



**FAKULTA APLIKOVANÝCH VĚD  
ZÁPADOČESKÉ UNIVERZITY  
V PLZNI**

**KATEDRA  
KYBERNETIKY**

## **Bakalářská práce**

# **Neuronové sítě pro porozumění řeči**

**Vladimíra Kimlová**





FAKULTA APLIKOVANÝCH VĚD  
ZÁPADOČESKÉ UNIVERZITY  
V PLZNI

KATEDRA  
KYBERNETIKY

## **Bakalářská práce**

# **Neuronové sítě pro porozumění řeči**

Vladimíra Kimlová

### **Vedoucí práce**

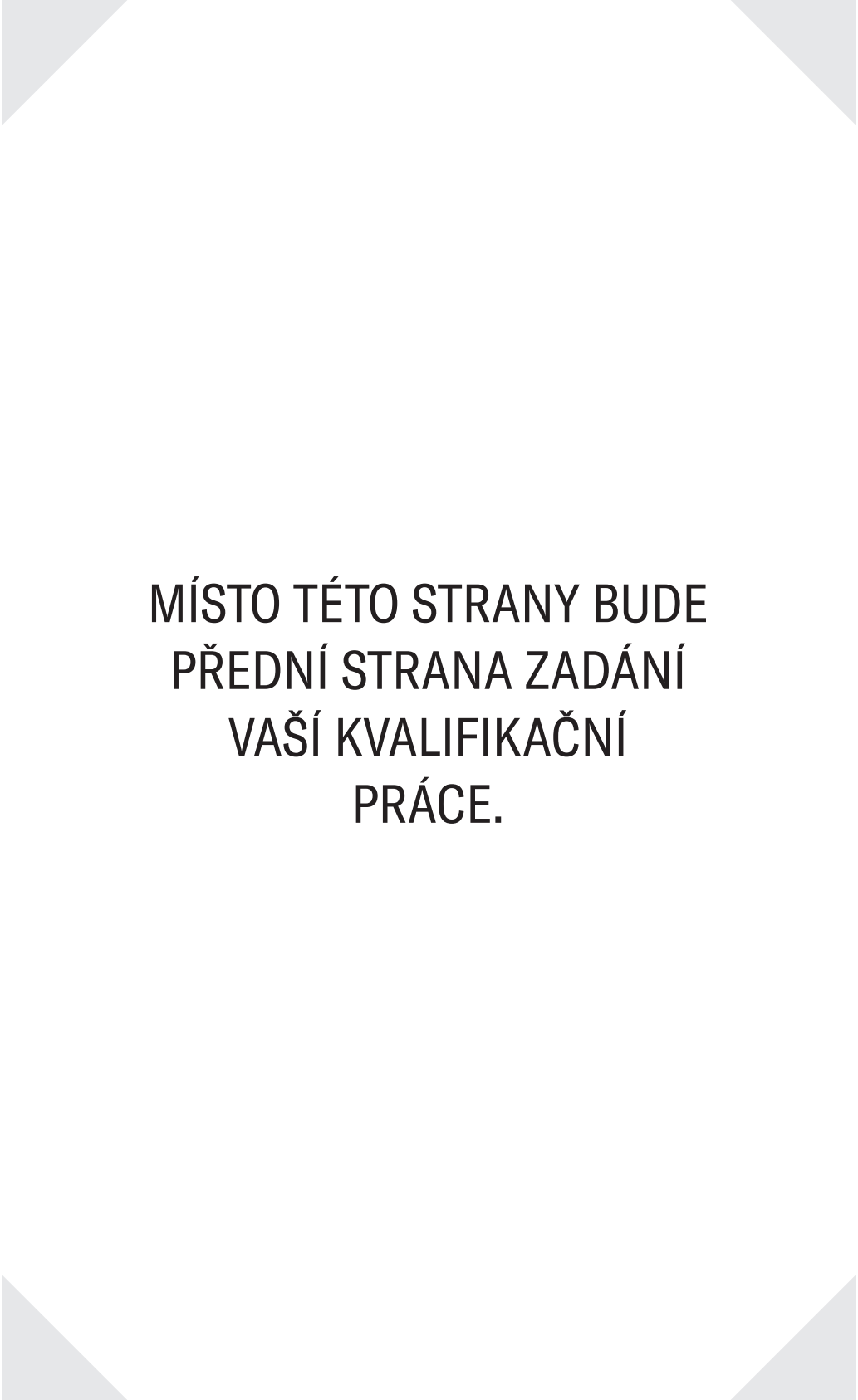
Ing. Bulín Martin, M.Sc.

© Vladimíra Kimlová, 2023.

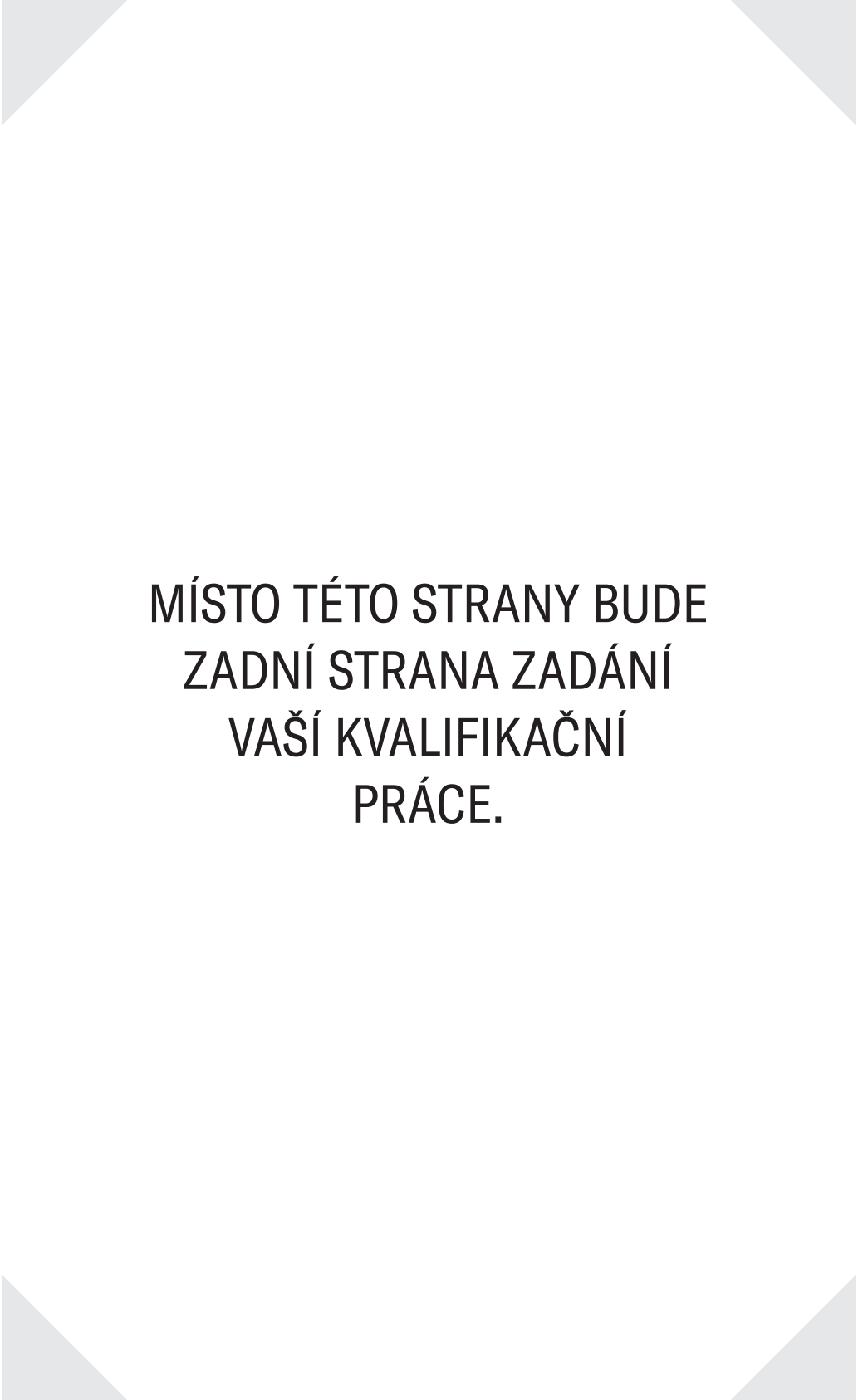
Všechna práva vyhrazena. Žádná část tohoto dokumentu nesmí být reprodukována ani rozšiřována jakoukoli formou, elektronicky či mechanicky, fotokopírováním, nahráváním nebo jiným způsobem, nebo uložena v systému pro ukládání a vyhledávání informací bez písemného souhlasu držitelů autorských práv.

**Citace v seznamu literatury:**

KIMLOVÁ, Vladimíra. *Neuronové sítě pro porozumění řeči*. Plzeň, 2023. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra kybernetiky. Vedoucí práce Ing. Bulín Martin, M.Sc.



MÍSTO TÉTO STRANY BUDE  
PŘEDNÍ STRANA ZADÁNÍ  
VAŠÍ KVALIFIKAČNÍ  
PRÁCE.



MÍSTO TÉTO STRANY BUDE  
ZADNÍ STRANA ZADÁNÍ  
VAŠÍ KVALIFIKAČNÍ  
PRÁCE.

# Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Plzni dne 22. května 2023

.....

Vladimíra Kimlová

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

# Abstrakt

Český abstrakt

# Abstract

Anglický abstrakt

## Klíčová slova

dopředná neuronová síť • Human in the loop • přetrénování (Retraining) neuronové sítě • klasifikace záměru





# Poděkování

Nejprve bych chtěla poděkovat Ing. Martinu Bulínovi, M.Sc, svému vedoucímu práce, za jeho odborné rady, cenné nápady a ochotu věnovat mi svůj čas a energii. Bez jeho podpory by tato práce nebyla možná.



Ráda bych také poděkovala své rodině a přátelům za jejich nekonečnou podporu, povzbuzení a laskavá slova během mého studia.




# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
1.1	Cíle práce . . . . .	2
1.2	Současný stav problematiky . . . . .	2
1.2.1	Human-in-the-loop . . . . .	2
1.2.2	Klasifikace záměru . . . . .	3
1.3	Struktura práce . . . . .	4
<b>2</b>	<b>Použité metody a technologie</b>	<b>7</b>
2.1	Dopředná neuronová síť . . . . .	7
2.1.1	Hyperparametry . . . . .	12
2.2	Sentence transformery . . . . .	15
2.2.1	FERNET-C5 . . . . .	17
2.3	Fine-tuning a Transfer learning . . . . .	17
2.4	ROS (Robot Operating System) . . . . .	18
2.5	Speechcloud . . . . .	20
2.6	Komunikační protokoly . . . . .	21
2.6.1	HTTP . . . . .	22
2.6.2	MQTT . . . . .	22
2.6.3	WebSockets . . . . .	23
2.7	Základní webové technologie . . . . .	23
<b>3</b>	<b>Učení neuronové sítě hlasovým dialogem</b>	<b>25</b>
3.1	Neuronová síť pro klasifikaci záměru . . . . .	25
3.2	Algoritmus dialogu . . . . .	26
3.3	Uživatelské rozhraní . . . . .	28
<b>4</b>	<b>Data</b>	<b>31</b>
4.1	Dataset D1 . . . . .	31
4.2	Dataset D2 . . . . .	31

<b>5</b>	<b>Experimenty a výsledky</b>	<b>33</b>
5.1	Porovnání frameworků . . . . .	33
5.2	Volba struktury sítě . . . . .	35
5.3	Trénování v závislosti na počtu vzorků a tříd . . . . .	37
5.4	Analýza použití předtrénované neuronové sítě . . . . .	40
5.5	Vyhodnocení metody učení dialogem . . . . .	43
<b>6</b>	<b>Aplikace</b>	<b>49</b>
6.1	Cílová robotická platforma . . . . .	49
6.2	Učení konkrétních záměrů . . . . .	52
<b>7</b>	<b>Diskuze</b>	<b>53</b>
7.1	Rekapitulace metod . . . . .	53
7.2	Shrnutí výsledků . . . . .	54
<b>8</b>	<b>Závěr</b>	<b>59</b>
8.1	Future Work . . . . .	59
<b>A</b>	<b>Dataset D1</b>	<b>61</b>
<b>B</b>	<b>Testované hyperparematry pro neuronovou síť na RPi</b>	<b>63</b>
<b>C</b>	<b>Graf k experimentu 5.4</b>	<b>65</b>
<b>D</b>	<b>Vzorový dataset pro učení dialogem</b>	<b>67</b>
	<b>Bibliografie</b>	<b>71</b>
	<b>Seznam obrázků</b>	<b>75</b>
	<b>Seznam tabulek</b>	<b>77</b>

V posledních letech se neuronové sítě staly dominantním nástrojem pro mnoho úloh v oblasti strojového učení a umělé inteligence. Nicméně k dosažení vysoké přesnosti při řešení dané úlohy je obvykle zapotřebí velkého množství anotovaných dat pro trénování neuronové sítě. Sběr a anotace dat mohou být časově náročné, drahé a mnohdy jsou data nedostupná, což může být pro některé aplikace překážkou. Trénink na velkém množství dat může být také drahý a zabere hodně času, což způsobuje problémy při iterativním vylepšování modelů, protože s rostoucím množstvím dat se stává stále obtížnější udržet přehled o tom, co se poslednímu modelu podařilo naučit. Tyto problémy jsou zvláště patrné v případech, kdy je potřeba rychle se adaptovat na nové podmínky a naučit se novému chování, jako například v oblasti *human-robot interaction*  v metodách učení přímo za běhu programu. Proto se v takových situacích často používá přístup zvaný „*human in the loop*“, který umožňuje zapojení lidí do tréninkového procesu. Tento přístup umožňuje využívat odborné znalosti a schopnosti člověka pro rychlou a flexibilní reakci na nové situace a zvyšuje kvalitu výsledných modelů tím, že lidé poskytují cennou zpětnou vazbu a pomoc při anotaci dat pro  učení modelu. Human-in-the-loop přístup se využívá v mnoha oblastech, včetně strojového učení, robotiky, medicíny, průmyslu, autonomních vozidel a online reklamy.

Tato práce nabízí metodu trénování, která umožňuje lidskou interakci s trénovacím algoritmem přímo za běhu programu (tzv. *human-in-the-loop* přístup [14]). Cílem je umožnit uživateli směřovat učení neuronové sítě pomocí zpětné vazby, což umožňuje neuronové síti učit se nové věci v reálném čase a rychleji se přizpůsobovat měnícím se podmínkám. Metoda je demonstrována na jednoduché dopředné neuronové síti používané pro klasifikaci záměrů. Dialog učení je realizován pomocí hlasového rozhraní a pro převod vstupních dat na vektory je používán SentenceTransformer [22]. V rámci experimentů byly analyzovány různé frameworky pro implementaci sítě, s důrazem na výběr optimálního frameworku (v tomto případě PyTorch). Dále byla zkoumána optimální velikost sítě pro daný

problém a pro nasazení na Raspberry Pi. Byl také analyzován vliv přeučení (Retrainingu) na výsledky klasifikace, včetně použití *fine-tuningu* pro inicializaci sítě. Výsledky experimentů ukázaly, ...

V rámci práce bylo dále navrženo řešení jednoho ze záměrů pro vyhledání informace pomocí vyhledávače Google, respektive pomocí tzv. *Featured Snippet*, což je zvláštní zobrazení výsledků vyhledávání na Google, které prezentuje nejrelevantnější odpověď na konkrétní dotaz. Navíc bylo vytvořeno grafické uživatelské rozhraní (GUI) pro aplikaci na robotickou platformu.

## 1.1 Cíle práce

Cíle práce jsou následující:

1. Nastudovat problematiku klasifikace neuronovými sítěmi a seznámit se s knihovnami Keras a PyTorch.
2. Seznámit se s platformou Speechcloud a naučit se používat její služby pro rozpoznávání a syntézu řeči.
3. Navrhnout neuronovou síť pro klasifikaci záměru ze vstupní promluvy.
4. Vyvinout řešení pro generování odpovědí pro vybrané kategorie (záměry).
5. Prozkoumat možnosti automatického přeučení klasifikátoru na nových datech.
6. Aplikovat vyvinutou metodu včetně hlasové interakce na reálnou robotickou entitu.


## 1.2 Současný stav problematiky

### 1.2.1 Human-in-the-loop


V dnešní době existuje velká poptávka po řešeních založených na strojovém učení, protože pokroky, které se v posledních letech v této oblasti udály, ho popularizovaly a přiblížily širšímu publiku. Vytváření systémů strojového učení je však složitý proces, který vyžaduje hluboké znalosti technik strojového učení. V tradičním konceptu jsou učící algoritmy navrženy, vyvinuty a testovány, a pak nabízeny veřejnosti bez dalších změn. Tento monolitický přístup s sebou však nese důsledky, jako jsou obtížná rozšiřitelnost, staticnost, nemožnost správného

vyhodnocení a snížení výkonu v důsledku změn kontextu. Byly proto definovány nové typy interakcí mezi lidmi a algoritmy strojového učení, které lze zařadit pod označení *human-in-the-loop*. Cílem není pouze zlepšit přesnost strojového učení nebo dosáhnout požadované přesnosti rychleji, ale také zvýšit účinnost a efektivitu zapojení lidí do tohoto procesu.

Vztah člověka a procesu strojového učení umožňuje rozdělit tuto problematiku do následujících kategorií [14]:

- *Aktivní učení (Active Learning)*: Tento proces využívá lidskou expertizu k anotaci dat, strojový model si sám vybírá, která data by měla být anotována člověkem, aby se zlepšila jeho schopnost učit se a získat přesnější výsledky.
  - *Interaktivní strojové učení (Interactive machine learning)*: Interaktivní strojové učení zahrnuje bližší interakci mezi člověkem a učícím se systémem. Lidé aktivně přispívají k procesu učení poskytováním informací a zpětné vazby. Tento proces je častý, postupný a zaměřený na specifické úkoly, což vede k dosažení lepších výsledků než u tradičních metod strojového učení.
  - *Machine teaching*: Při machine teachingu má lidský odborník kontrolu nad procesem učení a rozhoduje, jaké informace budou do strojového modelu přeneseny.
  - *Curriculum learning*: Curriculum learning je metoda strojového učení, při které se postupně a systematicky předkládají příklady strojovému modelu tak, aby byl schopen lépe porozumět datovým vzorcům a zlepšit svou výkonost. Tento postup zahrnuje strukturované a logické načítání informací a příkladů, které jsou určeny odborníky v dané oblasti.
-  *Explainable AI*: Explainable Active Learning se věnuje analýze schopnosti modelů vysvětlit lidským uživatelům, jak bylo dosaženo určitého rozhodnutí nebo výsledku. Cílem tohoto výzkumného oboru je zlepšit srozumitelnost výstupů, které jsou generovány AI systémy.

## 1.2.2 Klasifikace záměru

 Klasifikace záměru je oblast strojového učení, která se zaměřuje na určování úmyslů lidí z textových nebo hlasových dat. Tato technologie se používá v mnoha aplikacích, jako jsou chatboti, asistenti a automatizované telefonické systémy. V současné době se v této oblasti používají pokročilé techniky strojového učení, jako jsou například rekurentní neuronové sítě (RNN) nebo transformery, které dokážou efektivně zpracovávat dlouhé sekvence textových nebo hlasových dat.



Například v práci *A comparative study of neural network models for lexical intent classification* [21] byly mezi sebou porovnávány dopředné, rekurentní, GRU a LSTM neuronové sítě pro datasety ATIS a Cortana. Nejlepších výsledků (nejnižší chybovosti) při úkolu klasifikace textu dosáhly gated rekurentní sítě GRU a LSTM (1.30 %), které měly téměř srovnatelný výkon. Poté následovaly běžné rekurentní sítě (2.45 %) a nakonec dopředné sítě (3.60 %).

Porovnání řady různých modelů (např. LSTM, BLSTM, GRU, BGRU) bylo provedeno v práci [34]. Porovnání bylo provedeno pro datasety SMP a RWC. Nejlepších výsledků dosáhla hybridní architektura konvoluční neuronové sítě a obousměrné gated rekurentní neuronové sítě (CNN-BGRU), 93 % přesnosti na datsetu SMP a 97 % na datasetu RWC.

Práce *Dialogue Intent Classification with Long Short-Term Memory Networks* [13] navrhla hluboké hierarchické LSTM modely pro klasifikaci záměrů dialogu v oblasti elektronického obchodu (na datasetu obsahujícím 24760 vět), tyto dva modely zahrnují HLSTM a memory-augmented HLSTM. Výsledky experimentů ukazují, že tyto navržené modely mají lepší výkon (přesnost 81.6 % a 83.9 %) než základní přístupy použité jako srovnání (např. LSTM model měl přesnost 79.7 %).

Přesnost modelů na rámcových (*in-scope*) datech<sup>1</sup> a datech mimo rámec (*out-of-scope*) byla testována v článku *An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction* [11]. Na rámcových datech dosáhly největší přesnosti BERT (96.2 %) a MLP (93.4 %). U dat mimo rámec dosáhly největší přesnosti MLP (47.4 %) a Rasa (45.3 %). Nejhuře si v obou případech vedl FastText (89.0 % a 9.7 %).

### 1.3 Struktura práce

V úvodní kapitole je představena metoda trénování neuronové sítě, která umožňuje lidskou interakci s trénovacím algoritmem přímo za běhu programu. Dále jsou zde popsány cíle práce a shrnut současný stav problematiky týkající se přístupu *human-in-the-loop* a klasifikace záměru.

Ve druhé kapitole jsou popsány použité metody a technologie. Nejprve je popsána dopředná neuronová síť a proces jejího učení, včetně nastavitelných hyperparametrů. Dále následuje popis Sentence transformerů, fine-tuningu a transfer learningu, Robot operating systému a Speechcloudu. Nakonec jsou zmíněny komunikační protokoly a základní webové technologie.

Třetí kapitola je věnována popisu navrženého řešení, tj. učení neuronové sítě hlasovým dialogem. Je představena neuronová síť pro klasifikaci záměru a její

---

<sup>1</sup> Stejný dataset byl použit i v této práci, viz sekce 4.2.





napojení na **Sentence transformers**, dále algoritmus dialogu a přeučení neuronové sítě. Nakonec je ukázáno uživatelské rozhraní.

Čtvrtá kapitola se zabývá popisem použitých datasetů, tj. kolik mají vzorků a jak byly tyto vzorky rozděleny.

Pátá kapitola je o experimentech a jejich výsledcích. Experimenty se zabývaly porovnáním frameworků pro tvorbu neuronové sítě, dále volbou struktury sítě a také trénováním v závislosti na počtu vzorků a tříd. Dále byla provedena analýza použití předtrénované neuronové sítě a vyhodnocení metody učení dialogem.

Šestá kapitola je věnována konkrétním aplikacím této práce, je zde popsána cílová robotická platforma a konkrétní učené záměry.

Sedmá kapitola se zabývá diskuzí výsledků dosažených v experimentech.

V závěrečné kapitole jsou shrnuty dosažené výsledky v rámci této práce. A jsou navržena další vylepšení do budoucna.

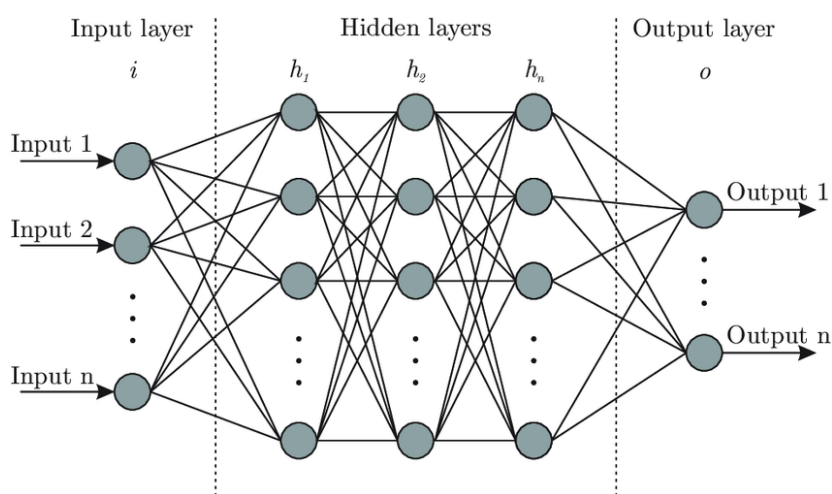


# Použité metody a technologie

## 2

### 2.1 Dopředná neuronová síť

Dopředná neuronová síť (Feedforward Neural Network) je typ umělé neuronové sítě, která se skládá z několika vrstev neuronů. První vrstva se nazývá vstupní, poslední výstupní a vrstvy mezi nimi jsou označovány jako skryté. Ve většině případů je každý neuron spojen se všemi neurony v následující vrstvě. Architektura je nastíněna na Obrázku 2.1.



Obrázek 2.1: Diagram architektury neuronové sítě (převzato z [6])

V dopředné neuronové síti je signál šířen pouze ve směru od vstupní vrstvy přes skryté vrstvy až k výstupní vrstvě. Matematicky lze dopřednou neuronovou síť popsat jako zobrazení vstupního signálu na výstupní signál. Obecně platí, že velikost vstupní vrstvy je dána dimenzí vstupních dat a velikost výstupní vrstvy je za předpokladu klasifikačního problému určena počtem tříd.

Nechť má neuronová síť  $L$  vrstev, kde  $l = 1, 2, \dots, L$  označuje vrstvu a  $n^{(l)}$  označuje

počet neuronů v  $l$ -té vrstvě. Na vstup sítě je přiveden je vektor vstupních dat  $x$ . Aktivace neuronů ve vrstvě  $l$  jsou označeny jako vektor  $a^{(l)}$  [16].

Každý neuron ve vrstvě  $l$  zpracovává vstup  $a^{(l-1)}$  z předchozí vrstvy sítě pomocí lineární transformace:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \quad (2.1)$$

kde  $\mathbf{W}^{(l)}$  je matice vah neuronů mezi  $l - 1$  a  $l$  vrstvou,  $\mathbf{b}^{(l)}$  je bias vektor pro vrstvu  $l$  a  $\mathbf{a}^{(0)} = x$  [16].

Následně se na lineární transformaci aplikuje nelineární aktivační funkce  $f$  (obecně může být různá v každé vrstvě sítě). Výsledná rovnice aktivace neuronů v dané vrstvě vypadá následovně:

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) \quad (2.2)$$

Tento proces se opakuje pro každou vrstvu a výstup poslední vrstvy neuronů je výstupem celé sítě.

Nejčastěji používanými aktivačními funkcemi jsou ReLU (Rectified Linear Unit), sigmoid, tanh a softmax.

ReLU je definována jako [26]:

$$f_1(z) = \max(0, z) \quad (2.3)$$

Sigmoidní aktivační funkce má tvar [26]:

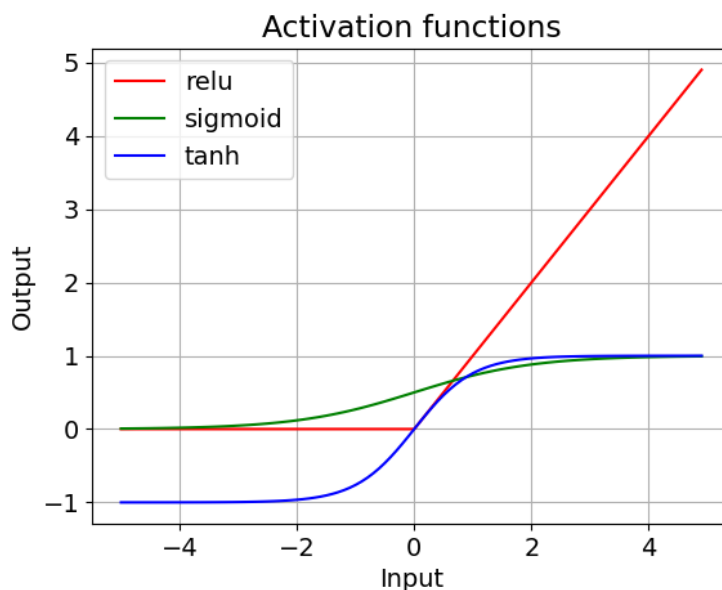
$$f_2(z) = \frac{1}{1 + e^{-z}} \quad (2.4)$$

Tanh aktivační funkce má tvar [26]:

$$f_3(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (2.5)$$

Funkce Softmax může být na rozdíl od sigmoidních funkcí, které se používají pro binární klasifikaci, použita pro problémy s klasifikací více tříd. Pokud má výstupní vrstva  $K$  neuronů, potom má Softmax funkce tvar [26]:

$$\mathbf{a}^{(L)}_j = \frac{e^{\mathbf{z}^{(L)}_j}}{\sum_{k=1}^K e^{\mathbf{z}^{(L)}_k}} \quad (2.6)$$



Obrázek 2.2: Aktivační funkce

Aktivační funkce ReLU, Sigmoid a Tanh jsou k vidění na Obrázku 2.2.

Cílem trénování neuronové sítě je nalézt takové hodnoty parametrů (vah synapsí  $\mathbf{W}$  a biasů neuronů  $\mathbf{b}$ ), aby síť byla schopna klasifikace nebo regrese vstupních dat s co největší správností. K porovnání cílové a předpovídané výstupní hodnoty neuronové sítě slouží tzv. ztrátová funkce (Loss Function), v rámci učení neuronové sítě je úkolem hodnotu této funkce minimalizovat.

Backpropagation je algoritmus pro učení s učitelem umělých neuronových sítí pomocí gradientního sestupu. Je využíván k výpočtu gradientu chybové funkce vzhledem k vahám umělé neuronové sítě, což umožňuje adaptovat váhy sítě tak, aby byla minimalizována chyba výstupu sítě. Tento postup umožňuje sítím adaptovat se na nová data a provádět přesné predikce. Algoritmus byl představen již v 60. letech, ale až téměř 30 let poté (v roce 1989) se stal populárním díky práci Rumelharta, Hinton a Williamse nazvané „Learning representations by back-propagating errors“ [24].

Algoritmus se skládá ze tří hlavních částí [33]:

- Dopředného šíření vstupního signálu
- Zpětného šíření chyby
- Aktualizace váhových hodnot neuronů



Tyto kroky jsou cyklicky opakovány, dokud není dosaženo dostatečně malé chyby sítě, mezního počtu iterací nebo jiného kritéria pro ukončení učení. Následuje podrobnější popis jednotlivých kroků.

Dopředné šíření (Forward Propagation):

Rovnice dopředného šíření odpovídá kombinaci Rovnic 2.1 a 2.2:

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) \quad (2.7)$$

Zpětné šíření (Backpropagation):

$C$  představuje tzv. ztrátovou funkci (Loss Function), označovanou také jako Cost function. Ztrátová funkce  $C$  může mít například tvar kvadratické funkce [16]:

$$C = \frac{1}{2} \sum_j (y_j - a_j^L)^2 \quad (2.8)$$

kde  $y_j$  je požadovaný výstup neuronu  $j$  a  $a_j^L$  je vypočtený výstup tohoto neuronu,  $L$  představuje počet vrstev v síti.

Obecně lze rozdělení ztrátových funkcí provést podle toho, zda se používají pro klasifikační nebo regresní problémy.

Klasifikační problémy mají za cíl predikovat diskrétní třídy nebo kategorie. Mezi ztrátové funkce vhodné pro klasifikační problémy patří například:

- *Categorical Cross-Entropy*: Tato funkce se používá pro klasifikační úlohy s více třídami. Její vzorec vypadá následovně [35]:

$$C_1 = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \log(p_{ij}) \quad (2.9)$$

kde  $N$  je počet datových vzorků,  $M$  je počet tříd,  $y_{ij}$  je indikátor, zda je  $i$ -tý vzorek ve třídě  $j$ , a  $p_{ij}$  je pravděpodobnost, že  $i$ -tý vzorek patří do třídy  $j$ .

- *Binary Cross-Entropy*: Jendá se o speciální případ Categorical Cross-Entropy pro úlohy, kde je počet tříd omezen na dvě. Lze ji popsat takto [35]:

$$C_2 = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (2.10)$$

kde  $N$  je počet datových vzorků,  $y_i$  označuje skutečnou třídu vzorku  $i$ , která může nabývat hodnoty 0 nebo 1,  $p_i$  je pravděpodobnost, že  $i$ -tý vzorek patří do třídy 1 (pravděpodobnost, že vzorek patří do třídy 0, je rovna  $1 - p_i$ ).

Regresní problémy mají za cíl predikovat spojité hodnoty. Mezi ztrátové funkce vhodné pro regresní problémy patří například:

- *Mean Squared Error (MSE)*: Funkce je citlivá na outliery. Používá se, pokud jsou cílová data normálně rozdělena kolem střední hodnoty, a když je důležité více penalizovat odlehlé hodnoty. Vypadá následovně [8]:

$$C_3 = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.11)$$

kde  $N$  je počet datových vzorků,  $y_i$  je skutečná hodnota a  $\hat{y}_i$  je predikovaná hodnota.

- *Mean Absolute Error (MAE)*: Tato funkce není citlivá na outliery. Nevýhodou MAE je, že velikost gradientu není závislá na velikosti chyby, pouze na znaménku  $(y_i - \hat{y}_i)$ , což může vést k problémům s konvergencí, protože velikost gradientu bude velká i při malé chybě. MAE vzorec je [8]:

$$C_4 = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (2.12)$$

kde  $N$  je počet datových vzorků,  $y_i$  je skutečná hodnota a  $\hat{y}_i$  je predikovaná hodnota.

Zpětné šíření se zaměřuje na to, jak změna vah a biasů v síti ovlivňuje hodnotu ztrátové funkce. Dále jsou tedy uvedeny parciální derivace  $C$  vzhledem k váhám a biasům sítě. Pro neuron  $j$  ve vrstvě  $l$  jsou tyto derivace následující [16]:

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \quad (2.13)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (2.14)$$

kde  $\delta_j^l$  je tzv. chyba neuronu  $j$  ve vrstvě  $l$ , která se vypočítá jako [16]:

$$\delta_j^l = f'(z_j^l) \sum_k w_{jk}^{l+1} \delta_k^{l+1} \quad (2.15)$$

kde  $f'(\cdot)$  je derivace aktivační funkce neuronu  $j$  ve vrstvě  $l$ ,  $w_{jk}^{l+1}$  je váha mezi neuronem  $j$  ve vrstvě  $l$  a neuronem  $k$  ve vrstvě  $l + 1$  a  $z_j^l$  je vážená suma vstupů neuronu  $j$  ve vrstvě  $l$  [16]:

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \quad (2.16)$$

Aktualizace vah a biasů:

Po výpočtu gradientu chyby se váhy a biasy sítě aktualizují pomocí gradientního sestupu. Nová hodnota váhy  $w_{jk}^l$  a biasu  $b_j^l$  je vypočtena jako [37]:

$$w_{jk}^l \leftarrow w_{jk}^l - \eta \frac{\partial C}{\partial w_{jk}^l} \quad (2.17)$$

$$b_j^l \leftarrow b_j^l - \eta \frac{\partial C}{\partial b_j^l} \quad (2.18)$$

kde  $\eta$  je koeficient učení (Learning Rate), což je nastavitelný hyperparametr optimalizačního algoritmu, který určuje velikost kroku, kterým se algoritmus posouvá směrem k minimu ztrátové funkce během trénování [15]. Obecně platí, že příliš nízká hodnota  $\eta$  může vést k pomalé konvergenci a příliš vysoká hodnota může vést k oscilaci a neefektivnímu trénování modelu.

Tyto kroky jsou cyklicky opakovány, dokud není dosaženo dostatečně malé chyby sítě, mezního počtu iterací nebo jiného kritéria pro ukončení učení [33].

## 2.1.1 Hyperparametry

Hyperparametry neuronových sítí jsou proměnné, které určují strukturu sítě a způsob, jakým je síť trénována. Tyto hyperparametry jsou nastaveny před trénováním sítě, tedy před optimalizací vah a biasu [20].

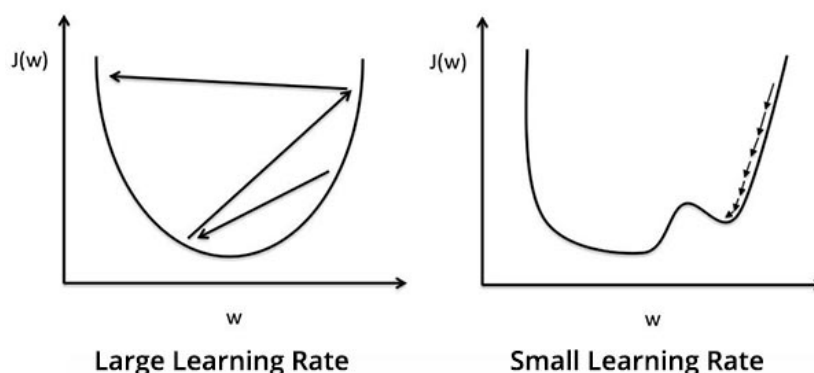
Hyperparametrické ladění je proces hledání optimálních hyperparametrů pro danou úlohu (např. maximalizace přesnosti modelu na validačním datasetu). Tento proces



je často prováděn pomocí opakovaného trénování modelu s různými hodnotami hyperparametrů a vyhodnocováním výsledků pomocí validační sady dat.

Následuje výběr nastavitelných hyperparametrů:

- *Počet skrytých vrstev*: Počet skrytých vrstev představuje počet vrstev mezi vstupní a výstupní vrstvou. Vyšší počet vrstev může způsobit delší čas trénování.
- *Počet neuronů ve skrytých vrstvách*: Jedná se o počet neuronů pro každou skrytou vrstvu, může být obecně různý pro jednotlivé vrstvy. Počet neuronů by měl být přizpůsoben komplexitě řešené úlohy [23]. Zvýšení počtu neuronů může zlepšit výkon sítě, ale zároveň zvyšuje náročnost trénování sítě.
- *Aktivační funkce*: Aktivační funkce je funkce, která je aplikována na výstup každého neuronu v neuronové síti. Slouží k zavedení nelinearity do modelů [20].
- *Koeficient učení (Learning Rate)*: Koeficient učení definuje rychlost aktualizace parametrů neuronové sítě během trénování. Nižší hodnota zpomaluje proces učení, ale zpravidla vede k hladší konvergenci. Naopak vyšší hodnota zrychluje učení, ale může mít za následek, že síť se nebude schopna dostat k optimálnímu řešení a bude oscilovat kolem něj [20]. Na Obrázku 2.3 je ilustrován průběh ztrátové funkce  $J$  v závislosti na hodnotě vah  $w$  vlevo pro velký, vpravo pro malý koeficient učení. Navíc je vlevo zobrazena i možnost uvíznutí v lokálním minimu při malém učícím koeficientu.



Obrázek 2.3: Vliv učícího koeficientu při gradientním sestupu (převzato z [2])

- *Momentum*: Tato metoda pomáhá řešit výše zmíněný problém uvíznutí v lokálním minimu. Zahrnuje informace o minulých změnách parametrů při

aktualizaci nových. Rovnice 2.17 a 2.18 s využitím této metody vypadají následovně [29]:

$$v_{jk}^l \leftarrow \mu v_{jk}^l - \eta \frac{\partial C}{\partial w_{jk}^l} \quad (2.19)$$

$$w_{jk}^l \leftarrow w_{jk}^l + v_{jk}^l \quad (2.20)$$

$$v_j^l \leftarrow \mu v_j^l - \eta \frac{\partial C}{\partial b_j^l} \quad (2.21)$$

$$b_j^l \leftarrow b_j^l + v_j^l \quad (2.22)$$

kde  $v_{jk}^l$  a  $v_j^l$  jsou momenty pro váhy a biasy,  $\mu$  je momentový koeficient (typicky se používají hodnoty 0.9 nebo 0.99 [4]). Výhodou této metody je rychlejší konvergence k optimálnímu řešení a menší pravděpodobnost uvíznutí v lokálních minimech.

- *Batch Size*: Batch size je počet vzorků, které jsou síti předkládány současně v rámci jedné iterace aktualizace parametrů [20]. Batch size ovlivňuje rychlost trénování sítě, větší batch size obvykle může zvýšit rychlost trénování.
- *Počet epoch*: Počet epoch značí, kolikrát je celý trénovací dataset předložen neuronové síti během tréninku [20]. Příliš málo epoch může vést k nedotrénování sítě, zatímco příliš mnoho epoch může vést k přeučení sítě [23].
- *Optimalizační algoritmus*: Optimizér je algoritmus používaný k úpravě vlastností neuronové sítě, jako jsou váhy a koeficient učení, s cílem snížit chybovost [7]. Existuje mnoho druhů optimizérů v oblasti strojového učení, některé z nejznámějších jsou:

– **Gradientní sestup**: Jedná se o nejprimitivnější typ optimizéru. Jeho výhodou je jednoduchost, rychlost a snadná implementace. Na druhou stranu ale může uváznout v lokálním minimu, může být pomalý, pokud je dataset velký a má mnoho příznaků, a má vysoké paměťové nároky [7].

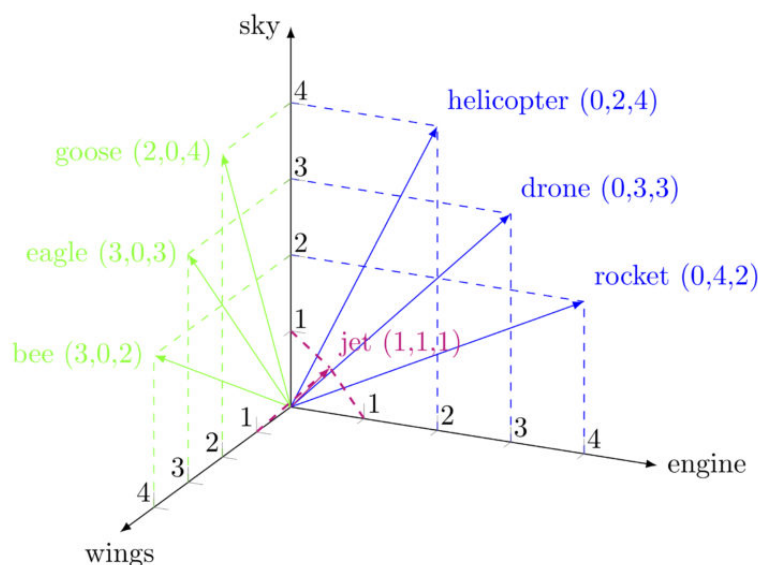
– **Stochastický gradientní sestup (SGD)**: Jedná se o variantu gradientního sestupu, která se snaží o častější aktualizaci parametrů. Mezi jeho výhody patří právě častá aktualizace parametrů a tím rychlejší konvergence, dále má nižší paměťové nároky. Nevýhodou je vysoká variabilita parametrů modelu a nutnost snižování učícího koeficientu pro dosažení stejné konvergence jako u gradientního sestupu. Algoritmus může být navíc nestabilní [7].

- *Adagrad*: Tento optimizér, narozdíl od všech výše zmíněných, mění koeficient učení pro každý parametr a v každém časovém kroku, odpadá tedy nutnost ručního ladění. Tento algoritmus je vhodný pro datasety s chybějícími vzorky. Je však výpočetně náročný a snižující se koeficient učení má za následek pomalé trénování [7].
- *Adadelta*: Jendá se o rozšíření metody Adagrad. Adadelta omezuje počet předchozích gradientů, které jsou započítávány, na pevnou velikost za pomoci klouzavého exponenciálního průměru a tím odstraňuje problém spojený se snižujícím se učícím koeficientem. Nevýhodou je opět výpočetní náročnost [7].
- *Root Mean Square Propagation (RMS-Prop)*: RMS-Prop je podobný Adagrad, rozdíl spočívá v použití exponenciálně klesajícího průměru místo součtu gradientů. V podstatě tedy kombinuje Momentum s AdaGradem. Kromě toho místo použití všech gradientů pro výpočet momenta, zahrnuje pouze nejnovější gradienty. To modelu umožňuje adaptaci koeficientu učení vzhledem k aktuální situaci. Nevýhodou je, že učení je pomalé [25].
- *Adaptive Moment Estimation (Adam)*: Adam optimizer lze považovat za kombinaci AdaGrad a RMS-Prop. Na rozdíl od RMSProp používá Adam průměr druhých momentů gradientů. Jedná se o nejvíce používaný optimalizační algoritmus pro řešení široké škály problémů, je vhodný pro velké datasety a je výpočetně efektivní. Výkon algoritmu Adam závisí na typu poskytnutých dat a jedná se o kompromis mezi rychlostí a generalizací [25].

## 2.2 Sentence transformers

Sentence Transformer je model strojového učení, který dokáže transformovat vstupní textové sekvence (např. věty) do vektorů reprezentujících sémantickou informaci. Tyto vektory pak mohou být využity například pro kategorizaci, porovnávání podobnosti nebo další úlohy zpracování přirozeného jazyka. Ilustrační vektorový prostor sedmi slov je na Obrázku 2.4.

Architektura zvaná transformer byla poprvé představena v roce 2017 v článku Attention Is All You Need. Tato architektura od svého nástupu překonávala do té doby tradiční rekurentní a konvoluční neuronové sítě postupně prakticky ve všech oblastech strojového učení. Modely dosahují lepší kvality, jsou více paralelizovatelné a vyžadují podstatně méně času na trénování. Klíčovou součástí transformeru je mechanismus pozornosti (Attention), který umožňuje modelu se



Obrázek 2.4: Vektorový prostor sedmi slov ve třech kontextech (převzato z [5])

zaměřit na důležité části vstupní sekvence a ignorovat méně významné informace

[32]

Vzorec pro výpočet **pozornosti** (Scaled Dot-Product Attention) vypadá následovně [32]:

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


kde:

- $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$  je matice dotazů (Query Matrix)
- $\mathbf{K} \in \mathbb{R}^{m \times d_k}$  je matice klíčů (Key Matrix)
- $\mathbf{V} \in \mathbb{R}^{m \times d_v}$  je matice hodnot (Value Matrix)
- $d_k$  je počet dimenzí vektoru klíče (Key Vector)
- $n$  a  $m$  jsou počty dotazů a klíčů/hodnot
- $\sqrt{d_k}$  slouží k normalizaci skalárního součinu a zlepšuje stabilitu gradientů v tré-nování


Multi-head attention lze chápat jako několik paralelních mechanismů pozornosti pracujících společně. Místo toho, aby byl použit jeden set vah pro výpočet



pozornosti, je použito více setů vah, tzv. heads. Každý head slouží jako zvláštní lineární transformace na vstupním prostoru a výsledky jsou následně kombinovány. Použitím více attention heads **se umožňuje reprezentace několika souborů vztahů současně** [3].

Hlavním cílem Sentence Transformeru je naučit se reprezentovat věty tak, aby výsledné vektory zachovávaly sémantickou podobnost mezi větami. To umožňuje využití těchto vektorů pro různé úlohy, jako například hledání podobností mezi větami, kategorizaci textů, vyhledávání odpovědí atd. Pro výpočet vektorové reprezentace vět se často používají předtrénované modely, jako  BERT nebo GPT, které jsou široce dostupné pro použití v různých aplikacích.

## 2.2.1 FERNET-C5

FERNET-C5 [12] je jednojazyčný BERT model, který byl od úplného počátku trénován na datech českého korpusu Colossal Clean Crawled Corpus (C5) – obdoba anglického datasetu C4 pro češtinu. Trénovací data obsahují téměř 13 miliard slov. Model má stejnou architekturu jako původní BERT model, tedy 12 transformačních bloků, 12 pozornostních hlav (Attention Heads) a skrytou velikost 768 neuronů. Na rozdíl od BERT modelů od Googlu byla použita tokenizace SentencePiece místo interní  nizace WordPiece od Googlu.

Model je veřejně k dispozici online a umožňuje integraci s různými aplikacemi, jako jsou chatboty, analýza sentimentu, nebo klasifikace textu.

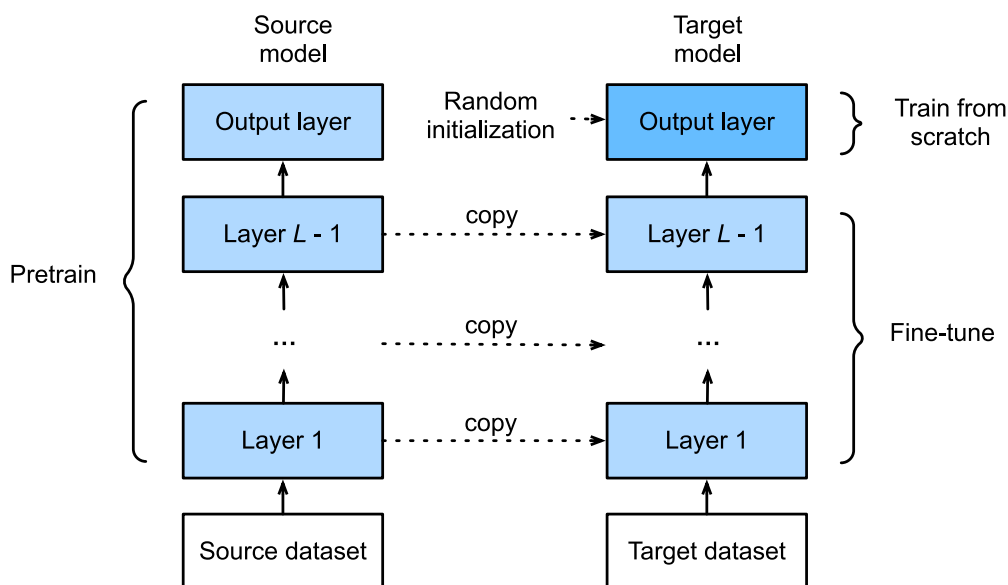
## 2.3 Fine-tuning a Transfer learning

Fine-tuning a transfer learning neuronových sítí jsou procesy přizpůsobení již natrénované sítě na nový úkol, který může být odlišný od původního úkolu, pro který byla síť trénována. Tyto procesy spočívají v použití vah již natrénované sítě jako počátečních hodnot pro trénování nové sítě [31]. Při použití fine-tuningu se předtrénovaný model přizpůsobuje nové úloze tím, že se upraví váhy neuronové sítě pomocí trénovacích dat specifických pro novou úlohu.

Fine-tuning začíná kopírováním (transferem) vah z již předtrénované sítě do nové sítě, která má být trénována. V případě klasifikační úlohy je často výjimkou poslední plně propojená vrstva, jejíž počet neuronů závisí na počtu tříd v původním datasetu. Běžnou praxí je nahradit poslední plně propojenou vrstvu předtrénovaného neuronového modelu novou plně propojenou vrstvou, která má stejný počet neuronů jako počet tříd v nové cílové aplikaci [31]. Její inicializace je zpravidla náhodná [36]. Nakonec je nový model trénován na cílovém datasetu.

Výstupní vrstva je trénována od začátku, zatímco parametry všech ostatních vrstev jsou vyladěny na základě parametrů zdrojového modelu [36]. Obecně lze provádět i fine-tuning čistě T5 transformer sítě, kde se zachovává kompletní architektura a jen se provede fine-tuning na nových doménových datech k naučení nové úlohy.

Nástin fungování fine-tuningu při nahrazení poslední plně propojené vrstvy je znázorněn na Obrázku 2.5.



Obrázek 2.5: Diagram fine-tuningu (převzato z [36])

Při transfer learningu se postupuje obdobně jako u fine-tuningu s tím rozdílem, že přenesené váhy jsou „zmrazeny“. Zmrazení vah znamená, že se během tréninku nové sítě váhy přenesené z původní sítě neaktualizují. Trénovány jsou pouze váhy nově přidaných vrstev, což snižuje množství dat potřebných k trénování a zkracuje dobu trénování nové sítě.

Fine-tuning a transfer learning jsou velmi užitečné nástroje, protože umožňují využít existující znalosti a zkušenosti z předtrénovaných modelů, které mohou být velmi složité a nákladné na trénování, a aplikovat je na nové úlohy s menším počtem trénovacích dat. To může být velmi užitečné, například pokud není k dispozici dostatečné množství dat pro natrénování nového modelu od začátku.

## 2.4 ROS (Robot Operating System)

ROS (Robot Operating System) je open-source softwarový framework určený pro vývoj robotických aplikací. Poprvé byl uveden v roce 2007 na Univerzitě v Stanfordu.

Poskytuje knihovny a nástroje umožňující snadno vytvářet a sdílet software pro řízení robotů [1, 18].

ROS byl vyvíjen veřejně s použitím permissivní licence BSD a postupně se stal široce používaným v komunitě výzkumníků robotiky. Na rozdíl od klasického přístupu, kdy všichni přispěvatelé umístí svůj kód na stejné servery, byl ROS vyvíjen v několika institucích a pro různé druhy robotů. Toto se stalo jednou z největších výhod ROS ekosystému. Současně je používán desítkami uživatelů po celém světě v oblastech od domácích projektů v rámci koníčků, po velké průmyslové automatizované systémy [18]. V současné době je ROS stále aktivně vyvíjen a jeho verze 2.0, známá jako ROS 2, je určena pro použití v průmyslových aplikacích s většími nároky na spolehlivost a bezpečnost [10].

Základními koncepty implementace ROS jsou uzly (Nodes), zprávy (Messages), témata (Topics) a služby (Services). Uzly jsou procesy provádějící výpočty, přičemž systém obvykle sestává z mnoha uzlů. Termín uzel je v tomto kontextu zaměnitelný se softwarovým modulem. Použití termínu uzel vzniká z vizualizací systémů pomocí grafu, kde jsou procesy zobrazeny jako uzly grafu a peer-to-peer spojení jako orientované hrany (Arcs) [18, 19].

Uzly mezi sebou komunikují předáváním zpráv. Zpráva je striktně typovaná datová struktura. Podporovány jsou jak standardní primitivní datové typy (celé číslo, desetinné číslo, boolean, atd.), tak pole primitivních datových typů. Zprávy mohou být složeny z jiných zpráv a polí jiných zpráv, v libovolné hloubce vnoření [18, 19].

Uzel odesílá zprávy publikováním na dané téma. Uzel, který má zájem o určitý druh dat, se přihlásí k příslušnému tématu. Pro jedno téma může existovat několik souběžných vydavatelů a odběratelů a jeden uzel může publikovat a/nebo odebírat více témat. Vydavatelé a odběratelé většinou nevědí o existenci druhého. Idea publikování a odběru témat v ROS umožňuje komunikaci mezi moduly na velmi flexibilní úrovni. Broadcast směřování však není vhodné pro synchronní transakce. Proto ROS poskytuje tzv. service (službu), která je definována názvem a dvěma přísně typovanými zprávami: jednou pro požadavek a druhou pro odpověď [18, 19].


Service ale nejsou vhodné pro úlohy, které mohou trvat dlouho, nebo pro situace, kdy je potřeba sledovat stav celého procesu. K tomuto účelu slouží balíček actionlib, který umožňuje uživateli během vykonávání požadavku jeho zrušení nebo získání zpětné vazby o postupu vykonávání požadavku. Actionlib poskytuje nástroje pro vytváření serverů vykonávajících dlouhodobé úkoly, které mohou být přerušeny. Kromě toho poskytuje klientské rozhraní pro odesílání požadavků na server [28].

Filozofické cíle ROS lze shrnout následovně:

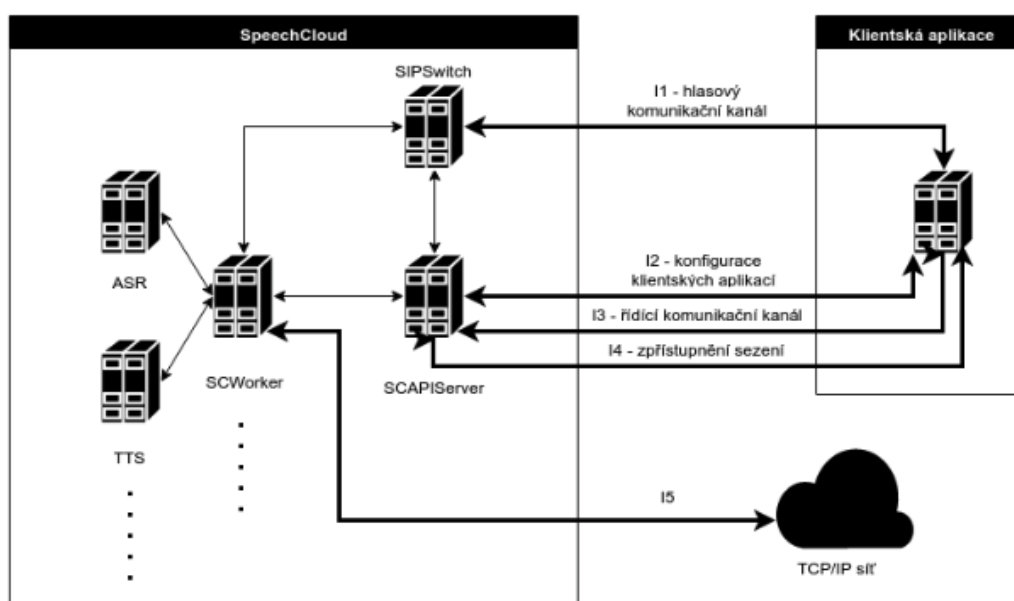
- *Peer-to-peer*: ROS by měl být navržen tak, aby jednotlivé uzly mohly komunikovat přímo mezi sebou, aniž by bylo nutné centralizované řízení, tj. aby byl decentralizovaný [18, 19].
- *Založení na nástrojích (Tools-based)*: ROS poskytuje mnoho nástrojů, jako jsou nástroje pro získávání a nastavování konfiguračních parametrů, vizualizaci topologie připojení peer-to-peer, měření využití šířky pásma, grafické vykreslování data zpráv, automatické generování dokumentace atd. Avšak nemá kanonické integrované vývojové a runtime prostředí, všechny úkoly jsou prováděny samostatnými programy, což podporuje vytváření nových, vylepšených implementací [18, 19].
- *Mnohojazyčnost (Multi-lingual)*: Softwarové moduly ROS lze psát v jakémkoli programovacím jazyce, pro který byla napsána klientská knihovna. Tyto moduly mezi sebou komunikují díky jazykově neutrálnímu a jednoduchému jazyku pro definici rozhraní (IDL), které slouží k popisu zpráv odesílaných mezi moduly [18, 19].
- *Thin*: Konvence ROS vybízí přispěvatele k vytváření samostatných knihoven. Tyto knihovny jsou následně zabaleny a umožňují komunikaci pomocí zpráv s jinými ROS moduly. Tato další vrstva umožňuje opětovné použití softwaru mimo ROS pro jiné aplikace [18, 19].
- *Zdarma a Open-Source*: Úplný zdrojový kód ROS je veřejně dostupný [18, 19].

## 2.5 Speechcloud

SpeechCloud je platforma pro zpracování řeči a analýzu hlasu vyvinutá společností SpeechTech a Katedrou kybernetiky na ZČU. Umožňuje například automatický přepis diktátu do psané podoby, generování **vysoce přirozené** řeči ze zadaných textů, **ověření a verifikaci** osob na základě jedinečných charakteristik jejich hlasu a hlasovou komunikaci mezi člověkem a počítačem. Platforma podporuje několik jazyků; češtinu, slovenštinu, angličtinu a němčinu. SpeechCloud je navržen tak, aby umožňoval snadné integrování s různými aplikacemi a systémy třetích stran. [27]. Architektura této platformy je naznačena na Obrázku 2.6.

Architektura SpeechCloudu je založena na Docker images a umožňuje využít výpočetní výkon velkého počítačového clusteru. Klient SpeechCloudu komunikuje s API serverem a spouští sezení pomocí specifické URL s požadovanými technologiemi. API server poté přidělí konkrétního pracovníka, který zpracovává zvuk a přenáší ho pomocí FreeSwitch mezi klientem a pracovníkem. Kromě zvuku jsou přenášeny i řídicí zprávy pomocí WebSocket připojení **jako JSON** **zako****de**





Obrázek 2.6: Diagram architektury SpeechCloudu (převzato z [30])

zprávy. Pracovníci mají také sadu nástrojů pro zpracování výstupů rozpoznávání řeči, včetně algoritmu pro detekci sémantických entit výstupu rozpoznávání. Kromě toho existuje nástroj pro údržbu modelů a automatickou interpolaci specifických jazykových modelů pro konkrétní dialogy [30].

Základní SpeechCloud technologie lze označit jako:

- Automatické rozpoznávání řeči (ASR) – SpeechCloud využívá pokročilé ASR technologie, které umožňují převádět mluvenou řeč na text. Tato technologie používá strojové učení a hluboké neuronové sítě, aby dosáhla vysoké přesnosti a robustnosti při rozpoznávání řeči.
- Text-to-speech (TTS) – SpeechCloud také umožňuje generovat řeč z textu pomocí TTS technologie. Podporováno je několik hlasových stylů.

## 2.6 Komunikační protokoly

Komunikační protokoly jsou pravidla a postupy, které slouží k výměně informací mezi dvěma nebo více zařízeními v počítačové síti. Tyto protokoly definují, jakým způsobem jsou data přenášena, jak jsou zabezpečena a jakým způsobem jsou přijímána a zpracovávána. Existuje mnoho různých komunikačních protokolů, které se liší podle účelu a specifikace použití. V této práci jsou využity následující: HTTP, MQTT a WebSockets.

## 2.6.1 HTTP

HTTP (Hypertext Transfer Protocol) je protokol určený pro přenos hypertextových dokumentů na internetu. Jedná se o bezstavový protokol, tj. každý požadavek je zpracováván odděleně a nezávisle na předchozích požadavcích. HTTP funguje na aplikační vrstvě v TCP/IP síťové architektuře a využívá standardní port 80 pro komunikaci. HTTP využívá model klient-server, kde klient pošle požadavek na server, který odpovídá příslušnou odpovědí.

HTTP požadavek se skládá z několika částí, včetně řádku požadavku, hlaviček a těla. Řádek požadavku obsahuje metodu požadavku (GET, POST, PUT, DELETE apod.), adresu URL, která má být požadována, a verzi protokolu HTTP, která má být použita. Hlavičky obsahují další informace o požadavku, jako jsou např. informace o klientovi a typy dat, které server může poskytnout. Tělo obsahuje samotná data, pokud jsou součástí požadavku.

HTTP odpověď se skládá také z několika částí, včetně řádku s kódem odpovědi, hlaviček a těla. Kódy odpovědí jsou standardizované a určují výsledek požadavku, zda byl úspěšně zpracován, nebo zda došlo k nějaké chybě apod. Hlavičky obsahují další informace o odpovědi, jako jsou např. informace o serveru a typy dat, které jsou součástí odpovědi. Tělo obsahuje samotná data, která jsou součástí odpovědi.

## 2.6.2 MQTT

MQTT (Message Queuing Telemetry Transport) je protokol pro komunikaci mezi zařízeními přes síť, který je navržen pro efektivní a spolehlivou výměnu zpráv v síťových prostředích s omezenými prostředky.

Protokol MQTT je založen na publish/subscribe architektuře, kde zařízení publikují zprávy na témata (Topics) a jiná zařízení mohou tyto zprávy odebírat ze stejných témat. Témata jsou řetězce textu, které umožňují organizovat zprávy do hierarchické struktury.

MQTT je navržen tak, aby byl velmi jednoduchý a lehký na použití, což znamená, že může být využíván i na zařízeních s omezenými výpočetními a paměťovými kapacitami. Díky této vlastnosti je MQTT ideální pro komunikaci mezi IoT zařízeními.

Další vlastností MQTT je zajištění kvality doručení zpráv. Zařízení mohou publikovat zprávy s různými úrovněmi kvality doručení (QoS), které ovlivňují spolehlivost přenosu.

## 2.6.3 WebSockets

WebSockets je komunikační protokol umožňující dvěma stranám – klientovi a serveru – navázat interaktivní a duplexní komunikaci v reálném čase. Jedním z hlavních rozdílů oproti klasickému HTTP protokolu je, že WebSockets nevyžadují, aby klient neustále dotazoval server na nová data. Místo toho umožňují otevřít trvalé spojení mezi klientem a serverem, které umožňuje okamžité posílání zpráv mezi oběma stranami.

WebSockets využívá standardní TCP protokol pro vytvoření spojení mezi klientem a serverem. Po navázání spojení mohou obě strany odesílat zprávy, přičemž každá zpráva je tvořena hlavičkou a tělem. Hlavička obsahuje informace o zprávě, jako je například typ zprávy a délka těla. Tělo zprávy obsahuje samotná data, která se mají přenést.

WebSockets je používán pro různé typy aplikací, například pro online chatování, online hry, real-time sledování dat a mnoho dalších. Díky rychlosti a nízké latenci jsou WebSockets obvykle preferovanou volbou pro aplikace, které potřebují rychlou a spolehlivou duplexní komunikaci.

## 2.7 Základní webové technologie

Základní webové technologie jsou klíčovými stavebními kameny pro vytváření webových stránek, umožňují tvůrcům stránek vytvářet komplexní a interaktivní obsah, který je přizpůsobený různým zařízením a uživatelům. V této práci byly pro tvorbu uživatelského rozhraní využity HTML, CSS a JavaScript.

HTML (*HyperText Markup Language*) je standardní značkovací jazyk používaný k tvorbě struktury a obsahu webových stránek. Pojem *HyperText* vyjadřuje možnost vzájemně propojovat texty pomocí odkazů, *Markup* označuje schopnost jazyka HTML dávat významy jednotlivým blokům textu s pomocí speciálních značek tzv. elementů či tagů. Mezi základní tagy patří např. `<html>`, `<head>`, `<body>`, `<div>`, `<p>`, `<img>` a `<a>`.

CSS (*Cascading Style Sheets*) je deklarativní jazyk používaný pro popis vizuální prezentace a formátování webových stránek psaných ve značkovacích jazycích. Umožňuje vytvářet vzhled jako druhou, na obsahu nezávislou vrstvu a různě ho měnit podle aktuálního kontextu.


JavaScript je objektově orientovaný, událostmi řízený programovací jazyk používaný k tvorbě interaktivního a dynamického obsahu na webových stránkách. Umožňuje tvůrcům stránek reagovat na uživatelské akce, jako je například kliknutí na tlačítko nebo odeslání formuláře.



# Učení neuronové sítě hlasovým dialogem

## 3

### 3.1 Neuronová síť pro klasifikaci záměru

 Neuronová síť pro klasifikaci záměru (Intent Classifier) je dopředná neuronová síť, viz 2.1, která se používá pro automatickou klasifikaci krátkých textů do předem definovaných kategorií (záměrů). Pro trénování takovéto sítě se používá dataset, který obsahuje příklady textů a jim odpovídající záměry. Vstupem do této sítě ale není samotný text, nýbrž jeho vektorová reprezentace. Po natrénování neuronové sítě ji lze použít pro klasifikaci záměru pro nové texty, které nebyly v trénovacím datasetu.


Pro vytvoření vektorových reprezentací textů se používá technologie zvaná sentence embedding, konkrétně Sentence Transformer, viz 2.2. Tato technologie umožňuje převádět texty na vektorové reprezentace zachycující sémantické vlastnosti.

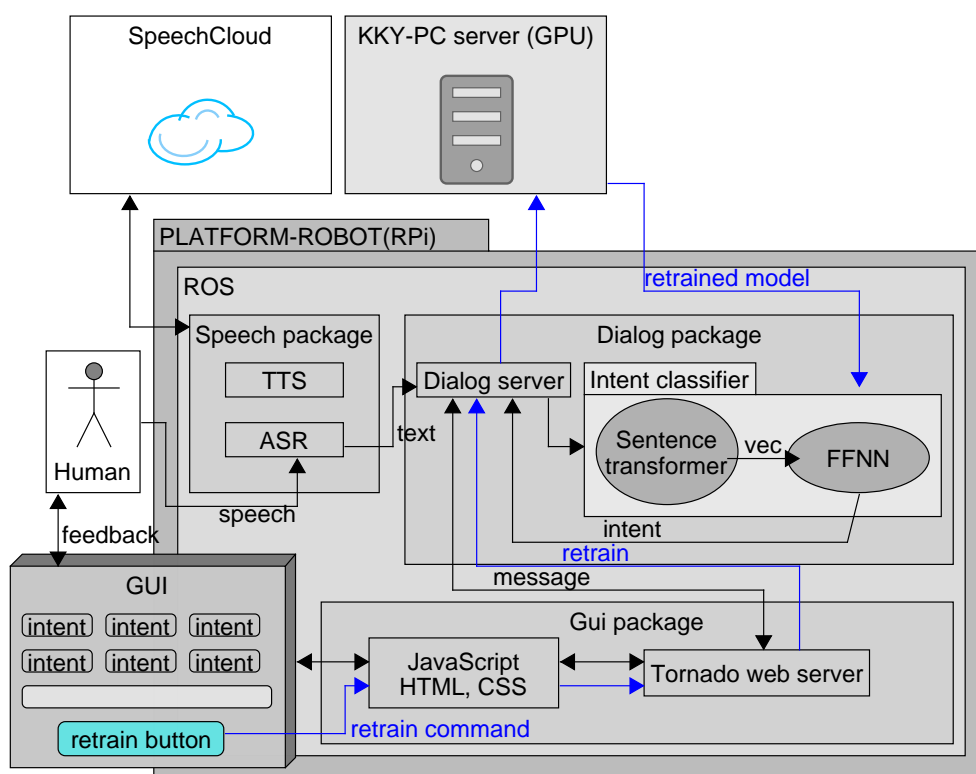
Ilustrativní napojení Sentence Transformeru FERNET-C5 (viz 2.2.1) na dopřednou neuronovou síť pro klasifikaci záměru je na Obrázku 3.1. Textový uživatelský vstup je pomocí transformeru FERNET-C5 převeden na vektor délky 768. Tento vektor pak slouží jako vstup do neuronové sítě, jejíž výstupem je záměr textu uživatele.






- GUI package – Zde běží Tornado webový server, který komunikuje s dialogovým balíčkem pomocí zpráv a s webovou stránkou (JavaScriptem) pomocí webových stránek.

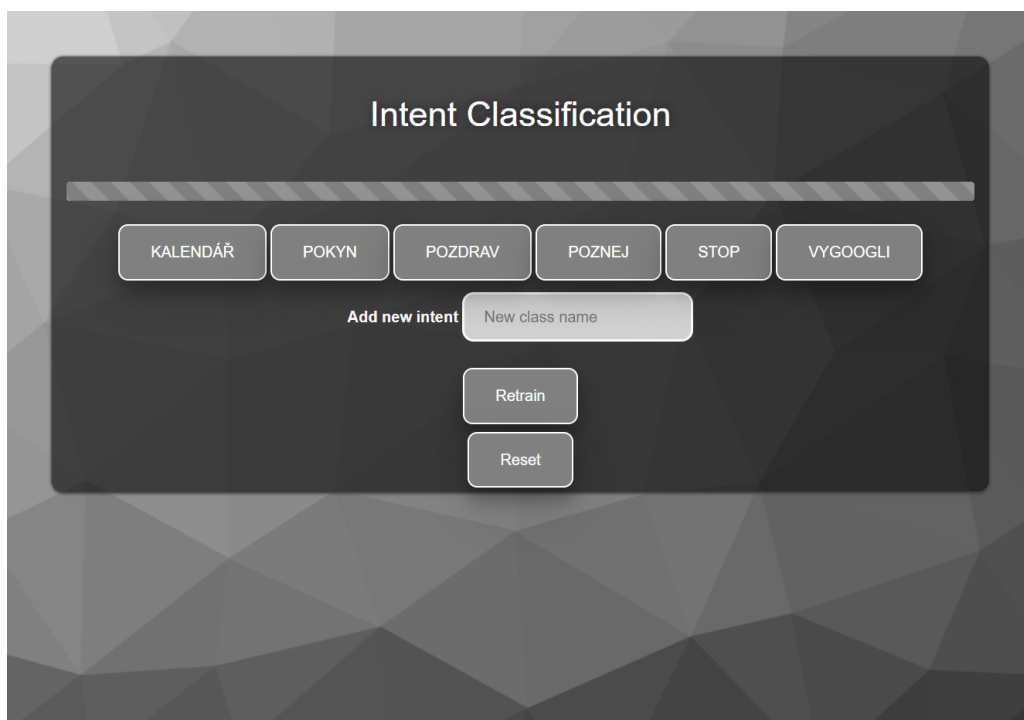
Hlasový vstup uživatele je převeden pomocí automatického rozpoznávání řeči do textové podoby. Tento text je předán jako vstup pro predikci neuronové sítě pro klasifikaci záměru (podrobněji o její struktuře viz 3.1), jejím výstupem je predikovaný záměr uživatele. Ten je pomocí zprávy zaslán do webového serveru a **příslušně** obrazen na webové stránce. Uživatel má následně možnost na tuto predikci adekvátně reagovat – potvrdit správnost nebo provést korekci. Tato zpětná vazba je zaslána webovému serveru a pomocí zprávy následně dialogovému serveru, který si ji ukládá. Uživatel má kromě zpětné vazby na určený záměr možnost dát pokyn pro přeučení celé neuronové sítě na základě předchozích zpětných vazeb. V tomto případě se analogicky jako u zpětné vazby dostane tato informace k dialogovému serveru. Ten v tomto případě však navíc zasílá HTTP požadavek stolnímu počítači pro přeučení modelu. Tato situace je v diagramu znázorněna modrou barvou.



Obrázek 3.2: Nastínění propojení jednotlivých částí v rámci celé aplikace

## 3.3 Uživatelské rozhraní

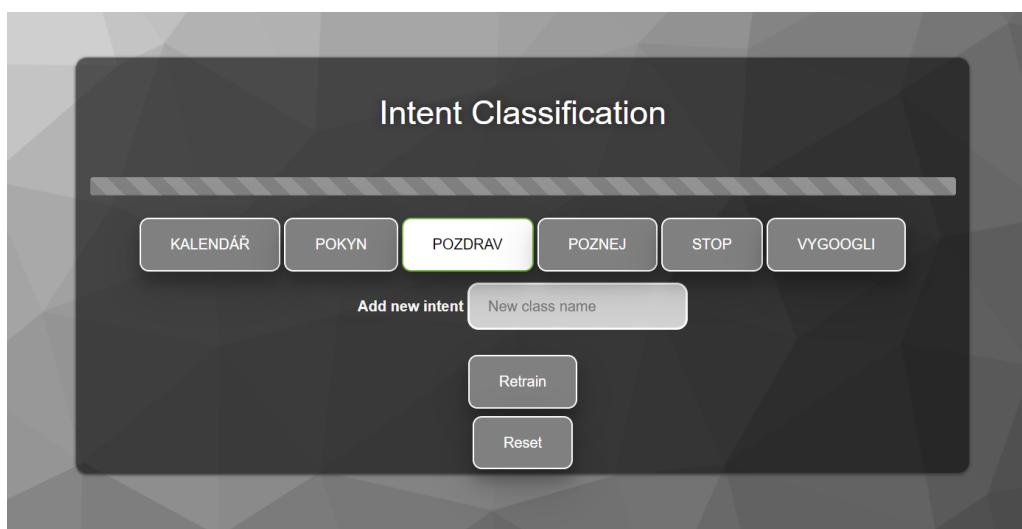
Samotné uživatelské rozhraní je zobrazeno  zeno **jednak** na displeji robotické entity, **jednak** ve webovém prohlížeči na adrese robot.local:7010. Nyní bude nastíněna funkčnost uživatelského rozhraní. Na Obrázku 3.3 je základní vzhled webové stránky v prohlížeči.



Obrázek 3.3: Celkový náhled na uživatelské rozhraní

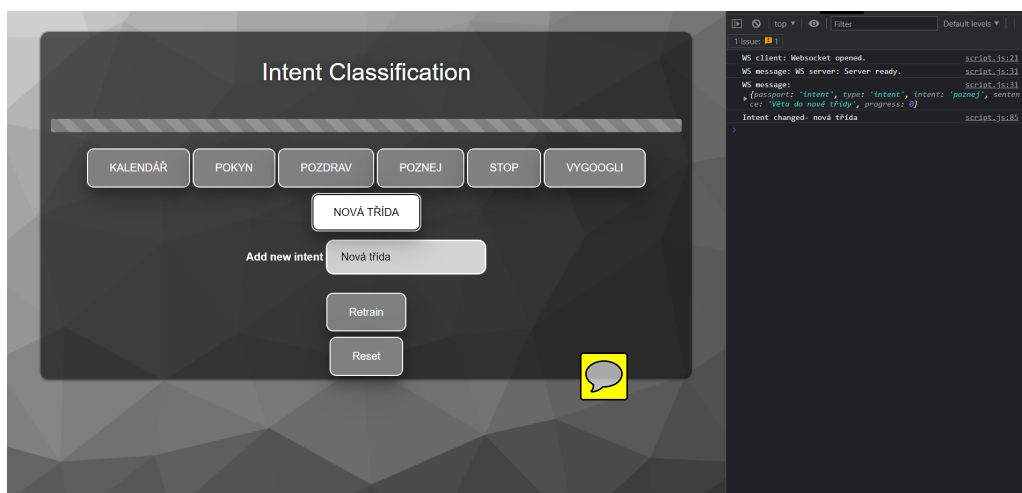
Obrázek 3.4 ilustruje situaci, kdy byl rozpoznán hlasový vstup uživatele a byl predikován jeho záměr. V této ilustrativní situaci byl rozpoznáný text „Ahoj, robote“, klasifikátor záměru predikoval, že záměr tohoto textu je „pozdrav“, v uživatelském rozhraní je proto tlačítko s touto třídou **orámováno zelenou barvou.**





Obrázek 3.4: Zvýraznění predikovaného záměru

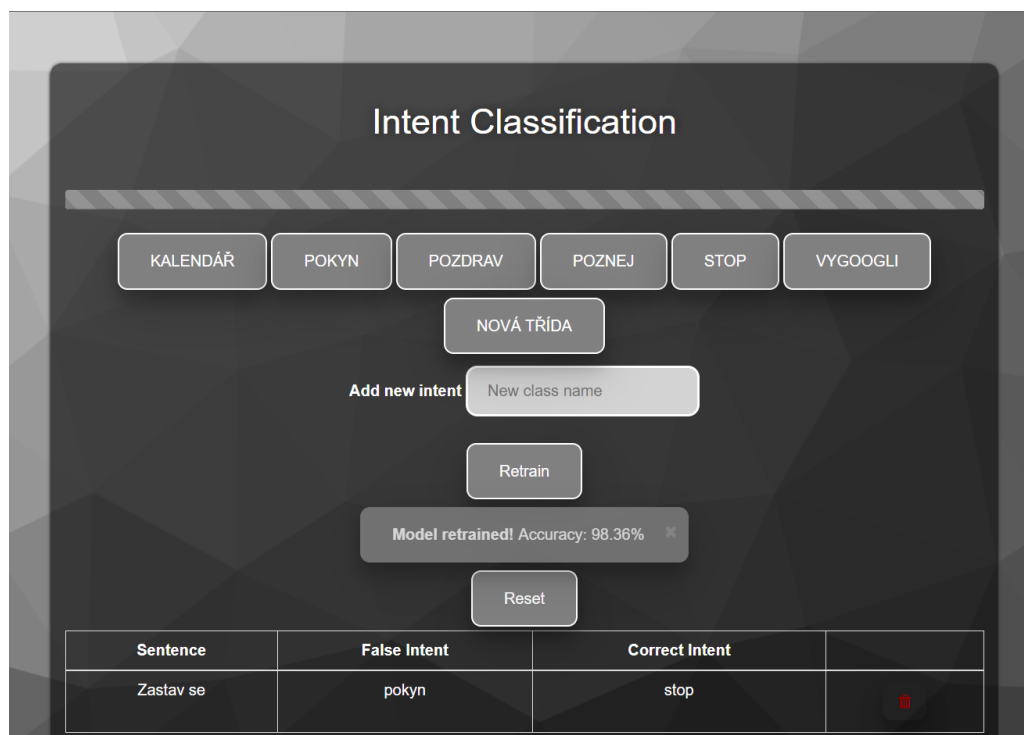
Po zobrazení predikovaného záměru má uživatel možnost poskytnout zpětnou vazbu na predikci. Potvrzení se provede kliknutím na **zeleně orámované tlačítko**. V případě, kdy shledá predikci jako nesprávnou může provést její korekci kliknutím na tlačítko se správným záměrem. Pokud záměr, který uživatel považuje za korektní, není na žádném z tlačítek, může si jej přidat – napsáním jména nového záměru a stiskem klávesy Enter. Přejde-li teď tedy od uživatele text do nové třídy, na Obrázku 3.5 je tímto textem „Věta do nové třídy“, změní uživatel její záměr z predikovaného „poznej“ na „nová třída“ kliknutím na příslušné tlačítko.



Obrázek 3.5: Korekce predikovaného záměru na záměr „Nová třída“

Tyto zpětné vazby od uživatele jsou průběžně ukládány a uživatel má možnost je propagovat do modelu klasifikátoru záměru pomocí tlačítka Retrain. Po stisknutí

tohoto tlačítka dojde k přeučení modelu na starém datasetu obohaceném o nové vzorky vytvořené z textového vstupu uživatele a jeho korekce predikce záměru pro daný text tj. správného záměru. Po přeučení, Obrázek 3.6, je zobrazena přesnost nového modelu a vzorky z celého datasetu, u nichž model při validaci chyboval. Uživatel má možnost tyto vzorky smazat kliknutím na ikonu koše.



Obrázek 3.6: Dokončení přeučení, možnost odstranění chybných vzorků

## 4.1 Dataset D1

Tento dataset obsahuje pouze 60 vzorků a nebyl rozdělen na trénovací, validační a testovací data. Vzorků je malé množství záměru, protože dataset sloužil především k odladění vyvíjených metod (*proof of concept*) a je možné tato data pojmout jako startovací bod a neuronovou síť dotrénovat navrženým algoritmem, tj. za základě zpětné vazby od uživatele ~~na predikce modelu~~ (*Human in the loop* přístup). Celý dataset je k nahlédnutí viz Příloha A. V Tabulce 4.1 jsou podrobnější informace o tomto datasetu.

Tabulka 4.1: Statistiky datasetu 1

Informace o datasetu	
Počet vzorků	60
Počet unikátních záměrů/tríd	6
Počet unikátních slov	114
Počet vzorků v jedné třídě	10
Průměrný počet slov na vzorek	2.78

## 4.2 Dataset D2



Tento dataset byl převzat z publikace ~~An Evaluation Dataset for Intent Classification and Out of Scope Prediction~~ [11]. Obsahuje více než 23000 vzorků, z toho 22500 **rámcových** (*in-scope*) vzorků pokrývá 150 různých záměrů, které lze rozdělit do deseti různých oblastí, zbylé vzorky jsou mimo rámec tzv. *out-of-scope*. Dataset je veřejně dostupný **na na** platformě GitHub<sup>1</sup>.

<sup>1</sup><https://github.com/clinc/oos-eval>

V rámci této práce bylo využito 22500 rámcových vzorků, které byly přeloženy do češtiny a byly rozděleny na trénovací, validační a testovací data. Dataset slouží k předtrénování většího modelu. Rozdělení dat a informace o nich jsou uvedeny v Tabulce 4.2.

Tabulka 4.2: Statistiky datasetu 2

Informace o datasetu			
	Trénovací	Validační	Testovací
Počet vzorků	15000	3000	4500
Počet unikátních záměrů	150	150	150
Počet unikátních slov	10633	4100	5114
Počet vzorků na jeden záměr	100	20	30
Průměrný počet slov na vzorek	6.78	6.78	6.70



# Experimenty a výsledky

## 5

### 5.1 Porovnání frameworků

Keras [9] a PyTorch [17] jsou dva populární frameworky pro vytváření a trénování neuronových sítí. Keras poskytuje vyšší úroveň abstrakce, což umožňuje snazší vytváření a trénování sítí bez nutnosti velkého množství kódu. Na druhé straně PyTorch poskytuje nižší úroveň abstrakce, což poskytuje větší kontrolu a flexibilitu při vytváření sítí.

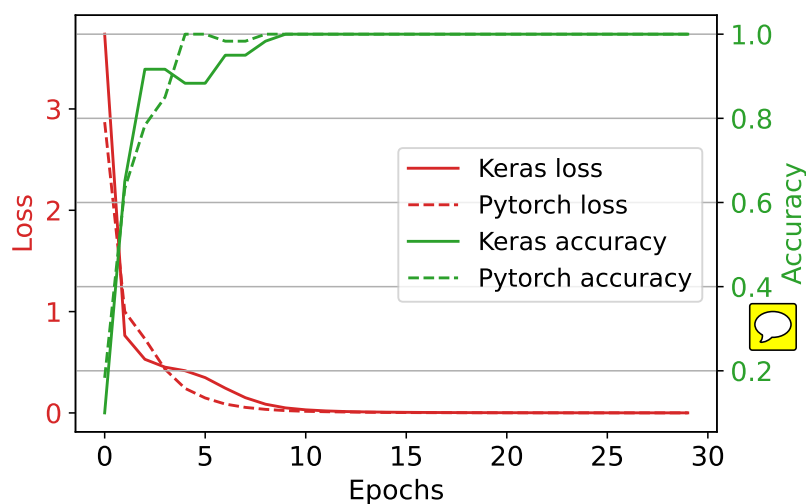
V rámci tohoto experimentu bylo provedeno porovnání rychlosti a efektivity frameworků PyTorch a Keras při použití stejných hyperparametrů neuronové sítě a stejného datasetu. Hyperparametry jsou vypsány v Tabulce 5.1, použitým datasetem byl Dataset 4.1 se šedesáti vzorky. Obě neuronové sítě dosahovaly s tímto nastavením přesnosti 100 %.

Tabulka 5.1: Hyperparametry neuronových sítí použitých v experimentu pro výběr frameworku

Parametr	Hodnota
Počet skrytých vrstev	1
Počet neuronů ve skryté vrstvě	35
Aktivační funkce	Relu, Softmax
Learning rate	0.01
Batch size	40
Počet epoch	30
Optimizér	Adam

Obrázek 5.1 ilustruje vývoj přesnosti (Accuracy) a ztráty (Loss) při trénování

neuronové sítě v Pytorch a Keras pro výše zmíněné hyperparametry a dataset.



Obrázek 5.1: Průběh trénování neuronové sítě v Pytorch a Keras

Experiment byl proveden na osobním počítači Lenovo IdeaPad Gaming 3 s procesorem Intel Core i5 a grafickou kartou NVIDIA GeForce GTX 1650 a na Raspberry Pi 4B. Během experimentu byly měřeny tři časy: doba importu knihoven, doba načtení modelu neuronové sítě a doba predikce třídy pro jeden vzorek. Bylo provedeno 10 realizací tohoto experimentu, výsledky jsou v Tabulce 5.2.


Tabulka 5.2: Doby načtení knihoven, načtení modelů a predikcí jednoho vzorku pro frameworky Pytorch a Keras (v sekundách).

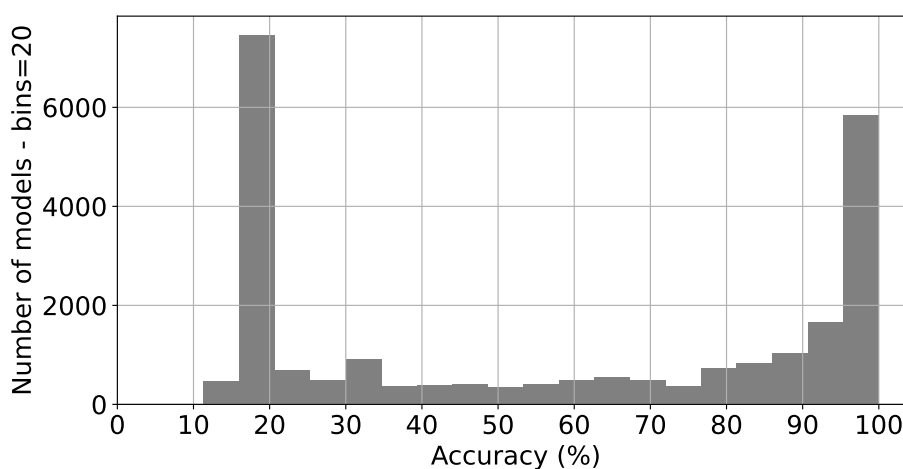
	Pytorch	
Operace	Čas [s] (laptop)	Čas [s] (RPi)
Načtení knihoven	3.329±0.055	13.785±0.087
Načtení modelu	0.001±0.000	0.004±0.000
Predikce jednoho vzorku	0.036±0.014	0.672±0.226
	Keras	
Operace	Čas [s] (laptop)	Čas [s] (RPi)
Načtení knihoven	1.805±0.046	5.862±0.529
Načtení modelu	0.001±0.000	0.004±0.001
Predikce jednoho vzorku	0.028±0.001	0.644±0.081

Výsledky experimentu ukázaly, že v případě importu knihoven byl PyTorch pomalejší než Keras (přibližně 2×). Pro načtení modelu byly časy téměř srovnatelné, v případě predikce jednoho vzorku byl Keras mírně rychlejší než Pytorch (o 0.008 sekund na osobním počítači, o 0.028 sekund na RPi).

## 5.2 Volba struktury sítě

Výběr velikosti sítě je jedním z klíčových rozhodnutí při návrhu neuronových sítí a ovlivňuje jak přesnost, tak rychlost učení modelu. V rámci tohoto experimentu bylo cílem nalézt hyperparametry pro neuronovou síť, která bude aplikována na Raspberry Pi 4B a to tak, aby tato síť dosahovala co největší přesnosti a zároveň byla dostatečně rychlá.

Experiment byl proveden na osobním počítači na jednoduché úloze klasifikace pro Dataset 4.1. Použit  framework PyTorch a implementace sítě s jednou skrytou vrstvou. Nejprve bylo testováno 24000 modelů s různými hyperparametry (viz Příloha B) a s pěti různými seedy<sup>1</sup> z hlediska jejich přesnosti. Histogram zobrazující četnosti modelů v závislosti na dosažené přesnosti je na Obrázku 5.2.

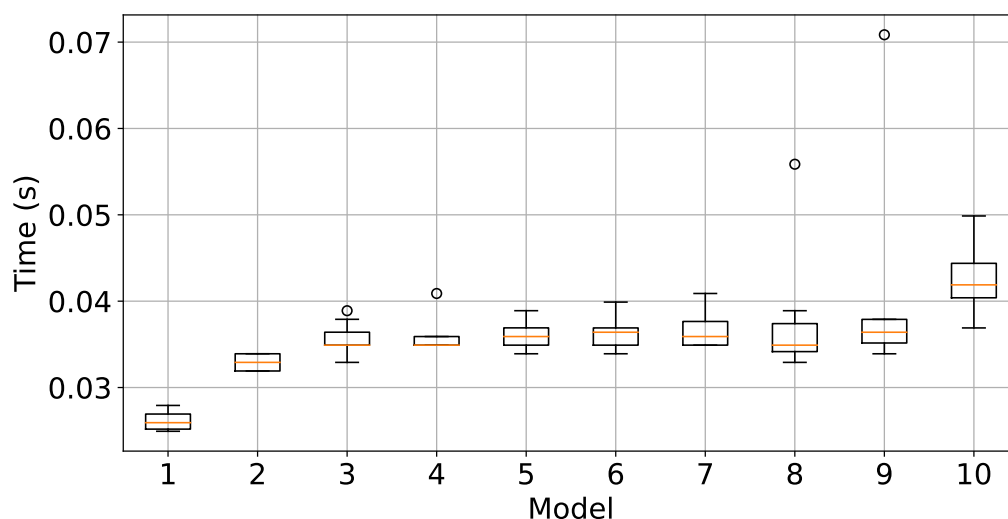


Obrázek 5.2: Histogram četností modelů v závislosti na jejich přesnosti

Dále byly vybrány neuronové sítě, které dosáhly přesnosti 100 % pro všech pět testovaných seedů. Tyto vybrané neuronové sítě byly znovu testovány pro dalších 10 různých seedů, aby byla důkladněji ověřena jejich kvalita, tj. jestli pro všechny tyto **realizace** dosahují přesnosti 100 %. Kromě toho byla pro jednotlivé realizace

<sup>1</sup> Pod pojmem seed se rozumí seed pro **generování** pseudonáhodných čísel, **česky též semínko nebo násada**.

měřena doba trénování sítě. Bylo získáno 514 neuronových sítí s přesností 100 % pro všechny realizace, z nich bylo vybráno 10 nejrychlejších sítí, boxplot doby jejich trénování je na Obrázku 5.3, konkrétní hyperparametry pak v Tabulce 5.3 (indexy modelů v grafu odpovídají řádkům v tabulce).



Obrázek 5.3: Boxplot doby trénování jednotlivých modelů

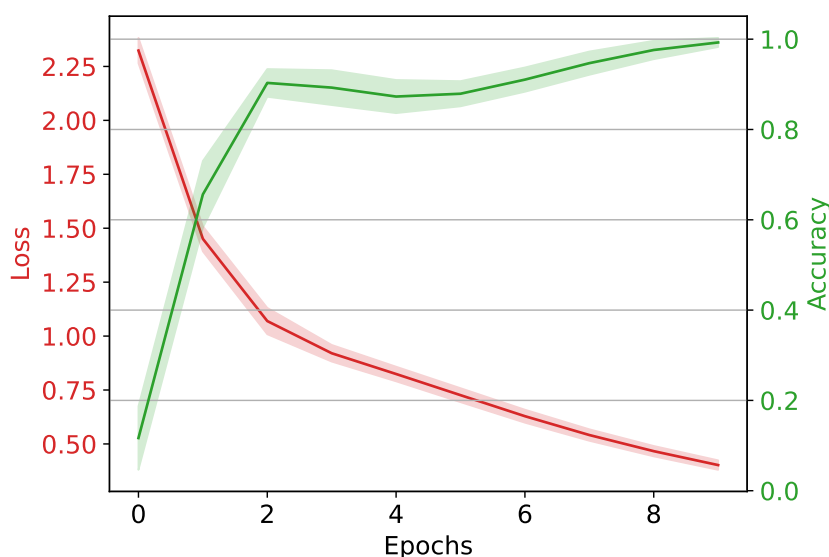
Tabulka 5.3: Hyperparametry deseti nejrychlejších neuronových sítí a jejich doby trvání trénování, koeficient učení byl ve všech případech 0.01 a optimizér Adam

Model	Počet neuronů ve skryté vrstvě	Počet epoch	Batch size	Doba trénování [s]
1	45	10	40	0.0262±0.0011
2	20	15	40	0.0329±0.0009
3	45	15	40	0.0355±0.0017
4	50	15	50	0.0358±0.0018
5	35	15	40	0.0361±0.0017
6	35	15	50	0.0363±0.0019
7	50	15	40	0.0366±0.0019
8	30	15	40	0.0373±0.0065
9	45	15	50	0.0396±0.0105
10	40	15	40	0.0424±0.0036

Z Tabulky 5.3 je zřejmé, že nejlepší výsledků dosáhla neuronová síť číslo 1 se



čtyřiceti pěti neurony ve skryté vrstvě, deseti epochami, učícím koeficientem 0.01, velikostí dávky 40 a optimizérem Adam. Na Obrázku 5.4 je zobrazen průběh trénovacího procesu této neuronové sítě.




Obrázek 5.4: Průběh trénování neuronové sítě s optimálními hyperparametry pro 10 realizací

## 5.3 Trénování v závislosti na počtu vzorků a tříd

U klasifikačních úloh je nezbytné mít dostatečné množství trénovacích dat, aby bylo možné naučit neuronovou síť rozlišovat jednotlivé třídy. V rámci tohoto experimentu byly analyzovány schopnosti neuronových sítí natrénovat se s různým počtem vzorků a tříd a vliv tohoto počtu na přesnost klasifikace.

V průběhu experimentu byly použity různé datové sady s různým počtem tříd a vzorků na třídu, vždy se jednalo o modifikace Datasetu 4.2. Tento experiment byl rozdělen na tři části; výběr hyperparametrů pro neuronovou síť pro klasifikaci záměru na Datasetu 4.2, analýza schopnosti neuronové sítě natrénovat se v závislosti na architektuře sítě a na počtu tříd, analýza schopnosti neuronové sítě natrénovat se v závislosti na počtu tříd a na počtu vzorků v jednotlivých třídách.

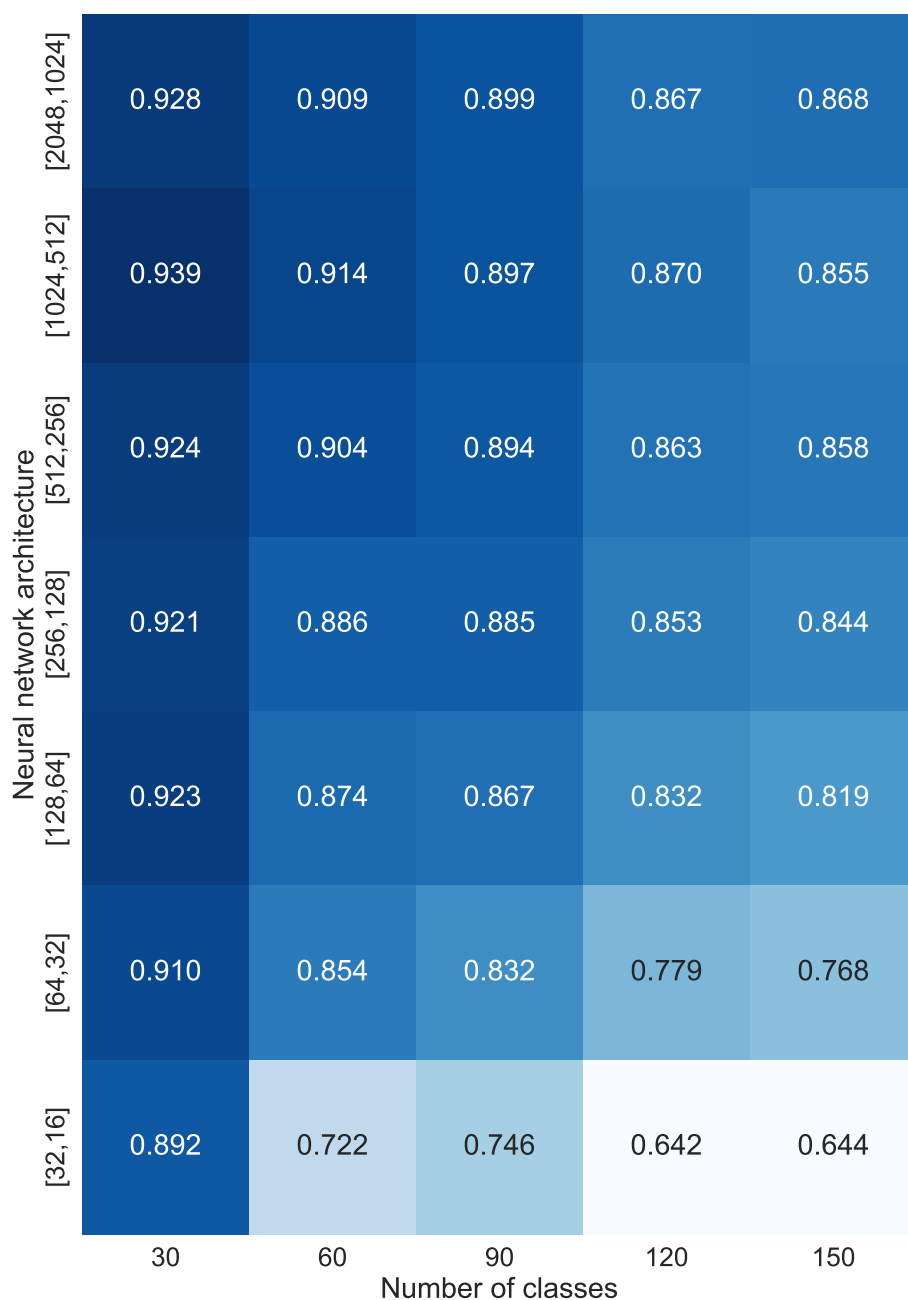
Výběr hyperparametrů pro neuronovou síť byl proveden na celém Datasetu 4.2 pro framework Pytorch. Validační data byla využita k výpočtu validační ztráty po každé epoše, přičemž pokud byla tato ztráta rostoucí po více než pětinu počtu epoch, bylo trénování neuronové sítě předčasně ukončeno. V průběhu trénování

neuronové sítě byl také dynamicky měněn koeficient učení, každých padesát epoch byl přenásoben koeficientem 0.9 a došlo tak k jeho poklesu. Experimentálně vybrané hyperparametry jsou vypsány v Tabulce 5.4. Tato neuronová síť dosahovala přesnosti  $0.854 \pm 0.002$  a ztráty  $0.588 \pm 0.015$  (testováno pro 15 realizací) 

Tabulka 5.4: Hyperparametry neuronové sítě pro klasifikaci záměru trénované na Datasetu 4.2

Parametr	Hodnota
Počet skrytých vrstev	2
Počet neuronů ve skrytých vrstvách	512, 256
Aktivační funkce	Sigmoid, Sigmoid, Softmax
Počáteční hodnota koeficientu učení	0.003
Batch size	2048
Počet epoch	300
Optimizér	Adam

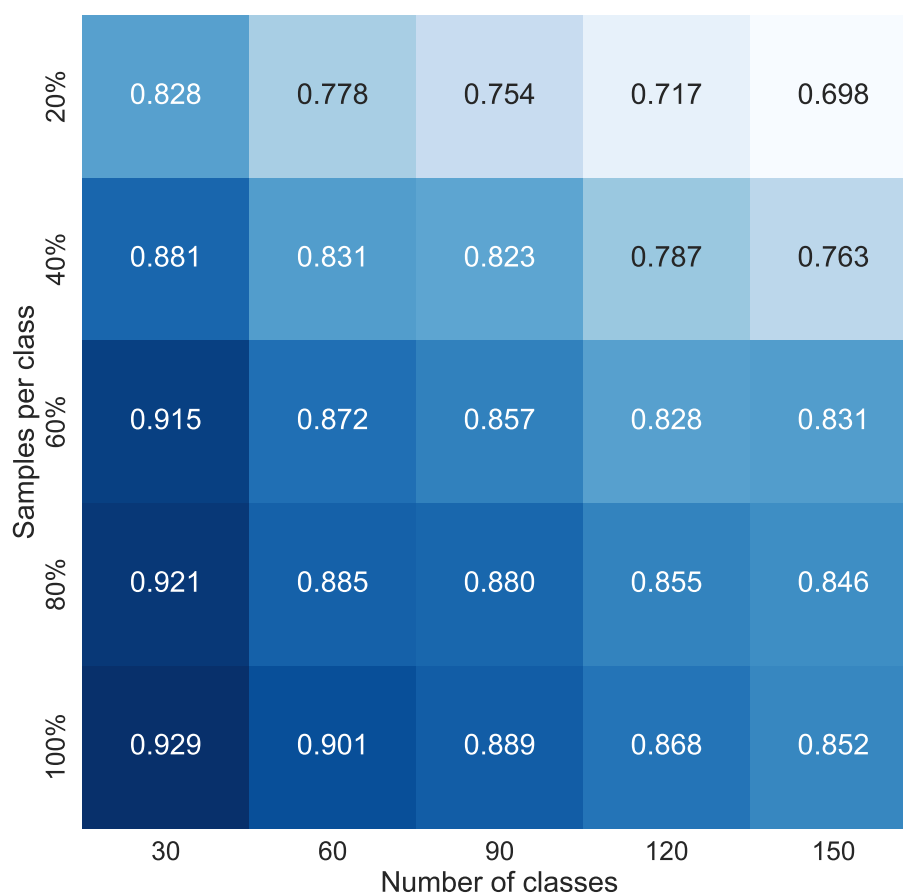
Pro analýzu schopnosti neuronové sítě natrénovat se v závislosti na architektuře sítě a na počtu tříd bylo vytvořeno sedm neuronových sítí s různým počtem neuronů ve skrytých vrstvách, ostatní hyperparametry odpovídaly hyperparametrům z Tabulky 5.4. Pro každou z těchto sítí bylo vytvořeno pět modifikací Datasetu 4.2 obsahující různé počty tříd. Vzhledem k tomu, že v původním datasetu bylo 150 tříd, byly počty tříd rozděleny po násobcích třiceti – 30, 60, 90, 120, 150. Na Obrázku 5.5 je heatmapa zobrazující přesnosti neuronové sítě v závislosti na její architektuře a počtu tříd. Na svislé ose jsou vypsány počty neuronů v jednotlivých skrytých vrstvách.



Obrázek 5.5: Přesnost neuronové sítě v závislosti na její architektuře (počtu neuronů ve skrytých vrstvách) a počtu tříd

Analýza schopnosti neuronové sítě natrénovat se v závislosti na počtu tříd a na počtu vzorků v jednotlivých třídách byla provedena pro neuronovou síť s hyperparametry z Tabulky 5.4. Počty tříd byly stejné jako u předchozí analýzy. Počty vzorků pro jednotlivé třídy byly opět děleny do pěti kategorií po dvaceti procentních úbytcích. Například pro testovací data odpovídaly tedy: 30, 24, 18, 12, 6. Výsledky této analýzy

jsou na Obrázku 5.6 představujícím heatmapu přesností neuronové sítě.



Obrázek 5.6: Přesnost neuronové sítě v závislosti na počtu tříd a počtu vzorků na třídu

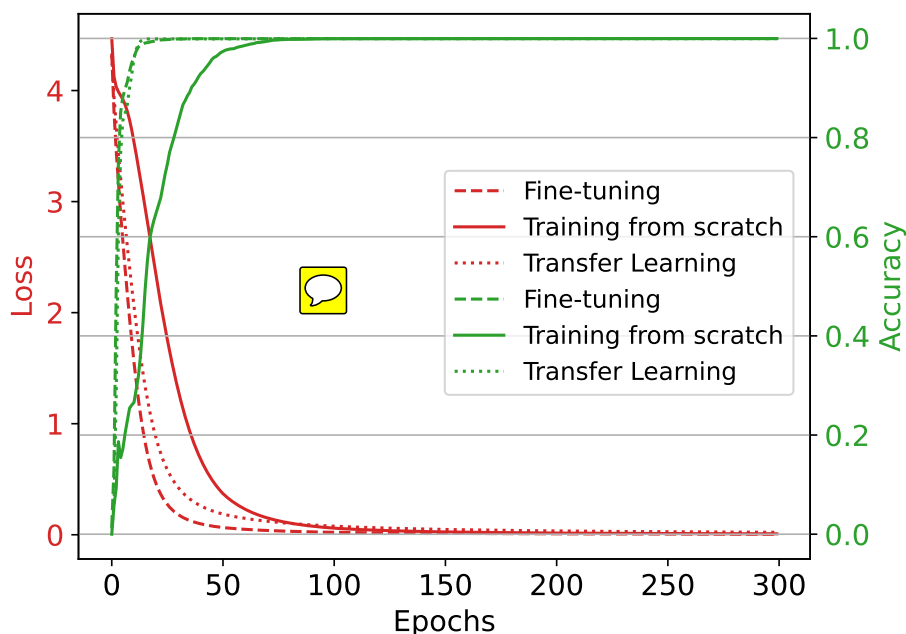
Výsledky experimentu ukázaly, že přesnost sítě se zvyšovala s rostoucím počtem neuronů ve skrytých vrstvách, zároveň tato přesnost klesala s rostoucím počtem tříd. Vyšší počet vzorků v jednotlivých třídách pak opět zvyšoval přesnost neuronové sítě.

## 5.4 Analýza použití předtrénované neuronové sítě

Fine-tuning, transfer learning a trénink „od nuly“ jsou tři způsoby trénování neuronových sítí. Při tréninku „od nuly“ se vytvoří nová síť a váhy se inicializují náhodně, zatímco při fine-tuningu a transfer learningu se používá již předtrénovaný model. Tento experiment se zabývá porovnáním těchto přístupů.

V rámci experimentu byla využita neuronová síť z experimentu 5.3 trénovaná na Datasetu 4.2. Byly porovnány tři modely neuronové sítě, první dva modely byly vytvořeny z předtrénovaného modelu, první pomocí fine-tuningu – poslední vrstva neuronové sítě byla nahrazena za náhodnou inicializaci, druhý pomocí transfer learningu – poslední vrstva neuronové sítě byla nahrazena za náhodnou inicializaci, ostatní vrstvy byly „zmrazeny“. Modely byly porovnávány z hlediska jejich průběhu trénování a validace a z hlediska výsledné přesnosti.

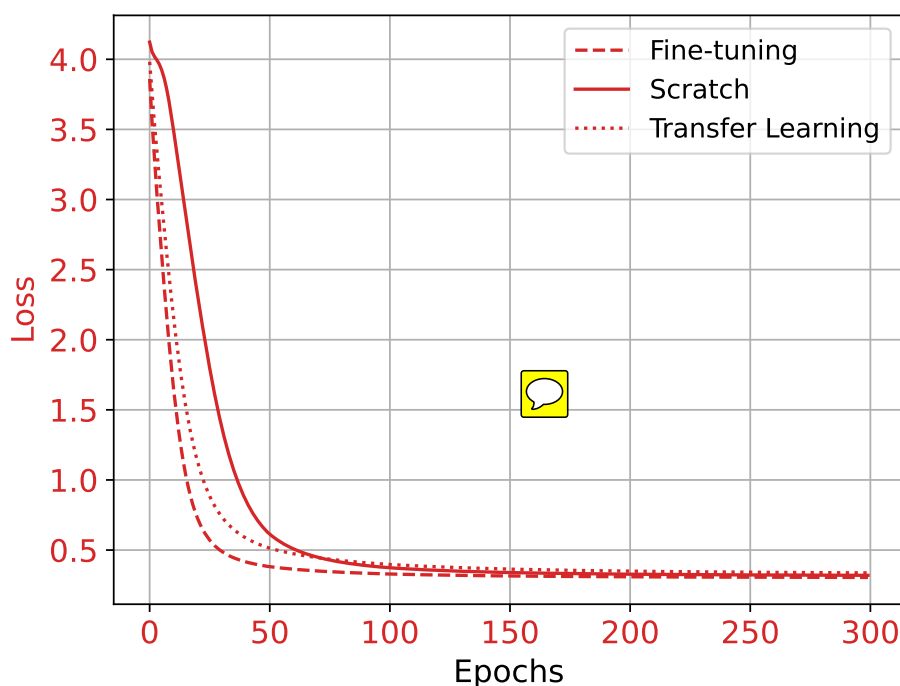
Nejprve byly tyto modely porovnány pro modifikaci Datasetu 4.2 o šedesáti třídách a šedesáti trénovacích, dvanácti validačních a osmnácti testovacích vzorcích. První model dosáhl přesnosti 0.89 a ztráty 0.41, druhý model dosáhl přesnosti 0.89 a ztráty 0.39, třetí model dosáhl přesnosti 0.87 a ztráty 0.46. Na Obrázku 5.7 je zobrazen vývoj přesnosti a ztráty při trénování těchto modelů. Je zřejmé, že model trénovaný pomocí fine-tuningu a model trénovaný pomocí transfer learningu mají téměř identický průběh přesnosti trénování a že dosahují přesnosti 1.0 přibližně o 60 epoch dříve než model trénovaný „od nuly“. V případě poklesu ztráty si vedl nejlépe model trénovaný pomocí fine-tuningu, mírně pomalejší je model trénovaný pomocí transfer learningu a nejpomalejší byl model trénovaný „od nuly“.



Obrázek 5.7: Vývoj přesnosti a ztráty při trénování modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 60 tříd

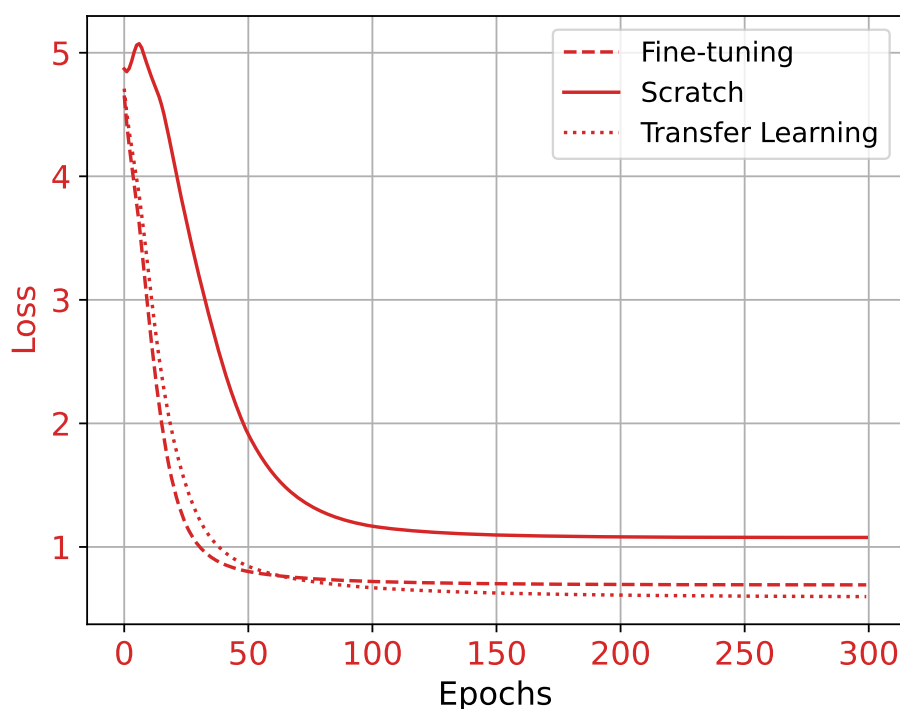
Obrázek 5.8 pak znázorňuje průběh ztráty při validaci těchto modelů, trend poklesu validační ztráty je analogický s trendem poklesu trénovací ztráty pro všechny tři

modely.



Obrázek 5.8: Vývoj ztráty při validaci modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 60 tříd

Dále byla testována situace, kde bylo z původního datasetu vybráno 130 tříd s dvaceti trénovacími, čtyřmi validačními a šesti testovacími vzorky. Jednalo se tedy o poměrně okrajový případ řídkého datasetu. V tomto případě dosáhl první model přesnosti 0.83 a ztráty 0.64, druhý model přesnosti 0.85 a ztráty 0.54 a třetí model měl přesnost 0.72 a ztrátu 1.12. Vývoj validační ztráty pro všechny modely je na Obrázku 5.9, graf vývoje přesnosti a ztráty při trénování je k nahlédnutí v příloze C. Je patrné, že validační ztráta modelu trénovaného „od nuly“ je značně vyšší (přibližně 2×) než u ostatních modelů a že po přibližně posledních 100 epoch již neklesala. Validační ztráta modelu trénovaného pomocí fine-tuningu klesala v prvních sedmdesáti epochách rychleji než v případě modelu trénovaného pomocí transfer learningu, po té se ale ustálila, zatímco validační ztráta modelu trénovaného pomocí transfer learningu dále klesala a ve výsledku dosáhla hodnoty o 0.1 nižší.



Obrázek 5.9: Vývoj ztráty při validaci modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 130 tříd

Výsledky experimentu naznačují, že pro pouhou modifikaci většího datasetu je fine-tuning nebo transfer learning efektivnějším přístupem než trénink „od nuly“, pokud je k dispozici již předtrénovaný model. Při porovnávání modelů pro dataset se šedesáti třídami dosahovaly modely trénované pomocí fine-tuningu a transfer learningu srovnatelných výsledků s přesností 0.89. Průběh poklesu trénovací i validační ztráty byl u modelu trénovaného pomocí fine-tuningu mírně rychlejší. Modelu trénovaný „od nuly“ dosáhl přesnosti pouze 0.87 a měl nejpomalejší (z hlediska počtu epoch) trend při trénování i validaci. V případě datasetu obsahujícího 130 tříd dosáhl nejlepších výsledků model trénovaný pomocí transfer learningu (přesnosti 0.85 a ztráty 0.54).



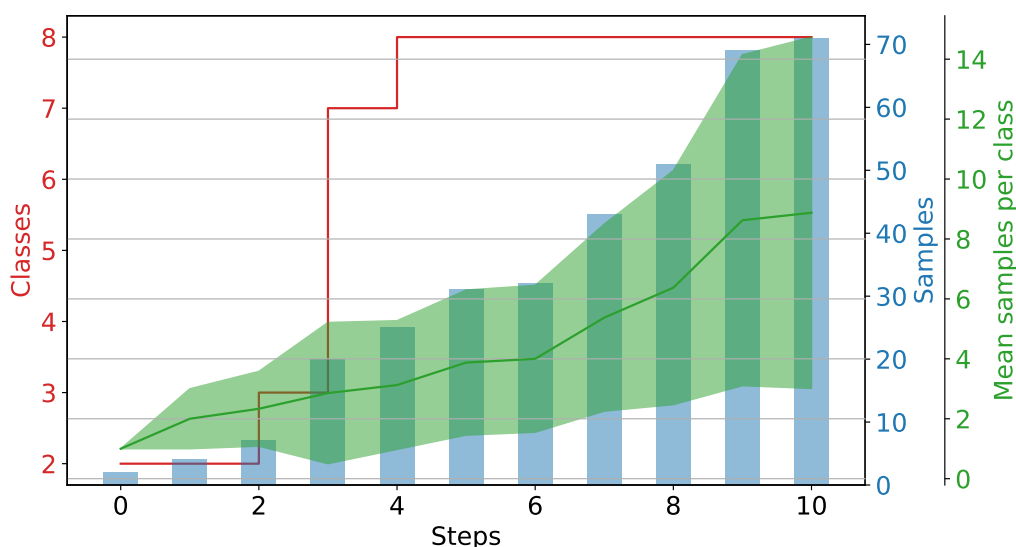
## 5.5 Vyhodnocení metody učení dialogem

Učení dialogem umožňuje interaktivně trénovat a zlepšovat schopnosti modelu prostřednictvím komunikace s uživateli v reálném čase. V tomto experimentu byla simulována situace opakovaného přeučování modelu na základě zpětné vazby od uživatele.

Tento experiment používá předtrénovaný model z předchozího experimentu (viz sekce 5.3), který byl trénován na Datasetu 4.2. Pro přeučování byl použit nový dataset, který byl iterativně rozšiřován o nové vzorky od uživatele, jeho celková podoba je k nahlédnutí v Příloze D. Protože je obecně předpokládáno, že uživatelem upravovaný dataset bude řídký, byl na základě předchozího experimentu zvolen transfer learning jako metoda přeučení sítě.

Simulace byla prováděna pro dvě počáteční podmínky, nejprve pro dataset začínající na dvou třídách o jednom vzorku, ve druhém případě pro dataset začínající se vzorky z Datasetu 4.1. V každém následujícím kroku bylo možné přidat libovolný počet vzorků, ať už do existujících tříd nebo vytvořit nové třídy. Proces přeučování probíhal taktéž iterativně, po přidání nových vzorků byl model přeučen. V každém kroku byla také měřena přesnost modelu a doba trénování. Jelikož se počet vzorků měnil v každém kroku, byly při přeučování modelu také počet epoch a batch size nastaveny na proměnlivou hodnotu, která byla závislá na aktuálním počtu vzorků.

Obrázek 5.10 znázorňuje iterativní rozšiřování datasetu pro první počáteční podmínku. Červenou barvou je znázorněn počet tříd v jednotlivých krocích, modré sloupce představují počty vzorků, zelená barva vykresluje průměrný počet vzorků na třídu spolu se směrodatnou odchylkou.

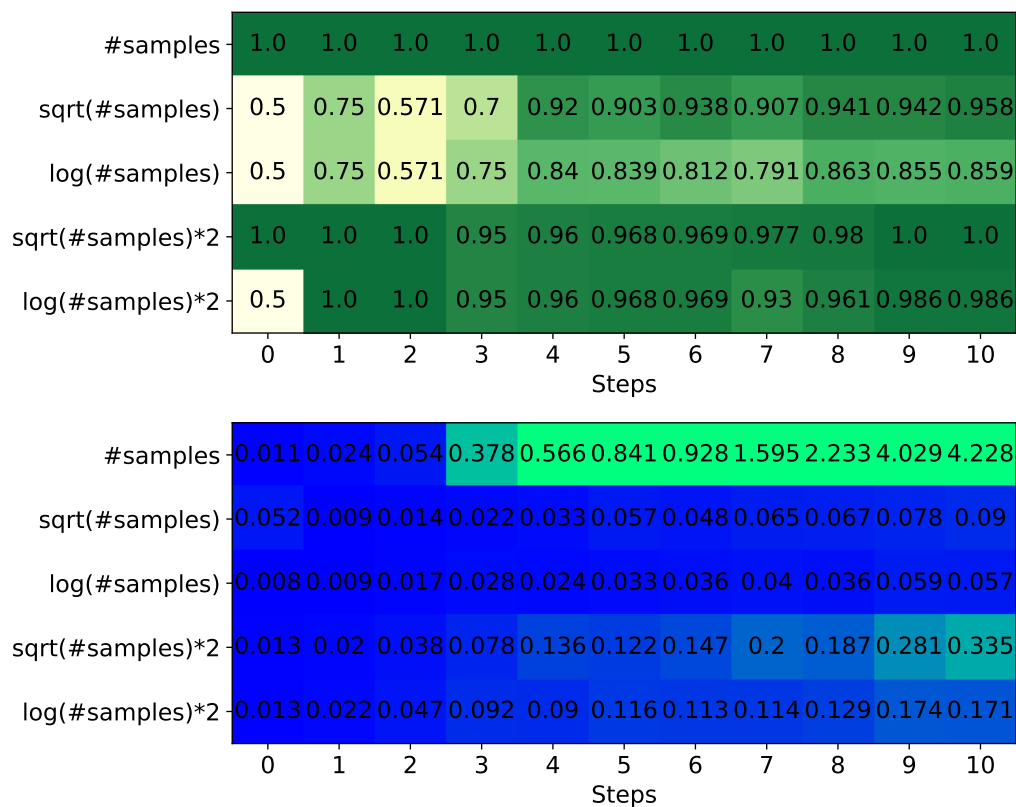


Obrázek 5.10: Iterativní rozšiřování datasetu při učení dialogem pro dvě počáteční třídy

Vývoj přesnosti v jednotlivých krocích (jednotlivé kroky odpovídají krokům v Obrázku 5.10) je pak na Obrázku 5.11, na svislé ose jsou popsány závislosti mezi počty epoch a počty vzorků, přičemž počty epoch byly zaokrouhleny na celá čísla.



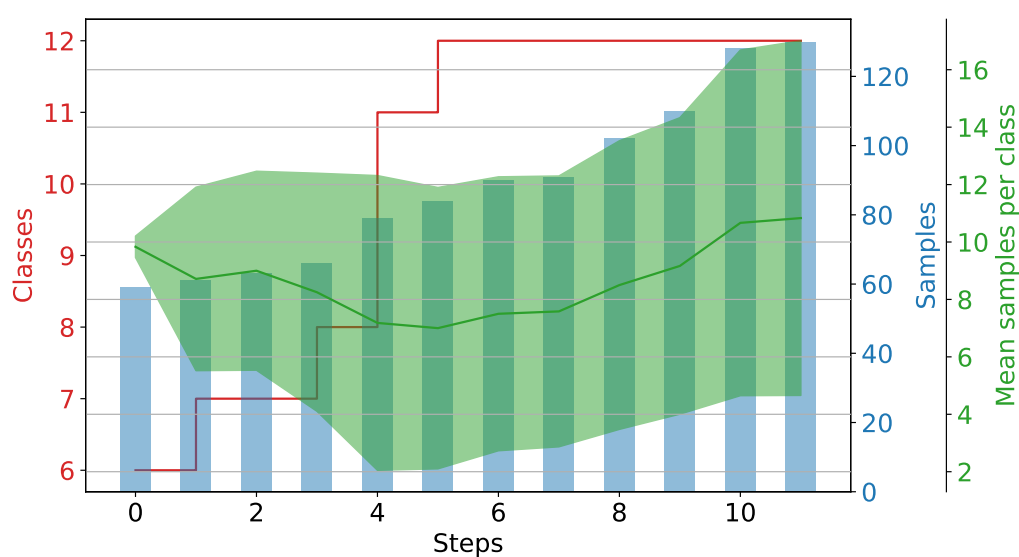
Hodnota batch size byla vždy dopočítána na základě počtu epoch takto:  $\lfloor \#vzorků / \#epoch \rfloor$ . Obrázek 5.11 také zobrazuje doby trvání<sup>2</sup> přeučení v jednotlivých krocích pro různé počty epoch.



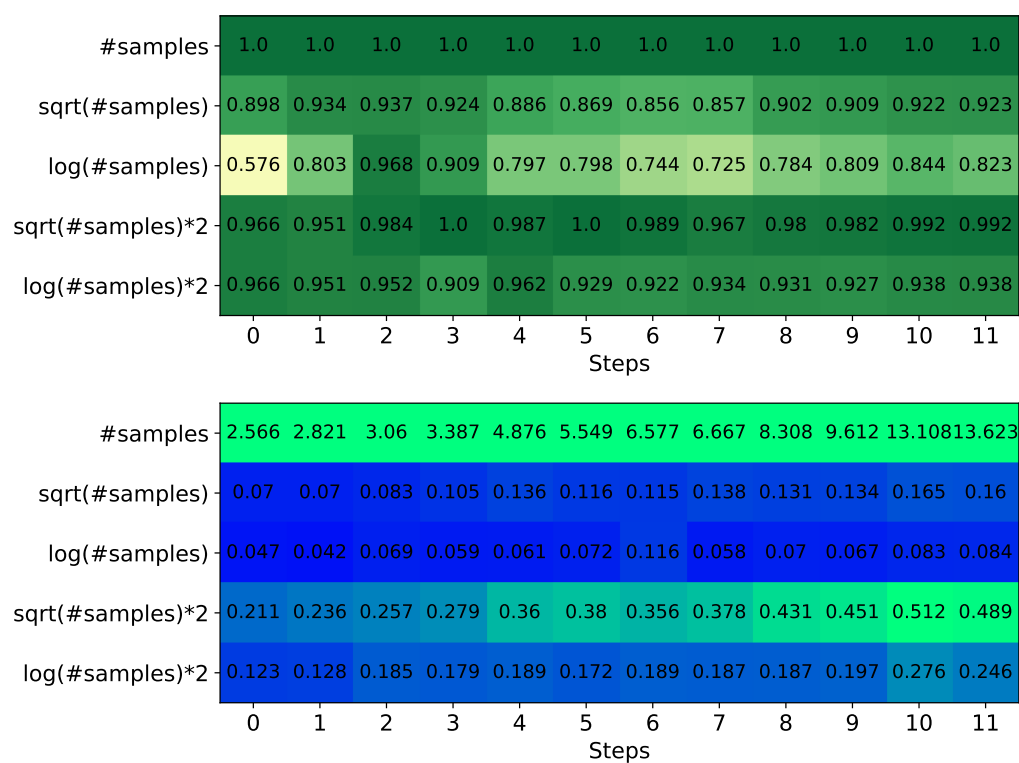
Obrázek 5.11: Vývoj přesnosti přeučeného modelu a doby trvání jeho přeučení [s] v jednotlivých krocích při učení dialogem pro dvě počáteční třídy

Ve druhém případě, **rac** najím se vzorky z Datasetu 4.1, byly v každém kroku přidávány **stejně prvky jako** u předchozího případu. Na Obrázku 5.12 je opět znázorněno postupné rozšiřování datasetu – vývoj počtu tříd, počtu vzorků a průměrného počtu vzorků na třídu. Stejně jako v předchozím případě byla měřena také přesnost modelu a doba jeho přeučení v závislosti na počtu epoch. Výsledky jsou k vidění na Obrázku 5.13.

<sup>2</sup>Časy byly měřeny na osobním počítači.



Obrázek 5.12: Iterativní rozšiřování datasetu při učení dialogem pro šest počátečních tříd



Obrázek 5.13: Vývoj přesnosti přeučeného modelu a doby trvání jeho přeučení [s] v jednotlivých krocích při učení dialogem pro šest počátečních tříd

Výsledky experimentu ukazují, že v obou případech byla nejvyšší přesnost dosažena

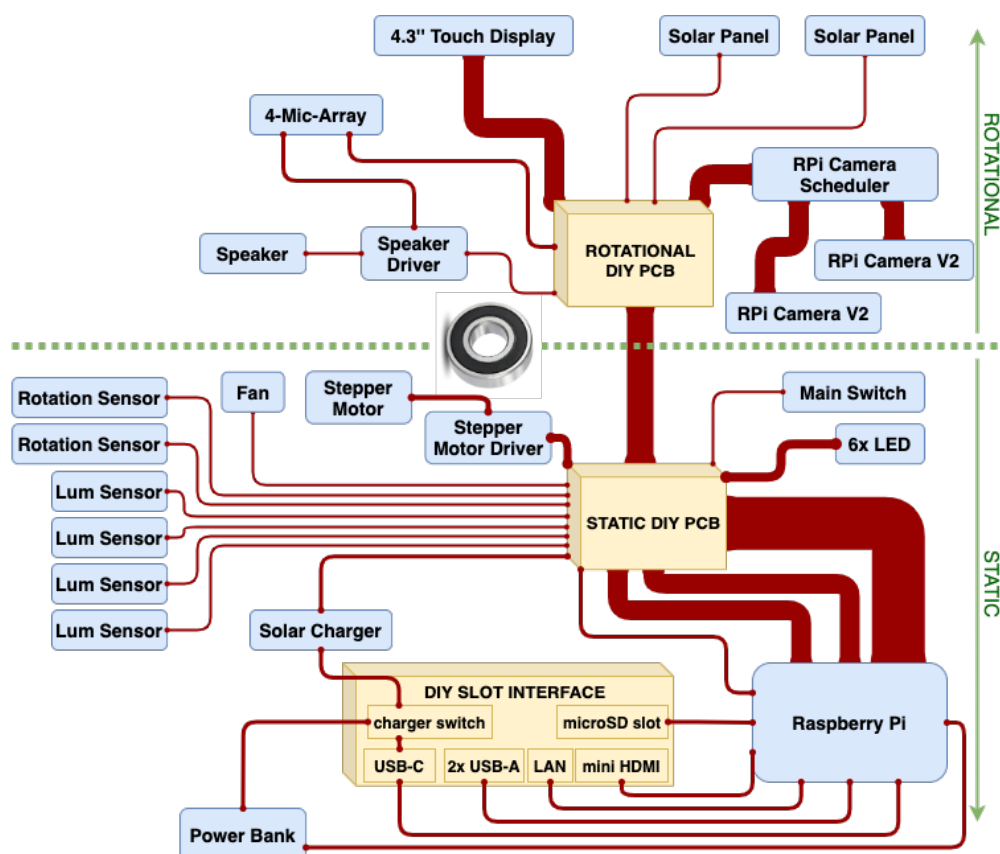
při proměnlivé hodnotě počtu epoch nastavené jako  $\#vzorků$ . Tento přístup však vedl k nejdelší době přeučování modelu. Dále je patrné, že v případech, kdy byl počet epoch příliš nízký, byla i přesnost modelu nedostatečná (pod 80 %), pro malý dataset proto modely s počtem epoch daným  $2\sqrt{\#vzorků}$  a  $2 \log \#vzorků$  dosáhly vyšší přesnosti než u modelů s počtem epoch  $\sqrt{\#vzorků}$  a  $\log \#vzorků$ .



## 6.1 Cílová robotická platforma

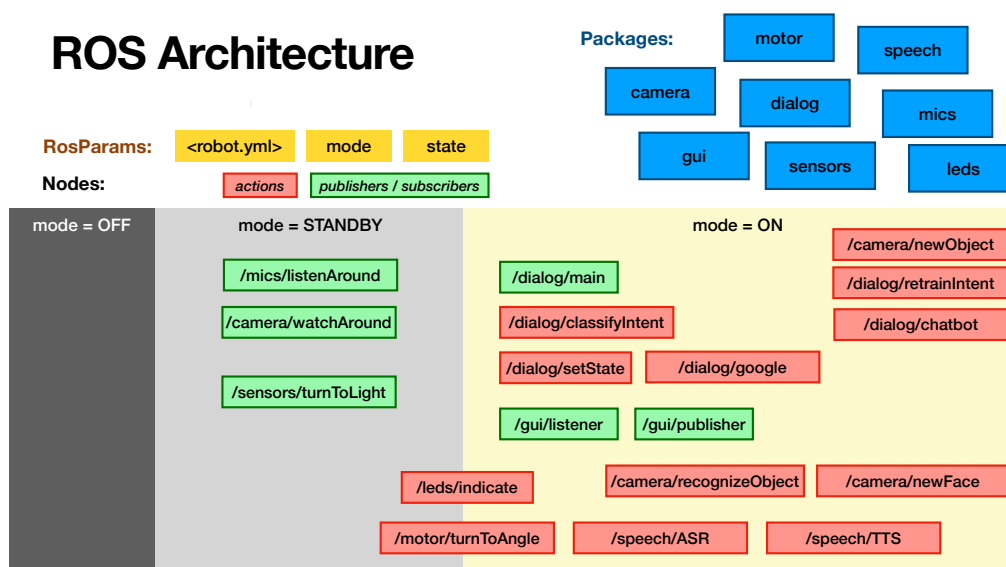
Cílová robotická platforma s pracovním názvem „Robot“ byla navržena jako fyzická entita, která je založena na jednodeskovém počítači typu Raspberry Pi. Tato platforma je navržena s cílem poskytnout nízkonákladové řešení pro testování a aplikaci vědeckých úloh v oblasti přirozené komunikace člověka a stroje.

Platforma je vybavena hlasovým a vizuálním rozhraním, které zprostředkovává komunikaci s uživatelem pomocí pole mikrofونů, reproduktoru, dotykového displeje a kamery. Kromě toho platforma obsahuje krokový motor, který umožňuje rotaci určité části robotu, senzory světla a solární panel pro zajištění energetického zdroje, indikační LED diody, **power banku** a displej pro zobrazení aktuálního stavu baterie a **jedno-polohový spínač**. Jednotlivé komponenty využití v robotické entitě a jejich umístění v rámci statické a rotační části je nastíněno na Obrázku 6.1, je zde také náčrt elektrického propojení těchto součástí. Významnou součástí celého projektu je také ROS (Robot Operating System), který zajišťuje hlavní funkčnost platformy. Obrázek 6.2 ukazuje náčrt ROS architektury a seznam použitých balíčků a uzlů.



Obrázek 6.1: Hardwarové propojení jednotlivých součástek použitých na robotické entitě

## ROS Architecture



Obrázek 6.2: ROS architektura robotické entity

Výsledná robotická platforma využívá SpeechCloud k syntéze a rozpoznávání řeči pro komunikaci s člověkem. Navíc dokáže rozpoznat různé objekty a známé tváře a umí provádět rotaci horní části tak, aby se natočila směrem k zdroji světla nebo zdroji zvuku. Řízení dialogu je zajištěno klasifikátorem záměru, jako jeden ze záměrů byl navržen chatbot založený na T5. Na Obrázku 6.3 je pak fotografie výsledného vzhledu vytvořené robotické entity.



Obrázek 6.3: Fotografie výsledné robotické entity

## 6.2 Učení konkrétních záměrů

Tato práce se na projektu „Robot“ podílela účastí na řešení hlavního dialogového manažeru. Pro potřeby projektu byla vytvořena neuronová síť, která dokáže klasifikovat záměr ze vstupní promluvy, a k ní byl navržen algoritmus přeučení. Dále bylo navrženo řešení generování odpovědi u vybraných záměrů.

Vzhledem k povaze navrženého algoritmu lze dle potřeby robota libovolně přeučovat a přidávat, respektive ubírat třídy příslušné jednotlivým záměrům.

Níže následuje výčet navržených záměrů a jejich uplatnění v rámci robotické entity:

- „otoč se“ – ovládání natočení rotační části robota
- „LED“ – kontrola pole LED světel, například zablikání příslušného světla
- „detekce světla“ – detekce zdroje světla
- „detekce objektu“ – rozpoznávání naučených objektů
- „seznam se“ – seznámení s novou osobou pro detekci známé tváře
- „překlad“ – přeložení libovolné promluvy do vybraného jazyka
- „Chat GPT“ – komunikace s chatovací platformou Chat GPT
- „vyhledávání“ – vyhledání požadované informace na internetu

Pro vyhledávání informací na internetu byla využita funkce *Featured Snippet* vyhledávače Google. Tato funkce umožňuje uživatelům získat přesnou odpověď na jejich dotaz přímo na výsledkové stránce vyhledávání, bez nutnosti klikat na odkazy a procházet další stránky. Pro získání těchto odpovědí byla použita technika web scraping, která umožňuje automatické stahování a extrakci dat z webových stránek. Pro tento účel byly v jazyce Python použity knihovny *requests*<sup>1</sup> a *BeautifulSoup*<sup>2</sup>. Knihovna *requests* umožňuje odesílat HTTP požadavky na server a získávat tak obsah stránek, pomocí knihovny *BeautifulSoup* lze z tohoto obsahu extrahovat požadované informace.



<sup>1</sup><https://pypi.org/project/requests/>

<sup>2</sup><https://pypi.org/project/beautifulsoup4/>



Přeučování neuronových sítí je významným aspektem vývoje umělé inteligence, který poskytuje možnost dosahovat výsledků přesahujících úroveň původního modelu. Umožňuje neuronovým sítím přizpůsobit se novým podmínkám a novým datům. Klíčovými faktory úspěšného přeučení neuronové sítě jsou nalezení vhodné struktury sítě, výběr vhodného datasetu a také zvolení vhodného způsobu přeučení.

## 7.1 Rekapitulace metod



Dopředná neuronová síť je nejjednodušší a nejčastěji používaný typ neuronové sítě. Jedná se o síť, ve které se signály šíří pouze v jednom směru, od vstupní vrstvy přes skryté vrstvy až k výstupní vrstvě, kde každá vrstva obsahuje několik neuronů. Vstupní vrstva slouží k přijímání vstupních dat a propagaci těchto dat do skrytých vrstev sítě. V těchto skrytých vrstvách dochází k zpracování dat pomocí vah, biasů a aktivačních funkcí neuronů. Na základě těchto zpracovaných dat výstupní vrstva generuje výstupní hodnoty, které představují výsledek sítě.

Učení neuronové sítě je pak proces, při kterém se síť učí přizpůsobovat vstupním datům a vytvářet relace mezi vstupem a výstupem. Během učení se upravují váhy a biasy jednotlivých neuronů tak, aby byla minimalizována chyba predikce výstupu sítě. Existují různé metody učení neuronových sítí, včetně fine-tuningu, transfer learningu a trénování "od nuly".

Trénování "od nuly" (training from scratch) je proces trénování neuronové sítě bez využití již existujícího předtrénovaného modelu. Fine-tuning využívá již natrénovanou síť a upravuje její váhy a biasy pro nový úkol. Transfer learning je podobný jako fine-tuning, ale upravuje jen některé části sítě pro nový úkol.

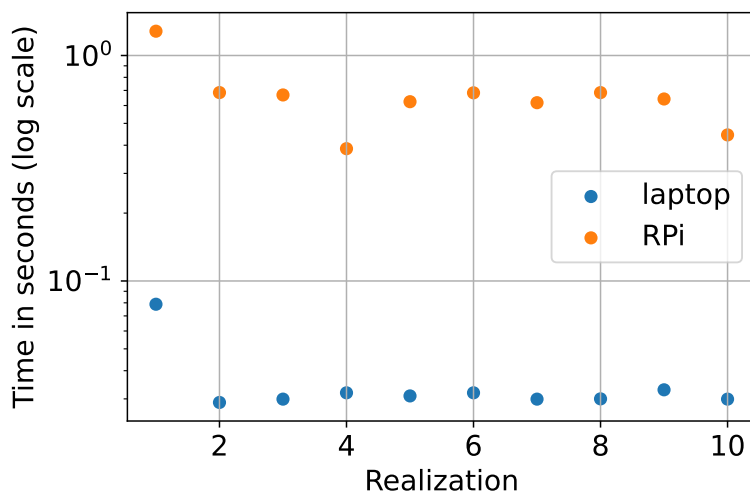
## 7.2 Shrnutí výsledků

**Experiment 5.1.** porovnával frameworky Keras a Pytorch pro pevně danou architekturu a dataset. Keras měl kratší doby načtení knihoven (přibližně 2×), srovnatelné doby načtení modelu a mírně rychlejší dobu predikce jednoho vzorku oproti Pytorch (o 0.008 sekund na osobním počítači, o 0.028 sekund na RPi).

To, že frameworky nemají stejný průběh křivek při trénování modelů navzdory tomu, že mají stejné hyperparametry, může být způsobeno tím, že Keras používá *glorot/xavier\_uniform* pro váhy a nuly pro bias, zatímco PyTorch používá *kaiming\_uniform* pro váhy a jinou uniformní inicializaci pro bias. Dalším důvodem může být různé výchozí nastavení pro optimalizační parametry, to může ovlivnit konvergenci sítě.

Delší doby načtení knihoven frameworku Pytorch mohou být způsobeny několika faktory; velikostí knihoven, jejich komplexností a počtem jejich závislostí. Pytorch je větší než Keras, proto může trvat déle, než se celá knihovna načte. Je také více flexibilní a umožňuje uživatelům vytvářet složitější modely. Dále má Pytorch více závislostí než Keras, což může způsobit delší doby načítání.

Při predikci pro jeden vzorek pomocí Pytorch došlo k tomu, že první predikce trvala značně déle než všechny následující, viz Obrázek 7.1. Zpomalení první predikce může být způsobeno tím, že během prvního průchodu jsou nastaveny různé věci jako cache, paměť na grafické kartě, optimalizace grafu a podobně. Následující predikce jsou pak rychlejší díky tomu, že jsou již využívány předem připravené a optimalizované výpočetní zdroje.



Obrázek 7.1: Doba predikce jednoho vzorku pomocí frameworku Pytorch – 10 realizací

Zanedbáním prvního měření doby predikce by framework Pytorch dosáhl hodnoty  $0.031 \pm 0.001$  sekund na osobním počítači a hodnoty  $0.604 \pm 0.105$  sekund na RPi, čímž by se ještě více přiblížil rychlosti Kerasu, v případě RPi dokonce dosahuje vyšší rychlosti o 0.04 sekundy.

Na základě výsledků tohoto experimentu bylo rozhodnuto, že zvoleným frameworkem pro aplikaci na robotickou entitu bude Pytorch. Delší načtení knihoven nebylo rozhodující, protože to probíhá jen jednorázově. Doba načtení modelu a doba predikce pak byly pro oba frameworky srovnatelné. Pytorch byl zvolen z toho důvodu, že nabízí větší kontrolu a flexibilitu při vytváření neuronových sítí. Pro situace s většími datovými sadami a výkonnostně náročnějšími úlohami pak navíc dosahuje Pytorch obecně vyšší rychlosti než Keras, což je také výhodou, protože použitý Dataset 4.1 slouží jen k základnímu předtrénování sítě a má být rozšiřován vzorky ze zpětné vazby od uživatele.

**Experiment 5.2.** se zabýval výběrem optimálních hyperparametrů pro neuronovou síť implementovanou pomocí Pytorch. Byl kladen důraz na přesnost této sítě a zároveň na její rychlost. Stoprocentní přesnost a nejrychlejší čas (v průměru 0.026 sekund) měla neuronová síť se čtyřiceti pěti neurony ve skryté vrstvě, deseti epochami, učícím koeficientem 0.01, velikostí dávky 40 a optimizérem Adam.

Velká četnost neuronových sítí s přesností mezi 15 % a 20 % na Obrázku 5.2 je zapříčiněna tím, že se tyto neuronové sítě nedokázaly natrénovat pro daný klasifikační problém (daná kombinace hyperparametrů a jejich hodnoty nebyly vhodné) a pro všechny testovací vzorky predikovaly jednu stejnou třídu.

Z analýzy Tabulky 5.3 plyne, že rychlost trénování neuronové sítě s jednou skrytou vrstvou a počtem neuronů v této vrstvě nižším než padesát není ovlivněna tímto počtem neuronů ve skryté vrstvě. Například třetí a pátý model v tabulce mají až na počet neuronů ve skryté vrstvě stejné hyperparametry a třetí model se čtyřiceti pěti neurony ve skryté vrstvě dosahuje lepšího času než pátý model, který jich má třicet pět. Dále lze vyvodit, že pro daný klasifikační problém stačilo neuronové síti menší množství epoch (nejvýše 15) pro to, aby se natrénovala na 100 % a dosahovala dostatečné rychlosti. Naopak batch size nabýval u všech deseti nejlepších neuronových sítí dvou nejvyšších testovaných hodnot – 40 nebo 50. Jako nejlepší hodnota pro koeficient učení se pro danou úlohu jeví 0.01. Nejvhodnějším optimizérem se zdá být Adam.

Celkově lze říci, že všechny modely se natrénovaly velmi rychle (v řádu desítek milisekund) díky využití výkonných algoritmů strojového učení a moderních výpočetních schopností počítače. To umožňuje využití lidské interakce s

trénovacím algoritmem, díky krátkým trénovacím časům je možné rychle iterovat a přeučovat modely, což zvyšuje efektivitu procesu a kvalitu modelů.

**Experiment 5.3.** se zabýval výběrem hyperparametrů neuronové sítě trénované na Datasetu 4.2 a analýzou vlivu architektury sítě (počtu neuronů ve skrytých vrstvách), počtu tříd a počtu vzorků na výslednou přesnost neuronové sítě.

V první části experimentu byly vybrány vhodné hyperparametry pro neuronovou síť, viz Tabulka 5.4. Neuronová síť s těmito hyperparametry dosáhla přesnosti  $0.854 \pm 0.002$  a ztráty  $0.588 \pm 0.015$  na testovacích datech.

Ve druhé části experimentu byla analyzována schopnost neuronových sítí natrénovat se v závislosti na architektuře sítě a na počtu tříd. Z heatmapy výsledků na Obrázku 5.5 lze vidět, že s rostoucím počtem tříd klesá přesnost klasifikace a že síť s větším počtem neuronů ve skrytých vrstvách dosahuje lepších výsledků. U největšího počtu neuronů ve skrytých vrstvách se trend růstu přesnosti začíná obracet, je to pravděpodobně způsobeno tím, že síť je pro danou úlohu již příliš velká a začíná se přetrénovávat (tzv. overfitting). Tato zjištění potvrzují, že při klasifikaci s větším počtem tříd je potřeba mít adekvátně větší neuronovou síť, aby bylo možné naučit síť rozlišovat jednotlivé třídy.

V poslední části experimentu byla analyzována schopnost neuronových sítí natrénovat se v závislosti na počtu tříd a na počtu vzorků v jednotlivých třídách. Z výsledků na Obrázku 5.6 je patrné, že s rostoucím počtem vzorků v jednotlivých třídách se zvyšuje přesnost klasifikace, což potvrzuje, že pro trénování neuronových sítí je důležité mít dostatečné množství trénovacích dat.

**Experiment 5.4.** porovnával chování modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“. Tyto modely byly porovnávány pro dvě modifikace Datasetu 4.2: první modifikace obsahovala 60 tříd, 60 trénovacích dat, 12 validačních dat a 18 testovacích dat, zatímco druhá modifikace měla 130 tříd, 20 trénovacích dat, 4 validační data a 6 testovacích dat. V obou případech dosáhly modely trénované pomocí fine-tuningu a transfer learningu lepších výsledků než model trénovaný „od nuly“. Pro první modifikaci datasetu dosáhly přesnosti 0.89, zatímco model trénovaný „od nuly“ dosáhl přesnosti 0.87, pro druhou modifikaci datasetu pak přesnosti 0.83 a 0.85 a model trénovaný „od nuly“ přesnosti 0.72. Kromě toho měly modely trénované pomocí fine-tuningu a transfer learningu obecně rychlejší průběh růstu přesnosti, respektive poklesu ztráty. Z hlediska poklesu validační ztráty byl rychlejší model trénovaný pomocí fine-tuningu, pro druhý dataset se ale hodnota jeho validační ztráty ustálila na hodnotě o 0.1 vyšší než u modelu trénovaného „od nuly“.

Při použití druhé modifikace datasetu došlo k tomu, že při validaci v posledních přibližně 100 epochách již dále neklesala validační ztráta modelu trénovaného „od nuly“ a ustálila se na hodnotě 1.12. Tato hodnota je o dost vyšší než u zbylých dvou modelů, ty měly výslednou hodnotu 0.64 a 0.54. To, že se hodnota validační ztráty modelu trénovaného „od nuly“ již dále neměnila a zůstala vyšší může být způsobeno tím, že model se pro danou úlohu přetrénoval, protože počet vzorků na třídu byl velmi omezený.

Z výsledků experimentu tedy vyplývá, že využití předtrénovaného modelu může být účinným způsobem, jak dosáhnout lepších výsledků než při trénování „od nuly“. Lze také říci, že v případě řídkého datasetu může být transfer learning lepší volbou než fine-tuning.

**Experiment 5.5.** se zabýval vyhodnocením metody iterativního přeučování modelu na základě zpětné vazby od uživatele. Byla testována přesnost a doba trvání přeučení modelů v závislosti na počtu epoch při využití dvou různých počátečních inicializací datasetu. V prvním případě začínal dataset se dvěma třídami o jednom vzorku, ve druhém případě začínal se šesti třídami o deseti vzorcích.

Výsledky experimentu naznačují, že volba proměnného počtu epoch pro trénování modelu může být klíčová pro dosažení nejvyšší přesnosti. V obou případech byla nejvyšší přesnost dosažena při proměnlivé hodnotě počtu epoch nastavené jako *#vzorků*. Nicméně tato strategie vedla k nejdelšímu přeučování modelu. Dále bylo zjištěno, že při použití příliš nízkého počtu epoch byla přesnost modelu nižší než 80 %. Ukázalo se ale, že přenásobení počtu epoch dvěma může přesnost modelu vylepšit – obzvláště pro málo vzorků, mírně se tím ale zvýší doba trénování modelu. Celkově lze tedy říci, že volba počtu epoch je kompromisem mezi výslednou přesností a dobou přeučování modelu.

Po porovnání přesností v obou případech lze také vyvodit, že rozšíření datasetu o více vzorků při inicializaci datasetu vede k obecnému zvýšení přesnosti modelu.



# Závěr

## 8

Conclusion text...

### 8.1 Future Work

Outlook...







# Dataset D1



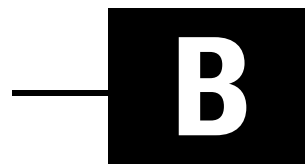
Záměr	Text
POZDRAV	Ahoj
POZDRAV	Dobrý den
POZDRAV	Čau
POZDRAV	Nazdar
POZDRAV	Pozdrav pánbůh
POZDRAV	Zdar
POZDRAV	Dobré ráno
POZDRAV	Dobrý večer
POZDRAV	Dobrej
POZDRAV	Zdravíčko
POKYN	Rozsviť LED
POKYN	Zamávej
POKYN	Otoč se
POKYN	Zapískej
POKYN	Zvedni levou ruku
POKYN	Zvedni pravou ruku
POKYN	Zablikej
POKYN	Jdi vpřed
POKYN	Rozjeď se
POKYN	Zatoč
KALENDÁŘ	Kolik je hodin
KALENDÁŘ	Co mám v plánu zítra odpoledne
KALENDÁŘ	Naplánuj schůzku na pátek ve 3
KALENDÁŘ	Kolikátého je dnes
KALENDÁŘ	Jaký je datum
KALENDÁŘ	Co je za měsíc

(tabulka pokračuje na další stránce)

(pokračování z předchozí stránky)

Záměr	Text
KALENDÁŘ	Jaký je rok
KALENDÁŘ	Co je dnes za den
KALENDÁŘ	Kdo má dnes svátek
KALENDÁŘ	Kdo má dnes narozeniny
VYGOOGLI	Kolik je států v Africe
VYGOOGLI	Najdi mi restauraci v Brně
VYGOOGLI	Jaký je počet obyvatel v Plzni
VYGOOGLI	Kde koupit bitcoin za CZK a jaký je aktuální kurz
VYGOOGLI	Kdo je prezident ve Francii
VYGOOGLI	Kde budou další olympijské hry
VYGOOGLI	Jak vybrat běžky
VYGOOGLI	Jak napsat životopis
VYGOOGLI	Jak rychle zhubnout
VYGOOGLI	Jak vydělat peníze
POZNEJ	Co to je?
POZNEJ	Jaký je to předmět
POZNEJ	Poznej předmět
POZNEJ	Poznej, co to je
POZNEJ	Urči, o co se jedná
POZNEJ	Jaká je to věc
POZNEJ	Co to mám
POZNEJ	Co to držím
POZNEJ	Co vidíš
POZNEJ	Co je tohle
STOP	Vypnout
STOP	Ukončit
STOP	Zastav
STOP	Přestaň
STOP	Konec
STOP	Stop
STOP	Dost
STOP	Vypni se
STOP	Ukonči se
STOP	Zastav se

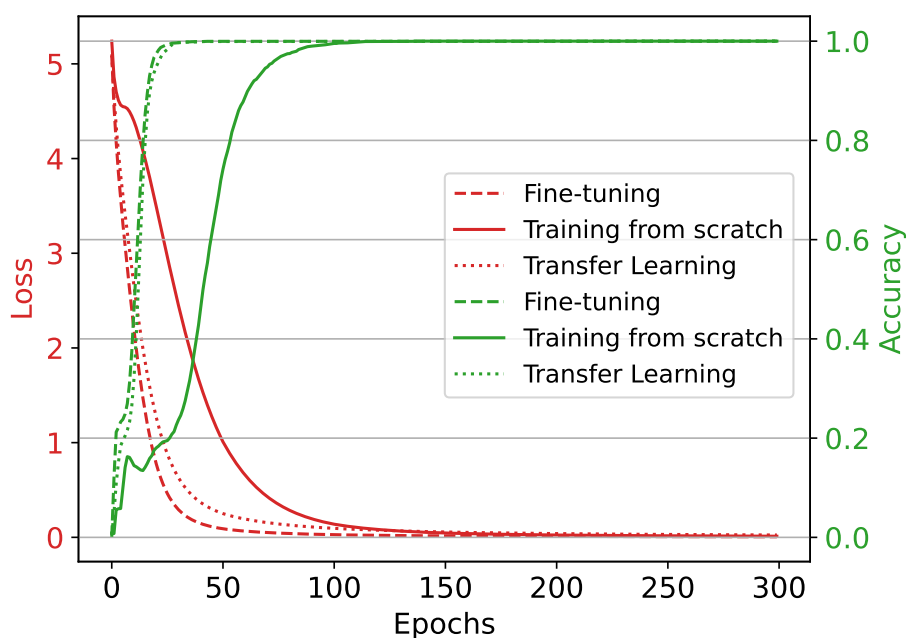
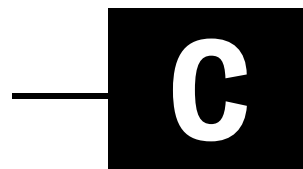
# Testované hyperparametry pro neuronovou síť na RPi



Parametr	Hodnoty
Počet neuronů ve skryté vrstvě	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
Aktivační funkce	(Sigmoid, Softmax)
Learning rate	0.1, 0.01, 0.001, 0.0001
Velikost dávky	5, 10, 20, 30, 40, 50
Počet epoch	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
Optimizér	SGD, Adam



## Graf k experimentu 5.4



Obrázek C.1: Vývoj přesnosti a ztráty při trénování modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 130 tříd



# Vzorový dataset pro učení dialogem



Záměr	Text
OTOČ SE	Otoč se doleva
OTOČ SE	Otoč se doprava
OTOČ SE	Otoč se o 30 stupňů
OTOČ SE	Otoč se o 45 stupňů doprava
OTOČ SE	Otoč se o 45 stupňů doleva
OTOČ SE	Obrať se doleva
OTOČ SE	Obrať se doprava
OTOČ SE	Rotuj o 10 stupňů doleva
OTOČ SE	Rotuj o 120 stupňů doprava
OTOČ SE	Zatoč se dokola
LED	Zablikej LED 1
LED	Zablikej LED 2
LED	Zablikej LED 3
LED	Zablikej LED 4
LED	Zablikej LED 3 a 4
LED	Rozsviř LED 1
LED	Rozsviř všechny LED
DETEKCE SVĚTLA	Odkud svítí sluníčko?
DETEKCE SVĚTLA	Jak moc svítí sluníčko?
DETEKCE SVĚTLA	Jak moc svítí světlo?
DETEKCE SVĚTLA	Odkud svítí světlo?
DETEKCE SVĚTLA	Kde je zdroj světla?
DETEKCE SVĚTLA	Kde je největší jas?
DETEKCE OBJEKTU	Co je tohle?
DETEKCE OBJEKTU	Co to je?

(tabulka pokračuje na další stránce)

(pokračování z předchozí stránky)

Záměr	Text
DETEKCE OBJEKTU	Co je to?
DETEKCE OBJEKTU	Copak je to?
DETEKCE OBJEKTU	Co je tohle za předmět?
DETEKCE OBJEKTU	Poznej, o co se jedná
DETEKCE OBJEKTU	Co je tohle za věc?
SEZNAM SE	Tohle je Anna.
SEZNAM SE	Tohle je Petr.
SEZNAM SE	Tohle je Karel.
SEZNAM SE	Tohle je Jana.
SEZNAM SE	Představuji ti Petra.
SEZNAM SE	Představuji ti Karolínu.
PŘEKLAD	Jak se řekne moře v angličtině?
PŘEKLAD	Jak se řekne auto v němčině?
PŘEKLAD	Jak se řekne stůl v polštině?
PŘEKLAD	Jak se řekne židle v ruštině?
PŘEKLAD	ve španělštině, seznamte se se mnou zítra se říká jak
PŘEKLAD	ve francouzštině, jak to mám říct, uvidíme se později
PŘEKLAD	jak se řekne ahoj v japonštině
PŘEKLAD	jak mohu říct „zrušit objednávku“ francouzsky
PŘEKLAD	jak řeknu španělsky večere
PŘEKLAD	jak se řekne sbohem francouzsky
PŘEKLAD	jak španělsky říct děkuji
PŘEKLAD	jak řeknu sbohem v čínštině
PŘEKLAD	jak mohu říci velmi děkuji v čínštině
PŘEKLAD	Potřebuji vědět, jak pozdravit ve Francii
PŘEKLAD	řekl byste mi, jak se rozloučit ve francii
PŘEKLAD	jak se řekne ahoj v Mexiku
PŘEKLAD	řekni mi, jak pozdravit v chile
PŘEKLAD	Chci vědět, jak se rozloučit ve Francii
PŘEKLAD	přeložil byste mi prosím větu do ruštiny
PŘEKLAD	přeložil byste mi frázi do mandarínštiny
PŘEKLAD	můžete mi toto jméno přeložit do španělštiny, prosím

(tabulka pokračuje na další stránce)



(pokračování z předchozí stránky)

Záměr	Text
PŘEKLAD	mohl byste mi to přeložit do čínštiny, prosím
PŘEKLAD	mohl byste mi přeložit vodu do nizozemštiny
CHAT GPT	Jak se máš?
CHAT GPT	Jak se jmenuješ?
CHAT GPT	Kdo jsi?
CHAT GPT	Máš rád čokoládu?
CHAT GPT	definujte mi prosím peníze
CHAT GPT	řekni mi význam hegemonie
CHAT GPT	co znamená rozhořčení
CHAT GPT	definujte mi prosím ambivalenci
CHAT GPT	prosím definujte institucionální rasismus
CHAT GPT	definuj mi prosím antagonistu
CHAT GPT	co rozlišovat znamená
CHAT GPT	co znamená slovo aliance
CHAT GPT	potřebuji vědět, co znamená dominovat
CHAT GPT	jaká je definice slova migrovat
CHAT GPT	řekněte mi prosím, co znamená paeon
CHAT GPT	chci vědět, co je smyslem života
CHAT GPT	Chci slyšet, o čem si myslíš, že je smyslem života
CHAT GPT	co je podle tebe smyslem života
CHAT GPT	jaký je účel existence
CHAT GPT	jaký je podle vás náš smysl života
CHAT GPT	chtěl bych znát smysl života
CHAT GPT	jaký je tvůj názor na smysl života
CHAT GPT	co je smyslem života
CHAT GPT	řekni mi všechno o smyslu života
CHAT GPT	řekni mi své myšlenky o smyslu života
CHAT GPT	máte nějaké představy o smyslu života?
VYHLEDÁVÁNÍ	Co je hlavní město České republiky.
VYHLEDÁVÁNÍ	Kde je největší hrad v České republice.
VYHLEDÁVÁNÍ	kolik je hodin ve Francii
VYHLEDÁVÁNÍ	kolik je hodin v Londýně právě teď
VYHLEDÁVÁNÍ	kolik je hodin v Londýně
VYHLEDÁVÁNÍ	kolik je hodin
VYHLEDÁVÁNÍ	kolik je hodin v londýně

(tabulka pokračuje na další stránce)

(pokračování z předchozí stránky)

Záměr	Text
VYHLEDÁVÁNÍ	kolik je hodin v pacifickém časovém pásmu
VYHLEDÁVÁNÍ	kolik je hodin v pacifickém standardním čase
VYHLEDÁVÁNÍ	je čokoláda pro vás špatná nebo dobrá
VYHLEDÁVÁNÍ	jsou hranolky zdravé nebo ne
VYHLEDÁVÁNÍ	řekni mi nutriční údaje běžného banánu
VYHLEDÁVÁNÍ	je čokoláda zdravá
VYHLEDÁVÁNÍ	jaké jsou nutriční informace pro krevety
VYHLEDÁVÁNÍ	jaká je aktuální předpověď počasí
VYHLEDÁVÁNÍ	jaké je počasí
VYHLEDÁVÁNÍ	jaká je předpověď na den
VYHLEDÁVÁNÍ	předpověď počasí prosím
VYHLEDÁVÁNÍ	jaké je dnes počasí
VYHLEDÁVÁNÍ	bude zítra pršet
VYHLEDÁVÁNÍ	jaké je venku počasí
VYHLEDÁVÁNÍ	jaké bude zítra počasí
VYHLEDÁVÁNÍ	jaké je počasí v atlantě
VYHLEDÁVÁNÍ	jaké je počasí v tokiu
VYHLEDÁVÁNÍ	jaká je dnes vlhkost
VYHLEDÁVÁNÍ	jaká byla včera teplota

# Bibliografie

- [1] Evan Ackerman a Erico Guizzo. *Wizards of ROS: Willow Garage and the making of the Robot Operating System*. 2022. URL: <https://spectrum.ieee.org/wizards-of-ros-willow-garage-and-the-making-of-the-robot-operating-system>.
- [2] Saugat Bhattarai. *What is gradient descent in machine learning?* 2018. URL: <https://saugatbhattarai.com np/what-is-gradient-descent-in-machine-learning/>.
- [3] James Briggs. *Sentence transformers and embeddings*. 2023. URL: <https://www.pinecone.io/learn/sentence-embeddings/>.
- [4] Jason Brownlee. *How to configure the learning rate when training deep learning neural networks*. 2019. URL: <https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/>.
- [5] Guillaume Desagulier. *Word embeddings: The (very) basics*. 2018. URL: <https://corpling.hypotheses.org/495>.
- [6] *Designing your neural networks*. URL: <https://www.kdnuggets.com/2019/11/designing-neural-networks.html>.
- [7] Sanket Doshi. *Various optimization algorithms for training neural network*. 2020. URL: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>.
- [8] Nagesh Singh Chauhan. *Loss functions in Neural Networks*. 2021. URL: <https://www.theaidream.com/post/loss-functions-in-neural-networks>.
- [9] Francois Chollet et al. *Keras*. 2015. URL: <https://github.com/fchollet/keras>.
- [10] Jongkil Kim, Jonathon M. Smereka, Calvin Cheung, Surya Nepal a Marthie Grobler. „Security and Performance Considerations in ROS 2: A Balancing Act“. In: *CoRR* abs/1809.09566 (2018). arXiv: 1809.09566. URL: <http://arxiv.org/abs/1809.09566>.

- [11] Stefan Larson et al. „An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction“. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, lis. 2019, s. 1311–1316. DOI: 10.18653/v1/D19-1131. URL: <https://aclanthology.org/D19-1131>.
- [12] Jan Lehecka a Jan Svec. „Comparison of Czech Transformers on Text Classification Tasks“. In: *CoRR* abs/2107.10042 (2021). arXiv: 2107.10042. URL: <https://arxiv.org/abs/2107.10042>.
- [13] Lian Meng a Minlie Huang. „Dialogue Intent Classification with Long Short-Term Memory Networks“. In: led. 2018, s. 42–50. ISBN: 978-3-319-73617-4. DOI: 10.1007/978-3-319-73618-1\_4.
- [14] Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán a Ángel Fernández-Leal. „Human-in-the-loop machine learning: a state of the art“. In: *Artificial Intelligence Review* 56 (srp. 2022). DOI: 10.1007/s10462-022-10246-w.
- [15] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029. URL: [https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr\\_1\\_2?ie=UTF8&qid=1336857747&sr=8-2](https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2).
- [16] Michael A Nielsen. *Neural networks and deep learning*. Sv. 25. Determination press San Francisco, CA, USA, 2015.
- [17] Adam Paszke et al. „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems* 32. Curran Associates, Inc., 2019, s. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [18] Morgan Quigley, Brian Gerkey a William D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. 1st. O'Reilly Media, Inc., 2015. ISBN: 1449323898.
- [19] Morgan Quigley et al. „ROS: an open-source Robot Operating System“. In: *ICRA Workshop on Open Source Software*. 2009.
- [20] Pranoy Radhakrishnan. *What are hyperparameters ? and how to tune the hyperparameters in a deep neural network?* 2017. URL: <https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a>.

- [21] Suman Ravuri a Andreas Stoicke. „A comparative study of neural network models for lexical intent classification“. In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. 2015, s. 368–374. DOI: 10.1109/ASRU.2015.7404818.
- [22] Nils Reimers a Iryna Gurevych. „Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks“. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, lis. 2019, s. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: <https://aclanthology.org/D19-1410>.
- [23] Kurnia Rendy. *Tuning the hyperparameters and layers of neural network deep learning*. 2022. URL: <https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/>.
- [24] David E. Rumelhart, Geoffrey E. Hinton a Ronald J. Williams. „Learning representations by back-propagating errors“. In: *Nature* 323 (1986), s. 533–536.
- [25] Nikita Sharma. *Exploring optimizers in machine learning*. 2021. URL: <https://heartbeat.comet.ml/exploring-optimizers-in-machine-learning-7f18d94cd65b>.
- [26] Sagar Sharma, Simone Sharma a Anidhya Athaiya. „Activation functions in neural networks“. In: *Towards Data Sci* 6.12 (2017), s. 310–316.
- [27] *SpeechTech* — [speechtech.cz](https://www.speechtech.cz/). <https://www.speechtech.cz/>. [Accessed 25-Feb-2023].
- [28] Stanford Artificial Intelligence Laboratory et al. *actionlib*. 2018. URL: <http://wiki.ros.org/actionlib>.
- [29] Ilya Sutskever, James Martens, George Dahl a Geoffrey Hinton. „On the importance of initialization and momentum in deep learning“. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. Sanjoy Dasgupta a David McAllester. Sv. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, s. 1139–1147. URL: <https://proceedings.mlr.press/v28/sutskever13.html>.
- [30] Jan Švec, Petr Neduchal a Marek Hruz. „Multi-modal communication system for mobile robot“. In: *IFAC-PapersOnLine* 55.4 (2022). 17th IFAC Conference on Programmable Devices and Embedded Systems PDES 2022 — Sarajevo, Bosnia and Herzegovina, 17-19 May 2022, s. 133–138. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2022.06.022>. URL: <https://www.sciencedirect.com/science/article/pii/S2405896322003378>.

- [31] Nima Tajbakhsh et al. „Convolutional neural networks for medical image analysis: Full training or fine tuning?“ In: *IEEE transactions on medical imaging* 35.5 (2016), s. 1299–1312.
- [32] Ashish Vaswani et al. „Attention Is All You Need“. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: <http://arxiv.org/abs/1706.03762>.
- [33] Eva Volná. *Neuronové sítě 1*. 2008.
- [34] Yufan Wang, Jiawei Huang, Tingting He a Xinhui Tu. „Dialogue intent classification with character-CNN-BGRU networks“. In: *Multimedia Tools and Applications* 79 (ún. 2020). DOI: 10.1007/s11042-019-7678-1.
- [35] Vishal Yathish. *Loss functions and their use in neural networks*. 2022. URL: <https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9>.
- [36] Aston Zhang, Zachary C. Lipton, Mu Li a Alexander J. Smola. „Dive into Deep Learning“. In: *CoRR* abs/2106.11342 (2021). arXiv: 2106.11342. URL: <https://arxiv.org/abs/2106.11342>.
- [37] XueFei Zhou. „Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation“. In: *Journal of Physics: Conference Series* 1004.1 (2018), s. 012028. DOI: 10.1088/1742-6596/1004/1/012028. URL: <https://dx.doi.org/10.1088/1742-6596/1004/1/012028>.

# Seznam obrázků

2.1	Diagram architektury neuronové sítě (převzato z [6]) . . . . .	7
2.2	Aktivační funkce . . . . .	9
2.3	Vliv učícího koeficientu při gradientním sestupu (převzato z [2]) . . . .	13
2.4	Vektorový prostor sedmi slov ve třech kontextech (převzato z [5]) . . .	16
2.5	Diagram fine-tuningu (převzato z [36]) . . . . .	18
2.6	Diagram architektury SpeechCloudu (převzato z [30]) . . . . .	21
3.1	Napojení Sentence Transformeru na neuronovou síť . . . . .	26
3.2	Nastínění propojení jednotlivých částí v rámci celé aplikace . . . . .	27
3.3	Celkový náhled na uživatelské rozhraní . . . . .	28
3.4	Zvýraznění predikovaného záměru . . . . .	29
3.5	Korekce predikovaného záměru na záměr „Nová třída“ . . . . .	29
3.6	Dokončení přeučení, možnost odstranění chybných vzorků . . . . .	30
5.1	Průběh trénování neuronové sítě v Pytorch a Keras . . . . .	34
5.2	Histogram četností modelů v závislosti na jejich přesnosti . . . . .	35
5.3	Boxplot doby trénování jednotlivých modelů . . . . .	36
5.4	Průběh trénování neuronové sítě s optimálními hyperparametry pro 10 realizací . . . . .	37
5.5	Přesnost neuronové sítě v závislosti na její architektuře (počtu neuronů ve skrytých vrstvách) a počtu tříd . . . . .	39
5.6	Přesnost neuronové sítě v závislosti na počtu tříd a počtu vzorků na třídu	40
5.7	Vývoj přesnosti a ztráty při trénování modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 60 tříd . . . . .	41
5.8	Vývoj ztráty při validaci modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 60 tříd . . . . .	42

5.9	Vývoj ztráty při validaci modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 130 tříd . . . . .	43
5.10	Iterativní rozšiřování datasetu při učení dialogem pro dvě počáteční třídy	44
5.11	Vývoj přesnosti přeučeného modelu a doby trvání jeho přeučení [s] v jednotlivých krocích při učení dialogem pro dvě počáteční třídy . . . .	45
5.12	Iterativní rozšiřování datasetu při učení dialogem pro šest počátečních tříd . . . . .	46
5.13	Vývoj přesnosti přeučeného modelu a doby trvání jeho přeučení [s] v jednotlivých krocích při učení dialogem pro šest počátečních tříd . . .	46
6.1	Hardwarové propojení jednotlivých součástí požitých na robotické entitě . . . . .	50
6.2	ROS architektura robotické entity . . . . .	50
6.3	Fotografie výsledné robotické entity . . . . .	51
7.1	Doba predikce jednoho vzorku pomocí frameworku Pytorch – 10 realizací	54
C.1	Vývoj přesnosti a ztráty při trénování modelu trénovaného pomocí fine-tuningu, modelu trénovaného pomocí transfer learningu a modelu trénovaného „od nuly“ pro 130 tříd . . . . .	65



# Seznam tabulek

4.1	Statistiky datasetu 1 . . . . .	31
4.2	Statistiky datasetu 2 . . . . .	32
5.1	Hyperparametry neuronových sítí použitých v experimentu pro výběr frameworku . . . . .	33
5.2	Doby načtení knihoven, načtení modelů a predikcí jednoho vzorku pro frameworky Pytorch a Keras (v sekundách). . . . .	34
5.3	Hyperparametry deseti nejrychlejších neuronových sítí a jejich doby trvání trénování, koeficient učení byl ve všech případech 0.01 a optimizér Adam . . . . .	36
5.4	Hyperparametry neuronové sítě pro klasifikaci záměru trénované na Datasetu 4.2 . . . . .	38