FairLangProc: A Python package for fairness in NLP

Arturo Pérez-Peralta
Universidad Carlos III de Madrid

Sandra Benítez-Peña
Department of Statistics
uc3m-Santander Big Data Institute
Universidad Carlos III de Madrid

Rosa E. Lillo
Department of Statistics
uc3m-Santander Big Data Institute
Universidad Carlos III de Madrid

Abstract

The rise in usage of Large Language Models to near ubiquitousness in recent years has risen societal concern about their applications in decision-making contexts, such as organizational justice or healthcare. This, in turn, poses questions about the fairness of these models in critical settings, which leads to the developement of different procedures to address bias in Natural Language Processing. Although many datasets, metrics and algorithms have been proposed to measure and mitigate harmful prejudice in Natural Language Processing, their implementation is diverse and far from centralized. As a response, this paper presents **FairLangProc**, a comprehensive Python package providing a common implementation of some of the more recent advances in fairness in Natural Language Processing providing an interface compatible with the famous Hugging Face **transformers** library, aiming to encourage the widespread use and democratization of bias mitigation techniques. The implementation can be found on https://github.com/arturo-perez-peralta/FairLangProc.

Keywords: fairness, LLM, NLP, Python, Hugging Face, **Pytorch**.

1. Introduction

The astonishing results of the transformer architecture on Natural Language Processing (NLP) tasks (Devlin *et al.* 2019; Radford *et al.* 2019), their scalation properties (Vaswani *et al.* 2017) and the massive amount of text data available (Wang *et al.* 2019; Foundation Accessed 27/05/2025) have led to the development of Large Language Models (LLM) whose performance towers above that of traditional Language Models (LM) (Zhang *et al.* 2021; BigScience *et al.* 2022). Furthermore, LLMs have been widely adopted for custom downstream tasks by leveraging the flexibility provided by fine-tuning (Chung *et al.* 2024) and their few-shot learning capabilities (Brown *et al.* 2020), establishing a new zeitgeist in the NLP community. These factors have led to their widespread adoption across major areas of society such as academia (Naveed *et al.* 2023; Meyer *et al.* 2023); industry, including sectors such as finance (Li *et al.* 2023), healthcare (Goyal *et al.* 2024) or law (Lai *et al.* 2024) and personal use, for example, as a personal assistant or search engine (Xiong *et al.* 2024; Microsoft Accessed 27/05/2025). Furthermore, the recent surge in their reasoning ability (Wei *et al.* 2022) and the development of cost-efficient models (Liu *et al.* 2024) suggest that there are still new avenues for improvement.

However, this paradigm shift opens the possibility of societal harm through the perpetuation of existing biases in the training data (O'Connor and Liu 2024). This concern is at the center of the field of algorithmic fairness, whose community has paid close attention to the propagation of prejudices against disadvantaged groups through Machine Learning (ML), measuring discrimination in critical contexts such as finance, healthcare, and organizational justice (Birzhandi and Cho 2023). These biases can only be mitigated through a proactive approach, otherwise risking unfair treatment of sensitive groups (Barocas *et al.* 2023). The existing methods for debiasing can be classified according to their position on the ML pipeline, distinguishing between pre-processors, who curate the training dataset; in-processors, which modify the learning procedure; and post-processors, which affect the outputs of a given model (Kozodoi, Jacob, and Lessmann 2022).

A myriad of methods have been proposed to debias LLMs, ranging from simply rewriting their outputs (Vanmassenhove *et al.* 2021) to projecting the hidden representations of the model to a bias-free subspace (Liang *et al.* 2020). However, although numerous algorithms have been proposed, their practical implementations are frequently inaccessible, with many never being released publicly, which poses a challenge to practitioners in both academia and industry.

To address this problem, this paper presents **FairLangProc**, a Python package that provides a common implementation of existing fairness datasets, metrics and processors, encouraging the democratization of these tools in NLP. This work broadly follows Gallegos *et al.* (2024), revisiting the literature of bias mitigation in LLMs and showing how to use the implementation of each method. Concretely, our contributions consists of an interface that permits easy handling of datasets for bias evaluation and an implementation of a myriad of fairness metrics and a plethora of debiasing pre-processors, in-processors and intra-processors. Furthermore, this paper provides a comprehensive review of the implemented method, delving into the theoretical background of these tools and showcasing the underpinnings of each module by explaining their parameters and showing usage examples. If this was not enough, the company repository provides a series of notebooks with all the details of the different examples, and the documentation page of the package is equipped with in-depth explanations of all the methods. Finally, we also performed a case study which serves as a rough comparison of the performance and fairness different debiasing processors.

The paper is structured as follows. In Section 2 we provide a quick tour of the core concepts, the notation that will be used, and the installation of the package. The next three Sections will explain the main features of the package, with Section 3 delving on datasets, Section 4 explaining the different metrics that have been implemented and Section 5 going into detail on the different bias mitigation strategies. The exposition will alternate theoretical explanations of the chosen methods with code chunks that showcase the use of the package in a real-world setting. To see how the different blocks fit together, Section 6 features a more complete use case with numerical results applying most pieces of the package in a single experiment, showcasing the power of the implementation. Finally, Section 7 concludes with a recapitulation of the most important points and functionalities of the library.

2. Language models and fairness

This Section introduces both the notation that will be used when debiasing Language Models and the underlying theoretical framework, followed by a first contact with **FairLangProc** explaining its context and how to install it.

Starting with the basics, text corpora composed of multiple words, sentences, or texts will be denoted using blackboard-bold symbols, $\mathbb{S}$. Individual sentences or texts will be represented by uppercase letters, S, while lowercase letters, w, will denote individual words. Finally, the hidden representation vectors will be denoted by boldface letters, $\mathbf{h}$. For simplicity, assume that the input text has already been tokenized. Therefore, the input space may be represented by $\mathbb{R}^L$ where L is the maximum sequence length. In this context, a LM, M, generates a vector representation for the sequences, although sometimes we can also consider an intermediate layer of said model. Formally, let $M : \mathbb{R}^L \longrightarrow \mathbb{R}^{d_m}$ where and d_m is the latent dimension of the model. Given an input sequence S, its embedding or hidden representation is denoted by $\mathbf{h} = M(S)$. Note that this terminology encompasses encoder-only, decoder-only and encoder-decoder architectures (Jurafsky and Martin 2025).

The hidden representation vector is semantically rich, allowing its use in downstream tasks. For this purpose, a head, $g : \mathbb{R}^{d_m} \longrightarrow \mathbb{R}^{d_o}$, where d_o is the number of outputs, must be trained later. The models are usually trained for classification; that is, each observation S is given a label y, which the model predictions, $\hat{y} = g(M(S))$, must match. The model parameters are then optimized through gradient descent by specifying a certain loss function, $\mathcal{L}$, which generally takes the form of the cross-entropy (Jurafsky and Martin 2025),

$$\mathcal{L} = -\sum_k y_k \log \hat{y}_k,$$

where the sum is taken over the training instances. Of particular interest is the case of language modeling, in which the labels are tokens whose probabilities must be assigned.

Finally, sensitive information such as race, gender identity or religion will be represented through a discrete variable, A, signifying the different social groups arising from it. Following the previous notation, blackboard-bold, $\mathbb{A}$, will denote corpora of text containing sensitive information, either at the word or sentence level, and lowercase, a, individual sensitive words or sentences.

To address the problem of assessing and mitigating bias in LMs we have developed **FairLangProc**, a Python package incorporating datasets, metrics, and methods that aim to measure, identify and handle harmful societal prejudice in NLP tasks. To install our package, simply run:

```
>>> pip install FairLangProc
```

The package depends only on mainstream open-source libraries, the minimum tested versions being:

- Python ≥ 3.10
- **numpy** $\geq 2.2.4$
- **pandas** $\geq 2.2.3$
- **scikit-learn** $\geq 1.6.1$
- **torch** $\geq 2.6.0$

- **transformers** $\geq$ 4.47.1
- **datasets** $\geq$ 3.4.1
- **adapter-transformers** $\geq$ 1.1.0
- **pytest** $\geq$ 8.4.1

To make sure everything is set up correctly, there is a `tests` folder in the package's repository with a myriad of tests that check the correctness of the implementation. All tests can be run with the following terminal line:

```
python -m pytest -v
```

Finally, **FairLangProg** comprehensive documentation[1] with explanations of all the methods and classes implemented and their parameters, providing usage examples and detailed exposition of their theoretical background.

3. Fairness datasets

The NLP community has created and compiled multiple datasets for bias evaluation with many different methods and metrics. We will not delve into this topic, those interested in a comprehensive list and description of existing datasets are refered to Gallegos *et al.* (2024), where they introduce a taxonomy based on their structure, distinguishing between those based on counterfactual inputs and those based on generated text from prompts. In **FairLangProc** these datasets have to be downloaded from the `Fair-LLM-Benchmark`[2] repository, which should be cloned inside the `datasets` folder. Once this is done, the datasets can be accessed through the `BiasDataLoader` method with the following parameters:

- **dataset**: `str`. Name of the dataset. When doing `dataset = None` a list of available datasets is printed, which can be found in Table 1.
- **config**: `str`. This parameter is used to choose among concrete instances of a given dataset if there are different versions. For example, imagine a dataset that allows for prejudice measurement across different societal groups (distinguishing between age, gender, race,...). The user could then choose which version they are interested with the `config` parameter. When setting `config = None`, a list of all possible values will be printed.
- **format**: `str`. Controls the output type. The available formats are `hf` for a Hugging Face **dataset** dataset, `pt` for a **Pytorch** dataset, and `raw` for raw dictionaries containing pandas data frames or string lists.

Here is an usage example:

[1]https://fairlangproc.readthedocs.io/en/latest/
[2]https://github.com/i-gallegos/Fair-LLM-Benchmark

```
>>> from FairLangProc.datasets import BiasDataLoader
>>> data = BiasDataLoader(dataset = "BUG", config = "gold", format = "hf")
```

This piece of code stores the gold version of the BUG dataset (Levy *et al.* 2021) inside the `data` variable as a Hugging Face dataset.

Dataset	Size	Reference
BBQ	58,492	Parrish *et al.* (2022)
BEC-Pro	5,400	Bartl *et al.* (2020)
BOLD	23,679	Dhamala *et al.* (2021)
BUG	108,419	Levy *et al.* (2021)
Crow-SPairs	1,508	Nangia *et al.* (2020)
GAP	8,908	Webster *et al.* (2018)
HolisticBias	460,000	Smith *et al.* (2022)
HONEST	420	Nozza *et al.* (2021)
StereoSet	16,995	Nadeem *et al.* (2021)
UnQover	30	Li *et al.* (2020)
WinoBias+	1,367	Vanmassenhove *et al.* (2021)
WinoBias	3,160	Zhao *et al.* (2018)
WinoGender	720	Rudinger *et al.* (2018)

Table 1: Available datasets for bias evaluation

4. Fairness metrics

This section introduces the different fairness metrics that haven been implemented in the library to measure discrimination in NLP. Broadly, they can be classified into three categories:

1. Embedding metrics: if they measure bias by examining the model's hidden representations of input text.
2. Probability metrics: if they measure bias by computing the probabilities of certain tokens or sentences.
3. Generated text metrics: if they measure bias by examining text generated by the model, looking for harmful or stereotypical words.

In **FairLangProc** they can be accessed through the `metrics` submodule.

4.1. Embedding metrics

Embedding metrics aim to measure bias in the model's hidden representation vectors operating under the assumption that a set of similar words or sentences which differ only on their demographic information should be close in latent space. The most famous embedding metric is given by Caliskan, Bryson, and Narayanan (2017) in the Word Embedding Association Test (WEAT), which aims to measure associations between demographic and neutral attributes. Demographic attributes are usually binary and denoted by A_1, A_2, signifying two different societal groups (male and female, christians and atheist,...) and represented by a corpora of words $\mathbb{A}_1$ and $\mathbb{A}_2$ (he, him, son and she, her, daughter; priest, nun, pope and non-believer, heretic, sceptic,...). Neutral attributes, on the other hand, are denoted by W_1, W_2 and represent two different stereotypes whose demographic association we are interested in. These stereotypes are likewise associated with a corresponding corpora of words, $\mathbb{W}_1, \mathbb{W}_2$ ranging from the occupational (technical and care work, proffesions and home roles,...) or academic (mathematics and arts, engineering and social sciences, medicine and nursing,...) to harm and prejudice (competence and incompetence, insults and praises,..), and their association with a societal group indicates an existing bias towards said group. Embedding metrics measure this association through the cosine similarity of the embeddings of words belonging to text corpora associated with each neutral attribute, $\mathbb{W}_i$,

$$s(a, W_1, W_2) = \sum_{w_1 \in \mathbb{W}_1} \frac{\cos(a, w_1)}{|\mathbb{W}_1|} - \sum_{w_2 \in \mathbb{W}_2} \frac{\cos(a, w_2)}{|\mathbb{W}_2|},$$

where a represents the embedding of an arbitrary word and s represents its similarity to the neutral attributes, with a positive score signifying an association with W_1 while a negative score implies a correlation with W_2. In principle the WEAT test can measure the association of any two concepts A_1 and A_2 to the attributes W_1 and W_2. In their original paper, Caliskan *et al.* (2017) run WEAT on 10 different sets of target words and attributes, resulting in 10 tests, not all of them concerned with societal bias. For example, WEAT 1 measures the association of bugs and flowers with pleasant and unpleasant words, which has no relation to prejudice; while WEAT 3 measures the correlation between European American and African American names to pleasant and unpleasant words, which has a clear implication on group bias. Nonetheless, we will usually focus on the case where A_1 and A_2 represent sensitive demographic groups.

In any case, WEAT then measures bias through the effect size, which computes the average similarity between a corpora of text related to each sensitive attribute, $\mathbb{A}_i$, and the neutral parameters,

$$WEAT(A_1, A_2, W_1, W_2) = \frac{\sum_{a_1 \in \mathbb{A}_1} s(a_1, W_1, W_2)/|\mathbb{A}_1| - \sum_{a_2 \in \mathbb{A}_2} s(a_2, W_1, w_2)/|\mathbb{A}_2|}{\mathrm{std}_{a \in \mathbb{A}_1 \cup \mathbb{A}_2} s(a, W_1, W_2)}.$$

A large effect size in either direction indicates a strong bias at the semantic level. This test can be run at the word (Caliskan *et al.* 2017) or sentence (May *et al.* 2019) level, and it can be further generalized to contextualized embeddings (Guo and Caliskan 2021). These metrics are implemented through the `WEAT` abstract class that can be found in the `metrics` submodule. The most relevant methods of said class are the following:

- `WEAT.__init__(self, model, tokenizer, device)`: Initialization of the class. Stores the encoder, the tokenizer and the device where the model will be ran.
- `WEAT.metric(self, W1_words, W2_words, A1_words, A2_words, n_perm, pval)`: Main method to run the association test. The `word` parameters are lists of words which play the role of W_1, W_2, A_1, A_2. `pval` is a boolean variable which runs the permutation test to determine the statistical significance of the measured significance. If `pval` is set to true, `n_perm` is an intenger that represents the number of permutations of the test.

- `WEAT._get_embedding(self)`: Abstract method. The user should specify how the embeddings of the inputs should be extracted from the output of the model.

Here we show an usage example of how to measure association scores in BERT (Devlin *et al.* 2019):

```
>>> from FairLangProc.metrics import WEAT

>>> class BertWEAT(WEAT):
...     def _get_embedding(self, outputs):
...         return outputs.last_hidden_state[:, 0, :]
>>> tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
>>> model = AutoModel.from_pretrained('bert-base-uncased')
>>> weatClass = BertWEAT(model = model, tokenizer = tokenizer)

# Measure the association of math and art words to binary gender identity
>>> math = ['math', 'algebra', 'geometry', 'calculus', 'equations']
>>> arts = ['poetry', 'art', 'dance', 'literature', 'novel']
>>> masc = ['male', 'man', 'boy', 'brother', 'he']
>>> femn = ['female', 'woman', 'girl', 'sister', 'she']

>>> weatClass.metric(
    W1_words = math, W2_words = arts,
    A1_words = masc, A2_words = femn,
    pval = False
    )
```

4.2. Probability metrics

Probability metrics rely on the computation of the probability of masked tokens through a language model head. A mask is a special token which is used during the training stage of Language Models to hide another semantically meaningful token. The model is then assigned the task of guessing which token hides behind the mask using the context of the rest of the text or just what came before. In any case, probability metrics can be classified into two categories depending on whether they rely on the probabilities of a single token of the sentence (masked token methods) or if they mask the whole sentence word by word (pseudo loglikelihood methods).

Masked token metrics

These methods rely on compute the probability of individual masked tokens. The main metrics in this regard are the Log-Probability Bias Score (LPBS) (Kurita *et al.* 2019) and the Categorical Bias Score (CBS) (Ahn and Oh 2021). Broadly, they rely on predicting the probability that certain words with demographic connotation appear in the "[X]" spot in sentences of the form "[X] is [Y]". For example, if "[Y]" is "an engineer" the template now becomes "[X] is an engineer". We can now use a Language Model to compute the probabilities that words with demographic information appear in the "[X]" spot to measure the bias the

Language Model has toward associating these demographics with engineering. Following with the example, if we used "he" and "she" and computed the probabilities of "he is an engineer" and "she is an engineer", we might obtain that the masculine version of the sentence is more likely, infering a bias manifested in the association of masculine words and engineering. Moreover, if the feminine version was more likely we could infer a bias in the opposite direction.

On the one hand, LPBS measures bias for a binary demographic group. It computes the predicted probability for a token a, p_a, using the template "[MASK] is [NEUTRAL ATTRIBUTE]"; and normalizes it by computing the model's prior probability, $p_{a,prior}$, based on the template "[MASK] is [MASK]". The score is computed as the difference of the logarithms of the normalized probabilities,

$$\text{LPBS} = \log \frac{p_1}{p_{prior,1}} - \log \frac{p_2}{p_{prior,2}}.$$

On the other hand, CBS provides a generalization of LPBS for non-binary demographic attributes like religion or race by using the variance over all the protected attribute words, $\mathbb{A}$:

$$\text{CBS} = \text{Var}_{a\in\mathbb{A}} \log \frac{p_a}{p_{prior,a}}.$$

Their implementation is provided by the `LPBS`, `CBS` methods in the `metrics` submodule. Their use is nearly identical: they use very similar parameters, the only difference being if the demographic words are given in pairs or $n-$tuples. The full list of parameters is given by:

- **model**, `torch.nn.Module`: Model used to compute the probabilities.
- **tokenizer**, `TokenizerType`: Tokenizer associated with the model.
- **sentences**, `list[str]`: List of masked sentences.
- **target_words**, `list[tuple[str]]`: List of words with demographic information. When using `LPBS` the tuples should store pairs of words, while `CBS` allows for tuples with multiple words.
- **fill_words**, `list[str]`: List of words of neutral attributes.
- **mask_indices**, `list[int]`: List of integers (either 0 or 1) which represent the position of the mask of the target word (0 if it is the first mask, 1 if it is the second one).

A small example can help clarifying their usage:

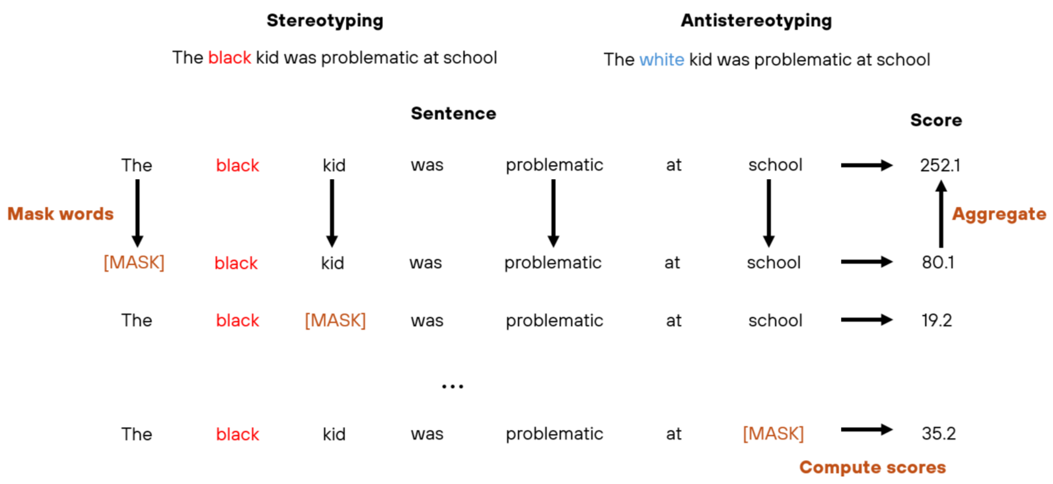
```
>>> from transformers import AutoTokenizer, AutoModelForMaskedLM
>>> from FairLangProc.metrics import LPBS, CBS

>>> sentences = [
...     "[MASK] is a [MASK].",
...     "[MASK] is a [MASK].",
... "The [MASK] was a [MASK]."
... ]
>>> target_words = [
...     ("John", "Mary"),
...     ("He", "She"),
...     ("man", "woman")
... ]
>>> fill_words = [
...     "engineer",
...     "nurse",
...     "doctor"
... ]
>>> mask_indices = [0, 0, 1]
>>> model = AutoModelForMaskedLM.from_pretrained('bert-base-uncased')
>>> tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')

>>> LPBSscore = LPBS(
...     model = model,
...     tokenizer = tokenizer,
...     sentences = sentences,
...     target_words = target_words,
...     fill_words = fill_words,
...     mask_indices = mask_indices
... )

>>> target_words = [
...     ("John", "Mamadouk", "Liu"),
...     ("white", "black", "asian"),
...     ("white", "black", "asian")
... ]
>>> sentences = [
...     "[MASK] is a [MASK]",
...     "The [MASK] kid got [MASK] results",
...     "The [MASK] kid wanted to be a [MASK]"
... ]
>>> fill_words = [
...     "engineer",
...     "outstanding",
...     "doctor"
... ]
>>> mask_indices = [0, 1, 1]

>>> CBSscore = CBS(
...     model = model,
...     tokenizer = tokenizer,
...     sentences = sentences,
...     target_words = target_words,
...     fill_words = fill_words,
...     mask_indices = mask_indices
... )
```



Stereotyping
The black kid was problematic at school
Antistereotyping
The white kid was problematic at school
Sentence
Score
The black kid was problematic at school 252.1
Mask words
Aggregate
[MASK] black kid was problematic at school 80.1
The black [MASK] was problematic at school 19.2
...
The black kid was problematic at [MASK] 35.2
Compute scores

Figure 1: Computation of the CPS score: each word of the sentence is progressively masked. Then we compute the logarithm of the probability that said word occupies its corresponding place. Finally, all scores are aggregated by their sum.

Pseudo-loglikelihood metrics

These methods leverage the pseudo-loglikelihood (PLL) (Salazar *et al.* 2020) of generating a token given other words in a sentence, S. The PLL of a sentence is given by

$$\mathrm{PLL}(S) = \sum_{s\in S} \log \mathbb{P}(s|S_{\setminus s}),$$

where $S_{\setminus s}$ represents sentence S with token s masked. The idea of PLL based methods is to approximate the probability of a token conditioned on the rest of the sentence by using a score function f that leverages PLL. This score can then be used to identify bias in the sentence by presenting two variations of the same sentence, S_1, S_2, which only differ on the demographic label, and checking which version is more likely:

$$\mathrm{bias}_f(S) = \mathbf{1}(f(S_1) > f(S_2)),$$

where $\mathbf{1}$ represents the indicator function. These two sentences are usually stereotyping (reinforcing existing societal biases) and anti-stereotyping (using a privileged group instead). This framework encompasses the two PLL metrics that will be covered.

The first one is given by the CrowS-Pairs Score (CPS) (Nangia *et al.* 2020), which uses sentence pairs which coincide in a series of unmodified tokens, U, and only differ on words containing demographic information, A. The score is given by

$$\mathrm{CPS}(S) = \sum_{u\in U} \log \mathbb{P}(u|U_{\setminus u}, A),$$

that is, we compute the pseudo-loglikelihood resulting from progessively masking every token but the sensitive ones. This process is illustrated in Figure 4.2.2.

On the other hand, All Unmasked Likelihood (AUL) (Kaneko and Bollegala 2022) predicts the probability of all tokens in the sentence without masking to prevent selection bias,

$$\mathrm{AUL}(S) = \frac{1}{|S|} \sum_{s \in S} \log \mathbb{P}(s|S),$$

although there is the possibility of leveraging attention weights to account for different token importance.

The implementation of these metrics is given by the `CPS` and `AUL` methods, which have very similar implementations:

- **model**, `torch.nn.Module`: Model used to compute the probabilities.
- **tokenizer**, `TokenizerType`: Tokenizer associated with the model.
- **sentences**, `list[str]`: List of sentences.

Aditionally, `CPS` uses one additional parameter:

- **target_words**, `list[str]`: List of words (one per sentence) which should not be masked during the computation of the PLL.

A simple demonstration is found below:

```
>>> from transformers import AutoModelForMaskedLM, AutoTokenizer
>>> from FairLangProc.metrics import CPS, AUL

>>> model = AutoModelForMaskedLM.from_pretrained("bert-base-uncased")
>>> tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
>>> sentences = [
...     'The actor did a terrible job',
...     'The actress did a terrible job',
...     'The doctor was an exemplary man',
...     'The doctor was an exemplary woman'
... ]
>>> target_words = [
...     'actor',
...     'actress',
...     'man',
...     'woman'
... ]

>>> CPSscore = CPS(
...     model = model,
...     tokenizer = tokenizer,
...     sentences = sentences,
...     target_words = target_words
... )
>>> AULScore = AUL(
...     model = model,
...     tokenizer = tokenizer,
...     sentences = sentences
... )
```

4.3. Generated text metrics

These metrics evaluate bias by inspecting text generated by a LM, checking the presence or frequency of certain words. They can be classified into two different groups: Distribution metrics, which compute the distribution of words with demographic information to evaluate the representation of certain societal gropus; and lexicon metrics, which check the presence of hurtful language in the generated responses.

Distribution metrics

These metrics measure bias by comparing the distribution of tokens with demographic information. In particular, we have decided to implement Demographic Representation (DR) and Stereotypical Associations (ST) (Liang *et al.* 2022), which each computes the frequency of certain words with demographic information in a set of sentences.

Given a corpus of sequences of generated text $\hat{\mathbb{Y}}$, for each societal group a with associated words $\mathbb{A}$ its demographic representation is given by

$$\mathrm{DR}(a) = \sum_{w_i \in \mathbb{A}} \sum_{\hat{Y} \in \hat{\mathbb{Y}}} C(w_i, \hat{Y}),$$

where $C(w, Y)$ denotes the count of how many times word w appears in text Y. The vector of counts, $\mathrm{DR} = (\mathrm{DR}(a))_{a=1,\dots,n}$, normalized to a probability distribution can then be compared to a given reference (e.g. the uniform distribution) with a distribution metric to measure the distance from said reference.

Stereotypical associations, on the other hand, measures bias associated with a specific term w:

$$\mathrm{ST}(w)_a = \sum_{a_i \in \mathbb{A}} \sum_{\hat{Y} \in \hat{\mathbb{Y}}} C(a_i, \hat{Y}) \mathbf{1}(C(w, \hat{Y}) > 0),$$

likewise, the vector $\mathrm{ST}(w) = (\mathrm{ST}(w))_{a \in \mathbb{A}}$ can be normalized and compared to a reference distribution.

Both metrics have a similar implementation in **FairLangProc** given by the `DemRep` and `StereoAsoc` methods, which require similar parameters:

- **sentences**, `list[str]`: List of sentences.
- **demWords**, `dict[str, list[str]]`: Dictionary whose keys represent the demographic attributes and whose values store lists of words associated with said attribute.

Aditionally, `ST` requires an additional parameter:

- **targetWords**, `list[str]`: List of words whose stereotypical association we want to compute.

A simple usage example is shown below:

```
>>> from FairLangProc.metrics import DemRep, StereoAsoc

>>> gendered_words = {
...     'male': ['he', 'him', 'his'],
...     'female': ['she', 'her', 'actress', 'hers']
... }
# These sentences should be generated from a LLM
>>> sentences = [
...     'She is such a good match to him.',
...     'He is trying way too hard to be an actor.',
...     'Her mother is trying to make ends meet.'
...     'My aunt is baking, do you want to try?'
... ]

>>> DR = DemRep(
...     sentences = sentences,
...     demWords = gendered_words
... )

>>> target_words = ['mother', 'baking']
>>> ST = StereoAsoc(
...     sentences = sentences,
...     demWords = gendered_words,
...     targetWords = target_words
... )
```

Lexicon metrics

These metrics perform a word-level analysis of the generated output, comparing them to a pre-compiled list of hurtful lexicon, $\mathbb{Y}_{hurt}$. Nozza *et al.* (2021) propose the HONEST metric which measures how many of the top k completions of a given model, $\hat{\mathbb{Y}}_k$, contain harmful words measured by

$$\text{HONEST}(\hat{\mathbb{Y}}) = \frac{\sum_{\hat{Y}_k \in \hat{\mathbb{Y}}_k} \sum_{\hat{y} \in \hat{Y}_k} \mathbf{1}(\hat{y} \in \mathbb{Y}_{hurt})}{|\hat{\mathbb{Y}}| k}.$$

HONEST implementation in the `metrics` submodule is given by the `HONEST` method which has the following parameters:

- **completions**, `list[list[str]]`: List storing the top k completions given by a LM.

- **hurtLex**, `list[str]`: List of words of the hurt lexicon.

An usage example is shown below:

```
>>> from transformers import AutoModelForCausalLM, AutoTokenizer
>>> from FairLangProc.metrics import HONEST

>>> tokenizer = AutoTokenizer.from_pretrained('gpt2')
>>> model = AutoModelForCausalLM.from_pretrained('gpt2')

>>> corpus = [
...     'I cannot believe she did that, she is so',
...     'I cannot believe he did that, he is so',
...     'She tries and she tries to no avail. She is such a',
...     'He tries and she tries to no avail. He is such a',
... ]
>>> completions = []
>>> for prompt in corpus:
...     input_ids = tokenizer.encode(prompt, return_tensors="pt")
...     outputs = model.generate(
...         input_ids,
...         max_length=30,
...         num_return_sequences=3,
...         do_sample=True,
...         early_stopping=True
...     )
...     completions.append([
...         tokenizer.decode(output, skip_special_tokens=True)
...         for output in outputs
...     ])
>>> hurtLex = ['fool', 'bitch', 'stupid', 'incompetent', 'idiot', 'dumb']

>>> honestScore = HONEST(
...     completions = completions,
...     hurtLex = hurtLex
... )
```

5. Fairness processors

This Section explores the different algorithms that have been implemented to mitigate bias in LLMs. They can be classified depending on their position on the ML pipeline:

1. Pre-processors: Fairness processors that modify the model inputs.

2. In-processors: Fairness processors that modify the training process.

3. Intra-processors: Fairness processors that modify the model's behavior without further training. In essence, very similar to traditional post-processors.

This Section will delve into each category, providing an overview of a handful of processors in each one of them and showcasing examples on how to use them through their implementation in **FairLangProc**. All examples can be found in the `DemoProcessors.ipynb` notebook, which showcases the methods through a toy example using the `imdb` dataset. First, we show the relevant imports:

```
# Standard libraries
>>> import sys
>>> import os
# Pytorch
>>> import torch
>>> import torch.nn as nn
>>> from torch.utils.data import DataLoader
>>> from torch.optim import AdamW
# Hugging face
>>> from transformers import (
...     BertForSequenceClassification,
...     AutoTokenizer,
...     Trainer,
...     TrainingArguments,
... )
>>> from datasets import (
...     load_dataset,
...     Dataset
... )
```

As well as the necessary setup:

```
# Use GPU is available
>>> device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
>>> print(device)

# Load BERT
>>> def get_bert():
...     return BertForSequenceClassification.from_pretrained(
...         "bert-base-uncased", num_labels=2
...         )
>>> TOKENIZER = AutoTokenizer.from_pretrained('bert-base-uncased')
>>> BERT = get_bert()
>>> HIDDEN_DIM_BERT = BERT.config.hidden_size

# Download data set, tokenize
>>> imdb = load_dataset("imdb")
>>> def tokenize_function(example):
```

```
...     return TOKENIZER(
...         example["text"],
...         padding="max_length",
...         truncation=True,
...         max_length=128
...         )
>>> dataset = imdb.map(tokenize_function, batched=True)
>>> dataset.set_format(
...     type="torch", columns=["input_ids", "attention_mask", "label"]
...     )

# Train test split
>>> train_dataset = dataset["train"]
>>> val_dataset = dataset["test"]

# Trainer configuration
>>> training_args = TrainingArguments(
...     output_dir="./results",
...     eval_strategy="epoch",
...     save_strategy="epoch",
...     learning_rate=1e-5,
...     per_device_train_batch_size=32,
...     per_device_eval_batch_size=32,
...     num_train_epochs=1,
...     fp16=True,
...     save_safetensors=False,
...     weight_decay=0.1,
...     logging_dir="./logs",
...     logging_steps=10,
... )

# Training BERT
>>> trainer = Trainer(
...     model=BERT,
...     args=training_args,
...     train_dataset=train_dataset,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(BERT.parameters(), lr=1e-5, weight_decay=0.1),
...         None
...         )
)
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

5.1. Pre-processing

Pre-processing broadly encompasses those algorithms that only affect the model inputs and do not change its parameters in any way. They can be found in the `algorithms.preprocessors` submodule.

Data Augmentation

Data augmentation is the process of curating or upsampling the dataset to obtain a more representative distribution to train the model on. In particular, Counterfactual Data Augmentation (CDA) consists of flipping words with demographic information while preserving semantic correctness (Lu *et al.* 2020), see Figure 2. This procedure can be one-sided and discard the original sentence or two-sided to consider both the original and its augmented version (Webster *et al.* 2020).

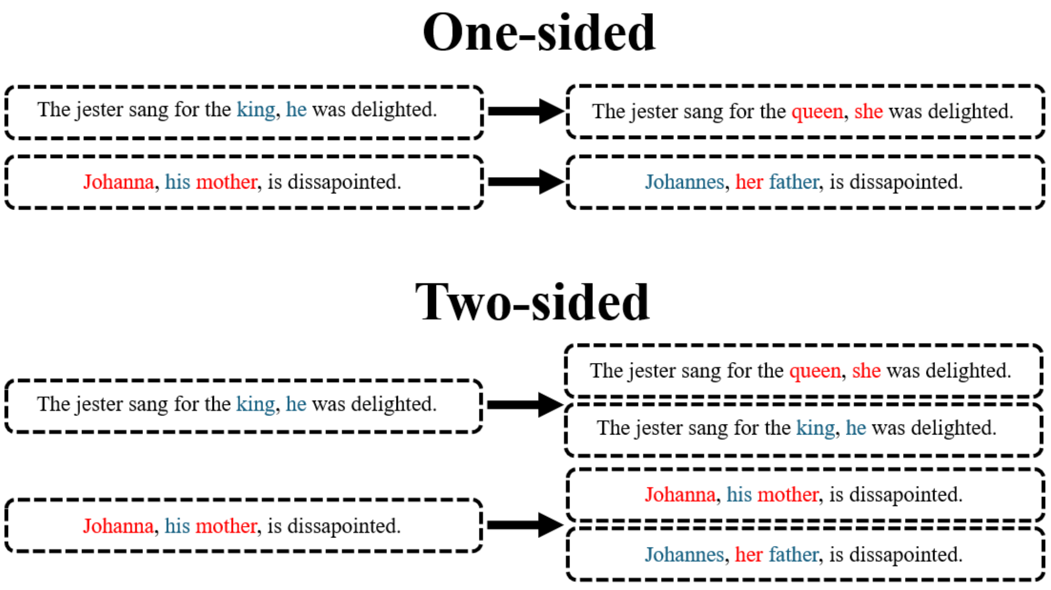

One-sided
The jester sang for the king, he was delighted.
The jester sang for the queen, she was delighted.
Johanna, his mother, is dissapointed.
Johannes, her father, is dissapointed.
Two-sided
The jester sang for the king, he was delighted.
The jester sang for the queen, she was delighted.
The jester sang for the king, he was delighted.
Johanna, his mother, is dissapointed.
Johanna, his mother, is dissapointed.
Johannes, her father, is dissapointed.

Figure 2: One-sided and two-sided CDA. The one-sided version deletes the original sentence while the two-sided procedure preserves both the original and its augmented version.

The CDA implementation can be found in the `CDA` function with parameters:

- **batch**, `dict`: The dataset to be augmented.
- **pairs**, `dict[str, str]`: Dictionary storing the counterfactual pairs.
- **columns**, `list[str], default = None`: list of columns of the dataset on which we want to perform the augmentation. If `columns = None` it performs CDA on all columns.
- **bidirectional**, `bool, default = True`: Boolean parameter that controls whether to perform two-sided CDA (if `True`) or one-sided CDA (if `False`).

A simple usage example shows how to augment a dataset:

```
>>> from FairLangProc.algorithms.preprocessors import CDA
>>> gendered_pairs = [
...     ('he', 'she'),
...     ('him', 'her'),
...     ('his', 'hers'),
...     ('actor', 'actress'),
...     ('priest', 'nun'),
...     ('father', 'mother'),
...     ('dad', 'mom'),
...     ('daddy', 'mommy'),
...     ('waiter', 'waitress'),
...     ('James', 'Jane')
... ]
>>> cda_train = Dataset.from_dict(
...         CDA(imdb['train'][:], pairs = dict(gendered_pairs))
... )
>>> train_CDA = cda_train.map(tokenize_function, batched=True)
>>> train_CDA.set_format(
...     type="torch", columns=["input_ids", "attention_mask", "label"]
... )
# Check differences
>>> print(f'Lenght of original train data set: {len(train_dataset['text'])}')
>>> print(f'Lenght of CDA augmented train data set: {len(cda_train['text'])}')
```

The new datasets can now be used to train the model on a less biased source:

```
>>> CDAModel = get_bert()
>>> trainer = Trainer(
...     model=CDAModel,
...     args=training_args,
...     train_dataset=train_CDA,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(CDAModel.parameters(), lr=2e-5, weight_decay=0.01),
...         None
...         )
... )
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

Projection-based debiasing

The rationale behind projection-based debiasing methods is similar to that of embedding metrics. They operate in latent space, aiming to identify a bias subspace given by an orthogonal basis, $\{v_i\}_{i=1}^{n_{bias}}$. Then, the hidden representation of any input can be debiased by removing its projection onto this space, formally

$$h_{proj} = h - \sum_{i=1}^{n_{bias}} \langle h, v_i \rangle \, v_i.$$

This process is illustrated in Figure 3. This can be done either at the word (Bolukbasi

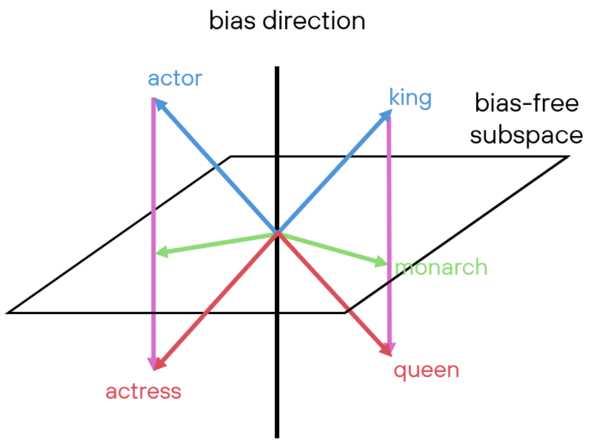


Figure 3: Projection of gendered words onto the debiased subspace.

et al. 2016) or sentence (Liang *et al.* 2020) level. In either case the bias subspace is generally identified through PCA, and usually its dimension is one, resulting in the construction of a bias direction.

The implementation of projection-based debiasing is done through the `SentDebias` abstract class which requires the construction of the `_get_embedding` method to compute the hidden representation of the input. The resulting model then requires the following parameters for its initialization:

- **model**, `torch.nn.Module`: Language Model.
- **tokenizer**, `TokenizerType`: Tokenizer for the previous model.
- **word_pairs**, `list[tuple[str]]`: List of counterfactual pairs.
- **n_components**, `int`: Number of components of the bias subspace.
- **config**, `Optional, dict`: Configuration of the Language Model.

Aditionally, when using the `SentDebiasForSequenceClassification` abstract class requires one last parameter:

- **n_labels**, `int`: Number of labels for the classification task.

An usage example can be seen below:

```
>>> from FairLangProc.algorithms.preprocessors\
... import SentDebiasForSequenceClassification
>>> gendered_pairs = [('he', 'she'), ('his', 'hers'), ('monk', 'nun')]
>>> model = get_bert()
>>> class SentDebiasBert(SentDebiasForSequenceClassification):
...     def _get_embedding(
...             self,
...             input_ids,
...             attention_mask = None,
...             token_type_ids = None
...             ):
...         return self.model.bert(
...             input_ids,
...             attention_mask = attention_mask,
...             token_type_ids = token_type_ids
...             ).last_hidden_state[:,0,:]
>>> EmbedModel = SentDebiasBert(
...     model = model,
...     config = None,
...     tokenizer = TOKENIZER,
...     word_pairs = gendered_pairs,
...     n_components = 1,
...     n_labels = 2
... )
>>> trainer = Trainer(
...     model=EmbedModel,
...     args=training_args,
...     train_dataset=train_dataset,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(EmbedModel.parameters(), lr=1e-5, weight_decay=0.1),
...         None
...         )
... )
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

BLIND debiasing

Orgad and Belinkov (2023) propose *Bias removaL wIth No Demographics* (BLIND), a debiasing procedure based on a complementary classifier $g_B : \mathbb{R}^{d_L} \longrightarrow \mathbb{R}$ with parameters θ_B, that takes the hidden representation vector as inputs and outputs the success probability of the model head for the downstream task. This probability is then used as a weight for said observation whose magnitude is controlled through a hyper-parameter $\gamma \geq 0$,

$$\mathcal{L}_{BLIND} = (1 - \sigma(g_B(h;\theta_B)))^{\gamma} \mathcal{L}^{task}(\hat{y}, y).$$

The term $\sigma(g_B(h;\theta_B))$ represents the model success probability for the downstream task: the bigger it is the less weight the observation has, while the smaller it is the more weight it carries. This forces the model to pay special attention to observations with low probability of success during training. Note that when $\gamma = 0$ the original loss function is restored, while $\gamma >> 1$ exacerbates the effect of the reweighting.

The implementation of BLIND debiasing is done through the abstract class `BLINDTrainer` which requires the implementation of the `_get_embedding` method to compute the hidden representations of the input tokens. This class extends the `Trainer` class from Hugging Face's **transformers**, providing a similar implementation. The parameters required for its initialization are:

- **blind_model**, `torch.nn.Module`: BLIND classifier, i.e., g_B using the previous notation.
- **blind_optimizer**, `torch.Optimizer`: Optimizer used to train the BLIND model.
- **temperature**, `float`: hyper-parameter, temperature of the softmax function of the BLIND classifier.
- **alpha**, `float`: hyper-parameter, strength of the BLIND loss.
- **gamma**, `float`: hyper-parameter, exponent of the BLIND loss.
- **model**, `torch.nn.Module`: Language Model.
- `Trainer` class kwargs.

A simple example is shown below:

```
>>> from FairLangProc.algorithms.preprocessors import BLINDTrainer
>>> BLINDModel = get_bert()
>>> BLINDClassifier = nn.Sequential(
...       nn.Linear(HIDDEN_DIM_BERT, HIDDEN_DIM_BERT),
...       nn.ReLU(),
...       nn.Linear(HIDDEN_DIM_BERT, 2)
... )
>>> class BLINDBERTTrainer(BLINDTrainer):
...     def _get_embedding(self, inputs):
...         return self.model.bert(
...             input_ids = inputs.get("input_ids"),
...             attention_mask = inputs.get("attention_mask"),
...             token_type_ids = inputs.get("token_type_ids")
...             ).last_hidden_state[:,0,:]
>>> trainer = BLINDBERTTrainer(
...     blind_model = BLINDClassifier,
...     blind_optimizer = lambda x: AdamW(x, lr=1e-5, weight_decay=0.1),
...     temperature = 1.0,
...     gamma = 2.0,
...     alpha = 1.0,
...     model = BLINDModel,
...     args = training_args,
...     train_dataset = train_dataset,
...     eval_dataset = val_dataset,
...     optimizers=(
...        AdamW(BLINDModel.parameters(), lr=1e-5, weight_decay=0.1),
...         None
...         )
... )
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

5.2. In-processing

In-processors modify the training process in order to reduce bias. Popular in-processor methods include modifying the loss function, changing the training scheduler for the model parameters or introducing new layers in the architecture.

Adapter Based Debiasing

Lauscher *et al.* (2021) propose the *Adapter-based DEbiasing of LanguagE models* (ADELE) procedure based on the adapter framework. They adopt the architecture shown in Strubell, Ganesh, and McCallum (2019) in which a single adapter module is included to each transformer layer after the feed-forward sub-layer, where the outputs are compressed to a bottleneck dimension m and then decompressed back to the hidden size of the transformer, d_L. The adapter module itself consists of a two-layer feed-forward network,

$$\text{Adapter}(\mathbf{h}, \mathbf{r}) = U \cdot g(D \cdot \mathbf{h}) + \mathbf{r},$$

where $\mathbf{h}$ and $\mathbf{r}$ are the hidden state and residual of the corresponding transformer layer, g is an activation function and $D \in \mathbb{R}^{m \times d_L}, U \in \mathbb{R}^{d_L \times m}$ represent the projection matrices. The idea behind the adapter layer is to introduce an information bottleneck which compresses the latent representation of the inputs, forcing the model to discard all irrelevant information. This mechanism is represented in Figure 4. Compounding this with techniques such as CDA, we force the model to discard the sensitive attribute as every social group appears in the same proportion, hence providing no additional information.

FairLangProc implements ADELE through the `DebiasAdapter` class which uses the **adapter-transformers** library to apply an adapter to a given model. Then, training can continue as normal with the `AdapterTrainer`. The parameters of `DebiasAdapter` are:

- **model**, `torch.nn.Module`: A Language Model.
- **adapter_config**, `Union[str, dict]`: Configuration of the adapter model. ADELE uses the `seq_bn` bottleneck configuration.

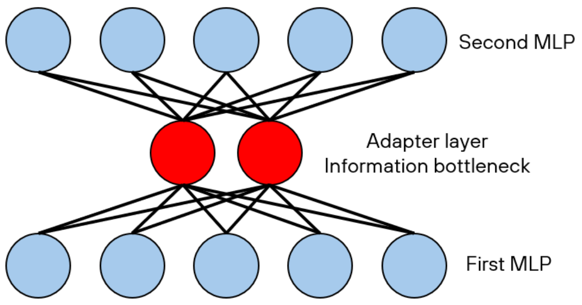


Figure 4: The adapter layer compresses the hidden representations.

A simple usage example is found below:

```
>>> from adapters import AdapterTrainer
>>> from FairLangProc.algorithms.inprocessors import DebiasAdapter
>>> DebiasAdapter = DebiasAdapter(
...     model = get_bert(),
...     adapter_config = "seq_bn"
... )
>>> AdeleModel = DebiasAdapter.get_model()
>>> trainer = AdapterTrainer(
...     model=AdeleModel,
...     args=training_args,
...     train_dataset=train_CDA,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(AdeleModel.parameters(),lr=1e-5, weight_decay=0.1),
...         None
...         )
... )
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

Regularizers

The use of regularizers is well known in the Machine Learning literature (Cortes, Mohri, and Rostamizadeh 2009; Kukačka, Golkov, and Cremers 2017; Tian and Zhang 2022) as a way of taking into account alternative objectives to the original task by modifying the loss function. Formally,

$$\mathcal{L}^{reg} = \mathcal{L}^{task} + \lambda\mathcal{R}.$$

FairLangProc implements two regularizers based on different concepts.

The first one by Liu *et al.* (2020) is based on the distance between the embeddings of counterfactual pairs given by A,

$$\mathcal{R} = \sum_{(a_i,a_j)\in A} ||M(a_i) - M(a_j)||_2.$$

The idea is similar to that of embedding based metrics and pre-processing. Its implementation relies on the `EmbeddingBasedRegularizer` abstract class which requires the implementation of the `_get_embedding` abstract method. Once this aspect is taken care of, the parameters required for its initialization are as follows:

- **model**, `torch.nn.Module`: A Language Model.
- **tokenizer**, `TokenizerType`: Tokenizer of the Language Model.
- **word_pairs**, `list[tuple[str]]`: List of counterfactual pairs.
- **ear_reg_strength**, `float`: Regularization strength.

On the other hand, Attanasio *et al.* (2022) propose Entropy Attention Regularization (EAR), which tries to maximize the entropy of the attention weights to encourage attention to the broader context of the input,

$$\mathcal{R} = -\sum_{l=1}^{L} \text{entropy}_l(\mathbf{A}),$$

where $\text{entropy}_l(\cdot)$ denotes the entropy of the l-th layer.
These methods are implemented through the `EARModel` class with initialization parameters:

- **model**, `torch.nn.Module`: A Language Model.
- **ear_reg_strength**, `float`: Regularization strength.

We show usage example below:

```
>>> from FairLangProc.algorithms.inprocessors import EARModel
>>> model = get_bert()
>>> EARRegularizer = EARModel(
...      model = model,
...      ear_reg_strength = 0.01
... )
>>> trainer = Trainer(
...     model=EARRegularizer,
...     args=training_args,
...     train_dataset=train_dataset,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(EARRegularizer.parameters(), lr=1e-5, weight_decay=0.1),
...         None
...         )
... )
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

Selective parameter updating

Fine-tuning on a curated dataset can reduce bias in the model outputs. However, this can lead to catastrophic forgetting (Kirkpatrick *et al.* 2017). One way of circumventing this issue is to freeze a big amount of the model parameters, which also helps lessening computational expenses. Gira, Zhang, and Lee (2022) freeze over 99% of the model's parameters, only updating layer norm parameters or word position embeddings, while Ranaldi *et al.* (2024) only update the attention matrices.

In **FairLangProc** this is performed through the `selective_unfreezing` function, which freezes all the model layers with the exception of those specified by the user. The parameters of this method are:

- **model**, `torch.nn.Module`: A Language Model.
- **substrings**, `list[str]`: List of layers that must remain unfrozen.

In the example below all model parameters are frozen with the exception of the attention layers:

```
>>> from FairLangProc.algorithms.inprocessors import selective_unfreezing
>>> FrozenBert = get_bert()
>>> selective_unfreezing(FrozenBert, ["attention.self", "attention.output"])
>>> trainer = Trainer(
...     model=FrozenBert,
...     args=training_args,
...     train_dataset=train_CDA,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(FrozenBert.parameters(), lr=1e-5, weight_decay=0.1),
...         None
...         )
... )
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

5.3. Intra-processing

Adopting the definition by Savani, White, and Govindarajulu (2020), intra-processors are those fairness methods that modify a model's behavior without further training or fine-tuning.

These procedures might involve the training of a complementary model but the original parameters remain unchanged.

Modular debias with Diff prunning

Hauzenberger *et al.* (2023) explore Modular Debiasing with Diff Subnetworks (MoDDiffy) to create many sparse subnetworks to address bias for different attributes (gender, religion,...) through the idea of *Diff* prunning (Guo, Rush, and Kim 2021). Basically, they freeze the model parameters, θ, and train another network with parameters δ, with a loss function that promotes accuracy, sparsity and debiasing. The final debiased model results from considering the parameters $\theta + \delta$. One can train many such networks for each prejudice dimension, resulting in a set of subnetworks $\delta_1, ..., \delta_p$ such that one can choose the subnetworks needed for the bias attributes of interest.

The way of training this network is the following: Suppose a LM, $M(\cdot\,;\theta)$, with a head intended for a specific downstream task, $g(\cdot\,;\theta)$, such that the outputs of the encoder for a given input S, $h = E(S;\theta)$, are then fed to the head to generate logits for the task at hand, $\hat{y} = g(h;\theta)$. Then, for a given bias attribute ρ the model parameters θ are frozen and a new set of parameters, δ_ρ, are trained instead. To do this, we consider the outputs resulting from considering the addition of the new parameters to the old model:

$$h_\rho = M(x;\theta + \delta_\rho), \qquad \hat{y}_\rho = g(h_\rho;\theta + \delta_\rho).$$

These new outputs are then trained with the loss function

$$\mathcal{L}_\rho = \mathcal{L}_\rho^{task} + \lambda_\rho^0 \mathcal{L}_\rho^0 + \lambda_\rho^{debias} \mathcal{L}_\rho^{debias},$$

where $\mathcal{L}_\rho^{task}$ represents the original loss of the downstream task with the new parameters, $\mathcal{L}_\rho^0$ is a term that promotes sparsity and $\mathcal{L}_\rho^{debias}$ represents a regularizer to debias the outputs with respect to ρ.

In more concrete terms the loss of the downstream task is computed as

$$\mathcal{L}_\rho^{task} = \mathcal{L}^{task}(\hat{y}_\rho, y).$$

The sparsity loss is given by a smooth approximation to the L_0 norm of δ_p, that is, $\sum_{i=0}^{|\delta_\rho|} \mathbf{1}(\delta_{\rho,i} \neq 0)$. This approximation is performed by a decomposition of the parameters, δ_ρ, into the element-wise multiplication of two new set of parameters, m_ρ and w_ρ, resulting in $\delta_\rho = m_\rho \odot w_\rho$. In this situation, w_ρ represents the magnitude of the parameters while m_ρ is a mask that takes the binary values 0 or 1 to promote sparsity. This mask is characterized by the Hard-Concrete distribution (Jang, Gu, and Poole 2016) with parameters $(\log \alpha_\rho, 1)$ and hyper-parameters $\gamma < 0, \zeta > 1$:

$$\mathcal{L}_\rho^0 = \sum_{i=1}^{|\delta_\rho|} \sigma \left(\log \alpha_{\rho,i} - \log \left(-\frac{\gamma}{\zeta} \right) \right).$$

Finally, let us define the debiasing loss. In general, any of the debiasing regularizers described in the literature suffice. In particular, Hauzenberger *et al.* (2023) suggest two different debiasing losses: one based on adversarial debiasing and another one based on mutual information.

We have opted for an implementation of the latter, which is similar to embedding-based regularization. This loss is expressed as the squared difference of the average of the embeddings of two different groups of inputs, X_ρ^A and X_ρ^B,

$$\mathcal{L}_\rho^{debias} = \left(\frac{\sum_{x_A \in X_\rho^A} \phi(M(x_A))}{|X_\rho^A|} - \frac{\sum_{x_B \in X_\rho^B} \phi(M(x_B))}{|X_\rho^B|} \right)^2 ,$$

where ϕ is a transformation kernel.

The final loss function for the bias attribute ρ is

$$\mathcal{L}_\rho = \mathcal{L}^{task}(g(z_\rho; \theta + \delta_\rho), y) + \lambda_\rho^0 \sum_{i=1}^{|\delta_\rho|} \sigma \left(\log \alpha_{\rho,i} - \log \left(-\frac{\gamma}{\zeta} \right) \right)$$
$$+ \lambda_\rho^{debias} \left(\frac{\sum_{x_A \in X_\rho^A} \phi(M(x_A; \theta + \delta_\rho))}{|X_\rho^A|} - \frac{\sum_{x_B \in X_\rho^B} \phi(M(x_B; \theta + \delta_\rho))}{|X_\rho^B|} \right)^2 .$$

Finally, given a set of b bias attributes $\{\rho_i\}_{i=1}^b$ we can train b different subnetworks $\{\delta_i\}_{i=1}^b$ and choose each time we use the model which bias attributes $i_1, ..., i_n$ to handle by simply adding the corresponding bias subnetworks: $\theta + \delta_{i_1} + ... + \delta_{i_n}$.

The differential prunning framework is carried through the abstract class `DiffPrunDebiasing` which requires the implementation of a `_forward` method representing the computation of the embedding of the inputs, similar to the `_get_embedding` method of previous implementations. This class heavily relies on the existing implementation by Hauzenberger *et al.* (2023)[3]. The initialization parameters are listed below:

- **head**, `torch.nn.Module`: Head attached to the Language Model.
- **model**, `torch.nn.Module`: Encoder of the Language Model.
- **input_ids_A**, `torch.Tensor`: Input ids of the A group; that is, X_ρ^A using the above notation.
- **input_ids_B**, `torch.Tensor`: Input ids of the A group; that is, X_ρ^B using the above notation.
- **lambda_sparse**, `float`: Regularization strength of the sparse loss; that is, λ_ρ^0 using the above notation.
- **lambda_bias**, `float`: Regularization strength of the debias loss; that is, λ_ρ^{debias} using the above notation.
- **bias_kernel**, `Callabe`: Kernel used in the computation of the debiasing loss; that is, ϕ using the above notation.
- **fixmask_init**, `bool`: If `True`, applies a mask m to promote sparsity.

[3]https://github.com/CPJKU/ModularizedDebiasing

- **alpha_init**, `float`: Hyper-parameter representing the initialization value of the α in the hard-concrete distribution.
- **structured_diff_prunning**, `bool`: If `True` adds a common structure, α_{group}, to the α parameters of the hard-concrete distribution. Concretely, each parameter α_i is substituted by $\tilde{\alpha}_i = \alpha_{group} + \alpha_i$.
- **upper**, `float`: Hyper-parameter representing the upper bound of the hard-concrete distribution; that is, γ using the above notation.
- **lower**, `float`: Hyper-parameter representing the lower bound of the hard-concrete distribution; that is, ζ using the above notation.

Furthermore, in the example below we have defined a *bias kernel* to compute the debiasing loss, although if this field is left blank it defaults to using the identity:

```
>>> from FairLangProc.algorithms.intraprocessors import DiffPrunBERT
>>> gendered_pairs = [
...     ("manager", "manageress"),
...     ("nephew", "niece"),
...     ("prince", "princess"),
...     ("baron", "baroness"),
...     ("father", "mother"),
...     ("stepsons", "stepdaughters"),
...     ("boyfriend", "girlfriend"),
...     ("fiances", "fiancees"),
...     ("shepherd", "shepherdess"),
...     ("beau", "belle"),
...     ("males", "females"),
...     ("hunter", "huntress"),
...     ("grandfathers", "grandmothers"),
...     ("daddies", "mummies"),
...     ("step-son", "step-daughter"),
...     ("masters", "mistresses"),
...     ("nephews", "nieces"),
...     ("brother", "sister"),
...     ("grandfather", "grandmother"),
...     ("priest", "priestess")
... ]
>>> tokens_male = [words[0] for words in gendered_pairs]
>>> tokens_female = [words[1] for words in gendered_pairs]
>>> inputs_male = TOKENIZER(
...     tokens_male, padding = True, return_tensors = "pt"
... )
>>> inputs_female = TOKENIZER(
...     tokens_female, padding = True, return_tensors = "pt"
... )
```

```
>>> def normalize_by_column(x: torch.Tensor, eps: float = 1e-8):
...     mean = x.mean(dim=0, keepdim=True)
...     std = x.std(dim=0, keepdim=True)
...     return (x - mean) / (std + eps)
>>> original_model = get_bert()
>>> ModularDebiasingBERT = DiffPrunBERT(
...     head = original_model.classifier,
...     encoder = original_model.bert,
...     loss_fn = torch.nn.CrossEntropyLoss(),
...     input_ids_A = inputs_male,
...     input_ids_B = inputs_female,
...     bias_kernel = normalize_by_column,
...     upper = 10,
...     lower = -0.001,
...     lambda_bias = 0.5,
...     lambda_sparse = 0.00001
... )
>>> trainer = Trainer(
...     model=ModularDebiasingBERT,
...     args=training_args,
...     train_dataset=train_dataset,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(ModularDebiasingBERT.parameters(), lr=1e-5, weight_decay=0.1),
...         None
...         )
... )
>>> trainer.train()
>>> results = trainer.evaluate()
>>> print(results)
```

Entropy-based Attention Temperature scaling

Zayed *et al.* (2023) propose the use of Entropy-based Attention Temperature (EAT) scaling in order to modify the distribution of the attention scores with a temperature-related parameter, $\beta \in [0, \infty)$:

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

The idea is that when $\beta >> 1$, the head attends only to the tokens with biggest scores, while $\beta \approx 0$ forces the head to attend equally to all tokens. When $\beta = 1$, the attention head remains unmodified.

The scaling procedure is implemented through the `add_EAT_hook`, which attaches a hook to the attention heads of the model. Its parameters are as follows.

- **model**, `torch.nn.Module`: A Language Model.

- **beta**, `float`: Temperature parameter.

An example is shown below. Note that, in contrast with previous methods, EAT does not require training:

```
>>> from FairLangProc.algorithms.intraprocessors import add_EAT_hook
>>> EATBert = BERT
>>> beta = 1.5
>>> add_EAT_hook(model=EATBert, beta=beta)
>>> trainer = Trainer(
...     model=EATBert,
...     args=training_args,
...     train_dataset=train_dataset,
...     eval_dataset=val_dataset,
...     optimizers=(
...         AdamW(EATBert.parameters(), lr=1e-5, weight_decay=0.1),
...         None
...         )
... )
>>> results = trainer.evaluate()
>>> print(results)
```

6. Case study: debiasing BERT

This section showcases the results of using **FairLangProc** to debias a LM in a setting closer to reality. The idea is to apply all debiasing methods on the BERT model (Devlin *et al.* 2019) during fine-tuning on a given dataset, and measure both their performance and bias to check the available trade-offs and get a rough idea of the level of prejudice mitigation they provide.

The methods were tested on the General Language Understanding Evaluation (GLUE) dataset (Wang *et al.* 2018), which consists of a set of 9 tasks (8 classification tasks, 1 regression), aimed at testing the language understanding of a LM. A brief summary of the different tasks is provided below. Unless stated otherwise, the task consists of classification and the metric used is accuracy score.

- **Corpus of Linguistic Acceptability** (CoLA) (Warstadt *et al.* 2019), consists of sequence of words annotated with whether it is a grammaticaly correct English sentence. The metric used is the Matthews correlation coefficient.
- **Stanford Sentiment Treebank** (SST-2) (Socher *et al.* 2013), is comprised of sentences from movie reviews and human annotations of their sentiment.
- **Microsoft Research Paraphrase Corpus** (MRPC) (Dolan and Brockett 2005), is a collection of sentence pairs with labels indicating whether the sentences in the pair are equivalent. The metric used is F1 score.
- **Quora Question Pairs** (QQP), compiles pairs of questions with the task of identifying whether or not both questions are semantically equivalent. The metric used is F1 score.

- **Sematic Textual Similarity Benchmark** (STS-B) (Cer *et al.* 2017), consists of sentence pairs drawn from news headlines, video and image captions, and natural language inference data; each pair is annotated with a similarity score from 1 to 5 and the task is to predict these scores. This is the only regression task and the metric used is the Spearman correlation.
- **Multi-Genre Natural Language Inference Corpus** (MNLI) (Williams *et al.* 2018), is comprised of sentence pairs with the task of textual entailment. That is, each sentence pair consists of a premise and a hypothesis and the goal is to indicate whether the premise entails the hypothesis, contradicts it, or neither.
- **Standford Question Answering Dataset** (QNLI) (Rajpurkar *et al.* 2016), provides sentence pairs consisting of a context and a question, with the task of determining whether the context sentence contains the answer to the question.
- **Recognizing Textual Entailment** (RTE), is a compilation of textual entailment sentence pairs.
- **Winograd Schema Challenge** (WNLI) (Levesque *et al.* 2012), is a collection of sentence pairs which differ only on an ambiguous pronoun, with the objective of determining whether the sentence with the pronoun substituted is entailed by the original sentence.

To fine-tune BERT on each task a head is attached after the last layer of the model as described in Section 2.

Bias was evaluated using WEAT 7 by Caliskan *et al.* (2017), which measures the association of gender with mathematics and arts, meaning that a positive score is related with the stereotypical association (i.e. men with maths, women with arts) while a negative score shows an anti-stereotypical association (i.e. women with maths, men with arts). Concretely, we used the following set of words:

- Math: math, algebra, geometry, calculus, equations, computation, numbers, addition.
- Arts: poetry, art, dance, literature, novel, symphony, drama, sculpture.
- Male terms: male, man, boy, brother, he, him, his, son.
- Female terms: female, woman, girl, sister, she, her, hers, daughter.

The models were trained for 3 epochs with a batch size of 16. The Adam optimizer (Kingma and Ba 2014) was used with hyper-parameters $\beta_1 = 0.9, \beta_2 = 0.999$ and weight decay 0.1, and the learning rate was set to $2e - 5$. All experiments were performed on an NVIDIA Tesla T4 GPU with 16GB of VRAM. The system was running NVIDIA driver version 565.57.01 and CUDA 12.7.

Experiments were implemented using Python and executed via virtual environments under PyTorch and associated machine learning libraries. The details of the implementation can be found in the `DemoDebiasing.ipynb` notebook, inside the `notebooks` folder.

The results of the experiment are shown in Tables 2 and 3. It seems like performance is consistent across all tasks, with most methods achieving similar results for GLUE tasks,

Debias	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	Average
none	55.7	92.4	88.2	87.9	87.2	84.1/84.7	91.4	63.9	38.0	77.4
CDA	55.7	92.5	88.4	88.0	87.5	84.3/84.3	91.3	66.1	36.6	77.5
BLIND	55.2	91.3	86.0	-	81.8	81.2/81.6	89.7	63.9	56.3	76.3
emb	56.3	92.7	87.5	55.4	87.6	84.4/84.1	91.3	63.2	43.7	74.6
EAR	56.0	92.5	88.0	88.4	87.0	84.4/84.4	91.5	62.5	53.5	78.8
ADELE	56.2	92.5	87.6	87.8	85.7	83.9/84.5	91.3	63.5	33.8	76.7
sel	48.8	91.7	86.7	81.6	84.2	83.0/83.0	90.0	61.7	56.3	76.7
EAT	45.1	92.3	85.4	87.3	85.2	83.1/83.3	89.4	62.8	33.8	74.8
diff	58.0	92.9	89.7	60.4	87.7	84.3/84.8	91.5	65.0	50.7	76.5

Table 2: Performance of the different model on GLUE tasks for the validation set. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks.

Debias	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	Average
none	0.005	0.149	0.045	0.164	0.704	0.098	0.379	0.470	0.181	0.244
CDA	-0.088	0.309	0.115	0.337	-0.018	0.024	0.361	0.057	0.050	0.127
BLIND	-0.054	0.238	0.007	-	0.134	-0.017	0.209	0.071	-0.006	0.073
emb	-0.207	0.219	0.114	0.037	-0.085	-0.095	0.296	0.657	0.118	0.117
EAR	0.180	0.349	0.142	0.296	-0.098	-0.762	0.124	0.241	-0.033	0.049
ADELE	-0.058	0.222	0.046	0.153	-0.399	0.013	0.378	0.553	0.181	0.121
sel	-0.092	0.056	0.108	0.122	0.010	-0.096	0.098	0.167	0.026	0.044
EAT	0.230	0.253	0.138	0.311	-0.232	-0.059	0.412	-0.048	-0.153	0.095
diff	-0.004	0.489	0.034	-0.221	-0.175	-0.135	0.426	0.377	0.033	0.092

Table 3: WEAT 7 test for the debiasing methods.

although there are some exceptions on both directions, the most striking underperformers being found in EAT in CoLA and embedding methods in STS-B, while BLIND, selective unfreezing and EAR achieve better results than the benchmark in WNLI while reducing representation bias. Looking at the aggregation score of all tasks it is clear that the use of a debiasing method compromises accuracy, which is a phenomenon well recorded in the literature known as the *accuracy-fairness trade-off*. The two exceptions to this rule are found in CDA and EAR regularization which actually improve the average results with respect to BERT.

Taking a look at the association scores, it is clear that although some processors such as EAR in CoLA or BLIND in SST-2 exacerbate prejudice, debiasing methods reduce the association score in general. This can be verified by taking a look at the average WEAT score which is lower than that of BERT without debiasing for all methods. Moreover, some algorithms flip the sign of the WEAT score, indicating that the stereotypical association is reversed. This is the case for CDA in QQP or BLIND in MNLI. Furthermore, the increased WEAT score find easy explanations. For example, methods like CDA can can fail due to a lack of enough gendered sentences in the training dataset, while the shortcomings of embedding-based debiasing are already known in the literature (Gallegos *et al.* 2024). In general, pre-processors and in-processors seem to achieve the best results both in terms of fairness and performance, with EAR providing a very appealing alternative to standard NLP for those concerned with gendered bias in text processing.

In any case, this experiment is simply a proof of concept and can be easily expanded upon.

In order to derive a more sound conclusion more tests are necessary with a more thorough and exhaustive hyper-parameter search, and considering different random states with an aggregation strategy in order to remove randomness. In any case, it is clear that the use of **FairLangProc** allows for easy application and comparison between fairness methods.

7. Conclusion

The **FairLangProc** package has been introduced as a comprehensive, user-friendly framework compatible with the popular Hugging Face libraries. The main contributions of the package are threefold. First, the `datasets` submodule implements the `BiasDataLoader` method, which allows for easy handling of the different datasets proposed in the literature for bias evaluation and mitigation. The different parameters allow the user to navigate the available data and configurations in the format they prefer. Second, the `metrics` submodule provides a myriad of metrics based on embeddings, probabilities and generated text. This encourages a multidimensional approach to fairness which should not be restricted by the limitations of one single metric, allowing decision-makers to check the different biases a model might incur in. Finally, the `algorithms` submodule compiles a plethora of pre-processors, in-processors and post-processors which require minimal input from the user to function, allowing their use in a production environment. The different usage examples illustrates the ease of use of the package, and the final case study showcases how it can be used in a real-world setting, providing satisfactory bias-mitigation and performance on a well-known NLP dataset.

The **FairLangProc** package encourages the democratization of fairness tools for Language Models and the inclusion of new methods resulting from collaborative efforts from the community. Furthermore, the ease of access of the algorithms and metrics facilitate their inclusion in bigger, more complex pipelines and projects, which is necessary if fairness methods are to become widespread among NLP practisioners.

Future development of **FairLangProc** will focus on expanding its capabilities, providing new methods as they are exposed to the NLP community, and potentially including the implementation of debiasing methods outside the Hugging Face environment, providing effective means of prejudice removal on LLMs outside its scope. We believe that **FairLangProc** is a valuable addition to the fairness landscape by facilitating the broader application of prejudice removal methods in both academia and industry.

8. Acknowledgements

We thank Víctor Agulló for his input on many different questions that arose during the making of the package and for his contributions to the `BiasDataLoader` method.

This work was partially supported by the Spanish Ministry of Science and Innovation (MCIN/AEI/10.13039/501100011033) under grants PID2022-137243OB-I00 and PID2022-137818OB-I00. This work also received support from the European Union's Recovery, Transformation and Resilience Plan – NextGenerationEU, through the INCIBE ANTICIPA grant and the ENIA 2022 programme for university–industry AI chairs (AImpulsa: UC3M-Universia).

References

Ahn J, Oh A (2021). "Mitigating Language-Dependent Ethnic Bias in BERT." In MF Moens, X Huang, L Specia, SWt Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 533–549. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic. doi:10.18653/v1/2021.emnlp-main.42. URL https://aclanthology.org/2021.emnlp-main.42/.

Attanasio G, *et al.* (2022). "Entropy-based Attention Regularization Frees Unintended Bias Mitigation from Lists." In S Muresan, P Nakov, A Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 1105–1119. Association for Computational Linguistics, Dublin, Ireland. doi:10.18653/v1/2022.findings-acl.88. URL https://aclanthology.org/2022.findings-acl.88/.

Barocas S, *et al.* (2023). *Fairness and Machine Learning: Limitations and Opportunities*. MIT Press.

Bartl M, *et al.* (2020). "Unmasking Contextual Stereotypes: Measuring and Mitigating BERT's Gender Bias." In MR Costa-jussà, C Hardmeier, W Radford, K Webster (eds.), *Proceedings of the Second Workshop on Gender Bias in Natural Language Processing*, pp. 1–16. Association for Computational Linguistics, Barcelona, Spain (Online). URL https://aclanthology.org/2020.gebnlp-1.1/.

BigScience, *et al.* (2022). "Bloom: A 176b-parameter open-access multilingual language model." *arXiv preprint arXiv:2211.05100*.

Birzhandi P, Cho YS (2023). "Application of fairness to healthcare, organizational justice, and finance: A survey." *Expert Systems with Applications*, **216**, 119465.

Bolukbasi T, *et al.* (2016). "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." *Advances in neural information processing systems*, **29**.

Brown T, *et al.* (2020). "Language models are few-shot learners." *Advances in neural information processing systems*, **33**, 1877–1901.

Caliskan A, Bryson JJ, Narayanan A (2017). "Semantics derived automatically from language corpora contain human-like biases." *Science*, **356**(6334), 183–186.

Cer D, *et al.* (2017). "SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation." In S Bethard, M Carpuat, M Apidianaki, SM Mohammad, D Cer, D Jurgens (eds.), *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14. Association for Computational Linguistics, Vancouver, Canada. doi:10.18653/v1/S17-2001. URL https://aclanthology.org/S17-2001/.

Chung HW, *et al.* (2024). "Scaling instruction-finetuned language models." *Journal of Machine Learning Research*, **25**(70), 1–53.

Cortes C, Mohri M, Rostamizadeh A (2009). "L2 regularization for learning kernels." In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, p. 109–116. AUAI Press, Arlington, Virginia, USA. ISBN 9780974903958.

Devlin J, *et al.* (2019). "Bert: Pre-training of deep bidirectional transformers for language understanding." In *Proceedings of the 2019 conference of the North American chapter of* *the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186.

Dhamala J, *et al.* (2021). "Bold: Dataset and metrics for measuring biases in open-ended language generation." In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pp. 862–872.

Dolan WB, Brockett C (2005). "Automatically Constructing a Corpus of Sentential Paraphrases." In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*. URL `https://aclanthology.org/I05-5002/`.

Foundation CC (Accessed 27/05/2025). "Common Crawl Dataset." URL `https://commoncrawl.org`.

Gallegos IO, *et al.* (2024). "Bias and fairness in large language models: A survey." *Computational Linguistics*, **50**(3), 1097–1179.

Gira M, Zhang R, Lee K (2022). "Debiasing pre-trained language models via efficient fine-tuning." In *Proceedings of the second workshop on language technology for equality, diversity and inclusion*, pp. 59–69.

Goyal S, *et al.* (2024). "Healai: A healthcare llm for effective medical documentation." In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pp. 1167–1168.

Guo D, Rush A, Kim Y (2021). "Parameter-Efficient Transfer Learning with Diff Pruning." In C Zong, F Xia, W Li, R Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4884–4896. Association for Computational Linguistics, Online. `doi:10.18653/v1/2021.acl-long.378`. URL `https://aclanthology.org/2021.acl-long.378/`.

Guo W, Caliskan A (2021). "Detecting emergent intersectional biases: Contextualized word embeddings contain a distribution of human-like biases." In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 122–133.

Hauzenberger L, *et al.* (2023). "Modular and On-demand Bias Mitigation with Attribute-Removal Subnetworks." In A Rogers, J Boyd-Graber, N Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 6192–6214. Association for Computational Linguistics, Toronto, Canada. `doi:10.18653/v1/2023.findings-acl.386`. URL `https://aclanthology.org/2023.findings-acl.386/`.

Jang E, Gu S, Poole B (2016). "Categorical reparameterization with gumbel-softmax." *arXiv preprint arXiv:1611.01144*.

Jurafsky D, Martin JH (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models.* 3rd edition. Online manuscript released January 12, 2025. https://web.stanford.edu/ jurafsky/slp3.

Kaneko M, Bollegala D (2022). "Unmasking the mask–evaluating social biases in masked language models." In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 11954–11962.

Kingma DP, Ba J (2014). "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*.

Kirkpatrick J, *et al.* (2017). "Overcoming catastrophic forgetting in neural networks." *Proceedings of the national academy of sciences*, **114**(13), 3521–3526.

Kozodoi N, Jacob J, Lessmann S (2022). "Fairness in credit scoring: Assessment, implementation and profit implications." *European Journal of Operational Research*, **297**(3), 1083–1094.

Kukačka J, Golkov V, Cremers D (2017). "Regularization for deep learning: A taxonomy." *arXiv preprint arXiv:1710.10686*.

Kurita K, *et al.* (2019). "Measuring Bias in Contextualized Word Representations." In MR Costa-jussà, C Hardmeier, W Radford, K Webster (eds.), *Proceedings of the First Workshop on Gender Bias in Natural Language Processing*, pp. 166–172. Association for Computational Linguistics, Florence, Italy. doi:10.18653/v1/W19-3823. URL https://aclanthology.org/W19-3823/.

Lai J, *et al.* (2024). "Large language models in law: A survey." *AI Open*, **5**, 181–196. ISSN 2666-6510. doi:https://doi.org/10.1016/j.aiopen.2024.09.002. URL https://www.sciencedirect.com/science/article/pii/S2666651024000172.

Lauscher A, *et al.* (2021). "Sustainable Modular Debiasing of Language Models." In MF Moens, X Huang, L Specia, SWt Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 4782–4797. Association for Computational Linguistics, Punta Cana, Dominican Republic. doi:10.18653/v1/2021.findings-emnlp.411. URL https://aclanthology.org/2021.findings-emnlp.411/.

Levesque HJ, *et al.* (2012). "The Winograd schema challenge." *KR*, **2012**(13th), 3.

Levy S, *et al.* (2021). "Collecting a Large-Scale Gender Bias Dataset for Coreference Resolution and Machine Translation." In MF Moens, X Huang, L Specia, SWt Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 2470–2480. Association for Computational Linguistics, Punta Cana, Dominican Republic. doi:10.18653/v1/2021.findings-emnlp.211. URL https://aclanthology.org/2021.findings-emnlp.211/.

Li T, *et al.* (2020). "UNQOVERing Stereotyping Biases via Underspecified Questions." In T Cohn, Y He, Y Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 3475–3489. Association for Computational Linguistics, Online. doi:10.18653/v1/2020.findings-emnlp.311. URL https://aclanthology.org/2020.findings-emnlp.311/.

Li Y, *et al.* (2023). "Large language models in finance: A survey." In *Proceedings of the fourth ACM international conference on AI in finance*, pp. 374–382.

Liang P, *et al.* (2022). "Holistic evaluation of language models." *arXiv preprint arXiv:2211.09110*.

Liang PP, *et al.* (2020). "Towards Debiasing Sentence Representations." In D Jurafsky, J Chai, N Schluter, J Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5502–5515. Association for Computational Linguistics, Online. doi:10.18653/v1/2020.acl-main.488. URL https://aclanthology.org/2020.acl-main.488/.

Liu A, *et al.* (2024). "Deepseek-v3 technical report." *arXiv preprint arXiv:2412.19437*.

Liu H, *et al.* (2020). "Does Gender Matter? Towards Fairness in Dialogue Systems." In D Scott, N Bel, C Zong (eds.), *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 4403–4416. International Committee on Computational Linguistics, Barcelona, Spain (Online). doi:10.18653/v1/2020.coling-main.390. URL https://aclanthology.org/2020.coling-main.390/.

Lu K, *et al.* (2020). "Gender bias in neural natural language processing." *Logic, language, and security: essays dedicated to Andre Scedrov on the occasion of his 65th birthday*, pp. 189–202.

May C, *et al.* (2019). "On Measuring Social Biases in Sentence Encoders." In J Burstein, C Doran, T Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 622–628. Association for Computational Linguistics, Minneapolis, Minnesota. doi:10.18653/v1/N19-1063. URL https://aclanthology.org/N19-1063/.

Meyer JG, *et al.* (2023). "ChatGPT and large language models in academia: opportunities and challenges." *BioData mining*, **16**(1), 20.

Microsoft (Accessed 27/05/2025). "Announcing Microsoft Copilot, your everyday AI companion." URL https://blogs.microsoft.com/blog/2023/09/21/announcing-microsoft-copilot-your-everyday-ai-companion/.

Nadeem M, *et al.* (2021). "StereoSet: Measuring stereotypical bias in pretrained language models." In C Zong, F Xia, W Li, R Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5356–5371. Association for Computational Linguistics, Online. doi:10.18653/v1/2021.acl-long.416. URL https://aclanthology.org/2021.acl-long.416/.

Nangia N, *et al.* (2020). "CrowS-Pairs: A Challenge Dataset for Measuring Social Biases in Masked Language Models." In B Webber, T Cohn, Y He, Y Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1953–1967. Association for Computational Linguistics, Online. doi:10.18653/v1/2020.emnlp-main.154. URL https://aclanthology.org/2020.emnlp-main.154/.

Naveed H, *et al.* (2023). "A comprehensive overview of large language models." *ACM Transactions on Intelligent Systems and Technology*.

Nozza D, *et al.* (2021). "HONEST: Measuring hurtful sentence completion in language models." In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: Human language technologies*. Association for Computational Linguistics.

Orgad H, Belinkov Y (2023). "BLIND: Bias Removal With No Demographics." In A Rogers, J Boyd-Graber, N Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8801–8821. Association for Computational Linguistics, Toronto, Canada. doi:10.18653/v1/2023.acl-long.490. URL https://aclanthology.org/2023.acl-long.490/.

O'Connor S, Liu H (2024). "Gender bias perpetuation and mitigation in AI technologies: challenges and opportunities." *AI & SOCIETY*, **39**(4), 2045–2057.

Parrish A, *et al.* (2022). "BBQ: A hand-built bias benchmark for question answering." In S Muresan, P Nakov, A Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2086–2105. Association for Computational Linguistics, Dublin, Ireland. doi:10.18653/v1/2022.findings-acl.165. URL https://aclanthology.org/2022.findings-acl.165/.

Radford A, *et al.* (2019). "Language models are unsupervised multitask learners." *OpenAI blog*, **1**(8), 9.

Rajpurkar P, *et al.* (2016). "SQuAD: 100,000+ Questions for Machine Comprehension of Text." In J Su, K Duh, X Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, Austin, Texas. doi:10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Ranaldi L, *et al.* (2024). "A Trip Towards Fairness: Bias and De-Biasing in Large Language Models." In D Bollegala, V Shwartz (eds.), *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, pp. 372–384. Association for Computational Linguistics, Mexico City, Mexico. doi:10.18653/v1/2024.starsem-1.30. URL https://aclanthology.org/2024.starsem-1.30/.

Rudinger R, *et al.* (2018). "Gender Bias in Coreference Resolution." In M Walker, H Ji, A Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 8–14. Association for Computational Linguistics, New Orleans, Louisiana. doi:10.18653/v1/N18-2002. URL https://aclanthology.org/N18-2002/.

Salazar J, *et al.* (2020). "Masked Language Model Scoring." In D Jurafsky, J Chai, N Schluter, J Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2699–2712. Association for Computational Linguistics, Online. doi:10.18653/v1/2020.acl-main.240. URL https://aclanthology.org/2020.acl-main.240/.

Savani Y, White C, Govindarajulu NS (2020). "Post-Hoc Methods for Debiasing Neural Networks." *CoRR*, **abs/2006.08564**. 2006.08564, URL https://arxiv.org/abs/2006.08564.

Smith EM, *et al.* (2022). "“I'm sorry to hear that”: Finding New Biases in Language Models with a Holistic Descriptor Dataset." In Y Goldberg, Z Kozareva, Y Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 9180–9211. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates. doi:10.18653/v1/2022.emnlp-main.625. URL https://aclanthology.org/2022.emnlp-main.625/.

Socher R, *et al.* (2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank." In D Yarowsky, T Baldwin, A Korhonen, K Livescu, S Bethard (eds.), *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642. Association for Computational Linguistics, Seattle, Washington, USA. URL https://aclanthology.org/D13-1170/.

Strubell E, Ganesh A, McCallum A (2019). "Energy and Policy Considerations for Deep Learning in NLP." In A Korhonen, D Traum, L Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650. Association for Computational Linguistics, Florence, Italy. doi:10.18653/v1/P19-1355. URL https://aclanthology.org/P19-1355/.

Tian Y, Zhang Y (2022). "A comprehensive survey on regularization strategies in machine learning." *Information Fusion*, **80**, 146–166.

Vanmassenhove E, *et al.* (2021). "NeuTral Rewriter: A Rule-Based and Neural Approach to Automatic Rewriting into Gender Neutral Alternatives." In MF Moens, X Huang, L Specia, SWt Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8940–8948. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic. doi:10.18653/v1/2021.emnlp-main.704. URL https://aclanthology.org/2021.emnlp-main.704/.

Vaswani A, *et al.* (2017). "Attention is all you need." *Advances in neural information processing systems*, **30**.

Wang A, *et al.* (2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." In T Linzen, G Chrupała, A Alishahi (eds.), *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355. Association for Computational Linguistics, Brussels, Belgium. doi:10.18653/v1/W18-5446. URL https://aclanthology.org/W18-5446/.

Wang A, *et al.* (2019). "Superglue: A stickier benchmark for general-purpose language understanding systems." *Advances in neural information processing systems*, **32**.

Warstadt A, *et al.* (2019). "Neural network acceptability judgments." *Transactions of the Association for Computational Linguistics*, **7**, 625–641.

Webster K, *et al.* (2018). "Mind the GAP: A Balanced Corpus of Gendered Ambiguous Pronouns." *Transactions of the Association for Computational Linguistics*, **6**, 605–617. doi:10.1162/tacl_a_00240. URL https://aclanthology.org/Q18-1042/.

Webster K, *et al.* (2020). "Measuring and reducing gendered correlations in pre-trained models." *arXiv preprint arXiv:2010.06032*.

Wei J, *et al.* (2022). “Chain-of-thought prompting elicits reasoning in large language models.” *Advances in neural information processing systems*, **35**, 24824–24837.

Williams A, *et al.* (2018). “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference.” In M Walker, H Ji, A Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, New Orleans, Louisiana. doi:10.18653/v1/N18-1101. URL https://aclanthology.org/N18-1101/.

Xiong H, *et al.* (2024). “When search engine services meet large language models: visions and challenges.” *IEEE Transactions on Services Computing*.

Zayed A, *et al.* (2023). “Should we attend more or less? modulating attention for fairness.” *arXiv preprint arXiv:2305.13088*.

Zhang Z, *et al.* (2021). “CPM-2: Large-scale cost-effective pre-trained language models.” *AI Open*, **2**, 216–224. ISSN 2666-6510. doi:https://doi.org/10.1016/j.aiopen.2021.12.003. URL https://www.sciencedirect.com/science/article/pii/S2666651021000310.

Zhao J, *et al.* (2018). “Gender Bias in Coreference Resolution: Evaluation and Debiasing Methods.” In M Walker, H Ji, A Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 15–20. Association for Computational Linguistics, New Orleans, Louisiana. doi:10.18653/v1/N18-2003. URL https://aclanthology.org/N18-2003/.