *true* and the label in the corrupted event position is not in the top-k predictions, the injected anomaly is successfully detected.

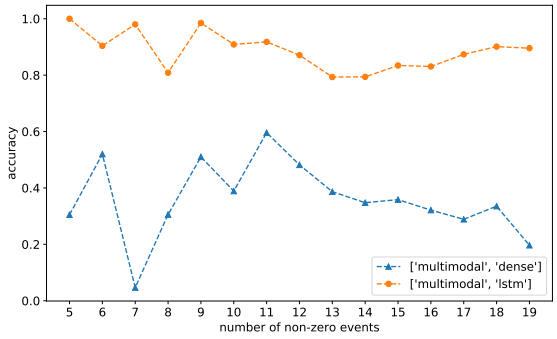

Fig. 8. Response time anomaly detection accuracy comparison of the multimodal LSTM and the baseline deep learning architecture evaluate for different trace lengths.

Anomaly Injections and Measurement of Accuracy for RTAD. We perform the anomaly injections by selecting the event response time rt_i at a position i of the trace. The anomaly is injected by increasing the response time of the event by a random value $r \in (2{*}rt_i, 5{*}rt_i)$. Once the anomaly is injected, we compute the mean squared error between the input and the predicted output. If the error is not in the 95% confidence interval computed from the Gaussian fitted on the training set, then the anomaly is detected successfully.

The accuracy is computed as the ratio of the number of successfully detected anomalies and the number of injected anomalies.

In practice, the anomaly may not appear only in a single event. We assume that a trace is anomalous if at least one event inside deviates from the normal behavior, making our approach applicable when the anomaly is spread across multiple events.

Baselines. We have two baseline models for comparison, single- and multimodal deep learning architectures composed of simple feed-forward neural networks. The architecture for the single-modality network is input, dense(50), dense(20), output, while for the multimodal is [D_1-input, D_2-input], dense(50) for D_1, dense(50) for D_2, concatenation, dense(20) for each, output for each. The decision for anomaly is done in the same way as previously described.

B. Results and Discussion

As shown in Figure 5, the best results in terms of accuracy for the task of structural anomaly detection are achieved using the multimodal LSTM predictions. The bar plot shows the accuracy of all models when different values of k are used. We observe that that the single-modality LSTM achieves a comparable accuracy, while the other two single- and multimodal dense architectures have low accuracies. The dense models can not take into account the temporal information. Compared to that of the single-modality LSTM architecture, the multimodal LSTM achieves a better accuracy owing to the additional response time information. The results for $k = 3$ and $k = 5$ are comparable, while the results for $k = 1$ show that the multimodal approach outperforms the single-modality by a large margin of 16%. Because of the low percentages obtained from the traditional models, we do not compare them below.

We evaluated the accuracy when the anomaly is injected in traces with different sizes, as shown in Figure 6, while ignoring the position of injection of the anomaly. Both proposed architectures achieve high accuracies for $k \in \{3, 5\}$. The multimodal slightly outperforms the single-modality approach in 9 out of 15 trace lengths for both k. Significantly better results are achieved for $k = 1$ for almost all of the trace lengths. The plot demonstrates that both approaches are stable, without performance reduction when the trace length is increased.

Further, for SAD, we compared the two best models when the anomaly is injected in different positions in the trace for $k \in 1, 3, 5$, while ignoring the trace length. Similarly, Figure 7 shows a significantly better performance of the multimodal approach for $k = 1$. Furthermore, the accuracy is also stable at all of the different positions of the trace. This implies that the model successfully detects injected anomalies for different events in different positions of the trace.

RTAD. The single-task models for sequential response time modelling have quite low performances than those of both multimodal models, and thus we discuss only those results. Figures 9 and 8 show comparisons of the two multimodal approaches for different positions of the injected anomaly and different trace lengths. In both figures, the multimodal approach achieves a higher accuracy for response time anomaly detection. Figure 8 shows that the accuracy slightly decreases when the trace length is larger. In Figure 9, the reason why

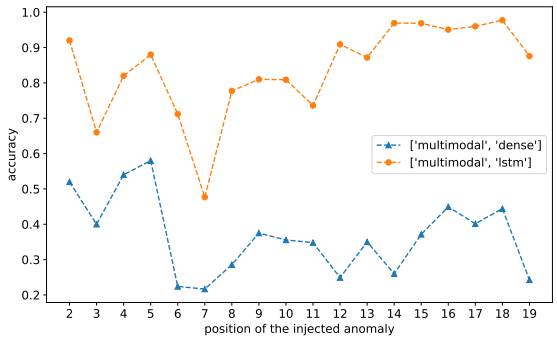

Fig. 9. Response time anomaly detection accuracy comparison between the multimodal LSTM and the baseline deep learning architecture, evaluated for different positions of the injected anomaly.

the accuracy drops at position 7 is probably because of the small signal-to-noise ratio and the existence of outliers. In our approaches, we have not applied any preprocessing or outlier removal techniques, but we assumed that the recorded data represent the normal behavior. The use of preprocessing techniques for the response time may eventually help improve the accuracy for the response time in critical positions.

The models are consistent and have high accuracy even when the length of the trace increases. This is because of the LSTMs which are able to learn long-term dependencies in sequential tasks.

Performance. The time needed to train the multimodal LSTM on the 1% representative sub-sample of over one million traces was approximately 30 min. The model is applicable for real-time anomaly detection with a prediction time per trace below 50 ms.

V. Conclusion and Future Work

This paper addressed an important and growing challenge from the field of AIOps: the anomaly detection in large-scale cloud infrastructures using tracing data that contains detailed information about inter-service calls.

We addressed the problem using sequential deep networks for structural anomaly detection and presented approaches to recognize dependent or concurrently invoked services. We further extended the approach by an architecture for multimodal anomaly detection. This approach enables to detect structural and response time anomalies by simultaneously considering the trace structure and the latency of the services.

Our evaluation with data from a real-world production cloud showed that our multimodal LSTM approach achieved over 90% accuracy in multiple experiments, outperforming the single-modality and the baseline dense neural networks, although the single-modality LSTM yielded comparable results in structural anomaly detection.

Our approach paves the way for development of new techniques that simultaneously consider application logs, resource metrics, or other observability data to create a joint representation of states to enable the anomaly detection in large-scale complex microservice systems. These achievements are fundamental for the development of zero-touch AIOps solutions for the automated anomaly detection, root-cause analysis, and remediation.

References

[1] F. Schmidt, A. Gulenko, M. Wallschlger, A. Acker, V. Hennig, F. Liu, and O. Kao, "Iftm - unsupervised anomaly detection for virtualized network function services," in *2018 IEEE International Conference on Web Services (ICWS)*, July 2018, pp. 187–194.

[2] A. Gulenko, F. Schmidt, A. Acker, M. Wallschlager, O. Kao, and F. Liu, "Detecting anomalous behavior of black-box services modeled with distance-based online clustering," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, Jul 2018, pp. 912–915.

[3] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.

[4] F. Schmidt, F. Suri-Payer, A. Gulenko, M. Wallschlger, A. Acker, and O. Kao, "Unsupervised anomaly event detection for cloud monitoring using online arima," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Dec 2018, pp. 71–76.

[5] D. Battre, N. Frejnik, S. Goel, O. Kao, and D. Warneke, "Evaluation of network topology inference in opaque compute clouds through end-to-end measurements," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 17–24.

[6] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," Google, Inc., Tech. Rep., 2010. [Online]. Available: https://research.google.com/archive/papers/dapper-2010-1.pdf

[7] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 IEEE International Conference on Data Mining (ICDM)*, 2009, pp. 149–158.

[8] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *Advances in neural information processing systems*, 2012, pp. 2222–2230.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[11] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, speech and signal processing (ICASSP)*, 2013, pp. 6645–6649.

[12] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *ESANN*, 2015.

[13] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 130–139.

[14] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, ser. MLCS'18. New York, NY, USA: ACM, 2018, pp. 1:1–1:8.

[15] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 689–696.

[16] OpenZipkin, "openzipkin/zipkin," 2018. [Online]. Available: https://github.com/openzipkin/zipkin

[17] N. M. Nasrabadi, "Pattern recognition and machine learning," *Journal of electronic imaging*, vol. 16, no. 4, p. 049901, 2007.

[18] F. Chollet *et al.*, "Keras," https://keras.io, 2018.

[19] A. Shrivastwa, S. Sarat, K. Jackson, C. Bunch, E. Sigler, and T. Campbell, *OpenStack: Building a Cloud Environment*. Packt Publishing, 2016.

[20] R. Ricci, E. Eide, and C. Team, "Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications," *The magazine of USENIX & SAGE*, vol. 39, no. 6, pp. 36–38, 2014.