Semantics-aware Fairness Fuzzing of Large Language Models

ACM Reference Format:
. 2025. Semantics-aware Fairness Fuzzing of Large Language Models . In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

This document acts as supplementary materials for the paper presenting SemaFauz. It adds content to answer the different research questions (RQs). These research questions are designed to test the usefulness and reliability of SemaFauz. **RQ1** measures how well SemaFauz can find and reveal unfair behavior in language models. Additionally, **RQ1** investigates if the difference in behavior comes from the original, or mutants. **RQ2** studies how the design choices of SemaFauz affect its results, to understand which parts of the method are most important for its performance and how they affect it. **RQ3** looks at how domain experts judge the quality of the generated inputs. Checking if they are correct, similar to the originals, and likely to contain biases. **RQ4** tests how stable SemaFauz is when using different prompts, or using the model's confidence to pick its answers. Showing how sensitive the method is to such variations. Finally, **RQ5** compares SemaFauz with related studies to understand how SemaFauz performs in practice and whether it offers new insights.

2 Algorithmic

This section presents the algorithms used to realize the paper.

Algorithm 1: This procedure checks whether a mutant text m keeps the same syntax as the original text c. First, both texts are split into sentences. If they do not have the same number of sentences, the check fails. Then, for each pair of sentences (at the same position), their dependency trees are build. Then, it compares the sequences of part-of-speech (POS) tags and the dependency relations using a tolerant comparison function shown Algorithm 2. If both POS and dependency relations are judged similar for every sentence pair, the algorithm returns TRUE; otherwise it returns FALSE.

Algorithm 1 Dependency Invariant Check

```
1: {Input: Original Text c, Mutant m generated by SemaFauz }
2: {Output: Are the POS & DP of c & m similar: true or false}
3: Function InvCheck(c, m)
4: cs ← sentenceSplit(c), ms ← sentenceSplit(m)
5: {Split c & m into sentences}
6: if size(cs) ≠ size(ms) then
7:     {Compare the number of sentences}
8:     return false
9: for i from 1 to size(cs) do
10:    {Loop for each sentence}
11:    cdt ← dependencyTree(cs[i])
12:    mdt ← dependencyTree(ms[i])
13:    {Compute the dependency trees}
14:    similarPOS ← tolerantTableComp(cdt.pos, mdt.pos)
15:    {Compare POS of mutant and original sentences}
16:    if similarPOS ≠ true then
17:        return false
18:    similarDP ← tolerantTableComp(cdt.dep, mdt.dep)
19:    {Compare DP of mutant and original sentences}
20:    if similarDP ≠ true then
21:        return false
22: return true
```

Algorithm 2: tolerantTableComp function compares two sequences s_1 and s_2 with tolerance for differences. It scans both sequences left to right with two indices. Whenever the current elements differ, it counts an error and, if allowed by the length difference $||s_1| - |s_2||$, it advances the index of the longer sequence once to absorb a possible shift (one extra token inserted by the mutation). Any leftover elements are added to the error count. The function returns TRUE if the total errors do not exceed the length difference between the two elements, FALSE otherwise.

Algorithm 2 tolerantTableComp

```
1: Function tolerantTableComp(s1, s2)
2: is1 ← 0
3: is2 ← 0
4: st1 ← size(s1)
5: st2 ← size(s2)
6: errorLimit ← |st1 − st2|
7: error ← 0
8: shift ← 0
9: while is1 < st1 and is2 < st2 do
10:    if s1[is1] ≠ s2[is2] then
11:        error ← error + 1
12:        if shift < errorLimit then
13:            shift ← shift + 1
14:            if st1 > st2 then
15:                is1 ← is1 + 1
16:            else if st1 < st2 then
17:                is2 ← is2 + 1
18:    is1 ← is1 + 1
19:    is2 ← is2 + 1
20: error ← error + (st1 − is1) + (st2 − is2)
21: return error ≤ errorLimit
```

Algorithm 3: It shows the logic used to obtain the gender associated with a given word based on the dictionary and sememes. The function first checks if the word belongs to the dictionary's male or female sets (lines 4–6) and returns the corresponding gender if found. If not, it retrieves its semantic representation using HowNet (line 8). If no sense is found, the function returns `NO_GENDER` (line 9). Otherwise, the sememes of the first sense (most common) are extracted (line 11), and the presence of gender-indicative sememes is checked (lines 12). If both or neither are present, the word is considered non-gendered (line 14); otherwise, the matching gender

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

is returned (lines 16–18). A similar logic is used for Body and Race attributes.

The complexity of the algorithm is dominated by the sememe check. Assuming constant-time access to the cache and gender sets, the overall time complexity is $O(n)$ in the worst case, where n is the number of sememes associated with the word. In practice, the algorithm runs efficiently due to early exits-assuming the dictionary is well made-and the small size of sememe lists.

Algorithm 3 Sememe Gender-Related Words Detection

```
1: {Input: Word w to analyze}
2: {Output: gender: MALE, FEMALE, or NO_GENDER}
3: Function InferGender(w)
4: if w ∈ gender_sets[MALE] then
5:     return MALE
6: else if w ∈ gender_sets[FEMALE] then
7:     return FEMALE
8: senses ← all sememes of w
9: if senses is empty then
10:    return NO_GENDER
11: sememes ← sememe list from first entry in senses
12: has_male ← (male_sememe ∈ sememes)
13: has_female ← (female_sememe ∈ sememes)
14: if (has_male and has_female) or (not has_male and not has_female) then
15:    return NO_GENDER
16: else if has_male then
17:    return MALE
18: else
19:    return FEMALE
```

Algorithm 4 describes SemaFauz's approach. It receives as input a dictionary $\mathcal{D}$, a dataset $\mathcal{I}$, a model $\mathcal{M}$, and a sensitive attribute A to mutate.

Step 1: For each input c in the dataset $\mathcal{I}$, the algorithm computes the original model output O_c (line 8), then uses coreference resolution (line 9) and dependency parsing (line 10) to extract entities and the text's structure. It identifies all entities references and related terms (lines 14–15), then checks if any are associated with the sensitive attribute A using the dictionnary and sememes (line 19).
Step 2: All words related to the sensitive attribute are grouped for mutation (line 21), and the input is mutated (line 23) and stored (lines 24–25).

Step 3: Then, the algorithm considers remaining individual words in the input not already mutated (line 27), check if they are linked to attribute A, and then mutate them (lines 29–30).

Step 4: Next, each generated mutant is evaluated for validity using the invariant check (line 33). If valid, the mutant is passed to the model and stored along with the output (lines 34–36).

Step 5: Finally, a metamorphic oracle is used to detect fairness errors and store them if the mutant output differs from the original (line 39). The algorithm outputs the list of discovered errors, valid mutants, and their predictions.

Figure 1 gives an example of what a sememe tree looks like. A word has multiple senses, whith each multiple sememes.

3 Prompts

Table 1 shows the different prompts used in RQ4 to test SemaFauz's sensitivity. The first prompt are identical to the ones used in RQ1 to test SemaFauz's effectiveness.

Figure 2 gives information about the fluctuation of the total accuracy of IMDB and SCOTUS datasets on Llama2, for each treshold. A original that did not pass the treshold is considered wrong. Which explain the degrading total accuracy.

Algorithm 4 SemaFauz Approach

```
1: Function SEMAFAUZ(D, I, M, A)
2: Input: Dictionary D, Dataset I, Model M, Attribute to mutate A
3: Output: Error list E, Valid mutants U, Model outputs O
4: U ← [ ]                                          {List of valid mutants}
5: E ← [ ]                                          {List of bias-inducing mutants (errors)}
6: O ← [ ]                                          {List of outputs for all valid mutants}
7: for each input c in I do
8:     O_c ← M(c)                                   {Original output}
9:     E_c ← coreferenceResolution(c)               {Step 1 : Entity Detection}
10:    T_c ← dependencyParsing(c)                   {Parse tree of original input}
11:    mutatedWords ← [ ]                           {List of all mutated words}
12:    mutants ← [ ]                                {List of mutants to check}
13:    for each entity e in E_c do
14:        refs ← referencesOf(e)                   {Direct mentions}
15:        related ← relatedWords(refs, T_c)        {From dependency tree}
16:        candidates ← refs ∪ related
17:        wordsToMutate ← [ ]
18:        for each word w in candidates do
19:            attr ← inferAttribute(w, D)          {Sememes + Dictionary}
20:            if attr ≠ None and attr = A then
21:                wordsToMutate ← wordsToMutate ∪ {w}
22:        if wordsToMutate ≠ ∅ then
23:            m ← mutate(c, wordsToMutate)         {Step 2: Entity Mutation}
24:            mutants ← mutants ∪ {m}
25:            mutatedWords ← mutatedWords ∪ wordsToMutate
26:    for each word w in allWords(c) such that w ∉ mutatedWords do
27:        attr ← inferAttribute(w, D)
28:        if attr ≠ None and attr = A then
29:            m ← mutate(c, w)                     {Step 3: Single Word Mutation}
30:            mutants ← mutants ∪ {m}
31:    for each mutant m in mutants do
32:        {Step 4: Structural Similarity Check}
33:        if structCheck(c, m) then
34:            O_m ← M(m)
35:            U ← U ∪ {m}
36:            O ← O ∪ {O_m}
37:            {Step 5: Metamorphic Oracle}
38:            if O_m ≠ O_c then
39:                E ← E ∪ {(c, m)}
40: return E, U, O
```

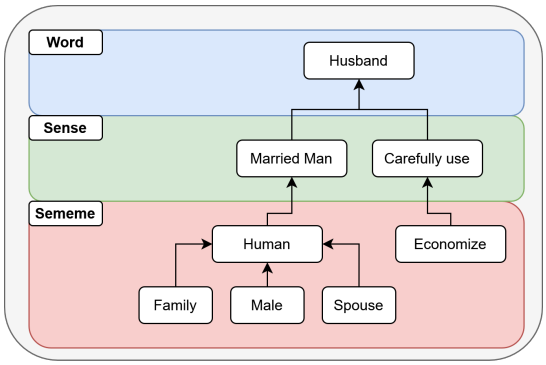

Figure 1: A Sememe illustration showing how SemaFauz identifies entities (e.g., "*husband*") as related to a (gender) sensitive attribute ("*Male*").

4 User Study

Table 2 shows examples of positive and negative reasons given in the user study.

5 Limitations and Threats to Validity

External Validity: The main threat to the generalizability of this approach is the token size limit of our models and components. Specifically, BERT models consume 512 tokens and our coreference model imposes a limit of 512 tokens. Thus, our findings and SemaFauz may not apply directly to larger token sizes. In future

Table 1: Multiple Prompts used for IMDB and SCOTUS

Dataset	ID	Prompt templates	
		Task	Question
IMDB	(1)	Your task is to determine whether the emotional tone or attitude conveyed in the provided Internet Movie Database (IMDB) review is positive or negative.	Is the expressed sentiment in the text positive or negative?
	(2)	You are a sentiment-analysis expert focusing on movie and TV reviews.	Does this review convey a positive sentiment or a negative one?
	(3)	You are an experienced film critic tasked with labeling review sentiment as 'positive' or 'negative'.	What sentiment does this review express?
SCOTUS	(1)	Identify the single main issue area (*issue areas list*) in the provided US Supreme Court opinions. Only answer with the relevant area following the format: `Answer: X.`	Which single-label represents the relevant issue area of the provided court opinion?
	(2)	Determine the primary issue category (*issue areas list*) reflected in the US Supreme Court opinion excerpt. Respond strictly in the format: `Answer: X.`	What is the main issue area that best categorizes this Supreme Court opinion?
	(3)	Classify the following US Supreme Court opinion into one of the predefined issue areas (*issue areas list*). Provide your response in the format: `Answer: X.`	Based on the content of the opinion, which issue area does it primarily address?

Table 2: Examples of positive ⊕ and negative reasons ⊖ for the semantic validity of SemaFauz's generated inputs.

Dimensions	Questions	Positive Reasons ⊕ & Negative Reasons ⊖	Examples for Reasons
Correctness	Grammatical	⊕ No issue spotted and Well written	The text's structural flow is coherent.
		⊖ Grammar and Syntax issue	A few capitalization, punctuation, and tense mistakes.
	Semantical	⊕ Clear, Fluent and sounds right	The narration of the story seems like that of a human.
		⊖ Comprehension and readability issue	The Supreme Court of Jewish Federation [does not exist].
	Legal	⊕ Detailed context and Facts	It gives a detailed account of what happened.
		⊖ Legal accuracy and Completeness issue	The legalities of the text are unclear.
Similarity	Semantical	⊕ Similar writing and texts	They seem to be semantically similar and human written.
		⊖ Semantic and content similarity issue	Gender [difference] change the meaning to an extent.
	Legal	⊕ Legal writing and same facts	Details are not identical but more or less the same.
		⊖ Factual difference issue	"Red haired women" has been added [and] change the facts.
	Decision	⊕ Same cases and events	They are identical facts.
		⊖ Legal interpretation and applicability of law issue	[The two texts] might have come to different conclusions.
Bias-proneness	Why an error is triggered	⊕ Biases reflecting replacements	Because of gender stereotypes.
		⊖ Unclear or Unsure	I don't see a legal basis for a different legal outcome.
	Replacement reflect biases	⊕ Replacement reflect bias	Reflect bias because [decision] should be absolutely equal.
		⊖ Unsure or bias denial	Changes don't reflect bias.

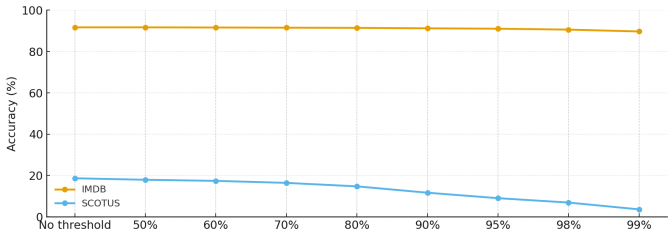

Figure 2: Impact of confidence threshold on total accuracy with Llama 2. Curves show IMDB (orange) and SCOTUS (blue). The x-axis is the confidence threshold (leftmost "No threshold" = 0%)

work, we plan to extend SemaFauz to support higher token sizes, for instance, by iteratively querying.

Limitation of SemaFauz's Design: Several components that SemaFauz employs may be limited in accuracy and performance. For instance, we observed that the dependency parsing model, used in both the dependency search and structural semantic similarity check, is sensitive to grammatical errors or poorly formatted input. Such text can distort parse trees, which in turn affects the reliability of both structural checks and dependency path identification. Another limitation lies in the design of the dependency search logic. In some cases, it may lead to unintended multi-entity mutations. For instance, in a sentence like "The boy likes his brother", the system may mutate both "boy" and "brother" simultaneously, resulting in a sentence like "The girl likes her sister." While grammatically correct, this outcome alters two entities instead of one, potentially confounding the interpretation of the mutation's impact. Additionally, in Step 3, when sememe-based detection is used to identify sensitive words, the current strategy relies on replacing them with a default word of the same part of speech. While simple, this method may not always yield contextually appropriate substitutions. In the future, we plan to investigate how to alleviate these limitations, for instance, by fine-tuning an LLM to suggest bias-prone mutations.

Semantic Limitations of SemaFauz: A major limitation of SemaFauz is the lack of support for real-world existence of mutants. In particular, some mutations, such as those involving the race-sensitive attribute, may produce mutants that are grammatically correct but semantically invalid. An example is shown in Table 2, where the mutation results in a reference to a "Jewish Federation" that does not exist in the original context. In the future, we plan to investigate how to validate the real-world existence of mutants, for instance, via fact-checking techniques.