COMP0173-CW2-NOTEBOOK-P6

December 11, 2025

1 COMP0173: Coursework 2

The paper HEARTS: A Holistic Framework for Explainable, Sustainable, and Robust Text Stereotype Detection by Theo King, Zekun Wu et al. (2024) presents a comprehensive approach to analysing and detecting stereotypes in text [1]. The authors introduce the HEARTS framework, which integrates model explainability, carbon-efficient training, and accurate evaluation across multiple bias-sensitive datasets. By using transformer-based models such as ALBERT-V2, BERT, and DistilBERT, this research project demonstrates that stereotype detection performance varies significantly across dataset sources, underlining the need for diverse evaluation benchmarks. The paper provides publicly available datasets and code [2], allowing full reproducibility and offering a standardised methodology for future research on bias and stereotype detection in Natural Language Processing (NLP).

While the HEARTS framework evaluates stereotype detection in English, this project adapts the methodology to the Russian context. Russian stereotypes often rely on grammatical gender, morphology, and culture specific tropes. Although Russian is not classified as a low-resource language and many high-performing NLP models are available, there is currently no publicly accessible model specifically designed to detect stereotypes in Russian language. Existing models detecting toxicity or sentiment identify stereotypical and biased sentences only when they include specific patterns, such as insults, slurs, or identity-specific hate speech [8].

To address this gap, I introduce two fine-tuned classifiers, `AI-Forever-RuBert` [10] and `XML-RoBERTa` [11] trained on datasets `RBSA`, and `RBS`, respectively. Understanding these patterns is essential for applications such as content moderation, ensuring the safety of Russian-language LLMs, and monitoring harmful narratives across demographic groups and underrepresented societies. Adapting the HEARTS framework to this new sociolinguistic context illustrates its transferability beyond the English-speaking context and enables a more culturally grounded approach to bias detection, thereby promoting SDG 5: Gender Equality, SDG 10: Reduced Inequalities, and SDG 16: Peace, Justice, and Strong Institutions [5].

2 Instructions

All figures produced during this notebook are stored in the project's `COMP0173_Figures` directory. The corresponding LaTeX-formatted performance comparison tables, jupyter notebooks are stored in `/COMP0173_PDF`. The compiled document are available as `COMP0173-CW2-TABLES.pdf` and `COMP0173_PDF/COMP0173-CW2-NOTEBOOK-XX.pdf`. All prompts used for data augmentation are stored in `COMP0173_Prompts` and the manually collected stereotypes (with English translations) are provided in `COMP0173_Stereotypes`. The datasets used for model training and evaluation are stored in `COMP0173_Data` which contains:

- rubias.tsv — RuBias dataset [6, 7]
- ruster.csv — RuSter dataset (see Part 2 of the notebook for source websites)
- rubist.csv — RBS dataset: RuBias + RuSter augmented with LLM-generated samples (Claude Sonnet), using a zero-shot prompt with examples
- rubist_second.csv — RBSA dataset: RuBias + RuSter augmented with LLM-generated samples using a second prompt version without examples

The notebooks `COMP0173_PDF/COMP0173-CW2-NOTEBOOK-P3.pdf` and `COMP0173_PDF/COMP0173-CW2-NOTEBOOK-P5.pdf` are replications of `COMP0173_PDF/COMP0173-CW2-NOTEBOOK-P2.pdf` and `COMP0173_PDF/COMP0173-CW2-NOTEBOOK-P4.pdf`, where P2 provides the new `RBSA` with second prompt (without examples) and P5 demonstrates the model running ON GPU (the results saved are from GPU fine-tuning).

3 Technical Implementation (70%)

[1]:
```
# %%capture
# pip install -r requirements.txt
# pip install transformers
# pip install --upgrade transformers
# pip install --upgrade tokenizers
# pip install -U sentence-transformers
# pip install natasha
# pip install datasets
# pip install --user -U nltk
# conda install -c anaconda nltk
# pip install --upgrade openai pandas tqdm
# pip install dotenv
# python -m spacy download ru_core_news_lg
```

[2]:
```
# pip install -U pip setuptools wheel
# pip install -U spacy
# python -m spacy download en_core_web_trf
# python -m spacy download en_core_web_sm
# python -m spacy download ru_core_news_lg

# # GPU
# pip install -U 'spacy[cuda12x]'
# # GPU - Train Models
# pip install -U 'spacy[cuda12x,transformers,lookups]'
```

[3]:
```
# Import the libraries
import random, numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
plt.style.use('seaborn-v0_8')
```

```
# To ignore warnings
import warnings
warnings.filterwarnings('ignore')
np.random.seed(23)

warnings.filterwarnings(
    "ignore",
    message="pkg_resources is deprecated as an API"
)
```

[59]:
```
# Import libraries
import pandas as pd
import os
import sys
import importlib.util, pathlib
from pathlib import Path
import warnings
from importlib import reload
from importlib.machinery import SourceFileLoader
from IPython.display import display
import pandas as pd
from pathlib import Path
import re
import difflib
import string
from collections import defaultdict
import json
import gc
import pandas as pd
import pandas as pd
from deep_translator import GoogleTranslator
from sklearn.metrics import accuracy_score, f1_score, classification_report

import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

from sklearn.metrics import f1_score
```

[34]:
```
import torch
import transformers
from transformers import AutoModelForMaskedLM, XLMWithLMHeadModel
from transformers import AutoConfig, AutoModelForSequenceClassification,␣
 ↪AutoTokenizer
from transformers import AutoTokenizer, AutoConfig
from transformers import TrainingArguments, Trainer
from sentence_transformers import SentenceTransformer, util
```

```
import platform
from datasets import Dataset
# import spacy
import requests
from tqdm import tqdm
import yaml
import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"
```

[6]:
```
sys.path.append("Exploratory Data Analysis")
sys.path.append("Model Training and Evaluation")
sys.path.append("Model Explainability")

from Initial_EDA import (
    prepare_target_variable_distribution,
    prepare_group_distribution,
    prepare_text_length_analysis,
    create_word_cloud
)

from Sentiment_Toxicity_Analysis_Ru import analyse_sentiment_and_regard
```

[7]:
```
# # Check the GPU host (UCL access)
# print("CUDA available:", torch.cuda.is_available())
# print("Device:", torch.cuda.get_device_name(0))

# # Path
# import os
# os.chdir("/tmp/HEARTS-Text-Stereotype-Detection")
# os.getcwd()
```

3.1 Part 5: Evaluate the adapted model, comparing performance metrics with the original study

Across all adapted models, `XLM-RoBERTa` achieved the highest overall performance, reaching a Macro F1 of 97.9% on the RBS dataset. However, this result is not fully reliable, as the model showed clear signs of overfitting to LLM-generated augmentation data, limiting its practical use.

In contrast, `AI-Forever-RuBERT` provided more stable and realistic performance across datasets, achieving around 74.3% Macro F1, making it the most trustworthy binary classifier for stereotype detection in Russian.

Logistic Regression baselines performed poorly particularly TF-IDF on RBSA (51.8%) highlighting the need for deep contextual modelling rather than surface-level lexical cues.

When broken down by stereotype type, RuBERT performed best on gender 78%, LGBTQ+ 77%, and nationality 76%, while profession stereotypes 66% were the hardest to classify, reflecting their broader semantic variability (i.e profession and socio-economic class). Multilingual tests further showed that `AI-Forever-RuBERT` generalises better than `XLM-RoBERTa` when translating English

sentences into Russian, though human supervision and carefully designed few-shot prompting remain necessary during data augmentation.

Table 4: Comparison of Adapted Model Macro F1 scores (%)

Model Type	Emissions	Training Data	Test Set Macro F1 Score%
LR-TFIDF	0.01g	RBS	93.1%
LR-TFIDF	0.01g	RBSA	51.8%
LR-Embeddings	0.21g	RBS	92.4%
LR-Embeddings	0.15g	RBSA	65.5%
DeepPavlov-RuBert	1.28g	RBS	97.1%
DeepPavlov-RuBert	1.09g	RBSA	72.4%
AI-Forever-RuBert	1.36g	RBS	97.2%
AI-Forever-RuBert	1.08g	RBSA	**74.3%**
XLM-RoBERTa	Unknown	RBS	**97.9%**
XLM-RoBERTa	Unknown	RBSA	73.8%

3.1.1 Helper Functions

[8]:
```python
def extract_macro_f1(path):
    df = pd.read_csv(path)
    row = df[df.iloc[:, 0].str.contains("macro", case=False, na=False)]
    return float(row["f1-score"].values[0])
```

[9]:
```python
def build_table(dataset_name):
    rows = []
    for model_name, folder in MODEL_FOLDERS.items():
        report_path = os.path.join(folder, dataset_name, "classification_report.
 ↪csv")
        if os.path.exists(report_path):
            macro = extract_macro_f1(report_path)
            rows.append([model_name, macro])

    df = pd.DataFrame(rows, columns=["Model", "Macro F1"])
    df = df.sort_values("Macro F1", ascending=False).reset_index(drop=True)

    return df
```

[ ]:
```python
def plot_table(df, title):

    """
    Single-series barplot:
    - seaborn whitegrid
    - soft y-grid only
    - no black bar outlines
    - clean title & ticks
    """
```

```
sns.set_style("whitegrid")

models = df["Model"].tolist()
scores = df["Macro F1"].values

x = np.arange(len(models))

fig, ax = plt.subplots(figsize=(6, 3), dpi=200)

# Bars (no outlines, seaborn-like colour)
palette = sns.color_palette("Purples_d", len(models))
ax.bar(
    x,
    scores,
    color=palette,
    edgecolor=None,
)

# X ticks
ax.set_xticks(x)
ax.set_xticklabels(models, rotation=0, fontsize=7)

# Y axis from 0 to 1 (Macro F1)
ax.set_ylim(0, 1.0)
ax.set_yticks(np.linspace(0, 1.0, 6))
ax.tick_params(axis="y", labelsize=9)
ax.set_ylabel("")

# Title in same style
ax.set_title(title, fontsize=14, fontweight="bold", pad=12, y = 1.1)

# Grid only on y
ax.grid(axis="y", color="0.75", linewidth=0.5)
ax.grid(axis="x", visible=False)

sns.despine(ax=ax)

for xi, val in zip(x, scores):
    ax.text(
        xi,
        val + 0.015,
        f"{val:.3f}",
        ha="center",
        va="bottom",
        fontsize=8,
    )
```

```
    fig.tight_layout()
    plt.show()
```

[ ]:
```
def f1_barplot(percent_table, title):

    sns.set_style("whitegrid")

    categories = percent_table.index.tolist()
    values = percent_table["Macro F1"].values

    # colour palette per group
    color_map = {
        "gender":      "#c0392b",
        "profession":  "#f1c40f",
        "nationality": "#27ae60",
        "race":        "#3498db",
        "religion":    "#2c3e50",
        "lgbtq":       "#8e44ad",
        "other":       "#7f8c8d",
    }
    colors = [color_map.get(cat, "grey") for cat in categories]

    x = np.arange(len(categories))

    fig, ax = plt.subplots(figsize=(6, 4), dpi=200)

    ax.bar(
        x,
        values,
        width=0.55,
        color=colors,
        edgecolor=None,
    )

    # x-axis
    ax.set_xticks(x)
    ax.set_xticklabels(categories, fontsize=11)

    # y-axis
    ax.set_ylim(0, 100)
    ax.set_yticks(np.arange(0, 101, 10))
    ax.tick_params(axis='y', labelsize=9)
    ax.set_ylabel("")

    ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda v, pos: f"{int(v)}%"))
```

```
    # title
    ax.set_title(title, fontsize=14, fontweight="bold", pad=12)

    # seaborn-style grid
    ax.grid(axis="y", color="0.75", linewidth=0.5)
    ax.grid(axis="x", visible=False)

    sns.despine(ax=ax)

    fig.tight_layout()
    plt.show()
```

[ ]:
```
def prepare_results(file_path, save_path):

    """
    Load full_results.csv, create 'categorisation',
    and save the updated CSV. Does NOT return a DataFrame.

    Rules:
    - If actual_label == 1 → 'stereotype_<stereotype_type>'
    - If actual_label == 0 → 'neutral'
    """

    df = pd.read_csv(file_path)

    def build_category(row):
        if row["actual_label"] == 1:
            return f"stereotype_{row['stereotype_type']}"
        else:
            return "neutral"

    df["categorisation"] = df.apply(build_category, axis=1)

    df.to_csv(save_path, index=False)
    print(f"Saved updated file to: {save_path}")
```

[ ]:
```
def sample_observations_balanced(file_path, n_per_status=5, seed=42):

    """
    For each categorisation:
      - sample up to n_per_status correctly classified examples
      - sample up to n_per_status misclassified examples

    Returns a DataFrame with ~ 2 * n_per_status * num_categories rows
    (less if some groups are too small).
    """
```

```
    data = pd.read_csv(file_path)

    if 'categorisation' not in data.columns:
        raise ValueError("The CSV must contain a 'categorisation' column.")

    if not {'predicted_label', 'actual_label'}.issubset(data.columns):
        raise ValueError("The CSV must contain 'predicted_label' and␣
↪'actual_label' columns.")

    sampled_parts = []

    # Group only by categorisation (as you requested)
    for cat, group in data.groupby('categorisation'):
        same_label = group[group['predicted_label'] == group['actual_label']]
        diff_label = group[group['predicted_label'] != group['actual_label']]

        # Sample up to n_per_status from each side
        n_same = min(n_per_status, len(same_label))
        n_diff = min(n_per_status, len(diff_label))

        if n_same > 0:
            same_sample = same_label.sample(n=n_same, random_state=seed)
            sampled_parts.append(same_sample)

        if n_diff > 0:
            diff_sample = diff_label.sample(n=n_diff, random_state=seed)
            sampled_parts.append(diff_sample)

    sampled_data = pd.concat(sampled_parts, axis=0).reset_index(drop=True)

    print("Rows per categorisation in the sample:")
    print(sampled_data['categorisation'].value_counts())
    print(f"\nTotal sampled rows: {len(sampled_data)}")

    return sampled_data
```

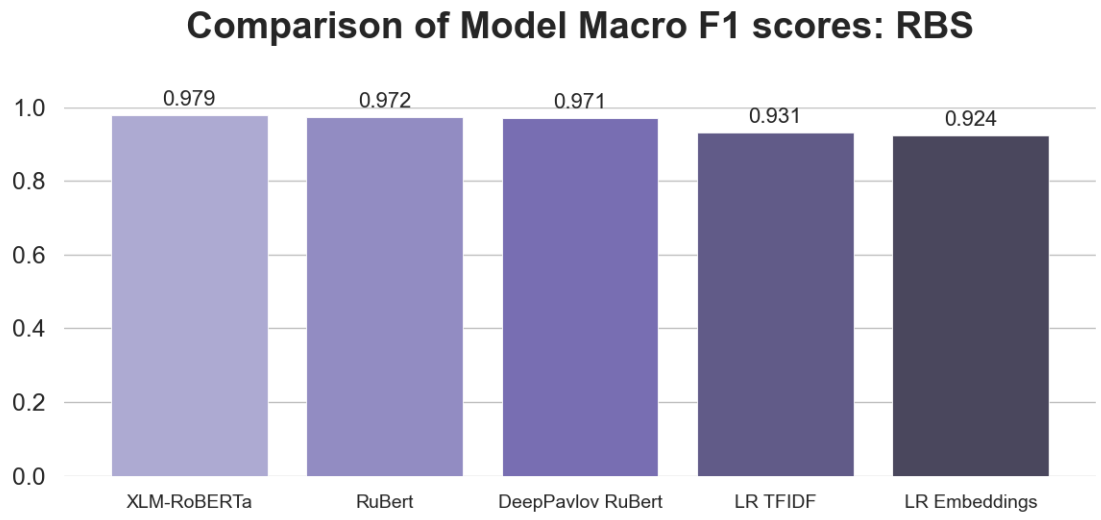
[ ]:
```
def build_token_ranking_string(shap_df, sentence_id, top_n=None):

    """
    Build a string like:
    '"housewife": 0.446, "woman": 0.159, ...'
    for one sentence_id, sorted by |SHAP| descending.
    """

    subset = shap_df[shap_df['sentence_id'] == sentence_id][['token',␣
↪'value_shap']]
```

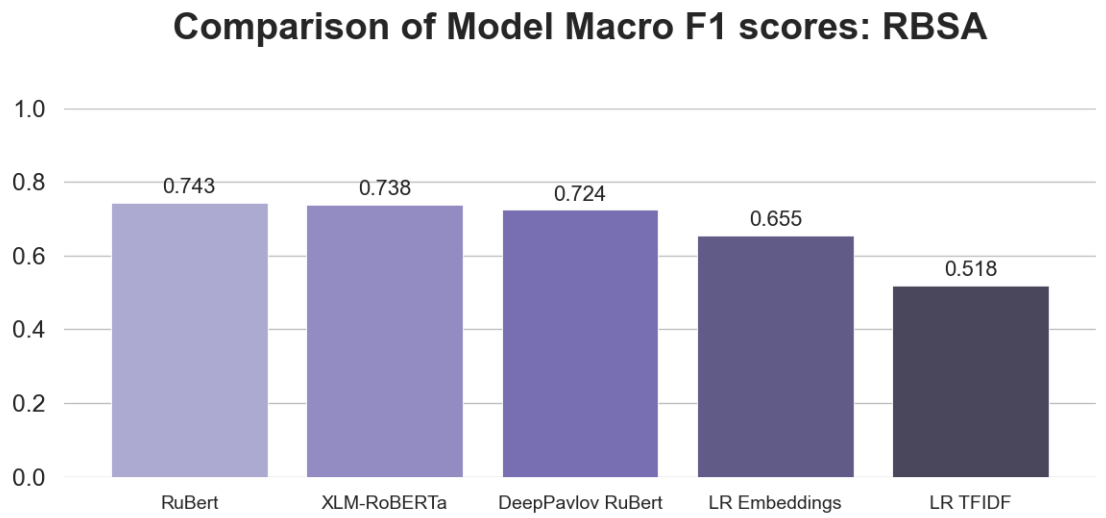
```
    # Sort by absolute SHAP
    subset = subset.sort_values('value_shap', key=lambda s: s.abs(),␣
↪ascending=False)

    if top_n is not None:
        subset = subset.head(top_n)

    parts = [
        f'"{tok}": {val:.3f}'
        for tok, val in zip(subset['token'], subset['value_shap'])
    ]
    return ", ".join(parts)
```

[ ]:
```
def build_explanation_table(shap_df, sentence_df, top_n_tokens=None):

    """
    shap_df: token-level SHAP results
    sentence_df: sentence-level similarity results (grouped)
                 must contain: sentence_id, cosine_similarity,
                                pearson_correlation, js_divergence
    top_n_tokens: how many tokens to show in 'Token Rankings' (None = all)

    Returns: pandas DataFrame with columns similar to HEARTS Table 2.
    """

    rows = []

    for _, srow in sentence_df.iterrows():
        sid = srow['sentence_id']

        # Grab metadata from any row with this sentence_id
        meta = shap_df[shap_df['sentence_id'] == sid].iloc[0]

        rows.append({
            "Text Instance": meta['sentence'],
            "Predicted Label": meta['predicted_label'],
            "Actual Label": meta['actual_label'],
            "Token Rankings": build_token_ranking_string(
                shap_df, sid, top_n=top_n_tokens
            ),
            "Cosine Similarity": srow['cosine_similarity'],
            "Pearson R": srow['pearson_correlation'],
            "JS Divergence": srow['js_divergence'],
        })

    return pd.DataFrame(rows)
```

3.1.2 Macro F1 Scores

[12]:
```
MODEL_FOLDERS = {
    "LR TFIDF": "result_output_LR_tfidf",
    "LR Embeddings": "result_output_LR_embedding",
    "DeepPavlov RuBert": "result_output_deeppavlov_rubert",
    "RuBert": "result_output_ruberta_base",
    "XLM-RoBERTa": "result_output_xlm_roberta_base",
}

DATASETS = ["rubist_trained", "rubist_second_trained"]

table_A = build_table("rubist_trained")
table_B = build_table("rubist_second_trained")

plot_table(table_A, "Comparison of Model Macro F1 scores: RBS")
plot_table(table_B, "Comparison of Model Macro F1 scores: RBSA")
```



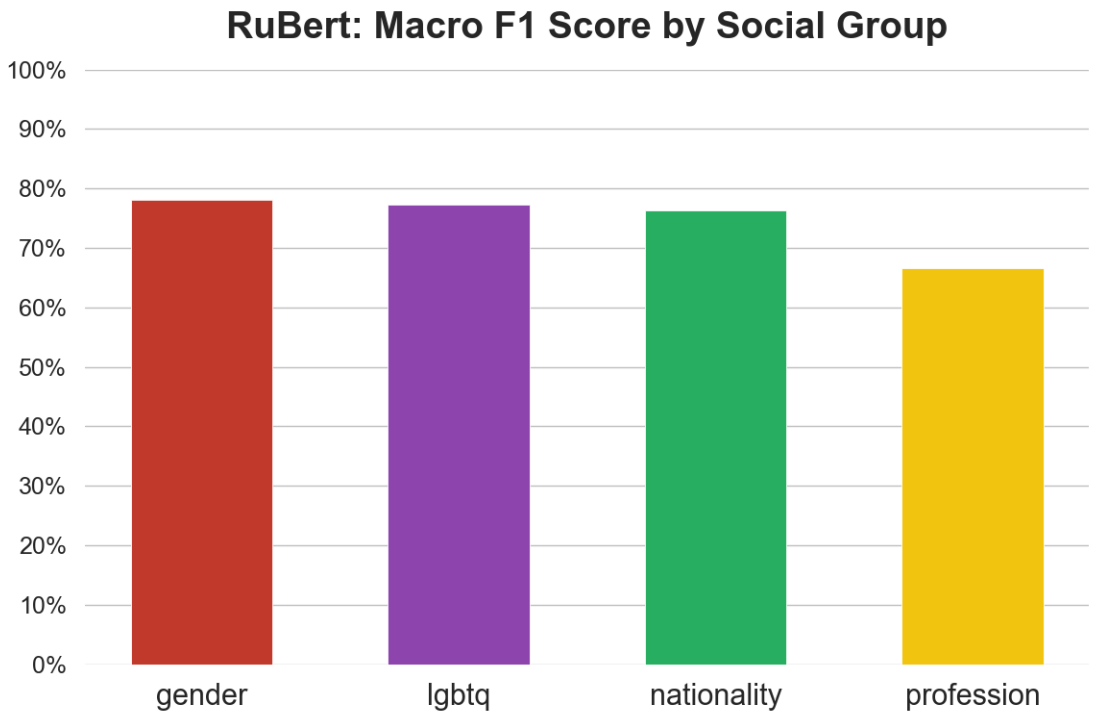


[50]:
```
df = pd.read_csv("result_output_ruberta_base/rubist_second_trained/full_results.
↪csv")

group_f1 = (
    df
    .groupby("stereotype_type")
    .apply(lambda g: f1_score(g["actual_label"], g["predicted_label"],␣
↪average="macro"))
    .sort_index()
)

# convert to % table compatible with plotting helper
percent_table = pd.DataFrame(
    {"Macro F1": group_f1.values * 100},
    index=group_f1.index
)
percent_table
```

[50]:

stereotype_type	Macro F1
gender	78.151261
lgbtq	77.228652
nationality	76.304606
profession	66.545455

[52]:
```
f1_barplot(percent_table, "RuBert: Macro F1 Score by Social Group")
```



3.1.3 SHAP & LIME

[29]:
```
from SHAP_LIME_Analysis_Ru import (sample_observations, shap_analysis,␣
 ↪lime_analysis, compute_cosine_similarity, compute_pearson_correlation,␣
 ↪compute_js_divergence)

file_path = 'result_output_ruberta_base/rubist_second_trained/full_results.csv'
model_path = "model_output_ruberta_base/rubist_second_trained"

prepare_results(
    "result_output_ruberta_base/rubist_second_trained/full_results.csv",
    "result_output_ruberta_base/rubist_second_trained/full_results.csv"
)
```

```
Saved updated file to:
result_output_ruberta_base/rubist_second_trained/full_results.csv
```

[30]:
```
# This gives you  50 rows (if each category has  5 correct and  5 incorrect)
sampled_data = sample_observations_balanced(
    file_path,
    n_per_status=5,
    seed=42
)
```

```
sampled_data.to_csv("COMP0173_Temp_Data/sampled_50_examples.csv", index=False)
sampled_data.head()
```

```
Rows per categorisation in the sample:
categorisation
neutral                   10
stereotype_gender         10
stereotype_lgbtq          10
stereotype_nationality    10
stereotype_profession     10
Name: count, dtype: int64

Total sampled rows: 50
```

[30]:

```
                                                 text  predicted_label  \
0                                                     0
1                                                         0
2                                …                    0
3                                                    0
4                                                      0

   predicted_probability  actual_label stereotype_type   dataset_name  \
0               0.760907             0          gender  rubist_second
1               0.959988             0     nationality  rubist_second
2               0.850051             0      profession  rubist_second
3               0.946351             0      profession  rubist_second
4               0.961002             0      profession  rubist_second

  categorisation
0        neutral
1        neutral
2        neutral
3        neutral
4        neutral
```

[31]:

```
# Shap
shap_results = shap_analysis(sampled_data, model_path)
print(shap_results)

lime_results = lime_analysis(sampled_data, model_path)
print(lime_results)

lime_results.to_csv('COMP0173_Results/lime_results.csv')
shap_results.to_csv('COMP0173_Results/shap_results.csv')

shap_df = pd.read_csv('COMP0173_Results/shap_results.csv')
lime_df = pd.read_csv('COMP0173_Results/lime_results.csv')
```

```
Hardware accelerator e.g. GPU is available in the environment, but no `device`
argument is passed to the `Pipeline` object. Model will be on CPU.
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
```

Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted

```
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
```

```
Hardware accelerator e.g. GPU is available in the environment, but no `device`
argument is passed to the `Pipeline` object. Model will be on CPU.
```

```
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
     sentence_id        token  value_shap  \
0              0               -0.009527
1              0              0.167718
2              0           -0.377149
3              0              -0.055029
4              0              0.000810
..           …           …          …
296           49              -0.046913
297           49              -0.020005
298           49             -0.127602
299           49                -0.035212
300           49           -0.027750

                                 sentence        dataset  \
0                           rubist_second
1                           rubist_second
2                           rubist_second
3                           rubist_second
4                           rubist_second
..                                     …            …
296                         rubist_second
297                         rubist_second
298                         rubist_second
299                         rubist_second
300                         rubist_second

            categorisation  predicted_label  actual_label
0                  neutral                0             0
1                  neutral                0             0
2                  neutral                0             0
3                  neutral                0             0
4                  neutral                0             0
..                     …              …           …
296  stereotype_profession                0             1
297  stereotype_profession                0             1
298  stereotype_profession                0             1
299  stereotype_profession                0             1
300  stereotype_profession                0             1
```

```
[301 rows x 8 columns]
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 0 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: neutral - Predicted Label: 1 - Actual
Label: 0
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 1
- Actual Label: 1
Probabilities shape: (100, 2)
```

```
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_gender - Predicted Label: 0
- Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 1 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_lgbtq - Predicted Label: 0 -
Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
```

```
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_nationality - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 1 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
```

```
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
Probabilities shape: (100, 2)
Dataset: rubist_second - Categorisation: stereotype_profession - Predicted
Label: 0 - Actual Label: 1
     sentence_id        token  value_lime  \
0              0                 0.027508
1              0               0.276595
2              0            -0.484843
3              0                0.055742
4              0               0.033380
..           …           …         …
296           49                0.015434
297           49                0.134527
298           49             -0.069445
299           49                -0.104911
300           49             0.064724

                                sentence        dataset  \
0                            rubist_second
1                            rubist_second
2                            rubist_second
3                            rubist_second
4                            rubist_second
..                                     …            …
296                          rubist_second
297                          rubist_second
298                          rubist_second
299                          rubist_second
300                          rubist_second

            categorisation  predicted_label  actual_label
0                  neutral                0             0
1                  neutral                0             0
2                  neutral                0             0
3                  neutral                0             0
4                  neutral                0             0
..                     …              …           …
296  stereotype_profession                0             1
297  stereotype_profession                0             1
298  stereotype_profession                0             1
299  stereotype_profession                0             1
300  stereotype_profession                0             1

[301 rows x 8 columns]
```

[ ]:
```
# Compute similarity scores by sentence
# Columns that are common to both shap_df and lime_df, except the value columns
common_columns = [col for col in shap_df.columns if col != 'value_shap']

# Merge on metadata (sentence_id, token, etc.)
merged_df = pd.merge(shap_df, lime_df, on=common_columns, suffixes=('_shap',␣
 ↪'_lime'))

# Group by sentence_id collect lists of SHAP and LIME values per sentence
grouped = merged_df.groupby('sentence_id').agg({
    'value_shap': list,
    'value_lime': list
}).reset_index()

# Compute similarities PER ROW (i.e. per sentence)
grouped['cosine_similarity'] = grouped.apply(
    lambda row: compute_cosine_similarity(row['value_shap'], row['value_lime']),
    axis=1
)

grouped['pearson_correlation'] = grouped.apply(
    lambda row: compute_pearson_correlation(row['value_shap'],␣
 ↪row['value_lime']),
    axis=1
)

grouped['js_divergence'] = grouped.apply(
    lambda row: compute_js_divergence(row['value_shap'], row['value_lime']),
    axis=1
)

grouped.to_csv('COMP0173_Results/sentence_level_similarity_results.csv',␣
 ↪index=False)
```

[33]:
```
rbsa_shap_lime = build_explanation_table(shap_df, grouped, top_n_tokens=10)
print(rbsa_shap_lime)
```

```
                                  Text Instance  Predicted Label  \
0                                                             0
1                                                                 0
2                                 …                 0
3                                                     0
4                                                       0
5                                                                 1
6                                 …                 1
7                                                         1
8                                                                   1
9                                       -                         1
```

```
10                                                    1
11                                                   1
12                                                    1
13                                                   1
14                                                    1
15                                                  0
16                                                   0
17                                                0
18                                                 0
19                                                 0
20                                                  1
21                                                   1
22                                                  1
23                                                 1
24                                                 1
25                                                0
26                              …                 0
27                              …                 0
28                                                   0
29                                                   0
30                                                           1
31                             …                 1
32                                                      1
33                                                    1
34                              …                 1
35                                                     0
36                                                  0
37       -                      …                 0
38                                                   0
39                                                    0
40                                                           1
41                                                      1
42                                                   1
43                                                   1
44                                                    1
45                              …                 0
46                              …                 0
47                                                     0
48                                                  0
49                                                    0

    Actual Label                                      Token Rankings  \
0              0  "      ": -0.377, "    ": 0.168, "  ":…
1              0  "    ": 0.150, "      ": 0.098, "  ":…
2              0  "   ": -0.148, "      ": -0.073, "  …
3              0  "    ": -0.218, "        ": -0.090, "…
4              0  "     ": -0.151, "   ": -0.112, "    …
5              0  "     ": 0.326, "   ": 0.039, "    "…
```

```
6              0  "    ": 0.209, "    ": 0.137, "    ": 0…
7              0  "    ": -0.067, "   ": -0.062, " ": 0.045…
8              0  " ": 0.194, "   ": -0.071, " ": 0.057, " …
9              0  " ": 0.135, "  ": 0.073, "    ": 0.064, "…
10             1  "   ": 0.163, "    ": 0.106, "   ": -0…
11             1  " ": -0.129, "   ": -0.052, "   ": -0.009…
12             1            "     ": -0.066, "    ": -0.008
13             1  "  ": -0.285, "    ": 0.162, "    ": 0…
14             1  "    ": 0.163, "    ": 0.142, "    ": …
15             1  "  ": -0.108, "    ": -0.089, "    ": …
16             1  "    ": -0.314, "    ": -0.029, "   ":…
17             1  "     ": 0.237, "    ": 0.229, "    …
18             1  "     ": -0.160, "   ": -0.069, "  ": …
19             1  "   ": 0.201, "     ": 0.121, "     …
20             1  "    ": -0.282, "   ": 0.266, "      …
21             1  "         ": 0.468, "  ": -0.111, " …
22             1  "   ": 0.284, "    ": 0.159, "  ": -0.1…
23             1  "  ": 0.434, "    ": -0.280, "   ": 0.0…
24             1  "     ": -0.325, "       ": 0.168, "…
25             1  "   ": 0.314, "   ": -0.153, "  ": -0.07…
26             1  "  ": 0.269, "    ": -0.101, "    ": -0…
27             1  "    ": -0.135, "    ": -0.112, "   …
28             1  "  ": -0.404, "     ": -0.080, "    …
29             1  "     ": 0.355, " ": 0.107, "   ": -0…
30             1  " ": 0.259, "   ": 0.240, "  ": -0.105, "…
31             1  "     ": 0.292, "      ": -0.166, " …
32             1  "    ": 0.124, "  ": 0.054, "      ":…
33             1  "    ": -0.238, "    ": -0.038, "  ": 0…
34             1  "   ": 0.198, "  ": 0.152, " ": 0.148, " …
35             1  "    ": -0.140, "   ": -0.093, " ": 0.0…
36             1  " ": 0.182, "  ": -0.106, "       ": 0…
37             1  "    ": -0.221, "    ": 0.084, "   …
38             1  "    ": -0.207, "   ": 0.124, "   ": -…
39             1  "   ": 0.244, "    ": -0.238, "    ":…
40             1  "   ": 0.276, " ": 0.173, "   ": -0.043,…
41             1  "   ": 0.353, "   ": -0.092, "   ": -0…
42             1  "  ": 0.448, " ": -0.150, "   ": 0.064, …
43             1  "    ": -0.139, "   ": -0.046, "    …
44             1  "    ": 0.477, "   ": -0.125, "   ": …
45             1  "     ": -0.071, "   ": -0.038, "   "…
46             1  "     ": -0.096, "    ": -0.082, "  …
47             1  "    ": -0.286, "   ": 0.154, "  ": 0…
48             1  " ": 0.155, "   ": -0.081, "  ": -0.075, …
49             1  "    ": -0.128, "   ": -0.047, " ": -0…

    Cosine Similarity  Pearson R  JS Divergence
0            0.964143   0.988781       0.027632
1           -0.348640  -0.002832       0.471263
```

2	0.676770	0.763590	0.099222
3	0.828183	0.613583	0.235564
4	0.855120	0.770816	0.125018
5	0.264992	0.610785	0.321379
6	0.667786	0.555056	0.256389
7	-0.862509	-0.871718	0.549105
8	0.064550	-0.231391	0.513021
9	0.794340	0.713386	0.356673
10	0.451358	0.357677	0.442147
11	-0.056242	0.368540	0.412894
12	-0.875928	-1.000000	0.832555
13	0.585602	0.750209	0.165684
14	0.787753	0.685872	0.362956
15	0.444283	0.150051	0.294675
16	0.809166	0.859023	0.092788
17	0.774558	0.817931	0.221960
18	0.182265	0.762314	0.283366
19	0.685520	0.815871	0.333061
20	0.811151	0.807285	0.373401
21	0.548231	0.367339	0.334435
22	0.886037	0.859557	0.288865
23	0.383890	0.481535	0.227895
24	0.159641	0.352168	0.279672
25	0.953830	0.975215	0.143841
26	0.713279	0.717822	0.210607
27	0.729420	0.727859	0.181422
28	0.639157	0.687330	0.310731
29	0.550193	0.587217	0.372541
30	0.491337	0.314169	0.444295
31	0.209800	0.149245	0.308859
32	0.516545	-0.269198	0.443304
33	-0.181583	0.093151	0.494417
34	0.577843	0.341301	0.300375
35	0.926056	0.973326	0.056813
36	0.571328	0.617878	0.311407
37	0.880381	0.939832	0.044177
38	0.269908	0.309949	0.326199
39	0.708573	0.763241	0.288905
40	0.667260	0.413746	0.369953
41	0.182220	-0.570094	0.464373
42	0.727205	0.710759	0.331780
43	-0.656870	-0.229484	0.490558
44	0.920951	0.908724	0.321756
45	-0.106654	0.376868	0.360637
46	0.182405	0.361077	0.278741
47	0.897708	0.946705	0.061052
48	0.863200	0.914605	0.239640
49	0.259434	0.552922	0.337102

3.1.4 Model Details and Configuration

[38]:
```
config = AutoConfig.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSequenceClassification.from_pretrained(model_path)

training_args_path = os.path.join(model_path, "training_args.bin")
training_args = torch.load(training_args_path)
training_args

batch_size    = training_args.per_device_train_batch_size
learning_rate = training_args.learning_rate
epochs        = training_args.num_train_epochs
```

[ ]:
```
# Key information
model_name      = "stereotype_bias_classifier_rubert"
base_arch       = type(model).__name__            # e.g.␣
 ↪'XLMRobertaForSequenceClassification'
num_params      = sum(p.numel() for p in model.parameters())
vocab_size      = config.vocab_size
labels          = list(range(config.num_labels))  # typically [0, 1]

# Model capacity / architecture
embedding_dim   = config.hidden_size              # XLM-R uses hidden_size as␣
 ↪embedding size
hidden_size     = config.hidden_size
intermediate    = config.intermediate_size
num_layers      = config.num_hidden_layers
num_heads       = config.num_attention_heads

# Regularisation hyperparameters
hidden_act      = config.hidden_act
hidden_dropout  = getattr(config, "hidden_dropout_prob", None)
attn_dropout    = getattr(config, "attention_probs_dropout_prob", None)
classifier_drop = getattr(config, "classifier_dropout", None)
layer_norm_eps  = config.layer_norm_eps

print("Parameters:", num_params)
print("Vocab size:", vocab_size)
print("Hidden size:", hidden_size)
print("Layers:", num_layers)
print("Heads:", num_heads)
```

[43]:
```
hyperparams = {
    "Batch Size": batch_size,
    "Learning Rate": f"{learning_rate:.1e}",
    "Epochs": epochs,
```

```
    "Training Device": "GPU (NVIDIA Tesla T4)",
    "Approximate Runtime": "< 10 Minutes",
}

table_hyper = pd.DataFrame(
    list(hyperparams.items()),
    columns=["Parameter", "Value"]
)
table_hyper
```

[43]:

	Parameter	Value
0	Batch Size	64
1	Learning Rate	2.0e-05
2	Epochs	6
3	Training Device	GPU (NVIDIA Tesla T4)
4	Approximate Runtime	< 10 Minutes

[46]:
```
details = {
    "Model Name": model_name,
    "Base Architecture": base_arch,
    "Number of Parameters": f"{num_params:,}",
    "Vocabulary Size": vocab_size,
    "Labels": str(labels),

    "Model Configuration and Capacity": "",
    "Embedding Dimensionality": embedding_dim,
    "Intermediate Layer Size": intermediate,
    "Hidden Layer Size": hidden_size,
    "Number of Hidden Layers": num_layers,
    "Number of Attention Heads": num_heads,

    "Regularisation Hyperparameters": "",
    "Hidden Layer Activation": hidden_act,
    "Hidden Layer Dropout Probability": hidden_dropout,
    "Attention Head Dropout Probability": attn_dropout,
    "Classification Layer Dropout Probability": classifier_drop,
    "Layer Normalisation Epsilon": layer_norm_eps,
}

table_details = pd.DataFrame(
    list(details.items()),
    columns=["Category", "Details"]
)
table_details
```

[46]:

	Category	\
0	Model Name	

```
1                          Base Architecture
2                      Number of Parameters
3                           Vocabulary Size
4                                    Labels
5          Model Configuration and Capacity
6                  Embedding Dimensionality
7                   Intermediate Layer Size
8                        Hidden Layer Size
9                   Number of Hidden Layers
10                Number of Attention Heads
11           Regularisation Hyperparameters
12                  Hidden Layer Activation
13         Hidden Layer Dropout Probability
14       Attention Head Dropout Probability
15  Classification Layer Dropout Probability
16              Layer Normalisation Epsilon

                             Details
0   stereotype_bias_classifier_rubert
1       BertForSequenceClassification
2                         178,308,866
3                              120138
4                              [0, 1]
5
6                                 768
7                                3072
8                                 768
9                                  12
10                                 12
11
12                               gelu
13                                0.1
14                                0.1
15                               None
16                                0.0
```

[ ]:
```
print(details)
```

3.1.5 Model vs Adapted Models Perfomance

Because the adapted Russian models and the original English HEARTS models operate in different languages, use different tokenisers and embeddings, and are trained on datasets with fundamentally different distributions, a direct metric-to-metric comparison would be invalid.

To enable a fairer evaluation, I designed a cross-lingual testing setup using translation pipelines. This allowed all models English and Russian to be assessed on versions of the same input, either translated from Russian to English or from English to Russian. This experiment tests two things:

- How much each model relies on language-specific biases and lexical patterns.

- Whether stereotype detection generalises beyond the training language.

Summary of Results (Table 7)

On Russian datasets, English `ALBERT-V2` performs moderately well when fed RU -> EN translations (67–77% Macro F1), showing that stereotype cues survive translation, but its performance remains below that of native Russian `AI-Forever-RuBERT`.

`AI-Forever-RuBERT`, trained natively on Russian, achieves much stronger performance on RBSA and RBS (73% and 72.9% Macro F1), confirming that it best captures Russian cultural and linguistic patterns. `XLM-Roberta` achieves very high scores on RBS (97.5%), but this again reflects earlier concerns about overfitting to augmented data — its performance should be interpreted cautiously.

On English MGSD, `ALBERT-V2` reaches the expected strong results (85% Macro F1).

When MGSD is translated EN -> RU, `AI-Forever-RuBERT` drops to 61.6%, and `XLM-Roberta` drops further to 53.1%.
This shows that Russian models struggle when fed English stereotypes, even after translation, likely because:

- MGSD stereotypes reflect Western cultural contexts absent from Russian data,
- Translation distorts subtle pragmatic patterns,
- `XLM-Roberta` overfitted to patterns caused by data augmentation on LLM,
- The moderate/insufficient dataset size (3,000 - 4,000)

Interpretation

- Stereotype detection is highly language and culture-dependent.
- `AI-Forever-RuBERT` generalises better across languages than `XLM-Roberta`, despite `XLM-Roberta`'s high performance in Russian language, likely due to overfitting.
- Translation alone cannot fully align cultural assumptions, meaning stereotype detection models must be trained on culturally grounded and diverse datasets.

Table 7: Performance of ALBERT, RuBERT and XLM-R across English and Russian stereotype datasets.

Dataset	Model	Input used	Accuracy	Macro F1
RBSA (RU)	ALBERT	RU → EN translation	67.5%	67.0%
RBSA (RU)	RuBERT	RU	73.0%	72.9%
RBS (RU)	ALBERT	RU → EN translation	77.5%	77.0%
RBS (RU)	XLM-R	RU	97.5%	97.5%
MGSD (EN)	ALBERT	EN	85.0%	85.0%
MGSD (EN)	RuBERT	EN → RU translation	64.0%	61.6%
MGSD (EN)	XLM-R	EN → RU translation	*55.0%*	53.1%

[113]:
```
# Download holistic ai model
albert_tokenizer = AutoTokenizer.from_pretrained("holistic-ai/
  ↪bias_classifier_albertv2")
```

```
albert_model = AutoModelForSequenceClassification.from_pretrained("holistic-ai/
 ↪bias_classifier_albertv2")

# Download rubert model on rbsa
rubert_tokenizer = AutoTokenizer.from_pretrained("model_output_ruberta_base/
 ↪rubist_second_trained")
rubert_model = AutoModelForSequenceClassification.
 ↪from_pretrained("model_output_ruberta_base/rubist_second_trained")

# Download xlm roberta model on rbs
xlm_tokenizer = AutoTokenizer.from_pretrained("model_output_xlm_roberta_base/
 ↪rubist_trained")
xlm_model = AutoModelForSequenceClassification.
 ↪from_pretrained("model_output_xlm_roberta_base/rubist_trained")
```

```
loading file spiece.model from cache at
/Users/rinlobachevskii/.cache/huggingface/hub/models--holistic-ai--bias_classifi
er_albertv2/snapshots/c1a5e087e6bfdfb7fc9e7333a8028d8f29ba6201/spiece.model
loading file tokenizer.json from cache at
/Users/rinlobachevskii/.cache/huggingface/hub/models--holistic-ai--bias_classifi
er_albertv2/snapshots/c1a5e087e6bfdfb7fc9e7333a8028d8f29ba6201/tokenizer.json
loading file added_tokens.json from cache at None
loading file special_tokens_map.json from cache at
/Users/rinlobachevskii/.cache/huggingface/hub/models--holistic-ai--bias_classifi
er_albertv2/snapshots/c1a5e087e6bfdfb7fc9e7333a8028d8f29ba6201/special_tokens_ma
p.json
loading file tokenizer_config.json from cache at
/Users/rinlobachevskii/.cache/huggingface/hub/models--holistic-ai--bias_classifi
er_albertv2/snapshots/c1a5e087e6bfdfb7fc9e7333a8028d8f29ba6201/tokenizer_config.
json
loading configuration file config.json from cache at
/Users/rinlobachevskii/.cache/huggingface/hub/models--holistic-ai--bias_classifi
er_albertv2/snapshots/c1a5e087e6bfdfb7fc9e7333a8028d8f29ba6201/config.json
Model config AlbertConfig {
  "_name_or_path": "holistic-ai/bias_classifier_albertv2",
  "architectures": [
    "AlbertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0,
  "bos_token_id": 2,
  "classifier_dropout_prob": 0.1,
  "down_scale_factor": 1,
  "embedding_size": 128,
  "eos_token_id": 3,
  "gap_size": 0,
  "hidden_act": "gelu_new",
  "hidden_dropout_prob": 0,
```

```
  "hidden_size": 768,
  "id2label": {
    "0": "Non-Stereotype",
    "1": "Stereotype"
  },
  "initializer_range": 0.02,
  "inner_group_num": 1,
  "intermediate_size": 3072,
  "label2id": {
    "Non-Stereotype": 0,
    "Stereotype": 1
  },
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "albert",
  "net_structure_type": 0,
  "num_attention_heads": 12,
  "num_hidden_groups": 1,
  "num_hidden_layers": 12,
  "num_memory_blocks": 0,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "problem_type": "single_label_classification",
  "torch_dtype": "float32",
  "transformers_version": "4.46.3",
  "type_vocab_size": 2,
  "vocab_size": 30000
}

loading weights file model.safetensors from cache at
/Users/rinlobachevskii/.cache/huggingface/hub/models--holistic-ai--bias_classifi
er_albertv2/snapshots/c1a5e087e6bfdfb7fc9e7333a8028d8f29ba6201/model.safetensors
All model checkpoint weights were used when initializing
AlbertForSequenceClassification.

All the weights of AlbertForSequenceClassification were initialized from the
model checkpoint at holistic-ai/bias_classifier_albertv2.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use AlbertForSequenceClassification for predictions without
further training.
loading file vocab.txt
loading file tokenizer.json
loading file added_tokens.json
loading file special_tokens_map.json
loading file tokenizer_config.json
loading configuration file
model_output_ruberta_base/rubist_second_trained/config.json
Model config BertConfig {
```

```
  "_name_or_path": "model_output_ruberta_base/rubist_second_trained",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "problem_type": "single_label_classification",
  "torch_dtype": "float32",
  "transformers_version": "4.46.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 120138
}

loading weights file
model_output_ruberta_base/rubist_second_trained/model.safetensors
All model checkpoint weights were used when initializing
BertForSequenceClassification.

All the weights of BertForSequenceClassification were initialized from the model
checkpoint at model_output_ruberta_base/rubist_second_trained.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use BertForSequenceClassification for predictions without
further training.
loading file sentencepiece.bpe.model
loading file tokenizer.json
loading file added_tokens.json
loading file special_tokens_map.json
loading file tokenizer_config.json
loading configuration file
```

```
model_output_xlm_roberta_base/rubist_trained/config.json
Model config XLMRobertaConfig {
  "_name_or_path": "model_output_xlm_roberta_base/rubist_trained",
  "architectures": [
    "XLMRobertaForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "eos_token_id": 2,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-05,
  "max_position_embeddings": 514,
  "model_type": "xlm-roberta",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "output_past": true,
  "pad_token_id": 1,
  "position_embedding_type": "absolute",
  "problem_type": "single_label_classification",
  "torch_dtype": "float32",
  "transformers_version": "4.46.3",
  "type_vocab_size": 1,
  "use_cache": true,
  "vocab_size": 250002
}

loading weights file
model_output_xlm_roberta_base/rubist_trained/model.safetensors
All model checkpoint weights were used when initializing
XLMRobertaForSequenceClassification.

All the weights of XLMRobertaForSequenceClassification were initialized from the
model checkpoint at model_output_xlm_roberta_base/rubist_trained.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use XLMRobertaForSequenceClassification for predictions without
further training.
```

[110]:
```python
def get_predictions(model, tokenizer, texts):
    preds = []
    for t in texts:
        enc = tokenizer(t, return_tensors="pt", truncation=True, padding=True)
        with torch.no_grad():
```

```
            logits = model(**enc).logits
        preds.append(int(torch.argmax(logits)))
    return preds
```

[117]:
```
from BERT_Models_Fine_Tuning_Russian import (data_loader, train_model,␣
 ↪evaluate_model)

_, test_data_rbsa = data_loader(csv_file_path='COMP0173_Data/rubist_second.
 ↪csv', labelling_criteria='stereotype', dataset_name='rubist_second',␣
 ↪sample_size=1000000, num_examples=5)
rubert_test = test_data_rbsa.groupby("category", group_keys=False).apply(
    lambda x: x.sample(n=100, random_state=42)
)

_, test_data_rbs = data_loader(csv_file_path='COMP0173_Data/rubist.csv',␣
 ↪labelling_criteria='stereotype', dataset_name='rubist', sample_size=1000000,␣
 ↪num_examples=5)
xlm_test = test_data_rbs.groupby("category", group_keys=False).apply(
    lambda x: x.sample(n=100, random_state=42)
)
```

```
First few examples from the training data:
     stereotype_type                                          text  category  \
1005          gender                                             1
1001          gender                                        0
2619     nationality                                          0
2213          gender                                   0
934           gender                                    0

          data_name
1005  rubist_second
1001  rubist_second
2619  rubist_second
2213  rubist_second
934   rubist_second
First few examples from the testing data:
     stereotype_type                                                text  \
80        profession
1814          gender
2277      profession                                  …
61        profession                                  …
755           gender                                 …

      category      data_name
80           0  rubist_second
1814         0  rubist_second
2277         0  rubist_second
```

```
61           0  rubist_second
755          0  rubist_second
Train data size: 2336
Test data size: 584
First few examples from the training data:
     stereotype_type                                                text  \
168       profession
2883     nationality                                 …
2919           lgbtq                                 …
3412      profession                                …
732            lgbtq

      category data_name
168          1    rubist
2883         0    rubist
2919         1    rubist
3412         0    rubist
732          1    rubist
First few examples from the testing data:
     stereotype_type                                                text  \
2726      profession                                 …
1066          gender                                …
2547     nationality                                …
1834      profession                                 …
1825     nationality                                 …

      category data_name
2726         0    rubist
1066         0    rubist
2547         0    rubist
1834         0    rubist
1825         0    rubist
Train data size: 3372
Test data size: 844
```

[118]:
```
from BERT_Models_Fine_Tuning import (data_loader)
_, test_data_mgsd = data_loader(csv_file_path='Model Training and Evaluation/
 ↪MGSD.csv', labelling_criteria='stereotype', dataset_name='MGSD',␣
 ↪sample_size=1000000, num_examples=5)

albert_test = test_data_mgsd.groupby("label", group_keys=False).apply(
    lambda x: x.sample(n=100, random_state=42)
)
```

```
First few examples from the training data:
        group                                                text  label  \
38518    race  Syria is a well known area nowadays. There is …      1
12979    race                   The people of germany are angry.      1
```

```
35494    race  Lebanon is a small country in the Middle East …      1
42213    race  We have a french tutor. She is very humble and…      0
16293  gender                                Every male acts dumb.      1

      data_name
38518      MGSD
12979      MGSD
35494      MGSD
42213      MGSD
16293      MGSD
First few examples from the testing data:
            group                                                text  label  \
17851        race      When i see an arab, i think they are a happy.      0
37449        race  People in India dress differently. The types o…      0
23634      gender    Everyone has a grandmother. The bridge is long.      0
36047        race  Cameroon is a country. It is a popular tourist…      0
28528  profession  The delivery man drove a truck. He was highly …      0

      data_name
17851      MGSD
37449      MGSD
23634      MGSD
36047      MGSD
28528      MGSD
Train data size: 41493
Test data size: 10374
```

[119]:
```python
# RBSA: rubert_test (RuBERT dataset)
rubert_test["text_en"] = rubert_test["text"].apply(
    lambda x: GoogleTranslator(source="ru", target="en").translate(x)
)

rubert_test["pred_albert"] = get_predictions(
    albert_model, albert_tokenizer, rubert_test["text_en"]
)

rubert_test["pred_rubert"] = get_predictions(
    rubert_model, rubert_tokenizer, rubert_test["text"]
)

print("ALBERT on translated RBSA")
print("Accuracy:", accuracy_score(rubert_test["category"],␣
 ↪rubert_test["pred_albert"]))
print("Macro F1:", f1_score(rubert_test["category"],␣
 ↪rubert_test["pred_albert"], average="macro"))
print(classification_report(rubert_test["category"],␣
 ↪rubert_test["pred_albert"]))
```

```
print("\nRuBERT on Russian RBSA")
print("Accuracy:", accuracy_score(rubert_test["category"],␣
 ↪rubert_test["pred_rubert"]))
print("Macro F1:", f1_score(rubert_test["category"],␣
 ↪rubert_test["pred_rubert"], average="macro"))
print(classification_report(rubert_test["category"],␣
 ↪rubert_test["pred_rubert"]))
```

```
Asking to truncate to max_length but no maximum length is provided and the model
has no predefined maximum length. Default to no truncation.
```

```
ALBERT on translated RBSA
Accuracy: 0.675
Macro F1: 0.6698412698412699
              precision    recall  f1-score   support

           0       0.64      0.80      0.71       100
           1       0.73      0.55      0.63       100

    accuracy                           0.68       200
   macro avg       0.69      0.68      0.67       200
weighted avg       0.69      0.68      0.67       200


RuBERT on Russian RBSA
Accuracy: 0.73
Macro F1: 0.7286704853783539
              precision    recall  f1-score   support

           0       0.70      0.80      0.75       100
           1       0.77      0.66      0.71       100

    accuracy                           0.73       200
   macro avg       0.73      0.73      0.73       200
weighted avg       0.73      0.73      0.73       200
```

[120]:
```
# RBS: xlm_test
xlm_test["text_en"] = xlm_test["text"].apply(
    lambda x: GoogleTranslator(source="ru", target="en").translate(x)
)

xlm_test["pred_albert"] = get_predictions(
    albert_model, albert_tokenizer, xlm_test["text_en"]
)

xlm_test["pred_xlm"] = get_predictions(
```

```
    xlm_model, xlm_tokenizer, xlm_test["text"]
)

print("\nALBERT on translated RBS")
print("Accuracy:", accuracy_score(xlm_test["category"],␣
 ↪xlm_test["pred_albert"]))
print("Macro F1:", f1_score(xlm_test["category"], xlm_test["pred_albert"],␣
 ↪average="macro"))
print(classification_report(xlm_test["category"], xlm_test["pred_albert"]))

print("\nXLM-R on Russian RBS")
print("Accuracy:", accuracy_score(xlm_test["category"], xlm_test["pred_xlm"]))
print("Macro F1:", f1_score(xlm_test["category"], xlm_test["pred_xlm"],␣
 ↪average="macro"))
print(classification_report(xlm_test["category"], xlm_test["pred_xlm"]))
```

```
ALBERT on translated RBS
Accuracy: 0.775
Macro F1: 0.7694613079228464
              precision    recall  f1-score   support

           0       0.71      0.93      0.81       100
           1       0.90      0.62      0.73       100

    accuracy                           0.78       200
   macro avg       0.80      0.78      0.77       200
weighted avg       0.80      0.78      0.77       200


XLM-R on Russian RBS
Accuracy: 0.975
Macro F1: 0.9749993749843746
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       100
           1       0.98      0.97      0.97       100

    accuracy                           0.97       200
   macro avg       0.98      0.97      0.97       200
weighted avg       0.98      0.97      0.97       200
```

[121]:
```
# Translate MGSD English → Russian
albert_test["text_ru"] = albert_test["text"].apply(
    lambda x: GoogleTranslator(source="en", target="ru").translate(x)
)
```

```
# ALBERT on original English MGSD
albert_test["pred_albert"] = get_predictions(
    albert_model, albert_tokenizer, albert_test["text"]
)

# RuBERT on translated Russian MGSD
albert_test["pred_rubert"] = get_predictions(
    rubert_model, rubert_tokenizer, albert_test["text_ru"]
)

# XLM-R on translated Russian MGSD
albert_test["pred_xlm"] = get_predictions(
    xlm_model, xlm_tokenizer, albert_test["text_ru"]
)

print("ALBERT on English MGSD")
print("Accuracy:", accuracy_score(albert_test["label"],␣
 ↪albert_test["pred_albert"]))
print("Macro F1:", f1_score(albert_test["label"], albert_test["pred_albert"],␣
 ↪average="macro"))
print(classification_report(albert_test["label"], albert_test["pred_albert"]))

print("\nRuBERT on translated MGSD (EN→RU)")
print("Accuracy:", accuracy_score(albert_test["label"],␣
 ↪albert_test["pred_rubert"]))
print("Macro F1:", f1_score(albert_test["label"], albert_test["pred_rubert"],␣
 ↪average="macro"))
print(classification_report(albert_test["label"], albert_test["pred_rubert"]))

print("\nXLM-R on translated MGSD (EN→RU)")
print("Accuracy:", accuracy_score(albert_test["label"],␣
 ↪albert_test["pred_xlm"]))
print("Macro F1:", f1_score(albert_test["label"], albert_test["pred_xlm"],␣
 ↪average="macro"))
print(classification_report(albert_test["label"], albert_test["pred_xlm"]))
```

```
=== ALBERT on English MGSD ===
Accuracy: 0.85
Macro F1: 0.8494580489763148
              precision    recall  f1-score   support

           0       0.81      0.91      0.86       100
           1       0.90      0.79      0.84       100

    accuracy                           0.85       200
   macro avg       0.86      0.85      0.85       200
```

```
weighted avg       0.86      0.85      0.85       200


=== RuBERT on translated MGSD (EN→RU) ===
Accuracy: 0.64
Macro F1: 0.616
              precision    recall  f1-score   support

           0       0.59      0.89      0.71       100
           1       0.78      0.39      0.52       100

    accuracy                           0.64       200
   macro avg       0.69      0.64      0.62       200
weighted avg       0.69      0.64      0.62       200


=== XLM-R on translated MGSD (EN→RU) ===
Accuracy: 0.55
Macro F1: 0.53125
              precision    recall  f1-score   support

           0       0.58      0.35      0.44       100
           1       0.54      0.75      0.62       100

    accuracy                           0.55       200
   macro avg       0.56      0.55      0.53       200
weighted avg       0.56      0.55      0.53       200
```

3.1.6 Failure Cases

Translation Noise
Literal translations can distort pragmatic cues, weakening or altering the original stereotype signals.

Overfitting in `XLM-Roberta`
`XLM-Roberta` performs poorly on English MGSD after being fine-tuned on Russian, indicating that:
- its strong performance in Russian reflects the memorisation of patterns generated by large language models,
- its multilingual representations are not sufficiently aligned for effective stereotype detection.

Limited Dataset Size and Diversity
The small scale of the RBS/RBSA (approximately 3,000 to 4,000 examples) limits the models' ability to learn stereotype trends, particularly in cross-lingual contexts.

Multi-stereotype Sentences
Sentences containing overlapping stereotypes (e.g., gender and profession) are often misclassified because models tend to default to the dominant pattern learned during training.

In overall, these failures indicate that cross-lingual stereotype detection relies not only on translation quality but also on cultural grounding, semantic robustness, and dataset diversity. Addressing these

challenges requires larger Russian datasets and improved human-supervised annotation. The RBS experiment also showed that adding zero-shot examples to large language model prompts can lead to repetitive, easily recognisable patterns that models may overfit. In the RBSA dataset, the issue was that the LLM retained some negative sentiment when generating "neutral" sentences.

4 References

[1] Theo King, Zekun Wu, Adriano Koshiyama, Emre Kazim, and Philip Treleaven. 2024. HEARTS: A holistic framework for explainable, sustainable and robust text stereotype detection. arXiv preprint arXiv:2409.11579. Available at: https://arxiv.org/abs/2409.11579 (Accessed: 4 December 2025). https://doi.org/10.48550/arXiv.2409.11579

[2] Theo King, Zekun Wu, Adriano Koshiyama, Emre Kazim, and Philip Treleaven. 2024. HEARTS-Text-Stereotype-Detection (GitHub Repository). Available at: https://github.com/holistic-ai/HEARTS-Text-Stereotype-Detection (Accessed: 4 December 2025).

[3] Theo King, Zekun Wu, Adriano Koshiyama, Emre Kazim, and Philip Treleaven. Holistic AI. 2024. EMGSD: Expanded Multi-Group Stereotype Dataset (HuggingFace Dataset). Available at: https://huggingface.co/datasets/holistic-ai/EMGSD (Accessed: 4 December 2025).

[4] University College London Technical Support Group (TSG). 2025. GPU Access and Usage Documentation. Available at: https://tsg.cs.ucl.ac.uk/gpus/ (Accessed: 6 December 2025).

[5] United Nations. 2025. The 2030 Agenda for Sustainable Development. Available at: https://sdgs.un.org/2030agenda (Accessed: 6 December 2025).

[6] Veronika Grigoreva, Anastasiia Ivanova, Ilseyar Alimova, and Ekaterina Artemova. 2024. RuBia: A Russian Language Bias Detection Dataset. Available at: https://arxiv.org/abs/2403.17553 (Accessed: 9 December 2025).

[7] Veronika Grigoreva, Anastasiia Ivanova, Ilseyar Alimova, and Ekaterina Artemova. 2024. RuBia-Dataset (GitHub Repository). Available at: https://github.com/vergrig/RuBia-Dataset (Accessed: 9 December 2025).

[8] Sismetanin. 2020. Toxic Comments Detection in Russian (GitHub Repository). Available at: https://github.com/sismetanin/toxic-comments-detection-in-russian (Accessed: 9 December 2025).

[9] DeepPavlov. 2019. RuBERT-base-cased (Hugging Face Model). Available at: https://huggingface.co/DeepPavlov/rubert-base-cased (Accessed: 9 December 2025).

[10] AI-Forever. 2023. RuBERT-base (Hugging Face Model). Available at: https://huggingface.co/ai-forever/ruBert-base (Accessed: 9 December 2025).

[11] Hugging Face. 2024. XLM-RoBERTa: Model Documentation. Available at: https://huggingface.co/docs/transformers/en/model_doc/xlm-roberta (Accessed: 9 December 2025).

[12] DeepPavlov. 2020. ruBERT-base-cased-sentence (Hugging Face Model). Available at: https://huggingface.co/DeepPavlov/rubert-base-cased-sentence (Accessed: 9 December 2025).