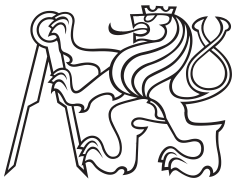
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Porovnání reprezentací souborů logů pro metody detekce anomálií

Martin Hubal

Školitel: Ing. Jan Drchal, Ph.D.
Leden 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hubal** Jméno: **Martin** Osobní číslo: **492166**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Porovnání reprezentací souborů logů pro metody detekce anomálií

Název bakalářské práce anglicky:

Comparing Log File Representations for Anomaly Detection Methods

Pokyny pro vypracování:

Cíle práce jsou následující:
1) Nastudujte moderní metody detekce anomálií v souborech logů. Zaměřte se na způsoby, jakými mohou být reprezentovány jednotlivé řádky logů či časová okna skupiny řádek. Věnujte se jak metodám založeným na šablonách (log keys, BOW reprezentace), tak i metodám založeným na číselných reprezentacích (embeddings) typu fastText.
2) Na vybraných datových sadách natrénujte vybrané typy modelů.
3) Porovnejte kvalitu modelů. Ověřte robustnost metod vzhledem ke zprávám logů, které nebyly přesně reprezentovány trénovacími daty.

Seznam doporučené literatury:

[1] Martin, Koryťák. "Anomaly Detection Methods for Log Files. MS thesis." České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021.
[2] Prokop, Černý. "Contextual Embeddings for Anomaly Detection in Log Files." MS thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021.
[3] Du, Min, et al. "Deeplog: Anomaly detection and diagnosis from system logs through deep learning." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017.
[4] Guo, Haixuan, Shuhan Yuan, and Xintao Wu. "LogBERT: Log Anomaly Detection via BERT." arXiv preprint arXiv:2103.04475 (2021).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jan Drchal, Ph.D. centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Jan Drchal, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu Janu Drchalovi za jeho pomoc a vedení při tvorbě bakalářské práce. Dále bych chtěl poděkovat své rodině a kamarádovi Šimonovi Zvárovi za podporu při celém mém studiu.

Prohlášení

Prohlašuji, že jsem zadanou práci zpracoval samostatně s přispěním vedoucího práce. Dále prohlašuji, že jsem všechny použité zdroje správně a úplně citoval a uvádím je v přiloženém seznamu použité literatury. Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů v platném znění

Abstrakt

Tato bakalářská práce se zabývá metodami detekce anomálií nad logy. Porovnávají se dva modely - AECNN1D, který je založen na číselné reprezentaci dat typu fastText, a Vanilla autoencoder, který je založen na šablonách, kde se data reprezentují pomocí bag-of-words. U obou modelů porovnáváme jejich robustnost tím, že se v testovacích datech objevují i typy logů, na které model nebyl natrénován. Pro porovnání jsou použity HDFS datové soubory. Modely jsou trénovány na upravených HDFS datech, kde se určité logové typy odstraní či zakryjí. Závěr experimentů je ten, že model AECNN1D nebyl schopný dobře reagovat na nové typy logů, na kterých nebyl natrénován. To si vysvětlujeme malým počtem rozdílných typů logů v HDFS datech.

Klíčová slova: detekce anomálií, soubory logů, NLP, strojové učení

Školitel: Ing. Jan Drchal, Ph.D.

Abstract

This bachelor's thesis deals with anomaly detection methods for log files. Two models are compared - AECNN1D, which is based on a numerical representation of the data with fastText embeddings, and the Vanilla autoencoder, which is based on templates, where the data is represented using bag-of-words. We compare the robustness of both of these models by including types in the test data on which the model was not trained. HDFS data files are used for comparison. The models are trained on modified HDFS data, where certain log types are removed or masked. The conclusion of the experiments is that the AECNN1D model was not able to perform well on the new log types on which it was not trained. This is explained by the small number of different log types in HDFS data.

Keywords: anomaly detection, log files, NLP, machine learning

Title translation: Comparing Log File Representations for Anomaly Detection Methods

Obsah

1 Úvod **1**
1.1 Motivace 1
1.2 Struktura práce 2

2 Moderní metody detekce anomálií **3**
2.1 Anomálie 3
2.1.1 Bodové (point) anomálie 3
2.1.2 Kontextové (Contextual) anomálie 4
2.1.3 Kolektivní (collective) anomálie 4
2.2 Typy strojového učení 5
2.2.1 Nesupervizované 5
2.2.2 Supervizované 5
2.2.3 Semi-supervizované 6
2.3 Proces detekce anomálií 6
2.3.1 Kolekce logů 6
2.3.2 Parsování logů 7
2.3.3 Extrakce funkcí 7
2.3.4 Detekce anomálií 8

3 Přehled použitých metod a dat **9**
3.1 HDFS 9
3.2 Drain3 10
3.3 Číselné reprezentace dat 10
3.3.1 Bag-of-words 10
3.3.2 FastText 11
3.4 Střední kvadratická chyba a ADAM 11
3.5 Metriky 13

4 Příprava na experimenty **14**
4.1 Vanilla autoencoder 14
4.1.1 Úprava číselných reprezentací 14
4.1.2 Autoenkodér 14
4.1.3 Popis modelu 14
4.1.4 Hyperparametry 15
4.2 AECNN1D 15
4.2.1 Úprava číselných reprezentací 16
4.2.2 Popis modelu 16
4.2.3 Hyperparametry 17
4.3 Příprava dat 18

5 Evaluation **20**
5.1 Vanilla autoencoder 20
5.1.1 Změněné logy 20
5.1.2 Odstraněné logy 21
5.2 AECNN1D 22

6 Závěr **24**

A Tabulky **26**

B Literatura **28**

Obrázky

2.1 Ukázka bodových anomálií. Jako modré body jsou označena normální data, kdežto červeně jsou označena data anomální. 3
2.2 Ukázka kontextové anomálie. Čas je kontextový atribut, teplota je behaviorální. Přestože body A i B mají stejnou hodnotu teploty, tak A je normální bod a B je anomální bod vzhledem ke kontextu. 4
2.3 Ukázka kolektivní anomálie. Červené body jsou anomální, protože společně mají podobné hodnoty po mnohem delší dobu než normální data. 5
2.4 Procesy analýzy logů. *Obrázek převzat z [8]* 6
2.5 Boxplot na odstraněných datech . 7

3.1 Příklad reprezentace bloku logů pomocí fastTextu. 11
3.2 Histogram výskytu šablon v HDFS1 datech s logaritmickou stupnicí. 11
3.3 Histogram výskytu délky bloků v HDFS1 datech s logaritmickou stupnicí. 12

4.1 Ukázka oříznutí bloků na velikost 4. 16
4.2 Ukázka logu s šablonou *PacketResponder <*> for block <*> <*>* změněného na log s šablonou *others: unknown <*>*. 18

5.1 The LOF caption 21
5.2 The LOF caption 22

Tabulky

1.1 Example class model. 2

3.1 Statistiky o HDFS1 datech. 9
3.2 Ukázka šablon logů v HDFS1 datech i s jejich Drain identifikátorem a počtem výskytů v datech. 10

5.1 Přehled charakteristik pro různý počet změněných typů logů Vanilla encoder modelu. Q1 a Q3 označuje první a třetí kvartil. Poslední sloupec jsou směrodatné odchylky. 21
5.2 Přehled charakteristik pro různý počet odstraněných typů logů u Vanilla encoder modelu. Q1 a Q3 označuje první a třetí kvartil, poslední sloupec jsou směrodatné odchylky. 22

A.1 Výsledné F1-score pro model Vanilla autoencoder se změněnými logy pro 10 různých permutací vybraných šablon. 26
A.2 Výsledné F1-score pro model Vanilla autoencoder s odstraněnými logy pro 10 různých permutací vybraných šablon. 27
A.3 Výsledné údaje pro model AECNN1D pro 10 různých permutací vybraných šablon. 27

Kapitola 1
Úvod

Softwarové a operační systémy vytváří logové soubory, aby zaznamenaly, jaké události se v nich staly. Tyto soubory obsahují mnoho důležitých informací pro vývojáře, kteří jsou s jejich pomocí schopni detekovat anomální událostí a zároveň tím i zjistit příčiny chyb. V dnešní době systémy vytváří tak velké množství logů, že je často nemožné, aby je byli lidé schopni manuálně analyzovat. Z důvodu velkého množství systémy vygenerovaných logů vzniklo v posledních letech na téma detekce anomálií mnoho výzkumů se snahou přiblížení se automatizace odhalování anomálií. Touto problematikou se zabýváme i v této bakalářská práce, jejíž tématem je porovnávání metod detekce anomálií.

1.1 Motivace

Tato bakalářská práce navazuje na práci [13], která se zabývá hledáním anomálií v logách pomocí nových modelů fungujících na principu NLP (zpracování přirozeného jazyka) v kombinaci se strojovým učením, v tabulce 1.1 jsou zobrazeny jejich výsledky. Modely jsou rozděleny na dvě části podle toho, jakým způsobem se logy zpracovávají. Oranžově jsou označeny ty modely využívající bag-of-words reprezentaci, ta vytváří číselný vektor z bloku logů se stejným identifikátorem (viz 3.3.2). Ostatní metody používají fastText, který vytváří číselné vektory ze slov (viz 3.3.2). Lze si všimnout, že si v hlavní metrice F1-score nejlépe vede model využívající bag-of-words.

V souborech logů se může stát, že se vyskytnou logy, jejichž typy jsme do té doby neviděli a z toho důvodu nejsme naučeni rozpoznat, jestli se jedná o anomálii. Tímto problémem se v této bakalářské práci zabýváme.

Naše hypotéza je taková, že modely využívající fastText budou schopny lépe reagovat na nové typy logů z toho důvodu, že fastText zvládne zpracovat do číselných vektorů i logy, jejíchž typy ještě neviděl. Oproti tomu modely reprezentované bag-of-words na nové typy logů schopny reagovat nejsou a model je nucen buď nové typy logů přeskočit, anebo je potřeba model přeučit.

V této práci jsme se rozhodli porovnat nejlepší model používající bag-of-words, kterým je Vanilla autoencoder, a nejlepší model používající fastText, kterým je AECNN1D. U těchto dvou modelů porovnáme jejich robustnost, jak jsou schopny reagovat na nové logy. Tím ověříme naší hypotézu, zda

Model	validace F1-score	testování F1-score
Local Outlier Factor	0,6062	0,5866
Isolation Trees	0,8155	0,8086
Vanilla autoencoder	**0,8696**	**0,8779**
TCN	0,5545	0,5334
CNN1D	0,8273	0,8043
CNN2D autoencoder	0,8089	0,7906
CNN1DTCN	0,8342	0,8173
AETCN	0,8400	0,8168
AECNN1D	**0,8528**	**0,8597**
SACNN1D	0,8280	0,8103
SACNN2D	0,7878	0,7694

Tabulka 1.1: Example class model.

skutečně model využívající fastText umí lépe pracovat s novými typy logů.

Pro experimenty jsou použita HDFS data, která jsou největšími veřejně dostupnými soubory logů, které mají anotované anomálie pro sekvence logů, čímž se v této bakalářské práci zabýváme.

1.2 Struktura práce

Práce je rozdělená do šesti kapitol. V druhé kapitole jsou představeny moderní metody detekce anomálií (state-of-the-art). Ve třetí kapitole se zabýváme metodami použitými při experimentech. Ve čtvrté kapitola je popsána příprava na experimenty a jsou popsány oba použité modely AECNN1D a Vanilla encoder. V páté kapitole se vyhodnotí naše experimenty. Nakonec v šesté kapitole je závěr celé této práce.

Kapitola 2

Moderní metody detekce anomálií

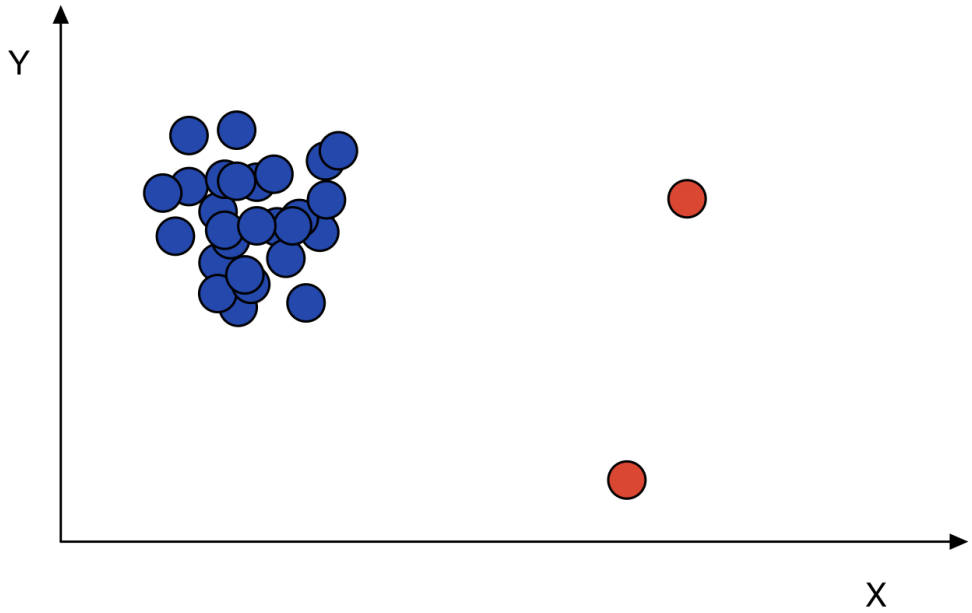
V této kapitole si představíme moderní metody detekce anomálií. Nejdříve si ale anomálie definujeme a rozdělíme.

2.1 Anomálie

Anomálie jsou vzory dat, které nevyhovují definovanému normálnímu chování. Rozdělujeme si je na tři typy, těmi jsou anomálie bodové, kontextové a kolektivní.

2.1.1 Bodové (point) anomálie

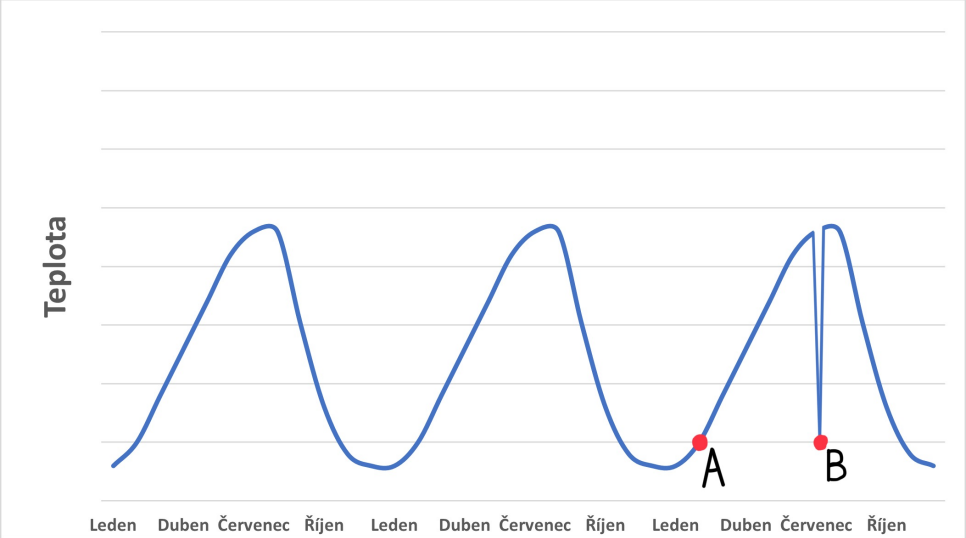
Pokud individuální instance dat může být brána jako anomální vůči zbytku dat, tak je brána jako bodově anomální. Příklad je na 2.1.



Obrázek 2.1: Ukázka bodových anomálií. Jako modré body jsou označena normální data, kdežto červeně jsou označena data anomální.

2.1.2 Kontextové (Contextual) anomálie

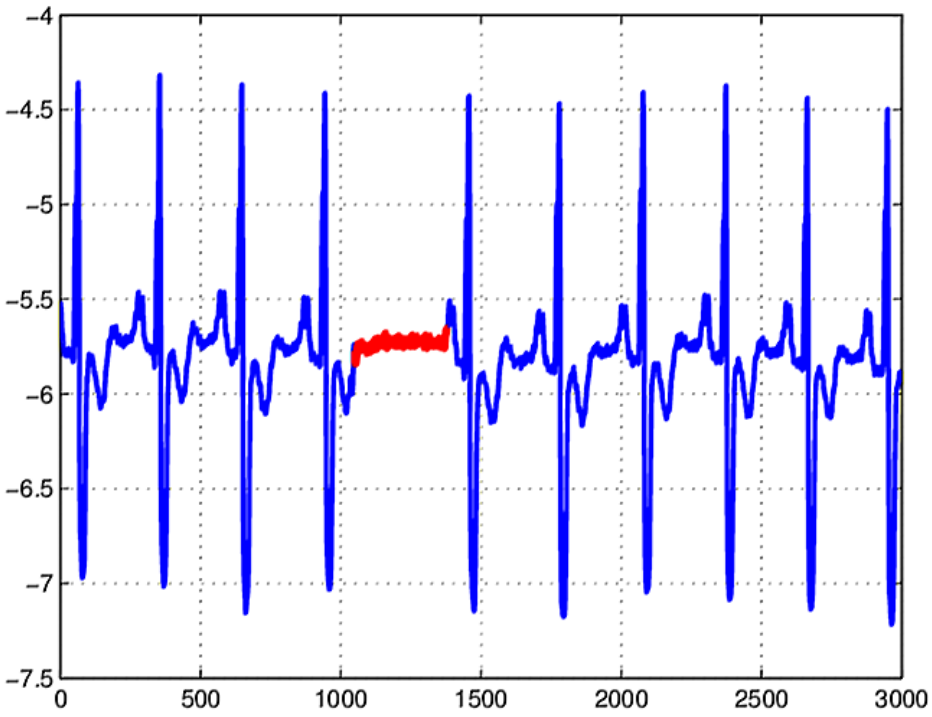
Pokud je data instance anomální v daném kontextu, ale jinak anomální není, tak je to kontextová anomálie. Každá datová instance má atributy kontextové a behaviorální. Kontextové atributy určují kontext, kterým může být například čas. Behaviorální pak vyjadřuje zbytek informací instance dat, která nejsou kontextová, jako je například teplota. Příklad je na obrázku 2.2.



Obrázek 2.2: Ukázka kontextové anomálie. Čas je kontextový atribut, teplota je behaviorální. Přestože body A i B mají stejnou hodnotu teploty, tak A je normální bod a B je anomální bod vzhledem ke kontextu.

2.1.3 Kolektivní (collective) anomálie

Pokud kolekce spolu souvisejících dat je anomální vůči všem datům, tak jde o kolektivní anomálii. Individuální data této kolekce anomální být nemusí, ale kolekce jako celek anomální je. Příklad je obrázek 2.3.



Obrázek 2.3: Ukázka kolektivní anomálie. Červené body jsou anomální, protože společně mají podobné hodnoty po mnohem delší dobu než normální data.

2.2 Typy strojového učení

Učení rozdělujeme na tři typy podle označení (label) dat na vstupu, tyto typy jsou nesupervizované, supervizované a semi-supervizované učení.

U označených dat máme informaci, jestli jsou anomální nebo normální.

2.2.1 Nesupervizované

V tomto typu učení nemáme u žádných dat označení, model předpokládá, že normální data se vyskytují mnohem častěji než anomální. Pokud tomu tak není, tak naučený model není schopen správně anomálie detekovat.

2.2.2 Supervizované

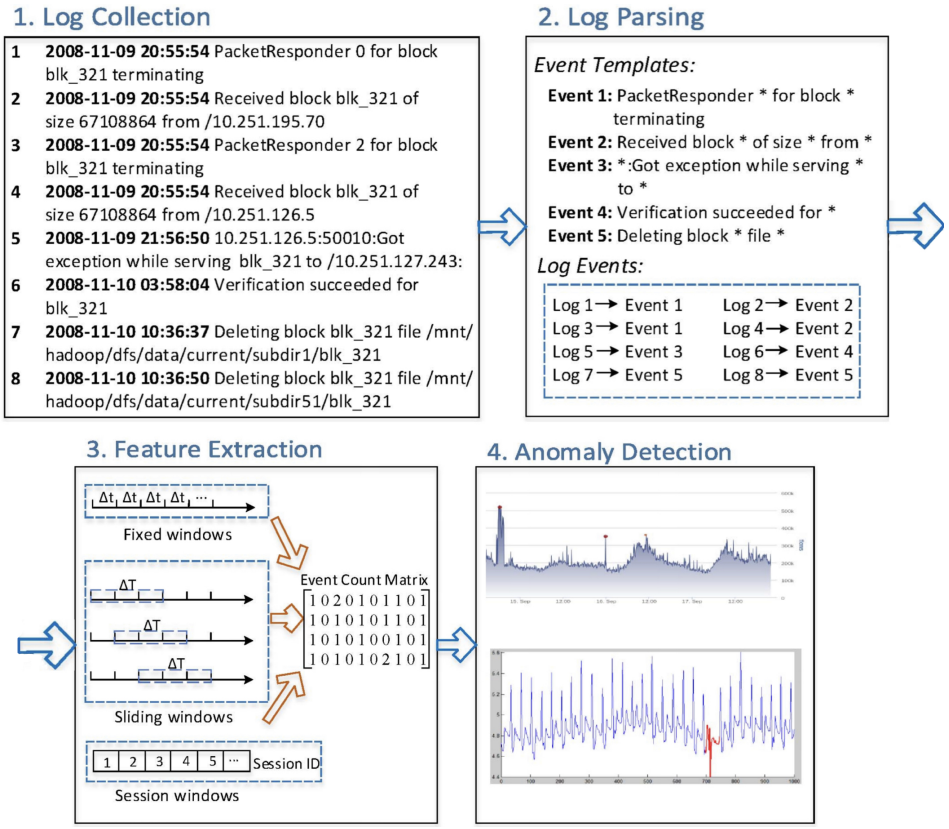
Model předpokládá, že trénovací data mají u sebe označení jak u normálních dat, tak i u anomálních. Pomocí označených dat se model snaží naučit predikovat normální a anomální data. Nevýhody supervizovaného způsobu jsou ty, že je velmi obtížné, často i nemožné, mít dostatečný počet označených dat. Dále výskyt anomálních dat je v porovnání s normálními daty mnohem menší, což vytváří nerovnováhu označení v naučeném modelu.

2.2.3 Semi-supervizované

Model předpokládá, že označení je přítomno jen u normálních dat. Model může být lépe schopný než supervizovaný přístup detekovat neobvyklé anomálie, které je obtížné zastoupit v trénovacích datech.

2.3 Proces detekce anomálií

Proces analýz popíšeme rozdělením do čtyř částí uvedených v [8]. Těmito části jsou kolekce, parsování, číselná reprezentace a detekce anomálií za pomoci modelů. Na obrázku 2.4 je ukázaný příklad tohoto rozdělení.



Obrázek 2.4: Procesy analýzy logů. *Obrázek převzat z [8]*

2.3.1 Kolekce logů

Nejdříve je potřeba logy generované rozsáhlými systémy shromáždit. Systémy pravidelně vytváří logy k ukládání systémového stavu a informací o běhu. Každý log obsahuje časový údaj a logovou zprávu, co se událo. Na obrázku 2.4 u části *Log Collection* je příklad 8 logů.

2.3.2 Parsování logů

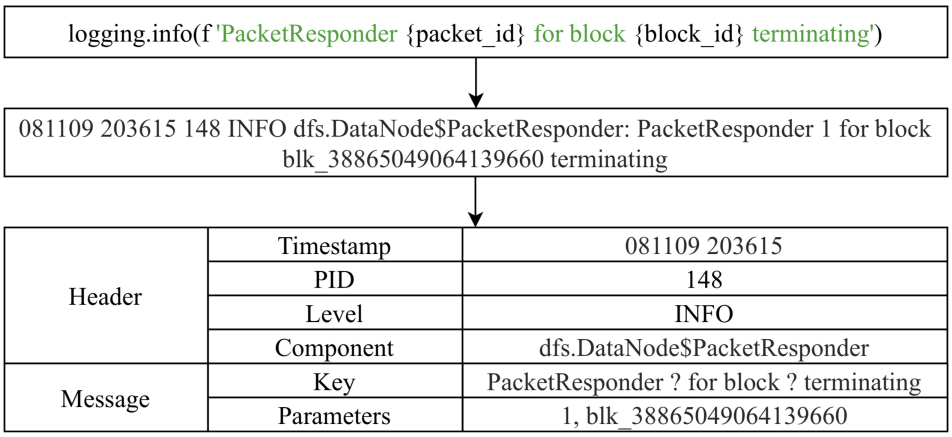
Generované logy nemají danou strukturu a obsahují volný text. Úkolem parsování je získat strukturovanou reprezentaci dat z nestrukturovaných logů, abychom je byli schopni lépe zpracovat.

Nejdříve si definujeme pár důležitých pojmů týkajících se logů, které v této bakalářské práci používáme. Na obrázku 2.5 je ukázka vzniku HDFS1 logu a jeho uložení do strukturované podoby rozdělené na hlavičku (Header) a zprávu (Message).

Hlavička obsahuje informace o kontextu logu jako je časový údaj události, PID a úroveň (Level). V uvedeném příkladu 081109 203615 značí čas 8. ledna 2009, 20:36:15. Úroveň určuje důležitost logové zprávy, jejími příklady jou INFO, WARN nebo ERROR. PID je identifikační číslo procesu.

Zpráva je rozdělena na **šablonu** (template) a parametry (parameters). Šablona je konstantní část zprávy, kterou získáme odebráním všech proměnných (parametrů) z celé zprávy. O šabloně v této bakalářské práci mluvíme také jako o **typu logu** (log key).

Metod parsování existuje mnoho, v [23] jsou různé metody porovnány. Jejich příklady jsou Iterative partioning(IPLoM [14]), Hierarchické shlukování (LKE [6]), nejdelší společná podposloupnost (Spell [4]), parsovací strom (Drain [7]). Nejlepších výsledků dosáhl Drain, který i využijeme v této práci viz sekce 3.2.



Obrázek 2.5: Boxplot na odstraněných datech

2.3.3 Extrakce funkcí

Po parsování potřebujeme strukturovaná data změnit na číselné vektory s předem danými vlastnostmi, na kterých modely strojového učení mohou být použity.

Reprezentaci logů číselným vektorem můžeme rozdělit na dva typy. První způsob reprezentuje sekvenci několika logů. Druhým způsobem reprezentujeme vektorem každý log zvlášť.

Logy můžeme rozdělit do sekvencí například podle časového okna, nebo podle podle počtu logů. U HDFS dále můžeme využít bloky, do kterých jsou logy rozděleny. Jedním ze základních metod reprezentace sekvencí číselným vektorem je bag-of-words, které je založeno na počtu výskytu šablon. Tuto metodu využijeme i v této bakalářské práce viz 3.3.1.

Další možností reprezentace sekvence je LogEvent2vec[18], který využívá word2vec[15]. Word2vec v tomto případě každý log bere jako slovo, s jejichž pomocí pak se pak sekvence reprezentují.

Při reprezentaci každého logu vektorem jsou často využity metody na reprezentaci slov jako jsou word2vec a fastText.

Odlišný způsob je použitý u Deeplogu[5], který oddělené šablony a parametry reprezentuje zvlášť.

2.3.4 Detekce anomálií

Po reprezentování logů číselnými vektory můžeme využít modely strojového učení. Modely jsou rozděleny podle dostupnosti označených dat uvedených v 2.2.

Příkladem nesupervizovaného modelu, který se používá na anomálie, je PCA[20] nebo rekurentní neauronová síť[2].

Supervizovanými modely jsou pak například logistická regrese, rozhodovací strom a metoda podpůrných vektorů, které byly porovnány v[9] s dalšími supervizovanými modely.

Kapitola 3

Přehled použitých metod a dat

V této kapitole si uvedeme metody a popíšeme data, které využijeme při našich experimentech.

3.1 HDFS

K experimentům využijeme data HDFS1 a HDFS2 [1]. Statistiky o HDFS1 datech jsou v 3.1. Tyto data obsahují více než 11 miliónů logů, které jsou rozděleny do přibližně 570 tisíc bloků, každý blok má k sobě označení, jestli se jedná o anomální či normální blok. Z celkového počtu bloků je 2,93 % bloků anomálních. Počet různých šablon logů je 48. HDFS1 logy jsou rozděleny do bloků podle identifikátorů, každý log obsahuje blk_$ID, kde $ID je číslo bloku, jehož součástí log je. V grafu 3.3 je zobrazen počet bloků pro dané délky. Z obrázku můžeme vidět, že délky bloků jsou velmi nevybalancované. Například počet bloků s délkou 13 je 96 317, oproti tomu počet bloků s délkou 16 jsou pouze dva.

Počet logů	11 175 629
Počet různých šablon	48
Celkový počet bloků	575,061
Počet anomálních bloků	16 838

Tabulka 3.1: Statistiky o HDFS1 datech.

HDFS2 data jsou v porovnání s HDFS1 daty mnohem větší, obsahují 71 118 073, které už ale nemají u bloků označení, jestli se jedná o anomální či normální blok.

[1] Dostupné na: `https://zenodo.org/record/3227177#.YoV01ahBxD8`

3.2 Drain3

Pro parsování dat využijeme novou verzi Drain [7], kterou je Drain3. Drain využívá derivační strom fixní délky. Pomocí Drainu zpracujeme HDFS1 a získáme z logů jejich šablony, každé šabloně se přiřadí **Drain identikátor**. Drain identikátory nabývají hodnot $id \in \{1, 2, \ldots, n\}$, kde n je počet různých šablon. Šablona prvního zpracovaného logu má identifikátor 1, pro každý další log s nově zpracovanou šablonou je tato hodnota o 1 vyšší než pro předchozí. V tabulce 3.2 jsou zapsány příklady šablon HDSF1 logů s jejich Drain identifikátory a počtem výskytů. V grafu 3.2 můžeme vidět počet výskytů každé z šablon ze všech HDFS1 logů. Šablony s nižším identifikátorem mají většinou vyšší zastoupení. To je dáno tím, že u šablon s vyšším zastoupením je vyšší pravděpodobnost, že Drain zpracuje log s touto šablonou dříve.

Z uvedených hodnot je zřejmé, že v HDFS1 datech jsou i šablony velmi nevybalancované.

id	Šablona logu	Výskyt
1	Received <*> src: <*> dest: <*>	1 723 232
2	BLOCK* NameSystem.allocateBlock: <*> <*>	575 061
47	PacketResponder <*> 1 Exception ...: The stream is closed	2
48	Exception in receiveBlock for block <*> ...: Broken pipe	3

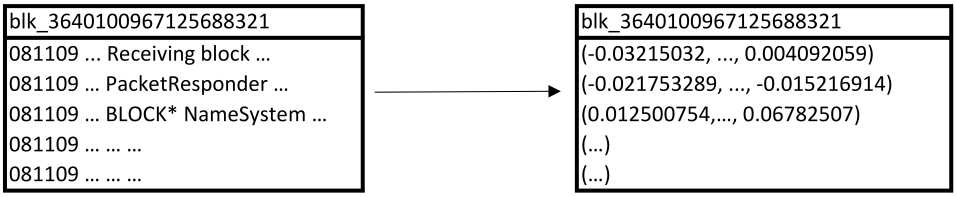
Tabulka 3.2: Ukázka šablon logů v HDFS1 datech i s jejich Drain identifikátorem a počtem výskytů v datech.

3.3 Číselné reprezentace dat

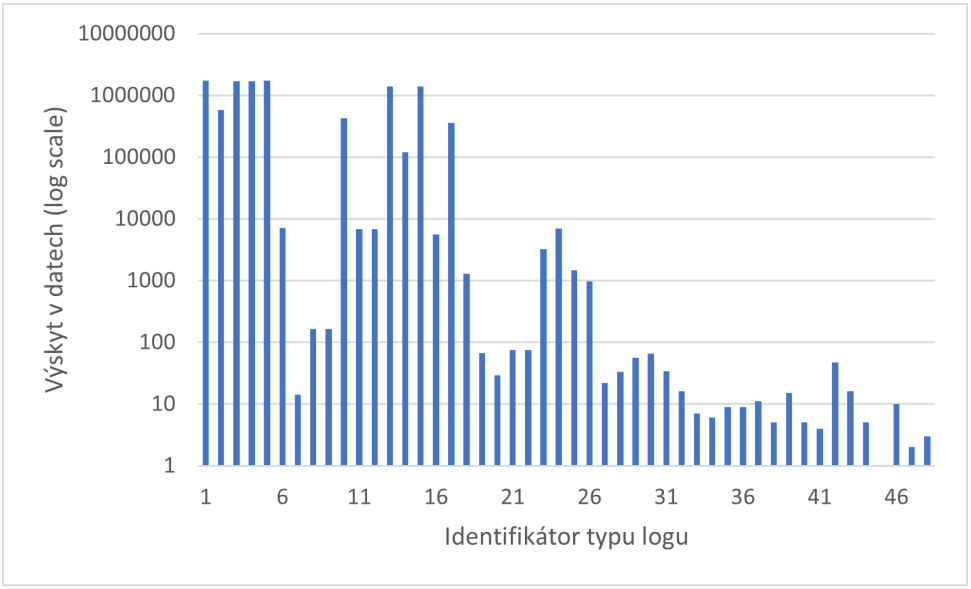
V této sekci představíme metody číselné reprezentace zpracovaných dat, které využijeme v experimentech. Těmito metodami jsou bag-of-words a fastText

3.3.1 Bag-of-words

Prvním způsobem číselné reprezentace dat je bag-of-words[21]. Touto metodou se číselným vektorem reprezentujeme bloky HDFS1 dat. Vytvoříme vektor délky rovný počtu různých šablon logů. Blok se reprezentuje tím způsobem, že na i-tém místě vektoru je hodnota rovna počtu výskytů logu se šablonou, jejíž Drain identifikátor je i. Jako jednoduchý příklad uvedeme případ, kdy se blok skládá z 10 logů, jejichž šablona je *Received <*> src: <*> dest: <*>*, tak je tento blok reprezentovaný jako $(10, 0, 0, 0, \ldots, 0)$.



Obrázek 3.1: Příklad reprezentace bloku logů pomocí fastTextu.



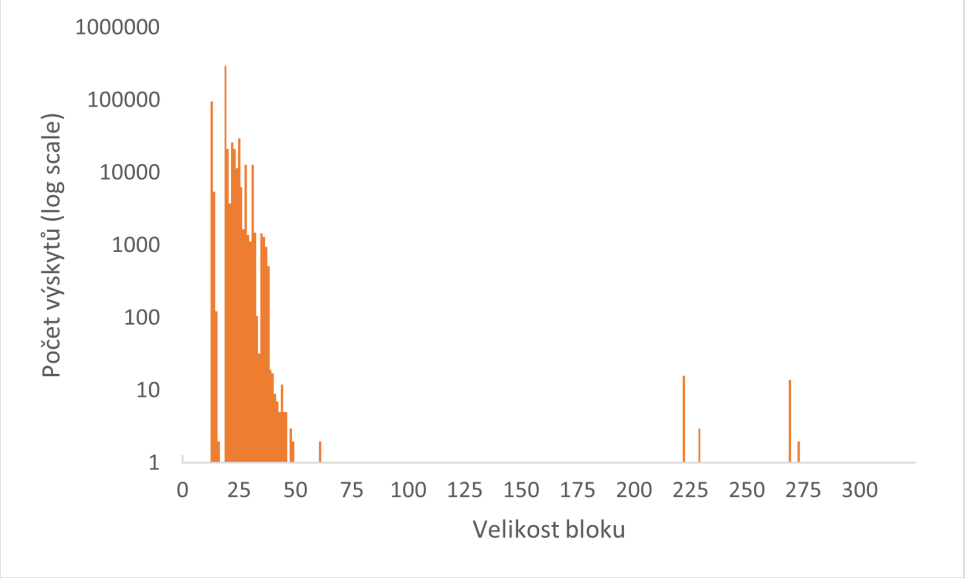
Obrázek 3.2: Histogram výskytu šablon v HDFS1 datech s logaritmickou stupnicí.

3.3.2 FastText

Další metodu číselné reprezentace dat, kterou využijeme, je fastText [10, 3]. FastText je metoda číselné reprezentace slov, která je trénovaná i na n-gramech slov, pomocí kterých můžeme zpracovávat i slova, která jsme dosud neviděli. FastTextem reprezentujeme věty číselným vektorem tak, že každý vektor reprezentující slovo ve větě vydělíme jeho eukleidovskou normou a tyto vektory zprůměrujeme. Stejným způsobem jako věty reprezentujeme logy.

3.4 Střední kvadratická chyba a ADAM

Pro vyhodnocení výsledků použitých modelů během procesu trénování je nutné mít možnost změřit odchylku mezi skutečným výstupem modelu a výstupem očekávaným. Díky zjištění této odchylky je dále možné v procesu



Obrázek 3.3: Histogram výskytu délky bloků v HDFS1 datech s logaritmickou stupnicí.

učení modelu postupovat tak, aby se rozdíl mezi skutečnými a očekávanými hodnotami s každým krokem snižoval. Pro měření odchylky je použita ztrátová funkce (loss function), kterých existuje určité množství druhů v závislosti na struktuře výstupu modelu. Například modely, jejichž výstupem jsou diskrétní hodnoty (například při úloze Natural Language Inference - NLI), budou využívat odlišný typ ztráty funkce než modely, jejichž výstup je spojitý (například při regresních úlohách). Mezi nejčastěji využívané ztrátové funkce patří například křížová entropie [22], logaritmická ztráta [17], exponenciální ztráta [19] a střední kvadratická chyba (MSE) [19]. Pro potřebu v naší úloze jsme při trénování modelů použili právě funkci MSE, také nazývanou L2 regularizace.

Jedná se o výpočet čtverce rozdílu mezi aktuálním výstupem modelu "pred" a očekávaným výstupem "true" dělený počtem výstupů. Funkce MSE je velmi citlivá na odlehlé (outlier) hodnoty, protože rozdíl je umocněn na druhou, funkce tudíž dává odlehlým hodnotám větší význam. MSE lze vyjádřit podle vzorce 3.1.

Jako optimalizační metoda je vybrána Adam[11].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Opravdová\_hodnota_i - Predikovaná\_hodnota_i)^2 \quad (3.1)$$

Chování funkce jako kvadratické křivky je užitečné zejména pro algoritmus gradientního sestupu, kde je gradient menší v blízkosti minima. Funkce MSE je také užitečná, pokud jsou pro problém důležité odlehlé hodnoty.

3.5 Metriky

Pro modely vyhodnotíme **precision** a **recall**. Obě hodnoty jsou v uzavřeném intervalu mezi 0 a 1. Recall nám určuje počet všech modelem správně identifikovaných anomálních bloků v poměru k počtu všech bloků v testovacích datech, které jsou anomální. Precision je počet všech správně identifikovaných anomálních bloků v poměru k počtu všech bloků, které model označil jako anomální. Recall a precision jsou na sobě závislé, obecně se při maximalizaci jedné z nich často druhá sníží, proto je jako hlavní metrika zavedené **F1-score**, které zohledňuje obě hodnoty, vypočtené podle vzorce 3.2.

$$F1\text{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.2}$$

Kapitola 4

Příprava na experimenty

V této kapitole si rozebereme přípravu na experimenty, popíšeme si použité modely a úpravu dat.

4.1 Vanilla autoencoder

Jako první model si uvedeme Vanilla autoencoder, který byl představen v práci [13].

4.1.1 Úprava číselných reprezentací

Tento model získá na vstupu číselné vektory velikosti počtu různých šablon, kde každý blok logů je reprezentovaný jako bag-of-words, které je popsáno v 3.3.1, tento číselný vektor se dále upraví pomocí množiny vah TF-IDF[1].

4.1.2 Autoenkodér

Autoenkodér se skládá ze tří hlavních částí enkodéru, bottlenecku a dekodéru. Enkodér i dekodér se skládají z určitého počtu vrstev. Vrstvy enkodéru postupně zkomprimují vstup, který se pak dostane do bottlenecku, dekodér se poté snaží ze zkomprimovaného vstupu v bottlenecku zreprodukovat počáteční vstup. Autoenkodér se používá na detekci anomálií tím způsobem, že se předpokládá, že naučený autoenkodér bude dávat vyšší ztrátové hodnoty pro anomální data než pro normální. To z důvodu, že dekodér nebude naučený dostatečně dobře zreprodukovat hodnoty anomálních dat.

4.1.3 Popis modelu

Ve vanilla autoencoder je za každou vrstvou enkodéru, dekodéru i bottlenecku ReLU aktivační funkce 4.1, která kladné hodnoty nechá nezměněné a záporné změní na 0, za kterou následuje dropout[16] vrstva. Dropout vrstva pomáhá překonávat přeučení (overfitting) modelu tím, že se při trénování náhodně nepoužívají některé neurony modelu.

Tento model je trénován semi-supervizovaným učením pouze na normálních blocích. Pomocí validačních dat poté určíme práh (threshold) tím, že pro každá

anomální data z validačních dat určí na natrénovaném modelu jejich ztrátu. Jako výsledný limit se vybere taková hodnota ztráty, která na validačních datech maximalizuje F1-score. Práh je hodnota, pomocí které naučený model při testování určí, jestli je testovaný blok anomální nebo normální. Naučený model určí pro každý blok jeho ztrátu. Pokud hodnota ztráty je vyšší než práh, model tento blok určí jako anomální, pokud menší, naopak ho určí jako normální.

$$R(x) = max(0, x) \tag{4.1}$$

4.1.4 Hyperparametry

Hyperparametr je parametr, jehož hodnota je určená před začátkem tréninku, během něhož se nemění. Tyto hodnoty pak ovlivňují učení. V našem případě model získá předem dané hyperparametry, které jsou vygenerovány náhodně.

V 1 je příklad hyperparametrů, které model získal. Na tomto příkladě pracuje model s batch velikostí 8. Jeho Dropout je 0,069, což pro každý neuron určuje pravděpodobnost toho, že nebude použit. Model je učen ve 3 epochách, dimenze vstupu je rovna dimenzi vektoru, do které jsou data transformována, což je v tomto případě 41. Vstup má dimenzi 41, protože se jedná o vektorovou reprezentací bloku pomocí bag-of-words. Délka vstupu 41 je na upravených HDFS1 datech, kde změníme 8 různých typů logů na log *others*, takže v datech zůstane 40 nezakrytých typů logů a logy typu *others*. Počet neuronů ve vrstvách enkodéru jsou v tomto případě v pořadí 152, 143 a 99. Dimenze bottlenecku je 17. Počet neuronů dekodéru je v pořadí 52, 162, 171 a 200. Koeficient učení je 0,22.

```
{
    "hyperparameters": {
        "batch_size": 8,
        "dropout": 0.06905397960484189,
        "epochs": 3,
        "input_dim": 41,
        "layers": [152, 143, 99, 17, 52, 162, 171, 200],
        "learning_rate": 0.22022019499873735
    }
}
```

Listing 1: Příklad náhodně vybraných hyperparametrů pro Vanilla encoder model.

4.2 AECNN1D

Dalším modelem si uvedeme AECNN1D, který využívá konvoluční vrstvy.

4.2.1 Úprava číselných reprezentací

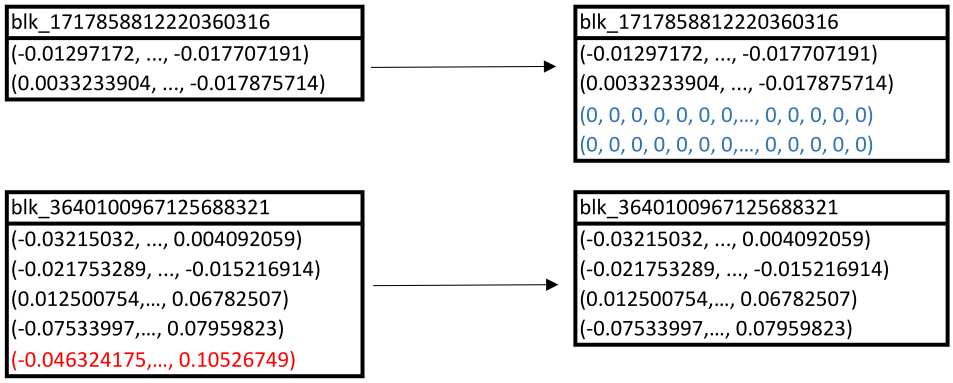
U tohoto modelu jsou data reprezentována pomocí fastTextu uvedeného v sekci 3.3.2. Každý blok HDFS1 dat je po použití fastTextu blok číselných vektorů, kde každý řádek reprezentuje jeden log daného bloku.

Time delta

Každý číselný vektor reprezentující log má první prvek určující hodnotu time delty, která určuje čas mezi dvěma po sobě jdoucími logy v bloku. Na toto číslo je poté použit desítkový logaritmus, aby méně vyčnívaly logy, které mají mezi sebou dlouhý interval.

Oříznutí polí

Problémem modelu AECNN1D je, že bloky vektorů mají rozdílnou velikost. AECNN1D využívá pytorch[1], který nepodporuje batch, kde bloky mají rozdílnou velikost. proto je nutné bloky upravit. Jako řešení je vybráno oříznutí bloků, každý blok se upraví tím způsobem, aby měly stejnou velikost. Před učením tuto velikost určíme. Všechny bloky, jejichž počet řádků je menší, se na určenou velikost rozšíří řádky nul. Naopak pokud je velikost bloku vyšší, tak se dolní řádky odříznou. Příklad je na obrázku 4.1



Obrázek 4.1: Ukázka oříznutí bloků na velikost 4.

4.2.2 Popis modelu

Model se skládá ze tří hlavních částí, kterými jsou enkodér, dekodér a mezi ně zapojený bottleneck. Enkodér je postaven na 1D konvolučních blocích s předem daným kernelem. Každý blok opakuje 1D konvoluční vrstvu, následovánou aktivační ReLU funkcí 4.1, za kterou je max-pooling.

[1] `https://pytorch.org/`

První konvoluční vrstva funguje tím způsobem, že přes řádky bloku vstupu se použije konvoluční filter[12], který vytvoří *feature mapy* s dimenzí vstupu sníženou o předem určenou hodnotou kernelu. ReLU aktivační funkce pak všechny záporné hodnoty ve feature mapách přepíše na 0. Následný max-pooling sníží dimenzi každé feature mapy na polovinu tím způsobem, že postupně porovnává dvojice prvků a ve feature mapě nechá prvek s vyšší hodnotou.

Další konvoluční bloky fungují stejným způsobem, kdy ten samý postup se provede na feature mapě vygenerované v předchozím konvolučním bloku.

Mezi enkodérem a dekódérem je úplně spojený bottleneck, tedy vrstva propojená s enkodérem i dekodérem všemi svými neurony.

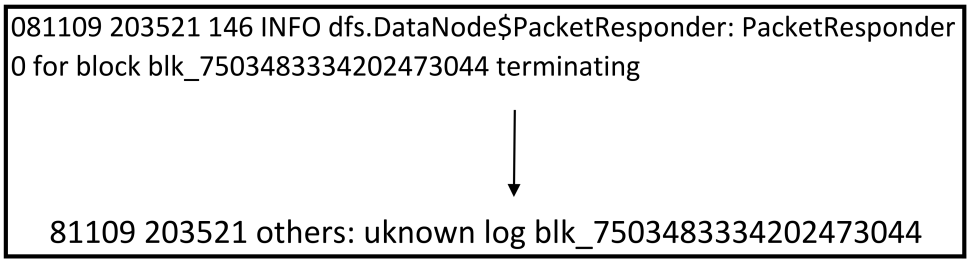
Dekódér obsahuje předem určený počet 1D transponovaných konvolučních bloků. Každý blok opakuje 1D transponovanou konvoluční vrstvu, aktivační funkci ReLU a upsampling vrstvu používající algoritmus nejbližších sousedů[2].

Transponované konvoluční vrstvy mají inverzní operace ke konvolučním vrstvám. Každá transponovaná vrstva zvýší dimenzi feature mapy v závislosti na předem určeném kernelu, který může mít jiné hodnoty než kernel k enkodéru. Následná ReLU vrstva opět přemění záporné hodnoty na 0 a upsampling vrstva dimenzi feature map zdvojnásobí.

4.2.3 Hyperparametry

I v tomto případě model získá hyperparametry, které jsou vygenerovány náhodně. V 2 je příklad hyperparametrů, které model získal. Na tomto příkladě pracuje model s batch velikostí 128, bottleneck dimenzí 462, velikost kernelu na dekodéru je 5 a na enkodéru 7, model je učen ve 4 epochách, dimenze vstupu je rovna dimenzi vektoru, kterým jsou logy reprezentovány, což je v našem případě 101. Vstup má dimenzi 101, protože vektorová dimenze reprezentací fastTextem byla nastavena na 100 a jeden prvek pak vyjadřuje time deltu. Počet vygenerovaných feature map u konvoluční vrstvy v enkodéru je 65 a 355, u dekodéru je to 193 a nakonec 101 na výstupu. Koeficient učení je 0,1 a velikost oříznutých bloků je nastavena na 43.

[2] `https://pytorch.org/docs/stable/generated/torch.nn.Upsample.html`



Obrázek 4.2: Ukázka logu s šablonou *PacketResponder <*> for block <*> <*>* změněného na log s šablonou *others: unknown <*>*.

```
{
    "hyperparameters": {
        "batch_size": 128,
        "bottleneck_dim": 462,
        "decoder_kernel_size": 5,
        "encoder_kernel_size": 7,
        "epochs": 4,
        "input_shape": 101,
        "layers": [65, 355, 193],
        "learning_rate": 0.1,
        "window": 43
  }
}
```

Listing 2: Příklad náhodně vybraných hyperparametrů pro AECNN1D model.

4.3 Příprava dat

V našich experimentech dáváme modelům upravená HDFS1 data. Jak už jsme zmiňovali v 3.1, HDFS1 data se skládají ze 48 různých šablon. Abychom otestovali a porovnali robustnost modelů, je potřeba, abychom vybrali určitý počet typů logů, jehož všechny logy v trénovacích datech zakryjeme a tento naučený model pak budeme testovat na datech, kde se nacházejí logy se všemi 48 typy logů. Je důležité, aby oba modely dostaly stejná trénovací, validační i testovací data. V této práci zhodnotíme experimenty pro 8, 19, 29 a 40 zakrytých logů. Vybereme 10 různých permutací pro 8 zakrytých logů a výsledky poté zprůměrujeme. Náhodným generátorem s nastaveným semínkem (seed), aby se data dala reprodukovat, vybereme 8 různých čísel mezi 1 a 48. Poté v datech zakryjeme všechny logy, jejichž Drain identifikátor šablony je jedno z vybraných čísel. Pro 19, 29 a 40 zakrytých logů je postup úplně stejný.

Logy jsou zakryty dvěma způsoby. V prvním se logy vybraných šablon vymažou z trénovacích dat úplně. Druhý způsob zakrytí je vyobrazen na obrázku 2, tyto logy jsou nahrazeny logem s šablonou *others*. Ukázka změněného logu s vybranou šablonou je na obrázku 4.2

Data jsou poté rozdělena na trénovací, validační a testovací data, kde testovací data obsahují 10 % všech bloků, zbytek dat je poté rozdělen do trénovacích a validačních dat v poměru 9:1. Trénovací data se použijí k učení modelu, validační data se využijí k evaluaci modelu při jeho učení. Nakonec se testovacími daty model vyhodnotí.

Na odkazu [3] jsou dostupné zdrojové kódy využité pro zpracování dat a experimenty.

[3] `https://gitlab.fel.cvut.cz/hubalmar/bakalarska-prace`

Kapitola 5
Evaluation

5.1 Vanilla autoencoder

Tato kapitola se zabývá experimenty a vyhodnocením modelů. Oba modely učíme pro 100 různých sad hyperparametrů. Pomocí validačních dat se poté vybere sada hyperparametrů s nejlepším F1-score, na modelu s těmito hyperparametry se poté spustí testovací data, která nám model vyhodnotí.

5.1.1 Změněné logy

Vanilla enkodér jsme nejdříve změřili na způsob, kde se vstupním datům změní všechny vybrané typy logů na *others*. To znamená, že se velikost vektoru bag-of-words sníží o daný počet zakrytých šablon. Následně se dimenze zvýší o jedna kvůli nové šabloně *others*. Například tedy s 48 rozdílnými logovými typy a 8 zakrytými logy má vstupní vektor délku 41.

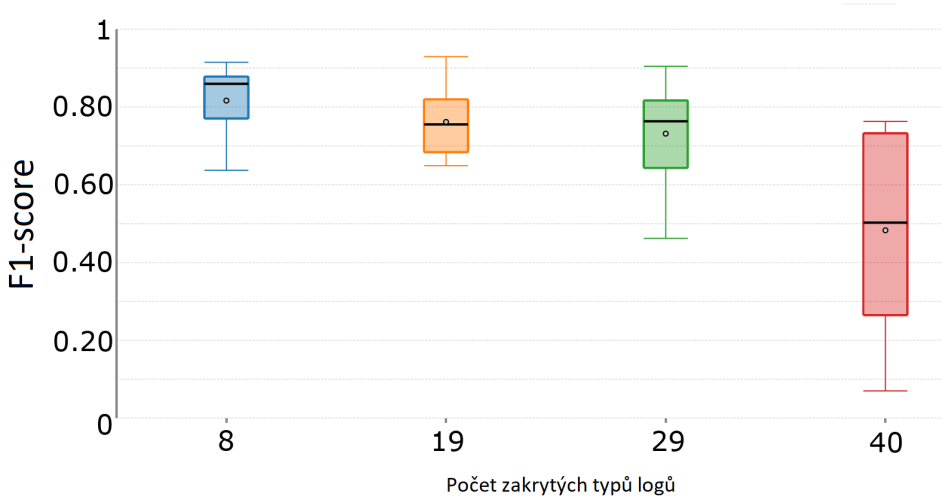
V A.1 můžeme vidět výsledné hodnoty F1-score měření pro 8, 19, 29 a 40 zakrytých logů, kde pro každý počet zakrytých logů je vybráno 10 různých permutací zakrytých logů, pro které se model vyhodnocoval. Na 5.1 je vyobrazen krabicový graf (boxplot), který nám umožní přehledně vidět například průměr, medián a první a třetí kvartil, přesná čísla jsou uvedena v 5.1.

Z výsledků je patrné, že se hodnoty pro různé permutace zakrytých logů velmi liší, pro 8 zakrytých typů logů je 8 z 10 hodnot F1-score vyšší než 0,8, ale hodnota pro čtvrté vyhodnocení je znatelně menší - 0,694, směrodatná odchylka dává hodnotu 0,0634. Pak v případě s 40 zakrytými typy logů jsou rozdíly pro různé permutace mnohem vyšší, směrodatná odchylka nabývá hodnoty 0,2614. Tyto velké rozdíly jsou způsobeny nevybalancovanými daty, viz 3.2. Kvůli obrovským rozdílům výskytu typů logů může být v extrémním případě pro 8 zakrytých typů logů zakrytých 7 miliónů logů z celkových 11 miliónů, v opačném extrémním případě se může stát, že zakrytých logů bude pouze v řádu stovek.

Z výsledků můžeme dále vidět, že data pro 19 zakrytých logů mají horší medián než pro 29. Lze ale vidět, že pro 29 logů má model menší minimální hodnoty, průměrnou hodnotu a větší směrodatnou odchylku. Lze předpokládat,

Zakrytých typů	Q1	Medián	Q3	Průměr	Sm odchylka
8 logů	0,8242	0,8640	0,8726	0,8320	0,0634
19 logů	0,6448	0,6660	0,7682	0,7139	0,1085
29 logů	0,6138	0,7901	0,8580	0,7042	0,2140
40 logů	0,2657	0,5057	0,7226	0,4786	0,2614

Tabulka 5.1: Přehled charakteristik pro různý počet změněných typů logů Vanilla encoder modelu. Q1 a Q3 označuje první a třetí kvartil. Poslední sloupec jsou směrodatné odchylky.



Obrázek 5.1: Krabicový graf[1](boxplot) pro Vanilla autoencoder s odstraněnými typy logů.

že při testování pro více různých permutací by si model s 19 zakrytými logy vedl lépe.

5.1.2 Odstraněné logy

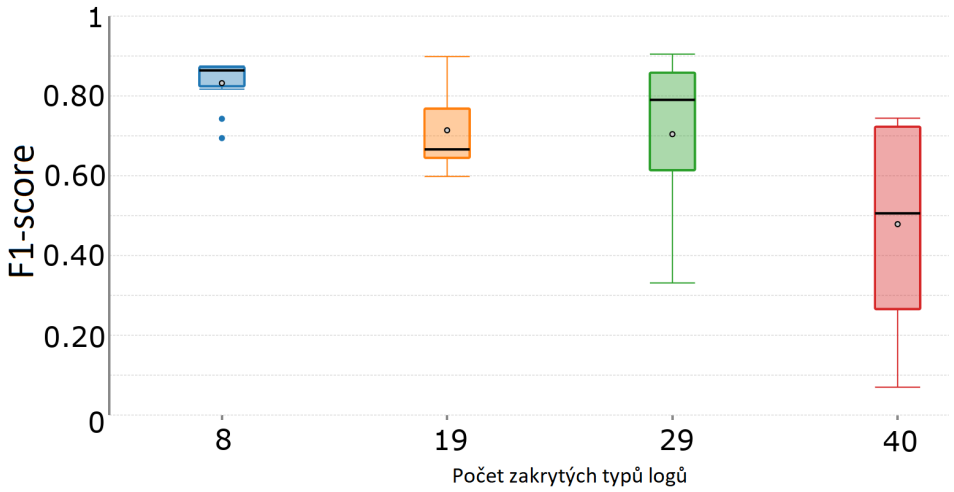
V druhém případě byl model Vanilla encoder měřený na datech, kde byly logy s vybranými typy logů plně odstraněny. Velikost vektoru bag-of-words se tedy změní o daný počet odstraněných typů. Pro 8 odstraněných typů logů je tedy velikost vektoru 40,

V tabulce A.2 jsou vypsány výsledky F1-score experimentů s odstraněnými logy pro 8, 19, 29 a 40 typy logů pro 10 různých permutací. Na obrázku 5.2 je uveden krabicový graf a v tabulce 5.2 jsou tyto hodnoty rozepsané.

[1]Vygenerováno na: `https://goodcalculators.com/box-plot-maker/`

Zakrytých typů	Q1	Medián	Q3	Průměr	Sm. odchylka
8 logů	0,7705	0,8597	0,8781	0,8166	0,0937
19 logů	0,6839	0,7553	0,8195	0,7617	0,0948
29 logů	0,6435	0,7633	0,8168	0,7315	0,1494
40 logů	0,2649	0,5027	0,7325	0,4829	0,2673

Tabulka 5.2: Přehled charakteristik pro různý počet odstraněných typů logů u Vanilla encoder modelu. Q1 a Q3 označuje první a třetí kvartil, poslední sloupec jsou směrodatné odchylky.



Obrázek 5.2: Krabicový graf[2](boxplot) pro Vanilla autoencoder s změněnými typy logů na others.

Z výsledků pro testování s plně odstraněnými logy vidíme, že se hodnoty velmi podobají výsledkům získaných způsobem, kdy jsme dané logy změnili na *others*. Pro 8 odstraněných logů je průměrná hodnota F1-score o malý rozdíl horší, pak pro 19, 29 i 40 jsou dokonce nepatrně lepší.

5.2 AECNN1D

Model AECNN1D jsme měřili na způsob, že se z trénovacích a validačních dat odstraní všechny vybrané typy logů, kdežto v testovacích datech budou logy všechny. Model jsme začali trénovat na 8 zakrytých typech logů a otestovali jsme je na testovacích datech. V tabulce 6 jsou zapsány výsledky F1-score, precision a recall prvních šesti permutací 8 zakrytých typů logů i s výsledky na validačních datech. Přes to, že s validačními daty je model natrénován na F1-score okolo hodnoty 0,8, tak na trénovacích datech jsou výsledky

[2]Vygenerováno na: `https://goodcalculators.com/box-plot-maker/`

mnohem horší. Nejjasnějším příkladem jsou výsledky z první permutace, na validačních datech vychází hodnoty F1-score 0,83, na testovacích datech je to 0,05. Můžeme si všimnout, že na testovacích datech vychází precision 0,0293 a recall 1, precision tedy odpovídá poměru počtu anomálních bloků vůči počtu všech bloků v HDFS1 datech, který je 2,93 %. Z toho vyplývá, že model všechny bloky v testovacích datech až na výjimky označuje za anomálie. Z toho usuzujeme, že model na těchto parametrech špatně reaguje na data, na která není naučený a model je přeučen (overfitting), čímž je příliš závislý na trénovacích datech a nezvládá reagovat na data, na která není z trénování zvyklý.

Po změření výsledků pro 8 zakrytých typů logů jsme se rozhodli netestovat pro 19, 29 a 40 typů logů, protože bychom dosáhli podobných výsledků. Následovaly experimenty, kde jsme se pokoušeli dokázat, že je model přeučený a zkoušeli jsme různé změny, kterými by se mohly výsledky zlepšit.

Přeučení jsme se pokusili dokázat přidáním dropout vrstvy do každého konvolučního bloku, dále jsme zkoušeli změnit generování hyperparametrů, kdy jsme nechali určité hyperparametry neměnné a pozorovali, pro které hyperparametry je model schopný lépe reagovat na nová data. Z měření na třech různých sadách dat si modely vedly lépe, pokud je vyšší koeficient učení, což podporuje naši hypotézu, že je model přeučen. Přestože s těmito změnami dosahoval model lepší výsledky, tak se F1-score v testování pohyboval v rozmezí 0,3 až 0,5 F1-score. To je v porovnání s průměrnými hodnotami Vanilla enkoderu mnohem horší.

Poté jsme se ještě pokusili změřit případ, kdy v trénovacích datech odstraníme 16 typů logů, ve validačních 8 a v testovacích datech jsou nechány všechny. Touto změnou se do určité míry naučí lépe reagovat na nová data. Tímto způsobem dosahoval model pro tři různé sady dat hodnoty F1-score 0,54, 0,53 a 0,28. Z našich experimentů tedy vyplývá, že se model nenaučil dobře reagovat na nové typy logů. Jedním z důvodů přeučení mohou být nevyhovující data z toho důvodu, že je pro model k naučení se na logách s novými šablonami 48 různých typů logů příliš málo.

Kapitola 6

Závěr

V této bakalářské práci jsme se zabývali detekcí anomálií nad logy. Základem pro tuto práci bylo nastudování moderních metod detekce anomálií a pochopení funkčnosti modelů Vanilla encoder a AECNN1D. Zabývali jsme porovnáním obou těchto modelů, kde jsme upravili data způsoby, že se odstraní či změní určité typy logů v HDFS1 datech pro ověření robustnosti obou modelů. Následně jsme tato data dále upravili, aby je bylo možné použít k našim experimentům. V experimentech vyhodnocujeme HDFS1 data pro 8, 19, 29 a 40 zakrytých typů logů, kde pro každý počet zakrytých logů jsme náhodným výběrem vybrali 10 různých permutací zakrytých typů logů.

Vanilla autoencoder, který je učen na datech reprezentovaných číselnými vektory pomocí bag-of-words, jsme testovali pro dva různé typy dat. V prvním případě byly logy s vybranou šablonou z dat úplně odstraněny, v druhém případě byly tyto vybrané logy změněny na *others*. Model AECNN1D je učen na datech reprezentovaných číselným vektorem pomocí knihovny fastText. Tento model byl naučen na trénovacích datech, která měla vybrané typy logů v trénovacích a validačních datech odstraněny, kdežto v testovacích datech zůstaly všechny.

Pro Vanilla autoencoder jsme provedli celkově 80 experimentů. Experimenty vycházely podle očekávání. Pro 8 zakrytých typů logů se výsledky výrazně nelišily od hodnot pro neupravené HDFS1 data. Zvyšováním počtu zakrytých logů se výsledky zhoršovaly. Kvůli velké nevybalancovanosti výskytu šablon velmi záleží, které typy logů jsou zakryty. V extrémním případě může být pro 8 zakrytých šablon zakryto 7 miliónů logů z celkových 11 miliónů, v opačném extrémním případě se může stát, že zakrytých logů bude pouze v řádu stovek.

AECNN1D model jsme testovali pro 8 zakrytých typů logů s 10 různými permutacemi vybraných šablon. Po výrazně horších výsledcích jsme zkoušeli upravit model přidáním dropout vrstvy a upravováním hyperparametrů. Po těchto pokusech jsme došli k závěru, že je model přeučen (overfitting) a z toho důvodu špatně reaguje na nové typy logů v testovacích datech. To se nám potvrdilo i tím, že jsme zkusili model učit na HDFS1 datech s 16 odstraněnými typy logů v trénovacích datech a s 8 odstraněnými typy ve validačních. Tím se zvýšila hodnota prahu (threshold) a hodnoty F1-score se zlepšily.

Pokud by se v budoucnu pořídila nová data většího objemu s více rozdílnými

typy logů, jejichž sekvence logů jsou označeny informací, jestli se jedná o anomálie, tak je možno podobný experiment zopakovat.

Příloha A

Tabulky

Model	8 typů logů	19 typů logů	29 typů logů	40 typů logů
1	**0,8749**	**0,8988**	0,9048	0,6819
2	0,8746	0,7772	0,7635	0,3325
3	0,8645	0,6767	**0,8911**	0,2451
4	**0,6942**	0,6140	0,8619	**0,7442**
5	0,8695	**0,5981**	**0,3312**	0,7362
6	0,8736	0,6437	0,8168	0,2294
7	0,8456	0,6553	0,5911	**0,0699**
8	0,8170	0,6481	0,6819	0,7407
9	0,8634	0,8859	0,3529	0,3273
10	0,7427	0,7411	0,8464	0,6789

Tabulka A.1: Výsledné F1-score pro model Vanilla autoencoder se změněnými logy pro 10 různých permutací vybraných šablon.

Model	8 typů logů	19 typů logů	29 typů logů	40 typů logů
1	0,8610	**0,9296**	**0,9048**	0,7078
2	0,8819	0,8800	0,7635	0,3264
3	**0,9150**	0,7035	0,7631	0,2451
4	**0,6940**	0,6774	**0,9048**	0,7442
5	0,7781	0,7515	**0,4622**	**0,7629**
6	0,6373	0,7590	0,8168	0,2294
7	0,8583	**0,6491**	0,7330	**0,0699**
8	0,9054	0,6635	0,6137	0,7407
9	0,8666	0,8373	0,5369	0,3241
10	0,7679	0,7660	0,8167	0,6789

Tabulka A.2: Výsledné F1-score pro model Vanilla autoencoder s odstraněnými logy pro 10 různých permutací vybraných šablon.

	Validační data			Testovací data		
	Precision	Recall	F1-score	Precision	Recall	F1-score
1	0,8535	0,9267	0,8886	0,0293	1.0000	0,0569
2	0,8746	0,7772	0,7635	0,0293	1.0000	0,0569
3	0,9731	0,6917	0,8086	0,0293	1.0000	0,0569
4	0,9392	0,8877	0,9127	0,0295	1.0000	0,0572
5	0,9601	0,9221	0,9407	0,0294	1.0000	0,0571
6	0,9312	0,8224	0,8734	0,1376	0,9329	0,2398
7	0,8619	0,8739	0,8679	0,1247	0,9507	0,2204
8	0,9506	0,6864	0,7972	0,0305	1.0000	0,0592
9	0,8333	0,9108	0,8703	0,0293	1.0000	0,0569
10	0,8909	0,8409	0,8651	0,0286	0,9732	0,0556

Tabulka A.3: Výsledné údaje pro model AECNN1D pro 10 různých permutací vybraných šablon.

Příloha B

Literatura

[1] Almeida, F., and Xexéo, G. Word embeddings: A survey, 01 2019.

[2] Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR abs/1803.01271* (2018).

[3] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics 5* (06 2017), 135–146.

[4] Du, M., and Li, F. Spell: Streaming parsing of system event logs. *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), 859–864.

[5] Du, M., Li, F., Zheng, G., and Srikumar, V. 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), pp. 1285–1298.

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

[7] He, P., Zhu, J., Zheng, Z., and Lyu, M. R. Drain: An online log parsing approach with fixed depth tree. *2017 IEEE International Conference on Web Services (ICWS)* (2017), 33–40.

[8] He, S., Zhu, J., He, P., and Lyu, M. R. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)* (2016), pp. 207–218.

[9] He, S., Zhu, J., He, P., and Lyu, M. R. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)* (2016), pp. 207–218.

[10] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* (Valencia, Spain, Apr. 2017), Association for Computational Linguistics, pp. 427–431.

[11] Kingma, D., and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations* (12 2014).

[12] Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D. J. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing 151* (2021), 107398.

[13] Koryt'ák, M. Anomaly detection methods for log files. *Masther's thesis, Czech Technical University in Prague* (2021).

[14] Makanju, A. A., Zincir-Heywood, A. N., and Milios, E. E. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2009), KDD '09, Association for Computing Machinery, p. 1255–1264.

[15] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space, 2013.

[16] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 56 (2014), 1929–1958.

[17] Vovk, V. The fundamental nature of the log loss function.

[18] Wang, J., Tang, Y., He, S., Zhao, C., Sharma, P. K., Alfarraj, O., and Tolba, A. Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things. *Sensors 20*, 9 (2020).

[19] Wyner, A. On boosting and the exponential loss.

[20] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, Association for Computing Machinery, p. 117–132.

[21] Zhang, Y., Jin, R., and Zhou, Z.-H. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics 1*, 1 (Dec 2010), 43–52.

[22] Zhang, Z., and Sabuncu, M. R. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Proceedings of the 32nd International Conference on Neural Information Processing*

Systems (Red Hook, NY, USA, 2018), NIPS'18, Curran Associates Inc., p. 8792–8802.

[23] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., and Lyu, M. R. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (2019), ICSE-SEIP '19, IEEE Press, p. 121–130.