Bias Mitigation and Detection for LLMs

Sidharth Kolladikkal Biju

January 6, 2026

Contents

Chapter 1

Introduction

Chapter 2

Prototype

2.1 Bias Neutralisation Subsystem

2.1.1 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 (Boonstra 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 $\rightarrow$ string

Role : enum $\leftarrow$ {SYSTEM, USER, ASSISTANT}
messages : Array[(Role, string)]

messages $\leftarrow$ append (Role.SYSTEM, 'tag biased sections')

for (original, tagged) in exampleData **do**
  messages $\leftarrow$ append(Role.USER, original)
  messages $\leftarrow$ append(Role.ASSISTANT, tagged)
end for

LLM.messages $\leftarrow$ messages

taggedText $\leftarrow$ LLM(userInput)

return taggedText

---

implementation - Python

```
def tag_biased_sections(userInput, exampleData, llm):
    messages = []
    messages.append({
      "role": "system",
      "content": "You are a bias detection system
                 that identifies where bias exists in text, and surrounds those
                 sections with <><> tags"
    })
```

```
    for example in exampleData:
      messages.append({"role": "user", "content": example[0]})
      messages.append({"role": "assistant", "content": example[1]})

    messages.append({"role": "user", "content": userInput})

  response = llm.chat.completions.create(model="gpt-4.1-mini", messages=messages)
  return response.choices[0].message.content
```

—

For the implementation I decided to tag the biased sections with `<><>` tags.

2.1.2 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

$regexPattern \leftarrow \langle\!\langle(.*?)\rangle\!\rangle$

$biasedSections \leftarrow$ empty array

$matches \leftarrow \text{matchAll}(taggedText,\ regexPattern)$

for all $match \in matches$ **do**

    Append $match$ to $biasedSections$

end for

return $biasedSections$

Algorithm 3 Neutralise Bias

Require: *section* : string
Require: LLM : function (string) $\rightarrow$ (string)
Require: *exampleData* : array of (*string*, *string*) pairs
Ensure: *neutralisedSection* : string

Define enum *Role* $\leftarrow$ {SYSTEM, USER, ASSISTANT}
messages $\leftarrow$ empty array of (*Role*, *string*)
Append (*Role*.$SYSTEM$, "neutralise the bias in the text") to *messages*
for all (*original*, *neutralised*) $\in$ *exampleData* **do**
    Append (*Role*.$USER$, *original*) to *messages*
    Append (*Role*.$ASSISTANT$, *neutralised*) to *messages*
end for
LLM.*messages* $\leftarrow$ *messages*
neutralisedSection $\leftarrow$ LLM(*section*)
return *neutralisedSection*

Implementation - Python

```
def get_parsed_text(text):
  return re.findall(r’<>(.*?)<>’, text)

def remove_section_bias(section, llm, exampleData):
  messages = []
  messages.append({
    "role": "system",
    "content":  "you are a bias neutralisation system. Given a
    sentence you should be able to neutralise bias, whilst keeping
    the same meaning within an unknown context"
  })

  for example in exampleData:
    messages.append({
      "role": "user",
```

```
    "content": example[0]
  })
  messages.append({
    "role": "assistant",
    "content": example[1]
  })

messages.append({
  "role": "user",
  "content": section
})


response = llm.chat.completions.create(model="gpt-4.1-mini", messages=messages)
return response.choices[0].message.content
```

2.1.3 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

body $\leftarrow$ *req*.getBody()
if *body* is invalid **then**
  return JSON$\big(\{$"error" : "invalid request"$\}\big)$ with status 400
end if
text $\leftarrow$ *body*["text"]
if *text* is empty **then**
  return JSON$\big(\{$"error" : "invalid request"$\}\big)$ with status 400
end if
$taggedText \leftarrow$ tagText(*text*)
taggedSections $\leftarrow$ getTaggedSections($taggedText$)
if *taggedSections* is empty **then**
  return JSON$\big(\{$"result" : *text*$\}\big)$ with status 200
end if
neutralisedSections $\leftarrow$ empty array
for all *section* $\in$ *taggedSections* **do**
  Append neutraliseSection(*section*) to *neutralisedSections*
end for
$neutralisedText \leftarrow$ replaceTaggedSections($taggedText$, *neutralisedSections*)
return JSON$\big(\{$"result" : $neutralisedText\}\big)$ with status 200

Implementation - Python (Flask)

```
@app.route("/remove-bias", methods=["POST"])
def remove_bias():
  body = request.get_json(silent=True)
  if body is None:
    return jsonify({"error": "invalid request"}), 400
```

```
  text = data.get("text")
  if text is None:
    return jsonify({"error": "invalid request"}), 400

  res = remove_text_bias(text)

  return jsonify("result": res), 200

def remove_text_bias(text):
  tagged_text = tag_biased_sections(text)
  biased_sections = get_biased_sections(tagged_text)

  if len(biased_sections) == 0:
    return text

  neutralised_sections = []

  for section in biased_sections:
    neutralised_sections.append(remove_section_bias(section))

  for section in neutralised_sections:
    neutralised_text = re.sub(r'<>(.*?)<>', section, neutralised_text, count=1)

  return neutralised_text
```

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.

2.1.4 Evalutation of Initial Prototype

The initial prototype works, but barely. It can turn an explicity biased text into one that is implicity biased. Wich 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 explicity 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.

2.2 Prototype Improvements

2.2.1 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.

2.2.2 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 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 would 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)

Here is an example of how I used meta-prompting to improve the system prompting (Boonstra 2025).

I also improved the general performance by including more diverse examples for the few-shot prompting methodology (Boonstra 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.

2.2.3 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 (Boonstra 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

$sentences \leftarrow$ split $text$ using regex `(?<=[.!?])\s+`
$segments \leftarrow$ empty list
for all $sentence \in sentences$ **do**
    $cursor \leftarrow 0$
    for all $match \in$ `BIAS_PATTERN.finditer(sentence)` **do**
        $(start, end) \leftarrow$ span of $match$
        if $start > cursor$ **then**
            $normal\_text \leftarrow \text{trim}(sentence[cursor : start])$
            if $normal\_text \neq \emptyset$ **then**
                Append $\{type : \text{"normal"},\ text : normal\_text\}$ to $segments$
            end if
        end if
        Append $\{type : \text{"bias"},\ bias\_type : match.group(1),\ text : \text{trim}(match.group(2))\}$ to $segments$
        $cursor \leftarrow end$
    end for
    if $cursor < \text{length}(sentence)$ **then**
        $normal\_text \leftarrow \text{trim}(sentence[cursor :])$
        if $normal\_text \neq \emptyset$ **then**
            Append $\{type : \text{"normal"},\ text : normal\_text\}$ to $segments$
        end if
    end if
end for
return $segments$

Algorithm 6 Parse Biased Sections with Context Window

Require: $text$: string

Require: $context\_window$: integer

Ensure: $results$: array of records

$segments \leftarrow$ splitContext($text$)
$results \leftarrow$ empty array
$section\_id \leftarrow 0$
for $i \leftarrow 0$ to length($segments$) $- 1$ **do**
  if $segments[i].type \neq$ "bias" **then**
    continue
  end if
  $ctx\_start \leftarrow \max(0,\ i - context\_window)$
  $ctx\_end \leftarrow \min(\text{length}(segments) - 1,\ i + context\_window)$
  $context\_parts \leftarrow$ empty list
  for $j \leftarrow ctx\_start$ to $ctx\_end - 1$ **do**
    Append $segments[j].text$ to $context\_parts$
  end for
  Append { $context$: join($context\_parts$, ""), $section\_id$: $section\_id$, $text$: $segments[i].text$ } to $results$
  $section\_id \leftarrow section\_id + 1$
end for
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

$id \leftarrow 0$

$i \leftarrow 0$

while $i <$ length(*tagged_text*) **do**

  if $tagged\_text[i] = \langle$ **and** $tagged\_text[i+1] \neq /$ **then**

    Insert string "id:"*id*" " at position $i + 1$ in *tagged_text*

    $id \leftarrow id + 1$

  end if

  $i \leftarrow i + 1$

end while

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.

Implementation - Python

```
def add_bias_id(tagged_text):
    id = 0
    for i in range(len(tagged_text)):
        if tagged_text[i] == '<' and tagged_text[i + 1] != '/':
            tagged_text = tagged_text[:i + 1] + f"id:{id} " + tagged_text[i + 1:]
            id += 1

    return tagged_text

def split_context(text):
    sentences = re.split(r'(?<=[.!?])\s+', text)

    segments = []

    for sentence in sentences:
        cursor = 0

        for match in BIAS_PATTERN.finditer(sentence):
```

```
            start, end = match.span()

            if start > cursor:
                normal_text = sentence[cursor:start].strip()
                if normal_text:
                    segments.append({
                        "type": "normal",
                        "text": normal_text
                    })

            segments.append({
                "type": "bias",
                "bias_type": match.group(1),
                "text": match.group(2).strip()
            })

            cursor = end

        if cursor < len(sentence):
            normal_text = sentence[cursor:].strip()
            if normal_text:
                segments.append({
                    "type": "normal",
                    "text": normal_text
                })

    return segments


def parse_biased_sections(text, context_window = 2):
    segments = split_context(text)
    results = []
    section_id = 0
```

```
for i, segment in enumerate(segments):
    if segment["type"] != "bias":
        continue

    ctx_window_start = max(0, i - context_window)
    ctx_window_end = min(len(segments) - 1, i + context_window)

    context_parts = [segments[i]["text"] for i in
      range(ctx_window_start, ctx_window_end)]

    results.append({
            "context": " ".join(context_parts),
            "section_id": section_id,
            "text": segment["text"]
            })

    section_id += 1

return results
```

2.2.4 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.

2.2.5 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.

2.3 Evaluation

After trying out different techniques, it's obvious that utilising solely a few-shot prompting 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.

For further development regarding prompt engineering, I would like to implement chain of thought reasoning (Boonstra 2025), where I might explain a chain of thought for determining whether there is bias against a specific group.

In addition, it's obvious that using a small dataset isn't enough to tackle a wide area like bias, even if it's been split into sub-categories, which is why I believe it is necessary to implement a RAG system, which can provide better context, and factual grounding for my system (Agada et al. 2025).

I also want to implement reinforcement learning, where I train an LLM as a grader and iteratively produce outputs until a desired level of quality is achieved. This seems to be the approach taken to tackle more open-ended criteria, such as bias, where we cannot have a mathematical specification to quantify the quality of a text (Zhou et al. 2025).

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.

Boonstra, Lee (Feb. 2025). *Prompt Engineering*. URL: https://www.onlinebrandambassadors.com/stable/Google-Prompt-Engineering-Boonstra.pdf.

OpenAI (2025). *ChatGPT (GPT-5.2)*. Large language model. URL: https://chat.openai.com/.

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.