Università della Svizzera italiana

Faculty of Informatics

Research Prospectus

June 17, 2021

Prospectus

Rahim Heydarov

Abstract

In this PhD, our global goal is to work on finding approaches and designing solutions for online failure prediction and fault localisation (FPFL) in complex software systems.
The nature of complex software systems makes failures in them inevitable, which often leads to rather high costs. Moreover, due to the indeterminism and a phenomenon of emergent behaviour [9], inherent in complex systems, their behaviour is not easy to predict by analysing the behaviour of their individual components, which makes the classical reductionist approach to testing useless in this context.
According to our findings, the most promising FPFL approaches in complex systems use statistical analysis and machine learning techniques to identify anomalies and correlate them with potential failures and fault resources. These techniques have their own specific limitations in the face of challenges imposed by the modern complex systems, which usually are overcome by using various combinations of the techniques and strategies.
The main challenges are of FPFL in modern complex systems are: (1) lack of labeled data for training; (2) typical challenges of complex systems (dynamic interdependency, dynamic anomaly characteristics caused mainly by the volatile load contexts of the application, nature of data); (3) dynamic execution context of the underlying environment (operating context). The first challenge often make the scenarios that exploit models pre-trained on labelled data, where labels distinguish normal behaviour from the abnormal one, not realistic in real world applications due to impossibility to determine all probable fault types/patterns, high cost of massive fault injection experiments and data collection. Most state-of-the-art solutions address this challenge by use of the profile-based strategy exploiting models pre-trained in unsupervised manner on data of normal, dominantly correct, system behaviour and detect anomalies by comparing the observed data to the baseline model. However, using a profile-based strategy is not a panacea. The effectiveness of that approaches is being dramatically decreased in a dynamic context, when the notion of normal and anomalous behaviour changes depending on the application workload fluctuations (challenge 2) and/or changes in underlying operating context (challenge 3).
In our work we aim to address these challenges, starting from elaboration on the state-of-the-art solutions design by our team recently to improve their effectiveness (challenges 1 and 2), and proceeding with working on the new effective (high accuracy) and efficient (low level of load on target system) approaches that address the challenges imposed by the dynamic operating context (challenge 3) and evaluate them on a wide range of possible fault types and patterns.

Research Advisor Prof. Research Advisor	Research Co-advisors Prof. Research Co-Advisor1, Prof. Research Co-Advisor2
Academic Advisor Prof. Academic Advisor	Review Committee Prof. Committee Member1, Prof. Committee Member2

Research Advisor's approval (Prof. Research Advisor): Date: .....

PhD Director's approval (Prof. Binder /Santini): Date: .....

1 Introduction

Unexpected behaviour has a significant impact on the smooth operation of systems and causes costly penalties due to the failures. Complexity of modern large systems, stemming from (1) the number and types of relationships between their components (that are sometimes operationally and managerially independent [55]) and between the system and its environment that change, (2) inherent indeterminism, and (3) a phenomenon of emergent behaviour [9], makes issues inevitable and traditional testing methods (especially those based on reductionist approach) not effective. The components of complex systems can be deployed on virtual machines or containers that can be migrated from one physical node to another within and across data centres. Applications can be distributed and/or multi-tier systems deployed on dedicated or shared server environments, doing computations across distributed nodes. FPFL in such systems is a difficult task since the issues may have many sources, maybe hidden in any one or more of the components, may emerge from the internal or system-environment interactions, and may propagate throughout the system due to high level of interdependencies.

The major challenges of FPFL solutions in modern complex systems are:

1. Limited knowledge of the system abnormal behaviour caused by the unavailability and/or the high cost of generating and/or collecting of labeled data for training.

2. Intrinsic specifics of complex systems:

   (a) Dynamic behaviour characteristics of applications - defining a priori all possible normal behaviour types is not always feasible. The notion of normality and its characteristics vary widely across application's load contexts.

   (b) Dynamic interdependency among application's components and between underlying resources - multiple interdependent application components deployed in data center servers with heterogeneous and interdependent resources.

   (c) Nature of data - different formats and semantics causes the presence of noise when the normal values may be similar to anomalies resulting in high false-positive detection rates. This category also includes a problem related to the high-dimensional metric space, where the data points are sparse and algorithms using the notion of proximity fail to retain their effectiveness (curse of dimensionality [27]).

3. Dynamic execution context (or operating context) of the underlying environment - resource sharing and dynamic resource allocation further aggravate the problem of dynamism in behavioural characteristics and interdependency of components in complex systems. Frequent performance variations exhibited by applications can often be caused by the fluctuations in the execution context of the underlying environment. Hence, failure prediction approaches for complex systems should consider not only the current state of the system resources and dynamics of the target application's behaviour but also must take into account the elasticity of the underlying environmental execution context.

To address the first challenge, sometimes the data with pre-seeded faults is used to train the model that later exploits the faults signatures to detect abnormal behaviour. The limitation of this approach is that it requires significant human efforts and can only handle previously known anomalies (has a significant false-negative rate). To tackle this problem, the majority of modern solutions uses a profile-based learning strategy, where the key idea is to observe the system behaviour over a certain period to make sense of underlying system dynamics, build a baseline model (can be a probability density function, a neural network with adjusted wights, clustered representation of dominant behaviour patterns of the system, etc.) and then, in online mode, define the deviation of new observations from the baseline profile. This deviation usually compared to a certain threshold value (predefined or dynamic) to reveal anomalous states. The baseline models can be built statically in the offline phase using data collected while system normal execution or while execution with faults seeded, or dynamically, in the online phase based on observed historical data. At the same time, besides the fact that a training data is not always available a priory, a pure profile-based approaches also have certain limitations (mostly caused by the challenges 2 and 3) that affect their accuracy (often conditioned by the high false-positive prediction rate). That limitations arise from relying on the the following premisses:

1. The distribution of metric measurements is static which is not always true considering dynamic behavioural pattern of the applications and their execution context. Baseline models pre-built in offline mode exploiting a priory available data set become soon obsolete and not valid for an explanation of the changed behaviour of the application.

2. What constitutes normal and abnormal behaviour is well delineated. Many solutions rely on thresholding with normality assumption which is not always relevant in the case of complex systems with its inherent dynamic anomaly characteristics. Additionally, the existence of noise in data makes false alarms even more frequent.

Thus, to address the accuracy-related limitations the profile-based FLFP approaches must be context-adaptive, which particularly means that solutions have to continuously retrain their models on real-time observed data to keep (1) the behavioural profile of the application and underlying system, as well as (2) the notion of normality, in an actual state. This, in its turn, rises the problems related to: (1) unacceptability of an extensive re-training (which is sometime is unavoidable) in real-time mode, (2) the generation of high runtime overhead that affects the normal execution of the systems under observation, and (3) horizontal scaling issues (handling amounts of KPIs).

2 State-of-the-Art

The key expectation from the FPFL techniques is the balance of high detection rates with low false-alarm rates while adapting rapidly to changes in system's performance. There are different strategies and meythods exist in the domain of FPFL for complex systems. The most promising ones exploit statistical analysis or machine learning methods. The choice of strategy is greatly influenced by the system observability, detection mode (real-time or post-mortem), availability of labeled data, dynamics of the application workload and the underlying context, nature of data.

2.1 Strategies

1. Signature-based approaches use prior knowledge about the characteristics of each kind of anomaly to identify previously known incidents. The limitation of such approaches is that they can detect only known fault types.

In [4], a time-series of each observed metric in a particular epoch (time interval) used to compute the medians of the measured values. Then, based on the past values of each median, the current values of the medians of each metric characterised as high, low, or normal, and these categories combined into the Epoch Summary Vector. Then, using the ML feature selection technique, the relevant metrics are selected for building the Epoch Fingerprint (subspace of the Epoch Summary Vector). Since most anomalies span multiple epochs, the consecutive Epoch Fingerprints combined into Anomaly Fingerprint by averaging the corresponding epoch fingerprints, thus summarising them across time. After that, the anomaly fingerprint is compared to already labeled by the administrator anomaly fingerprints to identify the underlying problem. The identicality of anomaly fingerprints is defined by thresholding of the Euclidean distance between them.

2. Profile-based detection strategy implies creation of representation of a dominant (typically normal) behaviour of the system to detect anomalies using deviations of the new observations with respect to this representation. Some methods of prediction of an unexpected behaviour imply (1) defining of the probability of the new observation to belong to the density distribution of the data corresponding to the normal behaviour and comparing this probability with a certain threshold; others imply (2) making a prediction using the model trained on data corresponding to the normal behaviour and comparing the deviation between the predicted and observed values with a certain threshold. Profile-based approaches often have higher false alarm rates, however they are more flexible than signature-based ones due to the ability to detect unknown anomalies.

In [39] a profile-based approach for network anomaly detection is presented. A modification of Ant Colony Optimisation meta-heuristic (based on the principles of swarm intelligence, inspired by the foraging behaviour of bio ant colony with its ability to find the shortest path between the nest and food source) [40], [41] is used as a clustering approach (to create a representation of normal behaviour) which seeks optimal solutions to grouping data through self-organised agents. Ants, using statistics and probabilities, travel through the search space represented by a graph. These agents are attracted to more favourable locations to optimise an objective function, in other words, those in which the concentration of pheromone deposited by ants which previously went through the same path is higher. In this paper, we assume that the paths are formed between the center of a cluster (centroid). The algorithm aims to optimise the efficiency of clustering, minimising the objective function value (finding the shortest paths to the centroids of the clusters), ensuring that each ow will be grouped to the best cluster (with a known number of clusters). Steps on each iteration: Build solutions (movement of ants by the states of the problem); Local Search (evaluation of solutions created through a local search); Pheromone Update.

3. Observational detection strategy implies that applications are observed through direct experimentation (in

contrast to exploiting baseline profiles), followed by an in-depth analysis of observed anomalies and root-cause identification. Because this approach depends mostly on experiential knowledge, it yields high accuracy in detecting known and unknown anomalies.

In [5] the run-time fault root-cause solution is presented. It makes use of online profiling and triggered when some of the transactions start presenting symptoms of performance anomaly. The first step is to detect if a performance variation is due to a workload change or it is an anomaly. This detection is done by measuring the correlation (Pearson correlation coefficient [6] is used) between the response time and the number of user transactions processed. Considering that the workload and response time might not be fully aligned (synchronized), an additional analysis based on the Dynamic Time Warping algorithm [7] is performed (Measures similarity between two sequences which may vary in time or speed. Keeps track of the distance necessary to keep them aligned). Both a sudden decrease of correlation and an increase in the distance to keep the workload and response time aligned is interpreted as a performance anomaly. Then, the solution checks whether the anomaly is caused by the application or the server change. This time the correlations between the aggregated workload and the application and server metrics are measured. The third step is to determine if the extra response time of a given transaction is caused by the application components or some remote service changes. The significance of the relation between the different metrics (estimators) and the total response time is estimated by use of ANOVA (calculates the F statistics and then the p-value or difference between means of response times obtained by use of different estimators) and the effect-size of estimators on the response time is measured by the coefficient of determination (R2).

4. Knowledge-based detection strategy implies an identification of performance issues and their causes based on historical records of previously observed anomalies a dynamic store (knowledge base) where definitions of known anomalies, their possible root-causes are maintained. The detection of new issues often triggers an update of the knowledge base. Although there exists some similarity between knowledge-based and signature-based detections, the generation of rules and definitions does not necessarily have to be entirely at run-time in the former.

In [8] the framework for automated performance bottleneck detection is presented. The rules for the detection of different bottlenecks are stored (added/modified) in the frameworks DB by its user. A bottleneck in DB is defined as a condition, called a rule, on a set of metrics. Each bottleneck is classified into one (or more) of the five dimensions (CPU, Memory, I/O, Communication, Threads). The BDE receives the metrics collected during observation, evaluates the rules, and composes a bottleneck description for all bottlenecks whose rules evaluate to be true (name, the area of code where it was evaluated).

5. Flow and dependency analysis strategy considers studying the ow of communication across components in distributed applications to identify anomalies. This approach typically involves real-time collection and analysis of traffic data (such as SNMP and TCP packets). To detect anomalies and their causes, frequency, correlation, and causal path analysis are usually performed.

In [10], the presented debugging tool (Spectroscope) aims to identify performance problems in distributed systems and analyse their root causes. It searches for the causes of changes in request execution performance by comparing the execution flows of those requests. It categories the flows to (1) Precursors (request flows corresponding to the executions of the non-problem period) and Mutations (request flows of the executions of problem periods). Identifying mutations and comparing them to their precursors helps localise sources of change and gives insight into their effects. Mutations are divided into two types: response time mutations (differ from the corresponding precursor only by response time), and structural mutations (requests that take different paths through the system in the problem period). Spectroscope groups identically structured requests into unique categories and use them as the basic unit for comparing request flows. To identify response-time mutations, for each category distributions of response times for the non-problem period and the problem period are extracted and input into the Kolmogorov-Smirnov two-sample non-parametric hypothesis test [11]. The category is marked as containing response-time mutations if the test rejects the null hypothesis. To identify the components or interactions responsible for the mutation, the Spectroscope extracts the critical path (the path of the request on which response time depends), which is the same for both precursors and mutations in case of response time mutations and runs the same hypothesis test on the edge latency distributions. Edges for which the null hypothesis is rejected are marked as fault locations. To identify structural mutations, Spectroscope assumes a similar workload was run in both the non-problem period and the problem period. As such, it is reasonable to expect that an increase in the number of requests that take one path in the problem period should correspond to a decrease in the number of requests that take other paths (i.e. the increase in one category should be balanced by the decrease in another category). Thus, if that or another category contains more requests from the problem period than requests from the non-problem one (difference compared with pre-defined

threshold), then this category is labeled as one which contains mutations. Otherwise, this category is labeled as one which contains precursors.

2.2 Statistical methods

Statistical methods can be categorised on parametric and non-parametric based on the assumptions they make regarding the underlying data. Parametric statistical techniques (ANOVA tests, Pearson correlation, t-tests) assume that some characteristics of the data are known a priori or can be inferred. For example, assuming that the probability density of a performance metric follows a Gaussian distribution. Nonparametric methods (CUSUM, Spearman correlation, Kruskal-Wallis, and Wilcoxon's test) require little or no assumptions about the underlying nature of the data.

1. Gaussian-based detection - exploit the assumption that underlying data distribution is normal, detects anomalous data points based on the distance from the distribution mean. The density distribution may also be exploited for detecting anomalous data points in GMM -parametric models of the probability distribution of continuous random variables estimated using the Expectation-Maximization (EM) algorithm [13, 14].

2. Regression analysis investigates relationships between performance metrics and quantifies the statistical significance of such relationships. The goal is to estimate the set of model parameters that minimises the absolute or the squared error. The magnitude of the residuals used to determine an anomaly score of new observations. New observations with deviation falling outside the confidence interval produced by the model may be classified as anomalous. Commonly used algorithms for estimating model parameters include Ordinary Least Squares (OLS), Least Angle (LA), and Recursive Least Square (RLS) [15].

In [38], the performance diagnostic framework (DAPA) is presented, which aims to help system administrators to analyse the application performance anomalies and identify potential causes of SLA violations in virtualised environments. The systems metrics considered as estimation variables and the observed application response time as the response variable. DAPA constructs a series of models using non-overlapping time windows that span across time between 2 hours before potential SLA violation and 1 hour after real SLA violation. The constructed models may exhibit different characteristics, however, those from the potential SLA violation phase may demonstrate characteristics of both SLA compliance and SLA violation. These models are then classified into different categories representing distinct system states (by use of k-means clustering with 2 centroids). Each model is represented by the vector of its regression coefficients, Euclidean distance is used as the distance measure. The final stage involves the aggregation of all the sample data belonging to the SLA violation cluster and creating a parsimonious regression model (least angle regression algorithm LARS - used) selecting the estimators with the highest explanatory predictive power. The top selected metrics from this model are then presented to the system administrator as suspicious system attributes.

3. Correlation Analysis - quantifies the degree of association between performance metrics. Commonly used techniques to estimate the correlation are the Pearson, the Kendal rank, and the Spearman algorithms [16].

The framework presented in [16] is used to distinguish the natural cause of unexpected behaviour (workload variations, upgrades) from the internal anomaly that may end up in a failure. In its implementation, the Pearsons r is used (1) by the performance analysis module to describe the degree of association between the response time of a given transaction and the number of processed transactions of the same type. A lower degree is a symptom of performance anomaly, and (2) by the anomaly detection module to describe the degree of association between the number of concurrent users per period and the collected parameters. A lower degree explains which parameter is mostly correlated with a detected performance anomaly.

4. Statistical process control (SPC) [17], is a quality control method widely used to monitor production processes for early detection of undesirable variation in process output. SPC provides a set of control charts, such as CUSUM, Shewhart charts, for monitoring process stability and variation.

In [18] two unsupervised incremental adapting (evolving with the data using limited parameter settings or offline retraining) learning techniques are proposed to address the problem of black-box real-time contextual anomalies detection in performance metric streams in the Internet services domain. This work addresses the issue of specificity of the failure prediction solutions due to their (1) rigid statistical assumptions and (2) targeting offline scenarios of model training using priorly collected data. Such approaches may result in high false-alarm rates when applied in online scenarios as the models become obsolete. These issues raise the need for techniques that can adapt to the evolving metric stream behaviour. Both proposed techniques (BAD - Behavior-based Anomaly Detection, and PAD -

Prediction-based Anomaly Detection) follow a two-step strategy - first, the underlying temporal property of the most recent behaviour of the stream is estimated, second, the statistically robust control charts applied to recognise deviation of new observation from the baseline. BAD scenario: (1) tracking the underlying statistical behaviour of the most recent period and generation of the PDF using KDE; (2) estimation of the probability density of current observation within this PDF; (3) construction of Shewhart Control chart (from the densities of observations) using the mean and the standard deviation of the PDF; (4) deriving of an adaptive threshold of probability density based on the lower side of the control chart; (5) comparing of the probability density of current observation with the threshold obtained. PAD scenario: (1) building of an adaptive Cubic Spline model (computationally fast and capable to pick up complex non-linear shapes) from observation of the most recent period. Coefficients of a spline model estimated via ordinary least squares (OLS) algorithm. (2) making a prediction for the current step using the built cubic spline (3) estimation of the prediction deviation (4) building an Exponential Weighted Moving Average control chart from the residuals of the most recent period; (5) alarm an anomaly in case the prediction deviation is out of the control limits of the EWMA.

Unlike statistical detection, learning techniques do not make assumptions about the underlying distribution of data. Commonly used learning techniques in literature: (1) Classification-based Techniques (SVM, ANN, Decision trees, Bayesian networks); (2) Neighbour-based Techniques (KNN [24]; KNNW [25]; LOF [26]); (3) Clustering-based Techniques (K-means, Expectation Maximisation (EM), SOM [34]); (4) Subspace-Based Detection (PCA, ICA, Autoencoders); (5) Ensemble-Based Detection (integration of results of various detection techniques and / or data subsets to achieve a consensus).

2.3 ML-based methods

1. Supervised learning - each data instance in a training dataset is assumed to belong to one of several classes. The goal is to build a generalised model that captures the relationship between the feature set and each class during the training phase. Well suited for recognizing known anomalies. Usage in dynamic environments is hampered by the cost of retraining due to the dynamic reconfiguration of application components and change in underlying execution environments.

In [29] LSTM RNN [30] is used for real-time detection of collective anomalies in the network intrusion analysis domain. LSTM model is trained with normal time-series data, adjusting its weights to obtain the ability to remember the context of the points in the training time-series in order to predict a coherent output in agreement with the context of the test sample. In runtime model also must decide whether a set of inputs within a number of the latest time steps forms a collective anomaly. The prediction deviations are measured within a certain period and collected in a circular array used to represent the level of the anomaly of the latest time steps (to consider the ensemble of points simultaneously). By analysing the circular array at every time step, the possibility of facing a collective anomaly is evaluated (based on predefined during the evaluation step threshold).

In [36] LSTM is used in a framework (DeepLog) for online anomaly detection from a log key sequence. DeepLog learns log patterns from normal execution during the learning phase (small training data set that consists of a sequence of normal log entries). After the training, it can recognise normal log sequences and can be used for online anomaly detection over incoming log entries in a streaming fashion. The key intuition behind the design of DeepLog is from natural language processing: log entries interpreted as an element of a sequence that follows certain patterns and grammar rules. DeepLog also provides a mechanism for a user to provide feedback to use a false positive to adjust its weights. DeepLog incrementally updates the model's weights to incorporate and adapt to new log patterns.

2. Unsupervised learning - requires no labeled training data (in some cases no training data at all). In unsupervised learning, in contrast to supervised one, no specific output value is provided. Instead, one tries to infer some underlying structure from the inputs. For instance, in unsupervised clustering, the goal is to infer a mapping from the given inputs (e.g. vectors of real numbers) to groups such that similar inputs are mapped to the same group [21]. The objective is to discover hidden patterns or regularities in the data. The algorithms cluster the input data into classes based solely on their statistical properties, no assumption is made of the distribution of the underlying data. For improved accuracy, it expected that normal data instances are more frequent in the dataset than abnormal instances. Particularly suitable for detecting unknown anomalies in cloud data centres where the precise definition of anomaly characteristics may not always exist.

Examples of unsupervised learning algorithms: (1) Clustering algorithms (Hierarchical clustering, k-means, GMM (EM), SOM; (2) Density-based algorithms - KNN, LOF, Cell-based algorithm; (3) Blind signal separation algorithms - PCA, ICA; (4) ANNs - AE (particularly RBM), Deep Belief Nets, VAE [45], GAN [46] paradigm based approaches.

In [33], HTM (Hierarchical Temporal Memory) network is used in anomaly detection in the streaming data domain. HTM network, ML algorithm derived from neuroscience, continuously learns and models the spatiotemporal characteristics of the time series data to predict the values of the next time step. Due to the continuous learning nature of HTMs, changes in the underlying system are handled gracefully making the resulting anomaly detection solution tolerant to noisy data and continuously adapting to changes in its statistics. After receiving an input vector of streaming data, the HTM component generates 2 vectors of an internal state representation of the current step: (1) the sparse binary representation of the current input vector and (2) the sparse binary representation of the prediction of the input for the next step. Then, a raw anomaly score of the current step computed as a deviation between the predicted input and the actual one. After that, the likelihood of abnormality of the current state is computed as a probability density of the current raw anomaly score in a rolling normal distribution of the last prediction errors (where the sample mean and variance are continuously updated from error values of the latest steps). This approach is useful to handle a problem of the dynamic context of the time-series where instantaneous predictions are often incorrect and thresholding the prediction error directly would lead to many false positives.

In [20], the authors present a black-box unsupervised behaviour learning (UBL) solution for virtualised cloud environments. UBL leverages a Self Organising Map (SOM) [34] which is capable of capturing complex system behaviour while being computationally less expensive than comparable approaches such as kNN [35]. During the training phase, SOM maps a high dimensional input space into a 2-dimensional map which represents a generalisation of the whole measurement vector space and can capture the normal system behaviours under different workloads. After learning, frequently trained neurones (centroids) will have modified the weight vector values of their neighbour neurones with the same input measurement vectors. As a result, the weight vectors of the neurones that are frequently trained will look similar to the weight vectors of their neighbour neurones. Since systems are usually in the normal state, neurones representing a normal state will be more frequently trained than the neurones representing a pre-failure or failure state. Thus, there will be a dominance of clusters representing different normal system behaviours. Then, in order to decide which of the system state represented by each cluster (normal, pre-failure, or failure), UBL calculates a neighbourhood area size for each neurone. If the neighbourhood area size is small, we know that the neurone is in a tight cluster of neurones, meaning the neurone is normal. During runtime, each measurement vector is mapped to that or another neurone using the same Euclidean distance, and the inputs state is characterised by the state of the neurone it is mapped to. UBL also supports incremental updates that can continuously adjust the SOM with new measurement vectors. The basic idea of anomaly component localisation in this work is to look at the difference between anomalous and normal neurones and output the metrics that differ most as faulty ones. Once a set of normal neurones has been found, the difference between the individual metric values of each normal neurone and those of the anomalous neurone is computed. After that, the metric differences are sorted from the highest to the lowest to determine a ranking order. After this process completes, there will be Q metric ranking lists which are then examined to determine a final order. UBL achieves scalable behaviour learning by virtualising and distributing the learning tasks among distributed hosts. For this purpose, UBL monitors the residual resources on each host by aggregating the resource consumption of all the VMs running on the host. If the available residual resources are insufficient, live migration is employed to move the learning VM to a host with sufficient residual resources.

In [37], the light-weight black-box runtime IaaS cloud-oriented performance anomaly detection/localisation tool presented (PerfCompass). It traces kernel-level system calls (low overhead) and performs real-time analysis to determine if the anomaly caused by internal (applications software bugs) or external fault (interference from other co-located applications, improper resource allocations). Thus, it is capable to diagnose previously unseen anomalies in a black-box fashion. PerfCompass first extracts different groups of closely related system calls (execution units), from the continuous raw system call traces. An execution unit is said to be affected by the fault in case the solution detects any outlier in either the execution time or in the frequency (based on the deviation from the mean) for any of the unit's system calls. Then, for fault localisation purposes, the fault onset time (interval between the unit start and anomaly detection time) is calculated for each affected execution unit. If the fault onset time is lower than a predefined threshold, the unit is considered as affected directly. Then, the total share of directly affected units is calculated (fault impact factor). If the value of the fault impact factor is close to 100 percent, the source of the fault is considered as external. Otherwise, the fault onset time dispersion is calculated (the standard deviation of the fault onset time among all the affected units). If a large fault onset time dispersion is observed, the system infers that the fault is an internal one. The idea is that internal fault is likely to directly affect a subset of threads executing the buggy code and then indirectly affect other threads that communicate with the directly affected threads.

3. Semi-supervised learning implies harnessing the large amounts of unlabelled data (which is available in many cases) in combination with typically smaller sets of labeled data [21]. Supervised learning methods can be applied

(with certain assumptions about its distribution) to scenarios where a lack of labeled data exists or if the unlabelled data can provide additional information to improve the prediction/classification model.

An example of semisupervised learning algorithms is wrapper methods. They utilise one or more supervised base learners and iteratively train these with the original labeled data as well as previously unlabelled data that is augmented with predictions from earlier iterations of the learners (pseudo-labeled data). The procedure usually consists of two alternating steps of training and pseudo-labelling. In the training step, one or more supervised classifiers are trained (using original labeled data plus, pseudo-labeled data from previous iterations). In the pseudo-labelling step, the resulting classifiers are used to infer labels for the previously unlabelled objects; the data points for which the learners were most confident of their predictions are pseudo-labeled for use in the next iteration.

In [19], the self-evolving anomaly detector (SEAD) for the cloud is presented. It does not require extensive training on previously labeled data and is capable to catch unknown anomalies. It self-evolves by learning from newly generated and verified (by the system administrator) detection results. SEAD includes two components - the anomaly detector and the working dataset. Anomaly detector is based on the continuously re-trained classification models (SVM, OCSVM). For a new data record, the detector calculates an abnormality score. If the score is above a threshold, a warning is triggered with the type of abnormality and the cloud operators verify and label the event. This labeled data point is selectively included in the working dataset and the anomaly detectors model is being retrained.

2.4 Recent trends

Recent failure prediction methods have mainly been focused on finding latent representations of the input data using deep neural networks. The most prominent example of such a method is autoencoders - a class of neural networks with one or more hidden layers that has the objective of reconstructing its input. Autoencoders attempt to find a lower-dimensional representation of the input space without sacrificing substantial amounts of information. Thus, they inherently act on the assumption that the input space contains lower-dimensional substructures on which the data lie [21]. Autoencoders can use different types of layers to learn meaningful representation of the data. For example, in [51], the convolutional neural networks (CNNs) are used as stacked layers of deep autoencoder for anomaly detection to automatically identify abnormal behaviours in wastewater systems. In [50] stacked autoencoder model (SAE), that is constructed by stacking a sequence of single-layer sparse autoencoders layer by layer, is used in combination with wavelet transforms (WT) and long-short term memory (LSTM) models for stock price forecasting. SAE is used to extract the core features of the time-series data de-noised by WT. The output of SAE is fed to the LSTM model for forecasting the closing price.

One of the recent anomaly detection approaches based on autoencoders was proposed by our team in [43]. The proposed approach, called EmBeD, is a runtime anomaly detection framework for cloud systems based on unsupervised model training techniques. The anomaly detector is based on the hypotheses that (1) erroneous states of the system reflect in collective anomalies and (2) the anomalous behaviours reflect in anomalous values of the Gibbs free energy (intuitively can be understood as an approximation of the status of the system). EmBeD detects anomalies at runtime using the baseline model which represents the distribution of normal behaviour as a function of time with a standard deviation. This baseline model built using the values of the Gibbs free energy function computed while training the Restricted Boltzmann Machine (RBM, which is a special case of autoencoder) with KPI values monitored during normal execution. The anomaly detector signals incoming failures as anomalies in the free energy, that is, deviations from the baseline model produced during training. While training an RBM computes the joint probability distribution of the input data with respect to the set of parameters (the weights W of the edges and the biases) by associating a scalar energy function to visible units combination after each back-propagation (log-likelihood function based on the free energy function value). The most likely combination of the visible units corresponds to the combination that best approximates the input vector and is obtained by minimising the free energy during the weights adjusting iterations.

Besides EmBeD, there are 2 other state-of-the-art failure prediction approaches that were presented by our team within the last several years - PreMiSE[48] and Loud[54]. PreMiSE is a lightweight approach that uses a combination of profile- and signature-based techniques for failure prediction and fault identification/localisation in a multi-tier distributed systems. PreMiSE exploits: (1) anomaly detection model trained offline (the anomaly detection component is represented by IBM ITOA-PI [67] Data Analytics solution) on the data collected under normal system execution (profile-based approach), and (2) fault identification/localisation model trained offline on the data with various faults seeded (signature-based approach). In offline phase the Anomaly Detector builds the model that represents a profile of correct behaviour of the system and a causality graph that represents a directed weighted

graph where nodes correspond to KPIs and edges represent causal relations among KPIs. In online phase Anomaly Detector identifies anomalies if (1) it detects KPIs with a value outside the acceptable variations encoded into the anomaly detection model or if (2) the causal relationships represented in the causality graph have broken.
Fault identification/localisation component is built on the model, pre-trained in offline mode using anomalies revealed by the anomaly detection model on data collected during system executions with seeded faults of the target types. In online phase Fault identification and localisation component uses the pre-trained signature-based model to distinguish between benign and failure-prone anomalies, to identify the faults and locate the resources likely responsible for the failure.
PreMiSE can predict failures with higher precision and less false positives than state-of-the-approach (IBM ITOA-PI), without incurring in extra execution costs on the target system (PreMiSE executes on a node independent from the target system, and limits the online interactions with the monitored applications to metric collection). Differently from state-of-the-art approach (IBM ITOA-PI), which only detects anomalies, PreMiSE can effectively identify the type of the possible failure and locate the related faults.The main limitation of PreMiSE is that the range of faults that can be predicted is limited by the kinds of faults used in the training phase.
LOUD is an approach for fault localisation in cloud systems, which is based on Granger causality concept and graph centrality indexes.
LOUD exploits: (1) the same anomaly detection approach that is used in PreMise (the anomaly detection component is represented by IBM ITOA-PI Data Analytics solution), where the model trained offline on the data collected under normal system execution (profile-based approach), and (2) the fault localisation mechanism, comprised of KPI analysis and Fault localisation components. LOUD approach implies usage of third-party component to classify the anomalies provided by the anomaly detection part to generate a failure alerts for the fault localisation module. When the failure alert is received, the KPI analysis component exploits the Granger causality graph (the directed weighted graph, built in offline phase by the IBM ITOA-PI Data Analytics, where nodes correspond to KPIs and edges represent causal relations among KPIs), and the set of anomalous KPIs (provided by the anomaly detector in online mode) to extract the propagation graph, which is the subgraph of the Granger Causality Graph that contains only the KPIs that were detected as anomalous at the current timestamp. In this way, the Propagation graph captures the causal relationships between the anomalous KPI, which, in our fault-localisation approach, is the basis to reason on which resources originated anomalies that directly or indirectly caused other anomalies across the system.
The Anomaly Resource Ranker identifies the likely faulty resources. It first ranks the anomalous KPIs according to their degree of centrality in the Propagation graph, then exploits the anomalies that appear in top part of the ranking to identify the faulty resources as the ones that originated most top-ranked anomalies (a faulty resource is likely to be present with multiple KPIs in the top part of the ranking). In contrast to PreMise, LOUD (1) exploits fault localisation model that is trained only on data collected under normal execution (without fault injection sessions used in signature-based approaches); (2) does not have an anomaly classification component. As well as PreMiSe, LOUD imposes negligible execution overhead being run on an independent machine, and good effectiveness, locating faults with an efficiency in line with the best state-of-the-art approaches.

Recently, Microservices oriented architectures became popular. Microservices are encapsulated services that, in contrast to virtualisation can be run very efficiently in parallel on the same host, therefore allowing for hundreds of Microservices running on a single host.They are usually deployed as individual containers within a containerised architecture and managed by a replication controller like Kubernetes such that they can easily be scaled up or down by adding or removing instances from the system. In contrast to services that are executed on a virtualised hosts, microservices are dynamic, small and independent and can be deployed and removed within seconds. At the same time, many of failure predictors are designed for the use in a static environment where the number of components is a known constant. Thus, in a microservice architecture, many failure predictors have become obsolete and new approaches have to be discovered. In [52], Hierarchical Online Failure Prediction (HORA) framework – failure prediction in container-based microservice oriented application - is proposed, that combines a failure prediction results of single components by considering the architecture of the whole system. The approach is based on the assumption that every container that is deployed on the same host can influence the behaviour of every other container within the same host and therefore there is some type of dependency between each container on the same host. The other type of dependency comes from the application-defined dependencies between the services. The failure prediction in the HORA is done in two steps: (1) failure prediction for every individual KPI using ARIMA forecast where the prediction deviation threshold is calculated by using a simple probability density function; (2) The combined forecast for each component by using the dependency graph from which it is possible to conclude the cause of the container interference problem. Experiments showed that containers can in fact be influenced by other containers that run completely independent from another in terms of software architecture. Thus, the operating context of the applications can be very unstable. The agility of distributed microservice environments where services are upgraded multiple times a day and where several versions of an application can be deployed at the same time, creates several new challenges.

As every update may change the response time behaviour of methods within applications, this could result in many false alarms of the application monitoring systems which use historical data and possibly forecasting approaches to detect anomalies. In [53], the following events considered that may have an impact on the false positive anomaly detection rate. **(1) Update-induced ramp-up anomalies** – an update of a service in a microservice environment usually means that it is replaced by a new version of the service. As the microservice should be available without interruptions, usually a new microservice instance is started before the old version is shut down. Due to the startup of a new service instance, increased load can be observed on the system. Such tasks affect the response time of a service which could lead to anomaly alerts if a classical anomaly detection approach would be used for monitoring. Due to the anomaly threshold, the increased response times caused by the initialisation phase of the microservice would be treated as anomalies. **(2) Update-induced behaviour changes** - an update introduced to the applications behaviour. For example, if an additional command is added to the source code of an application, the execution time of that command adds up to the total response time. This change may lead to an anomaly alert as the changed behaviour exceeds the defined threshold. The goal of the work is to reduce the false positive anomaly detection rate of existed anomaly detection solution in fast-changing microservice environments. This is achieved by taking event information (intended change of the environment, e.g., updates or restarts of microservices) into account when determining whether changes in the behaviour are actual anomalies. In case such update events occur, the anomaly score should be reconfigured for some time to allow the initialisation phases to be executed without raising an anomaly alert.

3 Objectives

The global objective of my PhD is finding effective and efficient approaches and designing solutions for failure prediction and fault localisation in complex systems. This objective is comprised of the following sub-goals:

1. To design a new solution that will improve over state-of-the-art techniques (PreMiSE [48], LOUD [54], IBM ITOA-PI [57]) by addressing their major limitations.
   The main limitation of PreMiSE is that it uses signature-based strategy to identify/localise faults. Thus, the range of identifiable faults is limited by those the model is trained on. The main limitation of LOUD is that it does not have the anomaly classification component that would distinguish the malign anomalies from the benign ones and would generate a failure alerts. Both approaches uses IBM ITOA-PI [57] solution for anomaly detection. LOUD uses IBM ITOA-PI also for construction of the Granger causality graph that models causal relations between the KPIs and used in online fault localisation phase.
   Requirements to the new solution: (1) must include data collection, anomaly detection, anomaly classification, and fault localisation components; (2) must be fault type agnostic (particularly, should not rely on signature-based strategy for fault localisation); (3) locate faults in static (without resource re-allocation mechanism) virtualised environment with an effectiveness and efficiency in line with the best state-of-the-art approaches.
   Thus, our aim in the first phase is: (1) To design and use our own anomaly detection component based on deep autoencoder model, that represents a pre-defined system behavioural profile, and pre-trained in offline unsupervised manner on the data collected under normal system execution; (2) To use our own implementation of anomaly classification component based on the one-class support vector machine technique to distinguish spurious anomalies from the symptoms of future failures. This will address the limitation of LOUD that does not have anomaly classification instrument and the limitation of PreMiSe which uses signature-based strategy for this purpose; (3) To use a fault localisation approach based on revealing of Granger causalities between the KPIs and exploiting centrality algorithms (the approach used in LOUD) to localise the system resources that are most likely responsible for the fault propagation. This will address the limitation of Premise which uses signature-based strategy to identify/localise the faults. In contrast to the implementation used in LOUD, we plan to design and use our own Granger causality graph building component. (4) To integrate the aforecited anomaly detection, anomaly classification and fault localisation components into a framework, augmented by the data collection component, perform experiments/evaluation in environment with static virtualised operative context. For this purpose we intend to deploy and configure a testbed on our local server machines in static virtualised environment to perform a series of (1) executions under normal conditions (to collect data for training of the anomaly detection and classification models) and (2) executions with injection of faults of various types/intensity patterns (to collect data for the evaluation of the approach).

2. To extend the functionality of the proposed approach to make it capable to accurately detect and classify anomalies, identify and locate the faults in a dynamically changing operative context.
   The dynamism of the operative context is conditioned by the elasticity of mechanism of resource allocation, which allows a provision/de-provision/re-configuration of underlying resources (i.e., VMs, containers [42])

insuring an uninterrupted operation mode of the system. This contextual volatility often leads to frequent changes in target application's metrics (both in KPI's values and correlations between KPIs), decreasing the applicability of the approaches that rely on baseline models of predefined system behaviour. Specifically, it becomes difficult for profile-based approaches to adequately identify deviations of new observations from normal, not anomalous, system states and to effectively conduct a root-cause analysis of the anomalous events. Thus, our goal in the second phase is to work on a class of solutions that (1) address the challenges of identifying and characterising execution contexts as they evolve over time and (2) accordingly adjust the internal representation of the system behaviour (profile). To conduct our experiments we plan to use a container-based environment, managed by systems for automating deployment, scaling, and reconfiguration of containerised applications, such as Kubernetes or Docker Swarm.

3. To study the effectiveness of our approach on a wide range of faults of various categories: network, resource leak, high overhead, container management, resource race, hardware issues, etc.

4 Work done

Within the first year of my PhD we have completed the first sub-goal of our global objective. The scope of work performed includes: (1) Set up of the static virtualised environment using Openstack; (2) Deployment and configuration of a testbed solution (Redis cluster) and implementation of the testbed management tools (custom implementation in Python); (3) Design of the data collection component, consisted of data monitoring instruments (ElasticSearch) and data retrieval/processing tools (custom implementation in Python). Data collection component is executed on an independent machine to avoid side effects on the target system, which is monitored with lightweight monitoring probes (ElasticSearch metricbeat agents). The metrics capture measurable aspects of the behaviour of the monitored system, for example, memory consumption or a number of requests served per minute. The resource can be any element of a system (a host, a virtual machine, a specific application); (4) Design of the traffic generation tool (custom implementation of client for the Redis cluster); (5) Design of the anomaly detection component based on a deep autoencoder model. The component was implemented in Python on top of the framework, developed by our team. It replaces the original component based on IBM ITOA-PI used in Loud and PreMiSe. The underlying anomaly detection model is trained on data of normal behaviour of the system (profile-based approach); (6) Design of the anomaly classification component that is built upon a One-class SVM-based classification model with RBF kernel. The component is implemented on top of the framework, developed by our team and used in Loud. The model captures the correct (albeit suspected as anomalous) behaviour while being trained offline on a set of spurious anomalies, generated by anomaly detection process performed on data collected from the execution under normal conditions. In online failure prediction phase the model discriminates spurious anomaly sets from the ones that can lead to failure; (7) Design of the component that builds a Granger causality graph (captures causal relations between KPIs that are used for fault localisation) using the time series of KPIs of normal behaviour (custom implementation in Python). This component used instead of the original component based on IBM ITOA-PI used in Loud; (8) Integration of the fault localisation component used in LOUD. We used a PageRank centrality algorithm to score the propagation graph's nodes (associated the the system's KPIs) according to both the number of incoming edges and the probability of anomalies to randomly spread through the graph (teleportation); (9) Long-term execution of the system under normal conditions (to simulate normal behaviour and to collect data for the training phase) and series of executions with faults seeded (to simulate a faulty behaviour to collect data for validation of the approach). We seeded faults of the memory leak, packet loss and CPU hog fault types following linear, random and exponential activation intensity patterns. We generated the workload with our traffic generator by using a classic traffic pattern also used for evaluation of PreMiSe[48]: higher number of calls in weekdays than during weekends, growing amount of calls at daytime and decreasing at nighttime, with peaks at 9am and 7pm; (10) Retrieval/processing of the collected data (11) Evaluation of the prototype and selection of the optimal hyper-parameter combination.
The designed solution consists of 2 subsystems: (1) Failure predictor and (2) Fault localiser. The Failure Prediction subsystem comprises of the data collection, anomaly detection, and anomaly classification components. The Fault Localiser is comprised of Granger causality graph builder and anomalous resource ranker. During the offline training phase the anomaly detection and anomaly classification models are trained, and the Granger causality graph is built. During the online prediction/localisation phase, the Failure Predictor (1) monitors the target system at constant intervals of 1 minute (data collector), checks the collected KPIs against the baseline models (2) to detect anomalies (anomaly detector), and (3) to classify the anomalous sets and generate a failure alert if necessary (anomaly classifier). When the Failure Predictor produces a failure alert, the Fault Localiser exploits the Granger causality graph and anomalous KPIs (received from the Failure Predictor) (4) to generate the propagation graph that is a subgraph of Granger causality graph with anomalous KPIs only (anomalous resource ranker) and uses the propagation graph (5) to rank the anomalous KPIs received from the Failure Predictor (anomalous resource ranker), and to localise the

fault at the resource-level (anomalous resource ranker).

The evaluation results show that our mechanism can identify faults on node-level with high accuracy and almost no computation overhead, without incurring in extra execution costs on the target system (since being executed on a node independent from the target system, and limiting the online interactions with the monitored applications to metric collection). Evaluation of the state-of-the-art solution (EmBeD) on the same data and comparison of the results with the results of our experiments did not reveal significant differences in the effectivenes/efficiency of these approaches. Thus, the first sub-goal of my PhD objective has been fully completed.

5 Work Plan

1. Year 1 (realisation of objective's goal 1): elaboration on the state-of-the-art techniques (PreMiSE [48], LOUD [54], IBM ITOA-PI [57]) to develop an approach and implement a solution to address their main limitations.
2. Years 2, 3 (realisation of objective's goal 2): Extension of the functionality of the proposed approach to the level of solutions, capable to effectively predict failures and identify the faults in a dynamically changing operative context.
3. Year 4 (realisation of objective's goal 3): Evaluation of the effectiveness of the proposed approach on a wide range of fault types and their intensity patterns

References

[1] Credit card fraud detection using autoencoders in h2o, 2019.

[2] Deep inside: Autoencoders, 2019.

[3] Deep learning book: Autoencoders, 2019.

[4] Ensemble learning in python, 2019.

[5] Outlier detection in streaming data, 2019.

[6] Sparse autoencoder. cs294a lecture notes, 2019.

[7] Understanding latent space in machine learning, 2019.

[8] C. C. Aggarwal. High-dimensional outlier detection: the subspace method. In *Outlier Analysis*, pages 149–184. Springer, 2017.

[9] C. C. Aggarwal and S. Y. Philip. An effective and efficient algorithm for high-dimensional outlier detection. *The VLDB journal*, 14(2):211–221, 2005.

[10] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[11] J. T. Andrews, E. J. Morton, and L. D. Griffin. Detecting anomalous data using auto-encoders. *International Journal of Machine Learning and Computing*, 6(1):21, 2016.

[12] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 15–27. Springer, 2002.

[13] J. P. Assendorp. *Deep learning for anomaly detection in multivariate time series data*. PhD thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2017.

[14] S. Barbhuiya, Z. Papazachos, P. Kilpatrick, and D. S. Nikolopoulos. Rads: Real-time anomaly detection system for cloud data centres. *arXiv preprint arXiv:1811.04481*, 2018.

[15] G. Bhattacharya, K. Ghosh, and A. S. Chowdhury. Outlier detection using neighborhood rank difference. *Pattern Recognition Letters*, 60:24–31, 2015.

[16] L. Bontemps, J. McDermott, N.-A. Le-Khac, et al. Collective anomaly detection based on long short-term memory recurrent neural networks. In *International Conference on Future Data and Security Engineering*, pages 141–152. Springer, 2016.

[17] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[18] T. S. Buda, B. Caglayan, and H. Assem. Deepad: A generic framework based on deep learning for time series anomaly detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 577–588. Springer, 2018.

[19] R. Chalapathy and S. Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[20] R. Chalapathy, A. K. Menon, and S. Chawla. Anomaly detection using one-class neural networks. *arXiv preprint arXiv:1802.06360*, 2018.

[21] R. Chalapathy, E. Toth, and S. Chawla. Group anomaly detection using deep generative models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 173–189. Springer, 2018.

[22] A. Das, F. Mueller, C. Siegel, and A. Vishnu. Desh: deep learning for system health prediction of lead times to failure in hpc. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 40–51, 2018.

[23] 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*, pages 1285–1298, 2017.

[24] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121–134, 2016.

[25] T. Ergen, A. H. Mirza, and S. S. Kozat. Unsupervised and semi-supervised anomaly detection with lstm neural networks. *arXiv preprint arXiv:1710.09207*, 2017.

[26] V. Hautamaki, I. Karkkainen, and P. Franti. Outlier detection using k-nearest neighbour graph. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 430–433. IEEE, 2004.

[27] D. M. Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[28] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[29] H. Huang, K. Mehrotra, and C. K. Mohan. Outlier detection using modified-ranks and other variants. 2011.

[30] H. Huang, K. Mehrotra, and C. K. Mohan. Rank-based outlier detection. *Journal of Statistical Computation and Simulation*, 83(3):518–531, 2013.

[31] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Loop: local outlier probabilities. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1649–1652, 2009.

[32] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Outlier detection in axis-parallel subspaces of high dimensional data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 831–838. Springer, 2009.

[33] H.-P. Kriegel, M. Schubert, and A. Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452, 2008.

[34] Z. Lan, Z. Zheng, and Y. Li. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):174–187, 2009.

[35] A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166, 2005.

[36] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[37] E. Muller, I. Assent, U. Steinhausen, and T. Seidl. Outrank: ranking outliers in high dimensional data. In *2008 IEEE 24th international conference on data engineering workshop*, pages 600–603. IEEE, 2008.

[38] E. Müller, M. Schiffer, and T. Seidl. Statistical selection of relevant subspace projections for outlier ranking. In *2011 IEEE 27th international conference on data engineering*, pages 434–445. IEEE, 2011.

[39] M. Radovanović, A. Nanopoulos, and M. Ivanović. Reverse nearest neighbors in unsupervised distance-based outlier detection. *IEEE transactions on knowledge and data engineering*, 27(5):1369–1382, 2014.

[40] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438, 2000.

[41] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402, 2018.

[42] B. Tang and H. He. A local density-based approach for outlier detection. *Neurocomputing*, 241:171–180, 2017.

[43] X. Xu, H. Liu, and M. Yao. Recent progress of anomaly detection. *Complexity*, 2019, 2019.

[44] L. Zhang, Z. He, and D. Lei. Shared nearest neighbors based outlier detection for biological se-quences. *International Journal of Digital Content Technology and Its Applications*, 6(12):1–10, 2012.

[45] A. Zimek, E. Schubert, and H.-P. Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012.