

Bias Mitigation and Detection for LLMs

Sidharth Kolladikkal Biju

January 7, 2026

Contents

1	Introduction	2
1.1	Motivation	2
1.2	Approach	3
2	Prototype	4
2.1	Overview	4
2.2	Bias Neutralisation Subsystem	4
2.2.1	Identifying Bias	4
2.2.2	Parsing and Removing Bias	5
2.2.3	Creating a RESTful API	7
2.2.4	Evaluation of Initial Prototype	9
2.3	Prototype Improvements	10
2.3.1	Overview	10
2.3.2	Improved Tagging	11
2.3.3	Improved Context	13
2.3.4	Highlighting and Rewriting Subsections	17
2.3.5	Deployment	19
2.4	Evaluation	19

Chapter 1

Introduction

1.1 Motivation

LLMs are increasingly being used in place of web browsers (Padilla et al. 2025), so it's critical to understand how LLMs handle misinformation and bias. Since users often see one result from a prompt rather than multiple results in a traditional web browser, users' diversity of sources have reduced, and therefore makes it far more critical that we ensure that LLMs produce neutral and factual information, that doesn't propagate any harmful biases, or misinformation.

It has already been found that LLMs may tend to misclassify left-leaning articles as neutral (Lin et al. 2025), and depending on how LLMs are trained we can inadvertently put inherent bias into the system e.g. ensuring overall performance might bias the model towards majority groups, and against minority groups (Ranjan, Gupta, and Singh 2024). Even modern models that seem to perform well in mitigating the results of bias, due to the availability of bias testing benchmarks have been found to be memorising patterns in these benchmarks. Augmenting the prompt reveals that when the text doesn't match these patterns the LLMs' biases start showing, especially in biases that aren't as well studied such as age, socio-economic, and appearance based biases (Miandoab et al. 2025).

It seems that traditional fine-tuning techniques such as few-shot prompting, role prompting contextual prompting, and system prompting (Chen et al. 2025), might work in specific conditions, but can't be reliably generalised when the input doesn't match the patterns the LLM was trained on (Miandoab et al. 2025).

1.2 Approach

I plan to use vocabulary-based matching (Lin et al. 2025) to identify and evaluate bias. Utilising this as a reliable quantifier for the bias, I plan to use reinforcement learning to debias text, as it has been shown to be effective in more open-ended situations where the chain of thought can't be accurately modelled for general cases (Zhou et al. 2025). I will also be using RAG, as it has been shown to provide a good way to provide context, and a factual foundation (Agada et al. 2025).

Chapter 2

Prototype

2.1 Overview

The prototype aims to explore traditional methodologies in prompt engineering (Chen et al. 2025), and where they might fall short. It also serves as a guide to indicate how the final system architecture should be designed to create the most efficient, and relevant solution.

2.2 Bias Neutralisation Subsystem

2.2.1 Identifying Bias

Rationale

First we need to identify whether or not the original text has any bias in the first place. There's no point trying to remove the bias in a text that has none. This step will save costs, and improve the efficiency of the overall process.

For the prototype I decided to use few-shot prompting, in combination with role prompting (Chen et al. 2025), and system prompting, which is quick and easy, but shown to be unreliable for NLP type tasks, especially when asked to explain their chain of reasoning (X. Ye and Durrett 2022).

Therefore I will only be using it as a placeholder for other methods I plan to implement, namely RAG.

Algorithm

Algorithm 1 Bias Tagger

Require: userInput : string

Require: exampleData : Array[(string, string)]

Require: LLM : string → string

Role : enum ← {SYSTEM, USER, ASSISTANT}

messages : Array[(Role, string)]

messages ← append (Role.SYSTEM, 'tag biased sections')

for (original, tagged) in exampleData **do**

 messages ← append(Role.USER, original)

 messages ← append(Role.ASSISTANT, tagged)

end for

LLM.messages ← messages

taggedText ← LLM(userInput)

return taggedText

2.2.2 Parsing and Removing Bias

Rationale

For the parsing I decided to use Regex, which is a standard way to parse strings. I scan the tagged string to see if it actually has any tags in it. I then append all of the tagged sections into an array. My reasoning for this is that by creating an array of biased sections to rewrite, rather than rewriting the

whole document, I can take advantage of two things:

1. Parallelisation
2. Enhanced user interactivity

Point 1 is key for performance, I have essentially created a job queue, where each job is independent of the other. This is the perfect opportunity to parallelise the workload. This will speed up the performance of the system, where the LLMs only have to rewrite a small amount of text at each time.

Point 2 is more related to user interaction, I plan to allow the user to select which parts of the text to rewrite, giving them control over what is rewritten in their text. It will also be useful for highlighting which parts of the text are biased, and also add additional annotations to these parts of the text.

Overall this method will reduce costs, because even if the user decides to rewrite all of the biased sections, it will only be a subset of the whole text.

For the actual de-biasing of the sections, I'm using an LLM post-trained through few-shot prompting.

Algorithm

Algorithm 2 Text Parser

Require: *taggedText* : string

Ensure: *biasedSections* : array of strings

```
1: regexPattern  $\leftarrow \langle\langle(.*?)\rangle\rangle
2: biasedSections  $\leftarrow$  empty array
3: matches  $\leftarrow$  matchAll(taggedText, regexPattern)
4: for all match  $\in$  matches do
5:   Append match to biasedSections
6: end for
7: return biasedSections$ 
```

Algorithm 3 Neutralise Bias

Require: *section* : string

Require: *LLM* : function (string) → (string)

Require: *exampleData* : array of (*string*, *string*) pairs

Ensure: *neutralisedSection* : string

```
1: Define enum Role ← {SYSTEM, USER, ASSISTANT}  
2: messages ← empty array of (Role, string)  
3: Append (Role.SYSTEM, "neutralise the bias in the text") to  
   messages  
4: for all (original, neutralised) ∈ exampleData do  
5:     Append (Role.USER, original) to messages  
6:     Append (Role.ASSISTANT, neutralised) to messages  
7: end for  
8: LLM.messages ← messages  
9: neutralisedSection ← LLM(section)  
10: return neutralisedSection
```

2.2.3 Creating a RESTful API

Rationale

I decided to use REST APIs to encapsulate all of the functionality, and allow a front-end to call the API. This will provide the user with a GUI, and allow them to interact with the system. For the prototype I implemented a web based GUI, using Typescript and React.

The prototype has a single endpoint:

(POST, "/remove-bias")

which removes the bias of the whole text, in the post body:

{text: string}

The above outlines a valid json request to the endpoint.

Algorithm

Algorithm 4 Bias Removal API

Require: Route (POST, "/remove-bias")

Require: *req* : HTTP request

Ensure: JSON response

```
1: body  $\leftarrow$  req.getBody()
2: if body is invalid then
3:     return JSON({“error” : “invalid request”}) with status 400
4: end if
5: text  $\leftarrow$  body[“text”]
6: if text is empty then
7:     return JSON({“error” : “invalid request”}) with status 400
8: end if
9: taggedText  $\leftarrow$  tagText(text)
10: taggedSections  $\leftarrow$  getTaggedSections(taggedText)
11: if taggedSections is empty then
12:     return JSON({“result” : text}) with status 200
13: end if
14: neutralisedSections  $\leftarrow$  empty array
15: for all section  $\in$  taggedSections do
16:     Append neutraliseSection(section) to neutralisedSections
17: end for
18: neutralisedText  $\leftarrow$  replaceTaggedSections(taggedText, neutralisedSections)
19: return JSON({“result” : neutralisedText}) with status 200
```

Front-End

The front-end is extremely bare-bones, literally just an input text box, a submit button which sends the text to the api, and an output box to display the result.

2.2.4 Evaluation of Initial Prototype

The initial prototype works, but barely. It can turn an explicitly biased text into one that is implicitly biased. Which might be worse in a way, since implicit bias is far more problematic.

For example take the following test case - biased text generated by chatGPT (OpenAI 2025):

```
tests/test_tag_biased_sections The manager claimed that
<>the younger women on the team shouldn't handle negotiations because
they're too emotional<>, and instead suggested giving the important
tasks to <>the older men, who he said are naturally more reliable<>.
He also remarked that <>employees from that specific Asian community usually
struggle with leadership roles and should stick to basic support work<>.
```

```
tests/test_rest_api.py::test_remove_bias The manager claimed that
Team members assigned to negotiations should be based on skills and
```

experience rather than assumptions about emotions, and instead suggested giving the important tasks to the older men, who he said Some individuals tend to be more reliable. He also remarked that Some employees may find leadership roles more challenging and may excel in support work, regardless of their community background.

It's very obvious about what the bias here is, even if it isn't explicitly stated, for example take the section "skills and experience rather than assumptions about emotions, and instead suggested giving the important tasks to the older men", It's obvious that this is targeted at young women, or young people in general even if the bias isn't explicitly stated.

Another point of failure is that the sentence might sometimes just stop making sense: "to the older men, who he said Some individuals tend to be more reliable," this sentence clearly has some grammatical issues.

These are the main limitations of the prototype as it is.

2.3 Prototype Improvements

2.3.1 Overview

To enhance the prototype, I decided to add the following features to the prototype to improve the system architecture, as well as the overall performance.

- Improved prompt-engineering for more descriptive tagging
- Improved context, to improve how well the rewrite fits with the text by providing the LLM with additional context.
- Highlighting subsections of the text, and allowing for specific subsections to be rewritten.
- Containerising the front-end, and back-end.

2.3.2 Improved Tagging

Rationale

I wanted to implement improved tagging so that I had more control over the outcome of the LLM. I wanted to distinguish between different sources of bias, which requires more descriptive tags.

Implementation

To improve the tagging, I had to improve the prompts used currently. For this I used a technique inspired by a concept called meta-prompting (Q. Ye et al. 2024), which simply involves using an LLM to iteratively create better prompts.

So, I used ChatGPT for the meta-prompting, and passed in the previous prompts, whilst specifying that I wanted tags that could determine a bias type.

```
"content": (
    "You are a bias detection system. "
    "Identify explicitly biased language in text. "
    "Surround ONLY the biased phrase(s) with XML-style tags indicating the bias type. "
    "Use ONLY the following tags: <gender>, <age>, <race>. "
    "If multiple biased phrases exist, tag ALL of them. "
    "Do NOT rephrase the sentence. "
    "Do NOT add explanations. "
    "If no bias exists, return the original text unchanged. "
    "Tag only the biased claim itself, not consequences or actions. "
    "If a biased phrase could belong to multiple categories, choose the single most appro
)
```

(OpenAI 2025)

After iterating with the meta-prompting methodology (Q. Ye et al. 2024), this is the output of the LLM.

I also improved the general performance by including more diverse examples for the few-shot prompting methodology (Chen et al. 2025):

```
{
  "role": "user",
  "content": (
    "During the hiring discussion, the director said that women are too emotional "
    "for leadership roles, that older employees struggle to adapt to new technologies, "
    "and that certain racial groups are naturally less intelligent."
  )
},
{
  "role": "assistant",
  "content": (
    "During the hiring discussion, the director said that "
    "<gender>women are too emotional for leadership roles</gender>, "
    "that <age>older employees struggle to adapt to new technologies</age>, "
    "and that <race>certain racial groups are naturally less intelligent</race>."
  )
},
}
```

This is an example where a sentence has multiple sources of bias.

```
{
  "role": "user",
  "content": (
    "The team decided to reschedule the meeting due to conflicting availability."
  )
},
{
  "role": "assistant",
  "content": (
    "The team decided to reschedule the meeting due to conflicting availability."
  )
},
}
```

This is an example where there are no sources of bias. I did this so that the LLM could better handle any edge cases.

2.3.3 Improved Context

Rationale

One of the main problems of the initial prototype was that it sometimes produced results that were grammatically incorrect. To fix this I decided to implement contextual prompting (Chen et al. 2025). But, to do this I had to dynamically generate the context for each biased section.

I did this mainly through the use of regular expressions, and also an added ID field to the tags to identify which context belonged to what tag.

Algorithm

Algorithm 5 Split Context into Normal and Biased Segments

Require: *text* : input string

Ensure: *segments* : list of segmented text blocks

```
1: sentences  $\leftarrow$  split text using regex  $(?<=[\.,\!?\,])\backslash s+$ 
2: segments  $\leftarrow$  empty list
3: for all sentence  $\in$  sentences do
4:   cursor  $\leftarrow$  0
5:   for all match  $\in$  BIAS_PATTERN.finditer(sentence) do
6:     (start, end)  $\leftarrow$  span of match
7:     if start  $>$  cursor then
8:       normal_text  $\leftarrow$  trim(sentence[cursor : start])
9:       if normal_text  $\neq \emptyset$  then
10:        Append {type : "normal", text : normal_text} to
11:          segments
12:        end if
13:      Append {type : "bias", bias_type : match.group(1), text :
14:        trim(match.group(2))} to segments
15:      cursor  $\leftarrow$  end
16:    end for
17:    if cursor  $<$  length(sentence) then
18:      normal_text  $\leftarrow$  trim(sentence[cursor :])
19:      if normal_text  $\neq \emptyset$  then
20:        Append {type : "normal", text : normal_text} to segments
21:      end if
22:    end for
23:  return segments
```

Algorithm 6 Parse Biased Sections with Context Window

Require: $text : \text{string}$

Require: $context_window : \text{integer}$

Ensure: $results : \text{array of records}$

```
1:  $segments \leftarrow \text{splitContext}(text)$ 
2:  $results \leftarrow \text{empty array}$ 
3:  $section\_id \leftarrow 0$ 
4: for  $i \leftarrow 0$  to  $\text{length}(segments) - 1$  do
5:   if  $segments[i].type \neq \text{"bias"}$  then
6:     continue
7:   end if
8:    $ctx\_start \leftarrow \max(0, i - context\_window)$ 
9:    $ctx\_end \leftarrow \min(\text{length}(segments) - 1, i + context\_window)$ 
10:   $context\_parts \leftarrow \text{empty list}$ 
11:  for  $j \leftarrow ctx\_start$  to  $ctx\_end - 1$  do
12:    Append  $segments[j].text$  to  $context\_parts$ 
13:  end for
14:  Append  $\{ context : \text{join}(context\_parts, "") \}, section\_id : section\_id,$ 
       $text : segments[i].text \}$  to  $results$ 
15:   $section\_id \leftarrow section\_id + 1$ 
16: end for
17: return  $results$ 
```

Algorithm 7 Adding IDs to Opening Tags

Require: *tagged_text* : string containing tagged elements

Ensure: *tagged_text* with unique IDs added to opening tags

```
1: id  $\leftarrow$  0
2: i  $\leftarrow$  0
3: while i < length(tagged_text) do
4:   if tagged_text[i] = < and tagged_text[i + 1]  $\neq$  / then
5:     Insert string "id:"id" " at position i + 1 in tagged_text
6:     id  $\leftarrow$  id + 1
7:   end if
8:   i  $\leftarrow$  i + 1
9: end while
10: return tagged_text
```

Algorithms Explanation

The second algorithm splits the text by biased sections and sentences so if a sentence has a biased section: e.g. This is an example <gender id=1> sentence that </gender> contains bias.

it would be parsed as:

```
[{
  {
    type: normal,
    text: This is an example
  },
  {
    type: bias,
    bias_type: gender,
    text: sentence that
  },
  {
    type: normal,
    text: contains bias.
```

```
    }  
]
```

The second algorithm produces the context for each bias section, and associates each section with an ID. If we have a context window n , it simply looks before and after the biased section by n sections, and joins them to create the context.

For example if we have a context window of 2, and we have a biased section at index 5, the context would be constructed by joining all of the sections between indexes 3 to 7.

The final algorithm simply adds the id to the tag, such that they match the section ids. This way we can understand which context belongs to which section, and where to locate the section.

2.3.4 Highlighting and Rewriting Subsections

Rationale

Where in the previous version, users had no control over what part was rewritten, here I aim to give the users full control. Sections identified as bias will be colour-coded, and highlighted. The user can click on the highlighted section to rewrite that specific section of the text.

This is to improve user interactivity, and allow users to see where sources of bias are. It also further optimise costs, since the user doesn't have to rewrite every section if they don't want to, which minimises the requests sent to the LLM.

Demo

I implemented this in React, so It's quite self-explanatory how everything works from just looking at how it looks:

Output:

In deciding who should be promoted within the organisation, management should recognise that young. women. older employees. Older employees are typically resistant to change, struggle to understand new technologies, and slow down teams that need to move quickly. Their experience is often outdated, and retraining them is usually a waste of time and resources. Older staff [are] a liability in competitive, innovation-driven environments. People from certain racial or ethnic backgrounds are commonly perceived as less articulate, less disciplined, and less capable of leadership. These individuals may struggle with communication, cultural fit, or professionalism, especially in global or client-facing roles. White employees are often viewed as more competent, trustworthy, and culturally aligned with corporate values, which makes them a safer and more effective choice for management and strategic positions. For these reasons, organisations that want to maximise efficiency and success should focus on hiring and promoting young, white men, while avoiding candidates who are female, older, or from minority racial backgrounds, as they are more likely to introduce inefficiency, conflict, and underperformance into the workplace.

This is an example after some bias text is inserted, text generated by ChatGPT (OpenAI 2025). So age based bias is highlighted with blue, gender bias is highlighted with yellow, and race based bias is highlighted in pink.

Output:

In deciding who should be promoted within the organisation, management should recognise that young. women. older employees. some employees may require additional support to adapt to change and new technologies, which can affect team pace. Their experience is often outdated, and retraining them is usually a waste of time and resources. Older staff [are] a liability in competitive, innovation-driven environments. individuals from some racial or ethnic backgrounds are sometimes subject to negative stereotypes regarding communication, discipline, and leadership skills. These individuals may struggle with communication, cultural fit, or professionalism, especially in global or client-facing roles. some employees are often perceived as more competent, trustworthy, and culturally aligned with corporate values, which influences their selection for management and strategic positions. For these reasons, organisations that want to maximise efficiency and success should focus on hiring and promoting young, white men, while avoiding candidates who are female, older, or from minority racial backgrounds, as they are more likely to introduce inefficiency, conflict, and underperformance into the workplace.

Here I have selected specific parts of the text to rewrite.

2.3.5 Deployment

For the deployment I have created Docker images for the front-end, and the back-end separately. This will make it easier to scale the application for more intensive tasks, and it will also help in modularising the application further.

2.4 Evaluation

After trying out a few different techniques, it's obvious that relying solely on a traditional engineering approach isn't good enough. Even after added context, there are still issues with grammar such as capitalisation, and the LLM seems to still be rewriting the text in such a way that it turns explicit bias into more subtle implicit bias.

As stated in the introduction, to make the system more reliable, I plan to implement RAG, and reinforcement learning.

Bibliography

- Agada, Joseph Oche et al. (July 2025). “A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions”. In: *arXiv*. DOI: 10.48550/arXiv.2507.18910. arXiv: 2507.18910. URL: <https://arxiv.org/abs/2507.18910>.
- Chen, Banghao et al. (2025). “Unleashing the Potential of Prompt Engineering for Large Language Models: A Comprehensive Review”. In: *Artificial Intelligence Review* xx.xx. Peer-reviewed survey on prompt engineering for LLMs, pp. xx–xx. DOI: 10.xxxx/xxxxxxxx.
- Lin, Luyang et al. (Jan. 2025). “Investigating Bias in LLM-Based Bias Detection: Disparities between LLMs and Human Perception”. In: *Proceedings of the 31st International Conference on Computational Linguistics*. Online: Association for Computational Linguistics, pp. 10634–10649.
- Miandoab, Kaveh Eskandari et al. (2025). “Breaking the Benchmark: Revealing LLM Bias via Minimal Contextual Augmentation”. In: *arXiv preprint* arXiv:2510.23921. Preprint. DOI: 10.48550/arXiv.2510.23921. URL: <https://arxiv.org/abs/2510.23921>.
- OpenAI (2025). *ChatGPT (GPT-5.2)*. Large language model. URL: <https://chat.openai.com/>.
- Padilla, Nicolas et al. (Dec. 2025). *The Impact of LLM Adoption on Online User Behavior*. SSRN Working Paper 5393256. Available at SSRN. SSRN. DOI: 10.2139/ssrn.5393256. URL: <https://ssrn.com/abstract=5393256>.
- Ranjan, Rajesh, Shailja Gupta, and Surya Narayan Singh (2024). “A Comprehensive Survey of Bias in LLMs: Current Landscape and Future Di-

- rections”. In: *arXiv preprint arXiv:2409.16430*. Preprint. DOI: 10.48550/arXiv.2409.16430. URL: <https://arxiv.org/abs/2409.16430>.
- Ye, Qinyuan et al. (2024). “Prompt Engineering a Prompt Engineer”. In: *Findings of the Association for Computational Linguistics (ACL)*, pp. 355–385. URL: <https://aclanthology.org/2024.findings-acl.21.pdf>.
- Ye, Xi and Greg Durrett (Oct. 2022). “The Unreliability of Explanations in Few-shot Prompting for Textual Reasoning”. In: *arXiv*. DOI: 10.48550/arXiv.2205.03401. arXiv: 2205.03401. URL: <https://arxiv.org/abs/2205.03401>.
- Zhou, Yang et al. (Aug. 2025). “Breaking the Exploration Bottleneck: Rubric-Scaffolded Reinforcement Learning for General LLM Reasoning”. In: *arXiv*. DOI: 10.48550/arXiv.2508.16949. arXiv: 2508.16949. URL: <https://arxiv.org/abs/2508.16949>.