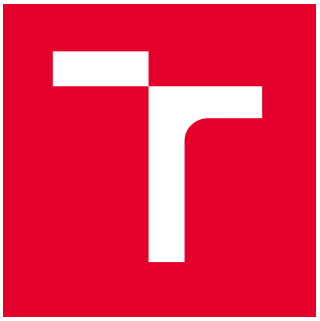
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2025 Bc. Ondřej Foltyn



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE POKROČILÝCH TECHNIK ROZŠÍŘENÍ DATOVÝCH SAD INTEGRUJÍCÍCH METODY STROJOVÉHO UČENÍ PRO ÚČELY SYNTAKTICKÉ ANALÝZY BEZPEČNOSTNÍCH LOGŮ

ADVANCED MACHINE LEARNING-ENHANCED AUGMENTATION TECHNIQUES FOR THE SYNTACTIC ANALYSIS OF SECURITY EVENT LOGS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Ondřej Foltyn**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Yehor Safonov**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Ondřej Foltyn ***ID:*** 230245

Ročník: 2 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Aplikace pokročilých technik rozšíření datových sad integrujících metody strojového učení pro účely syntaktické analýzy bezpečnostních logů

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem diplomové práce je aplikace pokročilých NLP technik pro rozšíření vlastních datových sad a porovnání efektivity vybraných technik hlubokého učení s klasickými metodami rozšiřování záznamů událostí. Proces rozšíření bude aplikován na datové sadě obohacené o data vygenerovaná prostřednictvím experimentálního pracoviště, které zahrnuje data simulovaných útoků vytvořených pomocí nástroje Atomic Red Team. V teoretické části práce bude provedena rešerše existujících augmentačních technik se zaměřením na analýzu bezpečnostních logových záznamů. Budou zohledněna specifika korelace logů pomocí SIEM řešení a problémy spojené s normalizací vstupních dat. Zaměřte se na typy a formáty záznamů událostí, jejich různorodost a relevanci pro daný sektor. Při rozšiřování logů bude kladen důraz na pokročilé techniky využívající embedding algoritmy (např. Word2Vec, GloVe, Doc2Vec, FastText), hluboké neuronové sítě typu Transformer a metody umožňující realizovat Masked Language Modeling. Nastudujte možnosti nástroje MLflow pro sledování experimentů a správy modelů. Po provedené rešerši implementujte pět vybraných algoritmů a porovnejte účinnost těchto technik oproti základnímu generování logů. Na základě vytvořené rozšířené datové sady argumentujte finální přesnost Transformer sítě typu T5 při řešení syntaktické analýzy logových záznamů. Diskutujte existující omezení a možnosti dalšího rozšíření.

DOPORUČENÁ LITERATURA:

[1] TUNSTALL, L., VON WERRA, L., WOLF, T. Natural Language Processing with Transformers, Revised Edition. O'Reilly Media, 2022. ISBN 978-1-0981-3676-5.

[2] MARTINEZ, Roberto Incident Response with Threat Intelligence. Packt Publishing, 2022. ISBN 1801070997.

Termín zadání: 10.2.2025 ***Termín odevzdání:*** 27.5.2025

Vedoucí práce: Ing. Yehor Safonov

prof. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

V moderních systémech správy bezpečnostních informací a událostí představuje zpracování bezpečnostních záznamů velkou výzvu, zejména kvůli jejich často nestrukturovanému charakteru. Na rozdíl od běžných textů vykazují záznamy událostí specifickou strukturu vět a obsahují informace specifické pro oblast kybernetické bezpečnosti. Tyto rozdíly významně komplikují implementaci tradičních technik rozšiřování dat, které pak často mohou narušit sémantickou integritu a narušit klíčové kontextové vazby nezbytné pro efektivní analýzu. Navzdory rostoucímu zájmu o integraci metod hlubokého učení do oblasti bezpečnostního monitorování zůstává problematika datové augmentace v oblasti bezpečnostních záznamů nedostatečně prozkoumaná, s omezeným množstvím dostupných publikací zabývající se tímto tématem. Tato práce se proto zabývá návrhem, implementací a validací pokročilých metod rozšíření dat založených na aplikaci jazykových modelů. V této práci je představen nástroj pro augmentaci záznamů událostí, který aplikuje různé techniky generování textu pro syntetické rozšíření specifických metadat (tzv. metaklíčů) s důrazem na zachování sémantických vazeb a doménové relevance. Pro rozšíření specifických metaklíčů maskovaných entit záznamů bylo testováno sedm jazykových modelů založených na architektuře Transformer. Konkrétně byly testovány čtyři modely typu Masked Language Modeling (MLM) a tři generativní modely typu Next Word Prediciton (NWP). Tyto modely byly v prvních krocích laděny na relevantních datech a následně testovány na vytvořené datové sadě a datové sadě obohacené o simulované bezpečnostní záznamy generované nástrojem Atomic Red Team. Validace augmentačních metod byla provedena na úlohách z oblasti zpracování přirozeného jazyka (NLP). Samotné testování potvrzuje rostoucí potenciál velkých jazykových modelů pro inteligentní augmentaci bezpečnostních záznamů a cílené rozšiřování doménově specifických metadat.

KLÍČOVÁ SLOVA

Augmentace, BERT, bezpečnost, generování textu, jazykové modely, logové záznamy, metody rozšíření, modelování maskovaného jazyka, SIEM, SOAR, Splunk, Transformer, umělá inteligence, záznamy událostí

ABSTRACT

In modern security information and event management systems, the processing of security records is a major challenge, especially due to their often unstructured nature. In contrast to regular text, event records exhibit a specific sentence structure and contain information specific to the cybersecurity domain. These differences significantly complicate the implementation of traditional data augmentation techniques, which in turn can often compromise semantic integrity and break key contextual links necessary for effective analysis. Despite the growing interest in integrating deep learning methods into the field of security monitoring, the issue of data augmentation in security records remains under-researched, with a limited number of publications available addressing this topic. Therefore, this paper addresses the design, implementation and validation of advanced data augmentation methods based on the application of language models. In this work, an event record augmentation tool is presented that applies different text generation techniques to synthetically augment specific metadata (called meta-keys) with an emphasis on preserving semantic links and domain relevance. Seven language models based on the Transformer architecture were tested for the extension of specific metakeys of masked record entities. Specifically, four Masked Language Modeling (MLM) models and three generative Next Word Prediciton (NWP) models were tested. These models were tuned on relevant data in the first steps and then tested on a created dataset and a dataset enriched with simulated security records generated by the Atomic Red Team tool. Validation of the augmentation methods was performed on natural language processing (NLP) tasks. The testing itself confirms the growing potential of large-scale language models for intelligent augmentation of security records and targeted extension of domain-specific metadata.

KEYWORDS

Artificial intelligence, augmentation methods, BERT, event logs, language models, log records, masked language modeling, SIEM, SOAR, Splunk, text generation, Transformer

FOLTYN, Ondřej. *Aplikace pokročilých technik rozšíření datových sad integrujících metody strojového učení pro účely syntaktické analýzy bezp. logů*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2025. Vedoucí práce: Ing. Yehor Safonov

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Ondřej Foltyn
VUT ID autora:	230245
Typ práce:	Diplomová práce
Akademický rok:	2024/25
Téma závěrečné práce:	Aplikace pokročilých technik rozšíření datových sad integrujících metody strojového učení pro účely syntaktické analýzy bezp. logů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.*

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora**

*Prohlašuji, že při zpracování této práce byly využity nástroje generativní umělé inteligence a to výhradně v asistenční roli pro stylistickou úpravu textu a vyhledávání zdrojů. Tyto výstupy byly kriticky posouzeny, upraveny a použity v souladu s interním doporučením univerzity.

**Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Yehoru Safonovi a také Ing. Pavlu Sikorovi za možnost zapojit se do výzkumu, odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále děkuji své rodině a všem blízkým za podporu a motivaci při realizaci této diplomové práce.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Stále se zrychlující technologický pokrok a hluboká digitalizace všech oblastí lidské činnosti přinášejí nejen nové možnosti, ale také zásadní bezpečnostní výzvy. V posledních letech dochází k výraznému nárůstu počítačových hrozeb, kdy v roce 2024 byl zaznamenán dramatický nárůst kybernetických útoků o 75 % oproti předchozímu roku, přičemž organizace čelily týdně v průměru 1 876 útokům [1]. Tento trend potvrzuje potřebu vývoje pokročilých, robustních detekčních systémů a mechanismů pro rychlou reakci na incidenty. Stovky až tisíce systémů po celém světě jsou denně napadány a narušovány, což vede k dopadům nejen v podobě finančních ztrát, ale také k poškození důvěry a reputace organizací napříč sektory od zdravotnictví, přes výrobu až po vládní instituce [1].

Vzhledem k těmto skutečnostem je důsledné monitorování a analýza síťových přenosů nedílnou součástí efektivního zabezpečení koncových zařízení. Klíčové prvkem je v tomto ohledu především sběr a zpracování záznamů událostí, které nesou informace o různých událostech a procesech [2]. Kvalitní a jednoduše interpretovatelné záznamy mohou poté sloužit jako stavební kámen odhalení podezřelého chování, zjišťování možných útoků nebo pozorování jejich průběhu [2].

Kvůli obrovskému objemu generovaných dat a rostoucím nárokům na zpracování záznamů se metody strojového učení stále více prosazují jako nástroje, které umožňují automatizovanou analýzu a interpretaci těchto dat [3, 4]. Nicméně problémem spojeným s těmito pokročilými přístupy je nedostatek kvalitních trénovacích dat, které mohou odrážet reálné scénáře útoků a incidentů. Tato výzva vede k rostoucímu zájmu o různé techniky rozšiřování dostupných textových dat s cílem zvýšit jejich variabilitu a reprezentativnost [5].

Z těchto nedostatků plyne jeden z hlavních cílů této práce. Tím je vytvořit nástroj, který bude pracovat na základě architektury neuronových sítí a bude schopen rozšířit dostupné datové sady sesbíraných a vytvořených logových záznamů. Tento proces je důležitý z hlediska nedostatku kvalitních datových sad, jež pomohou vytvořit robustní nástroje pro procesní analýzu sítí [6].

Diplomová práce je systematicky rozdělena do čtyř kapitol. První kapitola nabídne teoretický přehled problematiky kybernetické bezpečnosti a nástrojů pro bezpečnostní monitorování. Bude proveden teoretický rozbor záznamů událostí a jejich obecný popis. Prostor bude také dále věnován samotnému monitorování v kybernetickém prostoru a popisu systémů využívajících se pro detekci a vyhodnocování kybernetických hrozeb nebo incidentů. Velmi rozšířenými typy těchto systémů jsou SOAR nebo SIEM bezpečnostní zařízení, která se využívají pro sběr a analýzu bezpečnostních hrozeb. Dále také systémy XDR a EDR, umožňující integraci spolu s předchozími dvěma systémy. Kapitola bude uzavřena odezvou na bezpečnostní

incidenty a možné vylepšení stávajících systémů pomocí metod strojového učení.

Druhá kapitola se zaměří na teoretický přehled problematiky umělé inteligence, strojového a hlubokého učení. Nejprve budou představeny vlastnosti neuronových sítí a principy jejich fungování. Následovat bude popis odvětví klíčových výzev a strategií používaných ve strojovém a hlubokém učení, včetně metody přeneseného učení. Následně se pozornost přesouvá k oblasti zpracování přirozeného jazyka, která je stěžejní pro tuto práci, zejména pak popis hlubokých neuronových sítí typu `Transformer`, které tvoří základ moderních jazykových modelů. Závěr kapitoly bude věnován možnostem hodnocení kvality strojového učení a trénování modelů, a také aplikaci metod strojového učení v rámci syntetického rozšíření záznamů v oblasti kybernetické bezpečnosti.

Třetí kapitola této práce je věnována návrhu a implementaci metod pro rozšíření souborů dat bezpečnostních záznamů. Nejprve bude provedena rešerše současných přístupů k rozšíření (augmentaci) textu, který slouží jako prvotní inspirace pro návrh konečného řešení. Poté bude následovat charakteristika použitých datových souborů a vysvětlení procesu zpracování surového textu. Dále budou podrobněji představeny jazykové modely založené na architektuře `Transformer` pracující na principu metod *Masked Language Modeling* (MLM) a *Next Word Prediction* (NWP). Kapitola se poté přesune k procesu ladění vybraných jazykových modelů nad připravenými datovými sadami, včetně popisu učících procesů, použitých parametrů a vyhodnocení dosažených výsledků. Na závěr budou porovnány výsledky trénování jednotlivých modelů. Výstupem kapitoly budou jazykové modely, které budou připraveny k testování a následnému nasazení v navrženém systému pro rozšíření dat.

Čtvrtá a poslední kapitola diplomové práce bude zaměřena na implementaci a komplexní vyhodnocení nejen doladěných jazykových modelů. Hlavním cílem této části bude detailně popsat postup testování modelů na samostatné, doposud nevyužité testovací množině, který umožní nezávislé posouzení generalizační schopnosti modelů mimo původní tréninkovou množinu. Kapitola dále také přiblíží způsob možného nasazení těchto modelů k praktickému využití rozšíření datových sad. Důraz bude tedy kladen nejen na testování, ale také na hodnocení výstupů vybraných augmentačních metod, jak v rámci syntetických scénářů, tak při aplikaci na záznamy z oblasti simulovaných útoků. Pro vyhodnocení budou využity obecně známé metriky, jako je přesnost, preciznost generování, F-míra, či využití matice záměn. Součástí testování bude rovněž použití úloh rozpoznávání pojmenovaných entit (NER) pro posouzení kvality rozšířených záznamů. Kapitola také shrne vzniklé problémy během testování a navrhne možná řešení pro vylepšení systému.

1 Kybernetická bezpečnost a monitorování

Kybernetická bezpečnost označuje souhrn technologií, postupů a procesů navržených k ochraně sítě, zařízení, dat a aplikací před útoky, poškozením a neoprávněným přístupem. S rychlým rozvojem technologie dnešní doby roste také hrozba kybernetických útoků, proto je velmi důležité věnovat pozornost bezpečnostním opatřením, která proaktivně chrání před těmito počítačovými útoky a umožňují minimalizovat možná rizika [1, 7].

V této souvislosti je klíčové správně implementovat systémy umožňující protokolování a monitorování jednotlivých akcí pro možnou detekci bezpečnostních událostí v reálném čase, jejich předcházení a reakci na ně. Těmito opatřeními je zajištěn přehled o činnosti systému, následná identifikace potenciální hrozby a možnost rychlé reakce na jakékoliv neobvyklé chování. Pro minimalizaci rizik a ochranu před možnými útoky je tedy důležitá efektivní správa a monitorování záznamů [8].

1.1 Problematika záznamu událostí

Záznamy události, nebo také logové záznamy (zkr. logy), jsou systematicky generované záznamy obsahující informace o událostech a procesech probíhajících v síťových zařízeních (jako jsou routery nebo switche), operačních systémech, bezpečnostních aplikacích, softwarových aplikacích a dalších digitálních systémech (např. databáze, cloudové aplikace, weby, atd.) [3].

Původně byly logy používány primárně pro řešení problémů, které nastávaly v aplikacích, nyní je využívá většina organizací k různým funkcím, jako jsou například optimalizace výkonu systému, zaznamenávání akcí provedených uživateli nebo také poskytování užitečných údajů pro vyšetřování škodlivých aktivit. Co se bezpečnostního hlediska týče, jejich zpracování a vyhodnocování je klíčové pro analýzu stavu sítě, jejího zabezpečení a událostí, které v ní nastávají. Mohou poskytovat přehled o potenciálních útocích na infrastrukturu sítě, případně je lze využít k analýze předchozích incidentů a předejít tak možným útokům [2, 3].

Základními zdroji logových záznamů jsou tzv. *Log Event Sources* (LES) a *Packet Event Sources* (PES). LES zdroje generují události na základě strukturovaných dat vytvořených softwarovými aplikacemi nebo systémy zaznamenávajícími akce uživatelů a různé stavy, jako chyby nebo změny konfigurace. PES zachycuje síťové pakety a umožňuje tak analýzu dat přenášených mezi zařízeními v síti (např. požadavky a odpovědi HTTP, DNS dotazy, apod.), což je klíčové při detekci útoků [9].

Logové záznamy lze rozdělit do různých skupin podle jejich struktury. V první řadě je lze dělit podle toho, zda jsou strukturované, částečně strukturované (*semi-structured*) nebo nestrukturované [10]. Strukturovaný formát záznamů událostí má

jasný a konzistentní vzor a je čitelný lidmi i stroji. Pole jsou rozdělena znakem, jako je čárka (v souborech CSV), mezera nebo pomlčka a lze je také spojovat rovnítkem (například `name=John`). Takto strukturované záznamy může přijímat a zpracovávat většina systémů. Nestrukturovaný formát záznamu neobsahuje žádný konkrétní vzor, což ztěžuje rozdělení události a extrakci párů klíč-hodnota. Tento typ formátu vyžaduje vlastní parsování[1] pokud není v systému pro správu a zpracování vestavěný parser, což může výrazně zvýšit časovou náročnost. Polostrukturované logové záznamy jsou čitelné lidmi i stroji, protože obsahují schéma nebo vzor, ale mají složitější oddělovače polí a událostí. Systémy jsou schopné přijímat tyto logy, ale často vyžadují implementovaný parser [10].

Logové záznamy se mohou vyskytovat v různých formátech a jsou definovány řadou standardů. Mezi velmi časté formáty logů se řadí například `Windows Event Log`, `Apache Access Log`, `Syslog RFC 3164` nebo `Syslog RFC 5424` a další. `Syslog` protokol poskytuje rámec pro vytváření, ukládání a přenos záznamů protokolu, který je navržen pro využití jakýmkoliv operačním systémem, bezpečnostním softwarem nebo aplikací, pokud je k tomu navržen [2]. Řada zdrojů záznamů jej používá jako svůj nativní formát protokolů, nebo poskytuje konverzi svých formátů do `Syslog` formátů. `Syslog` záznam má být velmi jednoduchý a typicky obsahuje časovou značku, úroveň závažnosti, název hostitele nebo IP adresu zdroje a samotný obsah zprávy protokolu [2]. Protokoly operačního systému Windows jsou uloženy v proprietárním formátu známém jako `Windows Event Log` obsahující podrobné informace o událostech. Tento formát lze rozdělit do několika kategorií (například aplikační, bezpečnostní, systémový) a podle nich se poté liší typ uložených informací [2, 10]. `Apache access logy` zaznamenávají HTTP požadavky přijaté webovým serverem a každý záznam často obsahuje IP adresu klienta, časovou značku, použitou metodu, stavový kód a velikost odpovědi [11]. Čím dál častěji se také v moderních distribuovaných systémech využívají `JSON` (*JavaScript Object Notation*) formáty. Jsou polostrukturované, obsahují více párů klíč-hodnota a nabízejí tak flexibilní a lidsky i strojově čitelné záznamy. Jeden z dalších mnoha formátů je `CEF` (*Common Event Format*). Tento otevřený textový formát protokolu je využívaný zařízeními a aplikacemi souvisejícími se zabezpečením. Používají kódování UTF-8 a předpona obsahuje časové razítko a název hostitele. V záhlaví se potom nejčastěji objevuje verze softwaru `CEF`, výrobce zařízení, produkt zařízení, název a závažnost a zbytek zprávy protokolu obsahuje další vlastní pole, která ji obohacují [10].

[1] Syntaktická analýza logů, neboli parsování, je proces, při kterém se provádí analýza a převod do strukturované formy, za účelem extrakce užitečných dat z logových záznamů [3].

```
1. Syslog (RFC 3164):
<34>Oct 21 14:55:02 mymachine sshd[12345]: Accepted password for user1 from 192.168.0.2 port 51822 ssh2

2. Windows Event Log:
Event Type:    Audit Success
Event Source:  Security
Date:          2025-04-21 14:55:02
Event ID:      4624
Task Category: Logon
Level:         Information
User:          N/A
Computer:      DESKTOP-ABC123
Description:
An account was successfully logged on.

3. Apache Access Log
192.168.0.2 - frank [21/Apr/2025:14:55:02 +0000] "GET /index.html HTTP/1.0" 200 2326

4. JSON log:
{
  "timestamp": "2025-04-21T14:55:02Z",
  "level": "INFO",
  "service": "auth-service",
  "message": "User login successful",
  "user_id": "user1",
  "ip_address": "192.168.0.2"
}

5. CEF log:
<117>Mar 19 15:19:15 CEF:0|SecurityVendor|Firewall|1.0|100|Blocked inbound connection|5|src=192.168.0.2
dst=10.0.0.5 spt=443 dpt=56789 proto=TCP
\end{lstlisting}
```

Obr. 1.1: Příklady formátů logových záznamů.

1.2 Monitorování bezpečnosti v kybernetickém prostředí

Bezpečnostní monitorování je souhrnný název pro proces detekce kybernetických hrozeb a řízení bezpečnostních incidentů. Obecně se může dělit na dvě fáze [8]. V první fázi probíhá sběr a analýza logových záznamů, dat a příznaků bezpečnostních hrozeb. Druhá fáze se soustředí na reakci, která je zaměřena na nápravu zjištěných bezpečnostních rizik [8].

Monitorováním celé IT infrastruktury se věnuje tým odborníků na bezpečnost v kyberprostoru. Tento tým se nazývá Bezpečnostní operační centrum (*Security Operations Center*, SOC) a slouží jako monitorovací centrum, jehož cílem je odhalení, analýza a reakce na bezpečnostní incidenty v reálném čase [12].

V počáteční fázi provádí SOC preventivní údržbu formou pravidelných softwarových aktualizací, aktualizací firewallu a dalších bezpečnostních zásad a postupů. SOC také pomáhá při vytváření postupů zálohování dat pro zajištění kontinuity provozu, v případě, že by došlo k narušení bezpečnosti nebo jinému bezpečnostnímu incidentu. V další fázi provádí SOC monitorování IT infrastruktury, shromažďování

údajů ze zařízení v síti a detekci anomálií anebo potenciálních hrozeb. Pro mnoho SOC je hlavní technologií SIEM, která umožní toto monitorování a také technologie rozšířené detekce a reakce XDR. Poté probíhá analýza pro posouzení závažnosti jednotlivých událostí. Reakční fáze následuje v případě potvrzení bezpečnostního incidentu. Tato opatření zahrnují například izolaci napadených oblastí sítě, pozastavení nebo zastavení napadených aplikací nebo procesů, odstranění poškozených nebo škodlivých souborů a dalších akcí pro minimalizaci dopadu útoku [12, 13].

1.3 Systémy pro bezpečnostní monitorování

Pro zajištění včasné detekce, analýzy a reakce na možné incidenty byla vyvinuta řada nástrojů, podporujících funkce pro efektivní detekci, vyhodnocování a prevenci počítačových hrozeb. Tyto nástroje pak umožňují bezpečnostním týmům získat ucelený přehled o dění v síti, automatizovat rutinní úlohy a zrychlit tak rozhodovací proces. Mezi velmi často využívané platformy patří například SIEM technologie, která slouží ke shromažďování a korelaci bezpečnostně relevantních dat, nebo systémy SOAR, umožňující automatizaci reakce na bezpečnostní incidenty. V praxi se tyto systémy často také kombinují a usnadňují tak SOC týmům komplexní přehled nad provozem v síti a umožní tak efektivnější reakce na možné incidenty. Následující podkapitoly budou věnovány podrobnějšímu popisu nástrojům SIEM, SOAR, EDR a XDR, jejich hlavním funkcím a přínosům v rámci moderní bezpečnostní architektury.

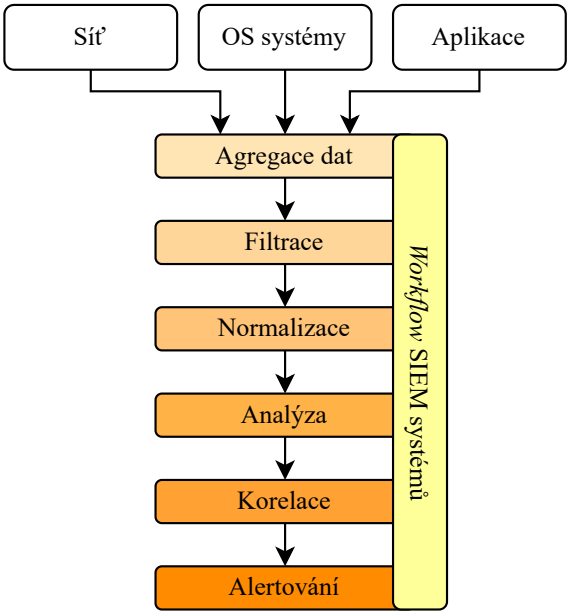
1.3.1 *Security Information and Event Management* (SIEM)

Bezpečnostní systém SIEM (*Security Information and Event Management*) spojuje kategorie SIM (*Security Information Management*) a SEM (*Security Event Management*). SIM se využívá pro sběr důležitých údajů, co se bezpečnostního hlediska týče, a pro generování záznamů. SEM stojí za analýzou bezpečnostních incidentů, korelací incidentů a upozorněními v reálném čase [3].

Technologie SIEM je zodpovědná za sběr dat, poskytujících informace ohledně možných kybernetických hrozeb a incidentů. Sbíraná data jsou ukládána v nezměněné podobě, ve formě takzvaných *raw* logů a následně jsou mezi nimi vytvořeny logické vazby – detekční a korelační pravidla [3, 14]. Tyto korelace vychází z předem definovaných pravidel a pomáhají identifikovat relevantní bezpečnostní události a odlišit je od falešných poplachů. Sběr a analýza těchto dat z aplikací, serverů nebo uživatelů probíhá tedy za účelem detekce a blokace možných útoků [14].

SIEM přijímá data událostí z různorodých zdrojů napříč celou IT infrastrukturou, mezi něž patří události od uživatelů, koncových bodů, aplikací, zdrojů dat, cloudových pracovních zátěží a sítí, stejně jako data z bezpečnostního hardwaru

a softwaru, jako jsou firewally nebo antivirové programy, a jsou následně v reálném čase shromažďována, korelována a analyzována. Některá SIEM řešení jsou také integrována s *Threat Intelligence* kanály třetích stran o hrozbách a korelují tak svá interní bezpečnostní data s dříve rozpoznanými signaturami a profily hrozeb. Tato integrace umožní týmům detekovat nebo blokovat nové typy signatur útoků [16].



Obr. 1.2: Workflow SIEM systému.

Před zahájením analýzy logů je potřeba provést řadu kroků, které zahrnují využití dostupných funkcí systému SIEM [3, 15, 16]. Tyto kroky jsou schematicky znázorněny na obrázku 1.2 a jejich funkce je následující:

1. **Zachycení a agregace záznamů**: Provádí se shromažďování informací z různých typů systémů a jejich následné ukládání na centralizované místo. Může být provedeno různými způsoby, nejčastěji jsou ale využívány dvě techniky. Technika *push*, která je založena na konfiguraci cílové IP adresy a portu uživatelem na koncovém zařízení (např. `Syslog`) a technika *pull*, kdy je potřeba připojení k zařízení a následnému stažení logů (např. `Windows Event Log`) [3]. Poté se provede seskupování logů podle podobných charakteristik, což vede k možnosti následné filtrace dat.
2. **Filtrace**: Filtrování nepotřebných a nebo neužitečných dat z hlediska bezpečnostního monitoringu.
3. **Normalizace záznamů**: Pro normalizaci se využívají konektory systému SIEM přizpůsobené jednotlivým verzím, typům zařízení a dodavatelům. Tyto konektory mají za úkol analyzovat vstupní události a převádět je do společného formátu srozumitelného pro SIEM.

4. **Analýza a korelace**: Korelační pravidla jsou velkou výhodou těchto řešení. V tomto kroku dochází k hodnocení normalizovaných událostí s cílem detekce bezpečnostních incidentů, kdy SIEM využívá naprogramovaná pravidla k rozpoznání anomálií. Moderní systémy kromě předem stanovených pravidel mohou rovněž využívat metody strojového učení za účelem odhalení nových a neznámých hrozeb.
5. **Generování upozornění a vytváření reportů**: Na základě nalezených hrozeb jsou vytvořena varování a reporty ohledně bezpečnostních událostí.

SIEM je tedy komplexní platformou, nabízející mnoho funkcí, jejichž stručný přehled lze vidět na obrázku 1.2. Pro zvýšení přesnosti detekce a automatizace reakce na bezpečnostní incidenty jsou moderní řešení systému SIEM obohacena integrací umělé inteligence a strojového učení [3, 15].

1.3.2 *Security Orchestration, Automation and Response* (SOAR)

Platforma SOAR (*Security Orchestration, Automation and Response*) umožňuje bezpečnostním týmům integrovat a koordinovat samostatné bezpečnostní nástroje, automatizovat opakující se úkoly a zefektivnit tak pracovní postupy při řešení bezpečnostních incidentů a hrozeb [17]. SOAR poskytuje týmům SOC centrální konzoli, v níž mohou být začleněny nástroje pro reakci a detekci do optimalizovaných scénářů a automatizovat tak opakující se úlohy v těchto postupech. Výhodou této konzole je také možnost spravovat všechna bezpečnostní upozornění, která byla generována těmito nástroji, na jednom centrálním místě [13, 17].

Technologie SOAR nabízí propojení platforem pro reakci na bezpečnostní incidenty, orchestraci a automatizaci zabezpečení a analýzu hrozeb. Tím umožňuje centralizované zpracování, korelaci a vyhodnocování bezpečnostní události. Pomocí takzvaných *playbooků* mohou být předem definovány scénáře reakcí a v případě detekce incidentu se automaticky spustí. Tyto *playbooky* mohou být plně automatizované, plně manuální nebo mohou kombinovat automatizované a manuální úkoly [17]. Na obrázku 1.3 níže je znázorněn příklad funkce SOAR systému integrovaného v bezpečnostních systémech.

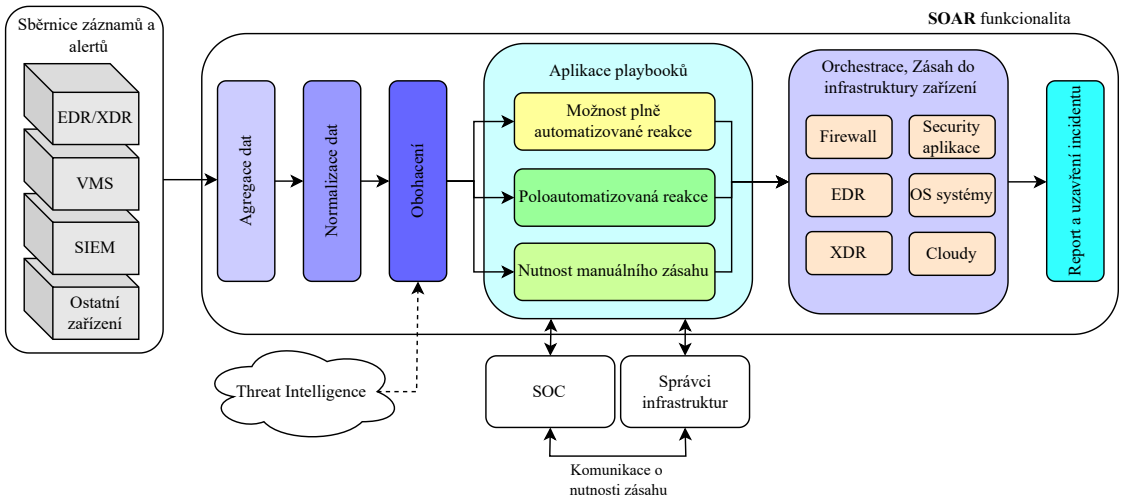
Základními funkcemi protokolu SOAR jsou:

1. **Orchestrace**: Orchestrace se vztahuje k tomu, jakým způsobem SOAR propojuje a koordinuje hardwarové a softwarové nástroje v rámci bezpečnostního systému společnosti. Lze tedy dosáhnout lepší koordinace a efektivnější reakce na bezpečnostní incidenty [17].
2. **Automatizace zabezpečení**: SOAR nabízí možnost automatizace časově náročných, opakujících se úloh nízké úrovně. Tyto úlohy mohou být například otevírání a uzavírání tiketů na technickou podporu, obohacování událostí

nebo určování priority výstrah. Tímto způsobem lze dosáhnout vyšší efektivity a rychlosti reakce na bezpečnostní incidenty prostřednictvím automaticky spouštěných integrovaných bezpečnostních nástrojů [17].

3. **Reakce**: Orchestrace a automatizace softwaru SOAR umožňují plnit funkci centrální konzole pro reakci na bezpečnostní incidenty. K prověření a řešení incidentů mohou analytici vzájemně porovnávat data z různých zdrojů, odfiltrovat falešně pozitivní signály, určovat prioritní výstrahy a rozpoznávat konkrétní hrozby. Poté je možné spustit připravený seznam úloh a zabránit tak škodám napáchaným útočníkem. Automatizace tak umožňuje efektivní a problémové reakce na incidenty [13, 17].

Díky integraci bezpečnostních nástrojů a automatizaci úloh dokážou platformy SOAR zefektivnit běžné bezpečnostní postupy, jako je například správa incidentů, správa zranitelností a reakce na incidenty [17].



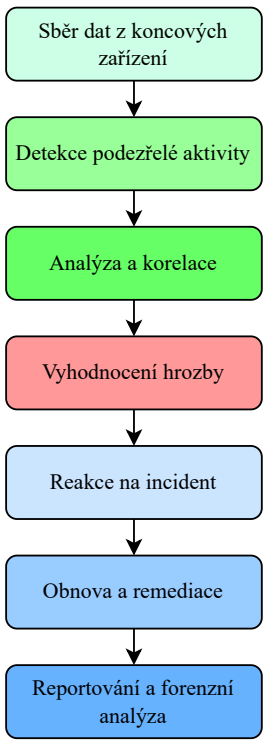
Obr. 1.3: Workflow SOAR systému.

Porovnání SOAR a SIEM

Technologie SIEM sbírá informace z bezpečnostních nástrojů a shromažďuje je v centrálním logu. Po následné analýze generuje anomálie. SOAR sbírá bezpečnostní informace a data ze SIEM pomocí centralizované platformy. Díky přidaným funkcím automatizace a orchestrace do SOAR řešení snižují potřebu zásahu analytika pro správu pravidel a upozornění [13, 17].

1.3.3 *Endpoint Detection and Response* (EDR)

Systémy EDR (*Endpoint Detection and Response*) představují pokročilé bezpečnostní nástroje, jejichž cílem je monitorování, analýza a reakce na hrozby vyskytující se na koncových zařízeních [18]. Tento software využívá analytických funkcí v reálném čase a automatizaci řízenou umělou inteligencí pro ochranu koncových uživatelů, koncových zařízení a IT prostředků organizací před počítačovými hrozbami, jež procházejí antivirovým softwarem a dalšími tradičními bezpečnostními nástroji pro koncové body [18].



Obr. 1.4: Workflow EDR systému.

Ze všech koncových bodů v síti, jako jsou stolní a přenosné počítače, servery, mobilní zařízení, zařízení IoT (*Internet of Things*) a další, EDR neustále shromažďuje data – typicky jde o informace o běžících procesech, výkonu, změnách konfigurací a v souborových systémech, chování koncových uživatelů nebo zařízení. Tato data následně v reálném čase analyzuje a vyhledává důkazy o známých nebo předpokládaných kybernetických hrozbách. Poté může systém reagovat automaticky nebo poloautomaticky, za účelem předcházení nebo zmírňování škody vzniklé v důsledku zjištěných hrozeb. Mezi typické reakce na identifikované hrozby může patřit izolace zařízení nebo uživatele od sítě, zastavení systémových nebo koncových procesů, anebo spuštění antivirového nebo antimalwarového softwaru ke kontrole ostatních koncových bodů v síti [18].

Moderní EDR řešení lze často integrovat se systémem SIEM nebo SOAR. Integrace SIEM umožňuje obohatit analýzu EDR o další souvislosti a kontext z jiných částí infrastruktury, jako například logové záznamy z firewallu. SOAR systémy nabízí automatizaci postupů pro bezpečnostní reakce na možné hrozby. Následně dochází k izolaci hrozeb, jejich nápravě a vytvoření záznamů pro budoucí vyhledávání a analýzu [18].

Nasazení EDR výrazně zvýší úroveň viditelnosti do dění na koncových bodech a pomůže bezpečnostním týmům s včasnou reakcí na incidenty a hrozby, které běžná antivirová řešení nemusí zachytit.

1.3.4 *Extended Detection and Response* (XDR)

XDR (*Extended Detection and Response*) je platforma kybernetické bezpečnosti integrující bezpečnostní nástroje. Sjednocuje operace zabezpečení napříč všemi úrovněmi ochrany – uživateli, koncovými body, e-maily, aplikacemi, sítěmi, cloudovými zátěžemi a dalšími daty [19].

Standardně se XDR využívá jako cloudové řešení nebo řešení typu SaaS (*Software-as-a-Service*). Může také sloužit jako hlavní technologie, která je základem nabídky poskytovatele cloudových nebo bezpečnostních řešení v oblasti řízené detekce a reakce (*Managed Detection and Response*, MDR). XDR lze integrovat jako [19]:

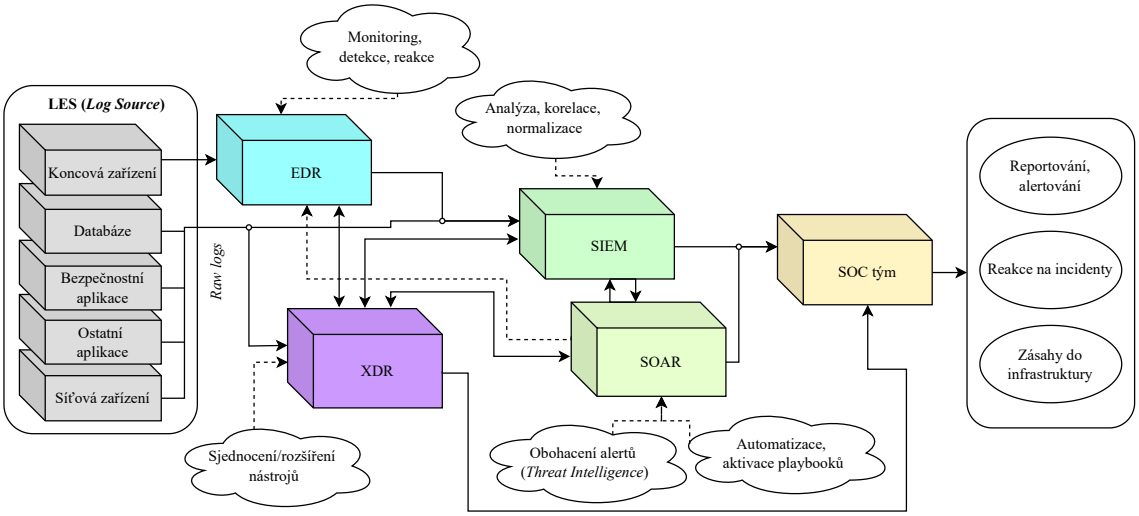
- Jednotlivé bezpečnostní prostředky (antivirus, firewall, apod.),
- Řešení pro jednotlivé vrstvy zabezpečení (EDR, platformy pro ochrany koncových bodů, analýza síťového procesu, atd.),
- Řešení shromažďující data a koordinující pracovní postupy napříč vrstvami zabezpečení, včetně SIEM nebo SOAR.

Porovnání EDR a XDR

XDR je stejně jako EDR řešení založené na analytických technologiích a umělé inteligenci pro detekci hrozeb v podnicích. Od EDR se tyto nástroje odlišují rozsahem ochrany, kterou nabízejí, a způsobem, jakým jsou poskytovány.

XDR integruje bezpečnostní nástroje napříč celou hybridní infrastrukturou organizace, které tak mohou spolupracovat a vzájemně koordinovat prevenci, detekci a reakci na počítačové hrozby. Stejně jako EDR využívá XDR integrace s technologiemi SIEM, SOAR a dalšími funkcemi spojenými s kybernetickou bezpečností. Díky stálému vývoji technologie XDR má potenciál výrazně zefektivnit přetížená SOC sjednocením bezpečnostních kontrolních bodů, telemetrie, analýzy a operací do jednoho centrálního systému [18].

Možnou vzájemnou provázanost jednotlivých nástrojů SIEM, SOAR, EDR a XDR znázorňuje schéma 1.5. Zatímco primárním zdrojem dat z koncových zařízení, vykonávajícím základní detekční a reaktivní úlohy, je EDR nástroj, SIEM je schopen tato data v širším kontextu korelovat a analyzovat. SOAR, často ve spolupráci s EDR, zajistí automatizovanou reakci na incidenty. Následnou možnost propojení informací z těchto systémů nabízí systém XDR, který poskytne jednotný přehled a konzistentní reakční rozhraní napříč celou infrastrukturou.



Obr. 1.5: Propojení bezpečnostních nástrojů SIEM, SOAR, EDR, XDR.

1.4 Proces reakce na bezpečnostní incidenty

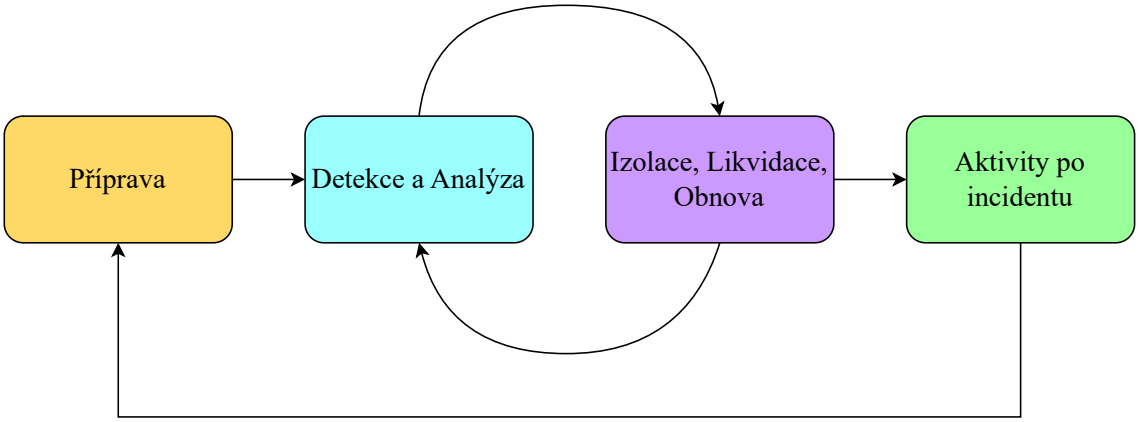
Reakce nebo odezva na bezpečnostní incidenty označuje procesy a technologie organizace pro detekci a reakci na možné kybernetické hrozby, bezpečnostní narušení nebo počítačové útoky. Formální plán reakce na incidenty umožňuje omezit nebo zcela předejít škodám způsobeným těmito bezpečnostními incidenty. Cílem je předejít kybernetickým útokům ještě před jejich vznikem a minimalizovat náklady a narušení provozu v důsledku případných útoků [9, 20]. Bezpečnostní incident nebo událost je jakékoliv digitální nebo fyzické narušení ohrožující důvěrnost, integritu nebo dostupnost informačních systémů organizace nebo citlivých údajů. Mezi nejčastější bezpečnostní incidenty patří např. útok typu *Man-in-the-middle*, *phishing*, *ransomware* nebo DDoS útoky [20].

Hlavními kroky odezvy na bezpečnostní incidenty jsou:

- **Příprava**: Tento krok zahrnuje výběr postupů, nástrojů a technik pro identifikaci, zvládnutí a obnovu po incidentu s minimálním dopadem na provoz. *Computer Security Incident Response Team* (zkráceně CSIRT) je tým, který pravidelně hodnotí možná rizika, identifikuje zranitelnosti a stanovuje priority bezpečnostních incidentů podle jejich potenciálního dopadu na organizaci.
- **Detekce a analýza**: v této fázi CSIRT monitoruje síť pro podezřelé aktivity a analyzuje data z bezpečnostních zařízení a nástrojů (firewally, antivirové programy, atd.), aby identifikoval a odfiltroval skutečné incidenty od falešných. Dále po analýze a určení závažnosti incidentu tým informuje odpovědné osoby a připravuje se na další kroky v procesu reakce na incident.
- **Izolace**: Zde dochází ke krátkodobým opatřením, jako je izolace infikovaných

systémů a zabránění dalšímu šíření hrozby. Mezi dlouhodobé opatření patří například segmentace citlivých databází a posílení ochrany nezasažených systémů. Dochází také k záloze postižených a nepostižených systémů, aby se zabránilo další ztrátě dat a shromáždily se důkazy o incidentu pro budoucí analýzu.

- **Likvidace**: Po odstranění hrozby přechází tým k úplné nápravě, což zahrnuje odstranění malwaru nebo vyřazení neautorizovaných uživatelů a kontrola všech zasažených i nezasažených systémů.
- **Obnova**: Po potvrzení, že byla hrozba eliminována, probíhá obnova všech systémů do běžného provozu, včetně nasazení záplat a obnovy dat ze záloh.
- **Aktivity následující po incidentu**: Po úspěšném zotavení jsou veškeré záznamy o útoku uchovány pro účely analýzy a vylepšení systému.



Obr. 1.6: Kroky reakce na bezpečnostní incident. [9]

1.5 Vylepšení monitorování bezpečnosti pomocí metod strojového učení

V kapitole 1.3 byly popsány nástroje SIEM, SOAR, EDR a XDR, umožňující bezpečnostní monitorování a reakci na kybernetické hrozby a incidenty. Nasazení těchto nástrojů s sebou ale také přináší řadu výzev a omezení. Jednou z těchto klíčových výzev v oblasti kybernetické bezpečnosti představuje integrace různorodých datových zdrojů do systémů SIEM. Tyto systémy musí v současných hybridních a cloudových prostředích agregovat, normalizovat a analyzovat obrovské množství dat z různých zdrojů, včetně síťových logových záznamů, protokolů, firewallů a bezpečnostních nástrojů třetích stran. Přestože existují definované normy, protokoly anebo formáty, viz kapitola 1.1, některé organizace stále využívají různé proprietární formáty protokolů. Rostoucí objemy dat mohou následně zahltit systémové zdroje a snížit tak

výkon při zajištění zpracování a škálovatelnosti v reálném čase [4].

Tento proces je zatížen řadou výzev a problémů, které mohou být rozděleny do několika oblastí, a jejich shrnutí lze vidět v tabulce 1.1 níže.

Oblast	**Hlavní výzvy**
Heterogenita dat	Rozmanitost formátů záznamů, různé typy dat, složitá korelace
Objem a rychlost dat	Obrovské množství dat, nároky na zpracování v reálném čase
Kvalita a integrita dat	Neúplná nebo chybná data, nepřesné údaje, nedostatek kontextu
Kompatibilita systémů	Starší systémy, proprietární formáty, nesourodá časová razítka
Škálovatelnost	Neustálý růst počtu zdrojů a dat
Korelace a analýza dat	Složitost korelačních pravidel, vysoký počet falešných pozitiv/negativ, rozdílná úroveň detailů
Náklady a zdroje	Vysoké provozní náklady a nároky na personál a údržbu
Bezpečnost a soukromí	Práce s citlivými daty a jejich šifrování, řízení přístupu k systémům
Dodržování právních předpisů	Dodržování norem a předpisů, auditovatelnost a dohledatelnost dat
Řízení výstrah a incidentů	*Alert fatigue*, efektivní prioritizace a automatizace
Vývoj a přizpůsobivost	Nové typy hrozeb a zranitelností, potřeba neustálé údržby a aktualizací

Tab. 1.1: Výzvy při integraci dat do SIEM [4].

Tyto výzvy poukazují na to, že efektivní využití SIEM systémů není pouze otázkou správného nasazení nástroje, ale také zahrnuje strategické plánování, pravidelnou údržbu a optimalizaci, a dále integraci s pokročilými technikami, jako je strojové učení, které napomáhá s automatizací a zvyšuje přesnost detekce [4]. Využití strojového učení může zvýšit účinnost při detekci hrozeb, snížit závislost na manuálně definovaných pravidlech a zkrátit dobu mezi detekcí hrozby a reakcí na ni. AI modely přinášejí možnost lepšího porozumění kontextu jednotlivých záznamů, na druhou stranu se zde objevují otázky a výzvy ohledně vysvětlitelnosti modelů,

jejich trénování na dostatečně kvalitních datech a následné integrace do bezpečnostních procesů.

V návaznosti na předchozí kapitoly se bude dále tato práce věnovat teoretickému rozboru strojového učení, jeho klíčovým vlastnostem a problémům. Důraz bude kladen na zpracování přirozeného jazyka a jaké možnosti přináší v kontextu rozšíření logových záznamů, jakožto jeden z perspektivních přístupů ke zlepšení analýzy logů a jejich rozšíření. Následně bude v rámci praktické části navržen a vytvořen vlastní generativní model umožňující aplikovat metodu *Masked Language Modeling* nebo *Next Word Prediction*, za účelem rozšíření datové sady bezpečnostních dat.

Takto rozšířená datová sada bude dále využita k testování v rámci klasifikační úlohy zpracování přirozeného jazyka a rozpoznání pojmenovaných entit v logových záznamech. Cílem bude vyhodnotit, zda je datová sada dostatečně kvalitní k tomu, aby byl model schopen správně identifikovat a přiřazovat specifické entity, jako jsou IP adresy, uživatelská jména, čísla portů a další. Pro tuto část bude využit model `T5`, který bude rovněž dále v práci detailněji popsán, a jehož schopnosti adaptace na různé NLP úlohy umožní objektivní vyhodnocení přínosu navrženého rozšíření.

2 Problematika strojového učení a umělé inteligence

Obor umělé inteligence je sám o sobě velmi rozsáhlý a zahrnuje různé přístupy k automatizaci a zvýšení efektivity různých technických odvětví. Tato kapitola je zaměřena na popis základních konceptů a principů v oblasti strojového učení a umělé inteligence. Jejím cílem bude poskytnout teoretické základy pro pochopení aplikovaných metod a technik v této práci, také přiblížit hlavní přístupy a architektury tvořící základ moderních systémů umělé inteligence. Na začátek budou popsány základní vlastnosti neuronových sítí a jejich struktury, které umožňují řešení složitých úloh. Následně budou popsány zásadní problémy a řešení v oblasti strojového a hlubokého učení. Dále bude přiblížen význam techniky zpracování přirozeného jazyka a úkoly souvisejícími s touto metodou, poté bude vysvětlena role pokročilé architektury `Transformer` v této oblasti. Kapitola bude uzavřena popisem hodnocení výkonnosti trénovacích procesů strojového učení a následné možnosti aplikace umělé inteligence pro rozšíření textových dat.

2.1 Vlastnosti neuronových sítí

Neuronové sítě jsou výpočetní rámce, které jsou modelovány podle struktury a chování biologických neuronů. Tyto rámce umožňují strojům optimálně přistoupit k řešení složitých problémů prostřednictvím adaptivního učení. Neuronové sítě dále pomáhají zefektivnit procesy a zvýšit efektivitu v různých odvětvích a jako páteř umělé inteligence jsou i nadále základním článkem inovací, které utvářejí budoucnost technologií [21, 22]. Neuronové sítě jsou schopny učení a identifikování vzorů přímo z dat bez předem definovaných pravidel. Klíčové komponenty neuronových sítí jsou [21]:

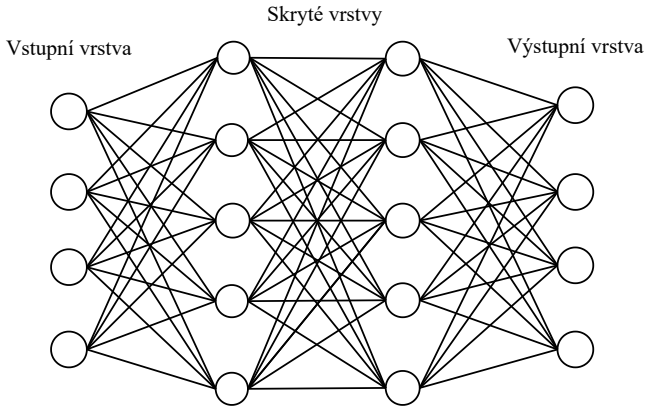
- **Neurony**: představují základní komponenty, které zpracovávaní vstupní data. Každý jeden neuron se řídí prahovou hodnotou a aktivační funkcí.
- **Spoje mezi neurony**: spojení, která přenáší informace a jsou řízeny váhami a zkresleními.
- **Váhy a zkreslení**: těmito parametry je určena síla a vliv spojení neuronových sítí,
- **Funkce šíření**: tento mechanismus pomáhá zpracovat a přenášet data napříč vrstvami neuronů.
- **Pravidla učení**: tyto pravidla určují metody, které upravují váhy a zkreslení v průběhu času za účelem zlepšení přesnosti.

Neuronové sítě nabízí efektivitu během přizpůsobování se k různým úkolům. Pokud je neuronová síť vystavena souboru dat nebo simulovanému scénáři, váhy nebo

zkreslení jsou aktualizovány v reakci na tato nová data nebo podmínky a s každou úpravou se sítě nadále vyvíjejí [21, 22]. Díky těmto schopnostem jsou vhodné pro učení se z velkého množství dat, což umožňuje rozvoj aplikací založených na zpracování přirozeného jazyka (NLP), automatizované odpovědi a podobně [21]. Architektura neuronových vrstev se skládá ze tří vzájemně propojených typů vrstev, což znázorňuje obrázek 2.1:

1. **Vstupní vrstva**: Tato vrstva slouží k příjmu vstupních dat. Každému prvku ve vstupních datech přísluší jeden vstupní neuron ve vrstvě.
2. **Skryté vrstvy**: Tyto vrstvy slouží primárně k provádění výpočetních prací. Síť může mít jednu nebo více těchto vrstev a každá vrstva je složena z jednotek (neuronů), které transformují vstupní data do formy, kterou je schopna zpracovat výstupní vrstva.
3. **Výstupní vrstva**: Tato poslední vrstva utváří výstup modelu. Formát výstupu je závislý na konkrétní úloze.

Hloubka učení je poté definována počtem skrytých vrstev a neuronů, přičemž tato struktura může být různá v závislosti na architektuře vyvinuté pro dané aplikace [22].



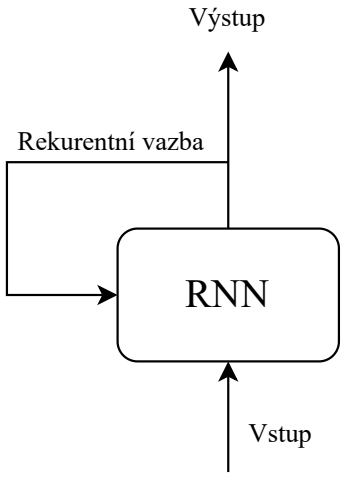
Obr. 2.1: Architektura neuronové sítě.

Neuronovou síť lze rozdělit do několika druhů. Velmi rozšířeným typem sítě jsou takzvané Rekurentní Neuronové Sítě (*Recurrent Neural Networks*, RNN), dále Konvoluční Neuronové Sítě (*Convolutional Neural Network*, CNN) nebo Dopředné Neuronové Sítě (*Feedforward Neural Network*, FNN).

Rekurentní Neuronová Síť

Rekurentní neuronové sítě (RNN) jsou navrženy pro zpracování sekvenčních dat. Tento typ sítě zpracovává sekvence tak, že iteruje přes jednotlivé prvky sekvence

a udržuje vnitřní stav, který obsahuje informace spojené s dosud zpracovaným vstupem. RNN tedy obsahuje vnitřní smyčku, viz obrázek 2.2. Stav sítě se pak resetuje po zpracování dvou různých, na sobě nezávislých sekvencí, čímž se zabrání ovlivnění dalšího vstupu [23]. Podstatné informace ze vstupních sekvencí ukládá takzvaný skrytý stav (stav paměti), který se tak stává charakteristickým rysem RNN. Díky této schopnosti jsou vysoce efektivní pro sekvenční úlohy [24].



Obr. 2.2: Skladba rekurentního neuronu.

Existují čtyři základní typy RNN sítí, které jsou závislé na počtu vstupních a výstupních sekvencí v síti [24]:

- *One-to-One* RNN: nejjednodušší architektura neuronové sítě, kdy existuje jeden vstup, kterému odpovídá jeden výstup. Tento typ se využívá často pro klasifikační úlohy, prvky kde vstupních dat nejsou závislé na předchozích.
- *One-to-Many* RNN: tato síť zpracovává jeden vstup a v průběhu času vytváří více výstupních sekvencí. Tento přístup je výhodný použít v situaci, kdy má jediný vstupní prvek generovat posloupnost předpovědí. Lze aplikovat např. pro úlohu popisu obrázků, kdy vstupem je jeden obrázek a výstupem je sekvence slov odpovídajících popisu obrázku.
- *Many-to-One* RNN: síť přijímá posloupnost vstupů a v návaznosti na ni generuje jediný výstup. Často se tento typ RNN používá v případě, kdy je potřeba k vytvoření jedné předpovědi celkový kontext vstupní sekvence. Typicky se využívá při analýze sentimentu vstupní sekvence.
- *Many-to-Many* RNN: je zpracovávána posloupnost vstupů, která následně generuje posloupnost výstupů. Typickou aplikací je jazykový překlad, kdy je vstupem posloupnost v jednom jazyce a výstupem odpovídající posloupnost v jazyce druhém.

Jednou z rozšířených variant rekurentní neuronové sítě je tzv. *Long Short-Term*

Memory (LSTM). Tento model řeší problém jediného skrytého stavu RNN sítí tak, že zavádí paměťovou buňku, pracující jako zásobník schopný uchovat informaci po delší dobu. Architektura LSTM buňky je tvořena třemi hradly: vstupní hradlo, zapomínací (*forget*) hradlo a výstupní hradlo. Tato hradla se podílejí na rozhodování o tom, která informace je přidána do paměťové buňky, která je z ní odebrána a která vypuštěna [23, 24].

Konvoluční Neuronová Síť

Pro extrakci příznaků z maticových datových sad, jako např. vizuální datové sady obrázků nebo videí, se převážně využívá konvoluční neuronová síť (CNN). Architektura CNN je složena z několika vrstev. Tyto vrstvy mohou být: vstupní vrstva, konvoluční vrstva, sdružovací (*pooling*) vrstva a plně propojené vrstvy. Konvoluční vrstva aplikuje filtry pro extrakci funkcí na vstupní obraz, *pooling* vrstva snižuje vzorkování obrazu pro snížení výpočtů a plně propojená vrstva provádí konečnou predikci. Pomocí metody zpětného šíření (*backpropagation*) a gradientního sestupu učí síť optimálně aplikovat filtry [25].

Tato práce využívá schopnosti neuronových sítí analyzovat vzory a vztahy pro datovou sadu logových záznamů. Primárně je využita architektura `Transformer`, ta se ve velké míře věnuje úkolům spadajícím do kategorie NLP. Tato architektura a NLP technika budou podrobněji popsány v následujících kapitolách 2.3 a 2.4.

2.2 Klíčové problémy a řešení strojového a hlubokého učení

V této kapitole budou představeny klíčové přístupy a výzvy v oblasti strojového a hlubokého učení, jež tvoří základ moderních systémů umělé inteligence. Budou popsány metody strojového učení a jejich praktické aplikace. Následně bude představeno hluboké učení. Na závěr kapitoly bude popsán koncept přeneseného učení, umožňující opětovné využití natrénovaných modelů pro nové úlohy.

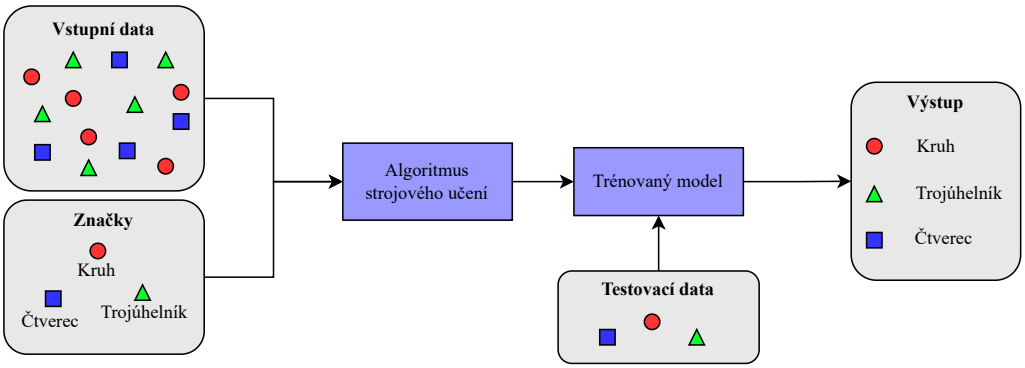
2.2.1 Strojové učení

Strojové učení dnes představuje širokou oblast, která nabízí různé přístupy k řešení specifických úkolů. Mezi typické příklady problémů strojového učení mohou být například binární klasifikace, skalární regrese nebo klasifikace do různých kategorií. Tyto úlohy jsou nejčastěji řešeny technikou tzv. učení s učitelem, jejíž cílem je naučit modely správně mapovat trénovací data na cílové hodnoty. Kromě řízeného

učení zahrnuje strojové učení také další přístupy, jako je učení bez učitele, zpětnovazební učení a samořízené učení. Každý z těchto přístupů bude dále v této kapitole podrobněji popsán [22].

Učení s učitelem

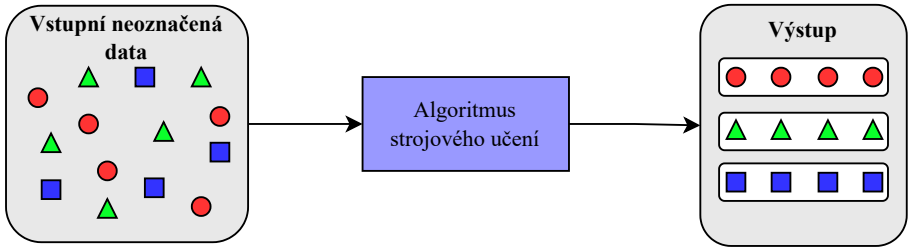
V praxi velmi běžným přístupem strojového učení je učení s učitelem, neboli *Supervised Learning*. Proces trénování vyžaduje vstupní trénovací soubor dat a jemu odpovídající množinu cílových hodnot (anotací). Jako nevýhoda se zde jeví vysoká náročnost na vstupní data, která musí kromě klasických požadavků na existenci trénovací, testovací a validační množiny obsahovat také správně mapované výstupní hodnoty. Přestože tato metoda umožňuje dotrénovat modely s vysokou přesností predikce, kdy pravděpodobnost správné předpovědi cílové hodnoty přesahuje 90%, mohou být tyto požadavky pro některé aplikace nesplnitelné. Mezi běžné aplikace této metody mohou patřit například aplikace typu optického rozpoznání znaků nebo řeči, překladu jazyka, detekce objektů, vytváření sekvencí nebo predikce syntaktického stromu [22, 26].



Obr. 2.3: Technika učení s učitelem.

Učení bez učitele

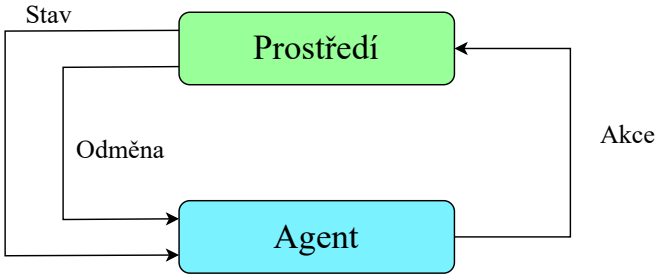
Učení bez učitele (*Unsupervised Learning*) se velmi často vyskytuje v aplikacích, jejichž cílem je vyřešit problém shlukování nebo redukce dimenzionality. Učící se systém má za úkol detekovat vzory, bez jakýchkoli předem existujících anotací nebo specifikací. Existuje tedy pouze vstupní tréninková datová sada, ve které se nevyskytuje žádný řídící prvek, který by dohlížel na řízení mapování vstupních dat na cílové výstupní hodnoty [26]. V tomto případě jsou výhodou menší nároky na vstupní datovou sadu. Příkladem může být technika shlukování pro seskupení zákazníků nebo trhů do jednotlivých segmentů, za účelem komunikace s konkrétnější cílovou skupinou [22, 27].



Obr. 2.4: Technika učení bez učitelem.

Zpětnovazební učení

Tento druh učení je od ostatních typů dost odlišný. Zpětnovazební učení (*Reinforcement Learning*) je ve své podstatě učení na základě interakcí s prostředím. Nejsou k dispozici žádná data ani značení, základem jsou agenti, sledované prostředí a odměna. Agenti jsou umístěni do sledovaného prostředí, ve kterém provádí různé akce, za něž jsou následně odpovídajícím způsobem odměněni. Cílem algoritmů založených na zpětnovazebním učení je potom maximalizovat tuto odměnu, proto se agent v každém kroku učí ze svých interakcí, aby byl schopen vykonat akce, vedoucí k co nejvyšší odměně [28, 29]. Princip metody zpětnovazebního učení je zobrazen na obrázku 2.5.



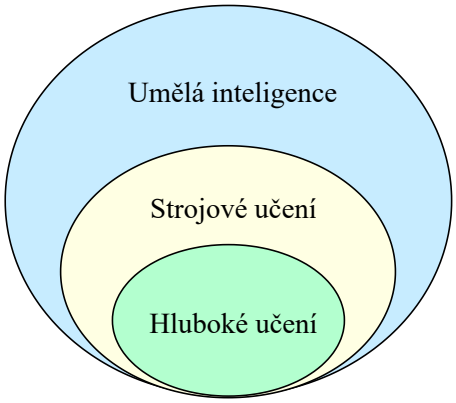
Obr. 2.5: Technika zpětnovazebního učení.

Samo-řízené učení

Specifický případ řízeného učení označovaný jako samo-řízené učení (*Self-Supervised Learning*) pracuje na podobných principech jako učení s učitelem. Existují zde jednotlivé anotace tříd, které jsou ale generovány ze vstupních dat, obvykle pomocí heuritického algoritmu, a následně jsou na tato data mapovány. Tato technika tedy probíhá bez zásahu člověka do procesu učení. Jako typický příklad samo-řízeného učení může být predikce dalšího snímku ve videu na základě předchozích snímků nebo predikce dalšího slova v textu na základě slov předchozích (využívá se zde případ časově řízeného učení - *temporally supervised learning*) [22, 26].

2.2.2 Hluboké učení

Jednou ze specifických podmnožin strojového učení je hluboké učení (*Deep Learning*). Tento přístup klade důraz na učení postupně jdoucích vrstev reprezentací znalostí. Hloubku modelu určuje počet vrstev jednotlivých reprezentací. Moderní systémy využívající hlubokého učení mohou zahrnovat desítky až stovky po sobě jdoucích vrstev reprezentací, a všechny tyto vrstvy jsou učeny na základě vystavení trénovacím datům [23]. Hierarchické postavení hlubokého učení v oboru umělé inteligence a strojového učení lze vidět na obrázku číslo 2.6. Učení vrstvených reprezentací se provádí pomocí neuronových sítí, které byly popsány v kapitole 2.1. S rostoucím počtem skrytých vrstev roste přesnost modelu, ale také celková složitost modelu.



Obr. 2.6: Hierarchie oboru umělé inteligence a umístění hlubokého učení [22].

2.2.3 Přenesené učení

Během procesu trénování neuronových sítí se naskytují dvě možnosti, jak toto trénování pojmout:

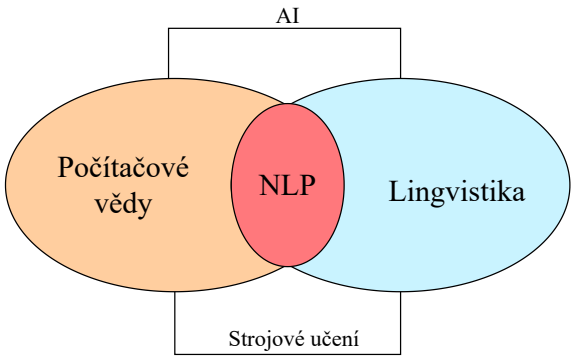
1. **Trénování od nuly**: inicializace neuronové sítě a její trénink (*supervised*) na cílovou úlohu.
2. **Přenesené učení**: síť je předem trénovaná na samostatném souboru dat a následně je doladěna na cílové úloze.

Proces předtrénování jazykových modelů se provádí na datové sadě, která je mnohem větší než následný, cílový soubor dat. Tento proces výrazně zlepšuje efektivitu dat a při následném doladění se model učí rychleji a dosahuje většího procesního výkonu [30]. Proces přeneseného učení se u NLP úloh provádí pomocí *self-supervised* tréninku s neoznačeným textem, např. pomocí modelování maskovaného jazyka (MLM, viz kapitola 3.4) nebo kauzálního modelování jazyka. Ve své podstatě se tedy trénink a učení na jedné úloze přenáší na jinou, zpravidla podobnou oblast,

doménu nebo úkol. Samotným vrcholem přeneseného učení je možnost využití jednoho samostatného modelu pro všechny úlohy zpracování textu, předem trénovaného na směsi úloh bohatých na různá data.

2.3 Zpracování přirozeného jazyka

Jedním z odvětví oboru umělé inteligence, které se zaměřuje na interakci mezi počítači a lidmi, a umožňuje pomocí strojového učení počítačům porozumět lidskému jazyku, a využít jej ke komunikaci, je tzv. *Natural Language Processing* (NLP), neboli zpracování přirozeného jazyka. K analýze, porozumění a tvorbě textu využívá metod strojového učení, čímž hraje NLP důležitou roli v oblasti strojového překladu a dalších aplikacích umožňujících efektivnější komunikaci mezi člověkem a strojem. Obrázek 2.7 ilustruje, jak NLP propojuje obor umělé inteligence a lingvistiky za účelem analýzy a tvorby přirozeného jazyka [31, 32].



Obr. 2.7: NLP jako průnik lingvistiky a informatiky [32].

Logové záznamy mohou poskytovat NLP analýze bohatý zdroj dat, jelikož obsahují různé informace ohledně procesů a událostí probíhajících v síťových zařízeních, viz kapitola 1.1. Schopnosti NLP umožňují identifikovat, klasifikovat a potažmo obohatit jednotlivé entity obsažené v logových záznamech o další informace [33, 34]. Jedna z klíčových úloh NLP je klasifikace textu a ta se v tomto kontextu může jevit jako zásadní. Identifikace významu struktur jednotlivých logových záznamů lze využít v aplikacích jako je detekce bezpečnostních incidentů, diagnostika problémů v reálném čase nebo možná selhání systémů. Tyto přístupy se široce uplatňují ve scénářích, jako je detekce spamů, klasifikace komerčních dokumentů nebo reakce na mimořádné události [34].

Modely hlubokého učení pracují výhradně s číselnými tenzory, nejsou schopny přímo zpracovat surový text, a proto je nutné tento text převést na číselnou reprezentaci prostřednictvím takzvaného procesu *vektorizace*. Tento proces lze provést

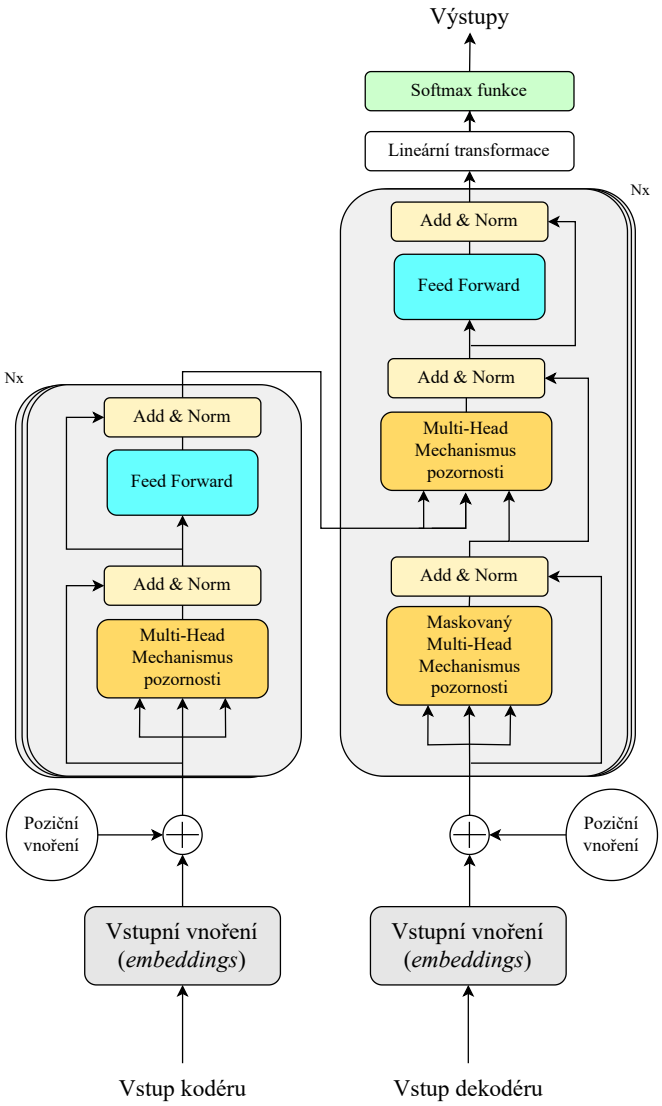
několika způsoby [22]:

- Rozdělení vstupní sekvence na slova a jejich následný převod na vektor,
- Vstupní sekvence je rozdělena na znaky, které se poté převádí na vektory,
- Ze vstupní sekvence jsou extrahovány *n-gramy* (překrývající se skupiny několika po sobě jdoucích znaků nebo slov), které jsou dále vektorizovány.

Tokenizace je proces, během kterého se rozdělí text do tokenů. Tyto tokeny jsou hromadný název pro různé jednotky, do kterých lze v prvních krocích vektorizace rozdělit text (znaky, slova, n-gramy). Tokeny jsou následně mapovány a sdružovány s generovanými vektory a poté jsou zabaleny do formy sekvenčních tenzorů a přiváděny na vstup hlubokých neuronových sítí [22, 23]. Proces mapování tokenů na odpovídající vektory lze uskutečnit několika způsoby. Pro mapování slov se výhradně využívá tzv. *vnoření slov* (*Word Embeddings*) - slovo je vkládáno jako standardní vektor do vektorového prostoru slov. Vnořená slova jsou vektorové reprezentace slov, jejichž účelem je mapovat lidský jazyk do strukturovaného geometrického prostoru. Vektory vnořených slov jsou nízkodimenzionální (tzn. jsou ve formě hustých vektorů) typu *float*, na rozdíl od vektorů získaných např. kódováním 1-z-n (*one-hot encoding*) [23]. Vnoření slov lze získat dvěma způsoby. První způsob pracuje na principu postupného učení se vnoření slov zároveň s modelem, který řeší hlavní úlohu (jako je klasifikace nebo predikce sentimentu). Jelikož má vnoření slov mapovat jazyk do geometrického prostoru, měly by vektory vnoření odrážet sémantický vztah mezi jednotlivými slovy. Jako druhá možnost se naskýtá vložení předtrénovaného vnoření slov (*pretrained word embeddings*) do modelu. Existují různé databáze vnořených slov jako např. `Word2Vec`, `GloVe` (*Global Vectors for Word Representation*) nebo knihovna `FastText` [22, 32].

2.4 Hluboké neuronové sítě typu Transformer

V roce 2017 publikovali vědci ze společnosti Google článek, ve kterém byla navržena nová architektura neuronových sítí pro sekvenční modelování (*sequence modeling*) zvaná `Transformer` [35]. Následně v úlohách strojového překladu tato architektura překonala rekurentní neuronové sítě jak z hlediska kvality překladu, tak z pohledu nákladů na trénování. Modely využívající architekturu `Transformer` jsou trénovány jako tzv. jazykové modely (*Language Models*, LM). Samotné trénování těchto modelů se provádí technikou *Self-Supervised* učení a vyžaduje velké množství nezpracovaného textu. Architektura `Transformer` jako taková je založena na struktuře kodér-dekodér, která se také široce využívá pro strojový překlad, kde je sekvence slov překládána z jednoho jazyka do druhého. Obrázek 2.8 znázorňuje takto vypadající architekturu [35, 36].



Obr. 2.8: Architektura Transformer modelu [35].

`Transformer` architektura tedy využívá kodér i dekodér pro zpracování vstupních a výstupních vnoření slov [35]:

- **Kodér**: Slouží k převodu vstupní posloupnosti tokenů na posloupnost vnořených (*embedding*) vektorů, nazývaných také jako skrytý stav (*hidden state*) nebo kontext. Skládá se z množiny $N = 6$ stejných vrstev. Každá vrstva pak obsahuje dvě podvrstvy:
  - *Multi-Head self-attention* mechanismus (vícevrstvý mechanismus pozornosti),
  - Jednoduchou, pozičně plně propojenou dopřednou síť.

  Kolem každé z obou vrstev je využito residuální spojení a následně provedena normalizace vrstev. Výstup každé z podvrstev je ve tvaru $LayerNorm(x +$

Sublayer(x)), kde *Sublayer(x)* představuje funkci implementovanou danou podvrstvou. Pro usnadnění residuálních spojení vytvářejí všechny podvrstvy v modelu i *embeddings* vrstvy výstupní sekvence o rozměru $d_{\text{model}} = 512$ [35, 36].

- **Dekodér**: Využívá skrytý stav kodéru k iterativnímu generování výstupní posloupnosti jednotlivých tokenů jeden po druhém. Skládá se rovněž jako kodér z množiny $N = 6$ stejných vrstev. Dekodér ale vkládá navíc třetí podvrstvu, která provádí *multi-head attention* mechanismus nad výstupem ze zásobníku kodéru. Pro dílčí vrstvy se podobně jako u kodéru využívá residuálních spojení a po nich následující normalizace. Je upravena *self-attention* podvrstva zásobníku dekodéru tak, aby se zabránilo tomu, že se jednotlivé pozice budou věnovat pozicím následujícím. Nakonec je tedy predikce pro pozici *i* závislá pouze na známých výstupech na pozicích *i-1*, což je zajištěno maskováním a posunutím výstupních vnoření (*embeddings*) o jednu pozici [35, 36].

Původně byla tato architektura navržená pro úlohy typu *sequence-to-sequence*, každopádně bloky kodéru i dekodéru byly velice brzy upraveny jako samostatné modely, a přestože existují stovky různých modelů na bázi `Transformer` architektury, většina z nich spadá do jednoho ze tří následujících typů modelů [36]:

1. ***Encoder-only***:
   Tyto modely jsou vhodné především pro úkoly jako je klasifikace textu nebo rozpoznání pojmenovaných entit. Pracují na principu převodu vstupní sekvence textu na bohatou numerickou reprezentaci, která je pro daný token závislá jak na levém (před tokenem), tak na pravém kontextu (za tokenem). Tento princip se také nazývá *Bidirectional attention*. Do této třídy mohou spadat například modely `BERT` a jeho varianty, jako je `RoBERTa` nebo `DistilBERT`.
2. ***Decoder-only***
   Tyto modely automaticky doplňují vstupní sekvenci postupným předpovídáním nejpravděpodobnějšího dalšího slova na základě vstupní textové výzvy (*prompt*). Výpočet numerické reprezentace tokenů je zde závislý pouze na levém kontextu (před tokenem). Tato metoda se často nazývá kauzální nebo autoregresivní pozornost. Do této třídy spadá řada modelů `GPT`.
3. ***Encoder-decoder***
   Tato architektura je využívána k modelování komplexních mapování jedné sekvence textu do druhé. Jsou tedy vhodné pro sumarizaci nebo úlohy strojového překladu. Kromě samotné architektury *Transformer*, která kombinuje kodér a dekodér, mohou do této skupiny patřit modely `BART` nebo `T5`.

Takto rozdělené modely nabízí širší možnosti aplikací NLP, v jejichž jádru leží *sequence-to-sequnce* úloha. Tyto aplikace mohou zahrnovat kromě klasifikace textu a strojového překladu také úlohy sumarizace textu, generování textu, odpovědi

na otázky v kontextu s textem *question answering* apod.

Vzhledem k úspěchu `Transformer` modelů u široké veřejnosti se v posledních letech objevilo rozsáhlé množství různých modelů, které na tuto architekturu navazují a dále ji rozvíjejí. Jak již bylo zmíněno, tyto modely poté nacházejí uplatnění v široké škále úloh. V kontextu této práce, jejíž hlavním zaměřením je rozšíření textových dat, jsou klíčové zejména modely určené pro úlohy generování textu a vyplňování masek. Vybrané modely pro zmíněné úlohy, které ovlivnily vývoj v oblasti zpracování přirozeného jazyka a jejichž vlastnosti jsou relevantní pro účely této práce, budou představeny a popsány dále v kapitole 3.4.

2.5 Možnosti hodnocení tréninkového procesu strojového učení

Průběžné vyhodnocení výkonnosti trénovaných modelů založených na neuronových sítích je nezbytné nejen z hlediska jejich optimalizace, ale také z pohledu jejich přenositelnosti a generalizace na dosud neviděná data. Pokud jsou sledované metriky vhodně zvoleny, umožňují tak lepší pochopení, zda se model učí efektivně, zda nedochází k přeučení (*overfittingu*) a zda má model potenciál zlepšit se s větším množstvím dat nebo s upravenými a zlepšenými hyperparametry tréninku.

Velmi často využívané možnosti hodnocení trénování jazykových modelů jsou tréninková ztráta, validační ztráta nebo perplexita. Ztráta při trénování, ztráta při validaci nebo při vyhodnocení se využívá k popisu různých ztrátových funkcí, které jsou použity během trénování a vyhodnocení modelu.

Tréninková ztráta je počítána během samotného procesu trénování a využívá se k aktualizaci parametrů modelu pomocí technik, jako je gradientní sestup, za účelem minimalizace této ztráty. Je tedy měřítkem toho, jak dobře si model strojového učení vede na tréninkových datech. V průběhu času by se tedy tato ztráta měla snižovat, každopádně velmi nízká tréninková ztráta nemusí nutně znamenat, že bude model dobře fungovat na nových, neznámých datech, jelikož mohl nadměrně přizpůsobit tréninková data (tzv. *overfitting*) [37, 38]. Jako běžné tréninkové funkce se využívá například Střední Kvadratická Chyba (MSE, *Mean Squared Error*) pro regresní úlohy nebo Ztrátová Křížová Entropie (*Cross-Entropy Loss*) pro klasifikační problémy. Ztráta křížové entropie pro jeden datový bod i je definována vztahem 2.1, kde y_i vyjadřuje pravdivou značku (0 nebo 1 pro binární klasifikaci, nebo jedno-číselný zakódovaný vektor pro klasifikaci více tříd) a p_i předpovězenou pravděpodobnost pozitivní třídy [37, 39].

$$L_{\text{train}}(i) = -\left[y_i \log p_i + (1 - y_i) \log(1 - p_i)\right] \tag{2.1}$$

Celková ztráta během procesu trénování se zpravidla počítá jako průměr ztrát napříč všemi datovými body použitými v trénování, a je definována rovnicí 2.2 [37].

$$L_{\text{train}} = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} L_{\text{train}}(i) \tag{2.2}$$

Měřítkem toho, jak dobře si vede model strojového učení na validačním souboru dat, je ***validační ztráta***. Během trénování je část dat vyčleněna pro validaci a nevyužije se tedy pro trénink. Po stanovené době (například po každé epoše) se na této množině výkon modelu pravidelně vyhodnocuje a vypočítává se validační ztráta, která pomáhá posoudit, jak dobře model zobecňuje na data, se kterými se během trénování nesetkal. Je využita ke sledování výkonu modelu a k odhalení *overfittingu* [37]. Růst validační ztráty při současném poklesu trénovací ztráty je jeden z hlavních indikátorů přeučení modelu. Validační ztráta se počítá podobně jako tréninková ztráta pomocí ztrátové funkce křížové entropie a pro každý datový bod se vypočítá stejným způsobem jako pro tréninková data. Průměr ztrát ze všech validačních datových bodů je popsán rovnicí 2.3 a značí se jako validační ztráta ($L_{\text{validation}}$) [37, 38].

$$L_{\text{validation}} = \frac{1}{N_{\text{validation}}} \sum_{i=1}^{N_{\text{validation}}} L_{\text{validation}}(i) \tag{2.3}$$

Poté, co je model natrénován a jeho hyperparametry byly vyladěny na základě tréninkového a validačního výkonu, je model vyhodnocen na testovací množině za účelem posouzení, jak dobře bude fungovat v reálných scénářích. Zde je měřítkem výkonnosti modelu na zcela nové, oddělené množině dat tzv. ***vyhodnocovací ztráta***. Tato ztráta je vypočítána pomocí křížové entropie a pro každý datový bod v testovací množině se ztráta počítá stejným způsobem jako při trénování a validaci. Vyhodnocovací ztráta ($L_{\text{evaluation}}$) je vyjádřena rovnicí 2.4 a je dána jako průměr ztrát ze všech testových bodů [37].

$$L_{\text{evaluation}} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} L_{\text{evaluation}}(i) \tag{2.4}$$

Jako dalším měřítkem hodnocení jazykových modelů je takzvaná ***perplexita***, která představuje jistou míru zmatenosti modelu. Obecně lze říci, že perplexita udává, jak přesně pravděpodobnostní model předpovídá daný vzorek, a v kontextu zpracování přirozeného jazyka je jedním ze způsobů hodnocení jazykových modelů [40].

Je potřeba zmínit, že je tato metrika dobře definovaná pro jazykové modely autoregresivní nebo kauzální, tedy pro klasické modely jako jsou GPT, pro maskované jazykové modely mohou být výsledky mírně zavádějící [41]. Perplexitu lze definovat jako exponenciální hodnotu průměrné záporné logaritmické pravděpodobnosti sekvence tokenů. Pro konkrétní tokenizovanou sekvenci $X = (x_0, x_1, \ldots, x_t)$ se perplexita určuje podle rovnice 2.5 následovně:

$$\mathrm{PPL}(X) = \exp\left\{-\frac{1}{t}\sum_{i=1}^{t}\log p_t\left(x_i \mid x_{i-1}\right)\right\} \tag{2.5}$$

kde $\log p_t(x_i \mid x_{i-1})$ představuje logaritmickou pravděpodobnost i-tého tokenu podmíněnou předchozími tokeny podle modelu. Perplexitu lze tedy chápat jako míru schopnosti modelu předpovídat následující token v sekvenci. Postup tokenizace má tedy přímý vliv na perplexitu modelu, což je potřeba zohlednit při porovnávání různých modelů. Více podrobnějších detailů viz [41].

Ve strojovém učení je dále velmi důležité kontrolovat přesnost a výkonnost modelů. Jednou z nejpoužívanějších metrik využívaných k hodnocení v klasifikačních úlohách[1] je takzvané ***F1 skóre***. Tato metrika spojuje ***přesnost*** (*precision*) a ***citlivost*** (nebo míru nalezených relevantních případů, angl. *recall*), a poskytuje jejich vyváženou hodnotu, zejména pokud je rozložení tříd nevyvážené. F1 skóre je odvozeno z harmonického průměru přesnosti a citlivosti [42], viz rovnice 2.8. Pro správnou interpretaci F1 skóre je tedy důležité těmto dvěma metrikám správně porozumět a jejich definice jsou uvedeny v rovnicích 2.6 a 2.7:

- ***Přesnost*** (*Precision*) – vyjadřuje míru správnosti pozitivních předpovědí a jedná se o podíl správně identifikovaných pozitivních předpovědí (*True Positives*) z celkového počtu všech pozitivních předpovědí modelu (*True Positives* + *False Positives*) [42].

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.6}$$

- ***Citlivost*** (*Recall*) – v tomto případě se jedná o poměr skutečně pozitivních případů, které byly modelem správně identifikovány, k celkovému počtu skutečných pozitivních případů [42].

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.7}$$

F1 skóre je tedy užitečné v případě, kdy je potřeba vyvážit obě metriky, a zajišťuje tak metodu pro hodnocení modelu, který není příliš nakloněn ani falešně pozitivním, ani falešně negativním chybám. Harmonický průměr následně pomáhá kombinovat přesnost a citlivost tak, že zprůměruje jejich vzájemné hodnoty a díky

[1] Výkon klasifikátoru lze shrnout do čtyř kroků [42]:

- Pravdivé pozitivní výsledky (TP): Správně předpovězené pozitivní případy.
- Falešně pozitivní (FP): Nesprávně předpovězené pozitivní případy.
- Pravdivě negativní (TN): Správně předpovězené negativní případy.
- Falešně negativní (FN): Nesprávně předpovězené negativní případy.

kombinaci výhod obou metrik poskytuje F1 jedinou hodnotu, která reprezentuje celkovou výkonnost modelu [42].

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.8}$$

Kromě výše popsaných metrik je také nezbytné, zejména u generativních modelů, zahrnout i evaluaci lidskou. Tato subjektivní metoda umožní posoudit kvalitu vygenerovaného textu z hlediska jeho plynulosti, srozumitelnosti a relevance, což jsou aspekty, které kvantitativní metriky nemusí plně zachytit.

Sledování a řízení tréninkového procesu

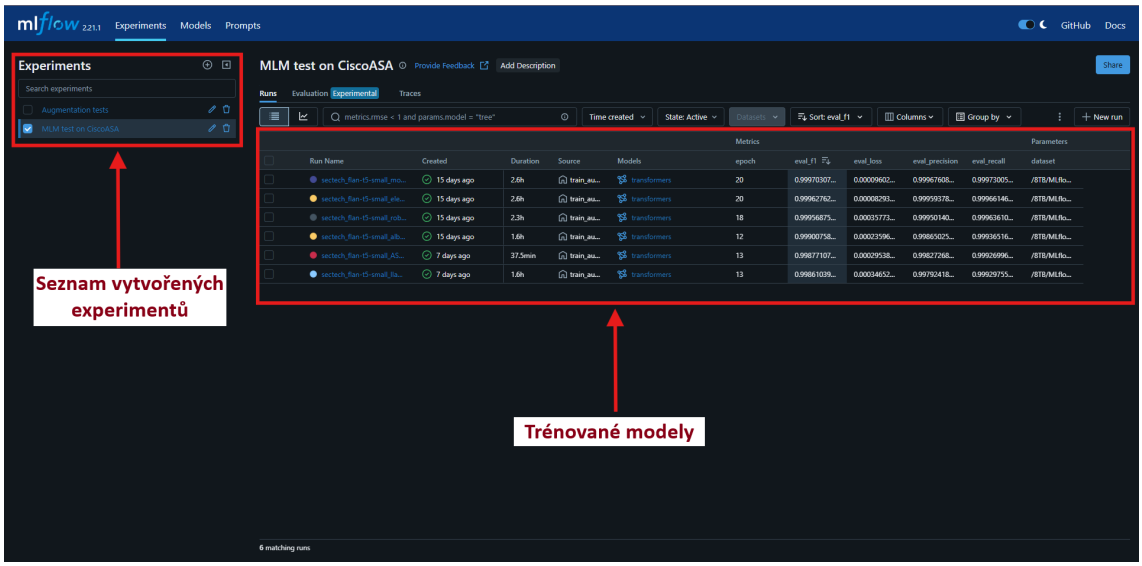
V současné době, jako reakce na potřebu efektivního řízení strojového učení, existuje již celá řada metod a nástrojů, které usnadňují hodnocení procesu učení a jeho výsledků. Mezi tyto nástroje patří například *open-source* platforma `MLflow`, která nabízí efektivní správu a monitoring celého procesu vývoje modelů, od úvodních experimentů až po jejich nasazení a pozorování v reálném provozu.

Tento *open-source* nástroj umožňuje zaznamenávat a sledovat parametry, metriky a artefakty, které jsou spojeny s experimenty strojového učení. Nabízí se jako centrální úložiště pro správu modelů a jejich verzí. Dále také poskytuje možnost ukládání a nasazení modelů na různé platformy [43].

Mezi jeho výhody patří:

- Kompatibilita s *frameworky* jako jsou například `TensorFlow`, `PyTorch`, `Scikit-learn` nebo další.
- Jednoduché nasazení lokálního serveru pomocí `Docker` kontejneru.
- Přehledné uživatelské rozhraní a možnost logování metrik jako je F1-skóre nebo přesnost a jejich následná vizualizace.

Přehlednost a jednoduchost uživatelského rozhraní může být na druhou stranu nevýhodou pro uživatele, kteří hledají nástroj, poskytující pokročilejší funkce nebo vizualizaci grafů. Jako alternativa `MLflow` se jeví nástroj `ClearML` [44]. Tento nástroj obsahuje podobné funkce jako výše popsaný `MLflow`, jeho výhodami jsou nejen bohatší a intuitivnější webové rozhraní, ale také větší automatizace, čímž se minimalizuje potřeba manuálního logování experimentů. Pro potřeby této práce však postačí nástroj `MLflow`, který se jeví jako jednodušší a intuitivnější. Ukázka grafického rozhraní tohoto nástroje je uvedena na obrázku 2.9. V přiloženém obrázku je zobrazen experiment *MLM test on CiscoASA*, který obsahuje šest instancí modelu `flan-T5-small` natrénovaných na testovací datové sadě, více viz praktická část v kapitole 4.3. Stahování těchto modelů je realizováno pomocí jednoduchého skriptu, jehož ukázka je uvedena v kapitole 4.1.1, konkrétně na obrázku 4.3.



Obr. 2.9: Grafické uživatelské rozhraní `MLflow` nástroje

2.6 Aplikace metod strojového učení pro rozšíření záznamů událostí

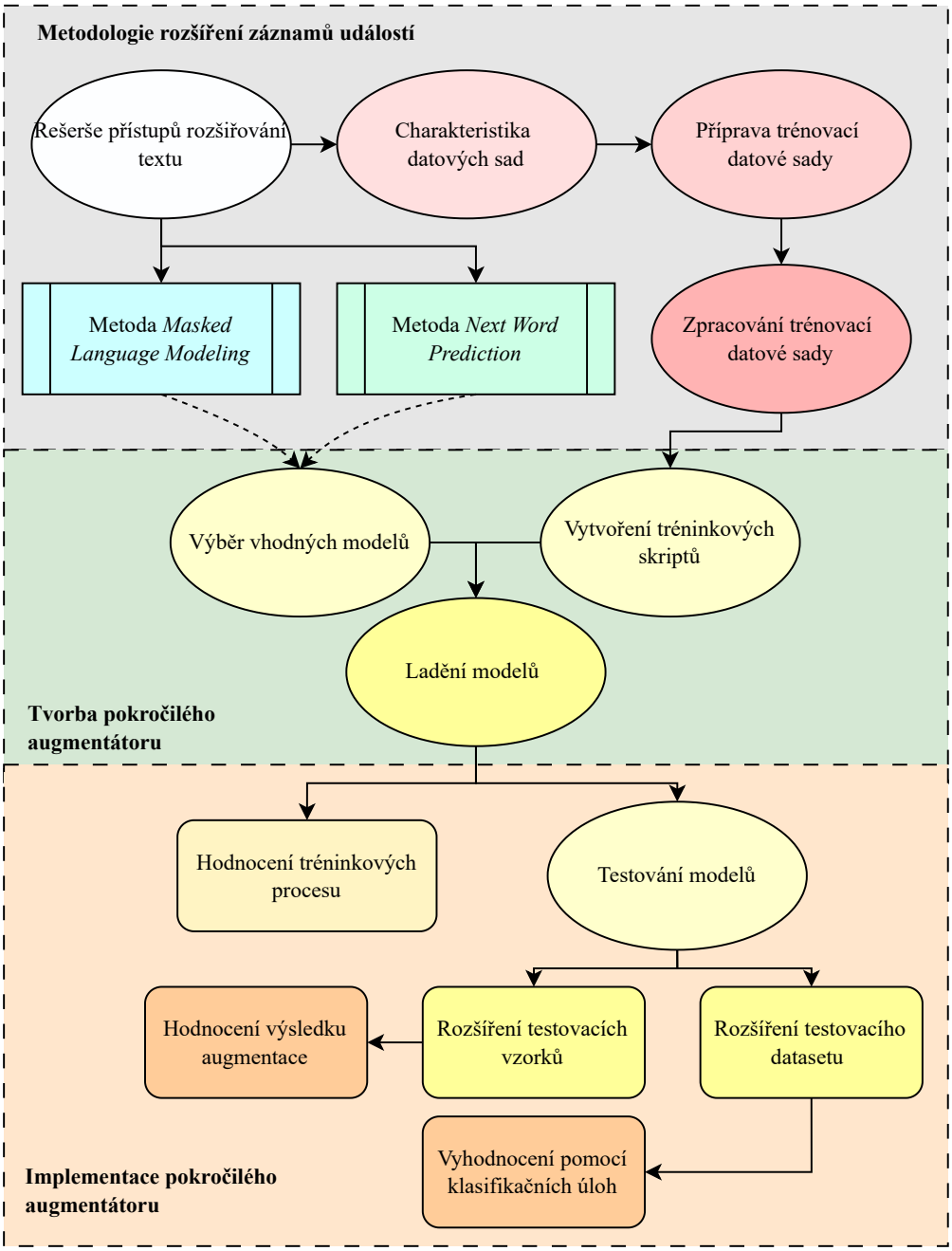
Teoretické znalosti týkající se vlastností neuronových sítí, architektur typu `Transformer` a principů zpracování přirozeného jazyka tvoří pevný základ pro návrh metodiky využití strojového učení a umělé inteligence k rozšiřování záznamů událostí. Motivací tohoto přístupu je snaha překonat omezení spojená s nedostatkem kvalitních trénovacích dat, nerovnováhou datových tříd a nízkou rozmanitostí vzorků, což může následně negativně ovlivňovat generalizační schopnosti modelů pro navazující úlohy, jako je klasifikace a rozpoznání entit.

V kontextu bezpečnostních záznamů lze mezi klíčové úlohy zpracování přirozeného jazyka mimo jiné zařadit analýzu a kategorizaci entit v protokolech, extrakci smysluplných entit, odhalování vzorců svědčících o bezpečnostních incidentech nebo korelaci událostí v různých lozích a tím identifikovat složité, vícestupňové útoky [33]. Automatizace, kterou NLP poskytuje, tak snižuje nejen pracovní zátěž bezpečnostních analytiků, ale také rychlost a přesnost detekce hrozeb. To umožní reagovat na potenciální bezpečnostní hrozby v reálném, nebo téměř reálném čase [33].

Klíčovým přístupem této práce je využití pokročilých technik rozšiřování logů a využití generativních technik, konkrétně generativních metod založených na *Masked Language Modeling* nebo *Next Word Prediction*. Cílem těchto metod bude doplňování maskovaných částí textu v logových záznamech. Tímto způsobem lze generovat syntetická data, která obohatí původní soubor dat, a také přispívají ke zlepšení výkonnosti a robustnosti detekčních modelů v praktickém nasazení.

Kvalitu rozhodování modelů a ovlivnění interpretace bezpečnostních záznamů přímo ovlivní úloha rozpoznání pojmenovaných entit (NER), která bude zahrnuta do testování jednotlivých modelů. NER je velmi cenná při identifikaci různých entit z textových zdrojů, které lze následně analyzovat pro získání dalších informací. Tato úloha bude detailněji představena a rozebrána v kapitole 4.3

Obrázek 2.10 schematicky znázorňuje, že celý tento proces lze rozdělit do tří fází. V první fázi se věnuje metodologii rozšiřování textu a záznamů událostí. V této části byla provedena rešerše dostupných metod rozšiřování textu, návrh vhodného přístupu strategie a následná příprava trénovacích datových sad. Druhá fáze se věnuje tvorbě tréninkových skriptů pro zvolené modely, využívající vybrané techniky generování textu a následný trénink modelů na připravených datových sadách. V třetí fázi budou tyto natrénované modely otestovány na vybraných testovacích vzorcích záznamů událostí a zároveň bude provedeno rozšíření testovací datové sady, která poslouží pro validaci modelu `T5` při řešení syntaktické analýzy logových záznamů. Tento přístup propojuje teoretický rámec představený v předchozích kapitolách s praktickou metodikou implementace pokročilého augmentátoru, která bude podrobněji rozebrána v následujících kapitolách 3 a 4.



Obr. 2.10: Proces aplikace metod strojového učení pro rozšíření záznamů událostí.

3 Metodologie rozšíření záznamů událostí a tvorba augmentátoru

Cílem této kapitoly je objasnit postup a přínos procesu rozšiřování dostupných datových sad pomocí pokročilého augmentátoru. Augmentace, neboli rozšíření textu nebo datové sady, je důležitý aspekt NLP k vytvoření nového, umělého korpusu. Tento vytvořený korpus lze dále využít k rozšíření tréninkových datových sad za účelem zvýšení výkonnosti modelů pracujících na NLP úkolech. Cílem augmentace je tedy generovat logické, rozmanité a sémanticky koherentní vzorky dat, které efektivně zajistí jak kvalitu, tak kvantitu tréninkových dat. Lze implementovat různé metody rozšíření těchto logových záznamů, ale kvůli jejich specifické struktuře mohou některé techniky augmentace selhávat. Trénování jazykových modelů v tomto ohledu přináší potenciálně příznivé výsledky, je ale důležité jednotlivé modely správně naučit a nasměrovat. Dále v této kapitole budou přiblíženy současné techniky a možné metody rozšiřování dat, následný popis využitých datových sad za účelem trénování modelů a jejich dalšímu testování. Dále bude rozebrán postup zpracování surového textu datových sad, popis technik rozšíření textu sítí `Transfomer` a výběr metod a odpovídajících modelů pro učení na datové sadě logových záznamů. Výstupem této kapitoly budou jazykové modely naučené na datové sadě logových záznamů, jejichž cílem je pracovat jako augmentátor maskovaných entit v záznamech událostí.

3.1 Současné přístupy k rozšiřování textu

S cílem zlepšit generalizační schopnosti jazykových modelů v oblasti zpracování přirozeného jazyka, zejména pak v případech s omezenými, nevyváženými nebo nekvalitními trénovacími sadami, se stále častěji využívá různých technik augmentace dat. Původně byly tyto techniky využívány především pro běžné textové domény, jako jsou články, recenze, dialogy nebo sociální sítě. V souvislosti s rychlým rozvojem umělé inteligence a jejími aplikacemi v průmyslové praxi tak nachází augmentace uplatnění i v technických oblastech, kde napomáhá budování robustních modelů schopných zpracovat složitější a specializované texty s následným vylepšením nástrojů a technik, umožňujících pracovníkům automatizovat úlohy, jak již bylo zmíněno v kapitole 1.5.

Cílem této kapitoly je představit vybrané přístupy k rozšíření nejen textových dat a provést stručnou analýzu stavu výzkumu a technologií v této doméně a vyhodnotit jejich silné a slabé stránky z pohledu možné aplikace na záznamy událostí. V následující části budou postupně popsány základní heuritické metody, distribuované reprezentace a pokročilé přístupy založené na jazykových modelech. Pozornost

bude primárně kladena na jejich využitelnost pro rozšíření logových záznamů, které budou sloužit jako trénovací data pro modely strojového učení za účelem syntaktické analýzy událostí.

3.1.1 Heuritické přístupy k rozšíření textu

Mezi jednoduché a často využívané metody rozšiřování textových dat, uskutečňující základní úpravy původního textu s cílem vytvářet nové syntetické texty, patří techniky známé jako *Easy Data Augmentation* (EDA). Tyto rozšiřovací metody poskytují přímé operace s textem na úrovních znaků, slov, vět či dokumentů.

Augmentace na znakové úrovni pracuje na základě zavádění znaků v náhodných textových pozicích. Využívá se technika zvaná *Random Noise Induction* (RNI). Při této technice se do textu vkládá náhodný šum, prostřednictvím náhodných změn, náhodného odstraňování nebo přehazování znaků nebo náhodného přehazování mezi dvěma znaky, to vše bez ohledu na celkový kontext. Technika rozšíření na úrovni slov je založena na transformaci nebo výměně individuálních slov v textu. Využívají se metody generování synonym, přehazování slov a dalších modifikací na úrovni slov, které vnesou variabilitu do textu, aniž by došlo k narušení celkového významu nebo kontextu textu [45]. Augmentační technika na úrovni vět se provádí pomocí změn struktury nebo skladby celých vět. Využívá se parafrázování, přehazování vět nebo zavádění gramatických variant. Cílem je rozšíření souboru dat tak, že je následně modelu představen s různými formulacemi myšlenek při zachování podstaty původního obsahu. V případě augmentace celých dokumentů se provádí základní změny v textu, jako je vkládání nebo odstraňování celých odstavců, změna pořadí oddílů nebo změna stylu psaní. Tím může být modelu představen dokument s různými strukturami nebo konvencemi psaní [45, 46].

Další možnou technikou pro rozšíření textových dat na úrovni vět je zpětný překlad (*Back-translation*). Zde je využito překladu věty do jiného jazyka a následně zpět do jazyka originálního, což má za následek vznik nových variant. Dále je také využíván přístup přeformulace výroků při zachování sémantické ekvivalence s využitím algoritmů parafrázování [45].

3.1.2 Distribuované reprezentace slov, vět a dokumentů

Zavedení distribuovaných reprezentací, jež modelují význam slov na základě jejich kontextu. Mezi často využívané metody patří například `Word2Vec`, `GloVe`, `FastText` nebo `Doc2Vec`.

`Word2Vec`

Model, který pracuje s reprezentací slov jako vektorů na základě jejich okolí ve větě. Vyžívá se ve dvou variantách: *Continuous Bag of Words (CBOW)* a *Skip-gram* [47].

- CBOW pracuje na principu předpovídání chybějícího slova na základě okolních slov.
- *Skip-gram* pracuje opačně a snaží se předpovídat slova, které se velmi často objevují v okolí počátečního slova.

Obě tyto metody tedy pomáhají modelu učit se vztahy mezi slovy na základě místního kontextu, tedy slov, která se vyskytují blízko sebe. Nabízí tak výhody zejména při porozumění vztahům mezi slovy v krátkých frázích nebo větách. Jako omezení této metody se může jevit to, že výsledné vektory slov jsou statické a každému slovu je přiřazen vektor bez globálního kontextu. Model tak nemusí být schopen rozpoznat odlišný význam slov v závislosti na jejich použití v různých situacích a nezachytává dlouhodobé závislosti a vztahy mimo dané kontextové okno [47].

`GloVe`

Tato metoda je založena na konceptu distribuovaných reprezentací slov, kde model kombinuje výhody statistických a prediktivních přístupů k učení vektorových reprezentací. Základní myšlenkou je, že aspekty významu se extrahují přímo z pravděpodobností společného výskytu, informace o významu slova se tedy kóduje v poměrech výskytů mezi slovy, a tak by měly být tyto poměry zachyceny vektorovou reprezentací. Model poté vychází z matice počtů ko-výskytu slov (*co-occurrence matrix*), kde každý prvek vyjadřuje pravděpodobnost, s jakou se určité slovo objevilo ve stejném kontextu s jiným slovem. `GloVe` tedy pracuje s globálními statistikami výskytu slov v rámci celé korpusové matice, zatímco modely jako `Word2Vec` se soustředí na predikci sousedních slov na základě lokálního kontextu. Ve srovnání s metodou `Word2Vec` nabízí `GloVe` jednoduchou interpretaci výsledných vektorů a vysokou kvalitu reprezentací díky zachycení sémantických i syntaktických vztahů. Na druhou stranu je každé slovo bráno jako jeden vektor, bez ohledu na jeho význam v konkrétním textu a primárně se zaměřuje na globální kontext [47, 48].

`FastText`

S tím, jak se vyvíjí NLP, vyvíjí se také jednotlivé modely. V této době se využívají modely, které v mnoha úlohách překonávají `Word2Vec` a `GloVe`. `FastText` patří mezi novější metodu, která rozděluje slova na pod-hesla, na rozdíl od výše zmíněných technik, které nakládají se slovy jako s jednotlivými jednotkami. Model si tedy lépe poradí se slovy mimo jeho slovní zásobu a je tedy užitečný v případě práce s jazyky

s bohatou morfologií nebo při zabývání se slovy specifickými pro danou oblast, málo se objevujících ve standardních slovnících [47].

`Doc2Vec`

Metoda, rozšiřující model `Word2Vec`, která generuje vektory pouze pro jednotlivá slova, na úroveň dokumentů a jejím cílem je vytvořit číselnou reprezentaci dokumentu bez ohledu na jeho délku. Tímto zachytí nejen význam celých slov, ale i kontext dokumentu jako celku. Existují dvě základní architektury tohoto modelu: *Distributed Memory version of Paragraph Vector* (PV-DM) a *Distributed Bag of Words version of Paragraph Vector* (PV-DBOW) [49].

- PV-DM – dokument reprezentuje jako vektor, který je kombinován s okolními slovy za účelem predikce cílového slova (podobně jako CBOW),
- PV-DBOW – model předpovídá slova, která náleží do dokumentu, pouze na základě vektoru dokumentu (podobně jako *Skip-gram*).

Ačkoliv model nabízí schopnost zachycení globálního kontextu vět nebo dokumentů a menší paměťovou náročnost než velké jazykové modely, je tento model náchylný na přetrénování, zejména u menších datových sad a v porovnání s novějšími metodami založenými na jazykových modelech poskytují slabší výsledky [49].

3.1.3 Pokročilé přístupy rozšíření textu jazykovými modely

U logových záznamů se distribuované reprezentace uplatní zejména při tvorbě vektorových reprezentací celých záznamů, které lze následně použít k vyhledávání podobných logů, jejich shlukování (klastrování) nebo generování alternativních variant. Nevýhodou těchto metod je absence práce s dynamickým kontextem. Tuto slabinu dokážou překonat pokročilé přístupy založené na jazykových modelech, jelikož zohledňují celý širší kontext slov. Například modely jako `BERT`, založené na architektuře `Transformer`, analyzují text obousměrně a umožní tak přesnější pochopení významu slov na základě jejich okolí. Na rozdíl od metod `Word2Vec` nebo `GloVe`, které využívají omezené kontextové okno nebo jednosměrný přístup (zleva doprava nebo zprava doleva), přináší jazykové modely vyšší míru sémantického porozumění, což je v doméně logových záznamů zásadní [47].

Při maskování a doplňování entit je důležité zachytit komplexní kontextové vztahy v záznamech událostí, což heuritické a *embedding* metody nemusí zvládat. V tomto ohledu nabízí pokročilé techniky založené na jazykových modelech s `Transformer` architekturou výhodu, proto je v této práci pohled zaměřen primárně na ně. Modely s touto architekturou, která byla popsána v kapitole 2.4, jsou schopny zohlednit pozici i kontext slov i v rámci složitých struktur logových záznamů.

V dalších kapitolách bude tato práce zaměřena na přípravu a předzpracování datových sad pro procesy trénování a testování jazykových modelů. Následně budou srovnány dvě populární metody – modelování maskovaného jazyka a předpověď dalšího slova, které využívá většina jazykových modelů architektury `Transformer`. Cílem bude komentovat, která z těchto metod nabízí efektivnější řešení pro rozšiřování datových sad záznamů událostí a nahrazování maskovaných entit.

3.2 Charakteristika datových sad

Zásadní vliv na výkonnost jazykových modelů mají především kvalitní, rozmanité a správně strukturované datové sady[1]. Proto, aby bylo trénování modelu relevantní pro specifické domény, jako v tomto případě analýza bezpečnostních logů, je důležité zajistit taková data, která budou bezprostředně souviset s touto problematikou. Správná příprava a charakteristika datových sad je velmi důležitá pro vývoj robustních jazykových modelů schopných doplňovat a rekonstruovat chybějící nebo maskované informace v síťových záznamech.

Pro účel této práce lze rozdělit použité datové sady na dvě skupiny a pro zjednodušení je označit jako trénovací a testovací. První sada, trénovací, obsahuje unikátní logové záznamy z různých systémových zdrojů. Tato sada je následně použita za účelem trénování jazykových modelů metodou bez učitele s jejich následným testováním na úkolu nahrazení maskovaných entit v logových záznamech. Druhá, testovací datová sada, je specificky vytvořená sada s výskytem maskovaných entit logových záznamů. Tyto masky budou následně doplněny naučenými modely. Poté bude doplněná sada využita pro trénování modelu `T5` a jeho validaci na úloze rozpoznání pojmenovaných entit (NER). Následující podkapitoly se věnují přípravě, analýze a popisu obou datových sad.

3.2.1 Datová sada surových záznamů událostí

Pro trénink jazykových modelů na úloze doplňování maskovaných entit logových záznamů byla sestavena datová sada sestávající z různorodých, nezpracovaných záznamů událostí. Cílem je modelu poskytnout vstupy, na kterých se bude schopen učit doplňovat maskované entity. Sada byla sestavena z veřejně dostupných zdrojů a reálných SIEM zařízení, přičemž data byla následně anonymizována a očištěna o duplicity. Nad databází MongoDB byla provedena další filtrace, kdy se omezily počty jednotlivých entit na maximálně 100 unikátních výskytů každého typu (například IP adresy, uživatelská jména, názvy zařízení, apod.). Cílem bylo eliminovat

[1]Datová sada (angl. *dataset, corpus*) označuje velkou skupinu dat, kterou je možné uspořádat či nastavit tak, aby vyhověla konkrétnímu účelu.

nadměrné zastoupení entit, aby se zajistilo rovnoměrné zastoupení tréninkových vzorků. Finální datová sada je složena ze 4 menších datových sad.

Použité záznamy pochází z různých systémových a síťových zdrojů, mezi něž mohou patřit operační systémy Linux a Windows, bezpečnostní zařízení Synology, Fortinet, Fortigate a Cisco ASA, nástroje pro detekci a správu logů Elastic, nástroje pro detekci průniků jako Snort, výzkumné a simulační nástroje Spirit nebo také VPN řešení strongSwan a webové servery Apache. V datové sadě jsou zastoupeny různé varianty standardů logů jako `Syslog`, `CEF` (*Common Event Format*), `Snort Unified Log`, `CLF` (*Common Log Format*) nebo `CBS` (*Component-Based Servicing*). Tato variabilita a pestrost zajistí, že bude model trénován na různorodých kontextech a naučí se pracovat s různými strukturami dat, což je pro generalizaci a robustnost modelu velmi důležité.

Dataset	**Počet záznamů**	**Typy záznamů**
První sada	49 058	Windows Security a různorodé Linux záznamy
Druhá sada	24 305	Anonymizovaný dataset ze SIEM NetWitness platformy
Třetí sada	13 155	Záznamy typu Fortinet, Fortigate a Windows Event
Čtvrtá sada	28 113	Různorodé unikátní záznamy z volně přístupných online dat

Tab. 3.1: Obsah datové sady pro trénování modelů.

Takto vznikl soubor obsahující více než 114 tisíc unikátních logových záznamů. Tabulka 3.2 níže nabízí ukázku použitých logových záznamů v datové sadě pro trénování modelů doplňující maskované entity. V této sadě se také vyskytují záznamy podobné těm, které byly představeny a shrnuty v kapitole 1.1.

Typ logu	Ukázka záznamu
Windows EVTX	<Event xmlns="http://schemas.microsoft.com/win/2004/08 /events/event»<System><Provider Name="Microsoft-Windows-Security-Auditing"Guid= "54849625-5478-4994-A5BA-3E3B0328C30D"/>...<Data Name="TargetUserSid»S-1-5-18</Data></EventData></Event>
Systemd (Linux)	2016-12-12T15:00:01-08:00 2hpisot-centos7 systemd: Starting user-0.slice.
Fortinet CEF	<117>Apr 21 18:42:56 172.25.164.254 CEF:0|Fortinet|Fortigate|v6.0.9|00020|traffic:forward accept|3|deviceExternalId=FG5H0E...
VMware	Dec 19 2022 00:18:05 %VMWARE_VC EVENT: Time=[Dec 19 2022 00:16:33], DeviceType=vpx, Type=UserLoginSessionEvent,...User=widget\tsmadmin,...

Tab. 3.2: Ukázky různých typů logových záznamů použitých pro trénink modelů.

3.2.2 Návrh experimentálního pracoviště za účelem přípravy datové sady

Za účelem simulace počítačových útoků na operační systémy, jejich zachytávání a vyhodnocování bylo navrženo experimentální pracoviště, které implementuje platformu SIEM (viz kapitola 1.3.1), systém `Logstash` a další nástroje, umožňující sběr a analýzu záznamů ze simulovaných útoků. Díky tomuto pracovišti je možné dále zpracovat takto generované záznamy na vysoce výkonnostním serveru pro práci s neuronovými sítěmi a augmentaci logovaných záznamů.

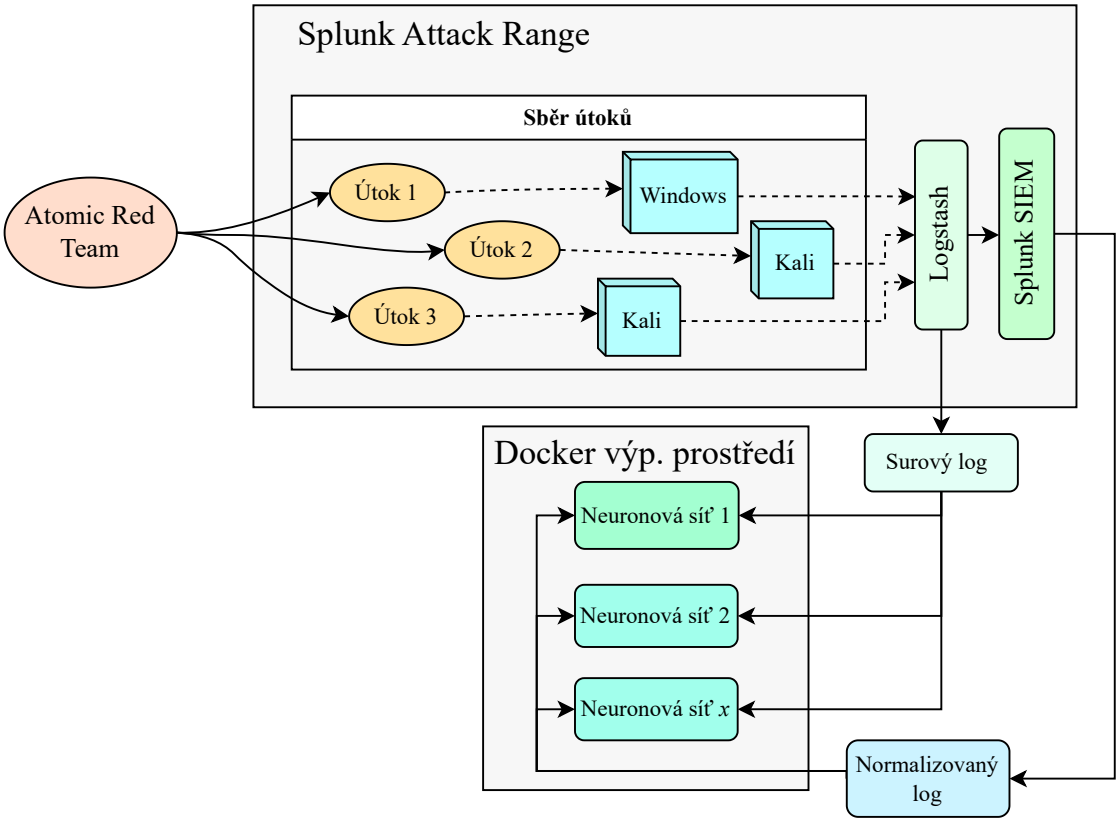
Byly provedeny dva návrhy pracoviště, které zahrnují integraci útoků na virtuální prostředí různých operačních systémů simulovaných pomocí `Atomic Red Team` nástroje, zachycení těchto útoků v prostředí `Splunk Attack Range` a následnou analýzu a zpracování logových záznamů pomocí neuronových sítí uvnitř výpočetního prostředí. Zpracování logů, generovaných během útoků, se provádí za účelem jejich augmentace pomocí různých jazykových modelů založených na neuronových sítích. První návrh pracoviště lze vidět na obrázku 3.1, který znázorňuje zjednodušenou systémovou architekturu. Zde jsou logové záznamy přeposílány ze systému `Logstash` na SIEM a ten umožňuje, po zpracování těchto záznamů, jejich další zpracování na `Docker` kontejneru. Toto schéma lze rozdělit do tří, výše zmíněných částí:

1. `Atomic Red Team` - nástroj pro simulaci různých počítačových útoků, podrobněji bude popsán dále v kapitole.
2. `Splunk Attack Range` - testovací prostředí pro zachytávání a analýzu dat,

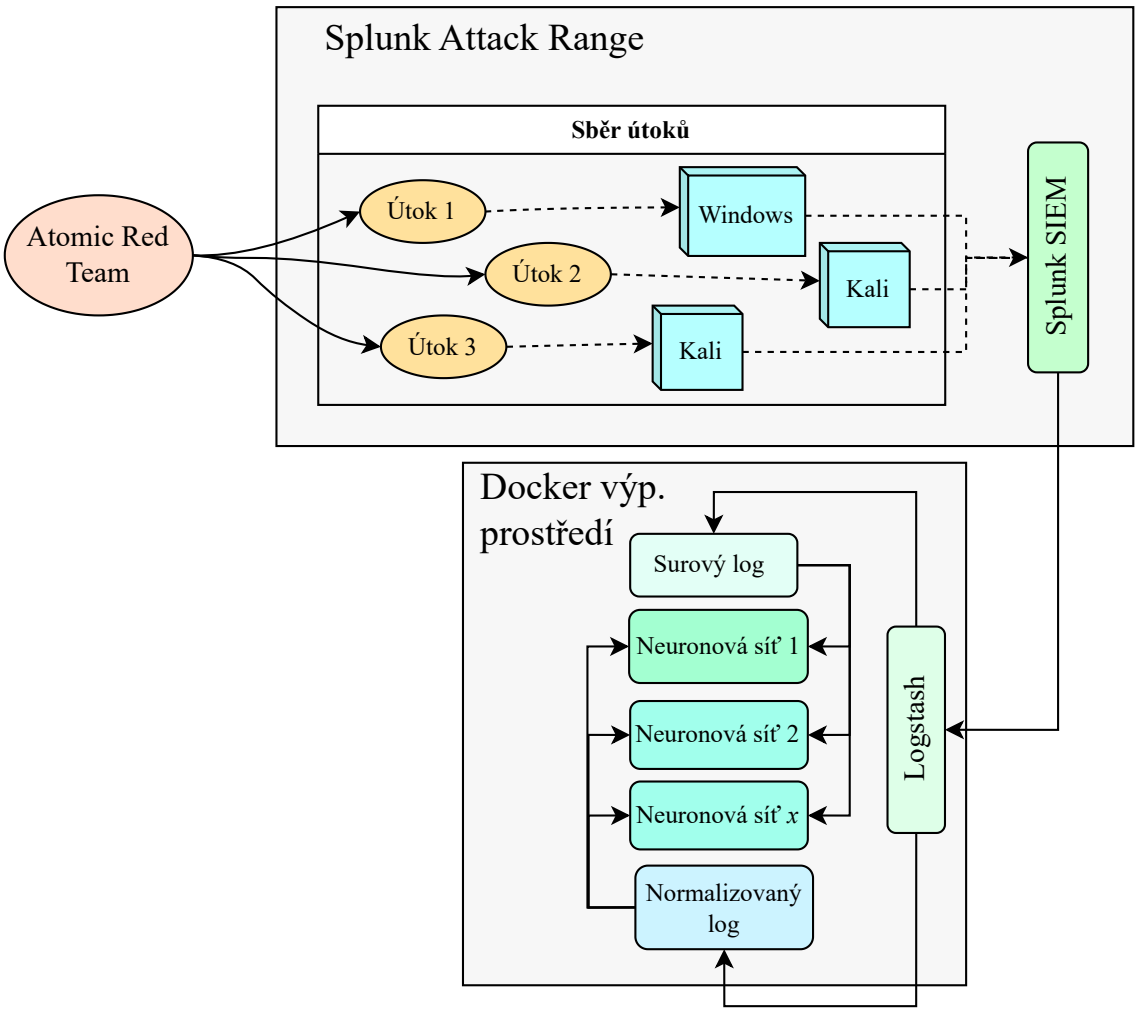
pramenících z kybernetických útoků. Pro zpracování logů je využit `Logstash`, který následně předává data systému SIEM. Nevýhodou takto implementovaného `Logstash` modulu může být složitá konfigurace, kterou je potřeba nastavit zvlášť pro každý jeden vstup a výstup.

3. `Výpočetní prostředí` - nasazený `Docker` kontejner vytvořený na vzdáleném serveru slouží k pokročilým činnostem s vytvořenými záznamy. Umožňuje práci s výkonnou grafickou kartou pro spouštění a úpravu náročných modelů.

Pro zjednodušení konfigurace by bylo možné pracoviště upravit tak, že se `Logstash` umístí za systém SIEM, který zpracuje logové záznamy z jednotlivých útoků a přeposílá je dál, viz obrázek 3.2. Problém zde může nastat při přeposílání záznamů. Kdyby data neměla správný formát, mohl by `Logstash` selhávat při jejich parsování.



Obr. 3.1: První navržené schéma experimentálního pracoviště.



Obr. 3.2: Druhé navržené schéma experimentálního pracoviště.

Splunk Attack Range

Nástroj `Splunk Attack Range` byl vyvinut společností Splunk, působící v oblasti vývoje softwaru pro sběr a analýzu takzvaných strojových dat. Tento *open source* projekt, spravovaný týmem *Splunk Threat Research Team*, umožňuje bezpečnostním týmům vytvořit detekční vývojové prostředí, které simuluje chování útočníka. `Splunk Attack Range` dokáže vytvořit připravené cloudové i lokální prostředí, simulovat útoky a generovat telemetrická data, která jsou odesílána do instance Splunku a následně tato data využít pro vývoj a testování účinnosti detekcí [50]. Detekční vývojová platforma `Attack Range` řeší tři hlavní problémy detekčního inženýrství [50]:

- Schopnost uživatele rychle vytvořit malou laboratorní úlohu co nejvíce podobající se produkčnímu prostředí.
- `Attack Range` provádí simulaci útoků pomocí integrovaných nástrojů jako je např. `Atomic Red Team` za účelem generování dat o útocích.

- Pro automatizaci testování detekčních pravidel lze bezproblémově integrovat do jakéhokoliv *CI/CD*[2] (kontinuální integrace a doručování) *pipeline*.

Nasazení tohoto nástroje je usnadněno pomocí několika předdefinovaných konfiguračních souborů, které umožňují definovat parametry prostředí, typ a počet zařízení, softwarové komponenty nebo také určují, zda se bude jednat o lokální nasazení anebo nasazení pomocí cloudových služeb, konkrétně *AWS* nebo *Azure*. V případě lokálního nasazení běží všechny služby na virtuálním přístroji *VirtualBox*, v němž se všechny potřebné specifikace na server definují v konfiguračních souborech. Alternativní možností je spuštění nástroje prostřednictvím kontejneru pomocí `Dockeru`, který umožní rychlejší a jednodušší implementaci [50].

Jak již bylo zmíněno, všechna vytvořená data jsou posílána na Splunk Server. Tento server je automaticky vytvořen při vytváření sítě a probíhá zde detailní analýza dat. Sběr dat, vytvořených při simulaci útoků, se provádí pomocí nástroje *Splunk Universal Forwarder*, který je automaticky nakonfigurovaný pro každý server a přeposílá všechny zaznamenané logy na server.

Výhodou `Attack Range` je také možnost následné implementace systému Splunk SOAR, který zajišťuje automatizaci tvorby a testování protiopatření na simulovaných útocích. Na druhou stranu může být problémem složitější konfigurace konkrétních zařízení a sítí. Lokální nasazení vyžaduje vyšší nároky na hardware, což může být omezení pro některé organizace. A jelikož je tato platforma primárně navržena pro společnost Splunk, kompatibilita s jinými řešeními SIEM, než těch od původního dodavatele, může být výrazně omezena.

Atomic Red Team

Tato *open source* knihovna testů je mapována na rámec *MITRE ATT & CK*, pomocí něhož jsou bezpečnostní týmy schopny rychle, přenositelně a opakovaně testovat svá prostředí. Výhodou je možnost spouštět testy přímo z příkazové řádky, bez nutnosti jakékoliv počáteční instalace [51].

ATT & CK je celosvětově dostupná databáze o taktikách a technikách útočníků založená na pozorováních z reálného světa. Používá se jako základ pro vývoj specifických modelů hrozeb a metodik v oblasti soukromého sektoru, ve státní správě a v komunitě služeb a produktů kybernetické bezpečnosti. Tato databáze byla vytvořena společností MITRE za účelem vývoje efektivnější ochrany kybernetického prostředí [52].

[2]Tato integrace zajišťuje, že každá revize kódu spouští automatický proces sestavení, testování a přípravu na nasazení. Tento přístup eliminuje testování jako překážku a podporuje efektivnější cestu k rychlému a spolehlivému poskytování vysoce kvalitního softwaru [53].

Výpočetní prostředí

Na vzdáleném serveru je nasazen `Docker` kontejner, který umožňuje využití výkonné grafické karty Nvidia GeForce RTX 4080 pro realizaci výpočetně náročných operací, jako je trénování neuronových sítí. Kontejner slouží primárně jako prostředí pro spouštění tréninkových skriptů, testů natrénovaných modelů a provádění dalších úloh spojených s vysokými nároky na výpočetní výkon. Hlavním cílem těchto operací je augmentace logových záznamů, které jsou generovány během simulovaných kybernetických útoků.

3.2.3 Testovací datová sada

Tato datová sada slouží primárně k testování modelů, sloužících jako generátor maskovaných entit a následného doladění modelu `T5` na NER predikci typů entit v záznamech událostí. Všechna data jsou synteticky vytvořena a vyvinuta reverzním inženýrstvím SIEM parseru. Unikátnost jednotlivých záznamů je zajištěna pomocí kartézského součinu[3]. Datová sada obsahuje primárně protokoly událostí z bezpečnostních zařízení Cisco ASA (*Adaptive Security Appliances*). Cisco ASA je bezpečnostní zařízení vyvinuté společností Cisco Systems. Je navrženo k zajištění bezpečnosti a ochrany sítí tím, že integruje funkce firewallu, antiviru, prevence narušení a virtuální privátní síť (VPN). Poskytuje ochranu proti širokému spektru kybernetických hrozeb, je schopen aktivně bránit útokům a zastavit je dřív, než se po síti rozšíří. Toto zařízení je cenné a flexibilní a může být využito jako bezpečnostní řešení jak pro malé, tak i velké sítě [54].

```
Příklad 1:
"%ASA-permission granted-1726: Call-Home Module started"

Příklad 2:
"%ASA-bounced-3615476: (<mask>) Standby unit failed to sync due to a locked <mask> config. Lock held by
priceangela "

Příklad 3:
"<mask> %ASA-new-945489506: Dynamic  filter   monitored  blacklisted VRRP traffic from
loA:158.182.177.183/39066 (26909/11550) to dockerJ:172.21.79.157/21827
(172.53.240.89/8d63:dcc8:a7d6:d207:ddf9:4d3f:9d28:4d43), destination <mask> resolved from <mask>
list:brown-nicholson.com threat-level: medium, category: 22"
\end{lstlisting}
```

Obr. 3.3: Příklady formátů logových záznamů testovací datové sady.

Příklady záznamů z této testovací sady jsou zobrazeny na obrázku 3.3, kde neznámé entity byly nahrazeny značkami `<mask>`, které budou následně doplněny trénovanými modely.

[3]Kartézský součin je matematická operace, kombinující všechny prvky dvou množin do uspořádaných dvojic.

3.2.4 Datová sada generovaná v prostředí Splunk Attack Range

Byla nalezena veřejně dostupná datová sada společnosti Splunk[4], která bude využita pro testování vybraných generačních modelů. Samotné testování probíhá v kapitole 4.2. Jedná se o sadu vytvořenou v experimentálním prostředí `Splunk Attack Range` simulujícím reálné bezpečnostní záznamy pomocí nástroje `Atomic Red Team`, který implementuje pokročilé techniky a taktiky definované ve frameworku MITRE ATT & CK. Datová sada obsahuje nespočet reálně simulovaných logových záznamů, které odpovídají různým fázím kybernetických útoků, a proto je vhodná k testování bezpečnostních analytických nástrojů v řízeném prostředí. Velikost sady je přibližně 22 GB a zahrnuje široké spektrum formátů záznamů, mezi které patří například formáty `XML`, `Windows Security Format`, `RAW` a `JSON` formáty. Pokrývají také data z různých operačních systémů, včetně Windows, Linux, MacOS nebo z cloudových prostředí. Důkladné pokrytí jednotlivých taktik a technik dle databáze MITRE ATT & CK a jejich dokumentace je velkou předností této sady. Obrázek 3.4 přiložený níže zobrazuje mapování jednotlivých technik využitých v této sadě, které jsou označeny červenou barvou. Toto mapování bylo pořízeno prostřednictvím webového nástroje `ATT & CK Navigator` pro anotování a zkoumání matic ATT & CK, která lze využít k vizualizaci obranného pokrytí, plánování červeného/modrého týmu nebo četnosti zjištěných technik a dalším funkcím[5].

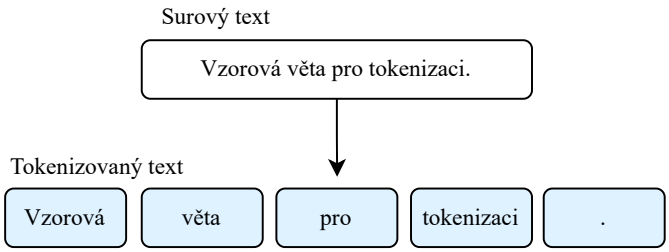
[4]Tato sada je dostupná na této adrese `https://github.com/splunk/attack_data`

[5]Nástroj je dostupný na adrese `https://mitre-attack.github.io/attack-navigator/`.

Obr. 3.4: Techniky útoků obsažené v datové sadě

3.3 Zpracování textových vstupů

Před samotnou aplikací procesu rozšíření textových dat je potřeba provést nezbytné předzpracování textu, což zajistí jeho správnou interpretaci jazykovým modelem. Tento proces zahrnuje kromě rozdělení vstupních datových sad do trénovacích a validačních částí, následné rozdělení textu na jednotlivé tokeny procesem tokenizace. Standardním přístupem je rozdělit dostupný korpus v poměru 80 % dat pro trénování modelu a 20 % pro jeho validaci. Takové rozdělení umožní modelu lépe generalizovat a pomáhá předcházet přeučení na trénovacích datech. Proces tokenizace textu představuje převod nezpracovaného surového textu na sekvenci diskrétních jednotek, takzvaných tokenů, viz také kapitola 2.3. Tento krok je nezbytný, protože jazykové modely s textem přímo pracovat neumí, ale jsou schopny zpracovat pouze jejich číselné reprezentace. Tokeny mohou odpovídat jednotlivým slovům, jejich částem (podslovům) nebo znakům původního textu, v závislosti na využitém algoritmu tokenizace. Příklad takového postupu je znázorněn na obrázku 3.5 níže. Proces tokenizace je prováděn prostřednictvím tokenizátoru, který je buď předem natrénován, nebo vytvořen na míru konkrétnímu textovému korpusu. Tokenizátor zároveň definuje slovník, tj. množinu všech možných tokenů, který je dostupný pro jednotlivé jazykové modely [55].

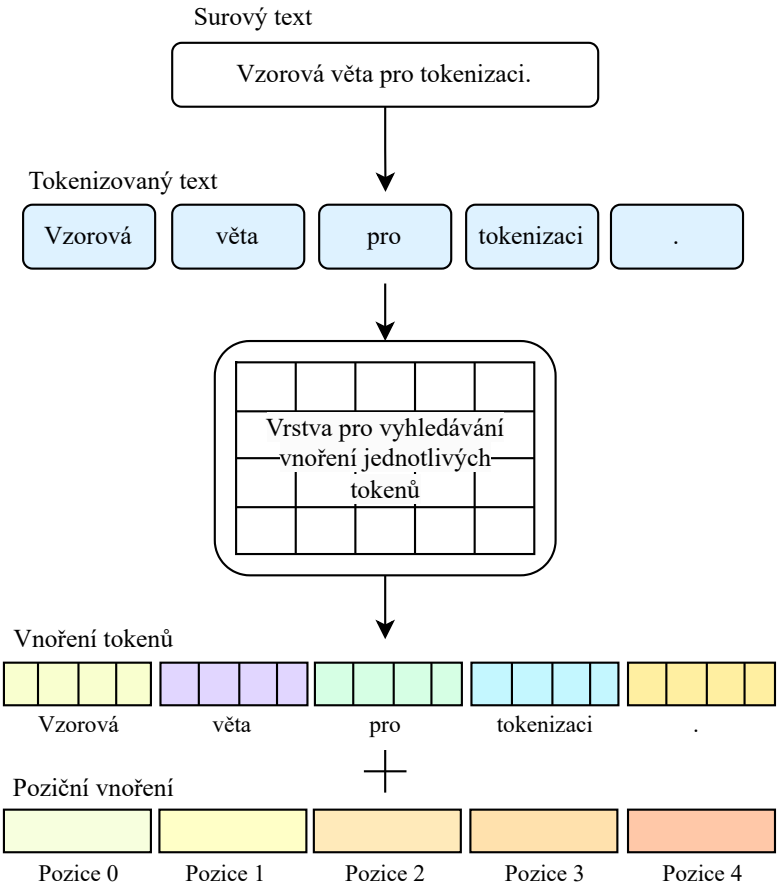


Obr. 3.5: Proces rozdělení vstupní sekvence na tokeny.

Existuje řada různých technik tokenizace vstupní textové posloupnosti, velmi často se využívá metoda kódování párů bajtů (*Byte Pair Encoding*, BPE), která pracuje na úrovni znaků a jejich častých dvojic. Další metodou je BPE na úrovni bajtů (*Byte-level* BPE, BBPE), která namísto textových znaků využívá jako základní jednotku bajty a nad těmito jednotlivými bajty poté operuje, čímž zajistí rozsáhlejší základní slovník (například Unicode znaky). Alternativou k těmto metodám může být algoritmus tokenizace dílčích slov WordPiece, který na rozdíl od BPE nevybírá nejčastější dvojici symbolů, ale vybírá tokeny maximalizující pravděpodobnost výskytu v trénovací sadě, anebo SentencePiece, který zachází se vstupem jako se surovým textem a umožní tokenizaci bez předchozího dělení na slova, čímž

podpoří jazykovou nezávislost nebo jednodušší zpracování textů s nejednoznačnými mezerami. Podrobnější popis těchto a dalších metod viz [56].

Následující krok po tokenizaci textu je takzvané vnoření tokenů (*token embeddings*). Pro jednotlivé tokeny se vyhledává vnoření v rámci vrstvy vložení (*embeddings layer*), která je uložena jako součást parametrů jazykového modelu. Tím se z posloupnosti textových tokenů vytvořených ze vstupu stává posloupnost vnořených tokenů. Poté je zapotřebí přidat jednotlivým tokenům takzvané poziční vnoření (*positional embeddings*). S těmito *embeddings* se zachází stejně jako s tokenovými a mají stejnou velikost. Jsou ukládány jako součást jazykového modelu a trénovány s ostatními parametry. Tento princip je založen na přiřazení *embeddingu* ke každé společné pozici, která existuje v rámci tokenizovaného vstupu [55]. Toto poziční vnoření se přidává, jelikož *self-attention* operace nemá k dispozici žádný způsob reprezentace pozice každého tokenu. Přidáním pozičních *embeddingů* je umožněno *self-attention* vrstvám použít pozici každého tokenu jako relevantní rys během procesu učení. Obrázek 3.6 znázorňuje celý tento proces [55].



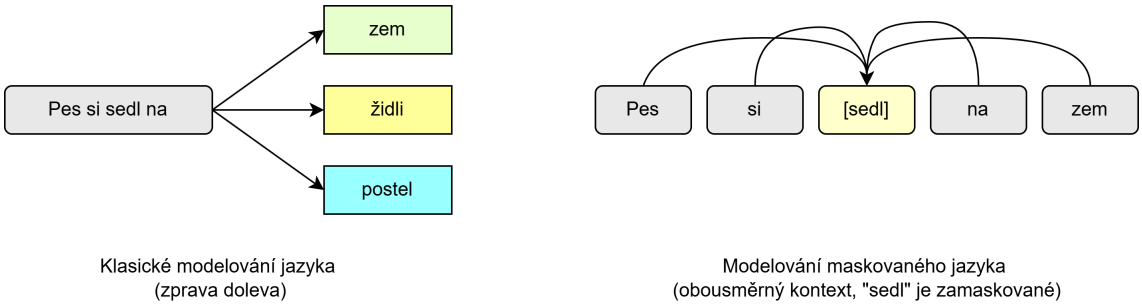
Obr. 3.6: Proces tokenizace obsahující vnoření tokenů a jejich poziční vnoření.

3.4 Techniky rozšíření textu pomocí Transformer architektury

za účelem rozšíření datových sad a zvýšení variability logových záznamů byly v této práci aplikovány modely založené na `Transformer` architektuře. Tato architektura, která již byla představena v kapitole 2.4, nabízí moderní a vysoce efektivní přístup ke zpracování sekvenčních dat. V rámci této práce byly aplikovány dvě základní techniky: metoda modelování maskovaného jazyka (*Masked Language Modeling*) a metoda predikce následujících slov ve větě (*Next Word Prediction*). Hlavním cílem této kapitoly je tedy představit principy těchto zmíněných technik a diskutovat výběr použitých jazykových modelů a jejich vlastností.

3.4.1 Metoda *Masked Language Modeling*

Hlavním cílem metody modelování maskovaného jazyka (MLM) je předvídat náhodně maskovaná slova v textu. Pro tuto úlohu byl primárně natrénován model `BERT`, od kterého se poté odvíjely další jeho varianty. Při předpovídání maskovaného slova bere model v potaz kontext z obou stran věty, čímž se odlišuje od klasických metod zpracování textu, které zpracovávají text zleva doprava [57].



Obr. 3.7: Srovnání metod modelování jazyka a maskovaného modelování jazyka.

`BERT` (*Bidirectional Encoder Representations from Transformers*), jak je patrné z názvu, pracuje na principu obousměrných kodérů, založených na původní architektuře `Transformer`, která byla popsána v kapitole 2.4. Tento model tedy přináší změnu oproti klasickým `Transformerům`, které implementují ve své architektuře jak kodér, tak dekodér. Byl trénován metodou učení bez učitele primárně pro dvě úlohy. Prvním úkolem je již zmíněné modelování maskovaného jazyka (MLM) a druhý úkol, na který je tento model trénován, je tzv. *Next Sentence Prediciton* (NSP), kdy model předpovídá další sekvenci textu na základě přiloženého vstupu. Protože je `BERT` koncepčně jednoduchý a empiricky výkonný, stal se tzv. *state-of-the-art* v jedenácti případech řešení úloh zpracování přirozeného jazyka [57, 58].

RoBERTa

Přestože nabízí BERT velice příznivé výsledky ve spoustě použitých úkolů, studie provedená po vydání BERT modelu odhalila možnost výrazného výkonnostního zlepšení úpravou schématu předtrénování [59]. Byl tedy navržen nový model *Robustly optimized BERT approach*, zkráceně RoBERTa, který implementuje tyto jednoduché změny, mezi něž patří delší doba celkového trénování, větší dávky (*batches*) založené na větší datové sadě, odstranění NSP metody pro zvýšení výkonnosti a trénování na delších sekvencích. Důležitým rozdílem během trénování je také to, že RoBERTa využívá dynamické[6] maskování, kdy se vzor maskování mění, oproti modelu BERT, který využívá maskování statické [36, 59]. Takto provedené úpravy přinesly výrazné zlepšení oproti všem post-BERT metodám. Pro detailnější srovnání výsledků trénování a porovnání s ostatními modely viz [59]. Model RoBERTA je založen na stejné architektuře, z jaké je složen BERT, upraveny jsou pouze klíčové hyperparametry a drobné úpravy vnoření [60]. Z těchto důvodů byl RoBERTA model využit pro další doladění na dostupné datové sadě logových záznamů s následným rozšířením datových sad obsahujících maskované záznamy a následného dotrénování modelu T5 pro predikci pojmenovaných entit.

Kromě modelu RoBERTa byly následně dotrénovány také 3 další modely, umožňující použití metody *Masked Language Modeling* – konkrétně ALBERT-Base-v2, MobileBERT Uncased a ELECTRA-small-generator, které nabízí odlišný přístup k optimalizaci architektury založené na BERT a to z pohledu parametrické efektivity, rychlosti inference nebo tréninkové strategie modelů.

ALBERT-Base-v2

Pro zefektivnění architektury kodéru zavádí model ALBERT tři změny. První změnou je oddělený rozměr vkládání tokenů (*token embedding*) od skrytého rozměru (*hidden dimension*) a tím umožňuje, aby rozměr vkládání byl malý, čímž šetří parametry, zejména v případě velkého slovníku. Druhou změnou je, že všechny vrstvy sdílejí stejné parametry, což dále snižuje počet efektivních parametrů. Poslední změna zahrnuje nahrazení úlohy NSP predikcí pořadí vět, kdy model předpovídá, zda je pořadí dvou vět jdoucích za sebou prohozeno, místo toho, aby zjišťoval, zda k sobě vůbec patří. Tyto změny umožňují dosáhnout vyššího výkonu v úlohách NLU (*Natural Language Undestanding*) [36].

[6]Statické maskování provádí maskování jednou během předzpracování dat, proto, aby se předešlo použití stejné masky každou instanci tréninku v každé epoše, dynamické maskování aplikuje maskovací vzor při každém zadání sekvence do modelu [59].

MobileBERT Uncased

Tento model je navržen tak, aby byl stejně hluboký jako je `BERT_LARGE`, ale je následně zmenšen a optimalizován s důrazem na efektivitu a možnost nasazení na zařízeních s omezeným výpočetním výkonem. Jeho architektura je založena na takzvaných inverzních *bottleneck* vrstvách a hlubších, ale užších `Transformerech`. Tímto je schopen dosáhnout výrazného snížení parametrů a zároveň zachovat vysokou přesnost. Model je trénován pomocí přenosu znalostí z původního, většího učitelského modelu `BERT_LARGE` [61].

ELECTRA-small-generator

`ELECTRA` = *Efficiently Learning an Encoder that Classifies Token Replacements Accurately*. Tento model řeší omezení standardního cíle předtrénování MLM, kdy v každém kroku trénování se aktualizují pouze reprezentace maskovaných tokenů, zatímco ostatní vstupní tokeny nikoliv. K řešení tohoto problému používá `ELECTRA` přístup založený na dvou modelech. Prvním je obvykle malý model, pracující jako standardní model maskovaného jazyka, který předpovídá maskované tokeny. Druhý model se nazývá *diskriminátor* a ten má za úkol předpovědět, které z tokenů na výstupu prvního modelu byly původně maskovány. Diskriminátor tedy provádí binární klasifikaci pro každý token. Přestože je pro tuto práci využit pouze `small-generator`, poskytuje tento model efektivní nástroj pro učení kvalitních jazykových reprezentací i při nižších výpočetních výkonech [36, 62].

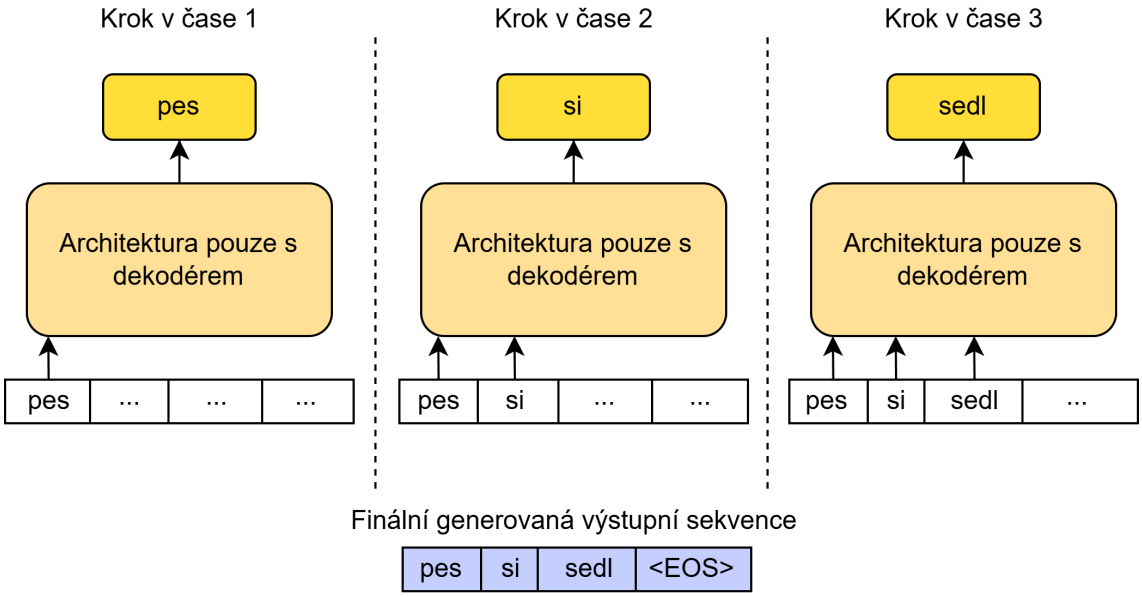
Pro přehledné srovnání vlastností a architektonických rozdílů jednotlivých modelů byla vytvořena tabulka 3.3 níže.

Vlastnost	**`ALBERT-Base-v2`** [63]	**`MobileBERT Uncased`** [64]	**`ELECTRA-small-generator`** [65]	**`RoBERTa-base`** [66]
Architektura	Transformer kodér			
Počet parametrů	11,8 milionů	25 milionů	14 milionů	125 milionů
Počet vrstev jádra	12	24	12	12
Dimenze vnoření	128	128	128	768
Tokenizér	SentencePiece	WordPiece	WordPiece	Byte-Level BPE
Velikost slovníku	30000	30522	30522	50265

Tab. 3.3: Srovnání charakteristik vybraných modelů umožňujících úlohu MLM.

3.4.2 Metoda *Next Word Prediction*

Metoda předpovědi dalšího slova (NWP) je založena na principu autoregresivního[7] generování textu, kdy model predikuje další token na základě předchozích tokenů ve vstupní sekvenci. Tento přístup využívají primárně generativní jazykové modely, jejichž architektura je založena na dekodérech typu `Transformer`, viz kapitola 2.4 nebo obrázek 3.9 níže. Použití dekodéru v této architektuře je velmi užitečné zejména pro úlohy modelování jazyka (*Language Modeling*, LM), jelikož maskované *self-attention* vrstvy uvnitř dekodéru zajistí, že model nemá možnost vidět dopředu sekvence při vytváření nové reprezentace tokenů. Model tedy generuje slovo po slově, dokud není dosaženo speciálního tokenu `<EOS>` (*End-of-Sequence*) nebo maximální délky sekvence [67]. Tento proces znázorňuje obrázek 3.8.

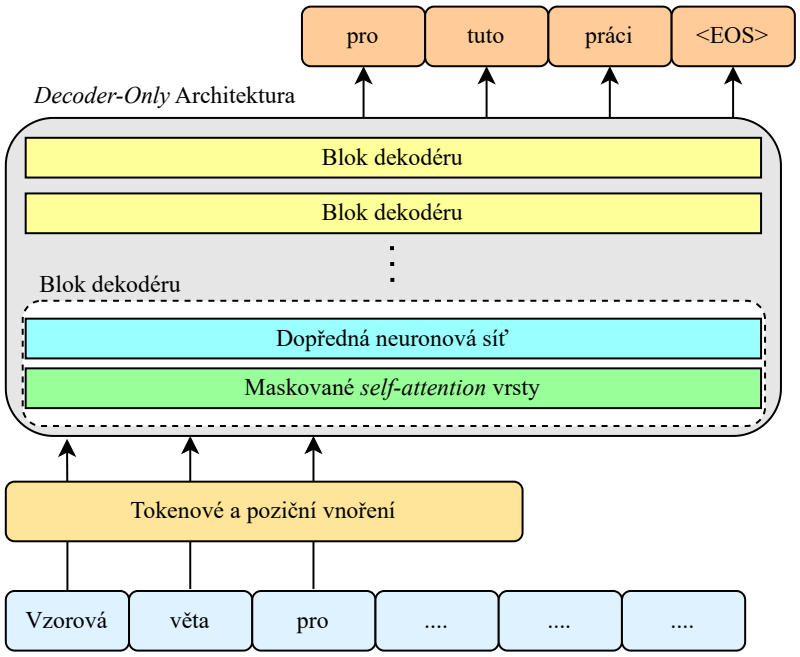


Obr. 3.8: Generování nové sekvence autoregresivním přístupem [67].

Samotné předpovídání tokenů probíhá na základě vkládání tokenových a pozičních *embeddingů* do `Transformer` dekodéru, který následně vytváří výstupní vektor odpovídající každému tokenovému vnoření. Tento výstupní vektor se využívá pro predikci dalšího tokenu v sekvenci. Díky autoregresivnímu principu nemají tokeny žádné znalosti o tokenech následujících a předpověď dalšího tokenu je tedy provedena pomocí jediného dopředného průchodu *decoder-only* Transformerem. Po vytvoření rozdělení pravděpodobnosti nad tokeny se obvykle jednoduše vybírá následující

[7] Autoregresivní princip v tomto případě znamená, že výstup modelu v čase t je použit jako vstup v čase $t+1$.

token. Pro volbu dalšího tokenu lze také využít různé strategie vzorkování nad pravděpodobnostním rozdělením, kterými mohou být například *Top-K Sampling*, *Greedy Decoding*, *Nucleus Sampling* nebo úprava teploty (*Temperature*) [55].



Obr. 3.9: Architektura založena pouze na `Transformer` dekodérech [67].

Na této `Transformer` struktuře, využívající pouze dekodér, je založena architektura modelu `GPT-2`. Ta je vyjma drobných detailů, jako jsou různé inicializace vah nebo použití většího slovníku, totožná s jeho předchůdcem `GPT`. Model `GPT-2` dosahuje velmi slibných výsledků v úlohách typu zodpovídání otázek, překlad, modelování jazyka (*Language Modeling*, LM) a dalších. Kromě `GPT-2` byly v rámci této práce zkoumány i další generativní jazykové modely založené na principu predikce dalšího slova (NWP) architektury `Transformer`. Těmito modely jsou modely `Llama 3.2`, `DeepSeek-R1` nebo model `SmolLM2-1.7B`, který nabízí kompromis mezi velikostí a dostupnou výpočetní technikou. Dále v této kapitole budou tyto modely také blíže představeny.

OpenAI GPT-2

Tento model je rozšířením původního modelu `GPT` (*Generative Pre-Trained Transformer*). Je schopen vytvářet dlouhé sekvence souvislého textu. K předpovědi dalších slov byl trénován na 40 GB internetových textových dat, což odpovídá datové sadě zhruba 8 milionů webových stránek. Zajímavostí je, že tento model byl vydáván postupně, vzhledem k obavám vývojářů z jeho možného zneužití. Původní model

obsahující 1,5 miliardy parametrů byl omezen a vydán ukázkový model s *open-source* kódem se 117 miliony parametrů [36, 68]. Modely `GPT-2` jsou dostupné ve čtyřech velikostech [69]:

- gpt2 - 110M parametrů
- gpt2-*medium* - 345M parametrů
- gpt2-*large* - 774M parametrů
- gpt2-*xl* - 1558M parametrů

Meta Llama 3.2

Model `Llama 3.2` byl představen společností Meta AI jako první multimodální[8] model této řady [70]. Je primárně zaměřen na dvě oblasti. První jsou multimodální modely s podporou vidění, což umožní analýzu vizuálních dat, generování popisků a odpovídání na otázky týkající se obrázků. Druhou oblastí jsou odlehčené jazykové modely pro mobilní a okrajová zařízení. Klíčovou schopností těchto modelů je analýza obrázků, grafů nebo map, zpracování textu i vizuálních informací a poskytnutí komplexních závěrů ve formě shrnutí nebo interpretace dat [70].

SmolLM2

Společnost Hugging Face vydala novou řadu menších modelů, které jsou speciálně optimalizovány pro aplikace v zařízeních, jako reakci na požadavek velkého výpočetního výkonu a paměti masivních jazykových modelů `Llama`, `GPT-4` a dalších. Modely řady `SmolLM2` byly trénovány na 11 bilionech tokenů z datových sad (například FineWeb-Edu, DCLM) zaměřujících se především na anglicky psané texty a jsou schopny pracovat přímo v zařízeních bez závislosti na rozsáhlé infrastruktuře založené na cloudu. Co se týče výkonu, `SmolLM2` překonává model `Meta Llama 3.2-1B` a také vykazuje lepší výsledky než například model `Qwen2.5-1B` [72, 73].

DeepSeek

`DeepSeek` je čínský startup s umělou inteligencí, jež dokázal vyvinout model `DeepSeek-R1` za mnohem nižší cenu než společnost OpenAI model `o1`. Chatbot aplikace této společnosti nabízí dvě varianty modelů:`DeepSeek-V3` a `DeepSeek-R1` [74, 75].

- Výchozí model `DeepSeek-V3` je univerzální velký jazykový model, který vyniká jako univerzální nástroj schopný zvládnout širokou škálu úloh. Jeho klíčovou vlastností je použití přístupu *Mixture-of-Experts* (MoE), který umožňuje využití pouze nejvhodnější části modelu pro konkrétní úlohu, což šetří výpočetní

[8] Multimodální umělá inteligence dokáže zpracovat a integrovat informace z různý typů vstupních dat s cílem vytváření komplexnějších a přesnějších předpovědí [71].

výkon. Tento model je tedy spolehlivý pro většinu každodenních úloh, každopádně jeho schopnost řešit problémy vyžadující uvažování nebo přicházení s novými odpovědmi, které se nenacházejí v trénovacích datech, je omezená, jelikož jako většina LLM pracuje pomocí predikce dalšího slova [74].

- Model `DeepSeek-R1` je výkonný model vytvořený pro řešení úloh, které vyžadují pokročilé uvažování a hluboké řešení problémů. Při dotazování je model schopen použít řetězové uvažování, s cílem zamyšlení se nad problémem. Vyniká tak při řešení kognitivních operací na vysoké úrovni a logicky náročných otázek. Tento model je tedy přímým konkurentem modelu `o1` společnosti OpenAI [74, 75].

Podobně jako v kapitole 3.4.1 je pro srovnání technických vlastností a rozdílů v architektuře vytvořena tabulka 3.4.

Vlastnost	**`DeepSeek-R1-Distill-Llama-8B`** [79]	**`GPT-2`** [76]	**`Llama 3.2-3B`** [77]	**`SmolLM2-1.7B`** [78]
Architektura	Transofmer dekodér			
Počet parametrů	8,03 miliard	1,5 miliardy	3,21 miliardy	1,7 miliardy
Počet vrstev jádra	32	12	32	24
Dimenze vnoření	4096	768	3072	2048
Tokenizér	Byte-Level BPE	BPE	BPE	BPE (`GPT-2` kompat.)
Velikost slovníku	128000	50257	128000	49152

Tab. 3.4: Srovnání charakteristik vybraných modelů umožňující úlohu NWP.

Model Text-to-Text Transfer Transformer

Text-to-Text Transfer Transformer, zkráceně „`T5`“, je založen na kodér-dekodér architektuře `Transformer`. Tento model byl předtrénován na rozsáhlé datové sadě *Colossal Clean Crawled Corpus* (`C4`) využitím metody *denoising*. V rámci cíle této úlohy je model trénován tak, aby předpovídal chybějící nebo jinak poškozené tokeny na jeho vstupu. Všechny jazykové úlohy byly během trénování přeformulovány do jednotného *text-to-text* formátu, včetně také úloh *benchmarků* `GLUE` a `SuperGLUE` [80]. `T5` je prvním modelem, který pomocí metod přeneseného učení, viz 2.2.3, umožňuje efektivní přenos znalostí, jako je strojový překlad jazyka, shrnutí textu, klasifikace textu a další. Na základě rozšířené datové sady pomocí modelů umožňujících MLM z kapitoly 3.2.3 bude otestována přesnost modelu `T5` při řešení syntaktické analýzy

logů a metody rozpoznání pojmenovaných entit, čemuž se bude věnovat kapitola 4.3. Jelikož tento model nebude použitý pro trénování na surových událostech s cílem nahrazování masek v záznamech, ale bude využit později až k testování, jeho vlastnosti budou popsány v kapitole 4.3.

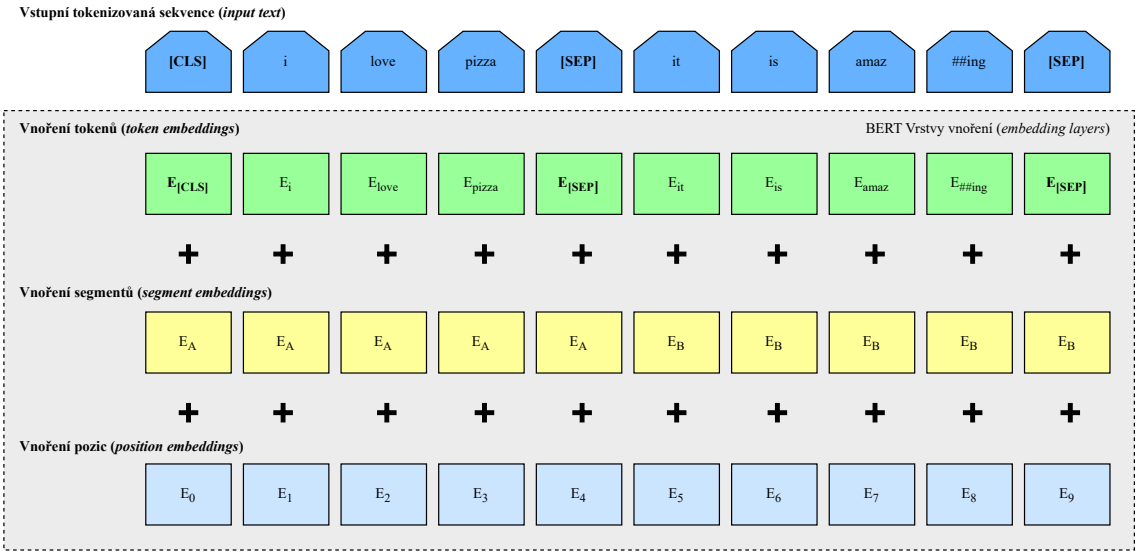
3.5 Proces ladění jazykových modelů

Pro metody a jazykové modely popsané v kapitole 3.4 byly vytvořeny tréninkové procesy. Tyto tréninky jsou ve své podstatě velmi podobné, pro každou metodu se zde liší přístup zpracování dat pro jednotlivé tréninky a následná technika samotného procesu tréninku. Nadcházející podkapitola bude rozdělena do dvou částí, v první části bude popsána metoda trénování modelů `RoBERTa`, `ALBERT-Base-v2`, `MobileBERT Uncased` a `ELECTRA-small-generator`, využívající techniku MLM. V druhé části bude popsán trénink generativních modelů a jejich možností, pracujících na základě předpovědi nových tokenů (slov nebo sekvencí) na základě vstupních dat.

Trénink všech modelů probíhal ve výpočetním prostředí na vzdáleném serveru, přiblíženém v kapitole 3.2.2. Jak již bylo zmíněno, toto prostředí zahrnuje výkonnou grafickou kartu `NVIDIA RTX 4090` disponující 24 GB pamětí.

3.5.1 Trénování modelů založených na metodě MLM

Aby byl vstupní surový text datové sady kompatibilní s trénovaným modelem, je nejprve nutné provést potřebné kroky popsané v kapitole 3.3. Surový text se převádí na jednotlivé tokeny, které přijímá model na svých vstupech. Reprezentace vstupní vrstvy vnoření se pro model `BERT` skládá ze tří dílčích vrstev: *Token Embeddings*, *Segment Embeddings* a *Positional Embeddings*. Zatímco poziční a tokenové vnoření bylo představeno v 3.3, *Segment Embeddings* přidává každému tokenu informaci o tom, do jaké části vstupní sekvence spadá, a přidává tak možnost rozpoznání sémantické závislosti mezi vstupními větami [81].



Obr. 3.10: Architektura vrstev vnoření modelu `BERT` [57].

Během procesu trénování se následně v tokenizované datové sadě náhodně nahrazují některé tokeny maskou pomocí `mask_token`. Kromě zamaskovaných tokenů jsou mezi vstupní sekvence vkládány také tokeny `cls` a `sep`. Pro model `BERT` začíná každá vstupní sekvence tokenem `cls` a končí tokenem `sep`, který značí konec věty nebo začátek nové. V případě výskytu více vět je mezi ně rovněž vložen tento oddělovací `sep` token.

`RoBERTa` model, který představuje rozšíření modelu `BERT`, využívá stejnou architekturu, ale přechází z původního BPE (*Byte Pair Encoding*) slovníku na BBPE (*Byte-Level BPE*) tokenizaci. Jelikož `RoBERTa` model postrádá identifikátory segmentů `token_type_ids`, není tedy potřeba explicitně určovat, ke kterému segmentu určitý token náleží. Segmenty jsou jednoduše rozděleny příkazem `tokenizer.sep_token` nebo pomocí `</s>` příznaku. Dalším významným rozdílem, který byl uveden i v kapitole 3.4.1, spočívá ve způsobu maskování. `RoBERTa` zavádí dynamické maskování, kdy jsou maskované tokeny vybírány náhodně v průběhu každé trénovací epochy. Naopak `BERT` model využívá statické maskování, kde dochází k nahrazení tokenu maskou již během předzpracování dat, což vede k jedné statické masce [66].

Tabulka 3.5 porovnává výše představené speciální tokeny, které jsou využity jednotlivými modely. Jejich stručný popis je následující [66]:

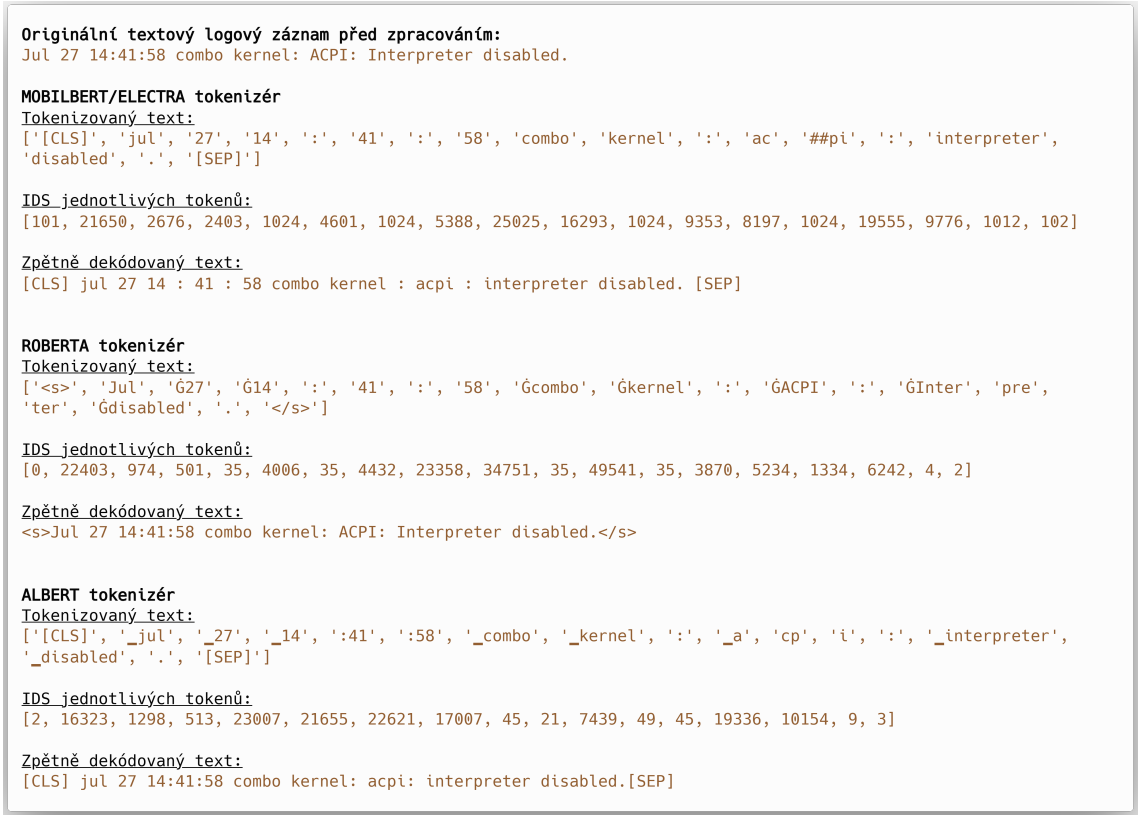
- **`bos_token`**: *Beginning of sequence token.* Během předtrénování značí začátek sekvence.
- **`eos_token`**: *End of sequence token.* Během předtrénování značí konec sekvence.
- **`unk_token`**: *Unknown token.* Token, který se nenachází ve slovníku a nelze jej tedy převést na ID.

- **sep_token**: *Separator token*. Využívá se při sestavování jedné sekvence z více sekvencí, například dvou sekvencí pro klasifikaci nebo pro zodpovídání otázek na základě textu a otázky. Je také používán jako poslední token sekvence, která je sestavena pomocí speciálních tokenů.
- **pad_token**: *Padding token*. Tento token se využívá při vyplňování, například při dávkování sekvencí různých délek.
- **cls_token**: *Classifier token*. Používá se při klasifikaci sekvence namísto jednotlivých tokenů a vyskytuje se jako první token sekvence při sestavování pomocí speciálních tokenů,
- **mask_token**: Používá se při trénování modelu pomocí MLM metody. Tento token se model snaží předpovídat.

Model	cls	mask	pad	sep	unk	bos	eos
albert-base	[CLS]	[MASK]	<pad>	[SEP]	<unk>	[CLS]	[SEP]
electra-small-gen.	[CLS]	[MASK]	[PAD]	[SEP]	[UNK]	-	-
mobilebert-uncased	[CLS]	[MASK]	[PAD]	[SEP]	[UNK]	-	-
roberta-base	<s>	<mask>	<pad>	</s>	<unk>	<s>	</s>

Tab. 3.5: Přehled speciálních tokenů používaných v předtrénovaných modelech.

Tímto způsobem byla tréninková sada představená v kapitole 3.2.1 před zahájením tréninku tokenizována. Obrázek 3.11 poskytuje detailnější pohled na postup tokenizace vstupního textu využívanými MLM modely.



Obr. 3.11: Průběh tokenizace logového záznamu pro tokenizátory MLM modelů.

Zpracovaný záznam na výstupu tokenizátoru obsahuje příznaky značící začátek a konec sekvence `<s>`/`[CLS]` a `</s>`/`[SEP]`. Tyto tokeny jsou následně převedeny na jejich číselné reprezentace (takzvané *token_ids*). Při zpětném převodu lze vidět, že je výsledná sekvence o tyto bity obohacena.

Při rozdělení původní sekvence na tokeny zavádí `RobertaTokenizerFast` speciální znak `Ġ`, pomocí kterého značí mezeru mezi znaky. `AlbertTokenizer` značí mezeru mezi pomocí znaku podtržítka. Tokenizér modelů `MobileBERT` a `ELECTRA` mezery neznačí žádným speciálním znakem, ale pokud se dané slovo nevyskytuje v jeho slovníku, je rozděleno pomocí předpony „##“ a každý token s touto předponou by měl být při zpětném převodu tokenů na řetězec sloučen s předchozím znakem [36].

Následné zajištění náhodného maskování slov během procesu trénování na tréninkové sadě probíhá pomocí objektu `DataCollator` (srovnávač), který je představen na obrázku 3.12. Tento srovnávač je schopen vytvořit dávky (*batche*) ze vstupních prvků datové sady a zároveň provádět další nezbytné operace, jako je například zarovnání délek pomocí *paddingu*. Pro úlohu typu MLM je tento objekt konfigurován s parametrem `mlm=True` a následně definovanou pravděpodobností, s jakou má objekt maskovat tokeny (v tomto případě 15 %). Takto se při každém průchodu

tréninkovou sadou zajistí náhodné maskování vybraných tokenů speciálním maskovacím tokenem.

```
# Definice DataCollatoru, který se používá pro zpracování dat před vstupem do modelu.
# DataCollatorForLanguageModeling je speciální třída z knihovny Hugging Face, která
# slouží pro úlohy jazykového modelování.
# Parametr `mlm` určuje, zda má být použito Masked Language Modeling (MLM).
# Parametr `mlm_probability` definuje pravděpodobnost, se kterou je token v trénovacím
# textu nahrazen maskou ([MASK])

data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=True,
    mlm_probability=0.15
)
```

Obr. 3.12: Definovaný `DataCollator` pro náhodné maskování.

```
######## Výstup definovaného DataCollatoru pro RobertaFastTokenizer ########

Původní textový záznam:
Jul 27 14:41:58 combo kernel: ACPI: Interpreter disabled.

Tokenizovaný záznam:
['<s>', 'Jul', 'Ġ27', 'Ġ14', ':', '41', ':', '58', 'Ġcombo', 'Ġkernel', ':', 'ĠACPI', ':', 'ĠInter', 'pre',
'ter', 'Ġdisabled', '.', '</s>']

Dekódovaný záznam po aplikaci DataCollatoru:
<s>Jul 27 14:<mask>:58 combo<mask>: ACPI: Interpreter disabled.</s>
```

Obr. 3.13: Náhodné maskování hodnot pomocí třídy `DataCollator`.

Po tomto procesu, kdy je vstupní datová sada rozdělena na tréninkovou a validační a následně tokenizována, je definován samotný trénink. V níže přiložené tabulce 3.6 jsou shrnuty základní parametry, které byly konfigurovány pro prvotní trénink.

Parametr	Hodnota	Význam
Epochy (*Epochs*)	10	Počet iterací nad trénovací datovou sadou při učení
Trénovací dávka (*Train_batch_size*)	8	Počet vzorků, předaných modelu k jeho trénování
Validační dávka (*Eval_batch_size*)	8	Počet vzorků, předaných modelu k jeho validaci
Rychlost učení (*Learning_rate*)	2e-5	Základní míra rychlosti učení sítě
Zahřívací kroky (*Warmup_steps*)	50	Počet kroků, během nichž se lineárně zvyšuje rychlost učení z 0 na počáteční hodnotu
Váhový útlum (*Weight_decay*)	0.01	Parametr regulující hodnoty váhových koeficientů, pomáhá předcházet přeučení

Tab. 3.6: Parametry tréninku pro MLM modely.

3.5.2 Trénování modelů založených na metodě NWP

Průběh přípravy surových logových záznamů probíhá velmi podobně jako u modelů založených na metodě MLM v kapitole 3.5.1. Nejprve je celková datová sada rozdělena na testovací a validační, přičemž validační sada tvoří 20 % původních dat a zbývajících 80 % dat je použito pro samotné trénování. Následně je v obou těchto částech provedena tokenizace vstupních textových vzorků.

Dochází zde k problému, který se týká velikostí těchto modelů. Vzhledem k tomu, že modely pracující na principu předpovídání dalšího slova často dosahují miliard parametrů, a jejich plné trénování je extrémně výpočetně náročné. Z tohoto důvodu byla pro jemné doladění těchto velkých jazykových modelů využita technika *Low-Rank Adaptaion* (LoRA). Tato technika dokáže zmrazit vybrané váhy předtrénovaného modelu a do každé vrstvy `Transformer` architektury vloží trénovatelné matice rozložení hodností, a tím výrazně sníží počet trénovatelných parametrů pro navazující úlohy [82].

Nastavení LoRA adaptace pro modely `Llama 3.2-3B, SmolLM2-1.7B` a `DeepSeek-R1-Distill-Llama-8B`[9] je zobrazeno na obrázku 3.14.

[9]Dále v textu bude tento model značen pouze jako `DeepSeek-R1-8B`, což usnadní práci s tabulkami.

```python
### Konfigura LoRA s hodností (rank) 16 ###
### Význam jednotlivých proměnných je následující:
# @r = Hodnost (rank) nízkorozměrné aproximace – určuje velikost přídavných trénovatelných matic (obvyklé rozmezí
# 4-32). Nižší hodnota zajistí větší kompresi, ale potenciálně menší výraznost.
# @lora_alpha = Škálovací faktor pro LoRA vrstvy – ovlivňuje sílu úpravy parametrů během učení. Vyšší hodnoty
   vedou k silnějším adaptačním efektům.
# @lora_dropout = Pravděpodobnost dropout – zabraňuje přeučení tím, že náhodně vynechává části LoRA vrstev. Vyšší
  hodnoty pomáhají zabránit nadměrnému přizpůsobení při učení
# @task_type = Značí typ úlohy - v tomto případě "casual language modeling" pro predikci dalších tokenů
# @bias = Kontroluje trénink výrazů zkreslení. Možnosti jsou „none“, „all“ nebo „lora_only“. „none“ je
nejčastější kvůli efektivitě paměti.
# Dalším možným parametrem je @target_modules, který určuje, na které moduly modelu se má LoRA využít. Možnosti
jsou „all-linear“ nebo konkrétní moduly jako „q_proj,v_proj“. Více modulů umožňuje větší přizpůsobivost, ale
zvyšuje využití paměti.

lora_config = LoraConfig(
    r=16,
    lora_alpha=8,
    lora_dropout=0.1,
    task_type="CAUSAL_LM",
    bias="none"
)
```

Obr. 3.14: Konfigurace LoRA adaptace pro doladění velkých jazykových modelů.

Každý model využívá při tokenizaci vlastní tokenizér, jejichž specifika byla uvedena v tabulce 3.4. Oproti předchozímu přístupu modelování maskovaného jazyka využívají generativní modely odlišené speciální tokeny. V rámci této práce se jedná maximálně o čtyři typy tokenů:

- **bos_token** – začátek sekvence,
- **eos_token** – konec sekvence,
- **pad_token** – doplnění na jednotnou délku,
- **unk_token** – neznámý token.

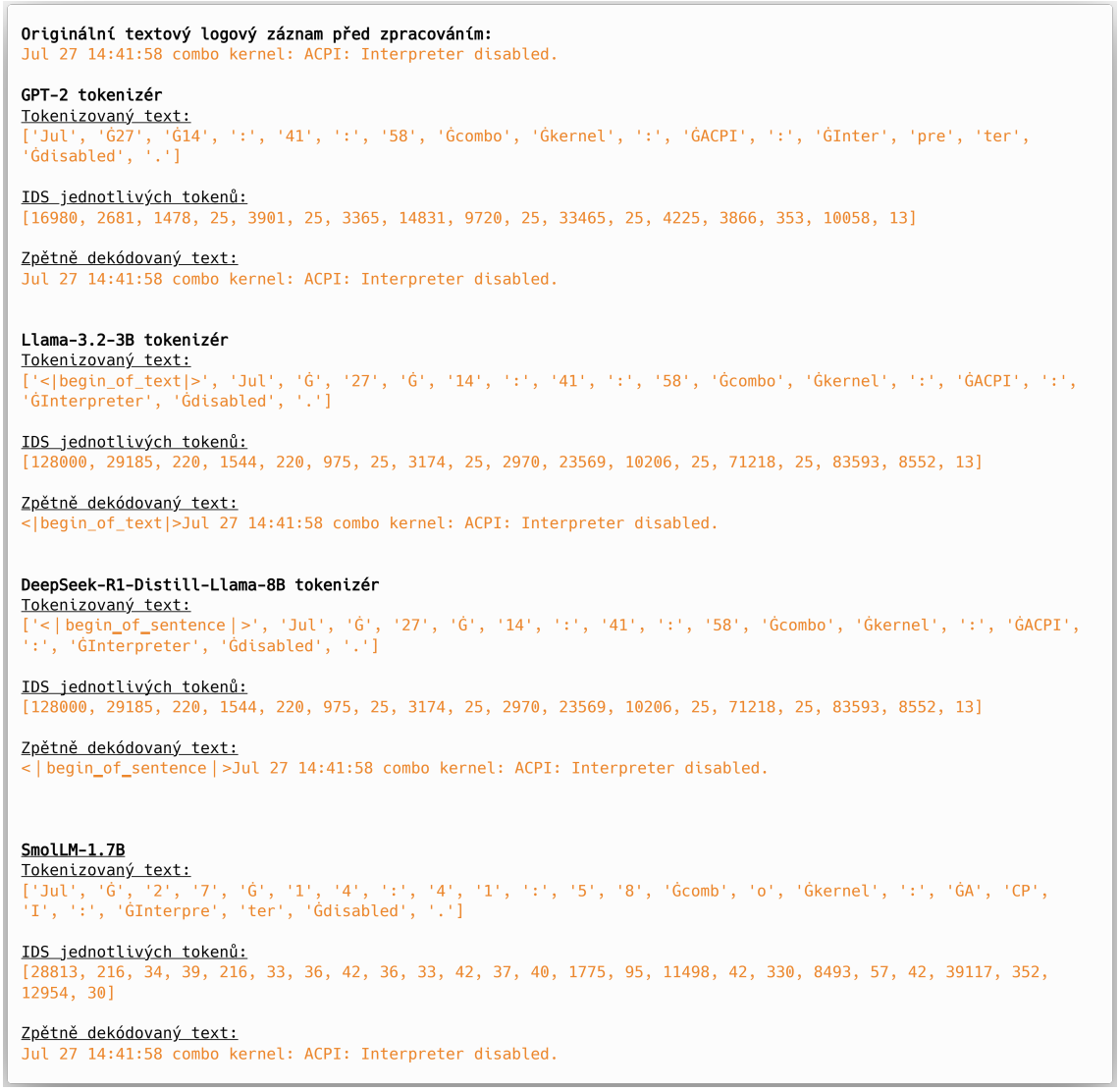
Jejich význam je stejný, jako je význam speciálních tokenů popsaných v kapitole 3.5.1. Přehled toho, které modely využívají jaké konkrétní tokeny, je uveden v tabulce 3.7 níže.

Model	bos	eos	pad	unk
DeepSeek-R1-8B	<|begin_of_sentence|>	<|end_of_sentence|>		-
GPT-2	<|endoftext|>		-	<|endoftext|>
Llama-3.2-3B	<|begin_of_text|>	<|eot_id|>	-	-
SmolLM2-1.7B	<|im_start|>	<|im_end|>		<|endoftext|>

Tab. 3.7: Přehled speciálních tokenů používaných v předtrénovaných modelech.

V této tabulce lze vidět, že tokenizér modelů `GPT-2` a `Llama-3.2-3B` nemá žádný `pad` token, proto se tedy nejčastěji nastavuje na hodnotu tokenu `eos`. V případě

potřeby speciálního tokenu lze nastavit na novou, jinou hodnotu pomocí funkce, kterou samotný tokenizér nabízí: `tokenizer.add_special_tokens('pad_token': '[PAD]')`.



Obr. 3.15: Průběh tokenizace logového záznamu pro tokenizátory NWP modelů.

Jako v případě MLM modelů, tréninková datová sada byla tímto způsobem tokenizována. Příklad rozkladu na jednotlivé tokeny a jejich následný převod na číselné hodnoty nabízí výše přiložený obrázek 3.15, kde lze vidět, jak tokenizátory jednotlivých modelů zpracovávají text. Všechny tokenizátory zpracovávají text velmi podobně, mezera mezi jednotlivými znaky je značena speciálním znakem `Ġ`. Poté jsou tyto tokeny převedeny na jejich odpovídající ID číselné hodnoty, které jsou během tréninku vloženy modelu.

Opět je definován `DataCollator`. V tomto případě je nastaven jinak, než tomu bylo v kapitole 3.5.1, viz obrázek 3.16. Jelikož se jedná o prosté kauzální modelování jazyka, parametr `mlm` pro náhodné maskování tokenů během tréninku je nastaven jako *False* a maskování je vypnuto. Srovnávač tak slouží pouze k doplnění vstupů na maximální délku *batche*, pokud původně stejnou délku nemají.

```
# Definice objektu DataCollator pro CasualLanguageModeling (kauzální modelování jazyka).
# Jelikož se nejedná o MLM úlohu, parametr "mlm" je nastaven na False, a tím se
# neprovádí náhodné maskování tokenů.
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False,
)
```

Obr. 3.16: `DataCollator` pro úlohu typu Causal Language Modeling.

V následující části jsou definovány parametry pro spuštění trénování. Tyto parametry byly upraveny na základě velikosti jednotlivých modelů a jejich výkonnostních požadavků.

Parametr	**DeepSeek-R1-Distill-Llama-8B**	**GPT-2**	**Llama 3.2-3B**	**SmolLM2-1.7**
Epochy (*Epochs*)	10			
Trénovací dávka (*Train_batch_size*)	2	8	2	4
Validační dávka (*Eval_batch_size*)	2	8	2	4
Rychlost učení (*Learning_rate*)	2e-5			
Zahřívací kroky (*Warmup_steps*)	50			
Váhový útlum (*Weight_decay*)	0.01			

Tab. 3.8: Nastavení parametrů tréninku pro NWP modely.

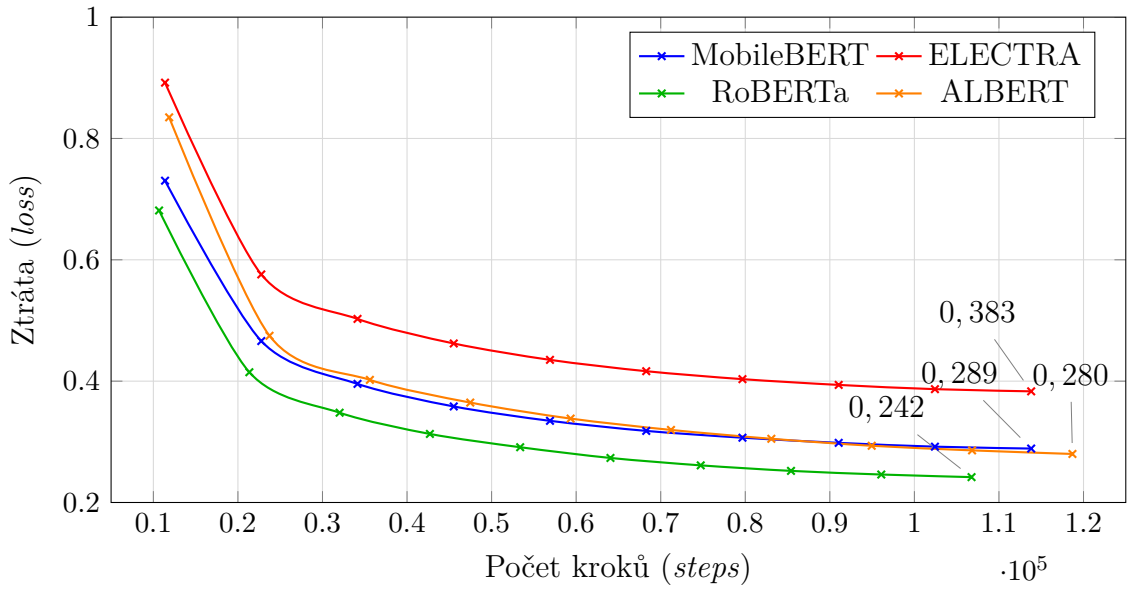
3.5.3 Hodnocení tréninkových procesů

Tato podkapitola se bude věnovat samotným výsledkům tréninkových procesů modelů představených v kapitolách 3.5.1 a 3.5.2. Průběh jednotlivých procesů byl monitorován a zaznamenán pomocí nástroje `MLflow`, který byl představen v kapitole 2.5, jehož využití sloužilo k efektivnímu vyhodnocení strojového učení.

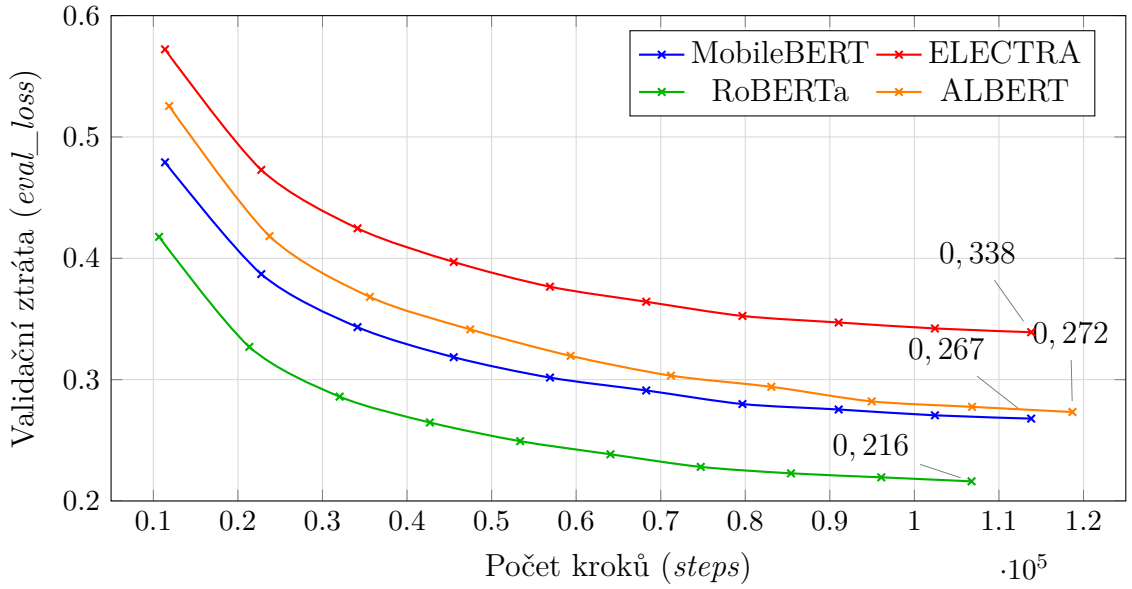
Validační ztráta a celková hodnota ztrátovosti i další metriky byly ve všech případech tréninkových procesů vyhodnocovány na konci každé epochy.

Trénink MLM modelů

Pro zvolené parametry v tabulce 3.6 bylo spuštěno trénování. Graf na obrázku 3.17 zobrazuje teoreticky očekávaný průběžný pokles ztrátovosti modelů s rostoucím časem během procesu učení. Nejmenší ztrátovosti dosáhl model `RoBERTa`, který po 10 epochách dosáhl hodnoty ztráty 0,242.



Obr. 3.17: Vývoj hodnoty ztráty (`loss`) v závislosti na počtu kroků jednotlivých MLM modelů.



Obr. 3.18: Vývoj validační ztráty (`eval_loss`) v závislosti na počtu kroků jednotlivých MLM modelů.

Výše přiložené grafy 3.17 a 3.18 poskytují přehled o vývoji hodnot ztrátových funkcí `Loss` a `Eval_loss` u všech modelů během procesu učení. Jak již bylo zmíněno, nejlepších výsledků zde dosahuje model `RoBERTa`, což je patrné z obou grafů, avšak všechny modely vykazují očekávaný pokles obou metrik, což naznačuje správný průběh doladění (*fine-tuning*) MLM modelů.

Parametry	**ALBERT**	**ELECTRA**	**MobileBERT**	**RoBERTa**
Epochy	10	10	10	10
Doba trvání	4 hod 06 min	1 hod 18 min	3 hod 06 min	3 hod 54 min
Ztráta (*Loss*)	0,280	0,383	0,289	**0,242**
Validační ztráta (*Eval_loss*)	0,272	0,338	0,267	**0,216**
Perplexita (*Perplexity*)	1,313	1,401	1,306	**1,242**

Tab. 3.9: Hodnocení procesu učení MLM modelů.

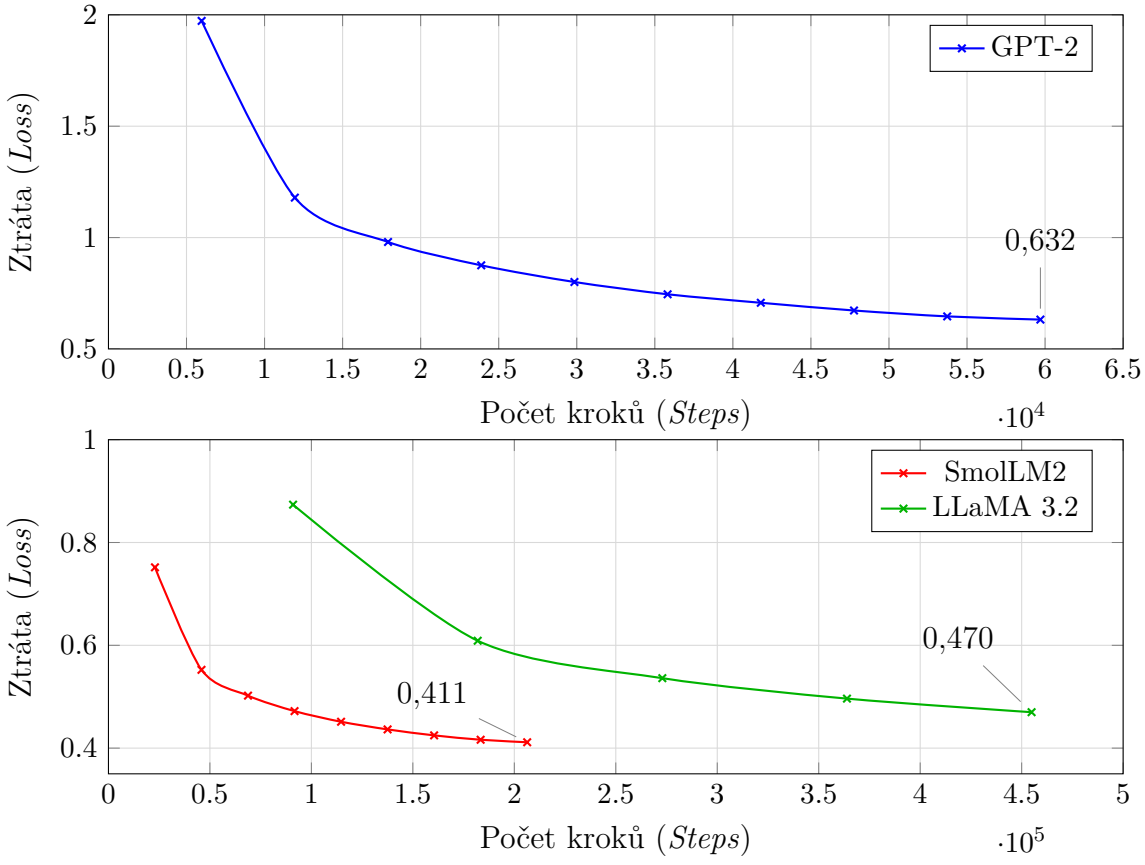
Detailnější srovnání výsledných hodnot učení modelů nabízí tabulka 3.9. Po dokončení tréninků každého modelu byla následně vypočítána hodnota zmatenosti (perplexity) na validační množině dat. Přestože je tato metrika přesnější pro autoregresivní modely, jak bylo zmíněno v kapitole 2.5, i v tomto případě může pomoci identifikovat modely, které by vykazovaly vysokou míru zmatenosti a potenciálně

horší schopnost generalizace. Všechny čtyři modely vykazují velmi nízkou hodnotu perplexity, a proto jsou vhodné k testování pro nahrazení masek v logových záznamech. Na základě dosažených hodnot lze jednoznačně označit jako nejúspěšněji doladěný model `RoBERTa`, který ve všech ohledech dosahoval nejlepších výsledků.

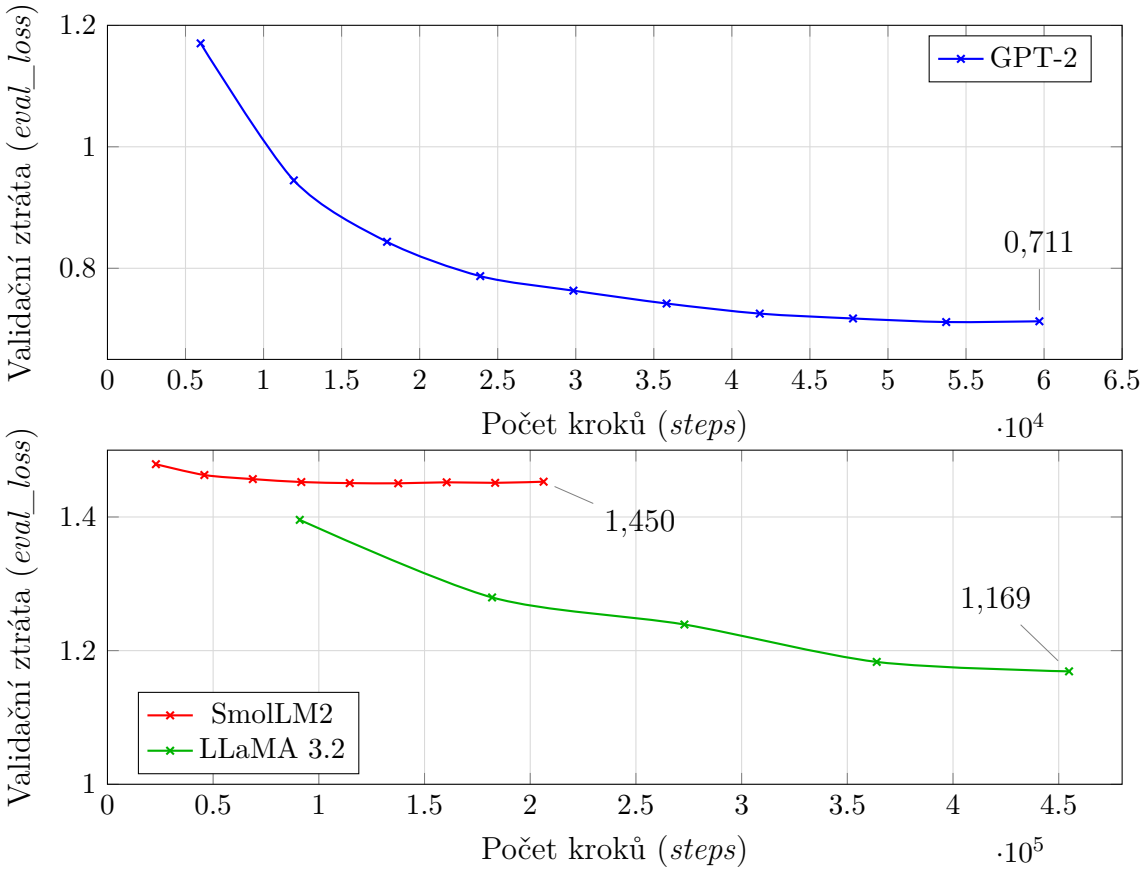
Trénink NWP modelů

Stejně jako u procesu učení MLM modelů byl spuštěn proces doladění generativních modelů, na základě zvolených parametrů v tréninkových skriptech uvedených v tabulce 3.8.

S modelem `DeepSeek-R1-8B` (i jeho menší variantou `DeepSeek-R1-3B`) bylo provedeno několik testů a úprav tréninkových parametrů, avšak jeho inicializace vedla k opakovaným chybám z důvodů nedostatečné paměťové kapacity dostupné GPU, i při pokusu o zmenšení modelu pomocí LoRA adaptace nebo optimalizace dávkování. Vzhledem k těmto omezením bylo od pokusu o doladění tohoto modelu upuštěno.



Obr. 3.19: Vývoj hodnoty ztrátovosti (`loss`) v závislosti na počtu kroků jednotlivých NWP modelů.



Obr. 3.20: Vývoj validační ztráty (`eval_loss`) v závislosti na počtu kroků jednotlivých NWP modelů.

Co se týče zbylých tří modelů, efektivita jejich doladění je shrnuta v tabulce 3.10 níže a vývoj ztrát během tréninkových kroků je zobrazen v grafech 3.19 a 3.20, kde horní část grafu ukazuje průběh pro model `GPT-2` a spodní pak srovnání modelů `SmolLM2` a `LLaMA 3.2`. Jako nejlépe proběhlý tréninkový proces se jeví pro model `GPT-2`. Tento model jako jediný úspěšně dokončil trénovací proces po celých 10 epochách v čase 1 hodina a 54 minut. Pokles hodnoty ztrátovosti v grafu naznačuje, že nastavení delšího trénování by nebylo přínosné a mohlo by tak dojít k přeučení modelu. Přesto má model `GPT-2` ze všech tří trénovaných modelů nejvyšší hodnotu ztráty (`Loss`), ve výši 0,632. Na druhou stranu dosahuje nejnižší validační ztráty a perplexity, konkrétně klesají na hodnoty 0,711 a 2,037. Tyto faktory z něj dělají vhodného kandidáta pro účely této práce, vzhledem ke stabilitě učení, zvládnutelným výpočetním nárokům a rychlosti konvergence hodnoty ztrátovosti.

Trénování modelu `Llama-3.2-3B` bylo z důvodů výpočetních nároků upraveno na pět trénovacích epoch s velikostí trénovacích a validačních dávek (*batch*) na 2. `Loss` hodnota podle grafu 3.19 měla tendenci konvergovat a výsledná hodnota 0,470 byla nakonec nižší než pro model `GPT-2`. Naopak hodnoty validační ztráty a perplexity výrazně zaostávaly, což může být způsobeno větší parametrickou složitostí modelu.

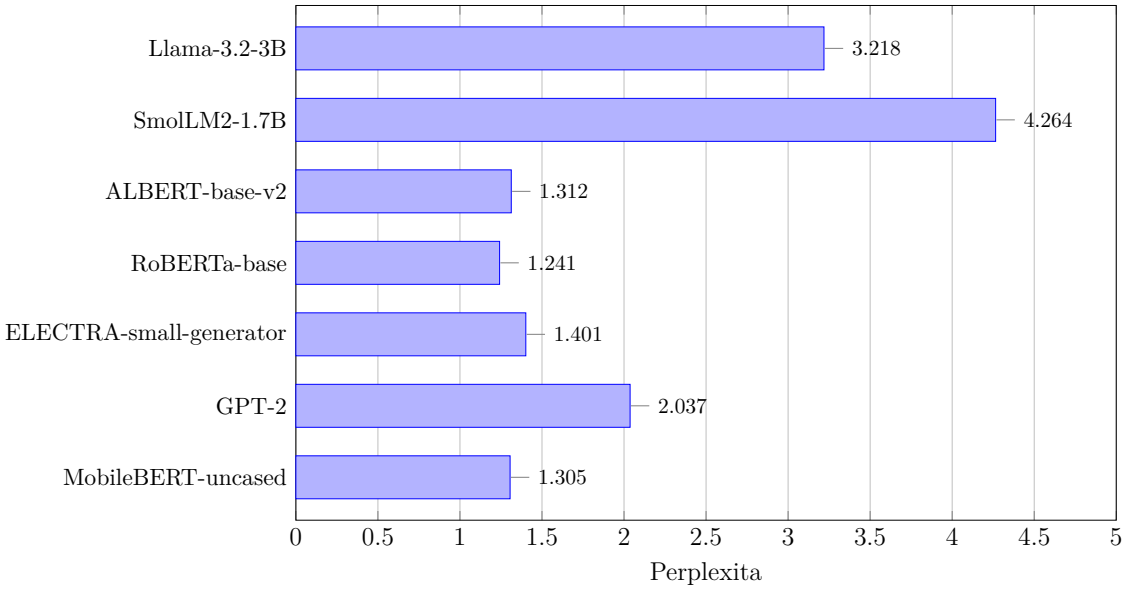
Model `SmolLM2-1.7B` se setkává s podobnými problémy. Zde bylo dosaženo pouze devíti epoch z důvodu málo konvergující hodnoty `Loss`. Každý trénovací proces je totiž doplněn o mechanismus předčasného ukončení pomocí objektu `EarlyStopping-Callback`. Tento objekt monitoruje vývoj hodnoty `Loss` a v případě její stagnace nebo zhoršení v několika epochách automaticky ukončí proces učení a zpětně načte nejlepší dostupný kontrolní bod trénování, které jsou průběžně ukládány. Ačkoliv tento model dosáhl nejnižší hodnoty `Loss`, perplexita 4,265 byla nejvyšší ze všech trénovaných modelů a jeho validační ztráta prakticky nejevila žádnou známku konvergence.

Parametry	`DeepSeek-R1-8B/3B`	`GPT-2`	`Llama-3.2-3B`	`SmolLM2-1.7B`
Epochy	-	10	5	9
Doba trvání	-	1 hod 54 min	13 hod 36 min	11 hod 30 min
Ztráta (*Loss*)	-	0,632	0,470	**0,411**
Validační ztráta (*Eval_loss*)	-	**0,711**	1,169	1,450
Perplexita (*Perplexity*)	-	**2,037**	3,219	4,265

Tab. 3.10: Hodnocení procesu učení NWP modelů.

Dosažené výsledky trénování jednotlivých velkých jazykových modelů odhalují problémy a praktická omezení vzniklá v průběhu řešení cílů této práce. Mezi nejzásadnější faktory patří především omezená dostupnost hardwaru a dostupnost výpočetní techniky s dostatečnou pamětí, a dále také nároky modelů na kvalitu, rozsah a velikost vstupní trénovací datové sady. Z těchto důvodů se jeví ladění LLM modelů v kontextu této práce jako neefektivní a výpočetně náročné modely `DeepSeek-R1` nebo `LLaMA-3.2-3B`, tak nebylo možné plnohodnotně natrénovat. Vhodnější volbou se tedy nabízí využití lehčích MLM modelů nebo dostupný generativní model `GPT-2`.

Obrázek 3.21 dále nabízí porovnání vypočtených hodnot perplexity všech trénovaných modelů. Model `RoBERTa-base` v tomto ohledu dosáhl nejlepšího výsledku, naznačující relativně vyšší schopnost orientace modelu na dané datové sadě.

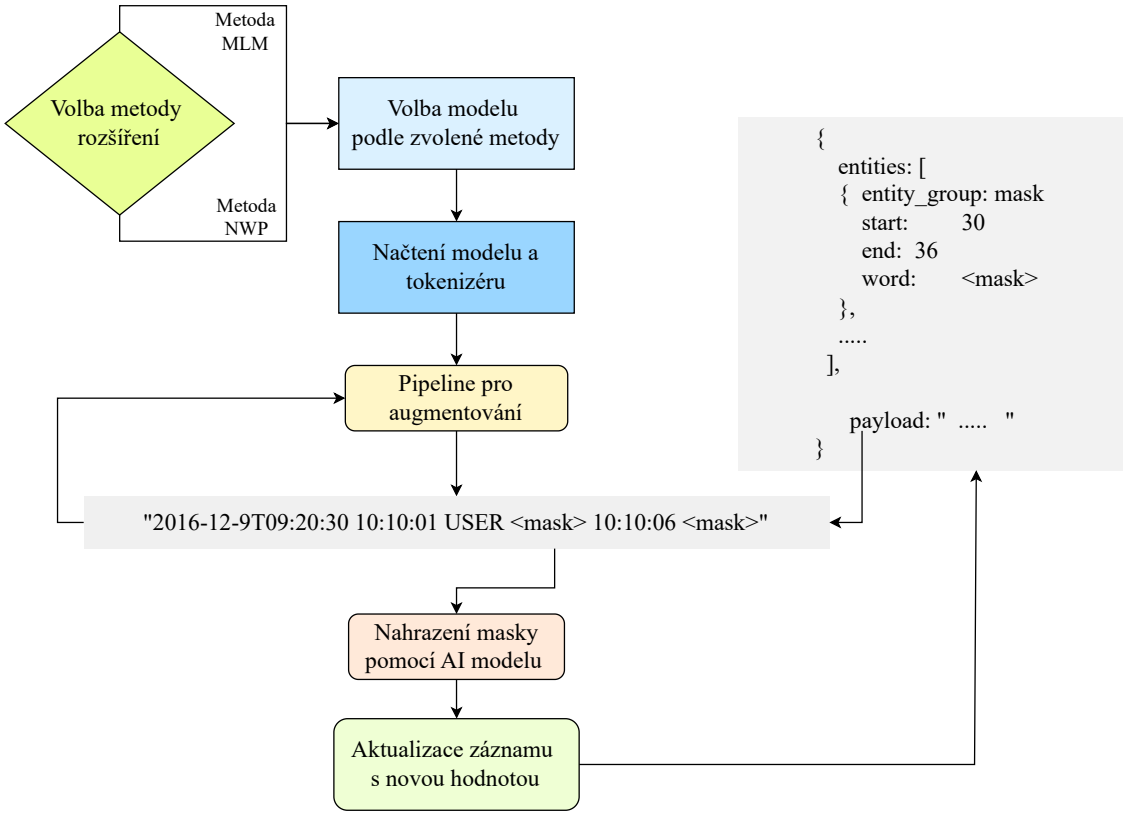


Obr. 3.21: Porovnání hodnot perplexity všech trénovaných modelů na validační množině.

Nyní se dospělo do momentu, kdy je potřeba všechny laděné modely otestovat na testovacích datech s cílem nahrazení maskovaných entit v logových záznamech. Samotné testování lze rozdělit na dvě části. V první fázi budou na vstup modelů přiřazeny logové záznamy z veřejně dostupné datové sady `Splunk Attack Data`, která obsahuje množství reálných logových záznamů. Popis této sady se nachází v kapitole 3.2.4. V druhé části budou trénované modely aplikovány na vytvořené datové sadě z kapitoly 3.2.3 a tato sada bude poté využita k argumentaci přesnosti `Transformer` sítě typu `T5` při řešení syntaktické analýzy logových záznamů. Toto testování bude popsáno a porovnáno v následující kapitole.

4 Implementace hlubokého augmentátoru a jeho validace

Hlavním cílem této práce bylo navrhnout a vytvořit takzvanou *pipeline*[1] (pracovní tok) pro načítání jazykových modelů a jejich aplikace v rámci augmentátoru pro rozšíření datových sad logových záznamů. Účelem této komponenty je jednoduše integrovat pokročilé funkce, které nabízí modely založené na strojovém učení a otestovat metodu maskování a generování textu. Vytvoření této struktury usnadní uživatelům práci s jazykovými modely pro nahrazování maskovaných hodnot a následného rozšiřování dat.



Obr. 4.1: Základní princip implementace jazykového modelu pro generování hodnot entit.

Výše přiložený obrázek 4.1 zjednodušeně znázorňuje proces rozšíření dat jazykovým modelem, který lze rozdělit do několika kroků. V první fázi je podle uživatelem zadaného typu úlohy inicializován konkrétní model a příslušný tokenizér. Následně

[1]Tento objekt umožňuje abstrahovat většinu složitostí a poskytnout snadno použitelné API určené pro různé úkoly, jako může být maskování modelovaného jazyka, analýza sentimentu a další, více viz [84].

dochází k inicializaci pracovní struktury *pipeline*. Na její vstup je přiložena textová část záznamu, která je analyzována. Pokud se v záznamu nevyskytuje žádná maska, vrací se jeho původní hodnota. Pokud však záznam maskovaný prvek obsahuje, vloží se tento log na vstup modelu, který provede typické zpracování vstupního textu ve formě tokenizace a model generuje novou hodnotu. Tato hodnota je poté převedena zpět do textové formy a dosazena do původního logu. Výstupem je potom synteticky upravený záznam připravený pro další využití.

```
masked_input = [
    {
        "entities": [
            {
                "entity_group": "level",
                "start": 6,
                "end": 13,
                "word": "<level>"
            },
            {
                "entity_group": "mask",
                "start": 67,
                "end": 73,
                "word": "<mask>"
            }
        ],
        "payload": "%FWSM-<level>-113001:01: Unable to open AAA session. Session limit <mask> reached."
    }
 ]
```

Obr. 4.2: Příklad logového záznamu obsahující maskovanou entitu.

Tato kapitola se dále zaměří na praktické testování jazykových modelů, které budou sloužit jako nástroje pro obohacení textových dat. První část bude věnována lokálnímu testování navržené struktury *pipeline* před samotným nasazením. Následně bude představena veřejně dostupná datová sada reálných záznamů událostí simulovaných v prostředí nástroje `Attack Range` a testování modelů na těchto záznamech. Na konec bude pomocí testovací sady z kapitoly 3.2.3 argumentována přesnost `Transformer` sítě typu `T5` při řešení syntaktické analýzy logových záznamů.

4.1 Lokální testování metod rozšíření logového záznamu

Tato podkapitola je logicky rozdělena na dvě části. V první části bude popsána praktická implementace metody rozšíření záznamů modelováním maskovaného jazyka (MLM). Druhá část podkapitoly bude věnována implementaci metody předpovídání dalšího tokenu (NWP) generativními modely. V obou částech budou generativní modely testovány na vzorovém záznamu z obrázku 4.2.

4.1.1 Rozšíření logových záznamů metodou MLM

Podrobnější seznámení s metodou založenou na nahrazování maskovaných entit v textu známou jako *Masked Language Modeling* již nabídla kapitola 3.4.1. Jak bylo uvedeno, tento přístup je velmi často využíván modely založenými na architektuře `BERT`, které zpracovávají text na základě oboustranného kontextu. Díky této vlastnosti modely potenciálně lépe zachycují sémantické vazby, což teoreticky umožňuje generovat kvalitnější syntetická data než augmentace pomocí metody NWP.

Testovací záznam 4.2 byl vložen na vstup struktury *pipline*, kde byl následně prohledán za účelem detekce maskované entity. Jakmile byla maska `<mask>` detekována, byla textová část záznamu předána na vstup modelu. Model provedl tokenizaci tohoto vstupu a poté generoval pět návrhů tokenů. Místo masky byl dosazen synteticky generovaný token, který je předpovídán s nejvyšším skóre. Toto skóre reprezentuje míru jistoty modelu ohledně správnosti dané předpovědi. Celá sekvence se poté převádí zpět do textové podoby a výstupem je synteticky obohacený logový záznam.

Princip načtení souboru s logovým záznamem, stažení modelu z `MLflow` serveru a následná inicializace *pipeline* je znázorněna na obrázku 4.3. Pro správnou funkčnost procesu stahování modelů z `MLflow` je nejprve zapotřebí nastavit adresu serveru pomocí funkce `mlflow.set_tracking_uri(uri=remote_server_uri)` a následně inicializovat klienta příkazem `client = MlflowClient()`. Poté může být zahájen proces stahování doladěných modelů a spuštění augmentace.

Třída `DeepAugmentator` byla implementována pro usnadnění práce s augmentačním nástrojem. Tato třída provádí veškeré nezbytné kroky od načtení modelu a příslušného tokenizátoru, inicializaci `Cuda` zařízení, pokud je dostupné, a následného volání funkce `augment` ze třídy `AI_augmentator`. Tímto způsobem lze jednoduše spustit proces rozšíření vstupní datové sekvence bez nutných znalostí vnitřní struktury kódu.

```python
# Main kód pro spuštění augmentačního procesu
if __name__ == "__main__":
    # Cesta k JSON souboru s logovým záznem
    input_file = "/8TB/MLflow/datasets/json_files/one_log_test.json"

    # Načtení dat z JSON souboru
    with open(input_file, "r", encoding="utf-8") as f:
        data = json.load(f)
        print(data)

    # Zvolení názvu modelu modelu a pod jakým aliasem je v MLflow uložen
    model_name = "mobilebert-uncased"
    model_version_alias = "newest"

    # Získání informace a verze modelu z MLFlow Clienta prostřednictvím názvu a aliasu
    model_version_info = client.get_model_version_by_alias(model_name, model_version_alias)
    model_version = model_version_info.version

    # Získání cesty k modelu a následná instalace jeho artifaktů
    model_uri = f"models:/{model_name}/{model_version}"
    local_model_path = mlflow.artifacts.download_artifacts(artifact_uri=model_uri)

    # Uložení cest artifaktů odpovídajícím modelu a tokenizéru
    model_path = f"{local_model_path}/model"
    tokenizer_path = f"{local_model_path}/components/tokenizer"

    # Inicializace @DeepAugmentator třídy, modelu, tokenizéru a spuštění @pipeline pro augmentaci
    deep_augmentator = DeepAugmentator(model_path=model_path,tokenizer_path=tokenizer_path)
    augmented_data = deep_augmentator.deep_augmentate(data, keep_mask=True)
```

Obr. 4.3: Spuštění hlavní funkce pro augmentaci metodou MLM.

Obrázek 4.4 znázorňuje generované predikce MLM modely. Každé predikci je přiřazeno skóre, se kterým model tuto hodnotu generoval. Token s nejvyšším skóre je následně zvolen jako náhrada za masku. Se svou predikcí si byl nejjistější model `RoBERTa-base`, který předpověděl token s hodnotou skóre 0,659.

```
ALBERT-base-v2
Predictions for mask at position 67-73:
1. was (0.319386)
2. is (0.190529)
3. : (0.088585)
4. out (0.049545)
5. s (0.047694)
Selected token: 'was' with score: 0.319386

ELECTRA-small-generator
Predictions for mask at position 67-73:
1. not (0.460734)
2. was (0.378790)
3. is (0.061788)
4. has (0.011290)
5. length (0.009279)
Selected token: 'not' with score: 0.460734

MobileBERT-uncased
Predictions for mask at position 67-73:
1. was (0.580861)
2. not (0.122728)
3. is (0.068246)
4. never (0.056996)
5. has (0.045248)
Selected token: 'was' with score: 0.580861

RoBERTa-base
Predictions for mask at position 67-73:
1.  is (0.658443)
2.  already (0.129408)
3.  was (0.096224)
4.  not (0.048255)
5.  has (0.038954)
Selected token: ' is' with score: 0.658443
```

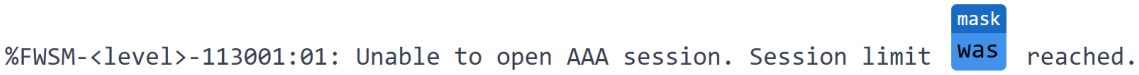
Obr. 4.4: Pět generovaných tokenů MLM modely pro vložený maskovaný logový záznam.

Převedené textové výstupy ve formě obohacených záznamů, uložené v objektu `augmented_data`, jsou přehledně zobrazeny[2] na obrázku 4.5. Každý přiložený příklad ukazuje, jak konkrétní model nahradil původní masku svým generovaným tokenem. Je evidentní, že jednotlivé doladěné modely generovaly slovní výrazy, avšak z kontextu daného záznamu lze usoudit, že by bylo vhodnější predikovat číselný údaj (například vztahující se k množství aktivních relací), což by z pohledu sémantiky bezpečnostního záznamu dávalo větší smysl.

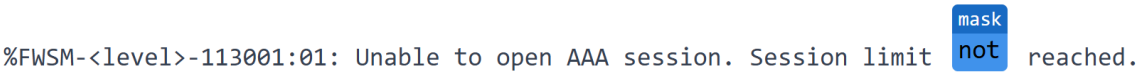
Dochází tedy k jistému omezení daných modelů, který upozorňuje na možnou potřebu hlubšího ladění, vylepšení trénovací datové sady nebo využití mnohem většího modelu. Pro tento konkrétní kontext se z hlediska generování relevantní hodnoty

[2]Jednotlivé logové záznamy jsou anotovány prostřednictvím vyvinuté aplikace v rámci výzkumného projektu s názvem *Platforma pro adaptivní dolování znalostí z logových záznamů pomocí technik umělé inteligence (VB02000059)*.

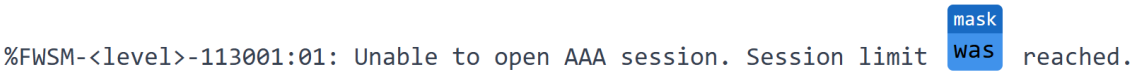
sice nejedná o ideální výstupy, ale v rámci textové augmentace a syntetického rozšíření dat tyto rozdíly nepředstavují zásadní problém. Modely proto budou i nadále využity k dalšímu testování.



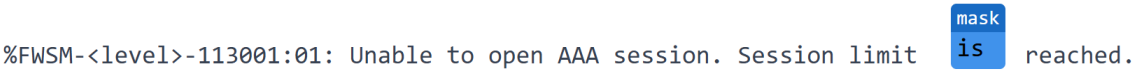
a) Záznam doplněný trénovaným modelem `ALBERTA`.



b) Záznam doplněný trénovaným modelem `ELECTRA`.



c) Záznam doplněný trénovaným modelem `MobileBERT`.



d) Záznam doplněný trénovaným modelem `RoBERTa`.

Obr. 4.5: Příklady doplněných masek logových záznamů trénovanými MLM modely.

4.1.2 Rozšíření logových záznamů metodou NWP

Druhou metodou rozšíření logových záznamů pomocí hlubokých neuronových sítí typu `Transformer` byla metoda *Next Word Prediction* představena v kapitole 3.5.2. Jelikož modely založené na této metodě pracují na autoregresivním principu, viz obrázek 3.8, přístup k samotné augmentaci je komplikovanější než tomu bylo u metody doplňování masek. Tokenizér těchto modelů totiž neobsahuje speciální token reprezentující masku, a proto je nutné explicitně modelu předat informaci o očekávaném výstupu.

Prvním navrženým přístupem k této metodě bylo oříznutí textu záznamu na pozici před výskytem masky. Následně model generoval syntetickou hodnotu, která nahradila masku. Za tuto hodnotu byl poté připojen zbytek původního textu. Tento proces měl iterativně pokračovat, dokud by nebyly všechny masky v záznamu nahrazeny. Pro tuto funkci byl vytvořen vstupní textový *prompt*, který se spolu s textovou částí logového záznamu, ve kterém má být provedeno nahrazení maskované hodnoty, předává modelu. Cílem tohoto vstupního textu je pomoci specifikovat daný úkol a nasměrovat model ke generování kvalitnějších hodnot. Obrázek 4.6 znázorňuje podobu takto definovaného *promptu*. Zobrazení v konzolovém výstupu ve stejném

obrázku ukazuje jeho kombinaci s textem záznamu 4.2, který byl oříznut o hodnoty *<mask> reached.* Tento přístup se ale v praxi neosvědčil, jelikož modely často generovaly nadměrné množství textu, což poté vedlo ke ztrátě konzistence a formátu původního logu. Výsledné výstupy poté nebyly vhodné k dalším syntaktickým nebo sémantickým analýzám.

```
prompt = (
  f"You are an expert in text augmentation specialized in augmenting log entries. "
  f"The payload where you should generate new entity word or number that fits the
context of the log instead of original '<mask>' is: {payload[:start]}"
)

================================================================================
Prompt: You are an expert in text augmentation specialized in augmenting log entries.
The payload where you should generate new entity word or number that fits the context
of the log instead of original '<mask>' is: %FWSM-<level>-113001:01: Unable to open
AAA session. Session limit
```

Obr. 4.6: Vytvořená vstupní posloupnost s oříznutým text logového záznamu.

Druhý navržený přístup znovu funguje na principu *promptování*. V tomto případě je navržen ve formě systémového *promptu*, který inicializuje konverzační rozhraní a rozdělí role na systém a uživatele. Tedy místo explicitní výzvy s kontextem je systémový *prompt* vložen přímo do modelu a definuje tak jeho požadované chování. Uživatel pak zadává pouze vlastní textovou část záznamu, jehož maskované hodnoty mají být nahrazeny.

V tomto případě je poskytnuta větší kontrola nad výstupem modelu a je také odstraněn problém s přebytečným generováním textu. Vstupní instrukce přímo určují roli modelu a stanovují pravidla, kterými se má řídit. Důležité je zajistit, aby model generoval pouze hodnoty pro maskované tokeny, přičemž zachová strukturu původního záznamu. Vstupní systémový *prompt* je uveden na obrázku 4.7 níže.

```
messages = [
  {
    "role": "system",
    "content":
    (
          "You are an expert in log analysis and text augmentation. "
          "Your task is to replace only <mask> tokens in system log entries with relevant values while maintaining
             log structure. "
          "Ensure to replace each <mask> with a single word or phrase that fits the context of the log entry and
             do not change structure of the log entry. "
          "You should not add any additional information or context outside of the log entry. "
          "Your response should only include the log entry with the replaced <mask> tokens. "
          "Do not include any other text or explanations. "
    )
  },
  {
    "role": "user",
    "content": f"{payload}"
  }
]
```

Obr. 4.7: Vytvořený systémový prompt.

Bohužel možnost využití konverzačního formátu poskytují pouze modely, jejichž tokenizér takovýto formát podporuje. Model typu `GPT-2` tyto funkce neposkytuje. Proto bylo potřeba u tohoto modelu systémový a uživatelský *prompt* spojit do jednoho řetězce a vložit na vstup modelu jako jediný textový vstup. Takovýto přístup sice neumožní přímé rozdělení rolí jako v případě využitých modernějších konverzačních modelů `Llama-3.2-3B` nebo `SmolLM2`, ale umožní model nasměrovat k požadovanému chování pomocí předem nastavených instrukcí umístěných na začátku vstupního textu.

Pro vytvořenou vstupní instrukci 4.7 byly testovány tři trénované modely z kapitoly 3.5.3. Na vstup modelů byl vložen testovací záznam 4.2 s cílem jeho rozšíření. Vstupní systémová instrukce zajistila, že modely byly schopny vrátit doplněnou masku původního logového záznamu. Textový výstup modelů byl pro přehlednost generované hodnoty anotován a je postupně znázorněn na obrázku 4.8.

```
%FWSM-<level>-113001:01: Unable to open AAA session. Session limit [mask: Reason: Failure. This event occurs when a session] reached.
```

a): Záznam doplněný trénovaným modelem `GPT-2`.

```
%FWSM-<level>-113001:01: Unable to open AAA session. Session limit [mask: %FWSM-3-113001:] reached.
```

b): Záznam doplněný trénovaným modelem `Llama-3.2-3B`.

```
%FWSM-<level>-113001:01: Unable to open AAA session. Session limit [mask: FWSM-<level>-11] reached.
```

c): Záznam doplněný trénovaným modelem `SmolLM2-1.7B`.

Obr. 4.8: Příklady doplněných masek logových záznamů trénovanými NWP modely.

Jak lze z těchto výsledků usoudit, trénink těchto velkých jazykových modelů nepřinesl zlepšení při generování syntetické hodnoty a pro jednoduchý logový záznam tyto modely selhávají. Již v kapitole 3.5.3 bylo uvedeno, že samotný proces učení velkých jazykových modelů je výpočetně náročný a v případě této práce je jejich ladění neefektivní. Z těchto důvodů bylo nad rámec cílů této práce integrováno API[3] na platformu `Ollama`. Tento *open-source framework* umožňuje uživatelům spouštět velké jazykové modely přímo na jejich lokálních systémech. Jednoduché API rozhraní tak usnadní využití, nasazení a správu velkých předpřipravených modelů v různých aplikacích, jako je například nahrazování masek v logových záznamech.

API platformy `Ollama` bylo napojeno na vytvořenou *pipeline*, která již byla nasazena pro úlohu doplňování masek pomocí MLM modelů. Tato integrace vyžadovala vytvoření jednoduchého systémového promptu, podobného výše zmíněnému 4.7, se kterým bude vybraný model pracovat.

```
messages = [
    {
    "role": "system",
    "content":
    (
        "You are the world's best professional log file analyst. "
        "I will provide you with a log where each line contains a <mask> token. "
        "Your task is to generate suitable replacements for each <mask>. "
        "Each generated value should be placed on a new line, in the same order as the masks appear. "
        "Output only the generated values for <mask>, nothing else – no explanations or additional text."
        "Dont use any words in user prompt."
    )
},
    {
        "role": "user",
        "content": payload
    }
]
```

Obr. 4.9: Nastavení systémové a uživatelské instrukce v API prostředí.

Na obrázku 4.9 lze vidět nastavení systémového *promptu*. Cílem modelu je nahrazovat masky, proto byla tato vstupní instrukce vytvořena tak, aby model generoval pouze hodnoty odpovídající jednotlivým maskám v textu, protože budou tyto hodnoty dále strojově zpracovány. Uživatelský vstup odpovídá hodnotě samotného logového záznamu, který se následně vkládá na vstup modelu. Po této jednoduché inicializaci je modelem generován požadovaný počet hodnot, odpovídající počtu výskytu masek. Zdrojový kód následně ověřuje, kolik masek model nahradil. Pokud počet generovaných hodnot neodpovídá počtu vyskytujících se masek, sníží se hodnota teploty (viz úryvek kódu na obrázku 4.10), která ovlivňuje rozmanitost generování modelem. V tomto případě se proces opakuje, dokud model nenahradí

[3]API, neboli *Application Programming Interface*, je mechanismus umožňující dvěma softwarovým komponentám vzájemně komunikovat pomocí souboru pravidel a protokolů [85].

přesný počet masek. V případě, že model nebyl schopen nahradit správný počet hodnot ani po několika pokusech, je nutné tyto logové záznamy manuálně filtrovat.

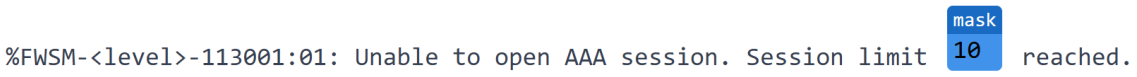
```
if len(replacements) == mask_count:
    logger.info(f"Generated {len(replacements)} replacements successfully.")
    return replacements
else:
    logger.warning(f"Generated {len(replacements)} replacements, expected {mask_count}.")
    logger.warning(f"Reducing temperature to {temperature - 0.1}")
    temperature = round(temperature - 0.1, 1)
```

Obr. 4.10: Snížení teploty v případě špatného počtu generovaných hodnot.

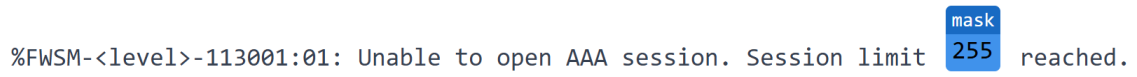
Velkou výhodou API integrace je možnost využití výrazně větších modelů, jejichž doladění na dostupné grafické kartě je prakticky nemožné. Lze tak například využít velký jazykový model řady `Llama3:70B`, který obsahuje 70 miliard parametrů, nebo model `DeepSeek-R1:32B` s 32 miliardami parametrů. Z těchto důvodů byla dosavadní *pipeline*, nyní nasazená pouze pro metodu MLM, rozšířena o generativní metodu využívající uvedené API rozhraní.

Pro tento účel byla vytvořena třída `DeepOllamaAugmentator`, která funguje podobně jako u metody MLM třída `DeepAugmentator`. Umožňuje načítání modelů z `Ollama` serveru a inicializaci procesu rozšíření prostřednictvím funkce `generate_fill_mask`, jejímž vstupem je záznam událostí s maskou a parametr `keep_mask`. Funkce `_generate_replacements`, která je rovněž implementována ve třídě `DeepOllamaAugmentator` zajišťuje samotné generování nové hodnoty pomocí vytvořeného systémového *promptu* a dalších zadaných instrukcí. Uvnitř této funkce se také nastavuje hodnota teploty pro určení rozmanitosti generování modelem.

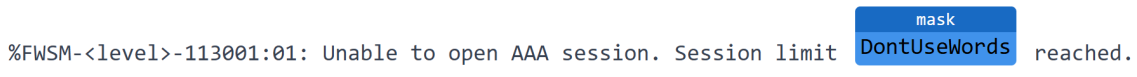
Pro testování byl opět využit testovací záznam 4.2, vložený na vstup vybraných modelů. Výstupem byl úspěšně rozšířený záznam, jehož maska byla nahrazena konkrétní hodnotou, podobně jako tomu bylo u MLM modelů. V případě tohoto vzorku dosahují modely `DeepSeek-R1:32B` a `Llama3:70B` na první pohled kvalitnějšího výsledku než ostatní modely, jelikož z kontextu celého záznamu bylo možné na místě masky očekávat číselnou hodnotu. Na druhou stranu model `SmolLM2-1.7B` znovu generoval kontextuálně irelevantní hodnotu, jako stejný doladěný model.



a) Záznam doplněný modelem `DeepSeek-R1:32B`.



b) Záznam doplněný modelem `Llama3:70B`.



c) Záznam doplněný modelem `SmolLM2-1.7`.

Obr. 4.11: Příklady doplněných masek logových záznamů velkými jazykovými modely skrz `Ollama` API.

Na základě provedeného lokálního testování a subjektivních hodnocení těchto metod a jejich výsledků generování na testovacím vzorku bylo usouzeno, že pro další testy budou využity natrénované modely založené na principu metody modelování maskovaného jazyka. Konkrétně se jedná o modely `RoBERTa-base`, `ALBERT-Base-v2`, `MobileBERT-uncased` a `ELECTRA-small-generator` a dále o velké jazykové modely `Llama3:70B` a `DeepSeek-R1:32B`, dostupné prostřednictvím vytvořeného API rozhraní na serveru `Ollama`.

4.2 Testování modelů na simulovaných záznamech nástrojem Atomic Red Team

Z datové sady Splunk Attack Range, představené v kapitole 3.2.4, bylo vybráno 10 záznamů událostí různých formátů. Cílem této části je otestovat natrénované MLM modely z kapitoly 3.5 a modely dostupné na platformě `Ollama` na dosud neviděných datech a ověřit tak jejich schopnosti generalizace. Vybrané záznamy v jejich původním formátu jsou uvedeny v příloze B a jejich stručný popis[4] nabízí tabulka 4.2 níže.

[4]Názvy útoků vychází z klasifikace MITRE ATT\&CK a pro zachování terminologické přesnosti jsou ponechány v původním anglickém jazyce. Viz oficiální stránky `https://attack.mitre.org/techniques/enterprise/`

Záznam	**B.1**
MITRE ID	T1585
Technika	*Establish Accounts*
Formát	`Windows Event Log`
Obsah	Ukončení `PowerShell` procesu spuštěného správcem.

Záznam	**B.2**
MITRE ID	T1550.002
Technika	*Use Alternate Authentication Material – Pass the Hash*
Formát	`Windows System Log`
Obsah	Přechod služby `Windows Update` do stavu zastaveno.

Záznam	**B.3**
MITRE ID	T1003.002
Technika	*OS Credential Dumping – Security Account Manager*
Formát	`Windows System Log`
Obsah	Program `bootmgr` nečekal na vstup uživatele.

Záznam	**B.4**
MITRE ID	T1562.001
Technika	*Impair Defenses – Disable or Modify Tools*
Formát	`PowerShell Operational Log`
Obsah	Skript `PowerShell` spuštěn jako `ScriptBlock` s ID: c842dcf0-e3e9-4b27-a5b4-d19749f6a53d

Záznam	**B.5**
MITRE ID	T1068
Technika	*Exploitation for Privilege Escalation*
Formát	`Sysmon for Linux (XML)`
Obsah	Bylo zjištěno zahájení neidentifikovaného procesu uživatelem `ubuntu`.

Tab. 4.1: Popis vybraných záznamů ze Splunk Attack Range datové sady část 1.

Záznam	**B.6**
MITRE ID	T1014
Technika	*Rootkit*
Formát	`Windows XML Event Log`
Obsah	Byla detekována instalace kernelového ovladače `dbutil_2_3.sys`, což je charakteristické chování pro `rootkit`.
Záznam	**B.7**
MITRE ID	T1136
Technika	*Create Account*
Formát	`Linux syslog`
Obsah	Detekce vytvoření nového uživatele `eviluser`.
Záznam	**B.8**
MITRE ID	T1190
Technika	*Exploit Public-Facing Application*
Formát	`JSON`
Obsah	Zaznamenán provoz protokolu `HTTP`. Tento provoz naznačuje zneužití zranitelnosti `ColdFusion` (CVE_2023_29360) z IP adresy `2.1.2.3` na cílový server s IP adresou `10.2.2.2`.
Záznam	**B.9**
MITRE ID	T1654
Technika	*Log Enumeration*
Formát	Splunk textový log
Obsah	Pokus o ověření prázdného JWT tokenu v prostředí Splunk.
Záznam	**B.10**
MITRE ID	T1212
Technika	*Exploitation for Credential Access*
Formát	`Webový záznam`
Obsah	`HTTP` dotaz na `http://5.188.210.227/echo.php` z externí IP, který může naznačovat pokus o útok typu *Remote File Inclusion*.

Tab. 4.2: Popis vybraných záznamů ze Splunk Attack Range datové sady část 2.

V každém z těchto vybraných záznamů byly náhodně zamaskovány hodnoty. Tyto hodnoty jsou v každém záznamu v příloze B zvýrazněny červenou barvou. Po zamaskování byly záznamy postupně předloženy na vstup generativních modelů zvolených na konci předchozí kapitoly. Generované a původní hodnoty byly následně vyhodnoceny na základě několika výpočetních metrik, kterými jsou:

- `BERTscore` umožňující výpočet metrik *F1-Score*, *Precision* a *Recall* počítané na jednotlivou generovanou hodnotu.
- Průměrné hodnoty *F1-Score*, *Precision* a *Recall*, počítány z metrik jednotlivých hodnot.
- *Levenshtein Distance* (*Levenshteinova* vzdálenost) pro jednotlivé hodnoty a následně její průměr skrze celou sadu.
- *Cosine similarity* (Kosinová podobnost) a skóre BLEU (*Bilingual Evaluation Understudy*).

`BERTscore`

Tato hodnotící metrika generování textu počítá skóre podobnosti každého tokenu predikované hodnoty s tokenem referenčním. Místo přesných shod pracuje s podobností tokenů pomocí kontextuálních *embeddingů*. `BERTscore` využívá předtrénované kontextové vnoření modelu `BERT` a porovnává predikované a referenční hodnoty pomocí kosinové podobnosti. Je tedy schopno korelovat s lidským posudkem při hodnocení na úrovni vět a systémů. Tato metoda nabízí výpočet přesností *Precision*, *Recall* a také F1 skóre. Jako omezení této metriky hodnocení generovaného textu je možné selhávání na úrovni porozumění kontextu v rámci jednotlivých slov [86].

Levenshtein Distance

Tato metrika vyjadřuje míru podobnosti dvou řetězců, která zohledňuje počet operací vložení, vymazání nebo záměny potřebných k přeměně jednoho řetězce na druhý [87]. V kontextu této práce platí, že čím je hodnota vzdálenosti menší, tím přesněji model generoval dané slovo ve srovnání s původní entitou. Jednoduchým příkladem může být situace:

1. `Levenshtein`(„start“, „start“) = 0 (identické řetězce);
2. `Levenshtein`(„start“, „stord“) = 2 (dvě změny – záměna „ö“ za „a“, „d“ za „t“);
3. `Levenshtein`(„start“, „end“) = 5 (jiné slovo, vyžaduje operace mazání, vkládání, nahrazení);

Cosine similarity

Kosinová podobnost měří míru podobnosti mezi dvěma nenulovými vektory výpočtem kosinu úhlu mezi nimi. Je široce používána v oblasti analýzy dat a strojového učení, například při analýze textu, porovnávání dokumentů nebo vyhodnocení významové shody textových vektorů. Podobně jako u *Levenshteinova* vzdálenosti platí, že čím menší je vzdálenost, tím je vyšší podobnost, a naopak větší vzdálenost znamená nižší podobnost. Kosinová podobnost je výhodná i v situacích, kdy jsou si dva

datové objekty významově podobné, avšak v Eukleidovském prostoru jsou od sebe vzdálené – vzhledem k malému úhlu mezi nimi mohou i přesto vykazovat vysokou míru podobnosti. Na druhou stranu problémem může být citlivost na řídká data [88]. Vzorec pro zjištění kosinové podobnosti je následující:

$$S_C(x, y) = \frac{x \cdot y}{\|x\| \times \|y\|}$$

kde $x \cdot y$ je skalární součin vektorů „x“ a „y“., $\|x\|$ a $\|y\|$ je délka (velikost) dvou vektorů „x“ a „y“ a $\|x\| \times \|y\|$ je vektorový součin dvou vektorů „x“ a „y“.

BLEU skóre

Bilingual Evaluation Understudy (BLEU), je algoritmus hodnocení kvality strojově přeloženého textu. Hlavní myšlenkou je, že čím více se strojový překlad shoduje s referenčním textem, tím je hodnota skóre lepší. Za kvalitu se tedy považuje shoda mezi výstupem strojového překladu a výstupem člověka. BLEU však porovnává pouze povrchovou shodu tokenů mezi predikcemi a referencemi, aniž by bral v úvahu jejich skutečný význam. Potenciál dosažení vyššího skóre mají kratší sekvence, jelikož je úplná shoda v menším prostoru pravděpodobnější [89].

Predikované hodnoty modelem `ALBERT-Base-v2` a referenční entity jsou porovnány a hodnoceny v tabulce 4.3 v pořadí, v jakém byly maskovány.

Pro zachování přehlednosti byly predikce zbývajících modelů `ELECTRA-small-generator`, `MobileBERT-uncased`, `RoBERTa-base`, `Llama3:70B` a `DeepSeek-R1:32B` zařazeny do přílohy C.

Z těchto tabulek lze vidět, že MLM modely vykazují omezení při generování víceslovných nebo specifických entit, jako jsou časové značky nebo kompletní data, protože jsou schopny standardně generovat pouze jeden token. Proto, pokud by bylo zapotřebí predikovat více tokenů, například celé datum nebo IP adresu atp., bude nutné do dané vložit více masek. To lze vidět například v záznamu B.9, kde bylo počáteční datum maskováno jako `<mask>-12-<mask>`, nebo v záznamu B.10, kde byla IP adresa maskována formou `5.188.<mask>.<mask>`. V těchto případech dosahují MLM modely velmi přesných hodnot.

V tomto ohledu přináší generativní modely výhodu, jelikož jsou schopny generovat složitější řetězce, jako jsou hashe, časové značky, IP a MAC adresy nebo souborové cesty. Názornou ukázkou jsou predikované hodnoty, umístěné v tabulce C.5, modelem `Llama3`, který je schopen v záznamu číslo 2 generovat časovou značku, v záznamu číslo 7 souborovou cestu nebo v záznamu 8 celou IP adresu. Navzdory jejich generativní schopnosti zůstává omezením velká výpočetní i časová náročnost a potenciálně menší konzistence výstupu, což může vyžadovat kontrolu rozmanitosti generace.

Log Index	Entity	Original Entity	Augmented Entity	F1 Score	Precision	Recall	Levenshtein Distance
B.1	1	9/17/2020	04	0.7877	0.8613	0.7257	9
	2	Information	generated	0.9685	0.9685	0.9685	8
	3	administrator	guest	0.9993	0.9993	0.9993	11
	Mean			0.9185	0.9430	0.8978	9.33
B.2	1	12:15:24	:08	0.8076	0.8212	0.7944	7
	2	7036	0	0.8777	0.9222	0.8373	3
	3	successfully	.	0.8366	0.8366	0.8366	12
	4	80474	1	0.8882	0.9310	0.8491	5
	Mean			0.8525	0.8777	0.8294	6.75
B.3	1	System	-	0.9083	0.9083	0.9083	6
	2	NOT_TRANSLATED	0	0.8043	0.8864	0.7361	14
	3	None	0	0.9772	0.9772	0.9772	4
	4	0	2	0.9796	0.9796	0.9796	1
	5	user	user	1.0000	1.0000	1.0000	0
	Mean			0.9339	0.9503	0.9203	5.00
B.4	1	Microsoft	0	0.9765	0.9765	0.9765	9
	2	Windows	0	0.9789	0.9789	0.9789	7
	3	4105	0	0.8589	0.9136	0.8103	3
	4	Command	task	0.9963	0.9963	0.9963	6
	Mean			0.9526	0.9663	0.9405	6.25
B.5	1	Linux	lin	0.9987	0.9987	0.9987	3
	2	5	40	0.9993	0.9993	0.9993	2
	3	sysmonlinux	it	0.8252	0.8812	0.7759	10
	4	ubuntu	local	0.8454	0.8942	0.8016	6
	Mean			0.9172	0.9434	0.8939	5.25
B.6	1	Service	service	0.9993	0.9993	0.9993	1
	2	Manager	manager	0.9989	0.9989	0.9989	1
	3	7045	1	0.8857	0.9385	0.8384	4
	4	2023	20	0.8831	0.9250	0.8448	2
	5	09	22	0.9997	0.9997	0.9997	2
	6	mode	mode	1.0000	1.0000	1.0000	0
	7	start	start	1.0000	1.0000	1.0000	0
	Mean			0.9667	0.9802	0.9545	1.42
B.7	1	eviluser	:	0.8198	0.8574	0.7853	8
	2	/home/eviluser	root	0.7843	0.8196	0.7518	13
	Mean			0.8020	0.8385	0.7686	10.50
B.8	1	903006245578527	"	0.7905	0.8590	0.7321	15
	2	2.1.2.3	s	0.8724	0.8774	0.8675	7
	3	TCP	no	0.9983	0.9983	0.9983	3
	4	true	"	0.9988	0.9988	0.9988	4
	5	true	"	0.9988	0.9988	0.9988	4
	6	true	"	0.9988	0.9988	0.9988	4
	Mean			0.9429	0.9552	0.9324	6.16
B.9	1	03	20	0.9990	0.9990	0.9990	2
	2	2024	03	0.9989	0.9989	0.9989	3
	3	None	get	0.9993	0.9993	0.9993	4
	Mean			0.9991	0.9991	0.9991	3.00
B.10	1	210	22	0.9933	0.9933	0.9933	2
	2	227	0	0.9753	0.9753	0.9753	3
	3	HTTP	:	0.8717	0.8717	0.8717	4
	4	google	html	0.9915	0.9915	0.9915	5
	5	KHTML, like Gecko	none	0.8150	0.8485	0.7841	16
	Mean			0.9294	0.9361	0.9232	6.00

Tab. 4.3: Generované hodnoty modelem `ALBERT-Base-v2` část 1.

Jak bylo zmíněno, `BERTscore` může selhávat u hodnocení jednotlivých tokenů, což se v tomto případě prokázalo zejména v případě, kdy modely místo textových řetězců generovaly číselné hodnoty. Například u logového záznamu B.3 generoval model `ALBERT` pomlčku a dvě nuly místo textového řetězce. Zatímco v případě hodnoty „*none*“ může být číselná hodnota „0“ sémanticky akceptovatelná a metrika může dosáhnout vyšší přesnosti, u jiných případů, jako u pomlčky místo hodnoty „*System*“ nebo „0“ místo „*NOT_TRANSLATED*“ toto hodnocení selhává a dosahuje pak velmi vysokých hodnot. Podobná situace nastává i v případě záznamu B.4.

Pro tabulky generovaných hodnot a jejich přesností byly následně vypočítány jejich průměrné hodnoty. Srovnání těchto průměrných metrik napříč deseti záznamy pro šest jazykových modelů nabízí tabulka 4.4. Model `DeepSeek-R1:32B` dosáhl nejlepších výsledků z hlediska přesnosti generace. Konkrétně dosáhl nejvyšších hodnot F1 skóre (0,9284) a *Recall* (0,9287). Zároveň vykázal také nejnižší kosinovou podobnost (0,0899) mezi všemi generovanými entitami, což ukazuje na jeho schopnost generovat sémanticky relevantní hodnoty. Po nahlédnutí do tabulky C.5 lze usoudit, že tento model je opravdu schopen velmi přesného nahrazování masek v záznamech. `RoBERTa-base` model byl schopen nahrazovat hodnoty s nejvyšším BLEU skóre (0,3464) a s průměrně nejmenším počtem změn mezi referenční a predikovanou hodnotou měřenou *Levenshtein* vzdáleností (18,70), avšak za cenu nejnižší celkové přesnosti (F1 = 0,8933). U tohoto modelu se také objevují situace, kdy model generoval před samotnou hodnotu také mezeru a příklad této situace lze vidět v případě augmentovaných logů v tabulce C.3. Model `Llama3:70B` generoval hodnoty s nejvyšší průměrnou *Levenshtein* vzdáleností, což indikuje vyšší míru rozmanitosti a kreativity v jeho generovaných hodnotách. Navzdory tomu, že může v některých případech docházet k odchýlení od referenčních hodnot, tato diverzita může být při syntetickém rozšíření dat výhodná, zejména v případech úloh s požadavkem širší variability výstupů.

Metrika	`ALBERT`	`ELECTRA`	`MobileBERT`	`RoBERTa`	`Llama`	`DeepSeek`
F1 Score	0,9215	0,9077	0,9167	0,8933	0,8949	**0,9284**
Precision	**0,9390**	0,9215	0,9317	0,9082	0,8792	0,9299
Recall	0,9060	0,8965	0,9031	0,8802	0,9139	**0,9287**
Cosine Similarity	0,0233	0,0563	0,0429	0,0873	0,0510	**0,0899**
BLEU Score	0,0486	0,1519	0,1386	**0,3464**	0,1469	0,2226
Levenshtein Distance	23,30	20,40	21,00	**18,70**	33,60	22,80

Tab. 4.4: Porovnání výkonu jednotlivých modelů podle různých metrik.

Na závěr je tedy potřeba říci, že hodnocení takto generovaných entit pomocí běžných metrik, využívaných v oboru zpracování přirozeného jazyka, nemusí vždy

poskytovat plně relevantní výsledky. V některých případech mohou být výsledné hodnoty mírně zavádějící, zejména pak při zohlednění sémantické správnosti nebo kontextuálního významu entity v rámci logového záznamu. V takovém případě je nutné doplnit automatická vyhodnocení o subjektivní posouzení ze strany člověka, který je schopen lépe reflektovat významové souvislosti textu. Na základě provedeného subjektivního vyhodnocení lze konstatovat, že generativní jazykové modely představují velmi vhodný nástroj pro rozšiřování textu o syntetická data. Velké jazykové modely, obsahující několik miliard parametrů, jsou schopny překonat menší modely založené na principu modelování maskovaného jazyka z hlediska kvality generování entit.

4.3 Vyhodnocení testovací datové sady úlohou NER

Posledním hodnocením je argumentace přesnosti modelu `T5` při doladění na klasifikační úloze rozpoznávání pojmenovaných entit rozšířené testovací sady, uvedené v kapitole 3.2.3, vybranými modely. Objekty reálného světa, jako jsou místa, lidé a produkty, se ve zpracování přirozeného jazyka (NLP) nazývají pojmenované entity. Extrakce těchto entit z textu je známá jako metoda rozpoznání pojmenovaných entit (*Named Entity Recognition*, NER). Toto vyhledávání entit lze formulovat jako přiřazení značky každému tokenu tak, že pro každou entitu existuje jedna třída a jedna třída pro „*no-entity*“ [36, 90].

Pro určení přesnosti syntaktické analýzy logových záznamů byl využit model `flan-T5-small` společnosti Google. Tento model představuje vylepšenou verzi původního modelu `T5`, stručně představeného v kapitole 3.4.2. V porovnání s původní verzí byl `flan-T5-small` doladěn na více než 1000 různorodých úlohách zahrnující různé jazyky, přičemž zachovává stejný počet parametrů [91]. Celý proces hodnocení lze shrnout do tří kroků:

1. Nahrazení masek ve vstupních záznamech událostí testovací sady pomocí pokročilého augmentátoru založeného na jazykových modelech;
2. Doladění modelu `flan-T5-small` na úloze rozpoznání pojmenovaných entit nad doplněnou datovou sadou;
3. Vyhodnocení úspěšnosti klasifikace na základě dosažených metrik: F1 skóre, přesnosti (*precision*), citlivost (*recall*) a analýza pomocí matice záměn.

Dále se tato kapitola bude zabývat jednotlivými výsledky procesu doladění modelu `flan-T5-small` pro pět augmentovaných testovacích datových sad.

Z důvodu výpočetních nároků a náročnosti strojového času využití velkých jazykových modelů byly pro augmentaci datové sady využity modely: doladěné MLM modely `RoBERTa-base`, `ALBERT-Base-v2`, `MobileBERT-uncased` a `ELECTRA-small-`

generator. Z modelů umožňujících techniku NWP byl použit model Llama3:70B skrz integrované API platformy Ollama.

Model rozšíření	**Počet vzorků** (*Train, Test*)	**Doba trénování** (Počet epoch)	**F1-skóre** (%)	**Přesnost** (%)	**Citlivost** (%)
ALBERT	67385, 16847	1 hod 36 min (12)	99,900	99,865	99,936
ELECTRA	67385, 16847	2 hod 36 min (20)	99,962	99,953	99,966
LLama3	63928, 15982	1 hod 36 min (13)	99,861	99,792	99,929
MobilBERT	67385, 16847	2 hod 36 min (20)	**99,970**	**99,967**	**99,973**
RoBERTa	67385, 16847	2 hod 18 min (18)	99,956	99,950	99,963
Původní sada	24833, 6209	37 min 30 sec (13)	99,877	99,827	99,926

Tab. 4.5: Doladění modelu flan-T5-small na augmentovaných datových sadách různými modely.

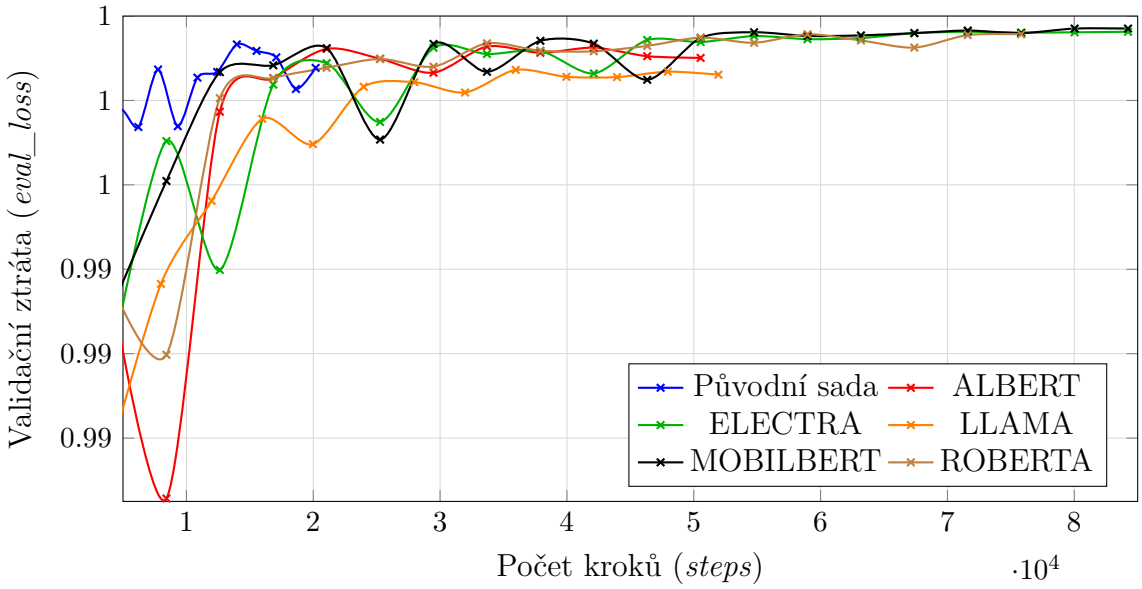
Tabulka 4.5 názorně srovnává velikost trénovacích a validačních sad, využitých k ladění modelu flan-T5-small. Modely založené na metodě modelování maskovaného jazyka jsou schopny doplnit všechny masky obsažené v datasetu. Původní sada, která dohromady obsahuje 31 042 záznamů událostí, tak byla rozšířena na 84 232 logových záznamů. To znamená, že MLM modely byly schopny doplnit 53 190 nových logových záznamů, v nichž se dříve vyskytovaly maskované entity. Proces rozšíření založený na metodě MLM probíhal v rozmezí 6 až 17 minut strojového času, v závislosti na využitém modelu. Nejkratší čas 6 minut 34 sekund vykázal model ALBERT-base-v2 a naopak nejdelší čas model MobileBERT a to 17 minut 48 sekund. Generativní Llama3 model rozšířil původní dataset na 79 910 vzorků logových záznamů. Samotný proces rozšíření byl výrazně delší a trval celkem 143,63 hodin strojového času, což odpovídá zhruba 5 dnům a 23 hodinám. Navíc model selhal v případě 4 322 logů, ve kterých nebyl schopen nahradit maskované hodnoty. Tyto záznamy musely být následně ručně filtrovány.

Z výsledků v tabulce 4.5 lze usoudit, že je model schopen správně parsovat entity záznamů již pro původní datovou sadu. Největší navýšení F1 skóre dosáhl model s datasetem, který byl obohacen modelem MobilBERT-uncased. F1 se zvýšilo z hodnoty původní, nerozšířené, sady 99,877 % o 0,093 % na hodnotu 99,970 %. Modely MobilBERT-uncased a ELECTRA-small-generator byly učeny po celých 20 epoch, což pro oba modely bylo zhruba 156 minut GPU času. Tyto dva modely také vykazují v porovnání s ostatními modely nejlepší přesnost. Trénování na datové sadě obohacené modelem RoBERTa dosáhlo 18 epoch, 138 minut GPU času, a finální přesnost dosahuje hodnoty 99,956 %. Procesy ladění modelu T5 na datových sadách obohacených modely ALBERT-base-v2 a Llama3:70B nevedly k výraznému zvýšení přesnosti, avšak jejich přesnost zároveň nijak neklesla.

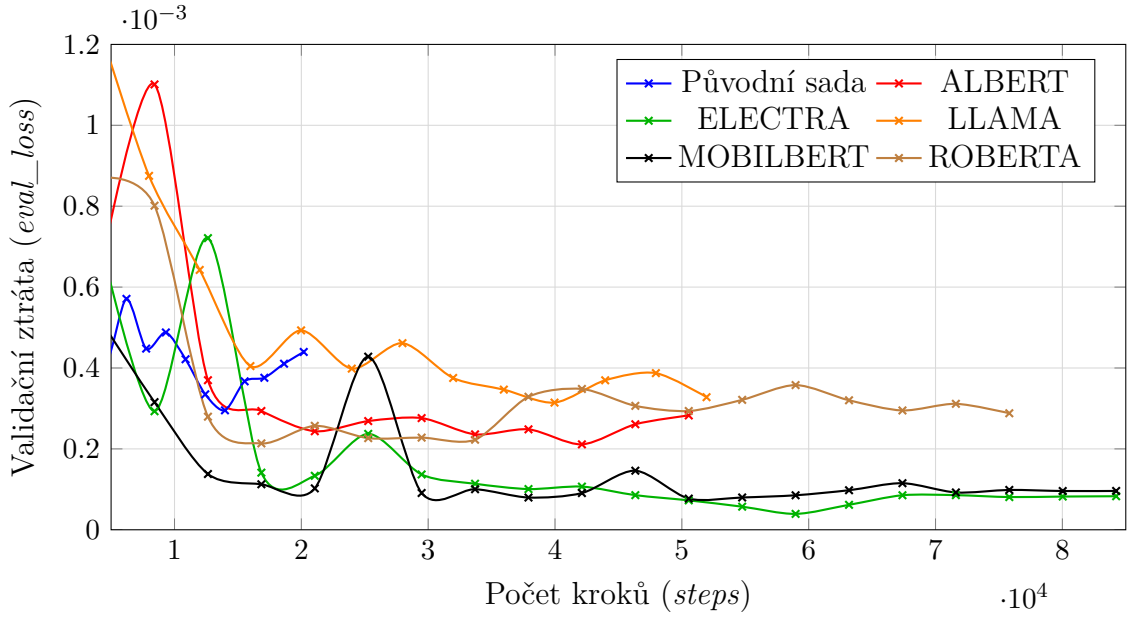
Vývoj hodnoty F1 skóre během procesu učení všech obohacených datových sad je znázorněn na obrázku 4.12. Z tohoto grafu je patrné, že trénování na původní datové sadě bylo značně nestabilní. Naopak trénování na obohacených sadách vykazovalo větší stabilitu s rostoucím počtem trénovacích kroků. Více než dvojnásobné navýšení počtu vzorků v datových sadách má tedy na proces učení jazykového modelu pozitivní vliv.

Podobné chování lze pozorovat také v grafu na obrázku 4.13, kde je zobrazen vývoj validační ztráty. U trénování modelu na původní sadě došlo po dosažení lokálního minima k opětovnému růstu ztráty, což vedlo k předčasnému ukončení učení.

Předčasné ukončení tréninkových procesů bylo zaznamenáno také u modelů, učených na obohacených sadách prostřednictvím modelů `ALBERT-base-v2`, `Llama3:70B` a `RoBERTa`. Důvodem je implementace mechanismu předčasného ukončení objektem `EarlyStoppingCallback`, který byl již zmíněn v kapitole 3.5.3. Tento objekt sledoval vývoj validační ztráty a pokud její hodnota nevykazovala zlepšení v průběhu tří po sobě jdoucích epoch, proces učení byl automaticky ukončen.



Obr. 4.12: Vývoj F1-skóre v závislosti na počtu kroků při NER ladění.



Obr. 4.13: Vývoj validační ztráty (`eval_loss`) v závislosti na počtu kroků při NER ladění.

Po dokončení procesu učení modelu `flan-T5-small` na rozšířených datových sadách byla provedena predikce pojmenovaných entit (úloha NER) na validační sadě. Všechny metaklíče (štítky entit), které se v datové sadě popsané v kapitole 3.2.3 vyskytují, byly představeny a shrnuty v tabulkách uvedených v příloze D. Po označení entit je použit formát značení BIO (zkratka pro *Beginning*, *Outside*, *Inside*), který umožní určení pozice tokenu v rámci víceslovných entit. Jedná se o proces vyhledávání tokenů, patřících ke stejné entitě a nazývá se *chunkování*. Předpona B- před značkou označuje začátek entity (*chunku*), předpona I- před značkou vyjadřuje, že se token nachází uvnitř *chunku*, ale nenachází se na jejím začátku, a označení O je využito pro token, který nepatří k žádné entitě [90].

Kromě vyhodnocení pomocí metrik uvedených v tabulce 4.5 byla pro posouzení kvality trénovaného modelu a objektivní porovnání výsledků sestavena matice záměn (*confusion matrix*). Tato matice vizualizuje vztah mezi předpovězenými a skutečnými třídami, čímž poskytuje ucelený přehled o výkonnosti klasifikačního modelu. Klasická 2x2 matice je interpretována prostřednictvím čtyř kategorií: **True Positive**, **True Negative**, **False Positive** a **False Negative**, o kterých již byly popsány v kapitole 2.5. Následně matice pomáhá při výpočtech hodnot přesností (*precision*, *accuracy*) a citlivosti (*recall*), které poskytují lepší představu o chování modelu, zejména v případě nevyvážených datech [92].

Na základě výsledků v tabulce 4.5 byla matice záměn sestavena pro doladěný model `flan-T5-small` nad ASA datovou sadou, augmentovanou modelem `Mobile-BERT-uncased`. Vzhledem k využití BIO formátu pro značkování entit (*chunkování*)

byla matice záměn rozdělena do dvou samostatných tabulek. Tabulky 4.14 a 4.15 tak obsahují informace o skutečných třídách (řádky matice, reprezentující různé kategorie entit v datech), předpovězených třídách (sloupce matice, udávající, do jaké třídy byly instance přiřazeny), celkovém počtu výskytu entit (*Total*, součet správně i nesprávně přiřazených predikcí každé třídy), počtu správných predikcí *True Positive*, falešných pozitivních predikcí (*False Positive*) a chybovosti (vyjádřena jako poměr falešně pozitivních predikcí k celkovému počtu predikcí dané třídy). Hodnoty na diagonále představují správně klasifikované predikce, zatímco hodnoty mimo diagonálu indikují nesprávné klasifikace.

Například třída `ip_src` byla modelem detekována nejčastěji. Tento štítek byl v sadě anotován 9 972krát jako počáteční entita (`B-ip_src`) a 122 946krát jako pokračující entita (`I-ip_src`). Model pak úspěšně označil 9 971 entit typu `B-ip_src` a 122 946 entit typu `I-ip_src`. Naopak nejvíce falešně pozitivních predikcí bylo modelem anotováno u třídy `ip_dest`, u které nesprávně klasifikoval 6 předpovědí jako `B-ip_dest` a 35 případů jako `I-ip_dest`. Vzhledem k vysokému celkovému zastoupení této třídy v datech je výsledná procentuální chybovost nulová. Naproti tomu třída `process_name` vykazuje chybovost 2 % již při jediné nesprávné predikci entity typu `B-process_name` z celkového počtu výskytů. Tato matice záměn tak poskytuje podrobný pohled na výkonnost modelu při identifikování jednotlivých entit v testovacích datech, a zároveň poukazuje na nevyváženost tříd v datech logových záznamů, což představuje při trénování jazykových modelů pro úlohy rozpoznávání pojmenovaných entit významnou výzvu.

Obr. 4.14: Matice záměn pro B- entity extrahované ze záznamů událostí.

	O	I-action	I-crypto	I-description	I-device_ip	I-domain_dest	I-domain_src	I-event_id	I-event_level	I-event_type	I-file_name	I-file_path	I-host_dest	I-host_name	I-host_src	I-interface_dest	I-interface_src	I-ip_dest	I-ip_src	I-mac_dest	I-mac_src	I-mask_src	I-network_protocol	I-os_version	I-port_dest	I-port_src	I-process_name	I-result	I-result_code	I-rule	I-service_name	I-session_id	I-severity	I-tags	I-timestamp	I-traffic_direction	I-url	I-user_dest	I-user_group	I-user_src	Total	TP	FP	%
O	8159787	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	8159791	8159791	4	0%
I-action	0	384	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	385	384	1	0%
I-crypto	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	27	0	0%
I-description	0	0	0	10401	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10401	10401	0	0%
I-device_ip	0	0	0	0	116	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	116	116	0	0%
I-domain_dest	0	0	0	0	0	2315	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2315	2315	0	0%
I-domain_src	0	0	0	0	0	0	1554	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1554	1554	0	0%
I-event_id	0	0	0	0	0	0	0	9169	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9169	9169	0	0%
I-event_level	0	0	0	0	0	0	0	0	27553	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27553	27553	0	0%
I-event_type	0	0	0	0	0	0	0	0	0	94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	94	94	0	0%
I-file_name	0	0	0	0	0	0	0	0	0	0	138	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	138	138	0	0%
I-file_path	0	0	0	0	0	0	0	0	0	0	0	352	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	352	352	0	0%
I-host_dest	0	0	0	0	0	0	0	0	0	0	0	0	299	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	299	299	0	0%
I-host_name	0	0	0	0	0	0	0	0	0	0	0	0	0	483	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	483	483	0	0%
I-host_src	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3648	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3648	3648	0	0%
I-interface_dest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5005	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5009	5005	4	0%
I-interface_src	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	7466	0	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7469	7466	3	0%
I-ip_dest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	61591	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	61626	61591	35	0%
I-ip_src	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	122946	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	122951	122946	5	0%
I-mac_dest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	28	0	0%
I-mac_src	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	231	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	231	231	0	0%
I-mask_src	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	526	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	526	526	0	0%
I-network_protocol	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1206	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1207	1206	1	0%
I-os_version	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	486	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	486	486	0	0%
I-port_dest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3060	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3060	3060	0	0%
I-port_src	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2522	0	0	0	0	0	0	0	0	0	0	0	0	0	2522	2522	0	0%
I-process_name	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	147	0	0	3	0	0	0	0	0	0	0	0	0	150	147	3	2%
I-result	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	9	9	0	0%
I-result_code	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6782	0	0	0	0	0	0	0	0	0	0	6782	6782	0	0%
I-rule	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	723	0	0	0	0	0	0	0	0	0	723	723	0	0%
I-service_name	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	827	0	0	0	0	0	0	0	0	832	827	5	1%
I-session_id	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7488	0	0	0	0	0	0	0	7488	7488	0	0%
I-severity	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	16	16	0	0%
I-tags	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	855	0	0	0	0	0	855	855	0	0%
I-timestamp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	82859	0	0	0	0	82859	82859	0	0%
I-traffic_direction	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	25	25	0	0%
I-url	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	529	0	0	529	529	0	0%
I-user_dest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17892	0	17892	17892	0	0%
I-user_group	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11641	11641	11641	0	0%
I-user_src	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	329	329	329	0	0%

Obr. 4.15: Matice záměn pro I- entity extrahované ze záznamů událostí.

Závěr

Hlavním cílem této práce bylo aplikovat pokročilé techniky zpracování přirozeného jazyka pro rozšíření vlastních datových sad obsahujících textová data logových záznamů, obohacených o simulovaná data nástrojem `Atomic Red Team`. Závěrem lze konstatovat, že stanovené úkoly v rámci zadání práce byly splněny. Po dosažení vytyčených cílů bude vývojový proces pokračovat dál s cílem dosáhnout co nejlepších výsledků v oblasti rozšiřování záznamů událostí a následné syntaktické analýzy. Tyto výsledky mají potenciál praktického uplatnění v oblasti počítačové bezpečnosti a zároveň mohou přinést reálný vědecký přínos.

V rámci první kapitoly byla představena teoretická část diplomové práce, zaměřená primárně na problematiku kybernetické bezpečnosti a význam monitorování v počítačovém prostředí. Byly popsány záznamy událostí, jejich struktura a formáty. Dále byly analyzovány bezpečnostní systémy pro monitorování, jako jsou SIEM, SOAR, XDR nebo EDR, včetně jejich přínosu pro zajištění bezpečnosti v kybernetickém prostředí. Závěr kapitoly se věnuje popisu procesu reakce na bezpečnostní incidenty a možnostem aplikace metod strojového učení v oblasti počítačové bezpečnosti.

Druhá kapitola nabízí náhled do problematiky umělé inteligence a strojového učení. V této kapitole byly představeny vlastnosti neuronových sítí a jejich různé typy. Byly popsány techniky strojového učení, jako je učení s učitelem, učení bez učitele využívané při trénování jazykových modelů, zpětnovazební učení a samořízené učení. Dále byla kromě hlubokého a přenosového učení rozebrána metoda zpracování přirozeného jazyka (NLP), která umožňuje strojům porozumět textu a generovat jazyk blízký lidskému. V souvislosti s touto metodou byly popsány hluboké neuronové sítě typu `Transformer`, jejich architektura a dostupné typy modelů. Kapitola se dále zaměřuje na popis hodnocení výkonnosti procesu strojového učení a výběr vhodného nástroje pro jeho sledování a řízení. V rámci tohoto popisu byly představeny možnosti nástroje `MLflow`, který umožňuje sledování a zaznamenávání průběhu učení a ukládání výsledných modelů. Kapitola byla uzavřena představením možností aplikace metod strojového učení v kontextu bezpečnostních záznamů událostí a jejich rozšiřování.

Kapitola 3 se věnuje problematice rozšiřování textových dat a logových záznamů. Úvodní část kapitoly obsahuje rešerši dostupných technik a současných přístupů k rozšiřování textových dat, a to s pomocí *embedding* algoritmů a pokročilých technik hlubokých neuronových sítí. V této části byl rovněž diskutován přínos modelů, založených na architektuře `Transformer`, které díky svým pokročilým schopnostem analýzy textu představují vhodné kandidáty k pokročilé augmentaci záznamů. Následuje technický popis připravené datové sady surových logů využitých pro ladění modelů,

návrh experimentálního pracoviště pro simulaci kybernetických útoků a sběr bezpečnostních záznamů, stejně jako následný popis zpracování trénovací sady a analýzy testovací sady. Dále kapitola představuje potřebné kroky předzpracování vstupních textových dat a popis metod rozšíření textu, konkrétně *Masked Language Modeling* a *Next Word Prediction*, které využívají `Transformer` architekturu. V této části kapitoly jsou technicky popsány zvolené modely určené k doladění na vytvořené trénovací datové sadě. Závěr této kapitoly se věnuje popisu procesu trénování zvolených jazykových modelů, konkrétně modelů založených na metodě MLM `ALBERT-base`, `ELECTRA-small-generator`, `MobileBERT-uncased`, `RoBERTa-base` a modelů pracujících na základě predikcí následujícího tokenu `DeepSeek-R1-Distill-Llama-8B`, `GPT-2`, `Llama-3.2-3B` a `SmolLM2-1.7B`. Sledování a vyhodnocení tréninkových procesů probíhalo v reálném čase pomocí nástroje `MLflow`. Z modelů typu MLM dosáhl nejlepších výsledků model `RoBERTa-base`, který zároveň vykázal nejnižší hodnotu perplexity (1,241). Všechny použité MLM modely vykazovaly pozitivní průběh i výsledky, a proto byly dále vybrány k testování. U modelů NWP se nepodařilo úspěšně doladit model `DeepSeek-R1-Distill-Llama-8B` ani v menší variantě s 3 miliardami parametrů, a tak byly dále testovány zbylé tři modely, přičemž nejpříznivějších výsledků dosáhl model `GPT-2`.

Čtvrtá kapitola se věnuje popisu návrhu a implementaci funkce pokročilého augmentačního nástroje. Popisuje funkci vytvořené *pipeline* pro obě zvolené metody rozšiřování textu. V úvodní části byly jednotlivé modely testovány lokálně na vybraném vzorku. Cílem bylo dosadit do jednoduchého záznamu s maskovanou entitou synteticky generovanou hodnotu. Během těchto testů byla nasazena *pipeline* pro MLM modely, které dokázaly na základě kontextu adekvátně nahradit masky. Na druhou stranu laděné modely NWP v tomto úkolu selhávaly a jejich ladící procesy byly označeny za neefektivní a od jejich dalšího využití bylo ustoupeno. Toto zjištění vedlo k implementaci API rozhraní na platformě `Ollama`, jež umožňuje využití několikanásobně větších generačních modelů, jako jsou využité modely `DeepSeek-R1:32B` a `Llama3:70B`. Pro tuto variantu byla rozšířena *pipeline*, která je nyní schopna rozšiřovat vstupní datové sady metodou doplňování masek jak pomocí doladěných modelů dostupných prostřednictvím `MLlow`, tak i velkými jazykovými modely přes `Ollama` API.

Těchto šest modelů bylo následně využito pro dva druhy testování. V první fázi byly aplikovány na deseti vybraných záznamech simulovaných nástrojem `Atomic Red Team`. Každý z těchto záznamů byl nejprve podrobně analyzován a poté náhodně maskován v některých svých entitách. Maskované záznamy byly následně vloženy na vstup jednotlivých modelů za účelem jejich augmentace. Výsledky byly zaznamenány, systematicky uspořádány do tabulek a vyhodnoceny pomocí obecně známých metrik. Mezi použité metriky patřily `BERTscore` (*F1-score, precision, recall*),

`Levenshtein distance`, `Cosine similarity` a `BLEU` skóre. Na základě porovnání výkonu jednotlivých modelů podle těchto metrik byl z pohledu přesnosti generování syntetických dat jednoznačně nejúspěšnější model `DeepSeek-R1:32B`. V oblasti generace složitějších entit výrazně zaostávaly MLM modely, a to především kvůli své architektuře, která umožňuje generovat pouze jeden token na každou masku. Při potřebě rozšiřování záznamů o složitější entity, jako jsou časová razítka, data, hashe nebo souborové cesty, je proto nutné umístit více maskovacích značek, nebo alternativně využít velkých jazykových modelů se schopností rozmanité generace celých posloupností. Problémem NWP velkých modelů zůstává jejich vysoká výpočetní náročnost a potřeba kontroly rozmanitosti generovaných výstupů.

Druhá část testování modelů byla zaměřena na rozšíření vytvořené datové sady pomocí jazykových modelů. V tomto případě byly využity čtyři MLM modely a velký jazykový model `Llama3:70B`. Od modelu `DeepSeek-R1:32B` bylo v tomto případě odstoupeno kvůli jeho časové náročnosti při augmentaci větší datové sady. Výhodou MLM modelů je schopnost nahradit veškeré masky umístěné v jednotlivých záznamech, čímž se podařilo rozšířit původní sadu o 31 042 záznamů na 84 232 záznamů. Tyto modely navíc umožňují rozšíření ve velmi příznivém časovém rozmezí pohybujícím se mezi 6 až 17 minutami strojového času. NWP generativní model `Llama3:70B` rozšířil původní sadu na 79 910 vzorků s nutností filtrování vzorků, na kterých selhal. Na těchto sadách byl následně laděn model `flan-T5-small`. Nejlepších výsledků dosáhl model trénovaný na datové sadě rozšířené pomocí augmentačního modelu `MobileBERT`, s přesností 99,970 %. Tento model byl následně testován na úloze rozpoznání pojmenovaných entit a vyhodnocen maticí záměn. Výsledná matice poskytla ucelený přehled o schopnosti modelu předpovídat anotované entity v použité datové sadě a zároveň poukázala na nevyváženost některých tříd – například entita `I-ip_src` byla v sadě zastoupena 122 946krát, zatímco entita `I-result` pouze devětkrát.

Jako možné rozšíření této diplomové práce se nabízí vylepšení trénovacích datových sad, širší využití generativních schopností dostupných velkých jazykových modelů a jejich aplikace nejen na rozšíření maskovaných metaklíčů, ale také na generování komplexnějších výrazů. Ve spolupráci s vedoucím práce je dále plánováno sepsání vědeckého článku v anglickém jazyce, který bude prezentovat výsledky provedeného výzkumu v odborném časopise zaměřeném na pokročilé metody rozšiřování logových záznamů.

Literatura

[1] GMCDOUGA, 2024. *A Closer Look at Q3 2024: 75% Surge in Cyber Attacks Worldwide.* Check Point Blog. Online. Dostupné z: https://blog.checkpoint.com/research/a-closer-look-at-q3-2024-75-surge-in-cyber-attacks-worldwide/. [citováno 2025-05-25].

[2] KENT, K a M P SOUPPAYA, 2006. *Guide to computer security log management.* Online. vyd. NIST SP 800-92. Gaithersburg, MD: National Institute of Standards and Technology. Dostupné z: http://doi.org/10.6028/NIST.SP.800-92. [citováno 2024-11-06].

[3] SAFONOV, Yehor a Michal ZERNOVIC, 2023. *Enhancing Security Monitoring with AI-Enabled Log Collection and NLP Modules on a Unified Open Source Platform.* Online. Brno, Czech republic: Brno University of Technology, Faculty of Electrical Engineering and Communication. ISBN 978-80-214-6154-3. Dostupné z: http://doi.org/10.13164/eeict.2023.217. [citováno 2024-11-04].

[4] Omoseebi, ADETOYESE, Jonson, DORCAS, Robert, DESIRE, 2022. *Challenges of integrating various data sources in SIEM.* Online. Dostupné z: https://www.researchgate.net/publication/388069911_Challenges_of_integrating_various_data_sources_in_SIEM. [citováno 2025-04-13].

[5] SHORTEN, Connor, Taghi M. KHOSHGOFTAAR a Borko FURHT, 2021. *Text Data Augmentation for Deep Learning.* Journal of Big Data. Online. ISSN 2196-1115. Dostupné z: http://doi.org/10.1186/s40537-021-00492-0. [citováno 2025-05-25].

[6] RING, Markus, Sarah WUNDERLICH, Deniz SCHEURING, Dieter LANDES a Andreas HOTHO, 2019. *A survey of network-based intrusion detection data sets.* Computers & Security. Online. 86, 147–167. ISSN 01674048. Dostupné z: http://doi.org/10.1016/j.cose.2019.06.005. [citováno 2025-05-25].

[7] KASPERSKY. 2017. *What is Cybersecurity? Types, Threats and Cyber Safety Tips.* Online. Dostupné z: https://www.kaspersky.com/resource-center/definitions/what-is-cyber-security. [citováno 2024-11-07].

[8] RAZA, Muhammad. *What's Security Monitoring in Cybersecurity?* Online. Splunk. Dostupné z: https://www.splunk.com/en_us/blog/learn/security-monitoring.html. [citováno 2024-11-07].

[9] SAFONOV, Yehor. *Pokročilý bezpečnostní monitoring počítačových infrastruktur a filozofie SIEM / SOAR řešení.* Brno: Vysoké učení technické, Fakulta informačních technologií, 2024. [citováno 2024-11-20].

[10] SHARIF, Arfan, 2022. *6 Common Log File Formats.* CrowdStrike. Online. Dostupné z: `https://www.crowdstrike.com/en-us/cybersecurity-101/next-gen-siem/log-file-formats/`. [citováno 2025-04-19].

[11] The Apache Software Foundation, 2025. *Log Files - Apache HTTP Server Version 2.4.* Online. Dostupné z: `https://httpd.apache.org/docs/current/logs.html`. [citováno 2025-04-19].

[12] SCAPICCHIO, Mark, DOWNIE, Amanda, FINIO, Matthew, 2024. *What Is a Security Operations Center (SOC)?* IBM. Online. 2021. Dostupné z: `https://www.ibm.com/topics/security-operations-center`. [citováno 2024-11-07].

[13] MAŤAŠ, Matúš, 2024. *Automatizovaná síť pro klamání útočníků prostřednictvím iluzivních aktiv v kyberprostoru.* Dostupné z: `https://www.vut.cz/studenti/zav-prace/detail/159176`. [citováno 2024-11-07].

[14] ARICOMA. *Komplexní systémy řízení bezpečnosti (SIEM).* Online. Dostupné z: `https://www.aricoma.com/cs/co-delame/kyberneticka-bezpecnost/sprava-bezpecnostnich-informaci/komplexni-systemy-rizeni-bezpecnosti-siem`. [citováno 2024-11-04].

[15] HRISTOV, Marian, Maria NENOVA, Georgi ILIEV a Dimiter AVRESKY, 2021. *Integration of Splunk Enterprise SIEM for DDoS Attack Detection in IoT.* 2021. IEEE 20th International Symposium on Network Computing and Applications (NCA): 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA). Online. s. 1–5. ISSN 2643-7929. Dostupné z: `http://doi.org/10.1109/NCA53618.2021.9685977`. [citováno 2024-11-06].

[16] IBM, 2023. *What is SIEM?* Online. Dostupné z: `https://www.ibm.com/think/topics/siem`. [citováno 2025-05-14].

[17] IBM, 2023. *What is SOAR (security orchestration, automation and response)?* Online. Dostupné z: `https://www.ibm.com/topics/security-orchestration-automation-response`. [citováno 2024-11-07].

[18] IBM, 2022. *What is Endpoint Detection and Response (EDR)?* Online. Dostupné z: `https://www.ibm.com/topics/edr`. [citováno 2024-11-07].

[19] IBM, 2023. *What Is Extended Detection and Response (XDR)?* Online. Dostupné z: `https://www.ibm.com/topics/xdr`. [citováno 2024-11-07].

[20] HOLDSWORTH, Jim, KOSINSKI, Matthew, 2024. *What is Incident Response?* IBM. Online. Dostupné z: https://www.ibm.com/topics/incident-response. [citováno 2024-11-20].

[21] *What is a Neural Network?* GeeksforGeeks. Online. Dostupné z: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/. [citováno 2024-12-07].

[22] CHOLLET, Francois a Rudolf PECINOVSKÝ. *Deep learning v jazyku Python: Knihovny Keras, TensorFlow.* Grada, 2019. ISBN 9788024731001. [citováno 2024-12-02].

[23] CHOLLET, François, 2021. *Deep Learning with Python, Second Edition.* New York: Manning Publications Co. LLC. ISBN 978-1-61729-686-4. [citováno 2024-11-17].

[24] *Introduction to Recurrent Neural Networks.* GeeksforGeeks. Online. Dostupné z: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/. [citováno 2024-12-08].

[25] *Introduction to Convolution Neural Network.* GeeksforGeeks. Online. Dostupné z: https://www.geeksforgeeks.org/introduction-convolution-neural-network/. [citováno 2024-12-08].

[26] SAFONOV, Yehor. *Filtrování spamových zpráv pomocí metod umělé inteligence.* Online. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2020. Dostupné z: http://hdl.handle.net/11012/189201. [citováno 2024-12-02].

[27] JANIESCH, Christian, Patrick ZSCHECH a Kai HEINRICH, 2021. *Machine learning and deep learning.* Electronic Market. Online. vol. 31, no. 3, s. 685–695. ISSN 1019-6781, 1422-8890. Dostupné z: https://doi.org/10.1007/s12525-021-00475-2. [citováno 2024-12-02].

[28] BURGET, Radim, [b.r.]. *Advanced Data Structures and Algorithms.* Brno: Vysoké učení technické v Brně, 2023. [citováno 2024-12-02].

[29] BUDUMA, Nikhil a Nicholas LOCASCIO, 2017. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms.* ISBN 9781492082187. [citováno 2024-12-02].

[30] PH.D, Cameron R. Wolfe, 2023. *T5: Text-to-Text Transformers (Part One).* Deep (Learning) Focus [online]. Substack newsletter. Dostupné z: https:

//cameronrwolfe.substack.com/p/t5-text-to-text-transformers-part. [citováno 2024-11-18].

[31] STRYKER, Cole, HOLDSWORTH, Jim, 2024. *What Is NLP (Natural Language Processing)?* IBM. Online. Dostupné z: https://www.ibm.com/topics/natural-language-processing. [citováno 2024-12-02].

[32] VARSHITHA, Konda Sai, Chinni Guna KUMARI, Muppala HASVITHA, Shaik FIZA, Amarendra K, 2023. *Natural Language Processing using Convolutional Neural Network.* In: 2023 7th International Conference on Computing Methodologies and Communication (ICCMC): 2023 7th International Conference on Computing Methodologies and Communication (ICCMC). Online. s. 362–367 Dostupné z: https://doi.org/10.1109/ICCMC56507.2023.10083608. [citováno 2024-12-02].

[33] WU, Jiawei a Jingxuan XIAO, 2024. *Application of Natural Language Processing in Network Security Log Analysis.* Online. ISSN 3007-4126. Dostupné z: https://doi.org/10.5281/ZENODO.13366745. [citováno 2025-05-25].

[34] HOWARD, Jeremy a Sebastian RUDER, 2018. *Universal Language Model Fine-tuning for Text Classification.* Online. Dostupné z: https://doi.org/10.48550/arXiv.1801.06146. [citováno 2024-12-04].

[35] VASWANI, Ashish, Noam SHAZEER, Niki PARMAR, Jakob USZKOREIT, Llion JONES, Aidan N. GOMEZ, Lukasz KAISER a Illia POLOSUKHIN, 2023. *Attention Is All You Need.* Online. 2. srpen 2023. Dostupné z: https://doi.org/10.48550/arXiv.1706.03762. [citováno 2024-12-02].

[36] TUNSTALL, Lewis, Leandro von WERRA, Thomas WOLF a Aurélien GÉRON, 2022. *Natural language processing with Transformers: building language applications with Hugging Face. First edition.* Beijing Boston Farnham Sebastopol Tokyo: O'Reilly. ISBN 978-1-09-810324-8. [citováno 2024-11-11]

[37] M, Vijay, 2023. *What is the difference between Training Loss Validation Loss and Evaluation Loss.* Medium. Online. Dostupné z : https://medium.com/@penpencil.blr/what-is-the-difference-between-training-loss-validation-loss-and-evaluation-loss-c169ddeccd59. [citováno 2025-02-09].

[38] *Training and Validation Loss in Deep Learning.* GeeksforGeeks. Online. Dostupné z: https://www.geeksforgeeks.org/training-and-validation-loss-in-deep-learning/. [citováno 2025-02-09].

[39] *What Is Cross-Entropy Loss Function?* GeeksforGeeks. Online. Dostupné z: https://www.geeksforgeeks.org/what-is-cross-entropy-loss-function/. [citováno 2025-02-09].

[40] CHIUSANO, Fabio, 2022. *Two minutes NLP — Perplexity explained with simple probabilities.* Generative AI. Online. Dostupné

z : https://medium.com/nlplanet/two-minutes-nlp-perplexity-explained-with-simple-probabilities-6cdc46884584. [citováno 2025-02-09].

[41] HUGGING FACE. *Perplexity of fixed-length models.* Online. Dostupné z: https://huggingface.co/docs/transformers/perplexity. [citováno 2025-02-09].

[42] GeeksforGeeks *F1 Score in Machine Learning.* Online. Dostupné z: https://www.geeksforgeeks.org/f1-score-in-machine-learning/. [citováno 2025-03-18].

[43] MLflow Project, a Series of LF Projects, LLC, 2025. *MLflow.* Online. Dostupné z: https://your-docusaurus-site.example.com/. [Citováno 2025-02-03].

[44] ClearML Inc., 2025. *ClearML.* Online. Dostupné z: https://clear.ml/. [Citováno 2025-02-03].

[45] ZAITON, Hoda a Sameh AL-ANSARY, 2024. *Natural Language Processing Approaches to Text Data Augmentation: A Computational Linguistic Analysis.* International Journal of Arabic-English Studies. Online. ISSN 1680-0982.. Dostupné z: http://doi.org/10.33806/ijaes.v25i1.682. [citováno 2024-10-29].

[46] *Text augmentation techniques in NLP.* GeeksforGeeks. Online. Dostupné z: https://www.geeksforgeeks.org/text-augmentation-techniques-in-nlp/. [citováno 2024-11-30].

[47] YADAV, Amit, 2025. *Word2Vec vs GloVe: Which Word Embedding Model is Right for You?* Biased-Algorithms. Online. Dostupné z: https://medium.com/biased-algorithms/word2vec-vs-glove-which-word-embedding-model-is-right-for-you-4dfc161c3f0c. [citováno 2025-05-11].

[48] PENNINGTON, Jeffrey, Richard SOCHER a Christopher MANNING, 2014. *GloVe: Global Vectors for Word Representation.* In: Alessandro MOSCHITTI, Bo PANG a Walter DAELEMANS, ed. EMNLP 2014: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) Online. Doha, Qatar: Association for Computational Linguistics, s. 1532–1543 Dostupné z: http://doi.org/10.3115/v1/D14-1162. [citováno 2025-05-11].

[49] SHPERBER, Gidi, 2019. *A gentle introduction to Doc2Vec.* Wisio. Online. Dostupné z: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e. [citováno 2025-05-11].

[50] SPLUNK THREAT RESEARCH TEAM, 2022. *Attack Range Docs — Attack Range 3.0.0 documentation.* Online. Dostupné z: https://attack-range.readthedocs.io/en/latest/. [citováno 2024-11-20].

[51] RED CANARY, 2024. *Atomic Red Team.* Online. Dostupné z: https://github.com/redcanaryco/atomic-red-team. [citováno 2024-11-20].

[52] MITRE. *ATT&CK®*. Online. Dostupné z: https://attack.mitre.org/. [citováno 2024-11-20].

[53] LANGE, Kayly, 2024. *Continuous Testing: The Ultimate Guide.* Splunk. Online. Dostupné z: https://www.splunk.com/en_us/blog/learn/continuous-testing.html. [citováno 2024-12-10].

[54] Cxtec, 2025. *What Is CISCO Adaptive Security Appliance (ASA)?* Online. Dostupné z: https://www.cxtec.com/blog/what-is-cisco-asa-security-appliance/. [citováno 2025-05-17].

[55] PH.D, CAMERON R. Wolfe, 2023. *Language Model Training and Inference: From Concept to Code. Deep (Learning) Focus.* Online.. Substack newsletter. Dostupné z: https://cameronrwolfe.substack.com/p/language-model-training-and-inference. [citováno 2024-11-18].

[56] *Summary of the tokenizers.* Online. Dostupné z: https://huggingface.co/docs/transformers/tokenizer_summary. [citováno 2024-12-06].

[57] DEVLIN, Jacob, Ming-Wei CHANG, Kenton LEE a Kristina TOUTANOVA, 2019. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* Online. Dostupné z: http://doi.org/10.48550/arXiv.1810.04805. [citováno 2024-12-05].

[58] INGHAM, Francisco, 2018. *Dissecting BERT Part 2: BERT Specifics. Dissecting BERT.* Online. Dostupné z: https://medium.com/dissecting-bert/dissecting-bert-part2-335ff2ed9c73. [citováno 2024-12-05].

[59] LIU, Yinhan, Myle OTT, Naman GOYAL, Jingfei DU, Mandar JOSHI, Danqi CHEN, Omer LEVY, Mike LEWIS, Luke ZETTLEMOYER a Veselin STOYANOV, 2019. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* Online]. Dostupné z: http://arxiv.org/abs/1907.11692. [citováno 2024-12-04].

[60] AI, Novita, 2024. *Introducing RoBERTa Base Model: A Comprehensive Overview.* Medium. Online. Dostupné z: https://medium.com/@marketing_novita.ai/

introducing-roberta-base-model-a-comprehensive-overview-330338afa082. [citováno 2024-12-06].

[61] SUN, Zhiqing, Hongkun YU, Xiaodan SONG, Renjie LIU, Yiming YANG a Denny ZHOU, 2020. *MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices* arXiv. Online. Dostupné z: http://doi:10.48550/arXiv.2004.02984. [citováno 2025-05-05].

[62] TSANG, Sik-Ho, 2022. *Brief Review — ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.* Medium. Online. Dostupné z:https://sh-tsang.medium.com/brief-review-electra-pre-training-text-encoders-as-discriminatorsrather-than-generators-9568050d3a864. [citováno 2025-05-05].

[63] Hugging Face, 2024. *albert/albert-base-v2*. Online. Dostupné z: https://huggingface.co/albert/albert-base-v2. [citováno 2025-05-15].

[64] Hugging Face, [b.r.]. *google/mobilebert-uncased*. Online. Dostupné z: https://huggingface.co/google/mobilebert-uncased. [citováno 2025-05-15].

[65] Hugging Face, 2025. *google/electra-small-generator*. Online. Dostupné z: https://huggingface.co/google/electra-small-generator. [citováno 2025-05-15].

[66] *RoBERTa*. Hugging Face. Online. Dostupné z: https://huggingface.co/docs/transformers/model_doc/roberta. [citováno 2024-12-09].

[67] PH.D, Cameron R. Wolfe, 2022. *Language Models: GPT and GPT-2.* Deep (Learning) Focus. Online. Substack newsletter. Dostupné z: https://cameronrwolfe.substack.com/p/language-models-gpt-and-gpt-2. [citováno 2024-12-06].

[68] DAR, Pranav, 2019. *8 Excellent Pretrained Models to get you Started with Natural Language Processing (NLP).* Analytics Vidhya. Online. Dostupné z: https://www.analyticsvidhya.com/blog/2019/03/pretrained-models-get-started-nlp/. [citováno 2024-12-04].

[69] TRAN, Tuan, 2024. *Fine-Tuning Large Language Model with Hugging Face & PyTorch.* Medium. Online. Dostupné z: https://tuanatran.medium.com/fine-tuning-large-language-model-with-hugging-face-pytorch-adce80dce2ad. [citováno 2024-12-07].

[70] OLTEANU, Alex, 2024. *Llama 3.2 Guide: How It Works, Use Cases & More.* Online. Dostupné z: https://www.datacamp.com/blog/llama-3-2. [citováno 2024-12-04].

[71] LUNA, Javier, Canales. *What is Multimodal AI?* Online. Dostupné z: .https://www.datacamp.com/blog/what-is-multimodal-ai citováno 2024-12-10].

[72] RAZZAQ, Asif, 2024. *SmolLM2 Released: The New Series (0.1B, 0.3B, and 1.7B) of Small Language Models for On-Device Applications and Outperforms Meta Llama 3.2 1B.* MarkTechPost. Online. Dostupné z : https://www.marktechpost.com/2024/10/31/smollm2-released-the-new-series-0-1b-0-3b-and-1-7b-of-small-language-models-for-on-device-applications-and-outperforms-meta-llama-3-2-1b/. [citováno 2024-11-11].

[73] Hugging Face, 2024. *HuggingFaceTB/SmolLM2-1.7B.* Online. Dostupné z: https://huggingface.co/HuggingFaceTB/SmolLM2-1.7B. [citováno 2024-11-11].

[74] AUBRY, François, 2025. *DeepSeek R1 vs V3: A Guide With Examples.* Online. Dostupné z: https://www.datacamp.com/blog/deepseek-r1-vs-v3. [citováno 2025-03-27].

[75] Hugging Face, 2025. *deepseek-ai/DeepSeek-R1.* Online. Dostupné z: https://huggingface.co/deepseek-ai/DeepSeek-R1. [citováno 2025-03-27].

[76] Hugging Face, 2024. *OpenAI GPT2.* Online. Dostupné z: https://huggingface.co/docs/transformers/model_doc/gpt2#openai-gpt2. [citováno 2024-12-11].

[77] Hugging Face, 2024. *meta-llama/Llama-3.2-3B.* Online. Dostupné z: https://huggingface.co/meta-llama/Llama-3.2-3B. [citováno 2024-11-11].

[78] ALLAL, Loubna Ben, Anton LOZHKOV, Elie BAKOUCH, Gabriel Martín BLÁZQUEZ, Guilherme PENEDO, Lewis TUNSTALL, Andrés MARAFIOTI, Hynek KYDLÍČEK, Agustín Piqueres LAJARÍN, Vaibhav SRIVASTAV, Joshua LOCHNER, Caleb FAHLGREN, Xuan-Son NGUYEN, Clémentine FOURRIER, Ben BURTENSHAW, Hugo LARCHER, Haojun ZHAO, Cyril ZAKKA, Mathieu MORLON, Colin RAFFEL, Leandro von WERRA a Thomas WOLF, 2025. *SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model.* Online. arXiv. Dostupné z: https://doi.org/10.48550/arXiv.2502.02737. [citováno 2025-05-15].

[79] Hugging Face, 2025. *deepseek-ai/DeepSeek-R1-Distill-Llama-8B.* Online. Dostupné z: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B. [citováno 2025-05-15].

[80] RAFFEL, Colin, Noam SHAZEER, Adam ROBERTS, Katherine LEE, Sharan NARANG, Michael MATENA, Yanqi ZHOU, Wei LI a Peter J. LIU, 2023.

Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Online. Dostupné z: https://doi.org/10.48550/arXiv.1910.10683. [citováno 2024-12-04].

[81] PH.D, Cameron R. Wolfe, 2022. *Language Understanding with BERT.* Deep (Learning) Focus. Online. Substack newsletter. Dostupné z: https://cameronrwolfe.substack.com/p/language-understanding-with-bert. [citováno 2024-12-09].

[82] HU, Edward J., Yelong SHEN, Phillip WALLIS, Zeyuan ALLEN-ZHU, Yuanzhi LI, Shean WANG, Lu WANG a Weizhu CHEN, 2021. *LoRA: Low-Rank Adaptation of Large Language Models.* Online. arXiv. Dostupné z: http://doi:10.48550/arXiv.2106.09685. [citováno 2025-05-15].

[83] RATH, Sovit Ranjan, 2024. *Text Generation using GPT2.* DebuggerCafe. Online. Dostupné z: https://debuggercafe.com/fine-tuning-gpt2-for-text-generation/. [citováno 2024-12-11].

[84] Hugging Face, 2024. *Pipelines.* Online. Dostupné z: https://huggingface.co/docs/transformers/main_classes/pipelines. [citováno 2024-12-11].

[85] GOODWIN, Michael, 2024. *What Is an API (Application Programming Interface)?* IBM. Online. Dostupné z: https://www.ibm.com/think/topics/api. [citováno 2025-05-17].

[86] ZHANG, Tianyi, Varsha KISHORE, Felix WU, Kilian Q. WEINBERGER a Yoav ARTZI, 2020. *BERTScore: Evaluating Text Generation with BERT.* Online. arXiv. Dostupné z: http://doi.org/10.48550/arXiv.1904.09675. [citováno 2025-05-22].

[87] GeeksforGeeks, 2024. *Introduction to Levenshtein distance.* Online. Dostupné z: https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/. [citováno 2025-05-20].

[88] GeeksforGeeks, 2025. *Cosine Similarity.* Online. Dostupné z: https://www.geeksforgeeks.org/cosine-similarity/. [citováno 2025-05-20].

[89] PAPINENI, Kishore, ROUKOS, Salim, WARD, Todd, ZHU, Wei-jing, 2002. *BLEU - a Hugging Face Space by evaluate-metric.* Online. Dostupné z: https://huggingface.co/spaces/evaluate-metric/bleu. [citováno 2025-05-20].

[90] Hugging Face. *Token classification - Hugging Face LLM Course.* Online. Dostupné z: https://huggingface.co/learn/llm-course/chapter7/2. [citováno 2025-05-21].

[91] HUGGING FACE, 2024. *google/flan-t5-small.* Online. Dostupné z: `https://huggingface.co/google/flan-t5-small`. [citováno 2025-05-17].

[92] GeeksforGeeks, 2025. *Understanding the Confusion Matrix in Machine Learning.* Online. Dostupné z: `https://www.geeksforgeeks.org/confusion-matrix-machine-learning/`. [citováno 2025-05-19].

Seznam symbolů a zkratek

NLP	Natural Language Processing – Zpracování přirozeného jazyka
MLM	Masked Language Modelling – Modelování maskovaného jazyka
NSP	Next Sentence Prediction – Předpověď následující sekvence
IP	Internet Protocol
SIEM	Security Information and Entity Management
SIM	Security Information Management
SEM	Security Event Managment
LES	Log Event Source
PES	Packet Event Source
JSON	JavaScript Object Notation
SOC	Security Operations Center – Bezpečnostní Operační Centrum
SOAR	Security Information and Event Management
EDR	Endpoint Detection and Response
IoT	Internet of Things
XDR	Extended Detection and Response
SaaS	Software as a Service – Software jako služba
MDR	Managed Detection and Response – Řízená detekce a reakce
DDoS	Distributed Denial of Service – Distribuované odepření služby
CSIRT	Computer Security Incident Response Team – Tým pro řešení počítačových bezpečnostních incidentů
RNI	Random Noise Injection – Náhodné zavádění šumu
RNN	Recurrent Neural Networks – Rekurentní Neuronové Sítě
CNN	Convolutional Neural Networks – Konvoluční Neuronové Sítě
FNN	Feedforward Neural Networks – Dopředné Neuronové Sítě

LSTM	Long Short-Term Memory
LM	Language Modeling – Modelování jazyka
LLM	Large Language Model – Velký jazykový model
Word2Vec	Word-to-Vector
GloVe	Global Vectors for Word Representation
BERT	Bidirectional Encoder Representations)
RoBERTa	Robustly optimized BERT approach
GPT	Generative Pre-Training
NLU	Natural Language Undestanding – Porozmění přirozenému jazyku
NER	Named Entity Recognition – Rozpoznávání pojmenovaných entit
EDA	Easy Data Augmentation – Snadné rozšiřování dat
BPE	Byte Pair Encoding
BBPE	Byte-level Byte Pair Encoding
NWP	Next Word Prediction – Předpověď dalšího slova
EOS	End-of-Sequence – konec sekvence
BOS	Beginning -of-Sequence – začátek sekvence
ASA	Adaptive Security Appliances
VPN	Virtual Private Network – Virtuální Privátní Síť
CBOW	Continuous Bag of Words – spojitý soubor slov
Doc2Vec	Document-to-Vector
PV-DM	Distributed Memory version of Paragraph Vector – Distribuovaná paměťová verze vektoru odstavců
PV-DBOW	Distributed Bag of Words version of Paragraph Vector – Distribuovaná verze odstavcového vektoru s množinou slov
LoRA	Low-Rank Adaptaion
API	Application Programming Interface

NER Named Entity Recognition – Rozpoznání pojmenovaných entit

Seznam příloh

A Struktura adresářů

Adresářová struktura projektu je následující:

- `/` ..... kořenový adresář projektu
  - `DeepAugmentator/` ..... zdrojový kód augmentačního nástroje
    - `MLflow_model_load/` ..... nástroje pro načítání trénovaných modelů
      - `MLM_models.ipynb` .. Jupyter notebook pro načítání modelů z MLflow
    - `pipeline/` ..... implementace pipeline skriptů
      - `AI_augmentator.py` ..... základní augmentace pomocí MLM modelů
      - `AI_Ollama_augmentator.py` .... augmentace pomocí Ollama modelu
      - `pipeline.py` ..... řízení běhu augmentační pipeline
    - `train_scripts/` ..... trénovací skripty pro různé modely
      - `gpt_train.py` ..... trénink GPT-2 modelu
      - `llama_deepseek_train.py` ..... trénink Llama (DeepSeek) modelu
      - `MLM_training.py` ..... trénink MLM modelů
      - `run.sh` ..... spouštěcí skript tréninkových úloh
      - `smollm2_train.py` ..... trénink SmolLM modelu
  - `Foltyn_Diplomova_Prace.pdf` ..... finální PDF verze diplomové práce

B Testovací záznamy událostí ze Splunk datové sady

```
09/17/2020 06:59:06 PM
LogName=Security
SourceName=Microsoft Windows security auditing.
EventCode=4689
EventType=0
Type=Information
ComputerName=win-dc-1603297.attackrange.local
TaskCategory=Process Termination
OpCode=Info
RecordNumber=7465312
Keywords=Audit Success
Message=A process has exited.

Subject:
  Security ID:    ATTACKRANGE\administrator
  Account Name:   administrator
  Account Domain:   ATTACKRANGE
  Logon ID:   0x4FDA1

Process Information:
  Process ID: 0x1498
  Process Name: C:\Windows\System32\WindowsPowerShell\v1.0\
     powershell.exe
  Exit Status:  0x0
```

Výpis B.1: Logový záznam číslo 1

```
10/15/2020 12:15:24 PM
LogName=System
SourceName=Microsoft-Windows-Service Control Manager
EventCode=7036
EventType=4
Type=Information
ComputerName=win-dc-800.attackrange.local
TaskCategory=None
OpCode=The operation completed successfully.
RecordNumber=80474
Keywords=Classic
Message=The Windows Update service entered the stopped state
   .
```

Výpis B.2: Logový záznam číslo 2

```
10/08/2020 09:48:11 AM
LogName=System
SourceName=Microsoft-Windows-Kernel-Boot
EventCode=32
EventType=4
Type=Information
ComputerName=EC2AMAZ-V5DC57V
User=NOT_TRANSLATED
Sid=S-1-5-18
SidType=0
TaskCategory=None
OpCode=Info
RecordNumber=79389
Keywords=None
Message=The bootmgr spent 0 ms waiting for user input.
```

Výpis B.3: Logový záznam číslo 3

```
11/25/2021 10:01:11 AM
LogName=Microsoft-Windows-PowerShell/Operational
SourceName=Microsoft-Windows-PowerShell
EventCode=4105
EventType=5
Type=Verbose
ComputerName=win-dc-266.attackrange.local
User=NOT_TRANSLATED
Sid=S-1-5-21-3499523948-2023901041-105020508-500
SidType=0
TaskCategory=Starting Command
OpCode=On create calls
RecordNumber=88855
Keywords=None
Message=Started invocation of ScriptBlock ID: c842dcf0-e3e9
   -4b27-a5b4-d19749f6a53d
Runspace ID: 17da6489-262c-44a8-8ea5-7379e8ac753b
```

Výpis B.4: Logový záznam číslo 4

```
<Event><System><Provider Name="Linux-Sysmon" Guid="{ff032593-
   a8d3-4f13-b0d6-01fc615a0f97}"/><EventID>5</EventID><
   Version>3</Version><Level>4</Level><Task>5</Task><Opcode
   >0</Opcode><Keywords>0x8000000000000000</Keywords><
   TimeCreated SystemTime="2022-01-14T00:18:39.316837000Z
   "/><EventRecordID>116948</EventRecordID><Correlation/><
   Execution ProcessID="4496" ThreadID="4496"/><Channel>
   Linux-Sysmon/Operational</Channel><Computer>
   sysmonlinux-51</Computer><Security UserId="0"/></System><
   EventData><Data Name="RuleName">-</Data><Data Name="
   UtcTime">2022-01-14 00:18:39.299</Data><Data Name="
   ProcessGuid">{00000000-0000-0000-0000-000000000000}</Data
   ><Data Name="ProcessId">13514</Data><Data Name="Image">&
   lt;unknown process&gt;</Data><Data Name="User">ubuntu</
   Data></EventData></Event>
```

Výpis B.5: Logový záznam číslo 5

```
<Event xmlns='http://schemas.microsoft.com/win/2004/08/
   events/event'><System><Provider Name='Service Control
   Manager' Guid='{555908d1-a6d7-4695-8e1e-26931d2012f4}'
   EventSourceName='Service Control Manager'/><EventID
   Qualifiers='16384'>\textbf{7045}</EventID><Version>0</
   Version><Level>4</Level><Task>0</Task><Opcode>0</Opcode><
   Keywords>0x8080000000000000</Keywords><TimeCreated
   SystemTime='2023-09-15T01:43:27.510804300Z'/><
   EventRecordID>155511</EventRecordID><Correlation/><
   Execution ProcessID='600' ThreadID='2556'/><Channel>
   System</Channel><Computer>ar-win-2.attackrange.local</
   Computer><Security UserID='S
   -1-5-21-390003789-444691441-1275789787-500'/></System><
   EventData><Data Name='ServiceName'>dbutil_2_3</Data><Data
    Name='ImagePath'>dbutil_2_3.sys</Data><Data Name='
   ServiceType'>kernel mode driver</Data><Data Name='
   StartType'>demand start</Data><Data Name='AccountName'></
   Data></EventData></Event>
```

Výpis B.6: Logový záznam číslo 6

```
Jul 22 13:04:31 ar-linux-2 useradd[3062925]: new user: name=
   eviluser, UID=1003, GID=1003, home=/home/eviluser, shell=/
   bin/bash, from=/dev/pts/1
```

Výpis B.7: Logový záznam číslo 7

```
{"timestamp":"2023-08-23T12:36:37.187172+0000","flow_id":
   903006245578527,"in_iface":"ens5","event_type":"flow","
   src_ip":"2.1.2.3","src_port":12346,"dest_ip":"10.2.2.2","
   dest_port":10080,"proto":"TCP","app_proto":"http","flow
   ":{"pkts_toserver":6,"pkts_toclient":5,"bytes_toserver
   ":710,"bytes_toclient":665,"start":"2023-08-23T12
   :35:23.341319+0000","end":"2023-08-23T12
   :35:36.887625+0000","age":13,"state":"closed","reason":"
   timeout","alerted":false},"tcp":{"tcp_flags":"1b","
   tcp_flags_ts":"1b","tcp_flags_tc":"1b","syn":\textbf{true
   },"fin":\textbf{true},"psh":true,"ack":true,"state":"
   closed","ts_max_regions":1,"tc_max_regions":1}}
```

Výpis B.8: Logový záznam číslo 8

```
03-12-2024 23:29:50.213 +0000 DEBUG JsonWebToken - Validating
    token:None.
```

Výpis B.9: Logový záznam číslo 9

```
5.188.210.227 - - [23/Aug/2021:02:13:36 +0000] "GET http
   ://5.188.210.227/echo.php HTTP/1.1" 400 650 "https://www.
   google.com/" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit
   /537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari
   /537.36" 361 0.000 [] [] - - - - 03
   f1eecc686e992939aa90f990c672f4
```

Výpis B.10: Logový záznam číslo 10

C Predikce maskovaných entit testovací datové sady Splunk Attack Range modely MLM a NWP

Log Index	Entity	Original Entity	Augmented Entity	F1 Score	Precision	Recall	Levenshtein Distance
1	1	9/17/2020	=	0.7561	0.8035	0.7140	10
	2	Information	user	0.9700	0.9700	0.9700	10
	3	administrator	administrator	0.9999	0.9999	0.9999	0
	Mean			0.9087	0.9245	0.8946	6.67
2	1	12:15:24	:	0.7500	0.7934	0.7111	7
	2	7036	start	0.8925	0.9382	0.8511	5
	3	successfully	successfully	0.9999	0.9999	0.9999	0
	4	80474	0	0.8740	0.9160	0.8356	4
	Mean			0.8791	0.9119	0.8494	4,00
3	1	System	system	0.9994	0.9994	0.9994	1
	2	NOT_TRANSLATED	start	0.8109	0.8987	0.7388	14
	3	None	start	0.9990	0.9990	0.9990	5
	4	0	the	0.9752	0.9752	0.9752	3
	5	user	the	0.9991	0.9991	0.9991	3
	Mean			0.9567	0.9743	0.9423	5.2
4	1	Microsoft	microsoft	0.8075	0.7626	0.8580	1
	2	Windows	windows	0.9987	0.9987	0.9987	1
	3	4105	0	0.8589	0.9136	0.8103	3
	4	Command	command	0.9337	0.9337	0.9337	1
	Mean			0.8997	0.9021	0.9002	1.50
5	1	Linux	microsoft	0.8092	0.7639	0.8602	8
	2	5	5	1.0000	1.0000	1.0000	0
	3	sysmonlinux	works	0.827	0.885	0.777	10
	4	ubuntu	system	0.846	0.895	0.802	6
	Mean			0.871	0.886	0.860	6.00
6	1	Service	service	0.999	0.999	0.999	1
	2	Manager	manager	0.999	0.999	0.999	1
	3	7045	600	0.884	0.936	0.837	3
	4	2023	2020	0.881	0.922	0.842	1
	5	09	09	0.9999	0.9999	0.9999	0
	6	mode	time	0.9928	0.9928	0.9928	3
	7	start	start	1.0000	1.0000	1.0000	0
	Mean			0.9651	0.9785	0.9529	1.29
7	1	eviluser	1	0.8716	0.9279	0.8218	8
	2	/home/eviluser	1	0.7853	0.8199	0.7534	14
	Mean			0.8285	0.8739	0.7876	11.00
8	1	903006245578527	1	0.7920	0.8609	0.7333	15
	2	2.1.2.3	host	0.8729	0.8776	0.8682	7
	3	TCP	daemon	0.9907	0.9907	0.9907	6
	4	true	1	0.9972	0.9972	0.9972	4
	5	true	0	0.9743	0.9743	0.9743	4
	6	true	0	0.9743	0.9743	0.9743	4
	Mean			0.9336	0.9458	0.9230	6.67
9	1	03	2016	0.9991	0.9991	0.9991	3
	2	2024	16	0.9990	0.9990	0.9990	4
	3	None	.	0.8347	0.8347	0.8347	4
	Mean			0.9443	0.9443	0.9443	3.67

Log Index	Entity	Original Entity	Augmented Entity	F1 Score	Precision	Recall	Levenshtein Distance
10	1	210	210	1.0000	1.0000	1.0000	0
	2	227	227	0.9999	0.9999	0.9999	0
	3	HTTP	##kit	0.8545	0.8007	0.9160	5
	4	google	microsoft	0.8061	0.7617	0.8560	8
	5	KHTML, like Gecko	-	0.7934	0.8050	0.7821	17
	Mean			0.8908	0.8735	0.9108	6.00

Tab. C.1: Generované hodnoty modelem `ELECTRA-small-generator`

Log Index	Entity	Original Entity	Augmented Entity	F1 Score	Precision	Recall	Levenshtein Distance
1	1	9/17/2020	19	0.7876	0.8605	0.7261	9
	2	Information	virus	0.9662	0.9662	0.9662	10
	3	administrator	system	0.9995	0.9995	0.9995	11
	Mean			0.9178	0.9421	0.8973	10.00
2	1	12:15:24	:	0.7500	0.7934	0.7111	7
	2	7036	1	0.8947	0.9397	0.8539	4
	3	successfully	successfully	1.0000	1.0000	1.0000	0
	4	80474	1	0.8882	0.9310	0.8491	5
	Mean			0.8832	0.9160	0.8535	4.00
3	1	System	system	0.9994	0.9994	0.9994	1
	2	NOT_TRANSLATED	administrator	0.8070	0.8936	0.7357	14
	3	None	1	0.9984	0.9984	0.9984	4
	4	0	is	0.9765	0.9765	0.9765	2
	5	user	user	1.0000	1.0000	1.0000	0
	Mean			0.9563	0.9736	0.9420	4.20
4	1	Microsoft	microsoft	0.8075	0.7626	0.8580	1
	2	Windows	windows	0.9987	0.9987	0.9987	1
	3	4105	information	0.8678	0.9221	0.8196	11
	4	Command	worker	0.9972	0.9972	0.9972	6
	Mean			0.9178	0.9202	0.9184	4.75
5	1	Linux	microsoft	0.8092	0.7639	0.8602	8
	2	5	5	1.0000	1.0000	1.0000	0
	3	sysmonlinux	pedro	0.8189	0.8352	0.8033	10
	4	ubuntu	system	0.8463	0.8954	0.8024	6
	Mean			0.8686	0.8736	0.8665	6.00
6	1	Service	service	0.9993	0.9993	0.9993	1
	2	Manager	manager	0.9989	0.9989	0.9989	1
	3	7045	0	0.8702	0.9227	0.8233	3
	4	2023	2020	0.8806	0.9224	0.8425	1
	5	09	07	0.9999	0.9999	0.9999	1
	6	mode	mode	1.0000	1.0000	1.0000	0
	7	start	start	1.0000	1.0000	1.0000	0
	Mean			0.9641	0.9776	0.9520	1
7	1	eviluser	root	0.8733	0.9295	0.8235	8
	2	/home/eviluser	root	0.7843	0.8196	0.7518	13
	Mean			0.8288	0.8745	0.7877	10.50
8	1	903006245578527	0	0.7861	0.8509	0.7305	14
	2	2.1.2.3	53	0.8755	0.8811	0.8700	6
	3	TCP	http	0.9983	0.9983	0.9983	4
	4	true	false	0.9991	0.9991	0.9991	4
	5	true	false	0.9991	0.9991	0.9991	4
	6	true	false	0.9991	0.9991	0.9991	4
	Mean			0.9429	0.9546	0.9327	6.00
9	1	03	2016	0.9991	0.9991	0.9991	3
	2	2024	04	0.9984	0.9984	0.9984	2
	3	None	0	0.9772	0.9772	0.9772	4
	Mean			0.9916	0.9916	0.9916	3.00
10	1	210	210	1.0000	1.0000	1.0000	0
	2	227	227	0.9999	0.9999	0.9999	0
	3	HTTP	/	0.8565	0.8565	0.8565	4
	4	google	microsoft	0.8061	0.7617	0.8560	8
	5	KHTML, like Gecko	compatible	0.8150	0.8492	0.7835	15
	Mean			0.8955	0.8935	0.8992	5.40

Tab. C.2: Generované hodnoty modelem `MobileBERT-uncased`

Log Index	Entity	Original Entity	Augmented Entity	F1 Score	Precision	Recall	Levenshtein Distance
1	1	9/17/2020	19	0.7876	0.8605	0.7261	9
	2	Information	User	0.9720	0.9720	0.9720	10
	3	administrator	Administrator	0.9970	0.9970	0.9970	2
	Mean			0.9189	0.9431	0.8984	7.00
2	1	12:15:24	-	0.7505	0.8019	0.7054	8
	2	7036	0	0.8777	0.9222	0.8373	3
	3	successfully	successfully	0.9999	0.9999	0.9999	1
	4	80474	0	0.8740	0.9160	0.8356	4
	Mean			0.8755	0.9100	0.8446	4.00
3	1	System		0	0	0	6
	2	NOT_TRANSLATED		0	0	0	14
	3	None	0	0.9772	0.9772	0.9772	4
	4	0	0	0.9999	0.9999	0.9999	1
	5	user	user	1.0000	1.0000	1.0000	1
	Mean			0.5954	0.5954	0.5954	5.2
4	1	Microsoft	Microsoft	1.0000	1.0000	1.0000	0
	2	Windows	Windows	1.0000	1.0000	1.0000	0
	3	4105	1	0.8703	0.9257	0.8212	3
	4	Command	Command	1.0000	1.0000	1.0000	1
	Mean			0.9676	0.9814	0.9553	1.00
5	1	Linux	Microsoft	0.9991	0.9991	0.9991	8
	2	5	5	1.0000	1.0000	1.0000	0
	3	sysmonlinux	shire	0.7942	0.7930	0.7955	9
	4	ubuntu	-	0.8060	0.8395	0.7751	6
	Mean			0.8998	0.9079	0.8924	5.75
6	1	Service	Service	0.9999	0.9999	0.9999	0
	2	Manager	Manager	1.0000	1.0000	1.0000	1
	3	7045	1500	0.8841	0.9370	0.8368	4
	4	2023	2020	0.8806	0.9224	0.8425	1
	5	09	09	0.9999	0.9999	0.9999	0
	6	mode	mode	1.0000	1.0000	1.0000	1
	7	start	start	1.0000	1.0000	1.0000	1
	Mean			0.9664	0.9799	0.9542	1.14
7	1	eviluser	root	0.8733	0.9295	0.8235	8
	2	/home/eviluser	root	0.7843	0.8196	0.7518	13
	Mean			0.8288	0.8745	0.7877	10.50
8	1	903006245578527	0	0.7861	0.8509	0.7305	14
	2	2.1.2.3	0	0.8675	0.8698	0.8652	7
	3	TCP	localhost	0.8666	0.8165	0.9233	9
	4	true	true	0.9999	0.9999	0.9999	0
	5	true	true	0.9999	0.9999	0.9999	0
	6	true	true	0.9999	0.9999	0.9999	0
	Mean			0.9200	0.9229	0.9198	5.00
9	1	03	2022	0.9991	0.9991	0.9991	4
	2	2024	19	0.9991	0.9991	0.9991	4
	3	None	succeeded	0.9982	0.9982	0.9982	9
	Mean			0.9988	0.9988	0.9988	5.67
10	1	210	210	1.0000	1.0000	1.0000	0
	2	227	227	0.9999	0.9999	0.9999	0
	3	HTTP	HTTP	1.0000	1.0000	1.0000	1
	4	google	youtube	0.9914	0.9914	0.9914	5
	5	KHTML, like Gecko	compatible	0.8150	0.8492	0.7835	15
	Mean			0.9613	0.9681	0.9550	4.20

Tab. C.3: Generované hodnoty modelem `RoBERTa-base`

Log Index	Entity	Original Entity	Augmented Entity	F1 Score	Precision	Recall	Levenshtein Distance
1	1	9/17/2020	AUDIT_SUCCESS	0.7662	0.7612	0.7713	13
	2	Information	INFO	0.9887	0.9887	0.9887	10
	3	administrator	administrator	1.0000	1.0000	1.0000	0
	Mean			0.9183	0.9166	0.9200	7.67
2	1	12:15:24	2020-10-15T14:35:00.000	0.8359	0.8136	0.8593	19
	2	7036	4624	0.9407	0.9407	0.9407	4
	3	successfully	successfully	1.0000	1.0000	1.0000	0
	4	80474	12345	0.9093	0.9093	0.9093	5
	Mean			0.9215	0.9159	0.9273	7.00
3	1	System	MyLog	0.8369	0.7685	0.9186	5
	2	NOT_TRANSLATED	EC2AMAZ-V5DC57V\Administrator	0.7764	0.7800	0.7729	28
	3	None	BootMgrInput	0.8449	0.7919	0.9056	10
	4	0	spent	0.9771	0.9771	0.9771	5
	5	user	Keyboard	0.9975	0.9975	0.9975	7
	Mean			0.8866	0.8630	0.9143	11.00
4	1	Microsoft	DC266PShell	0.8344	0.7767	0.9013	11
	2	Windows	PowerShellCore	0.8719	0.8332	0.9144	13
	3	4105	4101	0.8939	0.8939	0.8939	1
	4	Command	ScriptBlockIntialized	0.8420	0.7914	0.8994	18
	Mean			0.8605	0.8238	0.9023	10.75
5	1	Linux	Sysmon-Sys	0.8167	0.7539	0.8908	9
	2	5	116948	0.8398	0.7843	0.9038	6
	3	sysmonlinux	DESKTOP-51	0.8102	0.8043	0.8162	11
	4	ubuntu	root	0.8508	0.9013	0.8057	5
	Mean			0.8294	0.8110	0.8541	6.20
6	1	Service	MyService	0.8910	0.8512	0.9347	2
	2	Manager	Manager	1.0000	1.0000	1.0000	0
	3	7045	1000	0.8835	0.9362	0.8364	3
	4	2023	10/15/2021	0.7749	0.7366	0.8174	7
	5	09	43:27.5	0.7553	0.6980	0.8228	17
	6	mode	file	0.9101	0.9101	0.9101	3
	7	start	start	1.0000	1.0000	1.0000	0
	Mean			0.8878	0.8760	0.9031	4.57
7	1	eviluser	johnDoe	0.8324	0.8155	0.8500	7
	2	/home/eviluser	/home/johnDoe	0.9033	0.8925	0.9143	7
	Mean			0.8678	0.8540	0.8822	7.00
8	1	903006245578527	f8a41231-3456-4567-8901-234567890123	0.8138	0.7843	0.8456	28
	2	2.1.2.3	192.168.1.100	0.9267	0.9157	0.9380	8
	3	TCP	tcp	0.9964	0.9964	0.9964	3
	4	true	0	0.9743	0.9743	0.9743	4
	5	true	1	0.9972	0.9972	0.9972	4
	6	true	1	0.9972	0.9972	0.9972	4
	Mean			0.9509	0.9442	0.9581	8.50
9	1	03	2022	0.9991	0.9991	0.9991	3
	2	2024	abc123	0.8175	0.7565	0.8892	5
	3	None	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9	0.7060	0.6204	0.8190	34
	Mean			0.8409	0.7920	0.9024	14.00
10	1	210	210	1.0000	1.0000	1.0000	0
	2	227	227	1.0000	1.0000	1.0000	0
	3	HTTP	HTTP	1.0000	1.0000	1.0000	0
	4	google	example	0.9988	0.9988	0.9988	5
	5	KHTML, like Gecko	KHTML	0.9254	0.9783	0.8779	12
	Mean			0.9849	0.9954	0.9754	3.40

Tab. C.4: Generované hodnoty modelem `Llama3:70B`

Log Index	Entity	Original Entity	Augmented Entity	F1 Score	Precision	Recall	Levenshtein Distance
1	1	9/17/2020	Audit Success	0.7933	0.8482	0.7451	13
	2	Information	administrator	0.9685	0.9685	0.9685	10
	3	administrator	Normal termination	0.8598	0.8258	0.8967	13
	Mean			0.8739	0.8808	0.8701	12.00
2	1	12:15:24	4:37	0.8777	0.8898	0.8659	7
	2	7036	104	0.8915	0.9368	0.8503	3
	3	successfully	successfully	1.0000	1.0000	1.0000	0
	4	80474	37689	0.9396	0.9396	0.9396	5
	Mean			0.9272	0.9416	0.9140	3.75
3	1	System	System	1.0000	1.0000	1.0000	0
	2	NOT_TRANSLATED	NT Authority\System	0.8052	0.8341	0.7783	17
	3	None	Boot	0.9984	0.9984	0.9984	3
	4	0	3000	0.9767	0.9767	0.9767	3
	5	user	user	1.0000	1.0000	1.0000	0
	Mean			0.9561	0.9618	0.9507	4.60
4	1	Microsoft	Microsoft	1.0000	1.0000	1.0000	0
	2	Windows	Windows	1.0000	1.0000	1.0000	0
	3	4105	400	0.8683	0.9235	0.8194	2
	4	Command	PowerShell	0.9973	0.9973	0.9973	9
	Mean			0.9664	0.9802	0.9542	2.75
5	1	Linux	host	0.9982	0.9982	0.9982	5
	2	5	1	0.9994	0.9994	0.9994	1
	3	sysmonlinux	host-51	0.8047	0.8110	0.7985	10
	4	ubuntu	-	0.8060	0.8395	0.7751	6
	Mean			0.9021	0.9120	0.8928	5.50
6	1	Service	SC	0.997	0.997	0.997	6
	2	Manager	Control	0.998	0.998	0.998	6
	3	7045	7036	0.923	0.923	0.923	2
	4	2023	2023-11	0.901	0.870	0.935	3
	5	09	15	0.999	0.999	0.999	2
	6	mode	device	0.933	0.933	0.933	5
	7	start	start	1.0000	1.0000	1.0000	0
	Mean			0.9644	0.9599	0.9693	3.43
7	1	eviluser	user1003	0.8483	0.8539	0.8428	8
	2	/home/eviluser	/home/user1003	0.9138	0.9044	0.9234	8
	Mean			0.8811	0.8792	0.8831	8.00
8	1	903006245578527	123456789	0.8639	0.8838	0.8448	10
	2	2.1.2.3	192.168.1.10	0.9330	0.9238	0.9424	7
	3	TCP	tcp	0.9964	0.9964	0.9964	3
	4	true	true	0.9999	0.9999	0.9999	0
	5	true	false	0.9991	0.9991	0.9991	4
	6	true	true	0.9999	0.9999	0.9999	0
	Mean			0.9654	0.9672	0.9638	4.00
9	1	03	2023	0.8821	0.8442	0.9236	2
	2	2024	05	0.9985	0.9985	0.9985	3
	3	None	abcdef12-3456-789a-bcde-f0123456789a	0.7060	0.6212	0.8177	35
	Mean			0.8622	0.8213	0.9133	13.33
10	1	210	210	0.9999	0.9999	0.9999	0
	2	227	227	1.0000	1.0000	1.0000	0
	3	HTTP	HTTP	1.0000	1.0000	1.0000	0
	4	google	example	0.9988	0.9988	0.9988	5
	5	KHTML, like Gecko	KHTML	0.9254	0.9783	0.8779	12
	Mean			0.9849	0.9954	0.9754	3.40

Tab. C.5: Generované hodnoty modelem `DeepSeek-R1:32B`

D Metaklíče typů entit v testovacím datasetu

Štítek	Význam	Příklad
`action`	Konkrétní provedená činnost během události	"Modify"
`crypto`	Typ kryptografického primitiva nebo algoritmu	"SHA2-512"
`description`	Popis se vztahuje k podrobnému vysvětlení nebo shrnutí události	"*Unauthorized access attempt*"
`device_ip`	IP adresu zařízení, které vygenerovalo nebo nahlásilo událost protokolu	"192.168.1.100"
`domain_dest`	Název domény přidružený k cíli v síťové události	"example.com"
`domain_src`	Název domény spojený se zdrojem síťové nebo systémové události	"company.com"
`event_id`	Jedinečný identifikátor přiřazený každému typu události	4624, 4663
`event_level`	Závažnost nebo důležitost události	"Information"
`event_type`	Typ nebo kategorii události, která nastala	"Authentication"

Tab. D.1: Přehled štítků v testovací datové sadě – část 1.

Štítek	Význam	Příklad
`file_name`	Název souboru tak, jak je zobrazen v souborovém systému	"malware.exe"
`file_path`	Úplnou cestu k adresáři vedoucímu k určitému souboru	"C:\Windows\System32\..."
`host_dest`	Označení cílového hostitele, kterého se událost týká	"Finance Dept Server"
`host_name`	Název přiřazený konkrétnímu hostiteli (systému, serveru nebo zařízení)	"Server01"
`host_src`	Označení zdrojového hostitele nebo systému, který událost spustil nebo inicioval	"User Laptop"
`interface_dest`	Síťové rozhraní v cílovém systému, které přijalo síťový provoz nebo komunikaci	"eth1"
`interface_src`	Zdrojové síťové rozhraní, přes které událost nebo síťová komunikace vznikla	"eth0"
`ip_dest`	IP adresa cílového hostitele nebo systému, který přenos přijal	"192.168.1.100"
`ip_src`	IP adresa systému, který inicioval připojení, požadavek nebo síťovou komunikaci	"10.0.0.1"
`mac_dest`	MAC (*Media Access Control*) adresa cílového síťového zařízení	"00-1B-44-11-3A-B7"
`mac_src`	MAC adresa zdrojového zařízení, které iniciovalo komunikaci	"00-1B-44-11-3A-B8"

Tab. D.2: Přehled štítků v testovací datové sadě – část 2.

Štítek	Význam	Příklad
`mask_src`	Maska podsítě přidružená ke zdrojové IP	"255.255.255.128"
`network_protocol`	Síťový protokol aplikační vrstvy nebo síťový protokol vyšší úrovně	"TCP", "UDP"
`os_version`	Verze operačního systému (OS) spuštěného na hostitelském počítači	"10.0"
`port_dest`	Číslo cílového síťového portu, na který je komunikace zaměřena	"443"
`port_src`	Číslo zdrojového síťového portu, který je součástí komunikační události	"80"
`process_name`	Název procesu, proveden v rámci zaznamenané události	"notepad.exe"
`result`	Výsledek akce, tedy zda byla akce úspěšná, neúspěšná nebo měla jiný důsledek	"Success", "Failure"
`result_code`	Číselná nebo alfanumerická hodnota, odpovídající konkrétnímu výsledku události	"0", "1", "1003",
`rule`	Definovaná zásada, detekční pravidlo nebo konfigurace zabezpečení, které spustily událost	"*inbound policy*"
`service_name`	Název služby na úrovni systému, které se týká zaznamenaná událost	"EventLog, RemoteAccess, WinDefender, dnscache, EventLog"

Tab. D.3: Přehled štítků v testovací datové sadě – část 3.

Štítek	Význam	Příklad
`session_id`	Jedinečný identifikátor přiřazený relaci uživatele, logicky spojující uživatele nebo proces a systém	"3, 1047, 4625"
`severity`	Dopad nebo důsledek. Závažnost popisuje míru škody nebo rizika	"Low", "Medium", "High", "Critical"
`tags`	Značky spojené s událostí	"phishing", "e-mail"
`timestamp`	Časová značka události	"2024-04-12T14:23:05Z"
`traffic_direction`	Popisuje směr toku síťové komunikace vzhledem ke sledovanému systému, perimetru nebo zařízení	"inbound"
`url`	Zachycuje *Uniform Resource Locator*, který je součástí zaznamenané události	"http://malicious-site.com"
`user_dest`	Uživatelský účet, který je cílem události nebo který byl událostí ovlivněn	"foltyn.ondrej"
`user_group`	Skupina uživatelských účtů nebo skupina služby *Active Directory*	"Domain Admins"
`user_src`	Zdrojový uživatelský účet, který představuje uživatele, jenž událost inicioval	"jsmith"
`O`	Výchozí označení pro neurčité části textu (*Outside*)	Většina vstupního textu

Tab. D.4: Přehled štítků v testovací datové sadě – část 4.