TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DETEKCIA TOXICKÉHO OBSAHU NA WEBE I

Príloha B - Systémová príručka

Študijný program:	Hospodárska informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra kybernetiky a umelej inteligencie (KKUI)
Školiteľ:	Doc. Ing. Peter Bednár, PhD.

2025 Košice **Hryhorii Tiutchenko**

Obsah

Zoznam obrázkov 3

Zoznam tabuliek 4

1. Systémová príručka 5

2. DistilBERT na slovenskom súbore údajov 6

2.1. Príprava a spustenie projektu 6

2.1.1. Technológie 6

2.1.2. Inštalácia závislostí 6

2.1.3. Hlavné komponenty a funkcie 6

3. DistilBERT na anglickom súbore údajov 10

3.1. Príprava a spustenie projektu 10

3.1.1. Hlavné komponenty a funkcie 11

4. LSTM na slovenskom súbore údajov 14

4.1. Príprava a spustenie projektu 14

4.1.1. Technológie 14

4.1.2. Hlavné komponenty a funkcie 15

5. LSTM na anglickom súbore údajov 19

5.1. Príprava a spustenie projektu 19

5.1.1. Hlavné komponenty a funkcie 19

Zoznam obrázkov

Obr. 1 Knižnice pre projekt DistilBERT na slovenskom súbore údajov.....6
Obr. 2 Načítanie a rozdelenie údajov DistilBERT na slovenskom súbore údajov.....6
Obr. 3 Definícia datasetu a DataLoader DistilBERT na slovenskom súbore údajov 7
Obr. 4 Model a optimalizátor DistilBERT na slovenskom súbore údajov 7
Obr. 5 Cyklus učenia a hodnotenia(1) DistilBERT na slovenskom súbore údajov 8
Obr. 6Cyklus učenia a hodnotenia(2) DistilBERT na slovenskom súbore údajov 9
Obr. 7 Rozdelenie na train/val/test DistilBERT na anglickom súbore údajov.....10
Obr. 8 Tokenizácia a súbor údajov DistilBERT na anglickom súbore údajov 10
Obr. 9 DataLoader DistilBERT na anglickom súbore údajov 10
Obr. 10 Definícia modelu a optimalizátora DistilBERT na anglickom súbore údajov 11
Obr. 11Definícia train_Model(1) DistilBERT na anglickom súbore údajov 11
Obr. 12Definícia train_Model(2) DistilBERT na anglickom súbore údajov.....12
Obr. 13Definícia train_Model(3) DistilBERT na anglickom súbore údajov.....13
Obr. 14 Trénovanie modelu DistilBERT na anglickom súbore údajov 13
Obr. 15 Načítanie a rozdelenie údajov LSTM na slovenskom súbore údajov.....14
Obr. 16Tokenizácia LSTM na slovenskom súbore údajov 15
Obr. 17 FastText Embeddings LSTM na slovenskom súbore údajov.....15
Obr. 18 Padding modelu LSTM na slovenskom súbore údajov.....16
Obr. 19 Tvorba modelu LSTM na slovenskom súbore údajov 17
Obr. 20 Kolbeky LSTM na slovenskom súbore údajov.....18
Obr. 21 Trénovanie modelu LSTM na slovenskom súbore údajov 18
Obr. 22 Zmiešaná presnosť a jemné doladenie vektorizátora LSTM na anglickom súbore údajov 19
Obr. 23 tf.data.Dataset pipeline a rozdelenie LSTM na anglickom súbore údajov 20
Obr. 24 Definícia a kompilácia modelu LSTM na anglickom súbore údajov 21
Obr. 25 Trénovanie modelu LSTM na anglickom súbore údajov 21

Zoznam tabuliek

1. Systémová príručka

Táto systémová príloha predstavuje štyri projekty s modelmi hlbokého učenia založenými na DistilBERT a LSTM.

2. DistilBERT na slovenskom súbore údajov

2.1. Príprava a spustenie projektu

Výskum sa uskutočnil pomocou programu Pycharm a prostredia 3proj, v ktorom boli nainštalované CUDA verzia: 12.1 a cuDNN verzia: 90100.

2.1.1. Technológie

Python 3.10.13

PyTorch

Hugging Face Transformers

scikit-learn

tqdm

NumPy,

Matplotlib

```
import json
import random
import torch
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from torch.utils.data import Dataset, DataLoader
from torch.optim import AdamW
from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification
from tqdm import tqdm
[24]
```

Obr. 1 Knižnice pre projekt DistilBERT na slovenskom súbore údajov

2.1.2. Hlavné komponenty a funkcie

```
with open('slovakdata/train.json', 'r', encoding='utf-8') as f:
    train_data = [json.loads(line) for line in f]
with open('slovakdata/test.json', 'r', encoding='utf-8') as f:
    test_data = [json.loads(line) for line in f]


train_samples, val_samples = train_test_split(train_data, test_size=0.1, random_state=42)

[27]
```

Obr. 2 Načítanie a rozdelenie údajov DistilBERT na slovenskom súbore údajov

Súbory train.json a test.json, ktorých každý riadok obsahuje samostatný objekt JSON, sa načítajú a zhromaždia do zoznamov train_data a test_data.

train_data sa rozdelia pomocou funkcie train_test_split na trénovaciu množinu (90 %) a validačnú množinu (10 %) s pevným random_state=42 kvôli reprodukovateľnosti.

```python
# 2. Dataset class
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

class ToxicDataset(Dataset):
    def __init__(self, samples, tokenizer, max_len=128):
        texts = [s['text'] for s in samples]
        labels = [s['label'] for s in samples]
        enc = tokenizer(texts, padding=True, truncation=True, max_length=max_len, return_tensors='pt')
        self.input_ids = enc['input_ids']
        self.attn_mask = enc['attention_mask']
        self.labels = torch.tensor(labels)

    def __len__(self): return len(self.labels)

    def __getitem__(self, idx):
        return {
            'input_ids': self.input_ids[idx],
            'attention_mask': self.attn_mask[idx],
            'labels': self.labels[idx]
        }

# Create datasets and loaders
batch_size = 16
datasets = {
    'train': ToxicDataset(train_samples, tokenizer),
    'val': ToxicDataset(val_samples, tokenizer),
    'test': ToxicDataset(test_data, tokenizer)
}
dataloaders = {k: DataLoader(v, batch_size=batch_size, shuffle=(k == 'train')) for k, v in datasets.items()}
```

Obr. 3 Definícia datasetu a DataLoader DistilBERT na slovenskom súbore údajov

Inicializuje DistilBertTokenizerFast (model bez distilbertovej bázy).

Trieda ToxicDataset dedí Dataset, extrahuje texty a štítky z odovzdaných vzoriek v __init__, tokenizuje texty (padding, truncation, max_len=128) a ukladá input_ids, attention_mask a labels ako tenzory; implementuje __len__ a __getitem__.

Vytvoria sa tri inštancie ToxicDataset pre train/val/test a zabalia sa do DataLoader s batch_size=16, pričom náhodný výber (shuffle=True) je povolený len pre train.

```python
# 3. Model, optimizer
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
torch.backends.cudnn.enabled   = True
torch.backends.cudnn.benchmark = True
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased').to(device)
optimizer = AdamW(model.parameters(), lr=2e-5)

epochs = 5
history = {'train_loss': [], 'val_loss': [], 'train_acc': [], 'val_acc': [], 'test_loss': [], 'test_acc': []}
[29]
```

Obr. 4 Model a optimalizátor DistilBERT na slovenskom súbore údajov

Načíta predtrénovaný model DistilBertForSequenceClassification.

Konfiguruje históriu AdamW a metriky.

```python
epochs_range = range(1, epochs + 1)
for epoch in epochs_range:
    print(f"\n=== Starting Epoch {epoch}/{epochs} ===", flush=True)
    # 4. Train
    model.train()
    train_losses, train_preds, train_labels = [], [], []
    for batch in tqdm(dataloaders['train'], desc=f"Epoch {epoch}/{epochs} - Training", leave=False):
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attn = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids, attention_mask=attn, labels=labels)
        loss = outputs.loss
        logits = outputs.logits
        loss.backward()
        optimizer.step()
        train_losses.append(loss.item())
        train_preds += torch.argmax(logits, dim=1).cpu().tolist()
        train_labels += labels.cpu().tolist()
    train_loss = np.mean(train_losses)
    train_acc = accuracy_score(train_labels, train_preds)
    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
    print(f"Epoch {epoch} Training | loss={train_loss:.4f}, acc={train_acc:.4f}", flush=True)
```

Obr. 5 Cyklus učenia a hodnotenia(1) DistilBERT na slovenskom súbore údajov

Cyklus epochs_range uvedie model do režimu trénovania a pre každú dávku z dataloaders['train'] vykoná zero_grad, spustí input_ids a attention_mask cez model so štítkami, získa stratu a logity, vykoná loss.backward() a optimizer.step().

Po každom kroku sa strata.item() uloží do train_losses a predpovede (argmax podľa logits) a pravdivé štítky sa zhromaždia do zoznamov na výpočet metrík.

Na konci epochy sa vypočíta priemerná train_loss (dávkový priemer) a train_acc (prostredníctvom accuracy_score), pridajú sa do histórie a vypíšu sa na konzolu.

```
# Validation
model.eval()
val_losses, val_preds, val_labels = [], [], []
for batch in tqdm(dataloaders['val'], desc=f"Epoch {epoch}/{epochs} - Validation", leave=False):
    with torch.no_grad():
        input_ids = batch['input_ids'].to(device)
        attn = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids, attention_mask=attn, labels=labels)
        val_losses.append(outputs.loss.item())
        val_preds += torch.argmax(outputs.logits, dim=1).cpu().tolist()
        val_labels += labels.cpu().tolist()
val_loss = np.mean(val_losses)
val_acc = accuracy_score(val_labels, val_preds)
history['val_loss'].append(val_loss)
history['val_acc'].append(val_acc)
print(f"Epoch {epoch} Validation | loss={val_loss:.4f}, acc={val_acc:.4f}", flush=True)

# Test
test_losses, test_preds, test_labels = [], [], []
for batch in tqdm(dataloaders['test'], desc=f"Epoch {epoch}/{epochs} - Testing", leave=False):
    with torch.no_grad():
        input_ids = batch['input_ids'].to(device)
        attn = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids, attention_mask=attn, labels=labels)
        test_losses.append(outputs.loss.item())
        test_preds += torch.argmax(outputs.logits, dim=1).cpu().tolist()
        test_labels += labels.cpu().tolist()
test_loss = np.mean(test_losses)
test_acc = accuracy_score(test_labels, test_preds)
history['test_loss'].append(test_loss)
history['test_acc'].append(test_acc)
print(f"Epoch {epoch} Testing    | loss={test_loss:.4f}, acc={test_acc:.4f}", flush=True)
```

Obr. 6Cyklus učenia a hodnotenia(2) DistilBERT na slovenskom súbore údajov

Model sa uvedie do režimu eval(), potom sa spustí validačný dataloader bez gradientov: kumulujú sa straty val_loss, predpovede a pravdivé značky, z ktorých sa vypočítajú priemerné straty val_loss a presnosť val_acc, uložia sa do histórie a vypíšu na konzolu.

Podobný postup sa vykonáva pre testovací dataloader: zhromažďujú sa test_losses, test_predpovede, test_značky, vypočítajú sa priemerné test_loss a test_acc, ktoré sa tiež pridajú do histórie a na výstup. Výsledná história obsahuje validačné a testovacie metriky pre každú epochu.

3. DistilBERT na anglickom súbore údajov

3.1. Príprava a spustenie projektu

Vo fáze prípravy a spustenia projektu je všetko rovnaké ako v prípade SlovakDistilBERT

Načítanie a základné predbežné spracovanie údajov sa vykonáva od 3 do 7 buniek

```
# Separating data into training , validation and testing sets
from sklearn.model_selection import train_test_split


# we will pass "total_classes" column of the dataset to "stratify" parameter for the even distribution of data
X_rest, X_test, Y_rest, Y_test = train_test_split(Final_data.iloc[:,1],Final_data.iloc[:,2:], test_size=0.1, stratify=Final_data.iloc[:,9])
X_train, X_val, Y_train, Y_val = train_test_split(X_rest,Y_rest, test_size=0.1, stratify=Y_rest.iloc[:,-1])
```

Obr. 7 Rozdelenie na train/val/test DistilBERT na anglickom súbore údajov

```
# Creating Dataset class for Toxic comments and Labels
class Toxic_Dataset(Dataset):
    def __init__(self, Comments_, Labels_):
        self.comments = Comments_.copy()
        self.labels = Labels_.copy()

        self.comments["comment_text"] = self.comments["comment_text"].map(lambda x: tokenizer(x, padding="max_length", truncation=True, return_tensors="pt"))

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        comment = self.comments.loc[idx,"comment_text"]
        label = np.array(self.labels.loc[idx,:])

        return comment, label
```

Obr. 8 Tokenizácia a súbor údajov DistilBERT na anglickom súbore údajov

Trieda Toxic_Dataset dedí Dataset a pri inicializácii kopíruje DataFrame komentárov (Comments_) a štítkov (Labels_) a potom tokenizuje stĺpec comment_text, pričom zohľadňuje max_length, padding a truncation.

Metóda __len__ vráti počet vzoriek podľa dĺžky self.labels.

Metóda __getitem__ načíta tokenizovaný komentár zo self.comments.loc[idx, „comment_text“] a príslušný štítok ako pole NumPy zo self.labels.loc[idx, :] podľa indexu.

```
# Making Training, Testing and Validation of data using Dataset class
Train_data = Toxic_Dataset(X_train, Y_train)
Test_data = Toxic_Dataset(X_test, Y_test)
Val_data = Toxic_Dataset(X_val, Y_val)

# Making datasets into batches
Train_Loader = DataLoader(Train_data, batch_size=32, shuffle=True)
Test_Loader = DataLoader(Test_data, shuffle=True)        #batch_size=16,
Val_Loader = DataLoader(Val_data, shuffle=True)          #batch_size=16,
✓ [37] 2m 7s
```

Obr. 9 DataLoader DistilBERT na anglickom súbore údajov

Model inicializuje tri súbory Toxic_Datasets: na trénovanie (X_train, Y_train), testovanie (X_test, Y_test) a validáciu (X_val, Y_val).

Potom zabalí tieto súbory údajov do DataLoader: pre Train_Loader s parametrami batch_size=32 a shuffle=True, pre Test_Loader a Val_Loader - s predvolenou veľkosťou dávky (komentár uvádza 16) a shuffle=True.

Výsledkom je, že model dostane tri dávky loaderov pripravené na odoslanie do fáz trénovania, validácie a testovania.

3.1.1. Hlavné komponenty a funkcie

```
# DistilBERT
from transformers import DistilBertForSequenceClassification

Distil_bert = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")     Distil_bert

Distil_bert.classifier = nn.Sequential(
                    nn.Linear(768,7),
                    nn.Sigmoid()
                  )
print(Distil_bert)
✓ [39] 13s 48ms
```

Obr. 10 Definícia modelu a optimalizátora DistilBERT na anglickom súbore údajov

Model načíta predtrénovanú klasifikáciu DistilBertForSequenceClassification so základom bez distilbertovej bázy.

Hlava klasifikátora je nahradená postupnosťou vrstiev: lineárnou vrstvou 768 až 7 neurónov a Sigmoidovou aktivačnou funkciou.

```
from torch.optim import Adam
from tqdm import tqdm
from torch.nn import BCELoss
from torch.optim.lr_scheduler import StepLR

def train_Model(model,Train_DL, Val_DL, learning_rate, epochs):

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    Loss = BCELoss()
    Optimizer = Adam(params=model.parameters(), lr=learning_rate)
    scheduler = StepLR(Optimizer, step_size=212, gamma=0.1)

    model.to(device)
    model.train()

    train_acc_epochs = []
    train_loss_epochs = []
    val_acc_epochs = []
    val_loss_epochs = []

    for epoch in range(epochs):
        training_loss = {}
        training_accuracy = {}
        validation_loss = {}
        validation_accuracy = {}
        batch = 0

```

Obr. 11Definícia train_Model(1) DistilBERT na anglickom súbore údajov

Model definuje funkciu train_Model(model, Train_DL, Val_DL, learning_rate, epochs), vyberá zariadenie (cuda alebo cpu), nastavuje kritérium BCELoss, Adamov optimalizátor a StepLR LR shadulator.

Model prenesie sieť do zariadenia, uvedie ju do režimu train() a inicializuje zoznamy pre metriky podľa epoch (train_acc_epochs, train_loss_epochs, val_acc_epochs, val_loss_epochs).

Vnútri cyklu podľa range(epochs) model vynuluje slovníky training_loss, training_accuracy, validation_loss, validation_accuracy a počítadlo dávok pred vykonaním krokov trénovania a validácie.

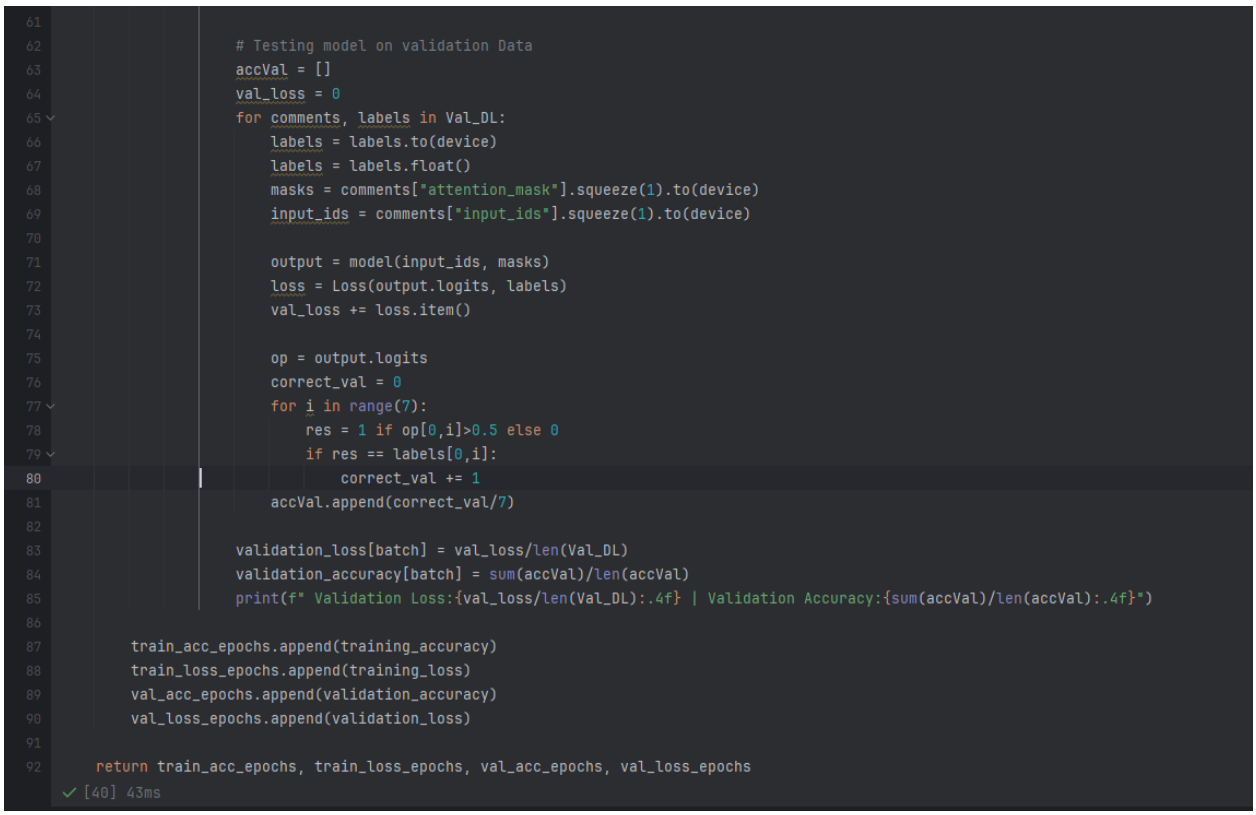
```
    for comments, labels in tqdm(Train_DL):

        labels = labels.to(device)
        labels = labels.float()
        masks = comments["attention_mask"].squeeze(1).to(device) # the model used these masks to attend only to the non-padded tokens in the sequence
        input_ids = comments["input_ids"].squeeze(1).to(device) # contains the tokenized and indexed representation for a batch of comments
        # squeeze is used to remove the second dimension which has size 1.
        output = model(input_ids, masks) # vector of logits for each class
        loss = Loss(output.logits, labels) # compute the loss

        Optimizer.zero_grad()
        loss.backward()
        Optimizer.step()
        scheduler.step()


        batch += 1
        if batch%53 == 0:
            with torch.no_grad():
                acc = []
                op = output.logits
                for lb in range(len(labels)): # note: labels is of shape (batch_size, num_classes(=7))
                    correct = 0
                    for i in range(len(labels[lb])):  # therefore len(labels[lb]) is 7
                        res = 1 if op[lb,i]>0.5 else 0
                        if res == labels[lb,i]:
                            correct += 1
                    acc.append(correct/len(labels[lb]))

                training_loss[batch] = loss.item()
                training_accuracy[batch] = sum(acc)/len(acc)
                print(f"Epoch:{epoch+1} | batch no:{batch}/{len(Train_DL)} | Loss:{loss.item():.4f} | Accuracy:{sum(acc)/len(acc):.4f}")
```

Obr. 12Definícia train_Model(2) DistilBERT na anglickom súbore údajov

Pre každú dávku model extrahuje input_ids a attention_mask, prenesie ich spolu so štítkami do zariadenia, vykoná priamy priechod a spočíta BCELoss podľa logitov a pravdivých štítkov.

Po strate.backward() model aktualizuje parametre prostredníctvom optimiser.step() a zníži lr prostredníctvom scheduler.step().

Každých päť dávok model vypočíta priebežnú presnosť porovnaním logitov s prahovou hodnotou 0,5, uloží training_loss[dávka] a training_accuracy[dávka] a na konzolu vypíše číslo dávky, hodnotu straty a presnosti.



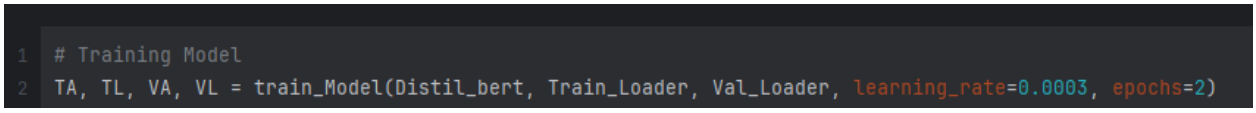
```
                # Testing model on validation Data
                accVal = []
                val_loss = 0
                for comments, labels in Val_DL:
                    labels = labels.to(device)
                    labels = labels.float()
                    masks = comments["attention_mask"].squeeze(1).to(device)
                    input_ids = comments["input_ids"].squeeze(1).to(device)

                    output = model(input_ids, masks)
                    loss = Loss(output.logits, labels)
                    val_loss += loss.item()

                    op = output.logits
                    correct_val = 0
                    for i in range(7):
                        res = 1 if op[0,i]>0.5 else 0
                        if res == labels[0,i]:
                            correct_val += 1
                    accVal.append(correct_val/7)

                validation_loss[batch] = val_loss/len(Val_DL)
                validation_accuracy[batch] = sum(accVal)/len(accVal)
                print(f" Validation Loss:{val_loss/len(Val_DL):.4f} | Validation Accuracy:{sum(accVal)/len(accVal):.4f}")

        train_acc_epochs.append(training_accuracy)
        train_loss_epochs.append(training_loss)
        val_acc_epochs.append(validation_accuracy)
        val_loss_epochs.append(validation_loss)

    return train_acc_epochs, train_loss_epochs, val_acc_epochs, val_loss_epochs
✓ [40] 43ms
```

Obr. 13Definícia train_Model(3) DistilBERT na anglickom súbore údajov

Model spustí validačný DataLoader v režime eval(), vypočíta logity pre každú dávku, vypočíta BCELoss a kumuluje súčet strát vo val_loss.

Pre každú vzorku porovná logity s pravdivými značkami v multilabelovom prahu 0,5, vypočíta podiel správne predpovedaných tried (correct_val/7) a uloží ho do accVal.

Na konci cyklu model vypočíta priemernú validačnú_stratu a validačnú_presnosť, uloží ich do slovníkov podľa čísla dávky, vypíše ich a pridá do zoznamov epochových metrík a potom vráti všetky štyri zoznamy (train_acc_epochs, train_loss_epochs, val_acc_epochs, val_loss_epochs).



```
# Training Model
TA, TL, VA, VL = train_Model(Distil_bert, Train_Loader, Val_Loader, learning_rate=0.0003, epochs=2)
```

Obr. 14 Trénovanie modelu DistilBERT na anglickom súbore údajov

Model zavolá funkciu train_Model a odovzdá jej sieť Distil_bert, načítavacie dávky na trénovanie a validáciu (Train_Loader, Val_Loader), rýchlosť trénovania 0,0003 a počet epoch 2.

Funkcia vráti štyri zoznamy metrík: TA (presnosť trénovania), TL (strata pri trénovaní), VA (presnosť validácie) a VL (strata pri validácii).

4. LSTM na slovenskom súbore údajov

4.1. Príprava a spustenie projektu

Výskum sa uskutočnil pomocou programu Pycharm a prostredia 2proj, v ktorom boli nainštalované CUDA verzia: 11.2.2 a cuDNN verzia: 8.1.0.77.

4.1.1. Technológie

Python 3.10.13

TensorFlow 2.10.0

Scikit-learn,

Pandas,

NumPy

Matplotlib,

Seaborn

```
# 1. Load Slovak data
with open('slovakdata/train.json', 'r', encoding='utf-8') as f:
    train_data = [json.loads(line) for line in f]
with open('slovakdata/test.json', 'r', encoding='utf-8') as f:
    test_data = [json.loads(line) for line in f]

texts = [d['text'] for d in train_data]
labels = [d['label'] for d in train_data]

# 2. Split train into train/val
texts_train, texts_val, y_train, y_val = train_test_split(
    texts, labels, test_size=0.1, random_state=42)


```

Obr. 15 Načítanie a rozdelenie údajov LSTM na slovenskom súbore údajov

4.1.2. Hlavné komponenty a funkcie

```
# 3. TokenizationQ
vocab_size = 20000
max_len = 150
tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
tokenizer.fit_on_texts(texts_train)
[35]
```

Obr. 16Tokenizácia LSTM na slovenskom súbore údajov

Model špecifikuje parametre tokenizácie: maximálna veľkosť vocab_size=20000 a maximálna dĺžka sekvencie max_len=150.

Model inicializuje objekt Tokenizer s obmedzením počtu slov a špeciálnym tokenom <OOV> pre neznáme slová.

Model trénuje tokenizér na korpuse cvičných textov texts_train, pričom naplní slovník a priradí indexy tokenom.

```
import numpy as np
# 1) пробегаем по .vec и собираем словарь: слово → вектор
embedding_index = {}
with open('cc.sk.300.vec', 'r', encoding='utf-8', errors='ignore') as f:
    next(f)  # если в первой строке указаны размеры (иногда fastText добавляет header)
    for line in f:
        values = line.rstrip().split(' ')
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs

# 2) создаём матрицу (vocab_size × embedding_dim)
embedding_dim = 100
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, idx in tokenizer.word_index.items():
    if idx < vocab_size:
        vector = embedding_index.get(word)
        if vector is not None:
            embedding_matrix[idx] = vector
```

Obr. 17 FastText Embeddings LSTM na slovenskom súbore údajov

Model otvorí súbor cc.sk.300.vec, pričom preskočí prípadnú hlavičku, a vytvorí slovník embedding_index.

Model potom vytvorí nulovú embedding_maticu v tvare (vocab_size, embedding_dim).

Model iteruje nad tokenizer.word_index a pre každé idx < vocab_size nahradí vektor z embedding_index do embedding_matrix[idx], ak je k dispozícii.

```
seq_train = tokenizer.texts_to_sequences(texts_train)
seq_val   = tokenizer.texts_to_sequences(texts_val)
seq_test  = tokenizer.texts_to_sequences([d['text'] for d in test_data])

pad_train = pad_sequences(seq_train, maxlen=max_len, padding='post')
pad_val   = pad_sequences(seq_val,   maxlen=max_len, padding='post')
pad_test  = pad_sequences(seq_test,  maxlen=max_len, padding='post')


batch_size = 32

train_ds = tf.data.Dataset.from_tensor_slices((pad_train, np.array(y_train)))
train_ds = (
    train_ds
    .shuffle(10_000)
    .batch(batch_size)
    .cache()
    .prefetch(tf.data.AUTOTUNE)
)

val_ds = tf.data.Dataset.from_tensor_slices((pad_val, np.array(y_val)))
val_ds = (
    val_ds
    .batch(batch_size)
    .cache()
    .prefetch(tf.data.AUTOTUNE)
)

test_ds = tf.data.Dataset.from_tensor_slices(
    (pad_test, np.array([d['label'] for d in test_data]))
)
```

Obr. 18 Padding modelu LSTM na slovenskom súbore údajov

Model konvertuje texty_train/val a textové polia z test_data na číselné sekvencie pomocou tokenizer.texts_to_sequences a následne ich vypĺňa na dĺžku max_len (pad_sequences).

Model vytvorí tf.data.Dataset.from_tensor_slices pre trénovacie, validačné a testovacie údaje kombináciou vstupných a label tenzorov.

Pre train_ds sa používa predbežné miešanie (shuffle(10000)), dávkovanie (batch(batch_size)), ukladanie do vyrovnávacej pamäte (cache()) a prefetch(AUTOTUNE)).

Pre val_ds a test_ds sa používajú podobné kroky bez miešania: dávkovanie, caching a prefetching.

```python
def build_model(vocab_size, embedding_dim, max_len, embedding_matrix):
    model = Sequential([
        Embedding(
            input_dim=vocab_size,
            output_dim=embedding_dim,
            weights=[embedding_matrix],
            input_length=max_len,
            trainable= True
        ),
        Bidirectional(LSTM(
            units=32,
            dropout=0.2,
            recurrent_dropout=0.2
        )),
        Dropout(0.3),
        Dense(
            units=32,
            activation='relu',
            kernel_regularizer=l2(1e-4)
        ),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(
        loss='binary_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
    )
    return model

model = build_model(
    vocab_size=vocab_size,
    embedding_dim=embedding_dim,
    max_len=max_len,
    embedding_matrix=embedding_matrix
)
model.summary()
```

Obr. 19 Tvorba modelu LSTM na slovenskom súbore údajov

Funkcia build_model vytvorí Keras-Sekvenčný model začínajúci vrstvou Embedding (veľkosť vstupu - vocab_size, výstup - embedding_dim, inicializovaný predtrénovanou embedding_matrix, natrénovaný). Nasleduje obojsmerný LSTM (32 jednotiek, dropout=0,2, recurrent_dropout=0,2), potom Dropout(0,3), hustá vrstva Dense(32, aktivácia='relu', kernel_regularizer=l2(1e-4)), opäť Dropout(0,3) a konečná Dense(1, aktivácia='sigmoid').

Model sa skompiluje so stratou='binary_crossentropy', adamovým optimalizátorom a metrikou presnosti, po čom sa vráti sieť pripravená na trénovanie a prostredníctvom funkcie model.summary() sa vypíše zhrnutie štruktúry.

```
# 4. Коллбэки для контроля переобучения
early = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=1,
    min_lr=1e-6
)
```

Obr. 20 Kolbeky LSTM na slovenskom súbore údajov

Model nastaví funkciu EarlyStopping, ktorá sleduje metriku val_loss, pozastaví učenie po 3 po sebe nasledujúcich neúspešných epochách a vráti váhy najlepšej iterácie.

Model pridá ReduceLROnPlateau, ktorý zníži rýchlosť učenia na polovicu, ak sa val_loss nezlepší do 1 epochy, pričom dolná hranica je 1e-6.

Tieto kolbaky slúžia na kontrolu nadmerného učenia a automatické prispôsobenie miery učenia.

```
# 5. Обучение
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=20,
    callbacks=[early, reduce_lr]
)
```

Obr. 21 Trénovanie modelu LSTM na slovenskom súbore údajov

Model sa trénuje pomocou funkcie model.fit, pričom sa mu poskytne súbor trénovacích údajov train_ds a súbor validačných údajov val_ds.

Trénovanie je navrhnuté na maximálne 20 epoch s použitím collabs early (zastavenie, keď nedôjde k zlepšeniu) a reduce_lr (zníženie LR pri plateau).

5. LSTM na anglickom súbore údajov

5.1. Príprava a spustenie projektu

Vo fáze prípravy a spustenia projektu je všetko rovnaké ako v prípade SlovakDistilBERT

5.1.1. Hlavné komponenty a funkcie

```
import tensorflow as tf
from tensorflow.keras import mixed_precision


mixed_precision.set_global_policy('mixed_float16')


path = 'jigsaw-toxic-comment-classification-challenge/train.csv'
import pandas as pd

df = pd.read_csv(path)


from tensorflow.keras.layers import TextVectorization

X = df['comment_text'].astype(str)
y = df[df.columns[2:]].values


MAX_FEATURES = 5000  # (before 10000)
SEQUENCE_LENGTH = 500  # (before 1000)
vectorizer = TextVectorization(
    max_tokens=MAX_FEATURES,
    output_mode='int',
    output_sequence_length=SEQUENCE_LENGTH
)
vectorizer.adapt(X.values)
```

Obr. 22 Zmiešaná presnosť a jemné doladenie vektorizátora LSTM na anglickom súbore údajov

Model nastavuje globálnu techniku mixed_float16 so zmiešanou presnosťou na urýchlenie výpočtu.

Model načíta súbor CSV Jigsaw Toxic Comment Classification do pandas-DataFrame a alokuje texty (X) a pole štítkov (y).

Vytvorí vrstvu Keras TextVectorisation s obmedzením slovníka na 5000 tokenov a dĺžkou sekvencie 500, potom ju prispôsobí na všetky komentáre pomocou funkcie vectorizer.adapt().

```python
# 3. Создание tf.data.Dataset с оптимизациями и конвейером векторизации
BATCH_SIZE = 32  # увеличение батча ускоряет обучение на GPU

dataset = tf.data.Dataset.from_tensor_slices((X.values, y))
dataset = dataset.shuffle(buffer_size=60000)
dataset = dataset.map(
    lambda text, label: (vectorizer(text), label),
    num_parallel_calls=tf.data.AUTOTUNE
)
dataset = dataset.batch(BATCH_SIZE)
dataset = dataset.prefetch(tf.data.AUTOTUNE)

# Разбиение на train/val/test
data_size = tf.data.experimental.cardinality(dataset).numpy()
train_size = int(0.7 * data_size)
val_size = int(0.2 * data_size)

train = dataset.take(train_size)
val = dataset.skip(train_size).take(val_size)
test = dataset.skip(train_size + val_size)
```

Obr. 23 tf.data.Dataset pipeline a rozdelenie LSTM na anglickom súbore údajov

Model vytvorí tf.data.Dataset z tuplov (X.values, y), potom použije operácie shuffle(buffer_size=60000), map(lambda text, label: (vectorizer(text), label), num_parallel_calls=AUTOTUNE), batch(BATCH_SIZE) a prefetch(AUTOTUNE) na optimalizáciu prenosu dát.

Model získa veľkosť celého súboru údajov prostredníctvom tf.data.experimental.cardinality().numpy(), vypočíta train_size = 0,7 * data_size a val_size = 0,2 * data_size.

Model rozdelí tok: prvé dávky veľkosti train_size sú train, ďalšie dávky veľkosti val_size sú val a zvyšné dávky sú test, pričom použije metódy take a skip.

```python

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

model = Sequential([
    Embedding(input_dim=MAX_FEATURES + 1, output_dim=32, input_length=SEQUENCE_LENGTH),
    # CuDNN-ускоренный LSTM (без recurrent_dropout)
    Bidirectional(LSTM(32)),
    Dropout(0.3),  # слегка снизили Dropout
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(6, activation='sigmoid')  # убрали лишние Dense-слои
])

model.compile(
    loss='binary_crossentropy',
    optimizer=Adam(learning_rate=2e-4),  # чуть более агрессивная скорость обучения
    metrics=['accuracy']
)
model.summary()

# Callbacks для контроля переобучения и адаптивной lr
early_stop = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=1)
checkpoint = ModelCheckpoint('best_toxic_lstm_jigsaw_fast.h5', monitor='val_loss', save_best_only=True)
```

Obr. 24 Definícia a kompilácia modelu LSTM na anglickom súbore údajov

Model vytvorí Keras Sequential s Embedding(input_dim=MAX_FEATURES+1, output_dim=32, input_length=SEQUENCE_LENGTH), po ktorom nasleduje obojsmerný LSTM s 32 jednotkami (CuDNN-accelerated), Dropout(0.3), skrytý Dense(128, aktivácia='relu'), opäť Dropout(0.3) a konečný Dense(6, aktivácia='sigmoid').

Zostavené so stratovou funkciou binary_crossentropy, optimalizátorom Adam(learning_rate=2e-4) a metrikou presnosti.

Kolbeky: EarlyStopping by val_loss (trpezlivosť=2, obnoviť najlepšie váhy), ReduceLROnPlateau (znížiť lr na polovicu, ak nedôjde k zlepšeniu za 1 epochu) a ModelCheckpoint na zachovanie najlepšej verzie.

```python
# 5. Обучение модели
EPOCHS = 10  # уменьшили число эпох
history = model.fit(
    train,
    epochs=EPOCHS,
    validation_data=val,
    callbacks=[early_stop, reduce_lr, checkpoint]
)

```

Obr. 25 Trénovanie modelu LSTM na anglickom súbore údajov

Model sa trénuje prostredníctvom volania model.fit, pričom súbory údajov train a val sa vkladajú počas 10 epoch.

Na kontrolu pretrénovania sa používajú tieto spätné volania: early_stop (zastavenie pri val_loss), reduce_lr (zníženie LR pri plateau) a checkpoint (zachovanie najlepších váh).