

TDK-dolgozat

Muzsai Lajos

Transformer alapú robusztus módszerek eseménynaplók alapján történő hibakeresésre

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR



Szerző:

Muzsai Lajos

autonómrendszer-informatikus MSc

1. évfolyam

Témavezető:

Lukács András

Matematikai Intézet, Számítógéptudományi Tanszék

Budapest, 2023

Tartalomjegyzék

1. Bevezetés	3
2. Anomália keresés naplóállományokban	5
2.1. Modellek	5
2.2. Logelemző módszerek	7
2.3. Fejlettebb anomália detektáló modellek	8
3. A UnifiedLog rendszer	10
3.1. UnifiedLog komponensei	10
3.2. Tanítás során használt adatok	11
3.2.1. Naplózási struktúrák	12
3.2.2. Adatforrások mérete	12
3.2.3. Címkézések	14
3.2.4. Naplósor hosszak	15
4. Kísérletek a UnifiedLog rendszerrel	19
4.1. Több adathalmazos tanítás és általánosítás	19
4.1.1. Legjobb teljesítményű modell	21
4.1.2. Logadathalmazok kihagyása a tanításból	22
4.1.3. Tanítóadat méretének hatásai	24
4.1.4. Előtanított kódolóval adathalmaz specifikus detektor	25
4.2. Ablációs mérések	27
5. Diszkusszió	29
5.1. Összefoglalás	29
5.2. A módszer korlátai	29
5.3. Fejlesztési lehetőségek	30
Köszönetnyilvánítás	32

Irodalomjegyzék	32
Ábrajegyzék	36
Táblázatjegyzék	37

1. fejezet

Bevezetés

A számítógépes naplók értékes információforrást jelentenek a rendszergazdák és mérnökök számára, mivel betekintést nyújtanak a számítástechnikai rendszerek teljesítményébe és viselkedésébe. A modern rendszerek által generált naplók óriási mennyisége azonban megnehezítheti a problémák azonosítását és diagnosztizálását. Az anomáliák észlelése a számítógépes naplókban, a normálistól eltérő minták és viselkedések azonosításának folyamata kiemelt feladat a számítógépes rendszerek megfelelő működése és biztonsága szempontjából. A számítógépes naplókban található anomáliák észlelésének problémáját széles körben tanulmányozták a számítástechnika és mérnöki munka területén. Különböző technikákat javasoltak, például szabályalapú, statisztikai és gépi tanuláson alapuló módszereket. Az elérhető megoldások azonban gyakran jelentős emberi erőfeszítést és szakértelmet igényelnek a hatékony alkalmazáshoz. Továbbá korlátozzuk, hogy a módszerek rossz hatásfokkal általánosítanak olyan naplókra, amelyek más forrásból származnak, mint amilyenre kitalálták őket.

Dolgozatunk célja egy újszerű megközelítés kialakítása volt a számítógépes naplók anomáliáinak észleléséről. A legtöbb kutatás ezen a területen egyszerre egy adathalmazra koncentrált, amelyre egy konkrét modellt tanítanak be. Amikor több adatkészleten értékelnek egy modell típust, akkor mindegyikre külön-külön tanítják a modellt, általában még az előfeldolgozási lépések is különböznek a különböző adatkészleteken. A dolgozatban bemutatjuk, hogy a gépi tanulás transformer architektúrái felhasználhatók több különböző forrásból származó eseménynapló egységes robusztus reprezentációjának megtanulására. Ehhez kialakítunk egy egységes adat-előfeldolgozási módszert. Majd igazoljuk, hogy az így létrehozott UnifiedLog nevű megoldásunk megtanulja a számítógépes naplók széles körének robusztus ábrázolását, és így képes több adathalmaz anomáliáinak egyidejű ész-

lelésére.

A UnifiedLog modelljeit a Loghub projekt különböző adatforrásai alapján tanítjuk és értékeltük ki [1]. A Loghub 16 adatkészletből áll, amelyek 6 különböző kategóriájú naplót tartalmaznak (elosztott rendszerek, szuperszámítógépek, operációs rendszerek, mobil rendszerek, szerveralkalmazások, önálló szoftverek), amelyek közül 5 adatkészletet címkézett.

A dolgozat legfontosabb eredménye, hogy megmutajtuk, hogy a reprezentáció előállítás során az egyes adathalmazokból lehet általánosítani más adathalmazokra is. Ezekkel a robusztos reprezentációkkal elérhető, hogy egyes log adathalmazokon közel tökéletes F1 értéket érjen el az anomáliák detektálása, még olyan esetben is, amikor a modell tanítás alatt az adott adathalmazt nem látta. Megmutatjuk, hogy érdemes a detektor modellt egyidejűleg több adathalmazon is tanítani. A UnifiedLog képes egyidejűleg a Loghub 5 címkézett adathalmazán átlagosan 0.86 F1 értékeket elérni az anomália detektálás feladatán.

A dolgozat felépítése a következő. A 2. fejezetben röviden bemutatjuk az eddigi megoldások közül azokat, amelyek a legnagyobb hatással voltak a loganomália detektálás területére és rávilágítunk e módszerek azon hiányosságaira, amelyek motiváltak, hogy egy általános modellt építsünk. A 3. fejezetben bemutatjuk a UnifiedLog keretrendszert: az adat előfeldolgozást, a robusztus reprezentációgyártást transformer kódolóval és a több adathalmazon működő transformer alapú anomália detektort. Továbbá ebben a fejezetben mutatjuk be részletesen a Loghubon található adatokat. A 4. fejezetben megvizsgáljuk a keretrendszer különböző komponenseit és megmutatjuk, azok hogyan befolyásolják az általánosító és predikciós képességét. Megmérjük a keretrendszer teljesítményét többféle tanító adathalmaz összeállításon, külön vizsgálva a reprezentációkat előállító és az anomália detektálást végző részeket.

A UnifiedLog forráskódja elérhető a GitHubon [2].

2. fejezet

Anomália keresés naplóállományokban

Ebben a fejezetben össze foglalkozunk az eseménynaplókban történő anomáliadetektálás területén elért eddigi eredményeket. A dolgozat sokat merít az előző megoldásokból, miközben azoknak a hiányosságait javítja. A terület áttekintésében sokat segítettek a [3, 4, 5] survey cikkek.

2.1. Modellek

A 2.1. ábrán látható transformer architektúra forradalmi megközelítést hozott a szekvenciális feladatok terén [6]. A transformer egy attention mechanizmust alkalmazó hálózat, amely a korábbi RNN (Recurrent Neural Network) és LSTM (Long Short Term Memory) modelleknél jóval hatékonyabb és skálázhatóbb megoldást nyújtott [7] [8]. A transformer fontos eleme a self-attention mechanizmus és az, hogy hatékonyan párhuzamosítható. A transformer architektúra magába foglalhat kódoló-dekóder szerkezetet, amely ismétlődő blokkokból áll és tartalmazza a self-attention mechanizmust. Az attention mechanizmus tanulható súlyokat rendel a bemenetekhez, ezáltal a modell képes lesz megtanulni, hogy egy bemenetnek melyek a fontos részei.

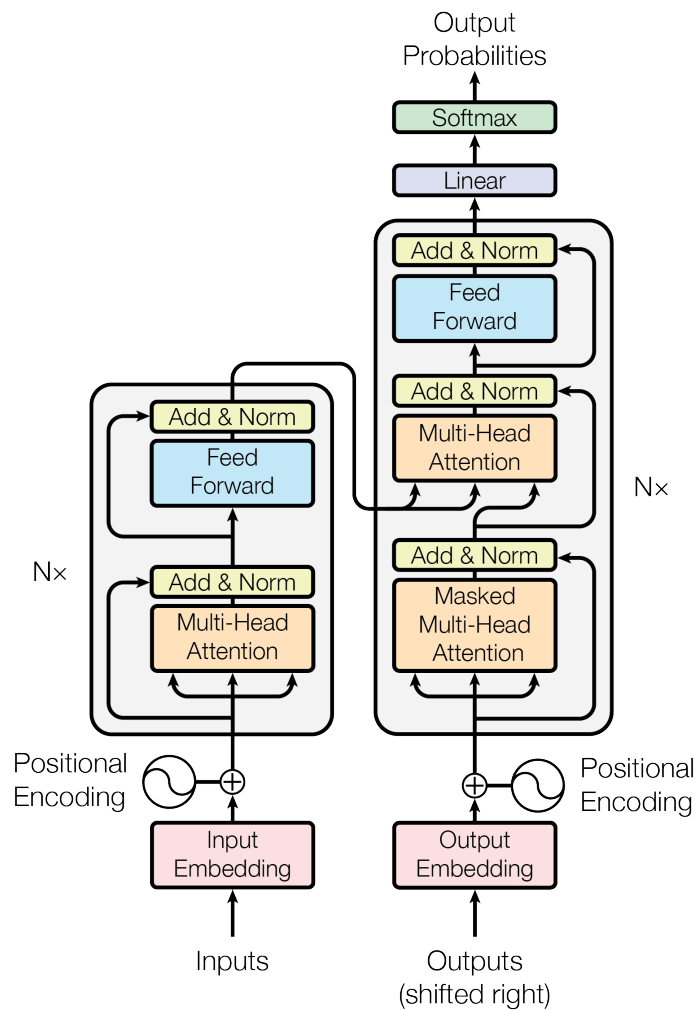
Az attention mechanizmus skálázott skaláris szorzat formájában van definiálva:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

ahol

- Q a lekérdezés mátrixa (query),
- K a kulcs mátrixa (key),
- V az érték mátrixa (value),
- d_k a kulcs dimenziója.

A két legelterjedtebben alkalmazott figyelmi mechanizmus a hozzáadási és szorzási típusok közé tartozik. A fent bemutatott képlet egy szorzási típusú figyelmi mechanizmust reprezentál, amely egy $\sqrt{d_k}$ faktort alkalmaz.



2.1. ábra. A transformer architektúrája [6]

A transformer architektúrák közül a BERT modell az egyik legelterjedtebb, a UnifiedLog is ebből építkezik. A BERT modell egy transformer kódoló architektúrán alapuló nagy nyelvi modell [9]. A modellt 20 különböző nyelven tanították. A modellben bevezetik a [CLS] tokent, amit minden szöveg elejére, és a [SEP] tokent, amit minden

szöveg végére szűrnak be. A szerzők a [CLS] token reprezentációját felhasználva képesek egy szöveges bemenethez egy összegző vektorreprezentációt rendelni.

A transformer modellek inputját tokenek sorozatával adjuk meg. Fontos eleme a modellek kialakításának a bemenet tokenekre bontása. A Wordpiece tokenizációt először a BERT modell használta [9, 10, 11]. A Wordpiece egy szabályalapú iteratív algoritmus, ami egy szövegkorpuszból képes egy előre meghatározott mennyiségű tokenszótárt létrehozni. Ez a tokenszótár egy és több karakteres tokenekből áll. Az algoritmus először listát készít azokról a karakterekről, amelyek a tanító szövegében szerepelnek, ezekből tokeneket készít és hozzáadja őket a tokenszótárhoz. Ezután az egymás mellett leggyakrabban előforduló tokenekből készít egy összevont tokenet és hozzáadja a tokenszótárhoz. Ezt a lépést addig ismétli az algoritmus, amíg el nem éri a kívánt szótárméretet.

Maszkolt nyelvi modellezés. A transformer modellek tanításában meghatározóak a self-supervised módszerek. A maszkolt nyelvi modellezés egy olyan öntanulási módszer, amely lehetővé teszi a modelleknek, hogy a szövegek szintaktikáját és szemantikáját tükröző reprezentációt hozzon létre. A módszer úgy működik, hogy a modellnek egy olyan szekvenciából, ahol véletlenszerűen elmaszkolunk, azaz kicseréljük tokeneket egy speciális [MASK] tokenre, meg kell tanulnia prediktálni, hogy mi volt az eredeti szekvencia. Ez a módszer azért hatékony, mert ahhoz, hogy a modell sikeresen prediktálni tudja, mi volt a maszk mögött, ahhoz meg kell tanulnia a szekvenciákban lévő komplex összefüggéseket.

2.2. Logelemző módszerek

Drain. A Drain egy log parsing módszer [12], tehát a strukturálatlan logokat strukturált formába rendezi. A Drain online módon képes működni és a használatához nem szükséges a felhasználónak különböző paramétereket megadni. A működési elve egy irányított körmentes gráfon alapul, aminek a részei: előfeldolgozás, hossz szint, token szint, hasonlósági szint, log csoport és a kimeneti szint. Ennek a log parsing módszernek a hátránya, hogy nem képes hatékonyan általánosítani újszerű naplókra.

DeepLog. A DeepLog modell a naplósorok reprezentációjához a sorokban található állandó szavaknak és a változóknak két külön vektort használ [13]. Egy LSTM architektúrát használnak arra, hogy egy időben előrejelző modellt tanítsanak be [7]. Az előrejelző modell egy toplistát ad arról, hogy mit tart legvalószínűbb következő állandó szavakat reprezentáló vektornak. Amennyiben a toplistában benne van a valóban következő, nor-

málisnak jelölik meg a naplósort, egyébként anomáliának. Ezt a módszert az általános felhasználhatóságban a logsor reprezentációs megközelítése akadályozza, újszerű naplókra nem képes hatékonyan működni.

LogAnomaly. A LogAnomaly framework egy természetes nyelvfeldolgozáson alapuló módszer anomáliák detektálására eseménynaplókban [14]. A cikk legfontosabb újítása a `template2vec` módszer, ami a logok szemantikai jelentését képes reprezentálni vektorokkal. A `template2vec` szinonimákat és ellentétes jelentésű szavakat használ naplósablonok definiálására. Így képes egy n méretű one-hot vektorral reprezentálni a naplósorokat. Az előállított naplósablonok sorozatát úgy használják anomáliadetektálásra, hogy egy időben előrejelző modellt tanítanak be hasonlóan a DeepLog módszeréhez. A `template2vec` módszert az általánosításban az korlátozza, hogy előzetes tudást használnak a szinonimák és ellentétes jelentésű szavak definiálására. Ezért az olyan naplósorokra, amikben új fogalmakat vannak nem képes hatékony megoldást nyújtani.

2.3. Fejlettebb anomália detektáló modellek

Logsy. A Logsy volt az első módszer, ami a transformer architektúrát alkalmazta [15]. A transformert ebben a modellben arra használják, hogy reprezentációkat állítsanak elő naplósorokból. Módszerük általánosító képességét úgy javították, hogy kiegészítő log adathalmazokkal is tanították a transformert. Ennek a célja itt még csak az volt, hogy megelőzzék a túltanulást, viszont a dolgozatban inspirációként használtuk ezt a megközelítést, mert ez az eredmény abba az irányba kezdett mutatni, hogy egyidejűleg több adatforrás használata javítja a modellek robusztusságát. Hasonlóan a DeepLog és LogAnomaly-hoz egy előrejelző megközelítést alkalmaznak a klasszifikációhoz.

NeuralLog. A NeuralLog modell volt a legnagyobb befolyással a dolgozatban bemutatott módszerre [16]. Ez az első olyan módszer, ami nem alkalmaz log parsingot a naplósorok reprezentálására. A log parsing általában úgy működik, hogy a változó részeket kiveszik a log sorokból és az így kapott lehetséges logokat tekintik egy-egy eseménynek. A cikkben bemutatott új megközelítés lényege, hogy egy alap többnyelvű BERT modellt használnak a naplókban található kulcsszavakra, és a kulcsszavak reprezentációinak az átlagával reprezentálnak egy naplósort [9, 6]. A NeuralLog az a kisebb transformer modell, amit felügyelt módon használnak anomáliadetektálásra a reprezentációk sorozatán. A dolgozatban bemutatott detektor architektúrája megegyezik az ebben a cikkben használt NeuralLog modellel.

TransLog. A TransLog egy általánosítás felé mutató transformer alapú módszer [17]. A modell tanítását két fázisba szervezik: az elsőben egy adathalmazon előtanítanak, majd a másodikban egy cél-adathalmazon finomhangolják a modellt. A cikkben megmutatják, hogy az előtanítás alatt olyan információkhoz jut a modell, ami egy eltérő cél-adathalmazon gyorsabb és hatékonyabb tanulást eredményez.

SemParser. A SemParser egy log parsing módszer amit hatékonyan fel lehet használni sokféle adathalmazokon [18]. Fontos újítása, hogy az új szavakat egy karakterszintű LSTM modellel klasszifikálja jelentés osztályokba, ezzel hatékony megoldást adva új szavak problémájára. Továbbá CNN és LSTM hálókkal szavak közti összefüggéseket is képes detektálni a logokban.

Szemantikai információ felhasználása. Az irodalomban előforduló módszereket két csoportba lehet sorolni a naplósorok reprezentációja terén: egyrészt, amelyek felhasználnak szemantikus információt és másrészt, amelyek nem. A 2.1. táblázatban különböző gépi tanulási architektúrák teljesítménye látható a BGL, a Loghub egyik annotált adathalmazán. A táblázat bal oldalán ezeknek az architektúráknak a teljesítménye látszik szemantikai információk felhasználása nélkül, míg a jobb oldalon a szemantikai információ felhasználásával. Látszik, hogy a szemantikai információ felhasználása javítja (vagy legalább befolyásolja) minden architektúra teljesítményét. Ez a megfigyelés inspirálta a dolgozatunkban teljesen szemantikára épülő reprezentáció előállítás megközelítését.

Modell	A BGL adathalmazon mért teljesítmények					
	szematika nélkül			szemantika felhasználásával		
	Precízió	Recall	F1	Precízió	Recall	F1
LSTM [14, 13]	0.935	0.989	0.961	0.946	0.989	0.967
Transformer [15]	0.935	0.977	0.956	0.917	1.0	0.957
Autoencoder [19]	0.791	0.773	0.782	0.942	0.92	0.931
Attn. BiLSTM [20]	0.989	0.977	0.983	0.989	0.977	0.983
CNN [21]	0.966	0.977	0.972	1.0	0.977	0.989

2.1. táblázat. A mélytanuláson alapuló módszerek teljesítménye szemantikai információ használatával, illetve használata nélkül [4].

3. fejezet

A UnifiedLog rendszer

Ebben a részben ismertetjük a UnifiedLog rendszer felépítését és részletezzük különböző komponenseit. Bemutatjuk a tanításhoz használható legnagyobb publikusan elérhető eseménynapló-adatbázist a Loghub projektet [1].

3.1. UnifiedLog komponensei

Az eseménynaplókban történő anomália detektálásra használt adatfeldolgozó pipeline a 3.1. ábrán látható. Első lépésként a különböző forrásokból érkező naplókat előfeldolgozzuk azzal a céllal, hogy a következő lépésben fel tudjuk használni őket egy transformer alapú kódolóval. A második lépésben a kódolót betanítjuk arra, hogy képes legyen egy logsort egy vektorral reprezentálni, majd ezzel a módszerrel elkódoljuk a logadatokat. A harmadik lépésben egy transformer alapú detektor modellt tanítunk és használunk anomália detektálásra.

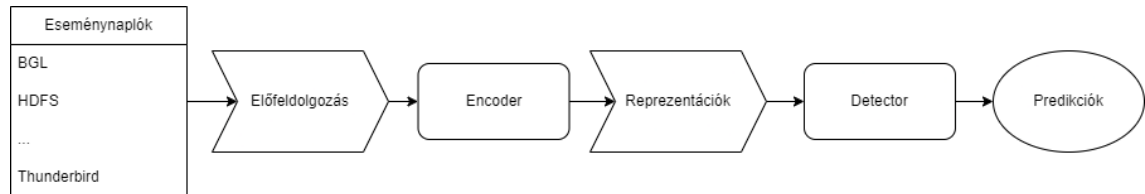
Előfeldolgozás. Az előfeldolgozás célja, hogy a gépi tanulási módszerek által használható formátumba konvertálja a természetes nyelvvel írt naplókat. Ehhez három fő lépést használunk:

1. A nem ascii karakterektől megtisztítjuk a naplókat.
2. A naplókban található számokat egy különleges "[NUM]" tokenra cseréljük.
3. Tokenizáljuk a naplókat a Wordpiece algoritmussal.

Transformer kódoló. A kódoló architektúráját a BERT modell inspirálta, tehát egy többszintű transformer kódoló. A naplósorokhoz tartozó reprezentációgyártásnál hasonlóan a BERT-hez, azt a megközelítést alkalmazzuk, hogy egy [CLS] tokent minden nap-

lósor elejére, egy [SEP] tokenet minden naplósor végébe beszúrunk, és a [CLS] token reprezentációjával ábrázoljuk az egész logsort. Az x-transformers¹ könyvtárat használtuk a megvalósításakor.

Transformer detektor. A detektor architektúráját a NeuralLog [16] modell inspirálta, tehát egy egyszintű kétirányú transformer bináris klasszifikációra használva.

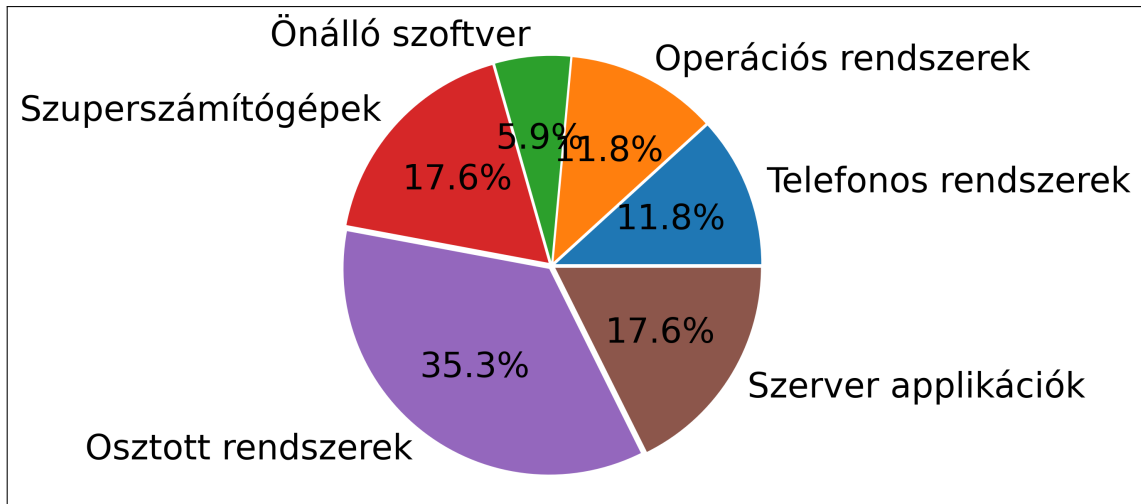


3.1. ábra. A UnifiedLog rendszer eseménynaplóval történő anomália detektálásra használt pipelineja.

3.2. Tanítás során használt adatok

Kísérleteink során a legnagyobb, publikusan elérhető számítógépes naplóadatgyűjteményt használtunk, a Loghub projektet [1]. A Loghub összesen 16 különböző adatforrásból áll, amelyek közül 5 címkézett. A projektben hat különböző adatkategória található: elosztott rendszerek, szuperszámítógépek, mobil rendszerek, szerveralkalmazások, operációs rendszerek és önálló szoftvernaplók. Az 3.3. ábra mutatja a naplókategóriák eloszlását; az önálló szoftverek kivételével mindegyik kategóriának legalább 2 különböző naplóforrása van, ahol az elosztott rendszerek a legváltozatosabbak, 6 különböző adatkészlettel.

¹<https://github.com/lucidrains/x-transformers>



3.2. ábra. Loghub naplókategóriák eloszlása.

3.2.1. Naplózási struktúrák

A Loghub 16 különböző eseménynaplózási rendszer naplóját foglalja össze. Ezek a különböző rendszerek két módon strukturálják a naplósorokat:

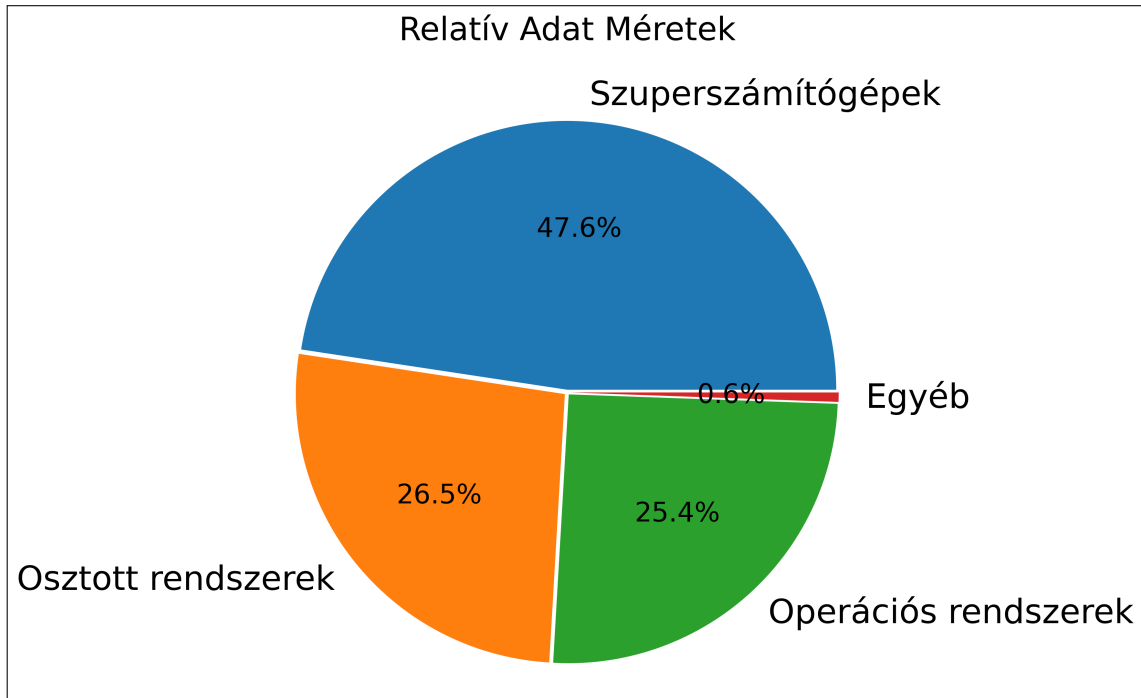
1. **Blokkokban.** Ez a naplózási módszer akkor fordul elő amikor a rendszer, amiről a naplók készülnek, több kisebb részre bontható fel és ezeknek a részeknek külön fájlokban vannak a naplói. Az osztott rendszerek jellemzően ilyen formában tárolják a naplókat.
2. **Időrendben.** Ez a naplózási módszer a gyakoribb, mivel ez a módszer egy helyre gyűjt minden naplósort időrendbe rendezve. A mi módszerünket tekintve ez a naplózási forma az ideális.

Habár minden blokkosított rendszer különbözik abban, hogy mit tekint egy blokknak, vagy egy ilyen blokkhoz mennyi napló tartozhat, megoldásunk egységesen tudja a blokkokat kezelni. A blokkokat különválogattuk és különálló egységként tekintünk minden blokkra, ahol a blokkon belüli naplósort időrendi sorrendben használjuk predikcióra. Ezt a megközelítést a HDFS és Hadoop adathalmazokon korábbi megoldások is alkalmazták, például a LogAnomaly és a NeuralLog.

3.2.2. Adatforrások mérete

Az adattípusok eloszlása gazdagnak tekinthető, de ha az egyes kategóriákban található sorok számát vizsgáljuk, akkor láthatjuk, hogy 3 kategóriában található az összes adatfor-

ráshoz tartozó sorok több mint 99%-a. A 3.3. ábrán egyértelműen látszik, hogy a napló-sorok majdnem fele szuperszámítógépekből származik, míg a másik fele elosztott rendszerekből és operációs rendszerekből. Ennek az egyensúlytalanságnak az oka az, hogy van három adatkészlet, amelyek nagy mennyiségű naplósorral rendelkeznek: a Thunderbird szuperszámítógép naplója több mint 200 millió sorra, a HDFS elosztott rendszer adatkészlete több mint 70 millió sorra, vagy a Windows operációs rendszer adatkészlete több mint 100 millió sorra.



3.3. ábra. Loghub naplósorok eloszlása

Ez az egyensúlytalanság kezelhető azzal, ha csak egy részét tartjuk meg e három nagy adatkészletnek. Mivel a modell tanításában fontos, hogy a felhasznált adattípusok között egyensúly legyen, ezért célszerű a különböző adatforrásokból származó sorok számát egy közös nagyságrendre hozni.

A Thunderbird adathalmaznál az irodalomban az az elfogadott módszer, hogy csak az első 10 millió sort használják fel belőle [3, 16]. Hasonló megközelítést használtuk mi is. Mi 5 millió sorban határoztuk meg bármelyik adathalmazból származó sorok számát. Ez a vágás mindig az idő mentén történik, az időben első 5 millió sort tartottuk meg. Az osztott rendszereknél a blokkstruktúra miatt nem lehetséges az idő mentén vágni, ezért annyi blokkot használtunk fel ezekből a rendszerekből, hogy az összesített sorok száma 5 millió legyen.

3.2.3. Címkézések

A rendelkezésre álló 16 különböző adatkészletből csupán 5 rendelkezik címkézéssel, és ez az 5 adathalmaz vagy osztott rendszerekhez vagy szuperszámítógépekhez tartozik. A legnagyobb kihívás a címkézés következetességével kapcsolatos:

- A Thunderbird és a BGL szuperszámítógépek naplói úgy vannak címkézve, hogy minden naplósorhoz hozzárendelnek egy címkét, amely megjelöli, hogy az adott naplósor rendellenes-e vagy sem.
- A HDFS naplói blokkonként vannak címkézve, ami azt jelenti, hogy a sorok mindig egy blokk művelethez tartoznak, és ebben a naplóban csak a blokkokat címkézik normálisnak vagy rendellenesnek.
- A Hadoop adatkészlet hasonló struktúrával rendelkezik, de itt külön mappában tárolt alkalmazások vannak címkézve, amelyekhez néhány száz sor napló tartozik, és ezek vannak közös címkékkel ellátva.
- Az Openstack osztott rendszer más módon van címkézve. Ebben az adathalmazban három logfájl található: az egyik teljesen rendellenesnek van címkézve, míg a másik két fájl teljesen normálisnak. Ez a kevésbé részletes címkézés okozta a legnagyobb kihívást, mert a több tízezer sornyi abnormálisnak címkézett logban vannak olyan részek, amelyek valójában normális működést írnak le, és ezek részek rontják a detektor modell működését.

A címkézéseket minden adatforrásnál olyan formátumra hoztuk, mint a BGL és Thunderbird esetén, tehát minden sorhoz rendelünk egy 0 vagy 1 értéket annak függvényében, hogy hibás-e az a sor:

- A HDFS esetében azt a megoldást választottuk, hogy blokkokat sorokra bontjuk és minden sorhoz azt a címkét rendeljük amit az a blokk kapott.
- A Hadoop esetén is hasonló módon járunk el, azzal a különbséggel, hogy itt alkalmazások vannak címkézve, amik már eleve külön mappákba vannak válogatva. Ezekhez az alkalmazásokhoz túl kevés mennyiségű naplósor is tartozhat, ahhoz, hogy fel lehessen használni. Amennyiben kevesebb, mint 20 sor tartozik egy alkalmazáshoz azt eldobjuk. Ilyen alkalmazásból 15 db van.
- Az Openstack esetében a három logtípus esetében úgy járunk el, hogy annál a résznél, ami normálisnak van címkézve, minden sort normálisnak címkézünk be. Azoknál a részeknél, amik abnormálisnak vannak címkézve, minden sort abnormálisnak címkézünk.

Címkeeloszlási különbségek. Az anomáliadetektálás jellemző kihívása, hogy a normálistól eltérő adatpontokból jelentősen kevesebb van, mint a normálisból. A mi esetünkben a címkézett adatoknak 1%-ban találunk anomáliának címkézett naplósort. Ez alól kivételt jelent az Openstack adathalmaz, ahol az adatok 66%-a van abnormálisnak címkézve.

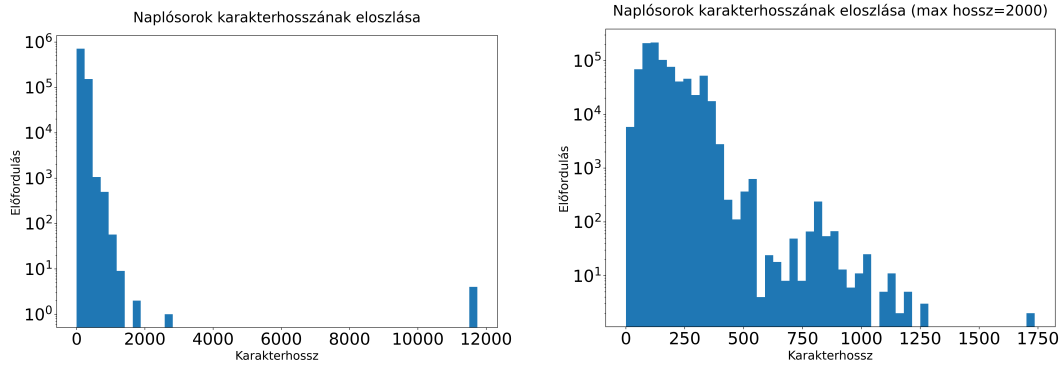
Azt a megoldást választottuk, hogy ezt az egyenlőtlenséget úgy csökkentjük, hogy korlátozzuk a normál működést leíró adatokat a tanító halmazban. Az Openstack adathalmazban található normál-abnormál eloszlást meghagyjuk. Ezzel megelőzzük azt a problémát, hogy a módszerünk mindent normálisnak fog prediktálni, mivel így is alacsony hibát tudna elérni a teljes adathalmazon. Azt az arányszámot ami a normál/abnormál adatmennyiségeket meghatározza, 1:1 arányban találtuk a leghatékonyabbnak.

3.2.4. Naplósor hosszak

Fontos technikai részlet a logsorok tokenizáció utáni hossza. A módszerünk kritikus eleme a kódoló által előállított reprezentációk minősége. Amennyiben egy logsor hosszabb mint a kódoló bemeneti mérete, információt veszünk, mert el kell hagynunk részeket belőle. Ezért kritikus volt meghatároznunk a tokenizáció alatt egy optimális szó-tárméretet és a kódolónak egy olyan bemeneti hosszt ami a legtöbb logsornál hosszabb, viszont elég kicsi ahhoz, hogy gyorsan lehessen futtatni.

A mérések során a Loghub mind a 16 adathalmazából legfeljebb ötvenezer sort vizsgáltunk, ami azt jelenti, hogy összesen 863000 sort. Ezt a méretet reprezentatívnak tekintjük.

Karakterszint. Abban az esetben, ha a karakterek számát vizsgáljuk a 3.4. ábra (a) részén látható, hogy a logsorok döntő része rövidebb mint 2000 sor. A (b) ábrán látható, hogy a 2000 karakternél rövidebb sorok gyakorisága úgy nő, ahogy a karakterek száma csökken.

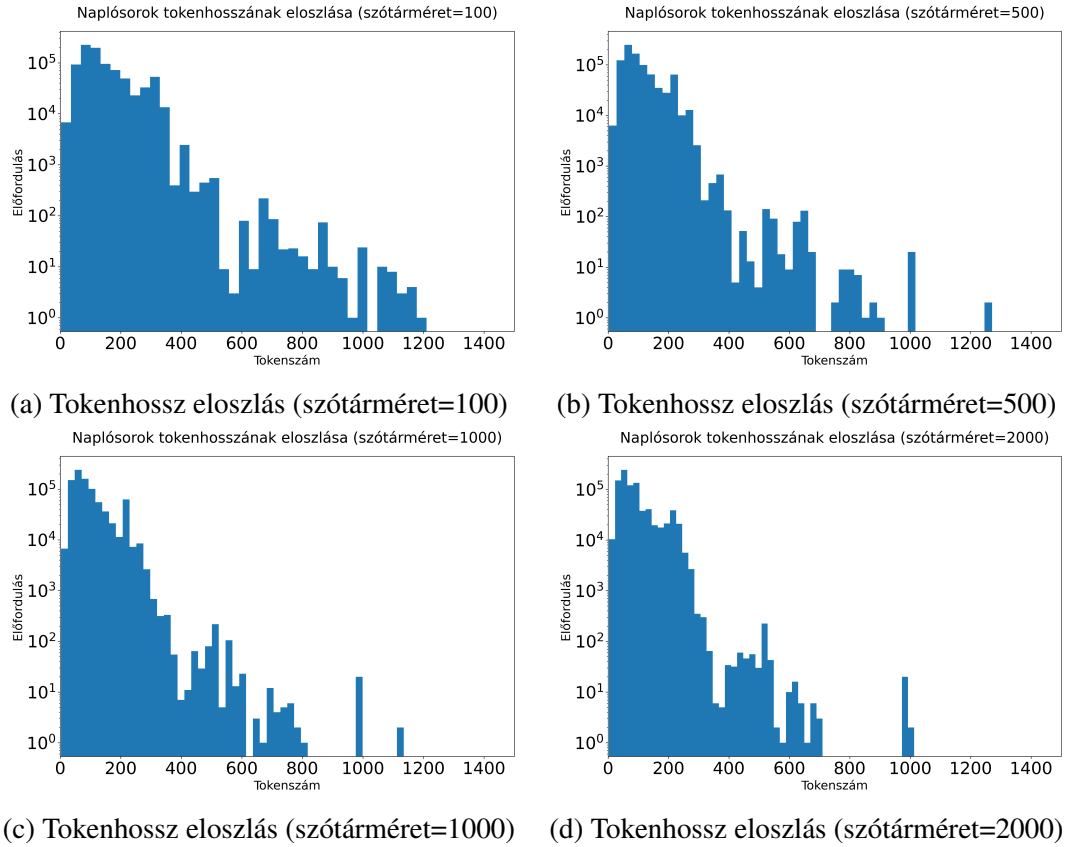


(a) A naplósorok karakterhosszának az teljes eloszlása

(b) A naplósorok karakterhosszának az eloszlása maximum 2000 karakterhosszra szűkítve.

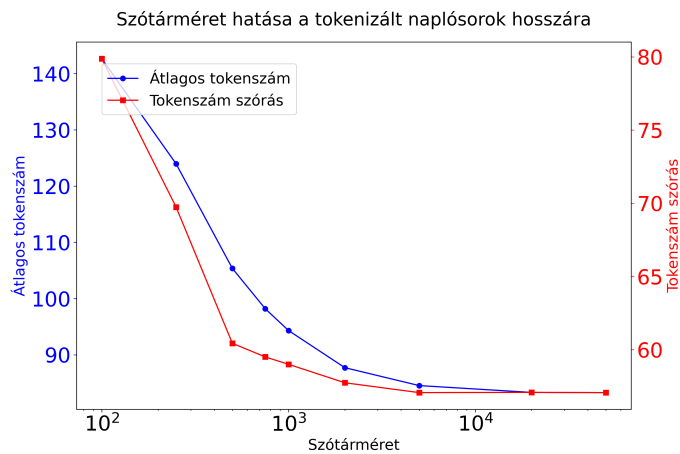
3.4. ábra. A naplósorok karakterhosszának az eloszlása

Tokenszint. A naplósorokat az előfeldolgozás során tokenizáljuk, ezzel csökkentjük azok hosszát és optimalizáljuk a kódolónk teljesítményét. A megfelelő szótárméret megtalálása a tokenizálás alatt fontos feladat, mert a túl kicsi és túl nagy szótárméret gyengébb modellteljesítményt eredményezhetnek. Az optimális szótárméretet a legkisebb olyan méretnek tekintjük, ami képes hatékonyan reprezentálni a logsorokat. A 3.5. ábrán 4 különböző szótármérettel látható a tokenizált sorok hosszainak az eloszlása. Ahogy csökken a szótárméret, úgy csökken a tokenizált sorok átlagos hossza és a szórása, ez a 3.6. ábrán is jól látszik.



3.5. ábra. Naplósorok tokenhosszainak elosztásai különböző szótárméretekkel

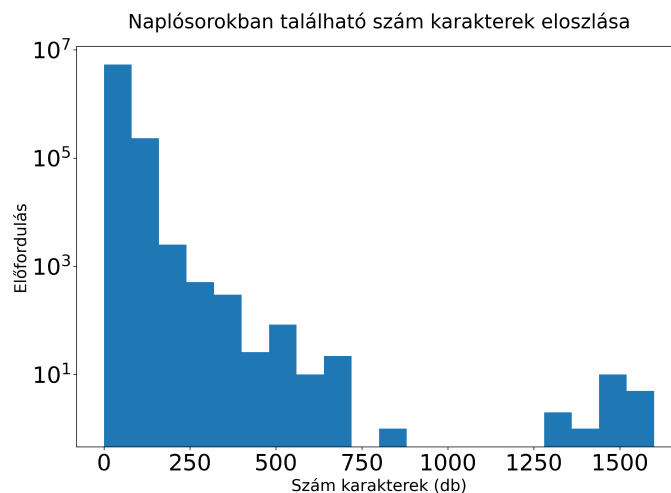
A 3.6. ábrán látható, hogy a tokenizált sorok átlaga és szórása lényegében 5000-es szótárméretig csökken. A teljes csökkenés több mint 80%-a a 100 és 1000-es szótárméretek között történik. Az optimális szótárméretet ezért 1000-nek határoztuk meg. Az 1000-es szótárméretnél az átlagos hossza egy logsornak 94, ezért a kódoló bemeneti hosszát ennél magasabbra, 128-ra állítottuk.



3.6. ábra. A szótárméret hatása a tokenizált sorok átlagos hosszára és szórására.

Speciális karakterek. A tokenizáció során felmerülő kérdés, hogy a nem ASCII karakterek és a számok miként legyenek kódolva. Van négy adathalmaz, amelyekben előfordulnak nem ASCII karakterek. A soroknak a 0,3%-a tartalmaz legalább egy nem ASCII karaktert. Ezek a karakterek sokfélék lehetnek, ezért ronthatják a tokenizáció minőségét azzal, hogy ritkán előforduló tokenekkel növelik a szótárméretet. Erre a problémára azt a megoldást alkalmazzuk, hogy a nem ASCII karaktereket elhagyjuk.

A naplósorokban nagyon gyakoriak a számok, ezek lehetnek időpontok, azonosítók vagy valamilyen változóért felelős értékek. A transformer architektúra nem képes ezeknek a hatékony reprezentálására [22]. Ez a limitációja a transformernek annál jobban megmutatkozik a teljesítményben, minél több helyértékű számokkal kell dolgoznia. A 3.7. ábrán a soronként előforduló szám karakterek darabszámainak eloszlása látható, átlagosan 37.72 szám karakter fordul elő egy naplósorban. Az eseménynaplókban gyakran előforduló számsorozatoknak a problémájára azt a megoldást találtuk, hogy minden számot lecseréltünk egy [NUM] tokenre. Ezzel a megoldással a számok nagyságrendjéről és pozíciójáról minden információt megtartunk, míg az értéküket, ami nehézséget okoz a transformernek eldobtuk.



3.7. ábra. A naplósorokban található szám karakterek számának az eloszlása.

4. fejezet

Kísérletek a UnifiedLog rendszerrel

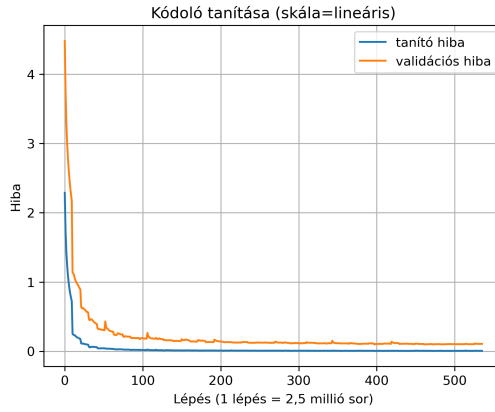
Több száz mérést végeztünk el a UnifiedLog rendszer változatainak vizsgálatára. A kutatási célt, egy olyan eseménynaplókban anomáliákat detektálni képes módszer kialakítását, ami egyidejűleg több különböző adatforráson képes jó eredményt elérni, mérési eredményeink alapján megállapíthatjuk, hogy sikerült elérni.

4.1. Több adathalmazos tanítás és általánosítás

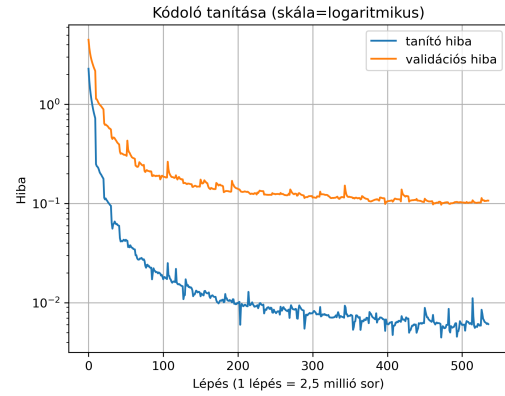
A kódoló és detektor részeknek a tanítását külön egységként kezeljük. A kódoló feladata általános reprezentációk gyártása tetszőleges logsorokhoz, amelyhez a Loghub összes adathalmazát felhasználtuk a tanítás során. A detektor feladata reprezentációk sorozatában történő anomália keresés, amelyhez a Loghub 5 címkézett adathalmazát használtuk tanítóadatnak.

A kódoló tanítása a Google BERT modellje inspirációjával történik [9]. Minden mérésnél 50 epoch hosszan (egy epochnak a Loghub összes adathalmazát együttesen tekintjük) és $3 \cdot 10^{-5}$ tanítási rátával tanítottunk. Az optimalizációs algoritmusnak az AdamW algoritmust használtuk [23]. A batch mérete 4096 volt minden mérés során, így pl abban az esetben ha 4 mélységgel és 6 fejjel 256 dimenziós reprezentációs mérettel tanítottuk a transformer modellnek 72 GB VRAM-ra volt szüksége tanítás alatt.

A 4.1. ábrán mutatjuk be a tanítás során elért tanító és validációs hibákat. Látható, hogy a tanító hiba lényegében 0-hoz konvergál és a validációs hiba sem nő meg, azon a modell nem tanul túl.



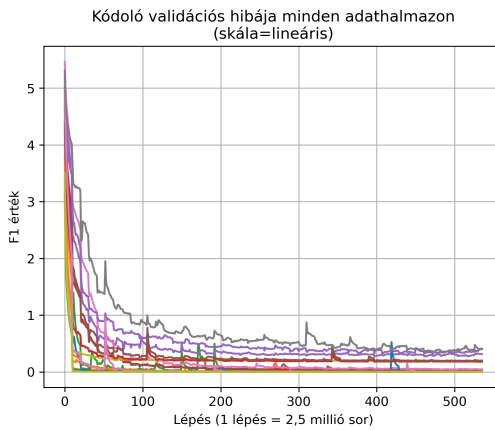
(a) A kódoló tanítása lineáris skálán.



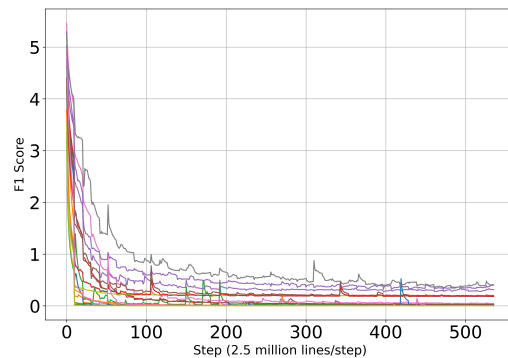
(b) A kódoló tanítása logaritmikus skálán.

4.1. ábra. A kódoló tanítása 50 epochon keresztül.

Általánosítás. Mivel a tanító adat 16 különböző adathalmazból áll ezért megvizsgáltuk, hogy a különböző adathalmazokat tekintve hogyan változik a validációs hiba. A 4.2. ábrán látható, hogy a modell teljesítménye minden adathalmazon javul, tehát nincsenek olyan adathalmazok amelyekre rátanulna. Az is látszik, hogy egyes adathalmazokon jobban teljesít a modell mint másokon. Ez nem kizárólag az adatmenyiség különbségei hatása. Az is hatással van erre a jelenségre, hogy egyes naplók komplexebb struktúrákkal rendelkeznek, mint mások.



(a) A kódoló tanítása lineáris skálán.



(b) A kódoló tanítása logaritmikus skálán.

4.2. ábra. A kódoló tanítása alatti validációs hibák minden adathalmazon 50 epochon keresztül.

Detektor tanítása. A detektor tanítása a NeuralLog modelléhez hasonlóan történik [16]. Minden mérésnél legfeljebb 200-as epoch számmal (egy epochnak a Loghob 5 címkezet adathalmazát tekintjük) és $3 \cdot 10^{-5}$ tanítási rátával tanítottunk. Az optimalizációs algoritmusnak az AdamW algoritmust használtuk. A batch méret 64 volt. Minden mérés-

nél tanulási ráta csökkenést alkalmaztunk 25-ös lépésközzel és 0.9-es gammával. Továbbá minden mérésnél korai leállást alkalmaztunk 3-as toleranciával és 0-ás deltával. Minden mérés alatt a mozgó ablakok módszert használtuk 20-as hosszal és 1-es lépésközzel.

4.1.1. Legjobb teljesítményű modell

A mérések során a legjobb teljesítményt úgy érték el, hogy a kódolót 32 dimenziós reprezentációs mérettel, 6 fejjel és 4-es mélységgel, 1002-es szótármérettel, 128-as input mérettel 50 epochon keresztül 4096-os batch mérettel tanítottuk maszkolt nyelvi modellezéssel. Tanítási adatnak a Loghubon található összes adathalmazt használtuk a Hadoop kivételével, és az egy adathalmazból származó sorokat 5 millióban maximalizáltuk. A maszkolási valószínűségnek 15%-ot, a cserélési valószínűségnek 90%-ot használtunk mint a BERT esetében tették [9]. A tanító, validációs és kiértékelő vágást a nem osztott rendszerek naplóján az idő mentén úgy határoztuk meg, hogy az első 80%-a az adatnak tanító, a következő 10-10%-a az adatnak validációs és teszt adathalmazok lettek. Az osztott rendszereknél pedig véletlenszerűen blokk szinten osztottuk fel az adatokat 80-10-10 százalékos arányban.

A legjobb teljesítményt elérő detektor az előző bekezdésben leírt kódoló reprezentációt használta. A beágyazási dimenzió ennek megfelelően 32 volt. A fejek száma 8. A sűrű réteg dimenzióját 256-ra állítottuk. A model tanítása alatt 20-as ablakméretet és 1-es lépésközt alkalmaztunk. A Loghub 5 címkézett adathalmazán tanítottuk 64-es batch mérettel és $3 \cdot 10^{-5}$ tanítási rátával. Tanítási ráta csökkenést alkalmaztunk 25-ös lépésmérettel és 0.9-es gammával. Továbbá korai leállást használtunk 3-mas toleranciával és 0-ás deltával. Kiesésnek 50%-os valószínűséget használtunk. A normál és abnormális tanító-adatok arányát 1:1-nek határoztuk meg.

A betanított modellünk egyidejűleg képes 5 különböző adathalmazon anomália detektálásra. Az így elért legjobb teljesítményt a 4.1. táblázatban írtuk le. A BGL, Hadoop és Thunderbird adathalmazokon közel tökéletes eredményt értünk el. A HDFS adathalmazon 0.696 F1 értéket. Az Openstack adathalmazon 0.766 F1 értéket. Az így elért átlagos teljesítmény az 5 címkézett adathalmazon **0.864** F1 érték.

Legjobb teljesítmény					
<i>Modell</i>	<i>BGL</i>	<i>Hadoop</i>	<i>HDFS</i>	<i>Openstack</i>	<i>Thunderbird</i>
<i>UnifiedLog</i>	0.99	0.99	0.69	0.76	1.00
<i>NeuralLog</i> [16]	0.98	-	0.98	-	0.96
<i>TransLog</i> [17]	0.98	-	0.99	-	0.99
<i>LogAnomaly</i> [14]	0.96	-	0.96	-	0.68
<i>LogRobust</i> [20]	0.75	-	0.99	-	0.68
<i>Logsy</i> [15]	0.65	-	-	-	0.99

4.1. táblázat. A legjobb teljesítmény a Loghub címkézett adathalmazain.

4.1.2. Logadathalmazok kihagyása a tanításból

Egy fontos módszere annak, hogy a módszerünk általánosító képességét bizonyítsuk az, hogyha megmutatjuk, hogy olyan adattal is működik, amit még nem látott. Ezért azt vizsgáltuk meg, hogy a kódoló (encoder) tanításánál bizonyos log adathalmazok kihagyásával, milyen F1 értéket képes elérni a detektor.

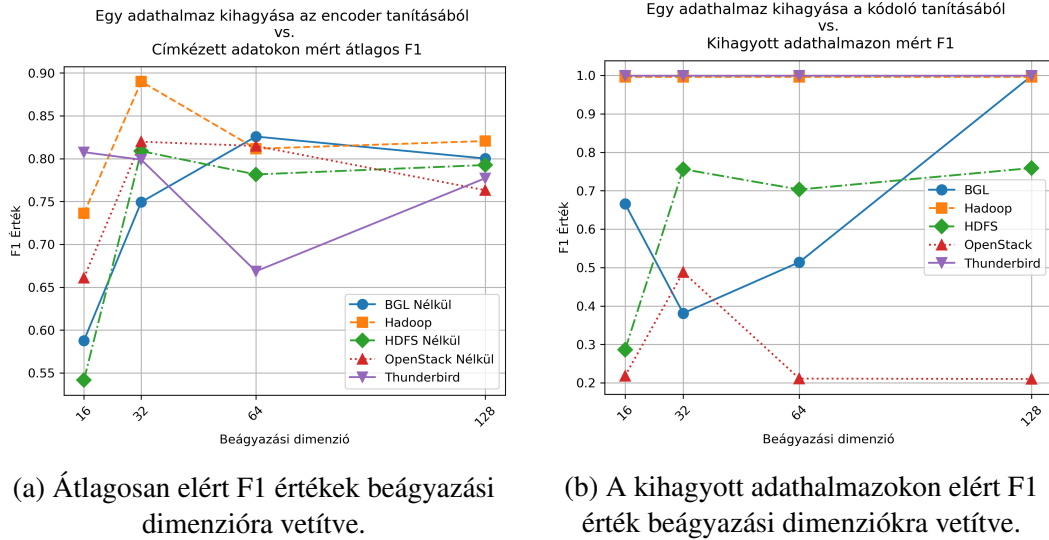
Megfelelő általánosítás esetén a kódolónak az a feladata, hogy tartalmas reprezentációkat legyen képes előállítani lényegében tetszőleges logsorokból. Azaz olyan forrásból jött logsorokat, amin nem volt tanítva a kódoló azt is képes hatékonyan elkódolni. Ez azért lehetséges, mert az eseménynaplók hasonló információkat írnak le különböző rendszerekben és az egyik rendszer eseménynaplóiból képes a modell következtetni egy másik rendszer naplósorainak a logikájára.

Egy tanítóadat kihagyása az encoder tanításából

A 16 elérhető adatforrásból 5 címkézett, ezért megvizsgáltuk, hogy ha a címkézett adathalmazok közül egyet-egyet kiveszünk a kódoló tanítóhalmazából, akkor a végén a detektor milyen eredménnyel képes azon a kihagyott adathalmazon anomáliák detektálására.

Eredmények. A 4.3. ábrán az (a) képen az látható, hogy 32 dimenziós beágyazási dimenzióval már képes a model hatékony teljesítményt elérni. A (b) kép mélyebben magya-

rázza ezt a jelenséget, a nagyobb beágyazási dimenzió segít jobban általánosítani az olyan adathalmazokat is amiket még nem látott. Habár az, hogy a nagyobb beágyazási dimenzió jobb teljesítményt eredményez nem meglepő, az hogy az olyan adathalmazoknál is jobb teljesítményt látunk, ahol nem tanult a kódoló azt sugalja, hogy a többi adathalmazból kinyert információ többletet használja fel ehhez.

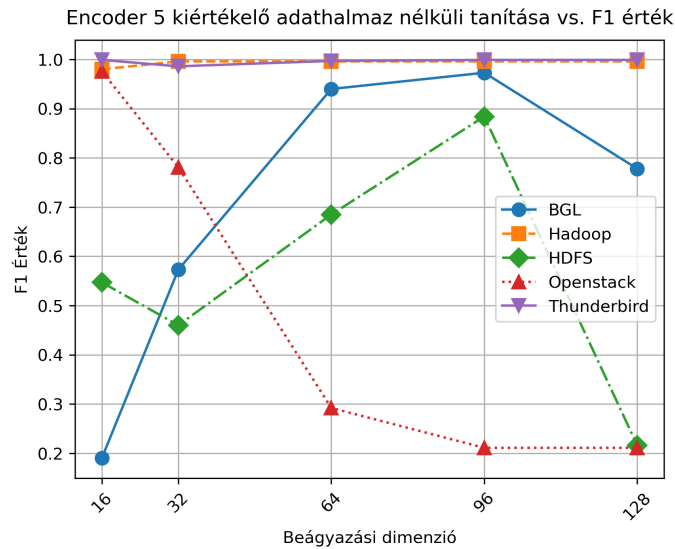


4.3. ábra. Az enkóder tanításából egy kihagyott adathalmazzal vizsgált detektor teljesítmény.

Több tanítóadat kihagyása a kódoló tanításából

Miután láttuk, hogy egy adathalmaz kihagyásával a kódoló képes tartalmas reprezentációkat előállítani, lemértük milyen teljesítményre képes abban az esetben, ha mind az 5 címkézett adathalmazt kihagyjuk ebből a tanításból. Így a kódoló a 11 címkézetlen adaton tanult. A detektort felügyelt tanítással tanítottuk a címkézett adathalmazokon.

Eredmények. A 4.4. ábrán látható, hogy a Thunderbird és Hadoop adathalmazokon beágyazási dimenziótól függetlenül közel tökéletes F1 értékkel képes a modellünk prediktálni. A BGL adathalmazon a 16-os beágyazási dimenzióval nagyon rossz teljesítménytől ($F1=0,2$) a 96-os beágyazásig eljut a modell 0.96 F1 értékig. A HDFS a 16 és 96 beágyazás méreteknél teljesítményben javul, 0.55 F1 értékről közel 0.9 F1 értékig. Érdekes módon a 128 dimenziós beágyazási méretnél mind a BGL és HDFS rosszabbul teljesít mint a 96 dimenzióval. Ebben a mérésben az a legmeglepőbb, hogy az Openstack adathalmazon 0.95 feletti F1 értéket ér el a modell 16-os beágyazási dimenzió mellett és ez az érték dinamikusan csökken nagyobb beágyazási dimenziók mellett.



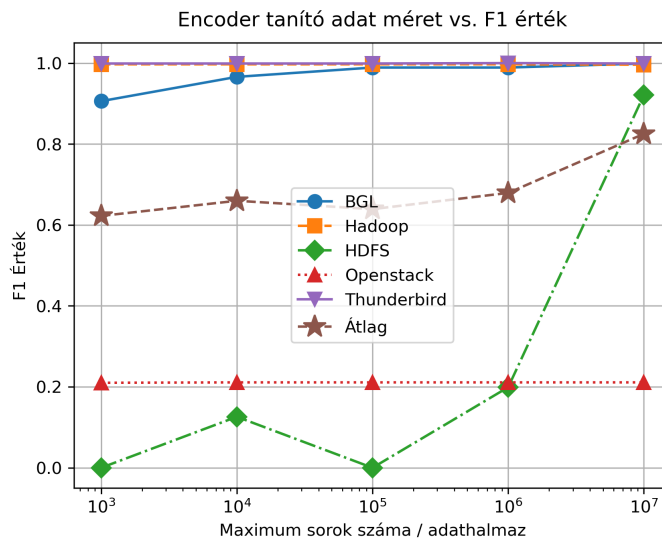
4.4. ábra. Az öt címkézett adathalmaz kihagyása a kódoló tanításából.

4.1.3. Tanítóadat méretének hatásai

A nyelvi modellek teljesítményét jelentős mértékben befolyásolja az, hogy mekkora adatmennyiségen tanultak. A kódoló tanítóadatának a mennyiségbeli változásával azt szeretnénk volna látni, hogy az hogyan befolyásolja a model reprezentációjának minőségét. Ekkor a 16 adatforrásból jövő sorok mennyiségét egyesével maximalizáltuk 1000-tól 5 millióig. Azoknál az adatforrásoknál, ahol időrendben vannak a naplósorok ezeket a sorokat az elejéről vettük, ott ahol blokkonként ott véletlenszerű blokkokat vettünk addig, amíg el nem értük a küszöbértéket, majd annál a blokknál választottuk szét az adatot az idő mentén.

Eredmények. A tanítóadat méretének növelése egyértelműen javítja a modell teljesítményét. Úgy tűnik, hogy különböző adathalmazokon különböző mértékben járul hozzá a teljesítményhez az adatmennyiség. A Thunderbird és Hadoop adathalmazokat már 1000 naplósorból képes a kódoló úgy reprezentálni, hogy az anomáliákat a detektor majdnem tökéletesen detektálja. A BGL-en mért teljesítmény 1000-10000 naplósornál bár 0.9 F1 érték feletti, nem tökéletes. Ennek az adathalmaznak a tökéletes reprezentálásához száz-ezer naplósor szükséges. Az Openstack adathalmazon a kódoló tanítóadathalmazának a mérete nem befolyásolta a módszer predikciós képességét. A tanítóadathalmazok mérete legnagyobb befolyással a HDFS-en mért teljesítményre volt. A HDFS adathalmazon úgy tűnik szüksége van a kódolónak minden naplósorra, hogy megfelelően tudja azokat reprezentálni.

A HDFS és BGL adathalmazokon mért javuló teljesítmény a tanítóadathal növekedésével azzal magyarázható, hogy az eseménysorokban sokféle új esemény fordul elő az adathalmaz minden részében. Ez azt jelenti, hogy ha csak egy részhalmazát adjuk oda a kódolónak a lehetséges eseményekről, azokból nem képes következtetni a még nem látott események struktúrájára. Ezt a jelenséget egyszerűen megszüntethetnénk azzal, hogy nem időrendben vagy a HDFS esetén blokkonként érjük el a maximalizált naplósorokat, hanem véletlen kiválasztással az összes naplósor közül. Ez a módszer valószínűleg már lényegesen kisebb tanítóadatmennyiségen is magas F1 értékeket érne el. Ennek a megközelítésnek viszont nem lenne gyakorlati haszna, hiszen a valós rendszereknél kizárólag időrendben kapjuk meg az eseménynaplókat, és egy robosztus módszernek alkalmazkodnia kell újszerű még nem látott naplókhoz.



4.5. ábra. Adathalmazonként maximalizált tanítóadat a kódolónak.

4.1.4. Előtanított kódolóval adathalmaz specifikus detektor

A módszerünket úgy is leteszteltük, hogy a kódolót az összes adathalmazon tanítottuk be, míg a detektort kizárólag egy címkézett adatforráson. Ezzel azt vizsgáltunk, hogy a kódoló által előállított reprezentációk tartalmazzák-e az összes adathalmazokon átívelő információt, vagy a detektort is több adathalmazos tanítással érdemes-e tanítani, hogy ezekhez teljesen hozzá tudjon férni.

Eredmények. A 4.2 táblázatban található méréseink azt mutatják, hogy érdemes a detektort egyszerre több adathalmazon tanítani. A BGL és Thunderbird adathalmazokon jó teljesítményt képes elérni az egy adathalmazos tanítással a detektor, bár a BGL adatokon a

kisebb beágyazási dimenzióknál nem elhanyagolható az F1 romlása a több adathalmazos tanításhoz képest. A Hadoop és HDFS adathalmazokon a specifikus detektor lényegesen rosszabb teljesítményt ér el, mint a több adathalmazos, amivel egyértelműen jobb teljesítményt lehet elérni. Az Openstacken elért teljesítményt nem befolyásolja, az hogy hány adathalmazon tanítjuk a detektort.

Előtanított kódolóval adathalmaz specifikus detektor F1 értékei					
<i>Beágyazási dimenzió</i>	<i>BGL</i>	<i>Hadoop</i>	<i>HDFS</i>	<i>Openstack</i>	<i>Thunderbird</i>
16	0.958	0.185	0.158	0.204	0.999
32	0.986	0.081	0.193	0.211	0.999
64	0.998	0.096	0.744	0.211	0.999
128	0.999	0.087	0.628	0.211	1

4.2. táblázat. Általános reprezentációkkal tanított specifikus detektor teljesítményei.

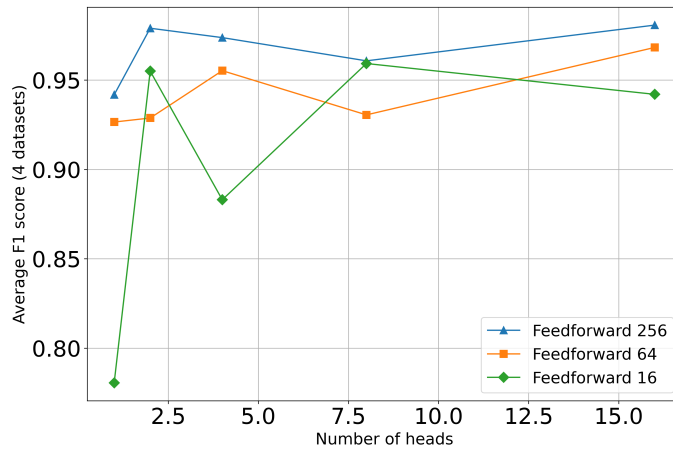
A teszt vágás hatása az Openstackre. Az Openstack adathalmazon az előző méréseknél visszatérően ugyanolyan gyenge ($F1=0.211$) teljesítményből arra következtettünk, hogy az adathalmaz pontatlan címkézettsége okozza. Ennek vizsgálatához lementük a modell teljesítményét különböző tanító-, kiértékelő- és teszt vágásokkal. A 4.3. táblázatban látható mérési eredmények azt mutatják, hogy az Openstacken elért teljesítményt jelentősen befolyásolja a tanító-, kiértékelő- és teszt vágás. Ezeknek az lehet egy magyarázata, hogy mivel az Openstack címkézése úgy történik, hogy amikor három nagy fájl kap egy-egy címkét, az abnormálisnak címkézett fájlban található olyan részek amik önmagukban normális működést írnak le. Ezek a félrecímkézett adatok azt eredményezik, hogy a modellünk olyan működést is hibásnak tanul meg ami teljesen normális, ezért látunk magas recall és alacsony precision értékeket. Ezért azt javasoljuk, hogy az Openstack adathalmazt a jövőben csak korlátozottan használjuk az általános modellek kiértékeléséhez.

Az Openstacken elért teljesítmény különböző tanító- és teszt vágásokkal			
Vágás	<i>Recall</i>	<i>Precision</i>	<i>F1</i>
10-45-45	0.997	0.167	0.286
20-40-40	0.996	0.348	0.515
30-35-35	0.998	0.319	0.483
40-30-30	1.000	0.431	0.602
50-25-25	0.998	0.250	0.400
60-20-20	0.996	0.210	0.348
70-15-15	0.995	0.375	0.544
80-10-10	1.000	0.118	0.211
90-5-5	1.000	0.118	0.211

4.3. táblázat. A legjobb teljesítmény a Loghub címkézett adathalmazain.

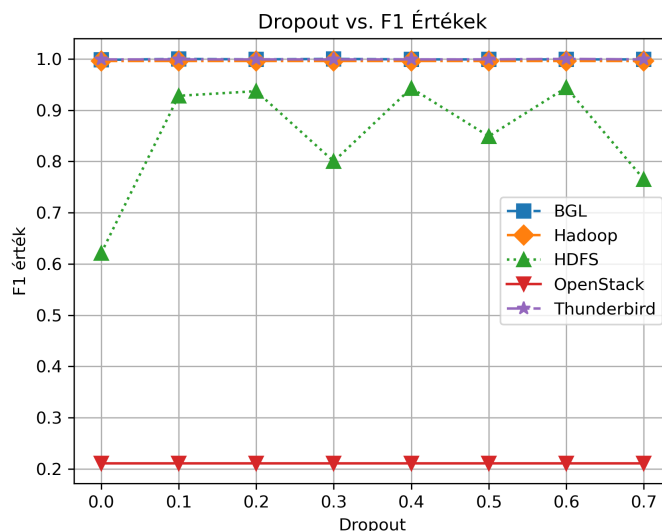
4.2. Ablációs mérések

A detektor méretének hatása. A teljes módszer működésében fontos kérdés, hogy megértsük a kódoló és detektor kapcsolatát. A legfőbb kérdés az, hogy az anomáliákra utaló jelek mennyire könnyen hozzáférhetőek a kódoló által előállított reprezentációkból. Ennek vizsgálatához különböző méretű detektor architektúrákat tanítottunk be ugyanannak a kódolónak a reprezentációin. Ehhez a méréshez 128 dimenziós reprezentációkat használtuk. A detektoroknál a transformerblokkokban található fejek számát és sűrű rétegek hatását vizsgáltuk az 5 címkézett adathalmazon elért átlagos teljesítményen. A 4.6. ábrán három különböző sűrűréteg mérettel és különböző fej számmal látható elért átlagos F1 teljesítményt. A legjobban a legnagyobb vizsgált modell teljesített míg a legrosszabbul a legkisebb. Úgy tűnik, hogy észrevehetően javítja a teljes módszer teljesítményét egy nagyobb detektor használata, bár az 1 fejet használó 16 dimenziós sűrű réteggel rendelkező detektor olyan kicsi, hogy arra lehet következtetni, hogy a kódoló reprezentációiból könnyen kinyerhetőek az anomáliákra utaló jelek.



4.6. ábra. A detektor méretének hatása a módszer teljesítményére fixált kódolóval.

Dropout a detektorban. Az általánosítóképesség javítása céljából a dropout beállítás kulcsfontosságú a legtöbb mélytanulási feladatban, ezért mi is megvizsgáltuk, hogy ez a paraméter a reprezentáció előállításánál hogyan befolyásolja a modellünk predikációs képességeit. A dropout paraméter az 5 vizsgált adathalmazból 4-re nem volt hatással. A 4.7. ábrán az látszik hogy a BGL, Hadoop és Thunderbird adathalmazokon közel tökéletes teljesítményt képes elérni a modell, míg az Openstack adathalmazon a 0,21-es F1 érték jelenik meg. A HDFS az egyetlen olyan adathalmaz ahol a dropout befolyásolta a teljesítményt. Azt a következtetést vontuk le, hogy a nem 0 dropout értékek kicsit habár, de észlelhetően jobban teljesítenek.



4.7. ábra. A dropout paraméter hatása az egyes adathalmazokon mért teljesítményre.

5. fejezet

Diszkusszió

5.1. Összefoglalás

A dolgozat során egy új megközelítést mutattunk be a számítógépes eseménynaplókban történő anomáliák detektálására. Létrehoztunk egy robusztos naplósor reprezentáló modellt. A módszer hatékonyságát a legnagyobb nyilvánosan elérhető eseménynapló gyűjteményén értékeltük ki, a Loghub projekt 5 különböző címkézett adathalmazán. A módszerünk egyidejűleg képes az öt adathalmazon átlagosan 0.86 F1 értéket elérni. Továbbá mérésekkel igazoltuk, hogy a módszer képes hatékonyan olyan adathalmazokra is általánosítani, amelyeken nem volt tanítva. Az eredményeink alátámasztják, hogy létrehozható olyan hatékony általános anomália kereső modell, amely felveszi a versenyt az egy-egy konkrét logtípusra specializált modellekkel. Méréseinkkel azt is megmutattuk, hogy érdemes egyidejűleg több adathalmazon tanítani mind egy naplósor reprezentáló modellt, mind egy naplósorszekvenciákban anomália detektálásra képes modellt.

A projekt forráskódja elérhető a GitHubon [2]. Ezzel lehetőséget teremtve az eredményeink ellenőrzésére és továbbfejlesztésére.

5.2. A módszer korlátai

A dolgozatban bemutatott módszer korlátai a kódoló transformer architektúrájából és az eseménynaplók lehetséges struktúrájából adódnak. Az előfeldolgozási lépések szükségesek a transformer megfelelő működéséhez, viszont korlátokat szabnak a módszer egészének.

XML/JSON/CSV formátumú naplók. A jelenleg használt előfeldolgozási lépések feltételezik, hogy a naplók sorokra bontva karaktersorozatokként elérhetőek. Az XML¹, JSON² és CSV³ formátumú naplók is elterjedt naplózási megközelítések. Ezeknek a feldolgozására nem képes a módszerünk, mert az XML és JSON nincsenek sorokra bontva és a CSV sorai nem tárolnak magukban elegendő információt az oszlopnevek nélkül.

A számok [NUM] tokenre cserélése. Az előfeldolgozási lépésben a szám karaktereket lecseréljük egy különleges [NUM] tokenre. Ez a lépés esetlegesen lényeges információt hagy el a naplósorból, mert lehetséges, hogy pont a számmal leírt értékből lehetne tudni, hogy hibás a naplózott rendszer működése.

A nem ASCII karakterek elhagyása. Az előfeldolgozásban a nem ASCII karakterek elhagyása lehetetlenné tehet bizonyos anomáliák detektálását. Például egy nem ASCII karaktereket használó felhasználónév megjelenése ott, ahol ez nem általános.

Számítási kapacitás. A több szintű kétirányú transformer alapú architektúra a kódolóban még kisebb beágyazási dimenzióknál is magas erőforrásigénnyel működik. Ez a gyakorlati felhasználását nehezíti a módszernek, mert korlátozza abban, hogy milyen gyorsan képes az anomáliadetektálásra. Ezen felül az üzemeltetési költségei is magasak egy nagy méretű kódolóban.

5.3. Fejlesztési lehetőségek

A dolgozatban bemutatott módszer teljesítményét több megközelítéssel is meg lehet próbálni javítani.

Több adat használata. Az egyik lehetséges javítási mód az, hogy ha több adatot használunk a modellek tanítására és kiértékelésére. Ez azt jelentené, hogy több különböző forrásból származó adatot használnánk a kódoló tanítására, így az képes lenne gazdagabb és általánosabb reprezentációk előállítására. Továbbá több címkézett adathalmazzal lehetne javítani a detektor teljesítményét és általánosító képességét. Ennek a megközelítésnek

¹Az XML formátum specifikációja: <https://www.w3.org/TR/REC-xml/>

²A JSON formátum specifikációja: <https://json-schema.org/specification>

³A CSV formátum specifikációja: <https://www.rfc-editor.org/rfc/rfc4180>

az a gyakorlati korlátja, hogy az eseménynaplók gyakran érzékeny információkat tárolnak, ezért nem szokták publikálni őket. Ezen felül az anomáliák címkézése költséges és időigényes feladat.

Felügyelet nélküli detektor. Egy könnyebben megvalósítható megközelítés a módszer javítására a felügyelet nélküli tanításon alapuló detektor alkalmazása. Ez a javítás a detektor tanítási adathalmazát a jelenlegi 5 címkézett rendszer naplóiról a teljes Loghub adatbázisra növelné meg. Ez növelné a módszer robusztusságát.

Kvantálás. A kódoló nagy számításigényét jelentősen lehetne csökkenteni kvantálással. Ez a megközelítés növelné a gyakorlati felhasználhatóságát a módszernek, hiszen az egységnyi idő alatt feldolgozott naplósorok számát megsokszorozná.

Köszönetnyilvánítás

A dolgozat elkészítésében köszönöm témavezetőmnek, Lukács Andrásnak a sok segítséget és értékes útmutatást.

A dolgozatban szereplő kutatást, amelyet a ELTE Matematikai Intézet, Mesterséges Intelligencia Kutatócsoport valósított meg, az Innovációs és Technológiai Minisztérium és a Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal támogatta a Mesterséges Intelligencia Nemzeti Laboratórium keretében és a Tématerületi Kiválósági Program 2021 - Nemzeti Kiválósági Alprogram „Mesterséges intelligencia, nagy hálózatok, adatbiztonság: matematikai megalapozás és alkalmazások” c. pályázatának keretében valósult meg.

Irodalomjegyzék

- [1] Shilin He és tsai. „Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics”. *CoRR* abs/2008.06448 (2020). arXiv: 2008 . 06448. URL: <https://arxiv.org/abs/2008.06448>.
- [2] Muzsai Lajos. *A program forráskódja*. <https://github.com/MuzsaiLajos/UnifiedLog>. 2023.
- [3] Van-Hoang Le és Hongyu Zhang. „Log-based anomaly detection with deep learning: how far are we?”: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. ACM, 2022. máj. DOI: 10 . 1145 / 3510003 . 3510155. URL: <http://dx.doi.org/10.1145/3510003.3510155>.
- [4] Zhuangbin Chen és tsai. *Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection*. 2022. arXiv: 2107 . 05908 [cs.SE].
- [5] Max Landauer és tsai. „Deep learning for anomaly detection in log data: A survey”. *Machine Learning with Applications* 12 (2023), 100470. old. ISSN: 2666-8270. DOI: <https://doi.org/10.1016/j.mlwa.2023.100470>. URL: <https://www.sciencedirect.com/science/article/pii/S2666827023000233>.
- [6] Ashish Vaswani és tsai. *Attention Is All You Need*. 2023. arXiv: 1706 . 03762 [cs.CL].
- [7] Sepp Hochreiter és Jürgen Schmidhuber. „Long Short-Term Memory”. *Neural Computation* 9.8 (1997), 1735–1780. old. DOI: 10.1162/neco.1997.9.8.1735.
- [8] Junyoung Chung és tsai. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].
- [9] Jacob Devlin és tsai. „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. 2018. URL: <https://arxiv.org/abs/1810.04805>.

- [10] Xinying Song és tsai. „Linear-Time WordPiece Tokenization”. *CoRR* abs/2012.15524 (2020). arXiv: 2012.15524. URL: <https://arxiv.org/abs/2012.15524>.
- [11] Hugging Face. *Tokenizers: A high-performance, efficient, and customizable tokenization library*. <https://github.com/huggingface/tokenizers>. 2023.
- [12] Pinjia He és tsai. *A Directed Acyclic Graph Approach to Online Log Parsing*. 2018. arXiv: 1806.04356 [cs.SE].
- [13] Min Du és tsai. „DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning”. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, 2017, 1285–1298. ISBN: 9781450349468. DOI: 10.1145/3133956.3134015. URL: <https://doi.org/10.1145/3133956.3134015>.
- [14] Weibin Meng és tsai. „LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.” *IJCAI*. 19. köt. 7. 2019, 4739–4745. old.
- [15] Sasho Nedelkoski és tsai. „Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs”. *2020 IEEE International Conference on Data Mining (ICDM)*. 2020, 1196–1201. old. DOI: 10.1109/ICDM50108.2020.00148.
- [16] Van-Hoang Le és Hongyu Zhang. *Log-based Anomaly Detection Without Log Parsing*. 2021. arXiv: 2108.01955 [cs.SE].
- [17] Hongcheng Guo és tsai. *TransLog: A Unified Transformer-based Framework for Log Anomaly Detection*. 2022. arXiv: 2201.00016 [cs.LG].
- [18] Van-Hoang Le és Hongyu Zhang. *Log Parsing with Prompt-based Few-shot Learning*. 2023. arXiv: 2302.07435 [cs.SE].
- [19] Amir Farzad és T. Aaron Gulliver. „Unsupervised log message anomaly detection”. *ICT Express* 6.3 (2020), 229–237. old. ISSN: 2405-9595. DOI: <https://doi.org/10.1016/j.icte.2020.06.003>. URL: <https://www.sciencedirect.com/science/article/pii/S2405959520300643>.
- [20] Xu Zhang és tsai. „Robust Log-Based Anomaly Detection on Unstable Log Data”. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing

- Machinery, 2019, 807–817. ISBN: 9781450355728. DOI: 10 . 1145 / 3338906 . 3338931. URL: <https://doi.org/10.1145/3338906.3338931>.
- [21] Siyang Lu és tsai. „Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network”. *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)*. 2018, 151–158. old. DOI: 10 . 1109 / DASC / PiCom / DataCom / CyberSciTec . 2018 . 00037.
- [22] Rodrigo Nogueira, Zhiying Jiang és Jimmy Lin. *Investigating the Limitations of Transformers with Simple Arithmetic Tasks*. 2021. arXiv: 2102.13019 [cs.CL].
- [23] Ilya Loshchilov és Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

Ábrák jegyzéke

2.1. A transformer architektúrája [6]	6
3.1. A UnifiedLog rendszer eseménynaplóval történő anomália detektálásra használt pipelineja.	11
3.2. Loghub naplókategóriák eloszlása.	12
3.3. Loghub naplósorok eloszlása	13
3.4. A naplósorok karakterhosszának az eloszlása	16
3.5. Naplósorok tokenhosszainak elosztásai különböző szótárméretekkel . . .	17
3.6. A szótárméret hatása a tokenizált sorok átlagos hosszára és szórására. . .	17
3.7. A naplósorokban található szám karakterek számának az eloszlása.	18
4.1. A kódoló tanítása 50 epochon keresztül.	20
4.2. A kódoló tanítása alatti validációs hibák minden adathalmazon 50 epo- chon keresztül.	20
4.3. Az enkóder tanításából egy kihagyott adathalmazzal vizsgált detektor tel- jesítmény.	23
4.4. Az öt címkézett adathalmaz kihagyása a kódoló tanításából.	24
4.5. Adathalmazonként maximalizált tanítóadat a kódolónak.	25
4.6. A detektor méretének hatása a módszer teljesítményére fixált kódolóval. .	28
4.7. A dropout paraméter hatása az egyes adathalmazokon mért teljesítményre.	28

Táblázatok jegyzéke

2.1. A mélytanuláson alapuló módszerek teljesítménye szemantikai információ használatával, illetve használata nélkül [4].	9
4.1. A legjobb teljesítmény a Loghub címkézett adathalmazain.	22
4.2. Általános reprezentációkkal tanított specifikus detektor teljesítményei. . .	26
4.3. A legjobb teljesítmény a Loghub címkézett adathalmazain.	27