St. Pölten University of Applied Sciences

/informatik & security

/fh/// st.pölten

Log Anomalie Erkennung in WebApps

Implementierung eines Prototypen

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

eingereicht von

Christoph Eberhart

11708944

im Rahmen des

Studienganges Information Security an der Fachhochschule St. Pölten

Betreuung

Betreuer/in: Dipl.-Ing. Dr. Martin Vasko, BSc

Mitwirkung: -

St. Pölten, 12. September 2023

(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

Fachhochschule St. Pölten GmbH, Campus-Platz 1, A-3100 St. Pölten,T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E: office@fhstp.ac.at, I: www.fhstp.ac.at

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich das Thema dieser Arbeit bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Der Studierende/Absolvent räumt der FH St. Pölten das Recht ein, diese Arbeit für Lehre- und Forschungstätigkeiten zu verwenden und damit zu werben (z. B. bei der Projektevernissage, in Publikationen, auf der Homepage), wobei der Absolvent als Urheber zu nennen ist. Jegliche kommerzielle Verwertung/Nutzung bedarf einer weiteren Vereinbarung zwischen dem Studierenden/Absolventen und der FH St. Pölten.

Laab im Walde, 12.09.2023

Ort, Datum

Unterschrift

Kurzfassung

Unterschiedliche Systeme, Geräte und Applikationen erzeugen Logs, die das Verhalten bei den diversen Anwendungen, Software und Geräten aufzeichnen. Somit ist es möglich, aussagekräftige Informationen über das Verhalten zu generieren. Logs beinhalten unterschiedliche Informationen und können ein besseres Verständnis für ein bestimmtes Verhalten der Applikation liefern. Beispielsweise können Fehlfunktionen oder frühzeitige Vorhersagen zu möglichen Fehlern erkannt werden. Durch den Anstieg an der Entwicklung von Applikationen und den Gebrauch von Geräten steigt das Log-Volumen stetig an. Hierbei ist es notwendig, die Vorgänge zu automatisieren, damit eine ausreichende Analyse des Programmverhaltens aus den riesigen unstrukturierten Daten, funktionieren kann. Dabei sind computerbasierte Analysetechniken unerlässlich, um eine Diagnose von möglichem Fehlverhalten aus den Log-Dateien zu erstellen. Hier kommen unterschiedliche leistungsstarke Werkzeuge wie Machine Learning (ML) und Deep Learning (DL) zum Einsatz. Durch die Analyse und das Erkennen von möglichen Anomalien in den Logs können weitere Sicherheitsmaßnahmen ergriffen und mögliche Schäden verhindert werden.

Die Arbeit umfasst zwei Forschungsfragen. Im ersten Teil der Diplomarbeit lässt sich die folgende Forschungsfrage stellen: "Welche Ansätze zur Log-Anomalie-Erkennung existieren, und worin unterscheiden diese sich? ". Um diese Frage zu beantworten, wurden ausgewählte Ansätze in Bezug auf Log-Anomalie-Erkennung mithilfe von Deep Learning definiert und gegenübergestellt.
Die zweite Forschungsfrage "Wie kann man mithilfe von Tensorflow.js eine eigene Anwendung zur Log-Anomalien-Erkennung in Web-Apps implementieren?"wird beantwortet, indem im Zuge dieser Diplomarbeit ein Prototyp namens 'LogDeeptector' implementiert wurde. Sowie die Umsetzung und das Ergebnis wurde dokumentiert, und die daraus resultierenden Erkenntnisse der Implementierung werden angeführt.

Das Ergebnis der Arbeit weist eine Zusammenfassung über die ausgewählten Ansätze auf, die definiert und gegenübergestellt wurden. Jeder dieser Ansätze verwendet dabei eine andere Vorgehensweise in Bezug auf das Log-Parsing und der Auswahl von den Deep Learning Modellen, dabei lassen sich Vor- und Nachteile feststellen. Ein weiteres Ergebnis ist der Prototyp und die daraus abgeleiteten Erkenntnisse der Implementierung von einem neuronalen Netzwerk zur Log-Anomalie-Erkennung in Webapplikationen.

Abstract

Different systems, devices and applications generate logs that record the behaviour of various applications, software and devices. This makes it possible to generate meaningful information about the behaviour. Logs contain different information and can provide a better understanding of a particular application's behaviour. For example, malfunctions or early predictions of possible errors can be detected. Due to the increase in the development of applications and the use of devices, the log volume is constantly increasing. Here, it is necessary to automate the processes so that sufficient analysis of programme behaviour from the huge unstructured data can function. Computer-based analysis techniques are essential to diagnose possible misbehaviour from the log files. Various powerful tools such as Machine Learning (ML) and Deep Learning (DL) are used here. By analysing and detecting possible anomalies in the logs, further security measures can be taken and possible damage can be prevented.

The thesis comprises two research questions. In the first part of the thesis, the following research question can be posed: "What approaches to log anomaly detection exist, and how do they differ? ". To answer this question, selected approaches to log anomaly detection using Deep Learning were defined and compared. The second research question "How can you implement your own application for log anomaly detection in web apps using Tensorflow.js?" is answered by implementing a prototype called 'LogDeeptector' in the course of this thesis. As well as the implementation and the result was documented, and the resulting findings of the implementation are cited.

The result of the work shows a summary of the selected approaches, which were defined and compared. Each of these approaches uses a different approach with regard to log parsing and the selection of deep learning models, and advantages and disadvantages can be identified. Another result is the prototype and the findings derived from the implementation of a neural network for log anomaly detection in web apps.

Inhaltsverzeichnis

1 Einleitung

Sei es das Starten und Herunterfahren von Prozessen, der aktuelle Status des Systems, das Neustarten von Maschinen oder der Zugriff auf Ressourcen - diese routinemäßigen Systemoperationen werden in sogenannten Logs protokolliert. Es können jedoch auch fehlerhafte bzw. unerwünschte Systemzustände, wie beispielsweise fehlgeschlagene Operationen, Sicherheitsverstöße oder Probleme bei der Verfügbarkeit in den Logs aufgezeichnet werden. Entwicklerinnen und Entwickler sowie Betreiberinnen und Betreiber können somit die protokollierten Logs nutzen, um den Status zu untersuchen und die möglichen Ursachen zu verstehen. Dies ist für Systembetreiberinnen und -betreiber, die zeitnah Maßnahmen ergreifen müssen, damit Systemschäden zu verhindern oder zu verringern, von entscheidender Bedeutung. Um eine sachgerechte Überwachung von Systemen zu gewährleisten, sind Logs eine ergiebige Datenquelle mit unverzichtbaren Informationen. [1] [2] [3]

Moderne Systeme wachsen und verlagern sich zum verteilten Cloud-Computing. Diese groß angelegten Systeme ermöglichen Anwendungen, die einen hohen Verarbeitungsaufwand erfordern, z. B. Prognosen sowie Online-Dienste wie soziale Netzwerke, E-Commerce und Suchmaschinen. Viele dieser Systeme sind für den Dauerbetrieb ausgelegt und bedienen Millionen von Menschen weltweit. Aufgrund der großen Menge an unterschiedlichen Protokolldaten ist eine Analyse der Logdateien schwierig. Hierbei wird versucht, die weniger interessanten Logs von den möglicherweise interessanten zu trennen. Zudem ist eine möglichst schnelle Erkennung von Ursachen, z. B. Unterbrechungen von Prozessen oder Problemen in der Verfügbarkeit, wichtig, um ein schnelles Eingreifen zu ermöglichen, damit der Schaden gering bleibt. Ausfälle von Diensten können sehr teuer werden. Eine menschliche Analyse ist bei diesen Mengen unpraktisch, daher ist ein automatisierter Prozess erforderlich. [1] [2] [3]

Mit dem Anstieg der Mengen an Logs und der Menge an unterschiedlichen Systemen hat sich das automatische Erkennen von Anomalien in Log-Dateien zu einem eigenen wichtigen Forschungsbereich etabliert. Eine manuell erstellte Signatur sowie ein Algorithmus, um nach bestimmten Termen bzw. Semantiken in den Logs zu suchen, hat einen sehr begrenzten Anwendungsbereich und ist für die immer veränderbare Systemumgebung ungeeignet. Einige Methoden wie Clustering [4], Workflow Mining [5], statistische Ana-

lyse von Ereignisparametern [6], Zeitreihenanalyse zur Ermittlung von Änderungen der Ereignishäufigkeit [7] und viele weitere [8], [9], wurden in der Vergangenheit bereits vorgeschlagen. Nun versucht man das maschinelle Lernen zu übertreffen und Forscherinnen und Forscher haben begonnen Deep Learning mit neuronalen Netzen für die Anomalie Erkennung einzusetzen. Hierbei werden die Daten von den Logs für die Eingabe in das neuronale Netz vorbereitet, was nicht immer einfach ist, da sie unstrukturiert sind und möglicherweise untereinander komplexe Verbindungen haben. Eine weitere Herausforderung ist es, das passende Modell für den Anwendungszweck zu schreiben. Die Automatisierung verspricht eine ständige Überwachung der Systeme und eine schnelle Warnung vor Unregelmäßigkeiten, sodass sich Betreiberinnen und Betreiber sowie Entwicklerinnen und Entwickler ausschließlich auf die Behebung von Problemen konzentrieren können. Auch wenn geeignete Ansätze wie DeepLog, LogAnomaly und LogRobust zur Verfügung stehen, ist es wichtig zu verstehen, dass die Möglichkeit von Fehlalarmen besteht. Diese Alarme können eine Herausforderung für die Betreiberinnen und Betreiber darstellen. [1] [2] [3]

Das Ziel dieser Diplomarbeit ist es einen Überblick über bereits existierende Ansätze (DeepLog, LogAnomaly, LogRobust PLELog, CNN und TransLog) zu Log-Anomalie-Erkennung zu liefern. Durch eine Gegenüberstellung werden die Unterschiede zwischen den Ansätzen herauskristallisiert und die jeweiligen Vor- und Nachteile werden tabellarisch dargestellt.
Ein weiteres Ziel dieser Arbeit ist die Implementierung eines Prototyps zur Erkennung von Anomalien in den Log-Dateien von Webapplikationen. Dieser Implementierungsprozess wird Schritt für Schritt akribisch protokolliert, um eine lückenlose Dokumentation zu gewährleisten. Anschließend werden drei unterschiedliche neuronale Netzwerke (LSTM, GRU und CNN) mithilfe des Prototyps getestet, um einen Vergleich anzufertigen. Die daraus gewonnenen Erkenntnisse der Implementierung werden in einer abschließenden Diskussion vorgestellt und kritisch bewertet, um einen vertieften Einblick zu ermöglichen.

Um die erste Forschungsfrage „Welche Ansätze zur Log-Anomalie Erkennung existieren, und worin unterscheiden sie sich?“ in dieser Arbeit zu beantworten, wird eine intensive Literaturrecherche betrieben, um die bereits existierenden Ansätze zu definieren. Diese werden ebenfalls gegenübergestellt, um die Vor- und Nachteile sowie die Unterschiede zu zeigen. Damit die zweite Forschungsfrage „Wie kann man mithilfe von Tensorflow.js eine eigene Anwendung zur Log-Anomalie Erkennung in Web-Apps schreiben?“ beantwortet werden kann, wird ein eigener Prototyp namens ’LogDeeptector’anhand der JavaScript-API TensorFlow.js implementiert, um Anomalien in Logs in sogenannten progressiven Web-Applikation zu erkennen.

Der Aufbau dieser Diplomarbeit ist wie folgt strukturiert: Weiterführend in der Einleitung dieser Abschlussarbeit werden die wesentlichen Herausforderungen und Probleme von Log-Anomalie-Erkennung angeführt, um anschließend zu verstehen, worauf man bei Erstellung eines neuen Ansatzes achten sollte.
Der Hauptteil dieser Diplomarbeit wird in zwei Teile betrachtet: Durch Recherche wird im ersten Abschnitt eine Basis des aktuellen Themenbereiches geschaffen. Die folgenden Themen wie Log-Daten, Machine- und Deep Learning, Anomalie-Erkennung sowie Log Parsing werden ausführlich erklärt. Anschließend werden aktuelle Ansätze wie DeepLog, LogAnomaly, LogRobust PLELog, CNN und TransLog nähergebracht. Zudem werden die Unterschiede sowie Vor- und Nachteile der einzelnen Ansätze evaluiert.
Im zweiten Abschnitt des Hauptteils geht es darum, einen eigenen Prototyp zu entwerfen. Dieser Prototyp soll dazu dienen, ungewöhnliche Log-Ereignisse in Webanwendungen zu erkennen. Konkret handelt es sich um eine mit Angular entwickelte Anwendung. Um fortgeschrittene Deep Learning Funktionen in dieser Anwendung nutzen zu können, wird die JavaScript-API von TensorFlow, genannt TensorFlow.js, verwendet. Neben der Präsentation des fertigen Prototyps wird der Implementierungsprozess und die notwendigen Schritte detailliert erläutert. Die Arbeit umfasst somit die nötigen Schritte des Implementierungsprozesses sowie den Aufbau und die Funktionsweise des Prototyps.
Am Ende der Diplomarbeit befindet sich abschließend eine kurze Zusammenfassung inklusive Diskussion über die erlangten Ergebnisse sowie Erkenntnisse in Bezug auf den recherchierten Ansätzen sowie der Implementierung des Prototyps. Hierbei werden kurz die Ansätze nochmals erklärt und das Resultat sowie Limitierungen des fertigen Prototyps diskutiert. Schlussendlich werden noch Richtungen bzw. Blickwinkel für mögliche zukünftige Arbeiten angesprochen.

1.1 Herausforderungen und Probleme

In diesem Abschnitt werden einige Herausforderungen und Probleme, die es mit der Anomalie-Erkennung mit sich bringt, genannt. Die folgenden Punkte werden aus der Literatur „ISSUES AND CHALLENGES IN ANOMALY DETECTION FOR WEB LOG DATA“ von Siwach Meena und Mann Suman aus dem Jahr 2022 [10] referenziert. Zudem wird auch eine Lösung zu dem jeweiligen Problem vorgeschlagen:

- **Redundanz:** Der Umgang mit einer Menge an Echtzeitdaten, die aus einer großen Anzahl von verschiedenen Systemen stammen, kann zu Problemen führen. Insbesondere, wenn Daten regelmäßig wiederholt werden. Es gibt bereits Technologien wie Hadoop und Spark, die das Verarbeiten solcher Mengen anhand Clustering erleichtern. Doch diese sind nicht in der Lage, Datenredundanz, Qualität, Konsistenz und Speicherkosten zu bewältigen. Zudem haben diese Probleme mit den großen

Datenmengen und der dazukommenden Redundanz. Als Lösung wird vorgeschlagen, dass man unter Einsatz von Filtermodulen die Daten erfasst, um die Redundanz zu verringern.

- **Aufwand der Berechnung:** In der Forschung wird versucht, mögliche Methoden zu fusionieren bzw. zu mischen, um ein genaueres Erkennen von Anomalien zu erlangen. Doch dies führt auch zu einem hohen Aufwand der Berechnungen. Eine Verknüpfung zwischen einem großen Stichprobenumfang und einer hohen Dimensionalität, führt ebenfalls zu immensen Verarbeitungskosten. Um die Anforderungen und die Kosten zu senken, wird das Verarbeiten parallel auf zahlreiche Cluster ausgedehnt. Durch Virtualisierung, Cloud-, Edge- und Fog-Computing können weiterführend die Ausgaben gesenkt werden.
- **Art der Eingabedaten:** Seien es Vektoren, Muster, Events, usw., die es als Eingabedaten bei der Erstellung eines Modells zu berücksichtigen gibt. Jedes dieser Eingaben hat unterschiedliche Aspekte, wie Variablen, Merkmale, Eigenschaften, Felder und Dimensionen. Zudem werden Attribute in binär, kategorisch und kontinuierlich unterschieden. Es existieren univariat oder multivariat Daten. Feststellbar ist, dass es unterschiedliche Arten von Eingabedaten gibt und je nach Merkmalen der Daten variiert auch das Erkennen von Anomalien. Eine hybride Machine Learning bzw. Deep Learning Methode kann dieses Problem lösen.
- **Verunreinigungen und fehlende Werte:** Durch die verschiedenen Netzwerke und Systeme werden Daten empfangen, die fehlende Werte und Verunreinigungen aufweisen können. Durch die Unordnung und den fehlenden Variablen kann es zu einer hohen Wahrscheinlichkeit von false-positive Alarmen kommen, weil die eigentlichen Anomalien, durch die verunreinigten Eingabedaten, verschleiert sind. Als Lösung wird ein automatisches Bereinigungsmodul in der Applikation für solche Eingabedaten vorgeschlagen.
- **Wahl der Parameter:** Für die Erkennung von Anomalien in Echtzeit, ist es von großer Bedeutung, die geeigneten Bedingungen, wie ein oder mehrere Hyperparameter, für den Algorithmus auszuwählen. Man sollte sich bewusst sein, dass der Parameter, die zu Beginn des Entwicklungsprozesses ein gutes Ergebnis geliefert haben, später nicht mehr gut funktionieren, aber genauso umgekehrt. Die Parameter können das Modelltraining erheblich beeinflussen oder verzögern. Es existieren auch parameterfreie Ansätze, doch der vorgeschlagene Lösungsansatz hierbei ist ein Exzentrizitätsansatz, um die Parameterwahl zu reduzieren.
- **Unzureichende Architektur:** Eine Echtzeit-Datenarchitektur kann kleine Datenmengen verarbeiten, aber bei großen Datenmengen kommt diese Architektur schnell zu ihren Grenzen. Nun wird versucht eine Big-Data-Architektur zu entwickeln, die laufende Daten sowie ruhende Daten unterstützt. Dabei

wird versucht Komponenten der Echtzeitarchitektur in die Big-Data-Architektur zu integrieren, da eine Big-Data-Architektur nutzlos ist, wenn die keine aktuellen Unternehmensdaten verknüpfen kann. Ebenso kann eine Analyse nicht durchgeführt werden, wenn Big Data nicht mit den Daten korreliert.

- **Datenvisualisierung:** Die analysierten und verarbeiteten Daten bzw. der daraus generierte Bericht muss von einer Benutzerin oder einem Benutzer verstanden werden können. Dazu ist es notwendig, die Informationen mit einem geeigneten Visualisierungsansatz darzustellen. Hierzu gibt es unterschiedliche Methoden, die zum Einsatz kommen können, sei es mit einem einfachen 2-dimensionalen Diagramm (Bsp. Balkendiagramm) bis hin zu komplexen 3-dimensionalen bzw. mehrdimensionalen Perspektiven. Zudem muss die Benutzerin und der Benutzer das Verständnis haben, um mit solchen Visualisierungen umgehen zu können. Als Lösung für diese Herausforderung stehen diverse Open-Source Visualisierungsansätze zur Verfügung, die in Frameworks eingebettet werden können.
- **Datenheterogenität:** Bei Daten unterscheidet man zwischen unstrukturiert und strukturiert. Bei den unstrukturierten Daten handelt es sich beispielsweise um aufgezeichnete Meetings, PDF-Dokumente, Faxübertragungen und E-Mails. Hingegen zu unstrukturierten Daten sind strukturierte Daten immer geordnet, kontrollierbar und haben eine gute Datenbankintegration. Zudem ist es teuer und auch nicht immer möglich, die unstrukturierten Daten zu strukturierten Daten konvertieren zu lassen. Um diese Herausforderung zu bewältigen, wird eine Kombination von hybriden oder erweiterten Algorithmen und Big-Data-Technologien in Echtzeit vorgeschlagen, um die Daten zu kategorisieren
- **Präzision:** Einige der derzeitigen Technologien sind in Bezug auf Genauigkeit unzuverlässig. In einigen Fällen geht dies mit einem höheren Rechenaufwand und mehr Zeit einher. Lösung ist es, hybride Algorithmen mit Big-Data-Technologien zu kombinieren, um ein Meta-Learning-Tool daraus zu erschaffen, welches enorme Datenmengen von einer Anwendung analysieren kann und dabei noch weniger Speicher und Strom verbraucht.

Die genannten Punkte beziehen sich allgemein auf die Erkennung von Anomalien. Während der Implementierung des Prototyps sind einige Herausforderungen bzw. Probleme im Zusammenhang mit Web-Applikation aufgetaucht. Die folgenden genannten Punkte beziehen sich speziell auf die Schwierigkeiten und Probleme, die bei Webanwendungen aufgetreten können:

- **Client-seitige Protokolle:** Einige Web-Applikationen stützen sich häufig auf clientseitige Technologien wie JavaScript, um ein umfassendes Benutzererlebnis zu bieten. Dies kann den Zugriff und die Analyse von Protokolldaten erschweren, da die Protokolle über verschiedene Geräte und Browser verstreut sein können. Während des Implementierungsprozesses ist es aufgefallen, dass das Importieren der Log-Dateien nicht reibungslos funktioniert hat und es auch Limitierung in Größe und Umfang der

Daten gegeben hat.

- **Netzwerkfehler:** Auch wenn der Prototyp offline ist und die Log-Datei lokal abgespeichert ist, können Web-Applikationen von Netzwerkfehler wie Verbindungsproblemen oder Latenzzeiten betroffen sein, die die Leistung der Anwendung beeinträchtigen können. Daher ist es wichtig, auf diese Fehler zu achten und die zuständigen Stellen zu benachrichtigen, wenn sie auftreten.
- **Benutzerinteraktionen:** Web-Applikationen sind so konzipiert, dass sie interaktiv und reaktionsschnell sind, was bedeutet, dass sie in kurzer Zeit eine große Anzahl von Protokollereignissen erzeugen können. Es ist wichtig, über ein System zu verfügen, das diese Datenmenge bewältigen und Anomalien erkennen kann.
- **Überwachung in Echtzeit:** Web-Applikationen müssen oft in Echtzeit überwacht werden, um sicherzustellen, dass sie ordnungsgemäß funktionieren. Somit können mögliche Ursachen schnell erkannt und gelöst werden. Dies kann die Verwendung von Plattformen zur Log-Aggregation oder kundenspezifische Überwachungslösungen beinhalten. Der Prototyp hat gezeigt, dass es bei der Überwachung von Echtzeitdaten entscheidend ist, die richtigen Schritte bei der Verarbeitung der Protokolldaten und der Datenkodierung einzuhalten. Andernfalls kann es zu längeren Wartezeiten kommen, bis das Ergebnis zur Verfügung steht. Auch das Trainieren eines Modells mit Echtzeitdaten benötigt Zeit.

1.2 Struktur der Arbeit

Diese Arbeit ist in folgende Teile organisiert. Kapitel 1 leitet das Thema ein und erläutert die Motivation und die dazugehörigen Probleme und Herausforderungen, die das Thema mit sich bringt. Der Abschnitt 2 beschreibt einige grundlegende Themen, um ein besseres Verständnis aufzubauen. Das Kapitel 3 listet die ausgewählten Ansätze als Related Work auf. Der Teil 4 beschreibt die Implementierung des Prototyps und berichtet über die resultierenden Ergebnisse. Abschließend fasst das Kapitel 5 alles zusammen und beendet die Arbeit.

2 Grundlagen

Um einen umfassenden Überblick über die Thematik dieser Arbeit zu erhalten, werden in diesem Kapitel die Grundlagen der Log-Anomalie-Erkennung erläutert, dabei werden grundlegende Konzepte wie Log-Daten, Log-Parsing, Anomalie-Erkennung und Maschine - und Deep Learning behandelt.

2.1 Log-Daten

Bei einem Log handelt es sich meist um einen unstrukturierten Text, der von Methoden, wie printf() und console.writeline(), angezeigt werden kann. Solch ein Log kann unterschiedlichste Informationen enthalten, beispielsweise kann ein Log folgende Felder wie Zeitstempel (Zeitpunkt des Auftretens des Ereignisses, z. B. 2022-12-09 19:34:45,676), Ausführlichkeitsgrad (Schweregrad des Ereignisses, z. B. INFO oder ERROR) und eine Beschreibung des Ereignisses in Textform, enthalten. Es ist wichtig, dass man sich bewusst ist, wie man einen Protokollierungsprozess definiert. Die OWASP bietet hierfür ein CheatSheet [11] an, damit Entwicklerinnen und Entwickler eine klare Anleitung zur Einrichtung von Protokollierungssystemen für Anwendungen, insbesondere im Hinblick auf die Sicherheitsprotokollierung haben. Wie im OWASP CheatSheet beschrieben, bietet es sich an, wenn man davor die folgenden Fragen, wie soll protokolliert werden, wo soll protokolliert werden und was soll protokolliert werden? , beantwortet. Bei der Frage "Wie protokollieren werden soll?"geht es um Anti-Patterns oder unerwünschte Muster, die beim Hinzufügen von Protokollierungsanweisungen zum Quellcode vermieden werden sollten. Der Aspekt, wo protokolliert werden soll, wird durch die Platzierung der Anweisungen im Funktionscode bestimmt. Wenn man nicht benötigte Daten aufzeichnet, kann sich das negativ auf die Performance auswirken und unnötigen Overhead produzieren. Was zu protokollieren ist, setzt sich aus der Auswahl der Nachricht, der Parameter, der Thread-Informationen, der Ebene und anderer Optionen zusammen.[12]

Sogenannte Logging-Lösungen spielen eine entscheidende Rolle, um Log-Daten effizient zu erfassen, zu analysieren und zu visualisieren, um Probleme zu erkennen. Anschließend kann die Leistung eines Systems optimiert und die Sicherheit gewährleistet werden. Es gibt eine Vielzahl von unterschiedlichen Lösungen,

die für diverse Anwendungsfälle verwendet werden können. Die folgenden Logging-Lösungen sind wohl die bekanntesten.

- **ELK-Stack:** Der ELK-Stack von Elastic [13] stellt eine Sammlung von Open-Source-Tools dar, die regelmäßig gemeinsam eingesetzt werden, um umfassende Prozesse der Datenerfassung, -speicherung, -analyse und -visualisierung zu ermöglichen. Das Akronym ËLKïst eine Abkürzung für die drei Hauptkomponenten des Stacks.
  - Die erste Komponente ist Elasticsearch, diese fungiert als Such- und Analyse-Engine. und dient dazu, Log-Daten zu akkumulieren, zu persistieren, zu indizieren und gleichzeitig filterbar zu machen. Diese Fähigkeit ermöglicht es der Benutzerin und dem Benutzer, große Datenmengen effizient zu durchsuchen und Anfragen auszuführen.
  - Die Aufgabe von Logstash ist es, die Logdaten aus einer Vielzahl von Quellen zu konsolidieren, zu transformieren und in homogene Datenformate zu überführen. Man spricht von einem Vermittler zwischen verschiedenen Datenproduzenten und der Komponente Elasticsearch.
  - Kibana bildet die Schnittstelle zur webbasierten Visualisierung und Analyse der in Elasticsearch archivierten Daten. Benutzerinnen und Benutzer können benutzerdefinierte Dashboards erstellen, um Logdaten zu inspizieren, Diagramme und Grafiken zu generieren und Suchanfragen zu formulieren. Dadurch können aus den vorhandenen Logdaten tiefere Einblicke gewonnen werden.
- **Splunk:** Splunk [14] ist eine kommerzielle Softwareplattform, die sich als integraler Bestandteil im Bereich der Datenerfassung, -speicherung, -analyse und -visualisierung etabliert hat. Wie beim ELK-Stack handelt es sich um eine Technologie, die es ermöglicht, Daten aus einer Vielzahl von Quellen zu extrahieren, zu sammeln, zu kategorisieren, zu analysieren und in aussagekräftigen Darstellungen zu präsentieren.
- **Vercel Log Drains:** Vercel Log Drains [15] sind eine Implementierung innerhalb der Vercel Plattform, die dazu dient, Protokolldaten von Vercel Webanwendungen effizient an externe Dienste oder Tools zu übermitteln. Diese Funktion ermöglicht die gezielte Weiterleitung von Protokolldaten an Log-Management-Systeme von Drittanbietern oder andere externe Dienste, die eine Verarbeitung oder Analyse von Protokolldaten erfordern.

Es existieren noch weitere Logging-Lösungen wie, Graylog, LogRocket, Loggly, Datadog und viele mehr. Jede Lösung hat ihre eigenen Vor- und Nachteile und kann auf unterschiedliche Weise in Ihre Webanwendungsinfrastruktur integriert werden, abhängig von Ihren spezifischen Anforderungen und Präferenzen.

Entwicklerinnen und Entwickler können die Information aus den Logs nutzen, um das Verhalten von einem System bzw. von einer Applikation zu analysieren und anschließend von unterschiedliche Möglichkeiten (Anomalie-Erkennung, Verifizierung des Programmes, usw.) Gebrauch machen. Durch die immense Menge an Log-Daten, die protokolliert werden, ist es schwer, die wichtigsten Informationen für eine mögliche Diagnose zu filtern. Oft ist dabei eine einfache Suche ineffizient. Zudem wird aus Bequemlichkeit der Großteil in Freitext Form protokolliert, was einen unstrukturierten Datensatz hervorbringt. Was eine direkte Analyse der Rohdaten schwierig macht. Um die Log-Daten in eine, für die Analyse, brauchbare Form zu bringen, müssen die erzeugten Protokolldateien aus ihrem rohen, unformatierten Zustand in eine strukturierte Form umgewandelt werden. Hierfür werden im nachfolgenden Abschnitt ‚Log-Parsing' ein paar Ansätze definiert, wie man unregelmäßige Eingaben in geordnete Protokolle umwandelt.[12]

2.1.1 Loghub

Unternehmen verwenden zunächst ihre eigenen branchenabhängigen Log-Daten, um Analysen durchzuführen. Doch diese werden ungern, aus Gründen des Datenschutzes, mit anderen geteilt. Zudem kann eine Strategie, die bei einer Art von Protokolldaten erfolgreich ist, bei einer anderen Art von Protokolldaten möglicherweise nicht günstig sein. Weiteres ist es für Entwicklerinnen und Entwickler mühsam und zeitaufwändig, die unterschiedlichen und nötigen Daten zu sammeln, um das gewünschte Ergebnis zu produzieren. Aus diesem Anlass wird in der Literatur „Loghub: a large collection of system log datasets towards automated log analytics" von He S., Zhu J., He P., und Lyu M. R. aus dem Jahr 2020 [16], die Studie ‚Loghub' vorgestellt. Dabei handelt es sich um eine umfangreiche Sammlung von Protokolldatensätzen, die zusammengestellt und arrangiert wurde. Damit wird erreicht, dass diese wichtige Lücke zwischen Wissenschaft und Wirtschaft geschlossen werden kann und zusätzliche Forschung zu KI-gestützter Protokollanalyse gefördert wird. Die Daten werden in einer Datenbank aufbewahrt und stehen zur freien Verfügung. Insgesamt bietet Loghub 77 Gigabyte an Log-Daten. Aufgrund des großen Umfangs werden in der genannten Studie nur 17 Datensätze, die von Systemen gesammelt wurden, aufgelistet. Diese sind in der Tabelle 2.1 ersichtlich und umfassen Systemen wie verteilte Systeme (Hadoop Distributed File System (HDFS), Hadoop, Spark, etc.), Supercomputer (BGL, HPC and Thunderbird), Betriebssysteme (Windows, Linux and Mac), Mobile Applikationen (Android), Server Applikationen (Apache and OpenSSH) und eine Standalone Software (Proxifier).[16]

System	Beschreibung	Datengröße	Beschriftet
Verteilte Systeme			
HDFS	Hadoop verteilte System Logs	1.47GB	Ja
		16.06GB	Nein
Hadoop	'mapreduced' Job Logs	48.61MB	Ja
Spark	Spark Job Logs	2.65GB	Nein
Zookeeper	Service Logs	9.95MB	Nein
OpenStack	Infrastruktur Logs	58.61MB	Ja
Supercomputer			
BGL	Blue Gene Logs	708.76MB	Ja
HPC	High Performance Cluster Logs	32MB	Nein
Thunderbird	Thunderbird Logs	29.6GB	Ja
Betriebssysteme			
Windows	Event Logs	26.09GB	Nein
Linux	System Logs	2.25MB	Nein
Mac	OS Logs	16.09MB	Nein
Mobile Systeme			
Android	Framework Logs	183.37MB	Nein
HealthApp	HealthApp Logs	22.44MB	Nein
Server Applikationen			
Apache	Web Server Error Logs	4.90MB	Nein
OpenSSH	Server Logs	70.02GB	Nein
Standalone Software			
Proxifier	Software Logs	2.42MB	Nein

Tabelle 2.1: Systeme von LogHub [16]

2.2 Log Parsing

Log Parsing ist der Prozess vom Umwandeln der unstrukturierten rohen Log-Dateien in eine strukturierte Datenform, damit anschließend eine Analyse darüber durchgeführt werden kann. Mit regulären Ausdrücken kann man die Protokolle durchlaufen und mittels Vorlage abgleichen, doch dafür ist Vorkenntnis zu allen denkbaren Strukturen erforderlich. Außerdem können keine neuen Log-Vorlagen erlernt werden und müssen immer von Hand implementiert bzw. angepasst werden. Weiteres ist es auch möglich, die Struktur, mittels statischer Analysen, aus dem Quellcode abzuleiten. Vielversprechend haben sich automatische Log-Parser in neuen Untersuchungen erwiesen, wie beispielsweise die automatisierten Log-Parser wie Drain [17], Spell [18], LenMa [19], IPLoM [20] und LogSig [21]. Diese erlernen automatisch die Syntax von Log-Templates, was zu einem großen Vorteil führt. Man unterscheidet zwischen online und offline bei dem automatischen Log-Parser. Beim Online Parsing werden die Logs in einem Streaming-Verfahren nacheinander verarbeitet, hingegen benötigt der offline Modus ein Stapel and Logeinträgen. Die Literatur „Survey on Online Log Parsers“ von Tejaswini S. und Azra Nasreen aus dem Jahr 2021 [12] präsentiert dazu folgende Ansätze in Bezug auf Online Log-Parser.

2.2.1 Drain

Drain [17] ist ein Ansatz, bei dem ein Parsing-Baum mit fester Tiefe in Verwendung kommt. Bei diesem Ansatz wird eine einfache Vorverarbeitung mit regulären Ausdrücken unter Verwendung von Domänenwissen, sobald eine neue rohe Protokollnachricht empfangen wird, durchgeführt. Anschließend wird anhand der speziell erstellten Regeln, die in den internen Knoten des Baums gespeichert sind, eine Protokollgruppe (Blattknoten) gesucht. Wenn eine kompatible Log-Gruppe gefunden wird, wird die Log-Meldung mit dem in dieser Log-Gruppe gespeicherten Log-Ereignis verglichen. Ist dies nicht der Fall, wird die Log-Meldung verwendet, um eine neue Log-Gruppe zu erstellen. Ein Online-Log-Parser muss entweder eine neue Log-Gruppe erzeugen oder die am besten geeignete Log-Gruppe für eine frische Log-Meldung suchen, wenn sie eintrifft. Eine schnelle Lösung für dieses Verfahren ist der Vergleich jedes einzelnen Log-Ereignisses mit der rohen Log-Meldung, die in jeder Log-Gruppe aufgezeichnet wurde. Da es jedoch so viele Protokollgruppen zu analysieren gibt, ist diese Technik recht träge.[17]
Um diesen Vorgang zu beschleunigen, wird ein Parse-Baum mit definierter Tiefe erstellt, um die Suche nach Protokollgruppen zu steuern. Dadurch wird die Anzahl der Protokollgruppen, mit denen eine rohe Protokollmeldung verglichen werden muss, erheblich eingeschränkt. Der Wurzelknoten befindet sich in der obersten Schicht des Parse-Baums, während sich die Blattknoten in der untersten Schicht befinden. Die anderen Knoten im Baum sind interne Knoten. Interne Knoten und der Wurzelknoten kodieren benutzerdefinierte Regeln für den Suchprozess. In ihnen gibt es keine Protokollgruppen. Der Einfachheit halber wird hier nur ein Blattknoten gezeichnet, an dem jede Route im Parse-Baum endet und der eine Liste von Protokollgruppen verwaltet. Jede Protokollgruppe besteht aus Protokollereignissen und Protokoll-IDs. Die Log-Meldungen dieser Kategorie, die aus dem konstanten Teil einer Log-Meldung bestehen, werden am besten durch die Log-Event-Vorlage beschrieben. Die IDs der Protokollmeldungen werden in dieser Kategorie über Protokoll-IDs festgehalten. Zusätzlich wird die Option 'maxChild' verwendet, die maximale Anzahl der Kinder eines Knotens begrenzt, um eine Explosion der Baumzweige zu verhindern. Die einzelnen Schritte, die ein Log bzw. eine Protokollnachricht durchläuft, sind wie folgt [17]:

1. Vorverarbeitung durch Fachwissen: Die rohen Protokollnachrichten werden, bevor sie in den Parse-Baum verwendet werden, noch vorverarbeitet. In Bezug auf Drain können einfache reguläre Ausdrücke auf der Grundlage von Fachwissen bereitgestellt werden, um weit verbreitete Variablen wie IP-Adressen und Block-IDs ausfindig zu machen. Die Token, die mit diesen regulären Ausdrücken in der rohen Protokollnachricht übereinstimmen, werden dann von Drain herausgefiltert. So kann man beispielsweise numerische Werte wie Temperatur oder die ID löschen.
2. Suche nach der Länge der Protokollnachricht: Der Ablauf beginnt mit der vorverarbeiteten Protokoll-

nachricht am Wurzelknoten des Parse-Baums. Die Knoten der ersten Ebene des Parse-Baums entsprechen den Protokollgruppen, deren Protokollnachrichten unterschiedlich lang sind. Die Anzahl der Token in einer Protokollnachricht bezeichnet man als Länge des Protokolls. Drain wählt in dieser Phase einen Pfad zu einem Knoten der ersten Schicht auf der Grundlage der Länge des vorverarbeiteten Logs aus. Zum Beispiel, um den internen Knoten *'Länge: 4'* für die Protokollnachricht *'Empfang von Knoten 4'* zu erreichen. Auch wenn es denkbar ist, dass Protokollnachrichten, die mit demselben Protokollereignis verbunden sind, unterschiedliche Längen haben, kann diese Situation mit ein wenig Nachbearbeitung gelöst werden.

3. Suche nach vorangehendem Token: Dieser Schritt basiert auf der Idee, dass die Token an den ersten Stellen einer Protokollnachricht mit größerer Wahrscheinlichkeit Konstanten sind. Drain wählt den nachfolgenden internen Knoten speziell auf der Grundlage der Token in den Anfangsstellen der Protokollnachricht aus. Beispielsweise geht Drain für die Protokollnachricht *'Empfangen von Knoten 4'* vom Knoten der ersten Schicht *'Länge: 4'* zum Knoten der zweiten Schicht *'Empfangen'*, da 'Empfangen' der Token an der ersten Stelle ist. Gelegentlich beginnt ein Protokollbericht mit einem Parameter, z. B. *'120 Bytes empfangen'*. Da jedes Argument (z. B. 120) in einem internen Knoten kodiert wird, können diese Arten von Protokolleinträgen zu einer Explosion der Verzweigung im Parse-Baum führen. In dieser Phase werden ausschließlich Token ohne Ziffern berücksichtigt, um solch eine Verzweigungsexplosion zu verhindern. Ein bestimmter interner Knoten '*' wird verwendet, wenn ein Token Zahlen enthält. Drain navigiert zum Beispiel zum internen Knoten '*' und nicht zu '120' für die obige Protokollmeldung.
4. Suche nach Token-Ähnlichkeit: Drain hat bereits vor diesem Schritt einen Blattknoten erreicht, der eine Liste von Protokollgruppen enthält. Die Protokollnachrichten dieser Protokollgruppen entsprechen den Regeln, die in den internen Knoten enthalten sind, die den Pfad enthalten. Drain wählt in diesem Schritt die beste Gruppe aus der Liste der Protokollgruppen aus. Die *simSeq*-Ähnlichkeit wird zwischen der Protokollnachricht und dem Protokollereignis jeder Protokollgruppe mit einer Formel bestimmt. Wenn die Protokollmeldung durch seq_1 und das Protokollereignis durch seq_2 dargestellt wird, ist $seq(i)$ der i-te Token der Sequenz und n die Länge der Protokollmeldung der Sequenz. Die Definition der Funktion equ lautet wie folgt. Angenommen, es gibt zwei Token, t_1 und t_2. Anschießend wird die Log-Gruppe mit dem höchsten *simSeq*-Wert mit einem vorgegebenen Ähnlichkeitsschwellenwert st verglichen. Drain gibt die Gruppe als die beste geeignete Log-Gruppe zurück, wenn $simSeq > st$ ist. Wenn es keine geeigneten Log-Gruppen gibt, gibt Drain ein Flag zurück, um dies anzuzeigen.

5. Aktualisieren des Parse-Baums: Drain fügt die Protokoll-ID der aktuellen Protokollmeldung zu den Protokoll-IDs in der zurückgegebenen Protokollgruppe hinzu, wenn in Schritt vier des Prozesses eine ausreichende Protokollgruppe zurückgegeben wurde. Zusätzlich wird eine Aktualisierung des Protokollereignisses in der zurückgegebenen Protokollgruppe vorgenommen. Wenn die beiden Token identisch sind, wird der Token an dieser Position nicht geändert. Ist dies nicht der Fall, wird der Token an dieser Stelle im Protokollereignis mit einem Platzhalter (z. B. *) geändert. Drain generiert eine neue Protokollgruppe auf der Grundlage der aktuellen Protokollmeldung, wenn es keine passende finden kann, und die Protokoll-IDs enthalten nur die ID der Protokollmeldung und der Protokollgruppe. Die neue Protokollgruppe wird dann von Drain in den Parse-Baum eingefügt.

2.2.2 Spell

Der Longest Common Subsequence-Algorithmus (LCS) wird von Spell [18] zum Parsen von Protokollen verwendet. Das Grundprinzip von LCS besteht darin, dass bei zwei Zeichenketten diejenige Teilzeichenkette gefunden wird, die in beiden angegebenen Zeichenketten vorkommt und die längste ist. Zum Beispiel erzeugen die Folgen *1, 3, 5, 7, 9* und *1, 5, 7, 10* eine LCS von *1, 5, 7*. Spell hält sich an die eigentliche Definition eines Online-Stream-Parsers, da es im Gegensatz zu Drain keinerlei Vorverarbeitung oder vorheriges Domänenwissen benötigt, um mit dem Parsen von Protokollen zu beginnen. Vergleicht man die Implementierung mit der zuvor genannten Baumtechnik, so ist sie ebenfalls recht einfach.
Jedes Wort wird, wie ein Token behandelt, da jeder eingehende Protokolleintrag in eine Reihe von Token umgewandelt wird. Als eindeutiger Bezeichner für jeden Protokolleintrag wird eine einfache Ganzzahl verwendet, die bei 0 beginnt und mit jedem weiteren Eintrag zunimmt. Die LCSMap, die der Parser im Speicher hält, enthält LCS-Objekte. So ein LCS-Objekt enthält sowohl die eindeutige Kennung des Protokolleintrags, welcher der Sequenz entspricht, als auch eine LCSseq, eine Folge von Token, die der Vorlage der Protokollmeldung entspricht. Um die Länge der längsten gemeinsamen Teilsequenz für jeden Vergleich zu bestimmen, wird die Token-Sequenz mit der LCSseq jedes LCS-Objekts nach der Tokenisierung verglichen. Ein Schwellenwert $t = -s - /2$ wird dann verwendet, um die längste Länge l aller Vergleiche zu vergleichen, wobei s die Länge der tokenisierten Protokollnachricht ist. Der Schwellenwert in Spell ist unglaublich einfach und intuitiv im Vergleich zu dem Schwellenwert in Drain. Sobald die Schwellenwertanforderung erfüllt ist, geht das Verfahren rückwärts, um eine neue Vorlage zu erstellen, die sowohl mit der neu zugelassenen Sequenz als auch mit der LCSseq übereinstimmt. Ist die Schwellenwertanforderung nicht erfüllt, wird ein neues LCS-Objekt mit der LCSseq als neuer Protokollmeldung und seiner eindeutigen Kennung gestartet. Logan [22] ist ein weiterer bekannter Online-Log-Parser, der auf dem Longest Common

Subsequence-Algorithmus basiert.[18]

2.2.3 LenMa und SHISO (Clustering)

Die Anzahl der Wörter in einem Protokoll kann als einfacher Clustering-Parameter verwendet werden. Das eingegebene Protokoll wird zunächst in zwei Vektoren umgewandelt: einen mit den einzelnen Wörtern des Protokolls und einen mit den Längen der einzelnen Wörter. Jedes Cluster enthält außerdem einen eindeutigen Wortvektor und einen Längenvektor. Durch Berechnung des Kosinus der beiden Vektoren wird die Kosinus-Ähnlichkeit des Nachrichtenvektors zu jedem Cluster mit der gleichen Vektorlänge berechnet. Es wird auch eine Positionsähnlichkeit berechnet, die die Wortvektoren vergleicht und die Anzahl der gemeinsamen Wörter angibt. Wenn beide Ähnlichkeiten größer als die vom den Entwicklern festgelegten Schwellenwerte sind, wird das neue Protokoll in dem Cluster aufgenommen und der Längenvektor wird mit dem der neuen Nachricht aktualisiert, während der Wortvektor mit '*' aktualisiert wird, wenn eine Nichtübereinstimmung auftritt. Andernfalls wird ein neues Cluster mit der Länge und dem Wortvektor des neuen Protokolls erstellt. Wie Spell erfordert auch LenMa [19] keine Vorverarbeitung der Daten. Allerdings ist der in LenMa verwendete Schwellenwert im Gegensatz zu den anderen diskutierten Parsern statischer Natur und wird von der Entwicklerin oder dem Entwickler festgelegt.
Eine der frühesten Implementierungen von Online-Parsern, SHISO [23], erstellt eine baumartige Struktur, die geparst werden kann, um das Protokollformat zu identifizieren, wobei die Kindknoten jedes Elternknotens als Cluster betrachtet werden können. Dies geschieht in zwei Phasen - der Suchphase und der Anpassungsphase. Ausgehend vom Wurzelknoten als übergeordnetem Knoten wird in der Suchphase nach einem Formatknoten im Baum gesucht, der der tokenisierten Protokollnachricht ähnlich ist, indem der euklidische Abstand gemessen wird. Wenn keiner der gemessenen Abstände den Schwellenwert überschreitet, der ein von Benutzerin oder Benutzer gewählter Wert zwischen 0 und 1 ist, wird jeder untergeordnete Knoten als übergeordneter Knoten betrachtet und der Rest des Baums wird geparst. Alle Kindknoten des Elternknotens müssen die gleiche Nachrichtenlänge haben und haben auch eine maximale Grenze für die Anzahl der Kindknoten. Sobald ein Knoten den festgelegten Schwellenwert überschreitet, beginnt die Anpassungsphase. In dieser Phase wird das Format des ausgewählten Knotens aktualisiert, indem generische Vorlagen erstellt werden. Dabei werden Variablen durch Platzhalter ersetzt und statische Wörter bleiben an ihrem Platz. Um die Anpassungsphase durchzuführen, verwendet SHISO n-Gramme, um die statischen und variablen Teile der Nachricht zu finden. Im Vergleich zu anderen Parsern ist bei SHISO keine explizite Vorverarbeitung erforderlich. Der Schwellenwert ist statisch und erfordert das Wissen des Entwicklers, um den richtigen Wert zu wählen. Der Parser hat auch den Nachteil, dass er Ungleichgewichte im Baum erzeugen kann, sodass

bestimmte Teile viel tiefer wachsen können als andere.[19]

2.2.4 Logram

Logram [24] wurde ursprünglich als N-Gramm-Offline-Batch-Parser entwickelt, aber es wird gezeigt, wie er leicht zu einem Online-Parser modifiziert werden kann. Ein N-Gramm, bei dem 'N' eine ganze Zahl größer als 0 ist, ist eine Teilfolge von 'N' Wörtern, die aus der Log-Meldung abgeleitet werden. Betrachten wir zum Beispiel den Satz *'Es regnet Hunde und Katzen'*, der fünf 2-Gramme bildet - *'Es ist', 'regnet', 'regnet Hunde', 'Hunde und' und 'und Katzen'*. Wie Drain hat auch Logram einen Vorverarbeitungsschritt, in dem es nur den Inhalt der Nachricht extrahiert, indem es alle trivialen Informationen wie Thread-Name, Level, Zeit und Datum entfernt. Außerdem wird die extrahierte Nachricht tokenisiert. Logram verwendet ein 2-Gramm und ein 3-Gramm-Wörterbuch, um das Parsing-Verfahren zu unterstützen. Beim Eintreffen der ersten Nachricht ist das Wörterbuch leer. Daher wird die Nachricht in ihre n-Gramme unterteilt, die direkt als Wörterbucheinträge zusammen mit der Anzahl ihrer Vorkommen verwendet werden. Wenn neue Nachrichten eintreffen, werden die n-Gramme mit dem Wörterbuch verglichen, und wenn die Anzahl der Vorkommen eines n-Gramms unter einem Schwellenwert liegt, wird das n-1-Gramm berücksichtigt. Liegt dieser Vergleich ebenfalls unter dem Schwellenwert, wird das n-Gramm als dynamisches n-Gramm betrachtet, d. h. es enthält eine Variable. Die n-gram-Einträge und die Anzahl der Vorkommen werden mit jeder neuen Nachricht aktualisiert. Der Schwellenwert wird mithilfe einer Glättungsfunktion namens loess [25] und einer eindimensionalen Clustermethode namens Ckmeans[26] berechnet, die eine Bruchstelle zwischen Gruppen statischer N-Gramme und dynamischer N-Gramme findet.[24]

2.3 Anomalie-Erkennung

Bei der Anomalie-Erkennung geht es darum, eine Abweichung, Ausnahme bzw. einen Ausreißer in einem Ereignis, einer Aktivität oder in einer Beobachtung zu erkennen. Man spricht über anomal, wenn das Verhalten nicht der Norm entspricht. Ein Beispiel für ein möglich anomales Verhalten ist es, wenn ein Unternehmen routinemäßig Ausgaben mit einem Durchschnittswert von 1.000 Euro hat, aber unerwartet eine Ausgabe über 10.000 Euro vorweist. Anomalien können sich auch in Verhaltensweisen wie plötzliche Erhöhung der Netzwerklatenz, veränderten Muster des Internetverkehrs oder sogar der CPU-Temperatur eines Servers äußern. In diesen Fällen weicht das typisch bekannte Muster ab, und dies ist zu erkennen und weiter nachzugehen, um mögliche Probleme festzustellen. Anomalien werden zwischen drei Typen kategorisiert: [27][9]

- **Punktanomalie:** Hierbei handelt es sich um die Art von Anomalie, die am einfachsten zum Identifizieren ist. Die Instanz ist über oder unter dem typischen Bereich und weicht deutlich von den restlichen Dateninstanzen ab, dabei spricht man von einer Punktanomalie. Beispiel wäre es, wenn eine typische Kreditkartenabrechnung 2.000 Euro beträgt und plötzlich erhält man eine Abrechnung von über 10.000 Euro.
- **Kontextbedingte Anomalie:** Die Dateninstanzen sind im Kontext anomal, dabei lässt sich aus der Struktur des Datensatzes das Konzept des Kontextes ableiten. Es gibt zwei Sätze von Merkmalen, den kontextbezogenen und den verhaltensbezogenen. Bei dem kontextbezogenen Merkmal wird der Kontext für diese Instanz bestimmt, in einem 2-dimensionalen Graphen stellt das meist die x-Achse dar, wie beispielsweise eine Zeitlinie. Das verhaltensbezogene Attribut stellt nicht den Kontext dar, sondern zeigt das jeweilige Verhalten zu dem jeweiligen kontextbezogenen Merkmal. Die y-Achse kann beispielsweise das Verhalten darstellen (die Menge an Niederschlag über die Zeit). Es ist wahrscheinlich, dass die Kreditkartenabrechnung im Laufe der Zeit schwankt. Doch wenn man den gesamten Kontext betrachtet, kann man eine unerwünschte Erhöhung erkennen.
- **Kollektive Anomalie:** Hier liegt ein Kollektiv von Instanzen vor, die sich von den anderen Instanzen unterscheiden. Dabei stellt jeder einzelne Datensatz aus dem Kollektiv keine Anomalie dar, nur in der Gruppe ist eine Abweichung zu erkennen. Bei einer Kreditkartenzahlung ist es erst verdächtig, wenn der Betrag mehrere Monate von 1.000 Euro auf 2.000 Euro ansteigt.

Unterschiedliche Klassifizierungstechniken wie durch Clustering oder k-Nearest Neighbor (kNN), sowie Applikationen wie diverse Intrusion - und Fraud Detection Anwendungen, wurden in den letzten Jahren vorgestellt, um Anomalien in Systemen ausfindig zu machen. Dabei werden durch diverse statistische Metriken der Mittelwert und die Standardabweichungen in großen Datenmengen (Big Data) berechnet. Im Grunde wird mit diesen Methoden versucht, Muster oder Schwankungen zu erkennen, die anomal sein können. Im Laufe dieser Arbeit werden einige ausgewählte Verfahren in dem späteren Kapitel 3 erklärt.[27] [9]
Um zu verstehen, welche Vorteile die Erkennung von Anomalien haben kann und in welchen Anwendungsgebieten es hilfreich ist, wurden die folgenden und offensichtlichen Punkte angeführt: [27][9]

- **Sicherheitsmanagement und Betrugserkennung:** Betrug, Sicherheitslücken bzw. probierte Sicherheitslücken werden automatisch erkannt. Die Administratorin oder der Administrator wird informiert, wenn die Anzahl an Anmeldeversuche zu groß ist, ungewöhnliche Signaturen auftauchen oder der Netzwerkverkehr ungewöhnlich ansteigt. Es wird versucht, Angriffe oder Ausfälle frühzeitig zu erkennen, um schnell Maßnahmen zu ergreifen.
- **Unternehmensbewertung:** Nicht nur IT-Sicherheit, sondern auch Business Intelligence (BI) kann

davon profitieren, indem man Schwankungen in der Beliebtheit von Artikeln feststellen kann, oder ob ein Produkt noch rentabel ist. Händlerinnen und Händler können die Verkaufsdaten überblicken und Ausreißer schnell erkennen.

- **Datenbank-Verwaltung:** Wenn Daten in der Datenbank fälschlicherweise hinzugefügt, entfernt, geändert oder dupliziert werden, kann dies zu Anomalien führen, die letztendlich den Datensatz beschädigen. Für zuverlässige, qualitativ hochwertige Daten ist es entscheidend, diese Art von Anomalien zu finden.
- **Messung des Benutzererlebnisses:** Da es sich auch oft um Echtzeit-Daten handelt, kann man dies nutzen, um Anomalien in Anfragen bzw. in Feedback von Benutzerinnen oder Benutzer zu erkennen. Dabei kann es sich um einen Serverausfall handeln oder um eine Anfrage bestimmter Ressourcen.
- **Überwachung der Systemleistung:** Infrastruktur-Anomalien können auftreten, wenn die Ein-´und Ausgaberaten schnell ansteigen oder abfallen, die Fehlerraten plötzlich ansteigen oder die Temperatur einer Computerkomponente plötzlich ansteigt. Diese Arten von Anomalien deuten häufig auf einen bevorstehenden Systemzusammenbruch hin.

2.4 Machine- und Deep Learning

Um einen Einblick in die Thematik Machine- und Deep Learning zu geben, werden in diesem Kapitel wichtige Begrifflichkeiten definiert. Zu diesem Zweck werden die Unterschiede zwischen Algorithmen des maschinellen Lernens, des künstlichen neuronalen Netzen und des tiefen neuronalen Netzes vorgenommen. Die Unterschiede werden wie in der Abbildung 2.1 verdichtet dargestellt. [28]
Künstliche Intelligenz kann wie folgt definiert werden „Im Allgemeinen bezieht sich künstliche Intelligenz (KI) auf jede Methode, die es Computern ermöglicht, die menschliche Entscheidungsfindung nachzuahmen oder zu übertreffen, um schwierige Aufgaben autonom oder mit wenig oder ohne menschliche Beteiligung zu erledigen“ [29]. Der Einsatz von KI findet in der heutigen Zeit viele Anwendungszwecke genauso wie in diversen Forschungsbereichen. Wenn man heutzutage über intelligente Systeme (KI) spricht, wird meist maschinelles Lernen verwendet. Dabei erlangt das System die Fähigkeit, aus einer Menge von Trainingsdaten zu lernen, um eine bestimmte Herausforderung bzw. eine Aufgabe zu bewältigen. Dabei werden relevante Daten in ein Modell eingefügt und verarbeitet. Aus diesem maschinellen Lernen hat sich ein weiteres Konzept ergeben, das ’Deep Learning’ genannt wird. Dieses Konzept verwendet sogenannte tiefe neuronale Netze und übertrifft bei weiten das flache maschinelle Lernen und die herkömmlichen Techniken zur Datenanalyse.[28]

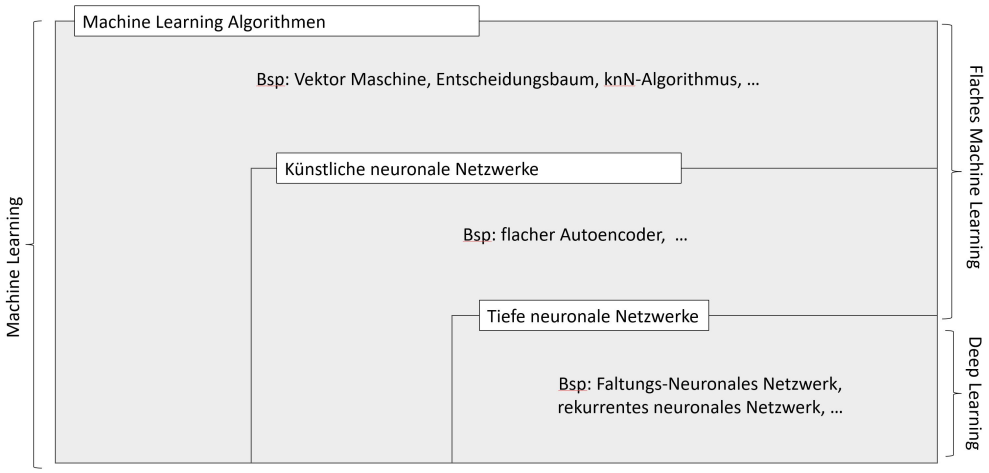


Abbildung 2.1: Übersicht Machine- und Deep Learning [28]

2.4.1 Machine Learning

Um kognitive Aufgaben wie die Erkennung von Objekten oder die Übersetzung natürlicher Sprache auszuführen, wird versucht, den Prozess der Erstellung analytischer Modelle zu automatisieren. Dies wird durch den Einsatz von Algorithmen erreicht, die wiederholt aus problembezogenen Trainingsdaten lernen und es Computern ermöglichen, komplizierte Muster und verborgene Erkenntnisse zu entdecken, ohne explizit programmiert zu werden. Clustering, Kategorisierung und Regressionen sind Beispiele für Anwendungsfälle für das maschinelle Lernen. Hierbei kann man durch Wiederholungen der früheren Berechnungen noch vertrauenswürdige Schlussfolgerungen treffen. Aus diesem Grund werden Algorithmen des maschinellen Lernens in einer Vielzahl von Bereichen effektiv eingesetzt, z. B. bei der Erkennung von Betrug, der Kreditwürdigkeitsprüfung, der Analyse des nächstbesten Angebots, der Audio- und Bilderkennung und der Verarbeitung natürlicher Sprache (Natural Language Processing (NLP)). Regressionsmodelle, Instanz-basierte Algorithmen, Entscheidungsbäume und Bayes'sche Ansätze sind nur einige der verschiedenen Arten von Algorithmen, die beim maschinellen Lernen zum Einsatz kommen. Je nach verfügbaren Daten und vorliegenden Problem kann man das maschinelle Lernen in folgende drei Typen unterteilen: [30]

- **Überwachtes Lernen:** Für den Trainingsdatensatz, ist es notwendig, dass die Eingabe mit Labels, Zielwerte bzw. Ergebnissen markiert sind. Durch die Verwendung von den beschrifteten Daten wird das Modell trainiert, um Daten richtig zu kategorisieren oder ein Ergebnis vorherzusagen. Sobald

das Modell erfolgreich trainiert wurde, kann es benutzt werden, um ein wahrscheinliches Ergebnis bzw. eine Vorhersage anhand der verwendeten Eingangsmerkmale zutreffen. Hinsichtlich der Art des überwachten Lernens kann zwischen Regressionsproblemen, bei denen ein numerischer Wert vorhergesagt wird, und Klassifizierungsproblemen, bei denen das Vorhersageergebnis eine kategoriale Klassenzugehörigkeit ist, wie z. B. 'normal' oder 'anormal', unterschieden werden.

- **Unüberwachtes Lernen:** Im Gegensatz zum überwachten Lernen, werden keine Beschriftungen bzw. Spezifikation verwendet. Die Trainingsdaten bestehen also nur aus einem Merkmal mit dem Ziel, strukturelle Informationen von Interesse zu finden. Clustering ist ein bekanntes Beispiel für unüberwachtes Lernen, beispielsweise werden Kunden oder Märkte in Segmente eingeteilt, um eine zielgruppenspezifische Kommunikation zu ermöglichen.
- **Reinforcement Learning:** Es werden keine Eingabe- und Ausgabepaare zur Verfügung gestellt, sondern der aktuelle Status des Systems wird bereitgestellt. Anschließend wird ein Ziel, welches das System erreichen soll, deklariert. Eine Liste zulässiger Aktionen und deren Umgebungsbedingungen für die Ergebnisse wird bereitgestellt. Das Modell versucht das Ziel mittels des Prinzips von 'Trial-and-Error' zu durchlaufen, um eine Belohnung zu maximieren. Bekannte Jump-n-Run Spiele können damit trainiert werden.

2.4.2 Künstliche neuronale Netze

Von biologischen Systemen, wie beispielsweise das Gehirn, inspiriert, sind künstliche neuronale Netze eine anpassungsfähige Form, um alle drei Arten von maschinellen Lernen flexibel zu ändern. Dabei werden Verarbeitungseinheiten mit mathematischer Darstellung, die sogenannten künstlichen Neuronen (ANN), miteinander verbunden. Jede Verbindung zwischen Neuronen erzeugt Signale, die während des Lernprozesses ständig verändert werden, ähnlich wie bei Synapsen im Gehirn. Hierbei können die Gewichte solcher Verarbeitungseinheiten verstärkt oder abgeschwächt werden. Eine Aktivierungsfunktion legt fest, ab welchen Schwellenwert ein bestimmtes Signal von den nachfolgenden Neuronen verarbeitet wird. Dabei hat dieses Netz meist mehrere angeordnete Ebenen von Neuronen, wobei eine Ausgangsebene nach Erhalt der Eingangsebene das Resultat erzeugt. Dazwischen liegt keine oder mehrere verborgene Schichten, die für die Ermittlung und Verarbeitung zuständig sind. Dabei entsteht eine nicht-lineare Zuordnung zwischen den Neuronen der jeweiligen Schichten. [31][28]

2.4.3 Tiefe neuronale Netze

Tiefe neuronale Netze sind künstliche neuronale Netze, die meist mehrere versteckte Schichten in ihren vernetzten Topologien haben. Zudem sind die Neuronen komplexer als die gewöhnlichen künstlichen neuronalen Netze. Anstelle einer einfachen Aktivierungsfunktion können sie beispielsweise ausgeklügelte Prozesse (wie Faltungen) oder zahlreiche Aktivierungsfunktionen in einem einzigen Neuron verwenden. Das Trainieren und Lösen mit solchen tiefen neuronalen Netzen wird Deep Learning bezeichnet. Deep Learning ist besonders hilfreich, wenn es um die Verarbeitung von hochdimensionalen Daten wie Text-, Bild-, Video-, Sprach- und Audiodaten geht. Es erschließen sich dadurch neue Anwendungsbereiche, da ein flaches maschinelles lernendes Verfahren nicht genügend ist.
Folgend sind einige Deep Learning Architekturen aufgelistet, die im Laufe der Zeit entwickelt wurden. Die unterschiedlichen Varianten lassen sich anhand der Art der Schichten, der neuronalen Einheiten und der Verbindungen unterscheiden. Jede Architektur ist je nach Anwendungsfall bzw. Datentyp besser geeignet als die andere. [32][28]

- **Convolutional neural Networks (CNN):** Diese Architekturvariante wird meist im Zusammenhang mit Computer-Vision und Spracherkennung eingesetzt, dabei ist die den anderen neuronalen Netzen überlegen. Dabei sind Bilder, Sprache oder Audio die Eingabedatensätze. Bei den Daten sind die Spalten und Zeilen nicht austauschbar. Die Netzwerkarchitektur umfasst eine Reihe von Stufen, die ein hierarchisches Lernen von Merkmalen ermöglichen, wie es durch die jeweilige Modellierungsaufgabe bestimmt wird. Die drei grundlegenden Typen an Schichten, die verwendet werden, sind die Convolutional-, die Pooling- und die Fully-connected Schicht. Dabei stellt die Faltungsschicht (Convolutional) die erste Ebene des Modells dar. Hierbei werden die grundlegenden Elemente wie Farben und Ränder betont. Danach können noch weiter Faltungsschichten oder ein oder mehrere Pooling-Schichten kommen. Das CNN wird mit jeder Schicht komplizierter und erkennt größere Bildbereiche. Die letzte Schicht bildet die Fully-connected Schicht, die dann die Merkmale schrittweise verdichtet, dass eine Ähnlichkeit mit den eigentlichen Objekten stattfindet.[33]
- **Recurrent Neural Networks (RNN):** Im Gegensatz zum CNN sind RNN für sequenzielle Datenstrukturen, wie eine Ordnung von Ereignissen, Zeitabfolgen oder natürliche Sprachen, konzipiert. Ihre Architektur bietet interne Rückkopplungsschleifen und ermöglicht daher sequenzielles Lernen von Mustern zur Modellierung zeitlicher Abhängigkeiten durch Bildung eines Gedächtnisses. Ein rekurrentes Netz zeichnen sich dadurch aus, dass jede Schicht die gleichen Parameter und denselben Gewichtungsparameter verwendet. Diese Gewichte werden mithilfe der Techniken der Backpropaga-

tion und des gradienten Abstiegs modifiziert. Einfache RNN-Architekturen sind problematisch, da sie unter verschwindenden Gradienten leiden, was dazu führt, dass frühe Erinnerungen wenig oder gar keinen Einfluss haben. Daher sollten höher entwickelte Architekturen wie Long-Short-Term-Memory (LSTM) oder Gated-recurrent-units (GRUs) genutzt werden. LSTM hat Neuronen bzw. Zellen mit drei Gates: einem Input-Gate, einem Output-Gate und einem Forget-Gate. Diese Gates regulieren den Informationsfluss und können, wie der Name schon verrät, die Information von vorhin in die Berechnung mit einbeziehen, um eine exakte Vorhersage zu treffen. GRUs ist vergleichbar mit LSTM und wurde entworfen, um das Kurzzeitgedächtnis von LSTM zu bewältigen. Dabei werden nur zwei Gates, wie ein Reset-Gate und ein Update-Gate, verwendet. Diese Gates regeln, welche Informationen gespeichert werden.[34]

- **Verteilte Repräsentationen:** Ist eine Architektur, die bei Sprachmodellierung in NLP-Aufgaben verwendet wird. Dabei bekommen die jeweiligen sprachlichen Einheiten, wie Wörter, Phrasen und Sätze eine einheitliche semantische, numerische Repräsentation. Worteinbettungen zum Beispiel kodieren diskrete Wörter in dichte Merkmalsvektoren mit geringer Dimensionalität. Im Gegensatz zu klassischen Textrepräsentationsmodellen wie One-Hot-Kodierungen und Bag-of-Words (BoW) überwinden Worteinbettungen das Problem der spärlichen Kodierung und erhalten gleichzeitig die semantischen Beziehungen zwischen den Wörtern. Damit ist gemeint, dass wenn Wörter im ähnlichen Kontext zueinanderstehen, dann sind diese auch im Vektorraum nah beieinander. Auf dieser Grundlage können fortgeschrittene Sprachmodelle entwickelt werden, um anspruchsvolle Aufgaben, wie die Beantwortung von Fragen oder die Analyse von Gefühlen, durchzuführen. Eine Kombination aus verteilter Repräsentation und RNN ist möglich, um sequenzielle Abhängigkeiten zu erkennen.[28]
- **Autoencoder:** Im Gegensatz zu der bereits genannten verteilten Repräsentation sind Autoencoder nicht auf natürlichsprachliche Daten beschränkt, obwohl die der Worteinbettung ähnlich sind. Dabei kann jede Art von Eingabe in eine niedrig dimensionale Repräsentation komprimiert werden. Eine Dekodierungsphase versucht, die ursprünglichen Daten zu rekonstruieren. Auf diese Weise ist das Netzwerk gezwungen, sinnvolle Informationen in der latenten Repräsentation beizubehalten, während irrelevantes Rauschen außer Acht gelassen wird. Anwendungsfall findet diese Variante beim unüberwachten Merkmals-Lernen und bei der Dimensionalität-Reduktion, somit können beispielsweise Anomalien erkannt werden.[28]
- **Transformer:** Bei der Verarbeitung natürlicher Sprache (NLP) in Deep Learning Projekten wird oft ein bestimmtes neuronales Netzwerkdesign verwendet, das als Transformator bezeichnet wird. Dieses wurde erstmals in einem Forschungsbericht namens 'Attention Is All You Need' von Vaswani

und andere im Jahr 2017 [35] vorgestellt. Im Gegensatz zu herkömmlichen Sequenz-zu-Sequenz-Modellen, die rekurrente neuronale Netze (RNNs) oder Faltungsneuronale Netze (CNNs) verwenden, nutzen Transformatoren sogenannte 'Self-Attention'-Prozesse zur Verarbeitung von Eingabesequenzen. Diese Selbstaufmerksamkeit erlaubt es dem Modell, sich auf verschiedene Elemente der Eingabesequenz zu konzentrieren, was zu einer genaueren Modellierung der langfristigen Beziehungen führen kann. Dieser Self-Attention-Mechanismus ist die Hauptkomponente eines Transformators, der eine gewichtete Summe der Eingabesequenz auf der Grundlage der Ähnlichkeit zwischen jedem Element und allen anderen Elementen berechnet. Diese gewichtete Summe wird als Eingabe an die nächste Schicht des Netzes weitergegeben. Um die Trainingsstabilität zu erhöhen, werden häufig Feedforward-Schichten und Schichtnormalisierung in Transformers verwendet. Transformers haben bei einer Vielzahl von NLP-Aufgaben, wie z.B. Textkategorisierung, maschinelle Übersetzung und Sprachmodellierung, Spitzenleistungen erzielt. Sie werden auch oft bei Computer-Vision Aufgaben wie Objekterkennung eingesetzt, indem der Mechanismus der Selbstaufmerksamkeit so modifiziert wird, dass er auf 2D-Gittern statt auf 1D-Sequenzen funktioniert.[35]

3 Stand der Forschung

Um einen Überblick über ein paar Ansätze in der Log-Anomalie-Erkennung zu erhalten werden im vorliegenden Abnschnitt die folgenden Ansätze aus der Literatur [3] behandelt. Diese Ansätze haben sich in den letzten Jahren als relevant und gegenwärtig aktuell erachtet. Der Ansatz TransLog [36] wurde darüber hinaus aufgenommen, da der Einsatz von Transformern in den letzten Jahren stark zugenommen hat und eine ausführliche Betrachtung dieses Ansatzes in diesem Kapitel unverzichtbar erschien.

3.1 DeepLog

Die wissenschaftliche Publikation von Du Min, Li Feifei, Zheng Guineng und Srikumar Vivek aus dem Jahr 2017 mit dem Titel "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning"[37] beschreibt einen Ansatz zur Anomalie-Erkennung und -Diagnose basierend auf einem LSTM-Modell (Long-Short-Term) namens DeepLog. Dabei wird die nächste zu erwartende Log-Nachricht anhand von vorangegangenen Ereignissen vorhergesagt. Die Architektur von DeepLog ist in der Abbildung 3.1 ersichtlich und besteht aus drei Hauptkomponenten bzw. Modellen:

- Modell zur Erkennung von Anomalien der Protokollschlüssel
- Modell zur Erkennung von Anomalien der Parameterwerte
- Workflow-Modell zur Diagnose der erkannten Anomalien

In der **Trainingsphase** (linke Seite) wird aus jedem Logeintrag ein Protokollschlüssel und ein Vektor aus den Parameterwerten extrahiert. Dabei werden die Schlüssel zum Training des Protokollschlüssel-Modells verwendet, um Diagnosezwecke für den Ausführungsgablauf zu generieren. Zusätzlich wird auch ein Modell anhand der Parameterwertvektoren trainiert, um mögliche Anomalien in der Systemleistung zu erkennen.
Aus einem neuen Logeintrag als Eingabe wird ebenfalls ein Protokollschlüssel und ein Parameterwertvektor extrahiert. Zuerst wird in der **Erkennungsphase** (rechte Seite) das Modell zur Erkennung von Anomalien der Protokollschlüsseln angewandt. Wenn keine Anomalie festgestellt wird, führt DeepLog eine Analyse der Parameterwertvektoren durch. Wenn eines der beiden Modelle ein Log als Anomalie erkennt, wird dieser als anomal deklariert und das Workflow-Modell bietet semantische Informationen für Benutzerin und Benutzer,

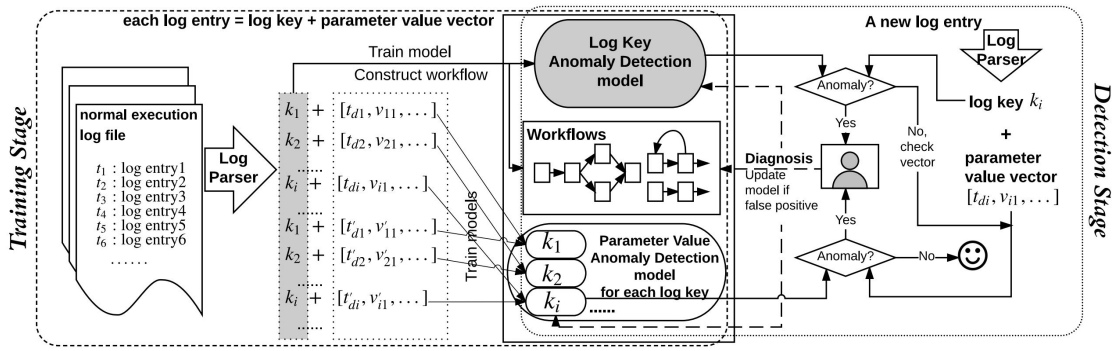


Abbildung 3.1: Architektur von DeepLog [37]

um die Anomalie zu identifizieren. Es kann passieren, dass sich der Ablauf in der Systemausführung ändert oder das Trainingsdaten fehlen. Zudem kann man Anomalien als false-positive kennzeichnen, um Feedback an das Modell zuliefern.

Ausführungspfad

Zunächst soll die Verwendung des Protokollschlüssel-Modells untersucht werden. Bei einem Quellcode existieren eine Anzahl von Print-Anweisungen, welche eine eindeutige Anzahl von Logs offenbaren. Diese Logs bestehen aus einer Sammlung von eindeutigen Protokollschlüsseln, welche vom System generiert werden und durch $K = \{k_1, k_2, ..., k_n\}$ dargestellt werden. Nach Extraktion aller Schlüssel aus den Logs erhält man eine Sequenz von Schlüsseln, welche den Ablauf darstellt. Dabei stellt m_i den Wert des Schlüssels in der logischen Schlüsselfolge an der Position i dar. Es ist offensichtlich, dass m_i einen der n möglichen Schlüssel von K erhalten kann.

DeepLog ist ein Klassifikationsalgorithmus, der zur Erkennung von Anomalien in Log-Dateien eingesetzt wird. Der Algorithmus verwendet eine MultiClass-Klassifizierung, bei der jeder Schlüssel eine Klasse identifiziert. Das Ziel ist es, eine Verteilung über die n Schlüssel von K zu erhalten, die angibt, mit welcher Wahrscheinlichkeit ein bestimmter Schlüssel k_i der nächste Schlüssel in der Sequenz ist. Um die Wahrscheinlichkeiten zu ermitteln, wird ein Fenster mit den aktuellsten Protokollschlüsseln h als Eingabe für die Kategorisierung verwendet. Dabei kann das Fenster w viele Instanzen des exakten Log-Schlüsselwerts enthalten.

In der Trainingsphase werden Ausgabepaare für jedes k_i in K erzeugt, um eine bedingte Verteilung der Wahrscheinlichkeiten $Pr[m_t = k_i|w]$ zu erhalten. Ein Beispiel für die Verwendung von DeepLog ist die Analyse einer winzigen, bei normaler Ausführung erzeugten Log-Datei, die in die folgenden Log-Schlüsselwerte zerlegt wird: $\{k_{22}, k_5, k_{11}, k_9, k_{11}, k_{26}\}$. Bei einer Fenstergröße von h = 3 würde das Eingabesequenz-

Fenster für das Training aus den Schlüsseln $\{k_{22}, k_5, k_{11}\}, \{k_5, k_{11}, k_9\}$ und $\{k_{11}, k_9, k_{11}\}$ bestehen, während die Ausgabepaare $\{k_{22}, k_5, k_{11} \rightarrow k_9\}, \{k_5, k_{11}, k_9 \rightarrow k_{11}\}$ und $\{k_{11}, k_9, k_{11} \rightarrow k_{26}\}$ wären.
In der Erkennungsphase von DeepLog wird eine Eingabe $w = \{m_{t-h}, ..., m_{t-1}\}$ eingegeben, um herauszufinden, ob ein eingehender Schlüssel m_t als normal oder anomal betrachtet wird. Als Ergebnis erscheint eine Wahrscheinlichkeitsverteilung $Pr[m_t|w] = \{k_1 : p_1, k_2 : p_2, ..., k_n : p_n\}$ für jeden Schlüssel aus K, der als nächster Log-Schlüsselwert in Abhängigkeit von der Historie erscheint. Man sollte sich bewusst sein, dass auch Logs wie *'Waiting for * to respond'* oder *'Connected to * '* auftauchen können, die den gleichen Zustand bedeuten. Solche Muster müssen dem DeepLog während des Trainings beigebracht werden. Hierbei sollte man auf ein traditionelles N-Gramm-Sprachmodell zurückgreifen, um natürliche Sprachen zu verarbeiten (NLP), um gleich bedeutete Informationen festzustellen.
DeepLog verwendet ein rekurrentes neuronales Netz (RNN), um die Modelle zu trainieren. Dabei wird der Ansatz von Long-Short-Term-Memory (LSTM) [38] verwendet, der bereits im vorherigen Kapitel 2 erklärt wurde.

Parameterwerte

Durch die Sequenz der Protokollschlüssel kann man ungewöhnliche Ausführungspfade feststellen. Doch es existieren ebenfalls Anomalien in unregelmäßigen bzw. unerwünschten Parameterwerten. Jeder Parameterwertvektor, der in einer bestimmten Zeitreihe stattfindet, wird zusammengefügt und in einen Vektor, für das Training des LSTM-Modells, eingefügt. Beispiele für einen Parameterwertvektor ist in der Tabelle 3.1 dargestellt.

Log Nachricht	**Log Schlüssel**	**Parameterwertvektor**
t_1 Deletion of file1 complete	k_1	$[t_1 - t_0, file1Id]$
t_2 Took 0.61 seconds to deallocate network	k_2	$[t_2 - t_1, 0.61]$
t_3 VM Stopped (Lifecycle Event)	k_3	$[t_3 - t_2]$
...	...	...

Tabelle 3.1: Beispiel von Parameterwertevektor [37]

Der Parameterwertvektor aus jeden Zeitstempel dient als Eingabe für jeden Zeitschritt. Der Durchschnitt und die Standardabweichung aller Werte von der gleichen Parameterposition in den Trainingsdaten werden verwendet, um die Werte in jedem Vektor zu normalisieren. Anschließend erhält man als Ausgabe einen Vektor mit Vorhersagen und Wahrscheinlichkeiten für den kommenden Parameterwertvektor. Dabei wird während dem Training, die Gewichte des LSTM-Modells so modifiziert, dass die Diskrepanz zwischen Vorhersage und beobachteten Parameterwertvektor verringert wird.

Zur Erkennung von Anomalien wird eine Methode verwendet, bei der der mittlere quadratische Fehler (MSE) zwischen vorhergesagten und beobachteten Parameterwertvektoren berechnet wird. Die Daten werden für die Berechnung prozentuell in Trainings- und Validierungsdaten aufgeteilt. In der Validierungsgruppe werden die Unterschiede zwischen den vorhergesagten und den tatsächlichen Vektoren bei jedem Zeitschritt als Gauß-Verteilung dargestellt. Ein eingehender Protokolleintrag wird als normal angesehen, wenn die Differenz zwischen der Vorhersage und dem beobachteten Wertvektor innerhalb eines hohen Konfidenzintervals der Gauß-Verteilung liegt, und als anomal, wenn dies nicht der Fall ist. Diese Methode kann mehrere Arten von Leistungsanomalien erkennen, da die Parameterwerte in einer Protokollmeldung auch wichtige Daten bezüglich Systemstatus erfassen kann. Eine Leistungsanomalie kann sich beispielsweise als 'Verlangsamung' manifestieren, und es ist wichtig zu beachten, dass die Zeitspanne zwischen aufeinanderfolgenden Protokolleinträgen von DeepLog in jedem Parameterwertvektor gespeichert wird. Durch die Modellierung des Parameterwertvektors als multivariate Zeitreihe kann das LSTM-Modell ungewöhnliche Muster in einer oder mehreren Dimensionen dieser Zeitreihe erkennen, wobei der verstrichene Zeitwert nur eine solche Dimension darstellt.

Workflow-Modell

DeepLog benötigt verschiedene Eingabeparameter wie beispielsweise die Top g Log-Schlüssel in der projizierten Ausgabe-Wahrscheinlichkeitsverteilungsfunktion sowie die Länge des Verlaufsfolgefensters h. Es ist besonders wichtig, eine gute Wahl bezüglich des Verlaufsfolgefensters zu treffen, da dies von der Problemstellung und den verfügbaren Daten abhängt. Ein größeres Verlaufsfolgefenster kann die Genauigkeit der Vorhersage verbessern, da mehr historische Daten im LSTM-Modell verwendet werden. Jedoch gibt es einen Punkt, an dem Schlüssel, die sehr weit zurückliegen und keinen Einfluss mehr auf die Vorhersage der auftretenden Schlüssel haben. Wenn das Verlaufsfolgefenster zu groß wird, kann dies zu einer Überlastung des Systems führen, da mehr Berechnungen erforderlich sind. Es ist also wichtig, eine optimale Größe des Verlaufsfolgefensters zu finden, um eine gute Leistung des DeepLog-Systems sicherzustellen.

Das Workflow-Modell kann hierbei behilflich sein, da ein Hinweis auf einen geeigneten Wert für h vorgeschlagen wird. Man kann h als die Länge des kürzesten Arbeitsablaufs festlegen, weil es intuitiv sinnvoll ist, dass h gerade groß genug sein muss, um alle relevanten Abhängigkeiten für eine solide Vorhersage zu berücksichtigen.

Das Workflow-Modell kann für jede von DeepLog entdeckte Anomalie verwendet werden, um die Diagnose der Anomalie zu unterstützen und festzustellen, wie und warum sie aufgetreten ist. Ein Beispiel ist in Abbildung 3.2 ersichtlich. Hier wäre die Vorhersage unter der Sequenz $[26, 37, 38]$ der Log-Schlüssel 39, aber

es erscheint der Log-Schlüssel 67, was eine Anomalie darstellt. Mithilfe eines Workflow-Modells können Benutzerinnen und Benutzer schnell den aktuellen Ausführungspunkt in dem betreffenden Workflow ermitteln und erfahren. Für die Aufgabe in der Abbildung 3.2 ist zu erkennen, dass dieses Problem während der Bereinigung nach der Zerstörung einer VM auftrat, genau nach *'Instance destroyed successfully'* und vor *'Deleting instance files ∗'*.

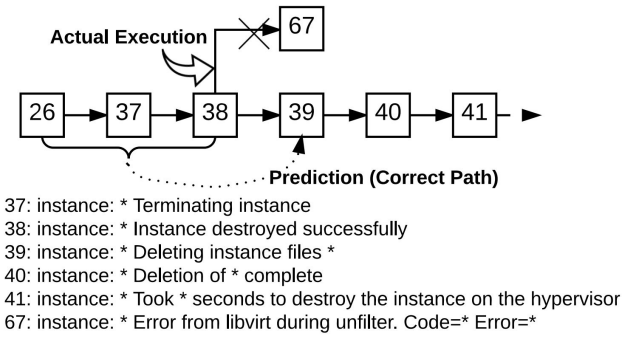


Abbildung 3.2: Anomalie-Diagnose mittels Workflow [37]

3.2 LogAnomaly

In dem Jahr 2019 veröffentlichten Meng Weibin, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun und Rong Zhou den Artikel "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs"[39], der einen weiteren Ansatz zur Erkennung von Anomalien in unstrukturierten Logs vorstellt. Ähnlich wie bei DeepLog wurde auch bei LogAnomaly ein Vorhersage-basiertes Modell entwickelt, um das nächste Log-Ereignis zu ermitteln. Sobald das untersuchte Log-Ereignis jedoch gegen die Vorhersageergebnisse verstößt, wird es als Anomalie markiert. Der Ansatz von LogAnomaly basiert auf den Techniken der natürlichen Sprachverarbeitung (NLP). Da ein System viele unstrukturierte Texte produziert, die oft nicht gekennzeichnet sind, können die unterschiedlichen Muster der Logs durch die Semantik eines Templates dargestellt werden. Diese Templates dienen als Trainingsdaten und werden anschließend für die Erkennung verwendet. Logs, die von den Templates abweichen, werden als anomal gekennzeichnet.

Überblick

Das Design von LogAnomaly ist in Abbildung 3.3 ersichtlich und besteht aus einer Trainings- und einer Erkennungsphase. In der **Trainingsphase** wird FT-Tree [40] verwendet, um die Templates aus vorherigen

Logs zu extrahieren. Anschließend werden die extrahierten Templates mit historischen Logs verglichen. Die Templates werden mittels template2Vec, eine Technik, die Templates auf verteilte Weise ausdrückt, in Worteinbettungsvektoren umgewandelt. Dabei wird durch Zusammenführung von Wortvektoren ein Templatevektor berechnet. Weiters wird eine Log-Sequenz zu einer Templatevektor-Sequenz umgewandelt. Anschließend werden die Vektorensequenzen verwendet, um ein LSTM-Modell zu trainieren. Dabei wird in regelmäßigen Abständen das Modell neu trainiert, um neue Templates zu erlernen.

In der **Erkennungsphase** werden Echtzeit-Logs mit den bestehenden Templates verglichen. Wenn ein Log mit einem Template in Verbindung gebracht werden kann, wird es zu einem Templatevektor konvertiert. Andersrum wird es basierend auf Ähnlichkeiten der anderen Templatevektoren, zu einem „temporären“ Templatevektor angenähert (approximiert). Durch den Vergleich werden die Echtzeit-Logs in Vektorsequenzen umgewandelt. LogAnomaly analysiert, ob eine Log-Sequenz anomal ist, basierend auf dem gelernten LSTM-Modell in der Trainingsphase.

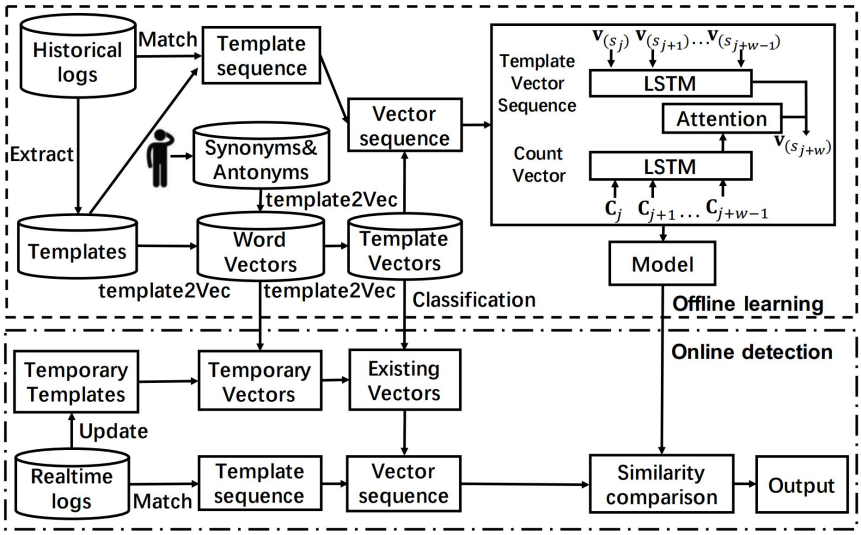


Abbildung 3.3: Übersicht von LogAnomaly [39]

Template2Vec

Template2Vec wurde von der Inspiration von Word2Vec [41] entwickelt. Word2Vec ist ein Verfahren zur Erstellung einer verteilten Darstellung von Wörtern, indem Vektoren erstellt werden, die die Eigenschaften und den Kontext von Wörtern darstellen. Jedoch werden bei Wörtern, die Synonyme, wie ’down’ und ’low’ oder Antonyme, wie ’down’ und ’up’, sind, der semantische Inhalt nicht erfasst. In vielen Fällen können Begriffe mit ähnlichem Kontext in zwei verschiedenen Log-Templates Antonyme sein, was dazu führt,

dass sich die beiden Templates stark voneinander unterscheiden. Daher wird versucht, bei der Erstellung von Vektoren zur Darstellung von Templates die semantischen Informationen zu erfassen. Hier setzt die neuartige Wortrepräsentationstechnik 'template2Vec' an, die auf Synonymen und Antonymen basiert und Begriffe in den Templates effizient beschreiben soll. Template2Vec besteht aus den folgenden drei Schritten, die in Abbildung 3.4 gezeigt werden:

1. Zuerst wird eine Liste aus Synonymen und Antonymen erstellt. Dabei werden Datenbanken (WordNet eine Lexikon-Datenbank [42]) mit den jeweiligen Informationen verwendet. Einige domänenspezifische Synonyme und Antonyme müssen manuell mit dem Fachwissen hinzugefügt werden.
2. Anschließend werden die Wortvektoren erzeugt. Die Autoren der Literatur verwenden hierbei dLCE [43], ein bekanntes Modell zur Einbettung des verteilten Vokabulars.
3. Als letzter werden die Templatevektoren generiert. Dabei wird der gewichtete Durchschnitt der Wortvektoren der Wörter für das Template genommen. Für neue Arten von Protokollen wird während der Erkennungsphase ein 'temporärer' Templatevektor mit den bestehenden verglichen.

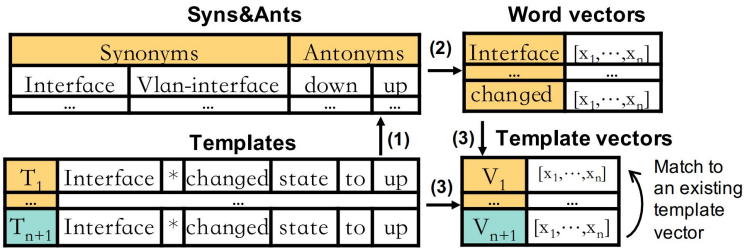
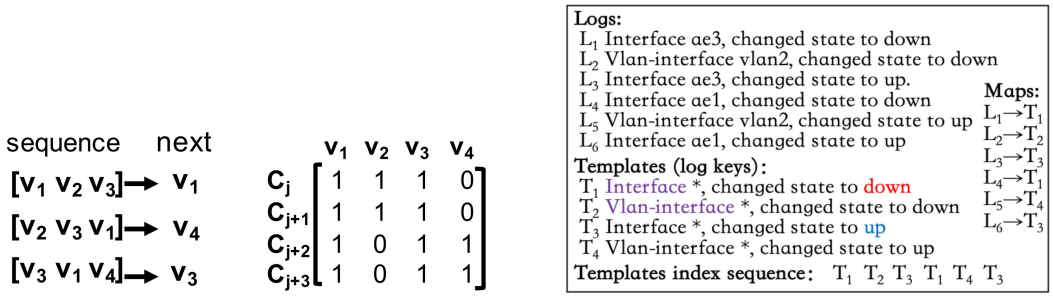


Abbildung 3.4: Beispiel von Template2Vec [39]

In einem System wird in der Regel ein bestimmter Ablauf befolgt, bei dem häufig durch Print-Funktionen Logs erzeugt werden. Auf diese Weise kann festgestellt werden, dass normale Logs einem bestimmten sequenziellen Muster folgen. Anders ausgedrückt tritt keine Anomalie in der Folge auf, wodurch das nachfolgende Template vorhersehbar ist. Die Menge der eindeutigen Templatevektoren wird als $\Omega = \{v_1, v_2, ..., v_n\}$ bezeichnet. Ähnlich wie bei DeepLog wird ein Fenster von w aktuellen Templatevektoren verwendet, das als Erkennungssequenz dient.

Die Abbildung 3.5 (a) zeigt eine sequenzielle und quantitative Anordnung von den Logs, die in der Abbildung 3.5 (b) gezeigt werden. Beispiel ist v_1 höchstwahrscheinlich der folgende Vektor von $[v_1, v_2, v_3]$. Wie schon erwähnt verwendet LogAnomaly wie DeepLog ein LSTM-Netzwerk [38], um die Muster von den Logs zu lernen.



(a) (b)

Abbildung 3.5: Sequenzielle und quantitative Darstellung von Programmverhalten [39]

Die vorliegende Abhandlung beschäftigt sich mit der Analyse von Templatevektor-Sequenzen, die neben sequenziellen Mustern auch quantitative Muster enthalten. Ein typisches Programmverhalten beinhalten dabei einige Invarianten, die über eine Reihe von Eingaben und Arbeitsbelastungen hinweg bestehen bleiben, während quantitative Beziehungen zwischen den Protokollen vorliegen. Zum Beispiel lässt sich feststellen, dass jede geöffnete Datei irgendwann geschlossen wird, weshalb die Anzahl der Protokolle, die das Öffnen einer Datei anzeigen, in einer typischen Situation gleich der Anzahl der Protokolle sein sollte, die das Schließen einer Datei anzeigen. Durch die Erfassung dieser quantitativen Beziehungen in den Protokollen kann das typische Programmausführungsverhalten erfasst werden. Im Falle einer Anomalie während der Systemausführung, bei der ein neues Protokoll gegen einige Invarianten verstößt, kann darauf geschlossen werden, dass eine Anomalie aufgetreten ist. Um das quantitative Muster von S_j zu lernen, kann ein Zählvektor der Log-Sequenz $(s_{i-w+1}, s_{i-w+2}, ..., s_i)$ für ein Log s_i erstellt werden. Dieser wird geschrieben als $C_i = (c_i(v_1), c_i(v_2), ..., c_i(v_n))$, wobei $c_i(v_k)$ die Anzahl von v_k in der Templatevektor-Sequenz $(v_{(s_i-w+1)}, v_{(s_i-w+2)}, ..., v_{(s_i)})$ ist. Schließlich wird $C_j, C_{j+1}, ..., C_{j+w-1}$ als Eingabe in das LSTM übergeben, um das quantitative Muster von S_j zu erlernen.

Die Programmausführung hat in der Regel verschiedene Verzweigungen, wie zum Beispiel das Öffnen, Lesen, Schreiben und Schließen einer Datei. Dies führt zu einer Vielzahl von Möglichkeiten für den nächsten Template-Vektor, der auf den quantitativen und sequenziellen Mustern der Programmausführung und Protokolle basiert. Um die Wahrscheinlichkeit des potenziellen nächsten Template-Vektors in einer Log-Sequenz zu bestimmen, wird ein LSTM-Modell verwendet. Wenn der beobachtete nächste Template-Vektor zu den besten k Möglichkeiten gehört oder mit diesen vergleichbar ist, wird dieser als normal betrachtet.

3.3 PLELog

Die vorliegende wissenschaftliche Arbeit mit dem Titel 'Semi-supervised Log-based Anomaly Detection via Probabilistic Label Estimation' von Lin Yang et al. (2021) [44] beschäftigt sich mit dem Problem der Anomalie-Erkennung mittels eines halbüberwachten Ansatzes namens PLELog. Dieser Ansatz basiert darauf, dass nur eine Teilmenge der Trainingsdaten beschriftet ist und dass historisches Wissen über Anomalien mithilfe des PU-Lernens (Positive and Unlabeled) [45] in die Anomalie-Erkennung miteinbezogen wird.
Die Trainingsdaten bestehen aus einer Sammlung von Log-Sequenzen, die anomale und normale Logs enthalten können, wobei die Beschriftungen der Logs bereits vorhanden sein kann oder nicht. PLELog verwendet probabilistische Schätzungen, um die Beschriftungen der nicht gekennzeichneten Log-Sequenzen zu schätzen. Diese geschätzten Beschriftungen werden dann verwendet, um ein überwachtes Modell zur Erkennung von Anomalien zu entwickeln. Insgesamt stellt diese Arbeit einen wichtigen Beitrag zur Anomalie-Erkennung dar, insbesondere in Fällen, in denen nur eine begrenzte Anzahl von beschrifteten Trainingsdaten vorhanden sind. Die Verwendung von PU-Lernen und probabilistischen Schätzungen trägt zur Verbesserung der Genauigkeit und Effektivität der Anomalie-Erkennung bei und bietet somit eine vielversprechende Methode zur Identifizierung von Anomalien in Log-Daten.

Architektur

Das Modell von PLELog besteht aus drei Schritten, (1) Semantische Einbettung, (2) probabilistische Label-Schätzung und (3) die Entwicklung eines Modells zur Erkennung von Anomalien, diese sind in der Abbildung 3.6 ersichtlich.

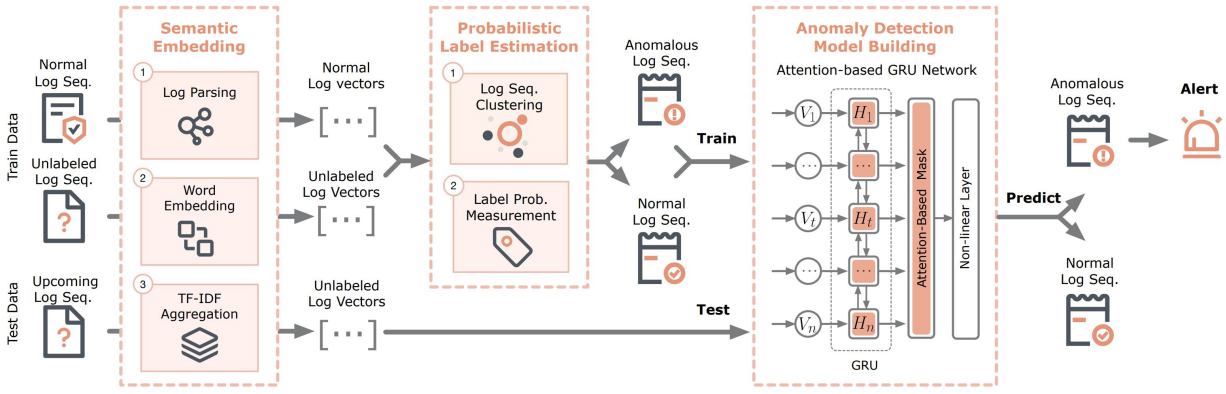


Abbildung 3.6: Architektur von PLELog [44]

Semantische Einbettung

Um instabile Logs effektiv zu handhaben, konvertiert PLELog ein Log zu einem sogenannten semantischen Vektor, indem die inhaltliche Semantik extrahiert wird. Die drei Schritte, die hierbei angewandt werden, sind wie folgt:

1. Parsen von Protokollen: Um die automatische Log-Analyse zu erleichtern, müssen die unstrukturierten Logs mit den enthaltenden Parametern zu einer einheitlich strukturierten Form organisiert werden. PLELog verwendet den Ansatz Drain [17], da er sich als effektiv und effizient erwiesen hat. Dieser Log-Parsing Ansatz wurde bereits im Abschnitt 2 betrachtet.
2. Wort-Einbettung: Jeder Log ist ein Satz in natürlicher Sprache, um die semantischen Informationen aus dem Log zu extrahieren wird ein Vektor erstellt. Ein Log besteht aus Begriffen wie beispielsweise englischen Wörtern und aus Nicht-Zeichen-Token (z. B. Begrenzungszeichen, Operatoren, Satzzeichen und numerische Ziffern). Bevor die Logs zu einem Vektor konvertiert werden, tut PLELog die Logs vorverarbeiten. Dabei werden die Nicht-Zeichen-Token und die sogenannten Stoppwörter entfernt und die zusammengesetzte Wörter werden separiert. Um dann die semantischen Informationen aus jedem Wort in einem verarbeiteten Log-Event zu extrahieren, verwendet PLELog das vortrainierte word2vec-Modell [41], um inhärente Verbindungen zwischen Wörtern zu erfassen. Dieses Modell basiert auf dem Common Crawl Corpus und die FastText Methode [46]. Als Ausgabe erhält man einen d-dimensionalen Vektor, wobei d=300 in FastText-Wortvektoren ist.
3. TF-IDF-basierte Aggregation: Nachdem ein Log zu einem d-dimensionalen Vektor konvertiert wurde, werden nun alle Wortvektoren in dem Log aggregiert. PLELog verwendet TF-IDF [47], ein beliebtes Verfahren zur Gewinnung von Informationen, indem die Relevanz jedes Worts betrachtet wird. Die Term-Frequenz (TF) schaut nach, wie oft ein bestimmtes Wort in einem Log-Ereignis vorkommt. IDF (Inverse Document Frequency) bezieht sich auf die Häufigkeit eines Wortes in allen Log-Ereignissen. Es wird geschaut, wie üblich oder selten ein Wort ist. Die Gewichtung eines Wortes kann mit $TF \times IDF$ dargestellt werden. Der letzte Schritt besteht darin, den semantischen Vektor V eines Protokollereignisses zu erstellen, indem alle Wortvektoren innerhalb des Log-Ereignisses unter Verwendung der TF-IDF-Gewichte wie in der Formel 3.1 addiert werden.

$$V = \frac{1}{N} \sum_{i=1}^{N} \omega \cdot v \tag{3.1}$$

Probabilistische Label-Schätzung

Nach der semantischen Einbettung werden die Beschriftungen der nicht gekennzeichneten Logs im Trainingsdatensatz geschätzt. Als Anhaltspunkt für die Schätzung dienen die beschrifteten Log-Sequenzen. Log-Sequenzen mit vergleichbarer Semantik sollten theoretisch mit größerer Wahrscheinlichkeit das gleiche

Label haben. Clustering und Wahrscheinlichkeit-Messungen sind die beiden Schritte, die in der probabilistischen Label-Schätzung passieren.

1. Log-Sequenz-Clustering: Als Clustering Algorithmus verwendet PLELog HDBSCAN (Hierarchical Density-Based Spatial Clustering of Application with Noise) [48]. Dabei werden alle Logs im Trainingsdatensatz je nach Semantik in verschiedene Gruppen geteilt. Dazu gehören sowohl bekannte Log-Sequenzen als auch unmarkierte Log-Sequenzen. Die Autoren haben sich aus folgenden Gründen für HDBSCAN entschieden. Es ist schwierig die Anzahl der Cluster im Voraus zu bestimmen, darum wurde ein Density-basierter Ansatz benötigt. Zudem wurde dieser schon in mehreren Bereichen verwendet und hat sich als erfolgreich und effizient erwiesen. Da er auch nur wenige Parameter braucht, ist er einfach zu verwenden. Zu beachten ist, dass durchzunehmende Datenmenge und Vektordimension die Effektivität sinken. Daher führt PLELog eine Dimensionsreduktion vor dem Clustering durch, um die Effizienz zu erhöhen.

   PLELog verfügt über eine FastICA-Methode [49], die versucht, unabhängige Komponenten zu finden und die Entdeckung der zugrundeliegenden Elemente durch Optimierung der negativen Entropie zu erleichtern.
2. Messung der Label-Wahrscheinlichkeit: Durch das Clustering wurden Gruppen mit ähnlicher Semantik erstellt. PLELog versucht nun auf Grundlage, der markierten Log-Sequenzen in den Gruppen, die Beschriftung der unmarkierten Logs zu schätzen. Es ist wahrscheinlicher, dass die Log-Sequenzen in einer Gruppe dasselbe Label haben. Wenn die Gruppe als Mehrheit 'normal' bzw. nicht-Anomalie beschriftete Log-Sequenzen umfasst, werden die nicht-markierten Logs mit einer hohen Wahrscheinlichkeit auch als normal angesehen; andernfalls werden sie eher als anomal betrachtet. Entsprechend den Ergebnissen der Clusterbildung berechnet PLELog genauer die Wahrscheinlichkeit, dass eine nicht beschriftete Log-Sequenz jedem Label entspricht. Jede Log-Sequenz in einer Gruppe erhält von HDBSCAN eine Punktzahl, die angibt, wie sicher die Gruppe ist, zu der die Log-Sequenz gehört. Der Score reicht von 0 bis 1, und je näher der Wert bei 0 liegt, desto sicherer kann eine Log-Sequenz der entsprechenden Gruppe zugeordnet werden. Es wird jedes vorläufig geschätzte Label in ein probabilistisches Label umgewandelt, indem die beiden Formeln $P(anomalous) = 1 - \frac{score}{2}$ und $P(normal) = \frac{score}{2}$ verwendet werden.

Modell zur Erkennung von Anomalien

In der vorliegenden Literatur wird ein aufmerksamkeitsbasiertes neuronales GRU-Netzwerk (Gated recurrent unit) vorgeschlagen, um ein effektives und effizientes Modell zu erstellen. Zur Erstellung dieses Modells dienen probabilistisch markierte Logs-Sequenzen als Trainingssatz. Dabei handelt es sich um ein rekurren-

tes neuronales Netzwerk, das speziell für die effiziente Verarbeitung von sequenziellen Daten konzipiert ist. Wie bereits im Kapitel 2 definiert, besteht eine GRU-Zelle aus einem Reset-Gate und einem Update-Gate, die bestimmen, wie viele Informationen von einem Zustand in einen anderen gesendet werden und wie viele Informationen aus der Vergangenheit verloren gehen sollen. Im vorliegenden Modell wird der semantische Vektor V_t als Eingabe des GRU-Netzwerks verwendet. Dieser Vektor bezieht sich auf das t-te Log-Ereignis von $e_t(1 \leq t \leq n)$ in einer Log-Sequenz $S = \{e_1, e_2, ..., e_n\}$, wobei n die Gesamtzahl der Log-Ereignisse in S darstellt und t zwischen 1 und n liegt.
Um das Aktualisierungsgatter z_t und das Rücksetzgatter r_t zum t-ten Zeitschritt in Übereinstimmung mit dem verborgenen Zustand H_{t-1} zum $(t-1)$-ten Zeitschritt zu bestimmen, werden die folgenden Formeln 3.2 verwendet.

$$\begin{aligned} z_t &= \sigma(W_z V_t + U_z H t - 1) \\ r_t &= \sigma(W_r V_t + U_r H t - 1) \end{aligned} \tag{3.2}$$

Der letzte verborgene Zustand H_{t-1} und das aktuelle Log-Ereignis sind die wichtigsten Faktoren für die Ausgabe des Modells. Die Effizienz der Anomalie-Erkennung kann sich verschlechtern, wenn eine Protokollsequenz eine übermäßige Anzahl von Protokollereignissen enthält, da das Problem der langen Abhängigkeit besteht. Durch die weitere Integration einer aufmerksamkeitsbasierten Maskierungstechnik in das neuronale GRU-Netzwerk geht PLELog diese Probleme an. Im Grunde wird geschaut wie stark Assoziation zwischen Log und Ergebnis der Anomalie-Erkennung ist. Anders ausgedrückt, je höher die Korrelation zwischen dem verborgenen Zustand H_i und dem Ergebnis der Anomalie-Erkennung ist, desto wahrscheinlicher handelt es sich um eine Anomalie. Auf diese Weise können verrauschte Log-Ereignisse durch eine geringere Gewichtung verschleiert werden, während wesentliche Log-Ereignisse durch eine höhere Gewichtung hervorgehoben werden können. Um abschließend das Ergebnis einer Log-Sequenz zu berechnet wird eine Aktivierungsfunktion $tanh$ verwendet, wobei H_N^A als Eingabe dient und W_n als Gewichtsmatrix, die während dem Training erstellt wurde, somit ergibt sich die folgende Formel 3.3.

$$P(normal, anomalous) = tanh(W_n H_N^A) \tag{3.3}$$

3.4 LogRobust

Die Literatur 'Robust log-based anomaly detection on unstable log data' von Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, und Dongmei Zhang aus dem Jahr 2019 [50] erklärt den Ansatz LogRobust. Dabei handelt es sich um ein sehr ähnliches Verfahren wie bei dem bereits definierten Ansatz PLELog [44]. LogRobust verwendet wie PLELog die semantischen Informationen aus einem Log, um diese anschließend in einem Vektor festzuhalten. Ebenfalls wird, wie bei PLELog ein aufmerksamkeitsbasiertes neuronales Netzwerk verwendet. Dieses Netzwerk kann verschiedene Protokollereignisse entsprechend ihrer relativen Bedeutung kategorisieren und kann kontextuelle Informationen erlernen. Auf diese Weise kann die Methode mit verirrten Protokollsequenzen umgehen. Die Architektur von LogRobust in Abbildung 3.7 besteht aus drei Schritten, (1) Log-Parsing, (2) Semantische Vektorisierung und (3) Aufmerksamkeitsbasierte Klassifizierung.

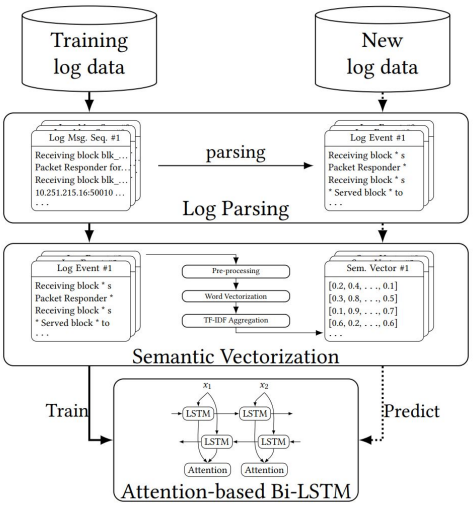


Abbildung 3.7: Architektur von LogRobust [50]

Log Parsing

Jedes unstrukturierte Log-Ereignis muss in eine einheitlich strukturierte Form gebracht werden, um eine automatische Analyse nicht zu behindern. Wie bei PLELog verwendet auch hier LogRobust den Log-Parser Drain [17], um die Genauigkeit und Effizienz zu maximieren. Zum Beispiel wird die rohe Protokollmeldung *'Receiving block blk 7503483334202473044 src:/10.251.215.16:55695 dest:/10.251.215.16:50010'* in das

Protokollereignis *'Receiving block * src: * dest: *'* konvertiert.

Semantische Vektorisierung

Die geparsten Logs werden anschließend in Vektoren mit fester Dimension umgewandelt. Dabei werden zuerst die Logs vorverarbeiteten, um beispielsweise Stoppwörter und Nicht-Wörter zu entfernen und um zusammengefügte Wörter wieder aufzutrennen. Wie bei PLELog werden diese dann anhand semantischer Informationen in einen Vektor konvertiert. Dabei behilft sich LogRobust mit der FastText-Methode [46], der auf dem Common Crawl Corpus-Datensatz trainiert ist. Abschließend wird dann mithilfe der TF-IDF-basierte Aggregation [47] berechnet, wie oft ein bestimmtes Wort im Log und in der gesamten Log-Sequenz vorkommt. Daraus ergibt sich eine Gewichtung für ein Wort.

Aufmerksamkeitsbasierte Klassifizierung

Um mit den instabilen Log-Sequenzen umzugehen, verwendet LogRobust das aufmerksamkeitsbasierte neuronale Bi-LSTM-Netzwerk (bidirektionales Long-Short-Term-Memory). Dabei dienen die semantischen Vektorsequenzen als Eingabe. Ein LTSM-Netzwerk kann die kontextuellen Informationen der Sequenz erfassen und kann somit ein zeitlich dynamisches Verhalten zeigen. Ein LSTM besteht aus einer Eingabeschicht, einer Schicht für versteckte Neuronen und einer Ausgabeschicht. Um genügend Informationen aus den eingegebenen Log-Sequenzen in beiden Richtungen zu sammeln, unterteilt das Bi-LSTM die Schicht der versteckten Neuronen eines regulären LSTM in zwei Richtungen (bidirektional): den Vorwärtspass und den Rückwärtspass. Demnach sind h_t^f undh_t^b die versteckten Zustandsvektoren im Vorwärts- bzw. Rückwärtsdurchlauf, wobei der Zeitschritt t die Position der Eingabeprotokollsequenz bezeichnet. Um Informationen aus beiden Richtungen zu erhalten, werden die beiden verborgenen Zustände als $h_t = concat(h_t^f, h_t^b)$ verkettet. Wie bei PLELog wird ein Aufmerksamkeitsmechanismus hinzugefügt, um die Relevanz eines Log-Ereignisses zu einer bestimmten Zeit festzulegen. Dabei wird die Gewichtung angepasst. Je höher die Gewichtung, desto mehr wird dieses Ereignis vom Modell berücksichtigt.

3.5 CNN

Ein weiterer Ansatz basierend auf einem Convolutional Neuralen Netzwerk (CNN) bzw. Faltungsnetzwerk, wird in der Literatur 'Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network' von Siyang Lu, Xiang Wei, Yandone Li und Liqiang Wang aus dem Jahr 2018 [51] vorgestellt. Obwohl es sich um eine allgemeine Methode handelt, wurde sie in der Literatur [3] zusammen mit anderen Methoden

wie DeepLog und LogAnomaly erwähnt. Daher ist es wichtig, dass auch dieser Ansatz in dieser Arbeit berücksichtigt wird.

CNN wurde entwickelt, um lokale semantische Informationen anstelle von globalen Informationen zu erfassen und die Probleme der Überanpassung in gewöhnlichen neuronalen Netzen zu überwinden. Bei der Faltungs-Technik, die in Faltungsschichten verwendet wird, werden Merkmale aus nahen gelegenen rezeptiven Feldern auf Merkmalskarten aus der vorherigen Schicht gezogen. Mithilfe einer Aktivierungsfunktion, z. B. Sigmoid, ReLU (Rectified Linear Units) oder Tanh, wird eine nicht lineare Transformation durchgeführt. Mit der Formel 3.4 kann man den Wert einer Einheit an der Position (m, n) in der j-ten Merkmalskarte der i-ten Schicht berechnen. Hierbei ist P_i die Höhe des Kerns, Q_i seine Breite, w_{ij}^{pq} der Parameterwert, b_{ij} die Bias-Funktion der Merkmalskarte und N die Gesamtheit der Merkmalskarten in der $(i-1)$-ten Schicht.

$$v_{ij}^{mn} = \sigma(b_{ij} + \sum_{N} \sum_{p=0}^{P_{i-1}} \sum_{q=0}^{Q_{i-1}} \omega_{ij}^{pq} v_{(i-1)N}^{(x+p)(y+q)}) \tag{3.4}$$

Das CNN-Modell kann zur Satzklassifizierung verwendet werden. Da es sich bei Logs um eine Art von Text handelt, kann die Protokollanalyse von den Fortschritten der NLP-Ansätze profitieren. Die Log-Analyse unterscheidet sich von der normalen Verarbeitung natürlicher Sprache (NLP). Im Gegensatz zu langen und komplexen Aussagen, die häufig in natürlichen Kontexten vorkommen, enthalten Protokolle nur wenige Schlüsselwörter. Während NLP darauf abzielt, verschiedene Phrasen in Gruppen einzuteilen, geht es bei der Log-Analyse darum, ungewöhnliche Abläufe in den Protokollen (Log-Schlüsselfolgen) zu erkennen. Das ist eine Art binäre Klassifizierung.

Überblick

Die Studie wandeln zuerst die Protokollschlüssel in einer Sitzung in Vektoren um, indem eine trainierbare Matrix oder ein sogenanntes Codebuch mit 29x128 Einträgen in der Einbettungsschicht erstellt wird. Zum Beispiel wird der Protokollschlüssel 5 in der Sitzungsgruppe $5, 5, 11, ..., 26, 26$ in der Einbettungsphase in die Werte $0,6312, 0,7192, ..., 0,9887$ kodiert. Die gesamte Sitzung wird als Matrix gespeichert. Diese Technik heißt logkey2vec. Jeder Logkey erstellt Log-Einbettungen basierend auf dem 29x128 Codebuch, anstatt Wörter als feinere Einheiten zu verwenden. Es wird eine trainierbare Schicht verwendet, die während des Trainings des neuronalen Netzes den Gradienten-Abstieg verwendet. Faltungsschichten sind die nächste Stufe des CNN. Dabei werden gleichzeitig drei einschichtige Faltungen (Filter) auf die eingebetteten Log-Vektoren anwendet. Für das CNN-Training nach der Kodierungsschicht werden drei parallele Faltungsschichten mit einer Größe von jeweils 3x128, 4x128 und 5x128 verwendet. Eine Leaky-ReLU wird

als Aktivierungsfunktion verwendet, um Überanpassung zu verhindern. Eine Max-Pooling-Schicht wird verwendet, um die Ausgabe der Faltungsschichten zu verketten und die Überanpassung zu reduzieren. Eine Dropout-Funktion wird in der vorletzten Schicht als Regularisierung verwendet. In der Ausgabeschicht wird eine Softmax-Funktion eingefügt. Abbildung 3.8 zeigt das Verfahren des CNN-Basismodells nochmals grafisch im Detail.

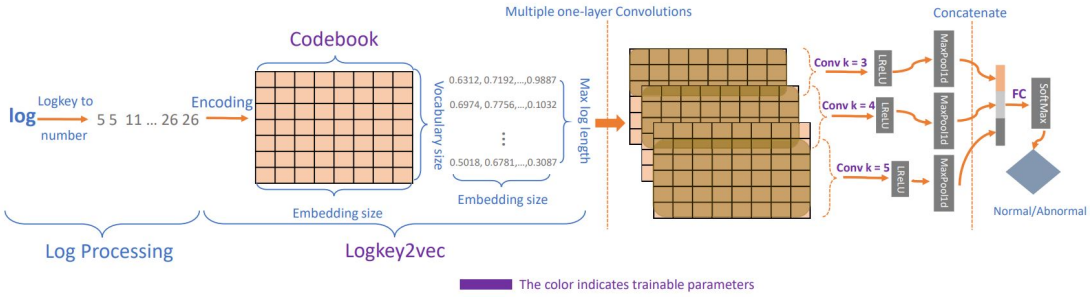


Abbildung 3.8: Übersicht von dem CNN-Ansatz [51]

3.6 TransLog

Der letzte Ansatz, der in dieser Arbeit genannt wird, benutzt einen sogenannten Transformer für die Erkennung von Anomalien in Systemprotokollen. Konkret handelt es sich hierbei um das Werk 'TransLog: A Unified Transformer-based Framework for Log Anomaly Detection' aus dem Jahr 2022 [36]. Das Werk stellt einen neuartigen Ansatz vor, der auf der Transformer-Architektur basiert, ein beliebtes Modell für Aufgaben im Bereich der natürlichen Sprachverarbeitung. Der Transformer hat sich als effektiv erwiesen, um langfristige Abhängigkeiten in sequenziellen Daten zu erfassen, was ihn für die Analyse von Systemprotokollen, die typischerweise in chronologischer Reihenfolge organisiert sind, gut geeignet macht. Insgesamt ist TransLog ein vielversprechender Ansatz zur Automatisierung der Protokollanalyse.

Überblick

Die Abbildung 3.9 zeigt den Aufbau von TransLog, der aus zwei Stufen besteht (1) Pre-Training und (2) adapterbasiertes Tuning. Um die Repräsentationen zu extrahieren, werden zunächst die gesamte Log-Sequenz in die das vortrainierte Sprachmodell, in der ersten Stufe Pre-Training, geladen. Der ressourcenreiche Datensatz der Quelldomäne wird verwendet, den Encoder des Transformers zu trainieren und gemeinsame semantische Daten zu sammeln. Um schließlich Informationen aus der Quelldomäne in die Zieldomäne zu übertragen, wird der Encoder eingerichtet und die Einstellungen des Adapters werden nur für den Datensatz

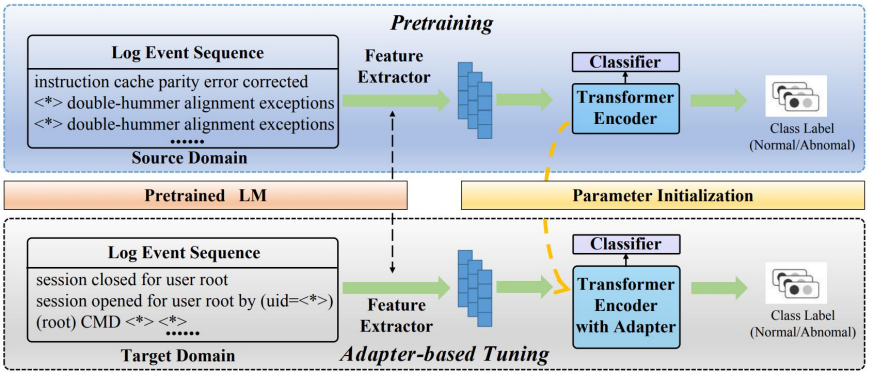


Abbildung 3.9: Übersicht von TransLog [36]

der Zieldomäne angepasst.

Log-Sequenzen werden mittels eines **Merkmalsextraktors** in Vektoren umgewandelt, welche eine einheitliche Dimension d aufweist. Hierbei wird das vortrainierte Satz-Bert-Modell [52] zur Repräsentation der Template-Sequenzen eingesetzt. Jüngere Verfahren extrahieren semantische Repräsentationen aus den neuen und rohen Protokollnachrichten, um Informationsverluste durch Parsing-Probleme in den Protokollen zu vermeiden. Aufgrund der großen Menge an Protokolldaten ist es jedoch nicht praktikabel, jede Protokollnachricht einzubetten. Experimente haben gezeigt, dass selbst bei Parsing-Fehlern nahezu alle Anomalien anhand einer Template-Sequenz identifiziert werden können. Aus diesem Grund werden nur die aktuellen Protokollvorlagen eingebettet. Da jede Sitzung eine konstante Länge l aufweist, kann $XeR^{l\times d}$ für jede Sitzung durch die Schicht bestimmt werden.

Kodierer mit Light Adapter (Abbildung 3.10) werden von den Encoder im Transformer als Basismodell verwendet, um die Eigenschaften der Eingaben effektiver zu kodieren. Der Selbstaufmerksamkeitsmechanismus des Encoders ermöglicht dies und überwindet die Nachteile von RNN-basierten Modellen. Die Formel 3.5 zeigt den grundlegenden Selbstaufmerksamkeitsmechanismus, um die Berechnung von Abfragen, Schlüsseln und Werten zu beschreiben.

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d/heads}})V \tag{3.5}$$

Eine konstante Positionseinbettung wird verwendet, um Informationen über die Reihenfolge einer Log-Sequenz zu erhalten, da eine falsche Ausführungsreihenfolge auch als anomal angesehen wird. Der Adapter wird nach den Schichten der Selbstaufmerksamkeit und des Feedforward eingeführt und verwendet zwei Projektionsschichten, um den verborgenen Vektor von der Dimension d auf die Dimension m und dann

wieder zurück auf die Dimension d abzubilden. Eine integrierte Skip-Connection-Funktion ist ebenfalls im Adapter enthalten. Der Ausgangsvektor des Adapters wird durch eine Matrixmultiplikation und eine $Tanh$-Funktion berechnet.

Adapter werden verwendet, um ein vortrainiertes Sprachmodell an eine neue Domäne anzupassen, indem nur eine kleine Anzahl von Merkmalen der Zieldomäne der Adapter geändert wird. Der Adapter kann nach oben und unten skaliert werden.

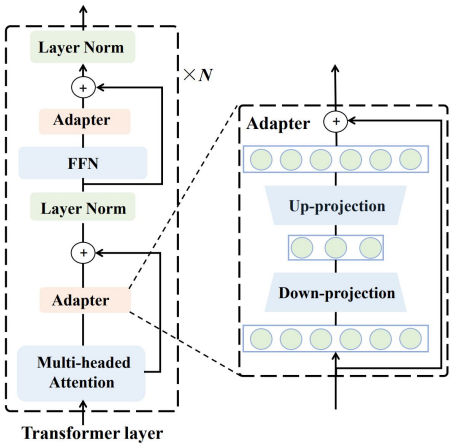


Abbildung 3.10: Adapter von TransLog [36]

Pretraining

Hier hat man sich vom BERT-Modell [53] inspirieren lassen, das eine Methode der Selbstüberwachung nutzt, um allgemeine Sprachmerkmale zu erlernen, die dann für verschiedene nachgelagerte Aufgaben genutzt werden können. Mit dem gestapelten Transformer-Encoder ermitteln wir die gemeinsame Ursache für Log-Anomalien. Das vortrainierte Modell lernt in dieser Phase die Gemeinsamkeiten zwischen verschiedenen Anomalien auf semantischer Ebene und leistet damit einen wichtigen Beitrag zur Erkennung von Anomalien in neuen Protokollquellen. Das Ziel dieser Phase ist es, Anomalien zu erkennen, wobei noch keine Adapter im Modell verwendet wird. Die Parameter des Encoders, die während dieser Phase trainiert werden, werden für die nächste Phase geteilt. Nach der Parameterinitialisierung werden diese Parameter während der Adapter-basierten Phase eingefroren.

Adapter-basiertes Tuning

In der adapterbasierten Abstimmungsmethode wird ein vortrainiertes Modell von der Quelldomäne auf die Zieldomäne angepasst, indem leichtgewichtige Adapter wie neuronale Netze oder rekurrente neuronale Netze verwendet werden. In der vorgestellten Literatur besteht der Adapter aus einer Abwärtsprojekti-

onsschicht, einer Aktivierungsschicht und einer Aufwärtsprojektionsschicht. Das vortrainierte Modell wird in der Vortrainingsphase erhalten. Im zweiten Schritt werden die Adapter in die Transformatorschichten des vortrainierten Modells eingefügt und während der Anpassung im Zielbereich werden nur die Parameter der Adapter aktualisiert. Die Parameter der Feedforward-Schicht und der mehrköpfigen Aufmerksamkeitsschicht im vortrainierten Modell werden beide eingefroren. Im Vergleich zur Feinabstimmung bietet TransLog eine Plug-in-Technik zur Wiederverwendung des vortrainierten Modells mit nur wenigen weiteren trainierbaren Variablen.

3.7 Evaluierung

Um abschließend die vorgestellten Ansätze zu der Anomalie-Erkennung zu wiederholen, wird im vorliegenden Kapitel eine Evaluierung auf Basis der angeführten Bewertungsmetriken aus der Literatur [3] gezeigt. Zusätzlich werden die groben Unterschiede anhand der Vor- und Nachteile zwischen den angeführten Ansätzen tabellarisch aufgezeigt.

Bewertungsmetriken

Um einen Ansatz zur Anomalie-Erkennung bewerten zu können, benötigt man geeignete Metriken, um dies zu bewerkstelligen. Dazu werden die folgenden unterschiedlichen Metriken verwendet. Die erste Metrik stellt die Anzahl der anomalen Log-Sequenzen, die das Modell richtig erkannt hat dar, diese wird als TP (True-Positive) bezeichnet. Die Anzahl der typischen Log-Sequenzen, die fälschlicherweise als Anomalien eingestuft werden, sind als FP (False-Positive) bezeichnet. Als FN (False-Negative) wird die Anzahl der anomalen Log-Sequenzen, die das Modell nicht identifizieren konnte, bezeichnet. Um eine Bewertung für einen Deep Learning Ansatz bzw. ein Modell durchzuführen, werden die folgenden bekannten Bewertungsmetriken aus der Literatur [3] aufgelistet.

- **Präzision:** Der Anteil aller vom Modell erkannten anomalen Logs, die richtig $(TP+FP)$ identifiziert wurden. $Prec = \frac{TP}{TP+FP}$
- **Recall:** Der Anteil der Logs über alle echten Anomalien, die genau als Anomalien $TP + FN$ klassifiziert wurden. $Rec = \frac{TP}{TP+FN}$
- **Spezifität:** der Anteil der Log-Sequenzen an allen tatsächlich normalen Sequenzen, die genau als normal klassifiziert wurden. $Spec = \frac{TN}{TN+FP}$
- **F-Measure:** Das harmonische Mittel aus Präzision und Recall. $F1 = \frac{2*Prec*Rec}{Prec+Rec}$

Es ist wichtig zu beachten, dass die Wahl der Methode zur Bewertung der Leistung des Modells von den

Anforderungen des Anwendungsfalls und den verfügbaren Daten abhängt.

In der Tabelle 3.2 werden die Bewertungsmetriken anhand der sechs genannten Ansätze dargestellt. Hierzu wurden zwei große Datensätze verwendet, und zwar der 'BlueGene/L' Datensatz (BGL) [54] und der 'Hadoop Filesystem' Datensatz (HDFS) [55]. Beide Datensätze wurden bereits im Abschnitt 2 als Teil der Datensammlung von Loghub[16] vorgestellt und sind somit jedermann zugänglich.
Die Autoren der Literatur [3] haben untersucht, wie gut die unterschiedlichen Modelle bei der Gruppierung von Sitzungsfenstern abschneiden. Die Ausführungsroute eines Auftrags kann durch Gruppen von Protokollen in Abhängigkeit von den Identifikatoren dargestellt werden. Nach jeder Sitzung wurde eine Anomalie-Erkennung durchgeführt. Für den BGL-Datensatz wurde deutlich, dass Sitzungsfenster bessere Ergebnisse liefern als feste Fenster. Im HDFS-Datensatz schneiden alle Modelle ebenfalls gut ab (alle F-Maß-Werte sind höher als 0,9).

Model	BGL				HDFS			
	Prec	Rec	Spec	F1	Prec	Rec	Spec	F1
DeepLog	0.997	1.0	0.997	0.998	0.835	0.994	0.994	0.908
LogAnomaly	0.997	1.0	0.998	0.999	0.886	0.893	0.961	0.966
PLELog	0.995	0.992	0.996	0.994	0.893	0.979	0.996	0.934
LogRobust	1.0	1.0	1.0	1.0	0.961	1.0	0.989	0.980
CNN	1.0	1.0	1.0	1.0	0.966	1.0	0.991	0.982

Tabelle 3.2: Ergebnis der Bewertungsmetriken der Ansätze [3]

Um die Evaluierung der Ansätze zu vervollständigen, ist das Ergebnis von TransLog [36] wie folgt. Für den HDFS-Datensatz sind die Werte; Prec: 0.99, Rec:0.99 und F1:0.99 und für den BGL-Datensatz; Prec: 0.98, Rec:0.98 und F1:0.98.
Das gezeigte Ergebnis 3.2 ist nur ein Ausschnitt aus der Literatur [3] und soll zusammenfassend zeigen, dass die vorgestellten Ansätze aus dieser Arbeit ein guter Weg für die Log-Anomalie-Erkennung sind.

In der Tabelle 3.3 werden die Vor- und Nachteile der Ansätze kurz zusammengefasst, um einen Überblick zu erhalten, damit eine geeignete Wahl für den entsprechenden Anwendungsfall getroffen werden kann.

Bei DeepLog ist es ausreichend, wenn man einfache und normale Daten für die Modellerstellung verwendet, da es keine anomalen Logs im Vorfeld benötigt. Zudem kann es mögliche Anomalien frühzeitiger erkennen

als andere Modelle. Eine schwächere Leistung wird bei komplexeren Datensätzen mit einer großen Anzahl von Ereignissen beobachtet. Außerdem ist DeepLog anfällig für Fehler bei der Log-Analyse, da es nur den Index der Log-Templates verwendet und deren Bedeutung ignoriert. Ein weiter Nachteil ist es, dass DeepLog einen beschrifteten Datensatz benötigt, damit das Modell trainiert werden kann.

Durch die Verwendung von sequentiellen und quantitativen Vektoren kann LogAnomaly das Datenrauschen, durch fehlerhaft beschriftete Logs, reduzieren. Ähnlich wie DeepLog erkennt LogAnomaly Anomalien frühzeitig und kann große Datenmengen verarbeiten, da es ebenfalls mit normalen beschrifteten Logs trainiert wird. Die Besonderheit bei LogAnomaly liegt in der Fähigkeit, ähnliche Protokollmuster bzw. Templates über semantische Vektoren zu vergleichen. Das führt zu einer Verbesserung in der Genauigkeit. Wie bei DeepLog trainiert auch LogAnomaly das Modell nur mit einem beschriften Datensatz. Zudem wird auf Basis des Index der Ereignisse trainiert, was es anfällig für Fehler bei der Protokollanalyse macht. Bei großen, komplexen Datensätzen kann es zu einem nicht optimalen Ergebnis kommen.

Bei PLELog wird eine Cluster-Methode verwendet, um probabilistische Schätzungen von unbeschrifteten Protokollsequenzen abzugeben. Das hat den Vorteil, dass PLELog, auch mit nur teilweise mit beschrifteten Logs arbeiten kann. Die Verwendung von semantischen Vektoren und eines aufmerksamkeitsbasierten GRU-Netzes erhöht die Effektivität von PLELog. Durch das Clustering wird ein längerer Prozess bei dem Trainieren des Modells benötigt. Außerdem kann es nicht gut mit Störsignalen in den Trainingsdaten umgehen und ist für die Früherkennung weniger geeignet.

LogRobust verwendet semantische Vektoren von Log-Mustern in Verbindung mit einem aufmerksamkeitsbasierten Bi-LSTM-Modell, wodurch es in der Lage ist, Kontextinformationen aus Protokollsequenzen zu erfassen und Datenrauschen zu verringern. Als überwachtes Modell benötigt LogRobust jedoch sowohl normale als auch abnormale Daten während der Trainingsphase, was einen erheblichen Aufwand bedeutet.

Die Faltungsoperationen im CNN-Modell ermöglicht es, nicht nur die Korrelation zwischen Protokollvorlagen zu erfassen, sondern auch die Korrelation innerhalb der semantischen Einbettung dieser Vorlagen. Mit einem überwachten Ansatz kann CNN eine hohe Genauigkeit auf vielen Datensätzen erreichen. Allerdings benötigt CNN auch eine beträchtliche Menge an beschrifteten Daten. Es kann auch zu einem Verlust an Genauigkeit bei der Verarbeitung von stark unausgeglichenen Daten und Datenrauschen kommen.

Model	Vorteil	Nachteil	Benötigte Daten
DeepLog	Relativ einfach, erfordert nur normale Daten. Gut geeignet, Anomalien frühzeitig zu erkennen.	Schlechte Leistung bei komplexen Datensätzen. Starke durch die Fehler beim Parsen der Protokolle beeinträchtigt.	beschriftete Daten
LogAnomaly	Erfordert nur normale Daten. Kann neue Protokoll-Vorlagen mit vorhandenen Vorlagen in den Trainingsdaten abgleichen.	Schlechte Leistung bei komplexen Datensätzen. Starke durch die Fehler beim Parsen der Protokolle beeinträchtigt.	beschriftete Daten
PLELog	Benötigt nur teilweise beschriftete Daten. Gute Leistung im Vergleich zu halbüberwachten Methoden. Kann die Bedeutung von Protokollvorlagen lernen	Kann nicht gut mit dem Rauschen in den Trainingsdaten arbeiten und in der Aufgabe der Früherkennung. Komplexer Entwurf und zeitaufwändig	teilweise beschriftete Daten
LogRobust	Gute Leistung bei vielen Datensätzen. Kann die Kontextinformationen von Protokollsequenzen erfassen, um Log-Parsing-Fehler und die Instabilität von Logs zu behandeln	Erfordern eine große Menge an markierten Daten. Erheblich durch falsch beschriftete Protokolle beeinträchtigt.	beschriftete Daten
CNN	Gute Leistung bei vielen Datensätzen. Kann mit Log Parsing-Fehlern umgehen.	Erfordern eine große Menge an markierten Daten. Erheblich durch falsch beschriftete Protokolle beeinträchtigt.	beschriftete Daten

Tabelle 3.3: Vor- und Nachteile von den Ansätzen [3]

4 Herangehensweise

Der angeführte Abschnitt bildet den zweiten Teil der Diplomarbeit und zeigt eine Herangehensweise in Bezug auf Log-Anomalie-Erkennung. Der hier erstellte Prototyp hat den Namen 'LogDeeptector' und wurde basierend auf der JavaScript-API TensorFlow.js [56] von TensorFlow [57] entwickelt. In den folgenden Kapiteln wird die Funktionsweise und die Architektur des genannten Prototyps definiert, genauso werden die nötigen Voraussetzungen genannt. Abschließend wird der Prototyp anhand unterschiedlicher Modelle trainiert und das erzeugte Ergebnis evaluiert.

4.1 LogDeeptector

Im ersten Teil der Arbeit wurden die Theorie über Log-Anomalie-Erkennung abgedeckt. Mit diesem Wissen wurde im Laufe dieser Diplomarbeit eine eigene Applikation implementiert, um Log Anomalien zu erkennen. Hierfür wurde Angular [58] als Web Entwicklungsframework verwendet. Die Anwendung nutzt die API TensorFlow.js, um das nötige Deep Learning Model zu entwickeln.

In der Abbildung 4.1 wird ein grober Überblick der Funktionsweise des Prototyps gezeigt. Grundsätzlich besteht der Prototyp aus drei Komponenten: (1) einem Parser, (2) einem Encoder und (3) einem Modell. Der Parser leitet die gefilterten und extrahierten Informationen aus den rohen Logs an den Encoder weiter. Es liegt in der Verantwortung von dem Encoder, die erforderlichen Daten in eine numerische Darstellung wie einen Tensor zu kodieren. Vor dem Training des Modells oder der Abfrage eines Ergebnisses mit einem trainierten Modell müssen beide Schritte durchgeführt werden. Das Ergebnis einer Vorhersage fließt direkt in das bestehende Modell ein, um eine Feedback-Schleife zu erzeugen.
Das entwickelte Modell von LogDeeptector benötigt einen Trainingsdatensatz, um anschließend eine Vorhersage zu treffen. Doch bevor die rohen und unstrukturierten Log-Daten verwendet werden können, müssen diese in eine strukturierte Form gebracht werden. Hierfür werden zwei Schritte benötigt.
Im ersten Schritt werden die Logs in eine definierte Struktur geparst. Ein sogenannter regulärer Ausdruck (Regex) wird angewendet, um die Informationen aus dem Log, das in der Regel aus einer Textzeile besteht,

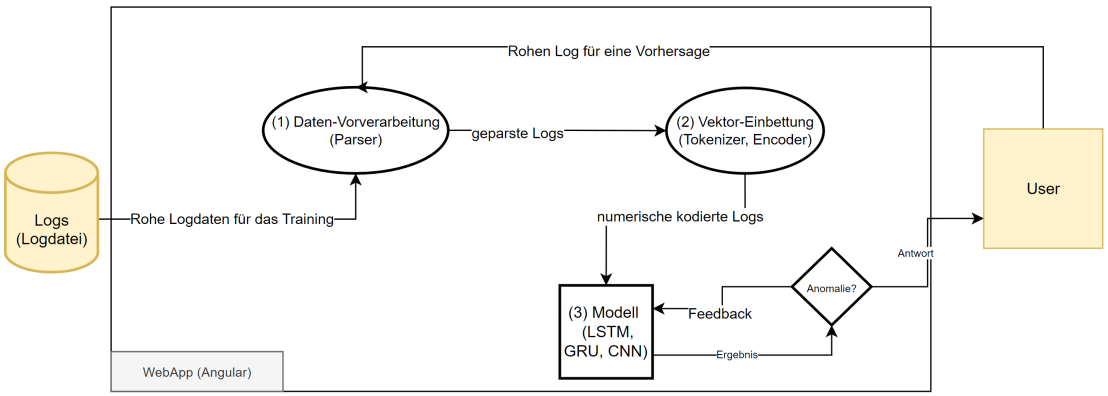


Abbildung 4.1: Überblick von LogDeeptector

zu extrahieren und zu filtern. Der Prototyp verwendet einen Regex der für Apache2 Access Logs definiert wurde. Je nach System und Art der Protokolle muss der reguläre Ausdruck angepasst werden. Es ist ebenfalls möglich bekannte Log-Parser wie Drain und Spell, die in den Kapitel 2 definiert wurden, zu nutzen. Zusätzlich werden unnötige Satzzeichen und Stoppwörter entfernt, damit nur die nötigen Daten erhalten bleiben, die später für das Modell-Training verwendet werden sollen.

In einem weiteren Schritt müssen die strukturierten Logs zu einem sogenannten Tensor bzw. zu einem NumPy-Array konvertiert werden. Das ermöglicht es, dass das neuronale Modell die Informationen als Eingabe verwenden kann. Bei der Einbettung von den formatierten Logs zu einem Vektor wird jedes Wort und jede Zahl, wie zum Beispiel die IP-Adresse oder der Port, zu einem nummerischen Wert umgewandelt. Durch die Verwendung von einem definierten Vokabular, wo jedes Wort bzw. Zahl einen nummerischen Wert zugewiesen ist, wird dies ermöglicht. Auch in diesem Schritt können bereits bestehende Vokabularien und Encoder verwendet werden.

Aufgrund dass ein unbeschrifteter Datensatz verwendet wird, muss dieser, bevor er im Modell trainiert werden kann, noch durch Kategorisierung beschriftet werden. Dazu wird eine kmeans-Clustermethode verwendet. Hierbei werden anhand der verwendeten Logs, k-zufällige Zentren generiert, die je nach Anzahl der angegebenen Iterationen sich neu zentrieren. Je nach Abstand der Datenpunkte zu den Zentren erhält man eine Anzahl von k Cluster. Der Cluster, der am wenigsten Datensätzen enthält, wird als Anomalie-Cluster und der Rest als normal kategorisiert. Somit erhaltet man einen beschrifteten Datensatz für das anschließende Modell-Training.

Das trainierte Modell kann anschließend verwendet werden, um eine Vorhersage zu treffen. Doch bevor die Logs, die bestimmt werden sollen, nach ’anomal’ und ’normal’ kategorisiert werden können, müssen

diese ebenfalls die bereits genannten Schritte, wie das Parsing und Embedding, durchlaufen. Nicht nur die Benutzerin und der Benutzer erhalten das Ergebnis der Vorhersage, sondern auch das Modell. Somit wird das neuronale Netzwerk anhand des Resultates direkt trainiert und es entsteht eine sogenannte Feedback-Schleife.

4.1.1 Prerequisites

Bevor mit der Entwicklung von dem Prototypen 'LogDeeptector' gestartet werden konnte, mussten ein paar Voraussetzungen erfüllt werden.
Um die Angular CLI [59] auszuführen, wird WSL2 [60] mit einer Ubuntu Distribution installiert. Dabei ist zu beachten, dass die PowerShell als Administrator ausgeführt werden muss. Anschließend wird mit folgendem Befehl *'wsl –install'* WSL installiert. Anschließend muss noch Ubuntu für die gewünschte Distribution *'wsl –install -d Ubuntu'* angeben werden.
Bevor die Angular Applikation erstellt werden kann, ist es notwendig, dass die Linux Distribution aktualisiert wird, indem in der Konsole die beiden Befehle *'sudo apt-get update'*, *'sudo apt-get upgrade'* ausführt werden. Danach wird der Node Package Manager (NPM) [61] installiert *'sudo apt install npm'*, um anschließend über den npm Befehl die Angular CLI zu installieren *'sudo npm install -g @angular/cli'*.
Der Prototyp ist auf der Node.js Version v18.15.0 entwickelt worden. Damit die Node.js Version leichter verwaltet werden kann, wurde der Node Version Manager (NVM) [62] installiert.
Die folgenden Module sind über den Node Package Manager zu installieren, um eine reibungslose Implementation zu gewährleisten:

- **TensorFlow.js (@tensorflow/tfjs)**: Erstellung von Tensoren und Deep Learning Modellen
- **TensorFlowVisual (@tensorflow/tfjs-vis)**: Visuelle Darstellung des Trainingsfortschritts
- **Stopword (stopword)**: Sammlung von Stoppwörtern

Sobald die jeweiligen Pakete und nötigen Versionen installiert bzw. aktualisiert wurden, wird eine Angular Applikation über die Angular CLI erstellt. Die Entscheidung Angular zu verwenden wurde getroffen, da es sich um ein beliebtes Open-Source-JavaScript-Framework handelt, um Webanwendungen zu entwickeln. Im Kern basiert Angular auf der Model-View-Controller (MVC)-Architektur, bei der die Anwendungslogik in drei Komponenten aufgeteilt wird: das Model, den View und den Controller. Das Model ist für die Verarbeitung der Daten zuständig, die View für die Darstellung der Benutzeroberfläche und der Controller für die Kommunikation zwischen beiden. Diese Trennung ermöglicht es den Entwicklern, sauberen und wartbaren Code zu schreiben, der leicht zu verstehen und zu ändern ist.[58]

4.1.2 Aufbau

Der Prototyp besteht aus verschiedenen Komponenten und Services, die die Funktionalität der Anwendung gewährleisten. Dieser Abschnitt befasst sich mit der Struktur und den einzelnen Funktionen des Prototyps. Insgesamt sind vier Dateien von Bedeutung: app.component.ts, model.service.ts, embedding.service.ts und parser.service.ts.

app.component.ts

Die Datei „app.component.ts“ ist vergleichbar mit einer Main-Funktion in anderen Programmiersprachen wie Java oder Python. Diese Datei besteht im Grunde aus drei Funktionen, und ist für die Steuerung und die damit verbundenen Methodenaufrufen verantwortlich.
Die erste Methode, die implementiert wurde, heißt *'onFileSelected'*, und dient dazu eine Datei über das Datei-Upload-Element auszuwählen. Die ausgewählte Datei wird in einer Variablen gespeichert, welche anschließend von einem FileReader ausgelesen wird. Der Inhalt wird in einer weiteren Variablen vom ParserService (parser.service.ts) gespeichert.

```
onFileSelected(event: any) {
    try {
      this.selectedFile = event.target.files[0];
      const fileReader = new FileReader();

      fileReader.onload = (e: any) => {
        this.parserService.fileContent = e.target.result;
      };
      fileReader.readAsText(this.selectedFile);
    }
    catch (error) {
      console.log('Error onFileSelected:', error)
    }
  }
```

Die restlichen beiden Methoden werden aufgerufen, um entweder das Model zu trainieren (*'trainModel'*) oder eine Vorhersage zu treffen (*'makePrediction'*). Dabei wird überprüft, ob eine Datei ausgewählt wurde, andernfalls wird eine Fehlermeldung ausgegeben.

```
trainModel(){
    if (this.selectedFile) {
      this.modelService.trainModel()
```

```
    } else {
      console.log('Es wurde keine Datei ausgewaehlt.');
    }
}


makePrediction(){
    if (this.selectedFile) {
      this.modelService.predict()
    } else {
      console.log('Es wurde keine Datei ausgewaehlt.');
    }
}
```

parser.service.ts

Dieser Service ist verantwortlich, um die Protokolle aus der ausgewählte Datei in eine strukturierte Form zu parsen und um die Information aus jedem einzelnen Log zu filtern. Dafür wurden zwei Methoden definiert. Die Funktion *'loadLog'* liest die einzelnen Log-Daten aus der ausgewählten Datei aus, indem er sich zeilenweise (durch regulären Ausdruck ‚/[\r \n]+/‘) die Logs holt. Anschließend wird bei jedem einzelnen Log das Vokabular (*'updateVocabulary'*) aus dem EmbeddingService aktualisiert, bevor es mittels der Methode *'parseLog'* geparst wird. Nachdem ein Log geparst und die Informationen extrahiert wurden, wird die Funktion *'embedWithVoc'* aus dem EmbeddingService aufgerufen, um eine eingebettete Darstellung der Logs zu vollbringen. Sowohl die geparsten als auch die eingebettete Darstellung wird in das Ergebnisobjekt als Rückgabewert zurückgegeben.

```
loadLog(): Promise<{ embeddedLogs: Array<Array<number>>, logs: Array<string> }> {
    return new Promise<{ embeddedLogs: Array<Array<number>>, logs: Array<string> }>((
        resolve, reject) => {
      let embeddedLogs: Array<Array<number>> = []
      let logs: Array<string> = []

      this.fileContent.split(/[\r\n]+/).forEach(log => {
        if (log) {
          this.embeddingService.updateVocabulary(this.parseLog(log))
          logs.push(this.parseLog(log))
          embeddedLogs.push(this.embeddingService.embedWithVoc(this.parseLog(log)))
        }
        else {
          reject("Kein Log im File enthalten")
```

```
        }
      });
      const result = { embeddedLogs: embeddedLogs, logs: logs };
      resolve(result)
    })
  }
```

Die *'parseLog'* Funktion analysiert und bereinigt einen einzelnen Log-Eintrag anhand eines vordefinierten regulären Ausdrucks. Der verwendete reguläre Ausdruck ist implementiert worden, um Access-Logs von Apache2 Systemen zu behandeln. Dabei werden relevante Informationen wie IP-Adresse, Benutzername, Zeitstempel, HTTP-Anfrage usw. extrahiert und zu einem neuen String zusammengeführt. Anschließend werden sogenannte Stoppwörter aus dem Text entfernt, damit ein bereinigter Log-Eintrag zurückgegeben werden kann.

```
parseLog(log: string): string {
    var matchedLog: any = log.match(this.APACHE2REGEX)
    this.parsedLog = matchedLog[1]
    if (matchedLog[2] != "-") {
      this.parsedLog = this.parsedLog + " " + matchedLog[2]
    }
    this.parsedLog = this.parsedLog + " " + matchedLog[4] + " " + matched-Log[5] + " "
        + matchedLog[6] + " " + matchedLog[7]
    if (matchedLog[9] != "-") {
      this.parsedLog = this.parsedLog + " " + matchedLog[9]
    }

    let logArray = this.parsedLog.toLowerCase().split(' ')
    this.parsedLog = sw.removeStopwords(logArray, [...sw.eng, ...sw.deu]).toString().
       replace(/\,/g, " ");
    return this.parsedLog
}
```

embedding.service.ts

Diese Datei definiert den bereits erwähnten EmbeddingService. Dieser Service ist veranwortlich, dass die geparsten Log-Einträge in der Methode *'loadLoag'* aus dem ParserService, in eine eingebettete numerische Darstellung definiert werden, um die Logs als Eingabe in ein TensorFlow-Modell zu verwenden.

Die Methode *'updateVocabulary'* wird verwendet, um eine das Vokabular anhand der Log-Sequenzen als Eingabe zu aktualisieren. Dabei wird jedes Wort in einem geparsten Log, in das Array *'vocabularyDic'* hinzufügt, sofern es nicht bereits im Vokabular vorhanden ist. Hierbei bekommt jedes Wort oder jede Zahl

aus dem Log eine bestimmte Nummer zugeordnet, was im Grunde die aktuelle Länge des Arrays ist, wann das Wort hinzugefügt wurde.

```
updateVocabulary(text: string): void {
    var textArray = text.split(' ')
    for (let i = 0; i < textArray.length; i++) {
      if (textArray[i] in this.vocabularyDic) {
      }
      else {
        this.vocabularyDic[textArray[i]] = Object.keys(this.vocabularyDic).length
      }
    }
}
```

Anschließend wird das aktualisierte Vokabular verwendet, um eine Log-Sequenz in eine eingebettete numerische Darstellung darzustellen. Verantwortlich ist die Methode *'embedWithVoc'*, die das Vokabular mit der Sequenz abgleicht und das entsprechende Wort mit der dazugehörigen Zahl ersetzt.

```
embedWithVoc(text: string): Array<number> {
    var embedText: number[] = []
    var textArray = text.split(' ')
    for (let i = 0; i < textArray.length; i++) {
      if (textArray[i] in this.vocabularyDic) {
        embedText.push(this.vocabularyDic[textArray[i]])
      }
    }
    return embedText
}
```

Für die Eingabe in das TensorFlow-Modell benötigen jede Log-Sequenz die gleiche Array-Länge, um die korrekte Dimension für das Tensor zu erreichen. Die Methode *'setLongestSubArray'* wird hierfür verwendet, um das längste Array bzw. Log-Sequenz von allen bereinigten und eingebetteten Logs zu identifizieren. Diese Information wird anschließend in der Variablen *'longestSubArray'* abgespeichert. Mit der Information des längsten Arrays wird in der Methode *'padSequences'* ein sogenanntes Padding implementiert. Dabei wird jeder Log auf die gleiche Länge angepasst, indem man einfach Nullen an jeder Sequenz anhängt.

```
setLongestSubArray(sequence: Array<Array<number>>): void {
    this.longestSubArray = sequence.reduce((acc, curr) => {
      return curr.length > acc.length ? curr : acc;
    }, []);
}
```

```
padSequences(sequence: Array<Array<number>>): Array<Array<number>> {
    const addPadding = (subArray: Array<number>, length: number): Array<number> => {
      if (subArray.length >= length) {
        return subArray.slice(0, length);
      } else {
        const padding = new Array(length - subArray.length).fill(0);
        return subArray.concat(padding);
      }
    };
    const paddedArray = sequence.map((subArray) => addPadding(subArray, this.
        longestSubArray.length));

    return paddedArray
}
```

model.service.ts

Das Model-Service bildet den Kern des Prototyps und ist im Grunde für die Hauptfunktionalitäten, wie das Training des Models, das Clustering und das Vorhersagen des Ergebnisses, zuständig.
Wie der Name der Methode *'trainModel'* schon verrät, ist diese verantwortlich für das Trainieren des neuronalen Netzwerkmodells. Dabei wird zuerst die ausgewählte Log-Datei mithilfe der bereits genannten Funktion *'loadLog'* vom ParserService in die gewünschte strukturierte Form extrahiert und eingebettet. Anschließend wird die Clustermethode *'kmeansClustering'* aufgrufen, um den unbeschrifteten Datensatz mit Kategorien (Anomalie/Nicht-Anomalie) zu versehen. Anschließend wird das Model mit den benötigten Schichten definiert und für das angehende Training kompiliert. Das im Code gezeigte Model ist ein LSTM-Ansatz. In der späteren Evaluierung wird dieses Model durch andere ersetzt, um einen Vergleich zu erstellen. Bei dem Training wird der Datensatz prozentuell, auf ein Training - und ein Testdatensatz, geteilt. Während dem Training wird eine visuelle Darstellung mithilfe des @tensorflow/tfjs-vis Moduls ausgeben, um den Fortschritt zu beobachten. Ist das Model fertig trainiert, wird dieses abgespeichert.

```
async trainModel(): Promise<any> {
    return new Promise<void>((resolve, reject) => {
      this.parserService.loadLog()
        .then((result) => {
          ...
          this.kmeansClustering().
            then(async () => {
              ...
```

```
this.model = tf.sequential(
  {
    layers: [
      tf.layers.embedding({ inputDim: 2, outputDim: 16, in-putLength:
          trainDataTensor.shape[1] }),
      tf.layers.lstm({ units: 32, returnSequences: false }),
      tf.layers.dense({ activation: "relu", units: 8 }),
      tf.layers.dense({ activation: "softmax", units: 2 })
    ]
  }
)

this.model.compile({
  loss: "categoricalCrossentropy",
  optimizer: "adam",
  metrics: ["accuracy"]
})

try {
  const loadedModel = await tf.loadLayersModel('localstorage://assets/
      model');
} catch (error) {
  console.log('assets/model/model.json existiert nicht');
}

this.model.fit(trainDataTensor, reshapedLabelSet, {
  batchSize: 32,
  epochs: 20,
  shuffle: true,
  validationSplit: 0.1,
  callbacks: [
    tfvis.show.fitCallbacks(
      document.getElementById("trainGraph") as HTMLElement,
      ["loss", "acc"],
      {
        callbacks: ["onEpochEnd"]
      }
    ), {
```

```
                    onEpochEnd: (epoch: any, logs: any) => {
                      console.log(`Epoch ${epoch + 1}, val_loss: ${logs.val_loss},
                          val_acc: ${logs.val_acc}, loss: ${logs.loss}, acc: ${logs.acc
                          }`);
                    }
                  }]
              }).then(async () => {
                console.log("Training finished !")
                this.model.summary()
                await this.model.save('localstorage://assets/model')

                resolve()
              });
            })

        })
        .catch((error) => {
          console.error('Fehler beim Laden der Log-Datei:', error);
        });
    })
  }
```

In der Methode *'kmeansClustering'* wurde ein kMeans-Algorithmus implementiert um den unbeschrifteten Datensatz zu beschriften. Dabei wird zuerst anhand der Datensätze k-zufällige Zentren generiert die dann nach jeder Iteration neu zentriert werden. Nachdem die angegebene Anzahl an Iterationsschritte durchlaufen ist, werden die Datensätze anhand der Distanzen zu den Clusterzentren zugeteilt. Der Cluster mit den geringsten Datenpunkten, wird als Anomalie-Cluster deklariert, dabei wir ein Log mit einer *'1'* definiert wenn er eine Anomalie darstellt und mit einer *'0'* wenn nicht.

```
async kmeansClustering(): Promise<void> {
    return new Promise<void>(async (resolve, reject) => {
      const k = this.k;
      const maxIterations = this.maxIterations;

      const dictionayMaxValueLength = Math.max(...Object.values(this.embeddingService.
          vocabularyDic))
      var centroids = tf.randomUniform([k, this.trainingSet[0].length], 0,
          dictionayMaxValueLength);
```

```
const trainingSetTensor = tf.tensor(this.trainingSet);
var assignments = tf.tensor([])

for (let i = 0; i < maxIterations; i++) {
  const distances = tf.sub(trainingSetTensor.expandDims(1), cen-troids);
  const positivesDistances = tf.abs(distances);
  const sumOfDistances = tf.sum(positivesDistances, 2);
  assignments = tf.argMin(sumOfDistances, 1);

  const newCenter = [];
  for (let j = 0; j < k; j++) {
    const mask = assignments.equal(j);
    const anyMask = mask.any().dataSync()[0];

    if (anyMask) {
      const clusterPoints = await tf.booleanMaskAsync(trainingSetTensor, mask);
      const mean = await tf.mean(clusterPoints, 0);
      newCenter.push(mean);
    }
  }
  centroids = tf.stack(newCenter);
}

const assignmentsArray = assignments.arraySync() as Array<number>
const assignmentCounts = new Map<number, number>();
assignmentsArray.forEach(label => {
  if (assignmentCounts.has(label)) {
    assignmentCounts.set(label, assignmentCounts.get(label) as number + 1);
  } else {
    assignmentCounts.set(label, 1);
  }
});

let lowestKey: number | undefined;
let lowestValue: number | undefined;
for (const [key, value] of assignmentCounts.entries()) {
  if (lowestValue === undefined || value < lowestValue) {
    lowestKey = key;
    lowestValue = value;
```

```
        }
      }
      const anomalies = [];
      for (let i = 0; i < assignmentsArray.length; i++) {
        if (assignmentsArray[i] == lowestKey) {
          anomalies.push(this.trainingSet[i]);
        }
      }

      for (let i = 0; i < this.trainingSet.length; i++) {
        if (anomalies.includes(this.trainingSet[i])) {
          this.labeledtrainSet.push(1);
        } else {
          this.labeledtrainSet.push(0);
        }
      }
      resolve()
    })
}
```

Um anschließend eine Vorhersage mit dem trainierten Modell treffen zu können, wurde die Methode *'predict'* implementiert. Wie in der Methode *'trainModel'* muss zuerst die Eingabe, worüber ein Vorhersage getroffen werden soll, ebenfalls geladen und verarbeitet werden. Hierzu wird die selbe Methode *'loadLog'* aus dem ParserService verwendet. Anschließend wird das gespeicherte Modell geladen und ein Vorhersagen mit der Standardfunktion *'predict'* von TensorFlow getroffen. Die erkannten Anomalien werden in einem neuen Trainingssatz aufgezeichnet und das Modell wird anschließend mit den Ergebnis erneut trainiert. Somit erhält man eine Art von Feedback-Schleife.

```
predict(): void {
    this.parserService.loadLog()
      .then(async (result) => {
        const embeddedLogs = result.embeddedLogs;
        const logs = result.logs;

        this.embeddingService.setLongestSubArray(embeddedLogs)
        const predictionSet = tf.tensor(this.embeddingService.padSequences(
            embeddedLogs))

        const loadedModel = await tf.loadLayersModel('localstorage://assets/model');
```

```
const predictions = loadedModel.predict(predictionSet) as tf.Tensor;
const arrayPrediction: Array<Array<number>> = predic-tions.arraySync() as
    Array<Array<number>>;

let updatedTrainData: number[][] = [];
let updatedLabels: number[] = [];
let predictionResult: string[] = [];

arrayPrediction.forEach((prediction, index) => {
  if (prediction[0] < prediction[1]) {
    console.log('Anomalie erkannt (' + logs[index] + ')')
    predictionResult.push('Anomalie erkannt (' + logs[index] + ')')

    updatedTrainData.push(embeddedLogs[index]);
    updatedLabels.push(1);

  }
})
let updatedTrainDataTensor = tf.tensor(this.embeddingService.padSequences(
    updatedTrainData))
let updatedLabelsTensor = tf.oneHot(updatedLabels, 2);

var resultPredictionElement = docu-ment.getElementById("resultPrediction") as
    HTMLElement
const listElement = document.createElement('ul');
predictionResult.forEach(item => {
  const listItemElement = document.createElement('li');
  listItemElement.textContent = item;
  listElement.appendChild(listItemElement);
});

if ((updatedTrainDataTensor.dataSync().length !== 0) && (updated-LabelsTensor.
    dataSync().length !== 0)) {
  resultPredictionElement.appendChild(listElement);

  loadedModel.compile({
    loss: "categoricalCrossentropy",
    optimizer: "adam",
    metrics: ["accuracy"]
```

```
        })

        loadedModel.fit(updatedTrainDataTensor, updatedLabelsTensor, {
          batchSize: 32,
          epochs: 20,
          shuffle: true,
          validationSplit: 0.2,
          callbacks: [
            tfvis.show.fitCallbacks(
              document.getElementById("trainGraph") as HTMLElement,
              ["loss", "acc"],
              {
                callbacks: ["onEpochEnd"]
              }
            ), {
              onEpochEnd: (epoch: number) => {
                console.log(`Epoch ${epoch + 1}`);
              }
            }]
        }).then(async () => {
          console.log("Update training finished !")
          loadedModel.summary()
          await loadedModel.save('localstorage://assets/model')
        });
      }
      else {
        resultPredictionElement.textContent = " Keine Anomalien gefunden"
      }
    })
    .catch((error) => {
      console.error('Fehler beim Laden der Log-Datei:', error);
    });
}
```

4.1.3 Evaluierung

In diesem Abschnitt werden die Ergebnisse von dem Prototyp gezeigt. Als Evaluierung werden die Werte von der Genauigkeit (acc), der Validierungsgenauigkeit (val_acc) sowie der Verlust (loss) und der Validierungsverlust (val_loss) ausgegeben.

Normalerweise werden die Bewertungsmetriken, die im Kapitel 3 erklärt wurden, verwendet, um Rückschlüsse auf die Performance von einem Modell zu geben. Da es sich aber um einen unbeschrifteten Datensatz handelt, und die Logs nicht nach Anomalie oder Nicht-Anomalie beschriftet sind, kann man daher keine fixen False- bzw. True-Positives ausfindig machen. Würde man einen beschrifteten Datensatz verwenden und es ist zu erkennen, welche Logs richtig vorhergesagt wurden, ist es problemlos, die anschließenden Berechnungen wie im Kapitel 3 durchführen.
Als Eingabedaten für das Training und der anschließenden Vorhersage wurde ein Datensatz von Kaggle [63] verwendet, dieser bildet Access-Logs aus dem Apache2 System ab. Als Limitierung musste die Anzahl an Logs stark reduziert werden, damit die Applikation diese laden konnte. Somit sind es insgesamt um die dreißigtausend Logs, mit dem jedes Modell trainiert wurde und mehr als fünftausend Logs, mit den eine Vorhersage getroffen wurde.
Für die Evaluierung werden drei unterschiedlichen Modelle (LSTM, GRU und CNN) verwendet, welche auch schon in dem Abschnitt 2 erklärt wurden. Dabei wird jedes Model mit denselben Datensätzen gefüttert, um einen Vergleich zu haben.

Zuerst wird das LTSM-Modell verwendet, dabei wird als erste Schicht eine Embedding-Schicht mit einer Eingabedimension von zwei und einer Ausgabedimension von 16 verwendet. Die Eingabelänge wird dynamisch anhand der Form des Tensor *'trainDataTensor'* bestimmt. Die zweite Schicht ist eine LSTM-Schicht (Long Short-Term Memory), die für das Verständnis von zeitlichen Abhängigkeiten in den Daten zuständig ist. Dabei werden 32 interne neuronale Einheiten oder 'Zellen' verwendet. Der Parameter *'returnSequences: false'* gibt an, dass die LSTM-Schicht nur den Ausgabevektor für den letzten Zeitschritt zurückgibt, anstatt eine Sequenz von Ausgabevektoren. Die beiden letzten Schichten sind sogenannte Dense-Schichten, das sind vollständig verbundene Schichten. Die erste Dense-Schicht besteht aus acht Neuronen, dabei werden ReLU-Aktivierungsfunktionen angewendet, um kein lineares Verhalten einzuführen. Die letzte Schicht hat zwei Neuronen, was bedeutet, dass sie eine Ausgabe in Form eines 2-dimensionalen Vektors liefert. Die Softmax-Aktivierungsfunktion normalisiert die Ausgaben, sodass sie als Wahrscheinlichkeiten für die beiden möglichen Klassen (Anomalie oder Normal) interpretiert werden können.

```
this.model = tf.sequential(
                {
                  layers: [
                    tf.layers.embedding({ inputDim: 2, outputDim: 16, in-putLength:
                        trainDataTensor.shape[1] }),
                    tf.layers.lstm({ units: 32, returnSequences: false }),
```

```
                tf.layers.dense({ activation: "relu", units: 8 }),
                tf.layers.dense({ activation: "softmax", units: 2 })
              ]
            }
          )
```

Das nächste Modell für die Evaluierung ist ein GRU-Modell (Gated Recurrent Unit). Das implementierte Modell hat dabei ebenfalls vier Schichten und ist dem LTSM-Modell sehr ähnlich. Hier wird einfach anstatt der LTSM-Layer als zweite Schicht einen GRU-Layer mit den gleichen Parametern verwendet.

```
const gruModel = tf.sequential(
            {
              layers: [
                tf.layers.embedding({ inputDim: 2, outputDim: 16, in-putLength:
                    trainDataTensor.shape[1] }),
                tf.layers.gru({ units: 32, returnSequences: false }),
                tf.layers.dense({ activation: "relu", units: 8 }),
                tf.layers.dense({ activation: "softmax", units: 2 })
              ]
            }
          );
this.model = gruModel
```

Ein weiteres Modell ist das CNN-Modell, welches insgesamt aus sechs Schichten besteht. Dabei bleibt die erste Schicht, die Embedding-Schicht, unverändert, da sie die Eingabedaten in einen dichten Vektorraum projiziert, bevor sie an die zweite Schicht, eine 1D-Convolutional-Schicht weitergeleitet wird. Eine 1D-Convolutional-Schicht ist ideal, wenn die Eingabedaten sequenziell strukturiert sind, wie es bei der Verarbeitung von Logs oft der Fall ist. Dabei wird die Anzahl der Filter, die in der Convolutional-Schicht verwendet werden, definiert, genauso wie die *'kernelSize'*, diese legt die Größe des Filters fest. Die Aktivierungsfunktion bleibt weiterhin ReLU. Als Eingabedimension wird *'1'* als Standardwert definiert, wenn *trainDataTensor.shape[1]* undefiniert ist. Die dritte Schicht ist eine Max-Pooling-Schicht, um die räumliche Dimension zu reduzieren und die Anzahl der Parameter im Modell zu reduzieren. Anschließend wird eine Flatten-Schicht hinzugefügt, um die Ausgabe der Convolutional-Schicht in einen flachen Vektor zu konvertieren, der an die Dense-Schichten, wie bei den vorherigen beiden Modellen, weitergeleitet werden kann.

```
const inputLength = trainDataTensor.shape[1] || 1;
const cnnModel = tf.sequential({
  layers: [
```

```
    tf.layers.embedding({ inputDim: 2, outputDim: 16, inputLength }),
    tf.layers.conv1d({ filters: 32, kernelSize: 3, activation: 'relu', inputShape: [
        inputLength, 16] }),
    tf.layers.maxPooling1d({ poolSize: 2, strides: 2 }),
    tf.layers.flatten(),
    tf.layers.dense({ activation: 'relu', units: 8 }),
    tf.layers.dense({ activation: 'softmax', units: 2 })
  ]
});
this.model = cnnModel
```

Die Ergebnisse sind in der folgenden Tabelle 4.1 dargestellt und zeigen, dass bei allen drei Modellen kein großer Unterschied herrscht. Die Genauigkeit (acc) war bei allen drei ziemlich gleich, nur bei dem Verlustwert (loss) hat man erkennen können, dass das CNN-Modell am niedrigsten ist.

Modell	**acc**	**loss**	**val_acc**	**val_loss**
LTSM	0.99	0.027	0.96	0.47
GRU	0.99	0.039	0.96	0.20
CNN	0.99	0.005	0.98	0.045

Tabelle 4.1: Ergebnisse des Prototyps

5 Conclusio

Zusammenfassend zeigt diese Diplomarbeit ein Verständnis des Themas der Log-Anomalie-Erkennung. Die Arbeit betont die Bedeutung der Protokollierung und Analyse von Logs in einer Ära, in der moderne Computersysteme unaufhörlich wachsen und sich weiterentwickeln. Diese Protokolle bzw. Logs zeigen nicht nur übliche Systemabläufe, sondern sind auch ein nützliches Werkzeug zur Identifizierung fehlerhafter Zustände. Entwicklerinnen und Entwickler und Systemverantwortliche können sie verwenden, um den Systemstatus zu verfolgen und mögliche Probleme zu finden. Eine geeignete Log-Analyse wird mit der Verbreitung verteilter Systeme und den steigenden Anforderungen der Nutzerinnen und Nutzer noch bedeutsamer.
Fortschrittliche Anomalieerkennungstechniken sind aufgrund der wachsenden Menge an Log-Daten und der Vielfalt der Systemumgebungen erforderlich. Modernere Methoden wie Deep Learning mit neuronalen Netzen haben sich als lohnend erwiesen, während traditionelle Techniken wie manuell erstellte Signaturen einige Begrenzungen aufweisen. Die Automatisierung dieser Prozesse verspricht eine kontinuierliche Überwachung und schnelle Reaktion auf Unregelmäßigkeite. Eine zusätzliche Herausforderung besteht darin, die Wahrscheinlichkeit falsch positiver Alarme zu verringern.
Die Diplomarbeit betont, dass die Entwicklung und Optimierung von Anomalieerkennungsmethoden in Logs für die Stabilität, Sicherheit und Leistungsfähigkeit moderner Computersysteme von entscheidender Bedeutung sind. Zukünftige Forschung sollte darauf abzielen, effektive Methoden zur Unterscheidung zwischen relevanten und unwichtigen Informationen zu entwickeln und die Genauigkeit und Effizienz automatisierter Prozesse zu erhöhen.

Im ersten Teil der Arbeit wurden aktuelle Ansätze aus der Literatur [3] eingeführt und miteinander verglichen. Dieser Vergleich hat gezeigt, dass alle Ansätze gute Ergebnisse mit Werten über 0,96 erzielen konnten. Unterschiede zeigten sich hauptsächlich in den Eingabedaten und der Verarbeitungsgeschwindigkeit, wobei einige Ansätze beschriftete Daten erforderten, während andere darauf verzichten konnten.
Die Arbeit von DeepLog [37] basiert auf einem LSTM-Modell und legt den Schwerpunkt auf die Verwendung historischer Daten für die Erkennung von Anomalien und die Vorhersage von Log-Ereignissen.

DeepLog zeichnet sich durch seine Fähigkeit aus, einfache und regelmäßige Daten für das Modell-Training zu verwenden, ohne sich auf vorherige Anomalien zu verlassen. Im Vergleich zu anderen Modellen erkennt DeepLog Anomalien in Protokollen früher, zeigt aber eine geringere Leistung bei komplexen Datensätzen mit hoher Ereignisdichte. Ein Nachteil ist seine Fehleranfälligkeit bei der Protokollanalyse, da es nur den Index der Protokollvorlagen verwendet und deren Bedeutung vernachlässigt. Darüber hinaus benötigt DeepLog einen markierten Datensatz zum Training.

Es wurde der Ansatz 'LogAnomaly' [39] vorgestellt, der ein effektives Modell-Training und die Bewertung neuer Ereignisse durch die Einbeziehung von Log-Templates ermöglicht. Die Erweiterung des Modells mit 'template2vec', basierend auf Synonymen und Antonymen, gibt dem System die Flexibilität, sich an neue Log-Sequenzen anzupassen und verbessert sowohl das Verständnis als auch die Erkennung von Anomalien. LogAnomaly verwendet sequentielle und quantitative Vektoren, um Datenrauschen zu reduzieren, das durch ungenau beschriftete Protokolle verursacht wird. Ähnlich wie DeepLog erkennt LogAnomaly Anomalien frühzeitig und kann große Datensätze verarbeiten, da es auch auf normal beschrifteten Protokollen trainiert wurde. Ein bemerkenswertes Merkmal von LogAnomaly ist seine Fähigkeit, ähnliche Protokollmuster mithilfe semantischer Vektoren zu vergleichen, was zu einer verbesserten Genauigkeit führt. Es teilt jedoch die Einschränkung von DeepLog, dass für das Training beschriftete Datensätze erforderlich sind und der Ereignisindex für das Training verwendet wird, was bei komplexen Datensätzen zu suboptimalen Ergebnissen führen kann.

PLELog [44] führt ein aufmerksamkeitsbasiertes neuronales GRU-Netz und eine probabilistische Erkennungsschätzung ein, um zwischen normalen und abnormalen Sequenzen zu unterscheiden und die Herausforderung einer unzureichenden Erkennung anzugehen. Im Fall von PLELog wird eine Clustering-Methode verwendet, um probabilistische Schätzungen aus unmarkierten Log-Sequenzen zu generieren. Dadurch kann PLELog mit teilweise beschrifteten Protokollen arbeiten. Die Verwendung semantischer Vektoren und eines aufmerksamkeitsbasiertes GRU-Netzwerks erhöht die Effektivität des Ansatzes. Der Clustering-Prozess erfordert jedoch mehr Trainingszeit, und PLELog kann mit dem Rauschen in den Trainingsdaten weniger gut umgehen, was seine Fähigkeit zur frühzeitigen Erkennung von Anomalien beeinträchtigen kann.

LogRobust [50] verwendet TF-IDF-Gewichte und FastText-Modelle, um eine fortschrittliche Methode zur Darstellung von Log-Mustern zu entwickeln, die dann in ein aufmerksamkeitsbasiertes Bi-LSTM-Modell eingespeist werden. LogRobust verwendet semantische Vektoren von Log-Mustern in Verbindung mit einem aufmerksamkeitsbasierten Bi-LSTM-Modell, um kontextuelle Informationen aus Log-Sequenzen zu erfassen und Datenrauschen zu reduzieren. Die überwachte Natur von LogRobust stellt jedoch höhere Anforderungen an das Training, da es sowohl normale als auch anormale Daten benötigt. Dieser Ansatz zeigt

vielversprechende Ergebnisse, erfordert jedoch eine umfangreiche Datenaufbereitung.
Der Einsatz von Faltungsnetzwerken (CNN) [51] für die log-basierte Anomalieerkennung erweitert den technologischen Spielraum und ermöglicht die Transformation von Log-Sequenzen in eine kategorisierbare Matrix für die Anomalieerkennung. Schließlich erlauben die Faltungsoperationen des CNN-Modells nicht nur die Erfassung der Korrelation zwischen Log-Vorlagen, sondern auch die Korrelation innerhalb der semantischen Einbettung dieser Vorlagen. Mit einem überwachten Ansatz erreicht CNN bei vielen Datensätzen eine hohe Genauigkeit. Er erfordert jedoch eine beträchtliche Menge an markierten Daten und kann bei stark unausgewogenen Daten und Datenrauschen an Genauigkeit verlieren.
Um die Erkennung von Anomalien in transformatorbasierten Protokollen zu optimieren, kombiniert 'TransLog' [36] verschiedene Ansätze und Techniken, darunter Pre-Training und adapterbasiertes Matching. Die Überlegenheit von TransLog gegenüber bisherigen Basissystemen bei gleichzeitiger Reduzierung der Parameter und Trainingskosten unterstreicht die Innovation dieses Ansatzes.
Insgesamt haben die Methoden und Modelle, die in dieser wissenschaftlichen Arbeit untersucht wurden, den Forschungsbereich erheblich vorangebracht, indem sie sowohl die Genauigkeit als auch die Effizienz der Erkennung verbessert haben. Diese Erkenntnisse ermöglichen es, zukünftige Sicherheits- und Überwachungssysteme zu entwickeln und zu optimieren.

Im zweiten Abschnitt der Arbeit wurde der Prototyp 'LogDeeptector' implementiert. Das TensorFlow.JjS Framework wurde in einer Angular-App verwendet, um ein neuronales Netzwerk zu erstellen. Während ein Embedding-Service eine numerische Darstellung mithilfe eines individuellen Vokabulars erzeugte, analysierte und aggregierte ein Parser-Service die Log-Daten. Der Model-Service verwendete einen k-means-Algorithmus, um Daten zu clustern und zu ordnen. Der Model-Service hat auch Vorhersagefunktionen und Trainingsmethoden integriert. Die Implementierung und das Training von drei verschiedenen Modellen (LSTM, GRU und CNN) waren Teil der Evaluierung des Prototyps. Die Ergebnisse zeigten, dass die Modelle ähnlich genau waren, aber das CNN-Modell hatte den niedrigsten Verlust (loss).

Abschließend ist zu erwähnen, dass die Diplomarbeit nicht nur einen umfassenden Überblick über die Erkennung von Log-Anomalien und deren Bedeutung bietet, sondern auch einen praktischen Prototyp namens 'LogDeeptector', der als Grundlage für zukünftige Entwicklungen und Forschungen dient, ist aus der Arbeit hervorgekommen. Die präsentierten Erkenntnisse und Schwierigkeiten legen den Grundstein für zukünftige Arbeiten, die sich auf die Verbesserung und Anpassung der Anomalieerkennung in einer sich ständig verändernden IT-Welt konzentrieren können.

5.0.1 Diskussion

Die Implementierung des Prototyps 'LogDeeptector' hat gezeigt, dass es möglich ist, eine eigene Anwendung zur Erkennung von Log-Anomalien zu entwickeln. Um eine Web-Applikation zu erstellen, wurde das relativ neue Framework TensorFlow.js ausgewählt, um zu demonstrieren, dass neuronale Modelle auch in einer Webumgebung umsetzbar sind. Anstelle von Python als bevorzugter Plattform führte diese Entscheidung zu einer gewissen Herausforderung, die bewusst angenommen wurde.

Obwohl der Prototyp erfolgreich war, gab es einige Einschränkungen. Um die Leistungsfähigkeit der Anwendung und des FileReaders zu gewährleisten, musste die Dateigröße und somit die geladenen Logs stark reduziert werden. Da es in Bezug auf Log-Parser keine passende JavaScript-Alternative gab, wurde auf bestehende Parser wie Drain oder Spell verzichtet. Dies veranlasste die Verwendung von regulären Ausdrücken (Regex) beim Parsing.

Obwohl ein vorab trainiertes TensorFlow.js-Modell für die Verwendung von Encodern und Vokabularien in Betracht gezogen wurde, wurde letztendlich ein eigenes Vokabular erstellt und für das numerische Encoding verwendet.

Da der verwendete Datensatz nicht beschriftet war, war die Frage der True-Falsch-Positiv-Klassifizierung eine Herausforderung. Darum war es schwierig, über Anomalien klare Aussagen zu machen. Somit konnte man keine klaren true-positives und false-positives deklarieren. Das erschwert die Evaluierung von den drei Modellen, da man keine Berechnung zu den Bewertungsmetriken machen konnte. Somit wurde als Ergebnis die Genauigkeit (acc) und der Verlust (loss) des trainierten Modells aufgezeichnet.

Es gibt Raum für Verbesserungen in Bezug auf die numerische Darstellung der eingebetteten Logs. Die aktuelle Version zeigt nur die Reihenfolge der Wörter im Vokabular an. In diesem Fall könnte mehr Logik hinzugefügt werden, um die Häufigkeit und Auffälligkeit der numerischen Werte zu berücksichtigen. Da anhand der Distanz zwischen den Datenpunkten und den Zentren klassifiziert wird, wären Datenpunkte mit bestimmten numerischen Werten aussagekräftiger, das würde das Clustering-Ergebnis erheblich verbessern und man hat eine höhere Wahrscheinlichkeit Anomalien zu erkennen.

Um einen Überblick zu verschaffen, wurden neben der praktischen Umsetzung auch die Grundlagen des Themas behandelt. Hier wurde bewusst eine grobe Form beibehalten, um als Basis zu dienen.

Im Laufe der Arbeit wurden verschiedene Methoden erläutert, um die aktuellen Forschungsergebnisse zu definieren. Die Literatur [3], hat sich zu diesem Zeitpunkt (Jahr 2022) als am vielversprechendsten erwiesen und wurde somit als Grundlage für die Auswahl der Methoden verwendet. Darüber hinaus wurde der Ansatz eines Transformers hinzugefügt, da er bisher in der Literatur [3] nicht erwähnt wurde und für aktuelle Entwicklungen wie ChatGPT relevant erschien.

5.1 Weiterführende Arbeiten

Zukünftige Forschungsgebiete und Fortschritte in der Log-Anomalie-Erkennung bieten interessante Möglichkeiten, um die fortlaufende Entwicklung von neuronalen Netzen und künstlicher Intelligenz zu nutzen. Angesichts der wachsenden Präsenz von KI-Technologien und der wachsenden Faszination für Tools wie ChatGPT wird es immer wichtiger, neue Ansätze zur Bewältigung komplexer Herausforderungen zu erforschen. Die kommenden Jahre werden durch die stetige Zunahme von Systemen gekennzeichnet, die große Mengen an Logs erzeugt werden. Die Verarbeitung und Analyse dieser Daten wird für die Effizienz und Sicherheit moderner IT-Infrastrukturen unerlässlich sein.
Um diese aufstrebenden Forschungsbereiche voranzutreiben, gibt es eine Reihe vielversprechender Strategien. Weitere Anomalieerkennungsmethoden könnten in zukünftige Arbeiten integriert und bewertet werden. Angesichts der Tatsache, dass neue Modelle und Methoden immer noch entwickelt werden, wäre es interessant, sie mit etablierten Methoden zu vergleichen, um das Potenzial und die Wirksamkeit verschiedener Ansätze besser zu verstehen.

Da der aktuelle Prototyp nur ein erster Schritt ist, gibt es zahlreiche Optionen zur Verbesserung. Beispielsweise kann der Parsing-Service optimiert werden, um eine breitere Palette von unterschiedlichen Log-Formaten verarbeiten zu können. Die Erweiterung der regulären Ausdrücke für die Log-Analyse würde die Anpassungsfähigkeit und Flexibilität des Prototyps erhöhen.

Die numerische Darstellung der Logs könnte durch eine Neugestaltung des Embedding-Prozesses und die Einbindung einer umfangreichen Sammlung von Vokabularien verbessert werden, was zu einem aussagekräftigeren Ergebnis des Clustering führen könnte.

Um die Leistung des Prototyps zu verbessern, gibt es Möglichkeiten, die Schichten und Parameter zu op-

timieren. Systematische Experimente könnten verwendet werden, um die optimalen Konfigurationen zu finden, um die Genauigkeit und Effektivität der Anomalieerkennung zu steigern.

Abschließend könnten die Grundlagen vertieft werden, indem zusätzliche Log-Parser präsentiert und ihre Leistung verglichen wird. Ein umfassendes Verständnis der verschiedenen Parsing-Techniken und deren Einsatzgebiete würde den Lesern helfen, die Vielfalt der verfügbaren Techniken zu verstehen.

Insgesamt scheint die Zukunft der Log-Anomalie-Erkennung vielversprechend zu sein. Dies passt gut zum breiteren Kontext der fortlaufenden Entwicklungen im Bereich der künstlichen Intelligenz und neuronalen Netze. Die fortlaufende Forschung in diesem Bereich wird dazu beitragen, effektive Lösungen für die Herausforderungen der wachsenden Datenmengen und komplexen Systemumgebungen zu finden.

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

API	Application Programming Interface
CNN	Convolutional Neural Network
GRU	Gated recurrent units
IT	Information Technologie
JS	JavaScript
KI	Künstliche Intelligenz
LSTM	Long Short-Term Memory
WEBAPP	Web Applikation

Literatur

[1] Max Landauer, Sebastian Onder, Florian Skopik und Markus Wurzenberger, "Deep Learning for Anomaly Detection in Log Data: A Survey", *arXiv preprint arXiv:2207.03820*, 2022.

[2] Marek Souček, "Log Anomaly Detection", 2020.

[3] Van-Hoang Le und Hongyu Zhang, "Log-based anomaly detection with deep learning: How far are we?", in *Proceedings of the 44th International Conference on Software Engineering*, 2022, S. 1356–1367.

[4] Max Landauer, Florian Skopik, Markus Wurzenberger und Andreas Rauber, "System log clustering approaches for cyber security applications: A survey", 2020, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.101739. Adresse: https://www.sciencedirect.com/science/article/pii/S0167404820300250.

[5] Shilin He, Jieming Zhu, Pinjia He und Michael R Lyu, "Experience report: System log analysis for anomaly detection", in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, IEEE, 2016, S. 207–218.

[6] Christopher Kruegel und Giovanni Vigna, "Anomaly detection of web-based attacks", in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, S. 251–261.

[7] Max Landauer, Markus Wurzenberger, Florian Skopik, Giuseppe Settanni und Peter Filzmoser, "Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection", *computers & security*, Jg. 79, S. 94–116, 2018.

[8] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin und Kuang-Yuan Tung, "Intrusion detection system: A comprehensive review", *Journal of Network and Computer Applications*, Jg. 36, Nr. 1, S. 16–24, 2013.

[9] Varun Chandola, Arindam Banerjee und Vipin Kumar, "Anomaly detection: A survey", *ACM computing surveys (CSUR)*, 2009.

[10] Meena Siwach und Suman Mann, “ISSUES AND CHALLENGES IN ANOMALY DETECTION FOR WEB LOG DATA”, 2022.

[11] *OWASP Cheat Sheet Series: Logging Cheat Sheet*, https://cheatsheetseries.owasp.org/cheatsheets/Logging$_{C}heat_{S}hee$ Accessed: 2023-08-29.

[12] S Tejaswini und Azra Nasreen, “Survey on Online Log Parsers”,

[13] *Elastic: Elastic Stack*, https://www.elastic.co/de/elastic-stack, Accessed: 2023-08-29.

[14] *Splunk: Splunk Log Observer*, https://www.splunk.com/en$_{u}s/blog/learn/splunk{-}log{-}observer.html$, Accessed: 2023-08-29.

[15] *Vercel: Working with Log Drains*, https://vercel.com/docs/observability/log-drains-overview/log-drains, Accessed: 2023-08-29.

[16] Shilin He, Jieming Zhu, Pinjia He und Michael R Lyu, “Loghub: a large collection of system log datasets towards automated log analytics”, *arXiv preprint arXiv:2008.06448*, 2020.

[17] Pinjia He, Jieming Zhu, Zibin Zheng und Michael R Lyu, “Drain: An online log parsing approach with fixed depth tree”, in *2017 IEEE international conference on web services (ICWS)*, IEEE, 2017, S. 33–40.

[18] Min Du und Feifei Li, “Spell: Streaming parsing of system event logs”, in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, IEEE, 2016, S. 859–864.

[19] Keiichi Shima, “Length matters: Clustering system log messages using length of words”, *arXiv preprint arXiv:1611.03213*, 2016.

[20] Adetokunbo AO Makanju, A Nur Zincir-Heywood und Evangelos E Milios, “Clustering event logs using iterative partitioning”, 2009, S. 1255–1264.

[21] Liang Tang, Tao Li und Chang-Shing Perng, “LogSig: Generating system events from raw textual logs”, in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, S. 785–794.

[22] Amey Agrawal, Rohit Karlupia und Rajat Gupta, “Logan: A distributed online log parser”, in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, IEEE, 2019, S. 1946–1951.

[23] Masayoshi Mizutani, “Incremental mining of system log format”, in *2013 IEEE International Conference on Services Computing*, IEEE, 2013, S. 595–602.

[24] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang und Tse-Hsun Chen, "Logram: Efficient Log Parsing Using n-Gram Dictionaries", *IEEE Transactions on Software Engineering*, Jg. 48, Nr. 3, S. 879–892, 2020.

[25] *RDocumentation: loess.as: Fit a local polynomial regression with automatic smoothing parameter selection*, https://www.rdocumentation.org/packages/fANCOVA/versions/0.6-1/topics/loess.as, Accessed: 2023-08-29.

[26] *RDocumentation: Ckmeans.1d.dp: Optimal K-means Clustering in One-dimension by Dynamic Programming*, https://www.rdocumentation.org/packages/Ckmeans.1d.dp/versions/3.4.0-1/topics/Ckmeans.1d.dp, Accessed: 2023-08-29.

[27] *What Is Anomaly Detection?*, https://www.splunk.com/en_us/data-insider/anomaly-detection.html, Accessed: 2023-03-02.

[28] Christian Janiesch, Patrick Zschech und Kai Heinrich, "Machine learning and deep learning", *Electronic Markets*, Jg. 31, Nr. 3, S. 685–695, 2021.

[29] Stuart J Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[30] *What is machine learning?*, https://www.ibm.com/topics/machine-learning, Accessed: 2023-03-02.

[31] *What is a neural network?*, https://www.ibm.com/topics/neural-networks, Accessed: 2023-03-02.

[32] *What is deep learning?*, https://www.ibm.com/topics/deep-learning, Accessed: 2023-03-02.

[33] *What are convolutional neural networks?*, https://www.ibm.com/topics/convolutional-neural-networks, Accessed: 2023-03-02.

[34] *What are recurrent neural networks?*, https://www.ibm.com/topics/recurrent-neural-networks, Accessed: 2023-03-02.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser und Illia Polosukhin, *Attention Is All You Need*, 2017. DOI: 10.48550/ARXIV.1706.03762. Adresse: https://arxiv.org/abs/1706.03762.

[36] Hongcheng Guo, Xingyu Lin, Jian Yang, Yi Zhuang, Jiaqi Bai, Tieqiao Zheng, Bo Zhang und Zhoujun Li, *TransLog: A Unified Transformer-based Framework for Log Anomaly Detection*, 2022. DOI: 10.48550/ARXIV.2201.00016. Adresse: https://arxiv.org/abs/2201.00016.

[37] Min Du, Feifei Li, Guineng Zheng und Vivek 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*, 2017, S. 1285–1298.

[38] Sepp Hochreiter und Jürgen Schmidhuber, “Long short-term memory”, *Neural computation*, Jg. 9, Nr. 8, S. 1735–1780, 1997.

[39] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun u. a., “LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.”, in *IJCAI*, Bd. 19, 2019, S. 4739–4745.

[40] *Github: FT-Tree*, https://github.com/WeibinMeng/FT-Tree, Accessed: 2023-03-02.

[41] Tomas Mikolov, Kai Chen, Greg Corrado und Jeffrey Dean, “Efficient estimation of word representations in vector space”, *arXiv preprint arXiv:1301.3781*, 2013.

[42] George A Miller, “WordNet: a lexical database for English”, *Communications of the ACM*, Jg. 38, Nr. 11, S. 39–41, 1995.

[43] Kim Anh Nguyen, Sabine Schulte im Walde und Ngoc Thang Vu, “Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction”, *arXiv preprint arXiv:1605.07766*, 2016.

[44] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong und Wenbin Zhang, “Semi-supervised log-based anomaly detection via probabilistic label estimation”, in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, 2021, S. 1448–1460.

[45] Ya-Lin Zhang, Longfei Li, Jun Zhou, Xiaolong Li, Yujiang Liu, Yuanchao Zhang und Zhi-Hua Zhou, “POSTER: A PU learning based system for potential malicious URL detection”, in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, S. 2599–2601.

[46] RichardSocher JeffreyPennington und ChristopherD Manning, “Glove: Global vectors for word representation”, in *Conference on Empirical Methods in Natural Language Processing. Citeseer*, 2014.

[47] Gerard Salton und Christopher Buckley, “Term-weighting approaches in automatic text retrieval”, *Information processing & management*, Jg. 24, Nr. 5, S. 513–523, 1988.

[48] Leland McInnes und John Healy, “Accelerated hierarchical density based clustering”, in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2017, S. 33–42.

[49] Erkki Oja und Zhijian Yuan, “The FastICA algorithm revisited: Convergence analysis”, *IEEE transactions on Neural Networks*, Jg. 17, Nr. 6, S. 1370–1381, 2006.

[50] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li u. a., "Robust log-based anomaly detection on unstable log data", in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, S. 807–817.

[51] Siyang Lu, Xiang Wei, Yandong Li und Liqiang Wang, "Detecting anomaly in big data system logs using convolutional neural network", in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, IEEE, 2018, S. 151–158.

[52] Nils Reimers und Iryna Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks", *arXiv preprint arXiv:1908.10084*, 2019.

[53] Jacob Devlin, Ming-Wei Chang, Kenton Lee und Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, 2018.

[54] Adam Oliner und Jon Stearley, "What supercomputers say: A study of five system logs", in *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*, IEEE, 2007, S. 575–584.

[55] Wei Xu, Ling Huang, Armando Fox, David Patterson und Michael Jordan, "Largescale system problem detection by mining console logs", *Proceedings of SOSP'09*, 2009.

[56] *TensorFlow.js*, https://www.tensorflow.org/js/, Accessed: 2023-08-02.

[57] *TensorFlow*, https://www.tensorflow.org/, Accessed: 2023-06-04.

[58] *Angular: The web development framework for building the future*, https://angular.io/, Accessed: 2023-08-30.

[59] *CLI Overview and Command Reference*, https://angular.io/cli, Accessed: 2023-06-06.

[60] *Windows Subsystem for Linux Documentation*, https://learn.microsoft.com/en-us/windows/wsl/, Accessed: 2023-05-06.

[61] *NPM*, https://www.npmjs.com/, Accessed: 2023-06-06.

[62] *Github: NVM*, https://github.com/nvm-sh/nvm, Accessed: 2023-05-06.

[63] *Kaggle: Level up with the largest AI and ML community*, https://www.kaggle.com/, Accessed: 2023-08-30.