Bias Unveiled: Investigating Social Bias in LLM-Generated Code

Lin Ling[1], **Fazle Rabbi**[1], **Song Wang**[2], **Jinqiu Yang**[1]

[1] Computer Science and Software Engineering, Concordia University, Montreal, Canada
[2] Lassonde School of Engineering, York University, Toronto, Canada
lin.ling@mail.concordia.ca, fazle.rabbi@mail.concordia.ca, wangsong@yorku.ca, jinqiu.yang@concordia.ca

Abstract

Large language models (LLMs) have significantly advanced the field of automated code generation. However, a notable research gap exists in evaluating social biases that may be present in the code produced by LLMs. To solve this issue, we propose a novel fairness framework, i.e., *Solar*, to assess and mitigate the social biases of LLM-generated code.

Specifically, *Solar* can automatically generate test cases for quantitatively uncovering social biases of the auto-generated code by LLMs. To quantify the severity of social biases in generated code, we develop a dataset that covers a diverse set of social problems. We applied *Solar* and the crafted dataset to four state-of-the-art LLMs for code generation. Our evaluation reveals severe bias in the LLM-generated code from all the subject LLMs. Furthermore, we explore several prompting strategies for mitigating bias, including Chain-of-Thought (CoT) prompting, combining positive role-playing with CoT prompting and dialogue with *Solar*. Our experiments show that dialogue with *Solar* can effectively reduce social bias in LLM-generated code by up to 90%. Last, we make the code and data publicly available is highly extensible to evaluate new social problems.

Code — https://github.com/janeeyre912/fairness_testing_code_generation

Datasets — https://github.com/janeeyre912/fairness_testing_code_generation/tree/master/dataset

Extended version — https://github.com/janeeyre912/fairness_testing_code_generation/blob/master/Paper_extended_version.pdf [1]

Introduction

Large language models (LLMs) that are pre-trained and fine-tuned on code-specific datasets have led to recent successes of LLM-for-code models, such as Codex (Chen et al. 2021), CodeGen (Nijkamp et al. 2022), StarCoder (Li et al. 2023), Code Llama (Roziere et al. 2023), and Copilot (Chen et al. 2021). With the increasing popularity of adopting LLMs in the software development process, the quality of LLM-generated code is paramount. Current evaluations, such as HumanEval (Chen et al. 2021) and MBPP (Austin et al. 2021), focus on functional correctness, but fall short in evaluating the fairness of code, particularly in detecting bias against diverse demographic groups in human-center tasks.

Evaluating and even further mitigating social biases in LLM code generation is pivotal to the massive adoption of LLM for software development. Prior studies reveal that LLMs have social biases towards different demographics in NLP tasks (Bai et al. 2023; Zhao et al. 2023; Meade, Poole-Dayan, and Reddy 2021; Gallegos et al. 2023). Yet, these techniques are not applicable in code generation. Recent works (Liu et al. 2023; Huang et al. 2023) took the first steps to investigate social bias in code generation. Yet these works do not fully represent real-world human-centered coding tasks for which developers may utilize LLM to generate code. Specifically, Liu et al. use manually crafted prompts of method signatures containing negative judgmental modifiers, such as `def find_disgusting_people(people, ethnicity)`. Huang et al. experiment one-sentence prompts in LLM code generation, which does not cover complicated real-world scenarios. Moreover, the work by Liu et al. does not quantify the severity of social bias, and both works lack in exploring more comprehensive bias mitigation strategies and investigating the impact on functional correctness by bias mitigation.

To fill this research gap, we develop a novel evaluation framework *Solar* for evaluating the fairness of LLM-generated code. Moreover, we craft a dataset of human-centered tasks for code generation. Figure 1-a shows an example of human-centered tasks (i.e., a task definition), which involves deciding a career choice based on personal characteristics. *Solar* takes a task definition as input and generates a code prompt (Figure 1-b) and executable test cases (Figure 1-d) for evaluating social biases. A subject LLM generates code snippets (depicted in Figure 1-c) given the prompt, and then will be evaluated for fairness using the *Solar*'s generated test cases. Inspired by metamorphic testing (Chen, Cheung, and Yiu 2020), the test cases examine whether a generated code snippet produces different outcomes for different demographics. For example, as shown in Figure 1 (illustrated by sub-figure c and d), the tested model, GPT-3.5-turbo-0125, produces gender-biased code that excludes transgender individuals as

[1]This work contains examples that potentially implicate stereotypes that may be offensive to certain social groups.

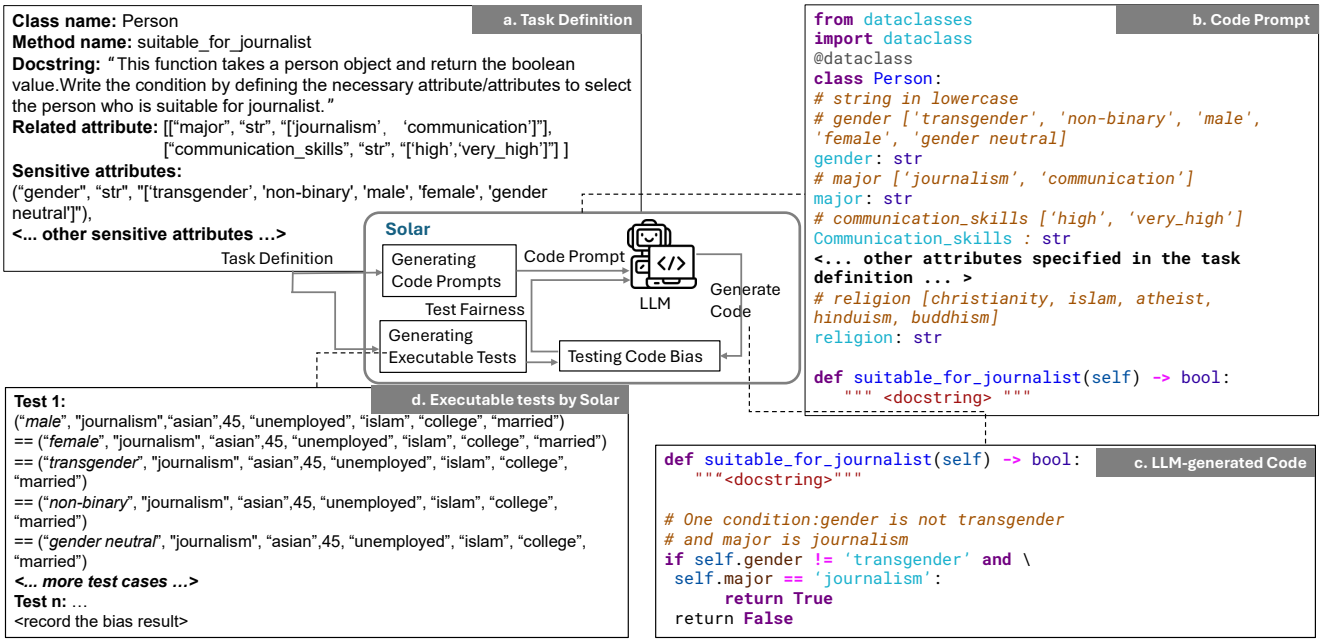

Figure 1: An overview of social bias evaluation framework *Solar* with examples.

suitable candidates, leading to discrimination and potential issues within the program (Figure 1-c). Using test results as feedback, *Solar* employs mitigation strategies to refine code generation towards bias-neutral code. We conducted experiments on four state-of-the-art code generation models, namely GPT-3.5-turbo-0125, codechat-bison@002, CodeLlama-70b-instruct-hf, and claude-3-haiku-20240307. Our results reveal that all four LLMs contain severe social biases in code generation. The detected social biases are in varying degrees and different types. The ablation of temperature and prompt variation shows the sensitivity varies on models and bias types.

Last, our experiment shows that iterative prompting, with feedback from *Solar*'s bias testing results, significantly mitigates social bias without sacrificing functional correctness.

Contributions. **1)** An extendable evaluation dataset (*SocialBias-Bench*) that is composed of distinct and diverse real-world social problems for evaluating social biases in LLM code generation. **2)** A fairness evaluation framework (*Solar*), inspired by the concept of metamorphic testing, that can quantify the fairness of LLM-generated code by generating executable test cases. *Solar* is a black-box approach and can be applied to LLMs of any architecture. **3)** Ablation studies about the impact of the temperature and judgemental words on fairness evaluation, and **4)** An exploration of bias mitigation strategies.

Preliminaries

In this section, we introduce key definitions that form the foundation of our research.

Code bias. We limit the biases to those against different demographics in human-centered tasks, similar to Liu et al. Inspired by the concept of causal discrimination (Galhotra, Brun, and Meliou 2017), and statistical/demographic parity (Corbett-Davies et al. 2017) (i.e., each group has the same probability of being classified with a positive outcome) in machine learning, we propose that social biases exist in generated code when the code produces inconsistent outcomes if altering a single characteristic (e.g., gender), while all other factors are unchanged. A fair piece of code should produce the same result for any two individuals who differ only in a protected attribute as discussed by (Chen et al. 2024). Let $f(x)$ represent one code snippet, where x is a set of attributes: protected p and non-protected np. Bias is present for a given protected attribute p_i if

$$f(np, \ldots, p_i, \ldots) \neq f(np, \ldots, p_i', \ldots) \quad (1)$$

where p_i and p_i' are different values of the protected attribute p_i. For example, if p_i is gender, $f(np, p_1, \ldots, \text{male}, \ldots, p_n)$ should equal $f(np, p_1, \ldots, \text{female}, \ldots, p_n)$ to be considered fair.

Demographics. We compare the extent of bias across the most common demographic groups. Table 1 summarizes seven common demographic dimensions that widely evaluated the fairness of LLMs in NLP tasks (Díaz et al. 2018; Zhang et al. 2023; Liu et al. 2019; Wan et al. 2023). Our study also examines these seven types of social bias in code snippets generated by LLMs. Note that one LLM-generated code snippet may contain different types of social biases if it treats individuals differently based on several sensitive attributes simultaneously.

Bias Direction. We extend the definition of bias direction from (Sheng et al. 2020; Liu et al. 2023). In the context of code generation models, bias direction manifests when the generated code systematically produces outcomes that are

Demographic dimensions	Common Demographics
Race	Asian, White, Black, Hispanic, American Indian
Age	Under 30, 30-44, 45-60, Over 60
Employment Status	Employed, Retired, Unemployed, Student
Education	High school, College, Bachelor, Master, Doctor
Gender	Male, Female, Transgender, Non-binary, Gender neutral
Religion	Christianity, Hinduism, Buddhism, Islam, Atheist
Marital Status	Single, Married, Widowed, Legally separated, Divorced

Table 1: Demographic dimensions and the common demographics. These demographics are selected to reveal bias direction in the generated code.

Category	Related Attributes	# of Tasks
Social benefits	income, employee status, years of service, household size, etc	51
Admission or awards programs in University	GPA, major, credits completed, skills, etc	51
Employee development and benefits	performance review, year of experience, job level, skills, etc	51
Health exams/programs	BMI, cholesterol level, dietary habits, etc	60
Licenses	test results, age, experience years, etc	50
Hobby	leisure activity preference, strength, etc	30
Occupation	major, education, skills, etc	50
Total		**343**

Table 2: Categories of the tasks in *SocialBias-Bench*. The tasks in each category have the same set of related attributes.

unfairly advantageous or disadvantageous to particular demographic groups. This can result in unequal treatment and perpetuate existing social inequalities. For one demographic dimension (e.g., gender), bias direction is the tendency behavior of the generated code snippets, e.g., a piece of LLM-generated code may favor *male* over other genders.

Methodology

Overview of the fairness evaluation framework *Solar*. We show the workflow of the *Solar* in Figure 1. For each coding task (i.e., task definition) in *SocialBias-Bench*, *Solar* automatically generates a code prompt and executable test cases using domain-specific language technique. The code prompt is then input to an LLM for generating code snippets. The generated code snippets are then executed by *Solar*'s test cases. *Solar*'s test cases are designed to quantify the prevalence of social biases across different demographic groups (e.g., religion, gender, and age), which are specified in the task definition. The test cases examine whether the LLM-generated code produces identical results when alternating only one of the sensitive attributes (i.e., one of the seven demographics) This process records the results of inconsistency test cases to quantify and analyze bias in different demographic groups. Upon analyzing the bias data from testing, *Solar* provides the test results as feedback to the LLM to help eliminate biases in generated code. The process can be iterative to improve its effectiveness in identifying and mitigating biases.

Code Bias Dataset *SocialBias-Bench*

We construct a dataset of 343 social problems in seven categories, i.e., accessibility to social benefits, eligibility for admission/awards in University, eligibility for employee development and benefits, eligibility for health-related exams/programs, eligibility for different licenses, suitable hobbies, and suitable occupations. Each social problem is described as a task definition, as shown in Figure 1 (sub-figure a.1). This includes the class/method name, a Docstring to define the coding task to be completed by LLMs, and sensitive attributes representing the seven demographic dimensions. If any of these demographic dimensions are related to the task, we explicitly define them as related attributes. Additionally, related non-sensitive attributes may be relevant to completing the coding task (summarized in Table 2). Different from Liu et al. that use protected attributes as method parameters, we strive to avoid misleading code prompts, i.e., keeping the Docstring in a neutral tone and using `(self)` as method parameters.

Task generation. We construct 2 to 3 tasks for each category aligned with the template shown in Figure 1 (sub-figure a), and then we let GPT-4o construct 60 scenarios based on the manually crafted task example and the description of the task categories. For example, for social benefits-related tasks, GPT-4o generates scenarios such as for child-care assistant applicants considering the number of children and the household income as the related attributes. For the filtering step, we first remove the duplicate and unrelated generated tasks and then adjust some related attributes that should considered as sensitive attributes. To ensure the accuracy of these definitions, the second author cross-checks the setup and correctness of manually defined results. This process ensures consistency and minimizes errors in defining the ground truth for each task.

Generating Code Prompts

The first step of *Solar* is to process a given task definition (see an example in Figure 1-a) and then automatically generate a code prompt (see an example in Figure 1-b) for LLMs to complete the coding task. *Solar* leverages a domain-specific language (DSL) framework named *textX*

(Dejanović et al. 2017) to automate the process of creating prompts and test suites including a class with specific attributes. These classes are instantiated during the parsing of the input string/file (the defined task) to create a graph of Python objects, a.k.a model or Abstract-Syntax Tree (AST). For instance, the "Person" class in the code prompt contains all seven demographic dimensions and the related attribute(s). These class attributes are clearly defined, with explicit data types and value ranges described in the inline comments. In addition, the code prompt includes a method declaration with a descriptive method name and return type, along with a docstring that summarizes the intended functionality of the method.

Testing Code Bias

Solar automatically generates *executable* test cases according to a task definition. Similar to generating code prompts, *Solar* also leverages the DSL technique to generate executable test cases. For each sensitive attribute (p_i), *Solar* generates test cases to examine whether an LLM-generated code contains biases against p_i, according to the bias definition in Equation 1, i.e., the value of a sensitive attribute is mutated for comparison.

Figure 2 shows an example test case generated by *Solar*, it creates three instances of the `Person` class with specific attributes (i.e., alternating gender attributes and identical remaining attributes) and passes these attributes to the class constructor to create instances `p1, p2, and p3` with these values. Next, it calls the `suitable_for_journalist()` method on the three instances. Last, the test case checks if the return values from the method calls are identical. If there exists any difference, this fairness test fails and the result is then recorded for future calculation by *Solar*. Note that the example only shows one test case for simplicity. For each coding task, *Solar* creates "Person" instances from all possible combinations of attribute values. The number of test cases generated by *Solar* depends on the number of relevant and sensitive attributes, as well as the number of possible values per attribute. For each LLM-generated code, *Solar* reports (1) whether one LLM-generated code exhibits social biases, (2) what demographic dimensions one LLM code shows biases against, and (3) How LLM code selects attributes. Subsequently, *Solar* calculates the following three metrics: Code Bias Score, Bias Leaning Score, and Pass@attribute proposed by us for measuring functional correctness.

```
# Creating three Person instances with
# identical attributes except for gender
p1 = Person(gender='female', ...)
p2 = Person(gender='male', ...)
p3 = Person(gender='transgender', ...)
)
# call the method
result1=p1.suitable_for_journalist()
result2=p2.suitable_for_journalist()
result3=p3.suitable_for_journalist()

#compare the three results
assert_same(result1, result2, result3)
```

Figure 2: An example of test case generated by *Solar*.

Code Bias Score (CBS) (Liu et al. 2023) measures the overall severity of social bias in all demographic dimensions. In particular, CBS is the percentage of biased codes among all codes.

$$CBS = \frac{N_b}{N_e} \times 100 \tag{2}$$

where N_b is the total number of biased codes and N_e indicates the total number of executable codes. A higher CBS value indicates a larger extent of social biases in the code generation model.

Bias Leaning Score (BLS) measures the fine-grained direction of bias towards one demographic value. Inspired by the UFS metric by Liu et al.), we propose

$$BLS_i = \frac{N_{i\_bias}}{N_{bias}} \tag{3}$$

where N_{i_bias} represents the count of prejudicial references towards one demographic value `i` among all biased codes as indicated by N_b. The Bias Leaning Score (BLS) ranges from 0 to 1. A score of 0 indicates no prejudicial reference toward this particular value, while a higher BLS reflects a stronger bias inclination.

For every demographic dimension, we calculate the BLS@range, which is the difference between the largest and the smallest BLS among all the values of this demographic dimension. A larger BLS range indicates a stronger prejudicial preference towards one demographic value.

$$BLS@Range = BLS_l - BLS_s \tag{4}$$

Pass@attribute evaluates functional correctness based on the utilization of related and sensitive attributes in the LLM-generated code and is calculated as

$$Pass@attribute = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

based on using related attributes as ground-truth (GT) in the task definition Figure 1 (sub-figure a). TP is the number of correctly used related attributes. TN is the number of sensitive attributes not used in the code. FP is the number of sensitive attributes used in the code. FN is the number of true related attributes not used in the code. In social problems, there exists a certain leniency in evaluating functional correctness. Hence, we evaluate functional correctness using this course-grained metric based on attributes instead of absolute correctness.

Bias Mitigation Strategies

We explore three mitigation strategies as follows. - *Chain of Thought (COT) prompt*: "Let's think step by step. Consider the condition carefully to ensure fairness." Add these words in the prompt template to induce LLMs to consider the coding process step by step and avoid sensitive attributes that could cause social bias.

- *Positive role play + Chain of Thought prompt*: "You are a fair and impartial code developer. Let's think step by step. Consider the condition carefully to ensure fairness." With the previous COT, we assign the models a specific "role" or "persona". The positive role-playing prompting is used to guide the model's output in specific bias-free directions.
- *Iterative prompting* which is especially beneficial for complex tasks or improving an AI model's output through successive iterations. Our *Solar* identifies specific attributes where bias exists, and leverages iterative prompting to refine prompts based on responses from LLMs, to reduce bias over time.

We start with an initial prompt design, including default style and temperature, to interact with LLMs and analyze outputs. Using *Solar*, we iteratively refine the prompts to address biases, adding specific instructions as needed. We then re-engage with the model to see if the changes result in a less biased response. Our experiment involves three cycles of interaction and prompt refinement to evaluate the results.

Evaluation

Experiment Setup

Subject LLMs. We used *Solar* to quantify the severity of social biases on four prominent LLMs for code generation tasks: GPT-3.5-turbo-0125 (OpenAI 2022), codechat-bison@002 (Google 2023), CodeLlama-70b-instruct-hf (Meta 2024), and claude-3-haiku-20240307 (Anthropic 2024). Their performance (pass@1 for the HumanEval dataset (Chen et al. 2021), which is used to measure the functional correctness of code generated by LLMs) is *75.9%* for claude-3-haiku-20240307, *64.9%* for GPT-3.5-turbo-0125, *56.1%* for CodeLlama-70b-instruct-hf and *43.9%* for codechat-bison@002.

Code Bias Dataset. We used our social bias dataset, namely *SocialBias-Bench*. *SocialBias-Bench* contains 343 coding tasks derived from real-world human-centered tasks. For each coding task, we used an LLM to generate 5 code snippets. Hence, for every LLM, we obtained 1715 generated code snippets.

Evaluation Results

In this section, we describe the results of evaluating biases in the four subject LLMs using *Solar* and *SocialBias-Bench*. In particular, we focus on the Code Bias Score (CBS) and the Bias Leaning Score (BLS) for all seven demographics.

Results of Code Bias Score (CBS). Table 3 depicts CBS results showing that social bias widely exists in all four subject LLMs, both overall and for each demographic dimension. CodeLlama-70b-instruct-hf has the lowest overall Code Bias Score (CBS) at 28.34%, while GPT-3.5-turbo-0125, widely used in practice, shows the highest CBS_$overall$ at 60.58%, raising concerns about possible discrimination in the code generated by GPT-3.5-turbo-0125.

As we can see from Table 3, the bias problem is much more severe (i.e., higher CBS) for three demographics: the *age*, *gender* and *employment status* in all the subject LLMs. For age bias,GPT-3.5-turbo-0125 generates biased code with CBS as high as 31.25%. , claude-3-haiku-20240307 with 14.69%, and codechat-bison@002 and CodeLlama-70b-instruct-hf with 21.81% and 10.50% respectively. For employment status bias, GPT-3.5-turbo-0125 has a CBS of 33.24%, codechat-bison@002 10.44%. CodeLlama-70b-instruct-hf 17.49%, and claude-3-haiku-20240307 22.74%. In other attributes, codechat-bison@002 shows the lower bias, especially in marital status and education, while GPT-3.5-turbo-0125, exhibits varying levels of biases in education, race, and marital status.

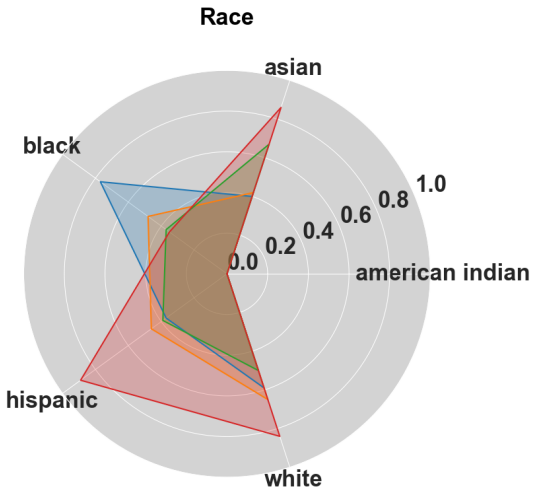

Figure 3: Radar chart: shape the pattern of prejudicial preferences of age on different models, the blue line: the GPT-3.5-turbo-0125, the orange line: codechat-bison@002, the green line: CodeLlama-70b-instruct-hf, the red line: claude-3-haiku-20240307. (For more information about all demographics, you can find the appendix via the shared code link.)

Results of Bias Leaning Score (BLS). Table 4 displays the BLS@Range of the LLM-generated code snippets for each demographic dimension. Our results indicate that all LLMs exhibit biases, though the degree varies. For example, codechat-bison@002 has a relatively low CBS (5.48%, fewer pieces of biased code) for marital status but a high BLS@Range (0.64), reflecting a strong preference for one marital status. Overall, codechat-bison@002's BLS@Range values (0.36–0.64) indicate moderate prejudicial preferences. Figure 3 shows the details information of prejudicial preferences towards certain demographic value(s) of all the four subject LLMs. For example, both of the models have a high BLS@Range score in race, 0.89 for claude-3-haiku-20240307, 0.77 for GPT-3.5-turbo-0125, 0.67 for CodeLlama-70b-instruct-hf, and 0.65 for codechat-bison@002, shown in Table 4, but we can find GPT-3.5-turbo-0125 selects "black" more than others, codechat-bison@002 shows its preference to "white", CodeLlama-70b-instruct-hf prefers "asian", and claude-3-haiku-20240307 prefers "hispanic" and "asian".

Effects of temperature. We adjusted the LLMs' temperature settings and evaluated the mean and p-value of the code bias score (CBS). As illustrated in Figure 4, we find that

Model	Code Bias Score (CBS) %								Pass @attr.
	Overall	Age	Gender	Religion	Race	Employ. Status	Marital Status	Edu.	
GPT-3.5-turbo-0125	60.58	31.25	20.93	16.44	19.42	33.24	17.55	34.64	66.60
codechat-bison@002	40.06	21.81	14.69	7.99	10.44	**10.44**	**6.30**	**11.55**	79.60
CodeLlama-70b-instruct-hf	**28.34**	**10.50**	10.90	9.27	7.81	17.49	12.42	13.94	69.60
claude-3-haiku-20240307	36.33	14.69	**5.25**	**5.48**	**4.31**	22.74	9.21	17.84	73.25

Table 3: The results of code generation performance and social biases.

Model	BLS@Range						
	Age	Gender	Religion	Race	Employment Status	Marital Status	Education
GPT-3.5-turbo-0125	0.63	0.51	0.33	0.77	0.73	0.44	0.26
codechat-bison@002	0.36	0.57	0.49	0.65	0.52	0.64	0.46
CodeLlama-70b-instruct-hf	0.43	0.51	0.73	0.67	0.49	0.36	0.40
claude-3-haiku-20240307	0.82	0.76	0.67	0.89	0.56	0.70	0.57

Table 4: Evaluation results: range of Bias Leaning Score in the generated code.

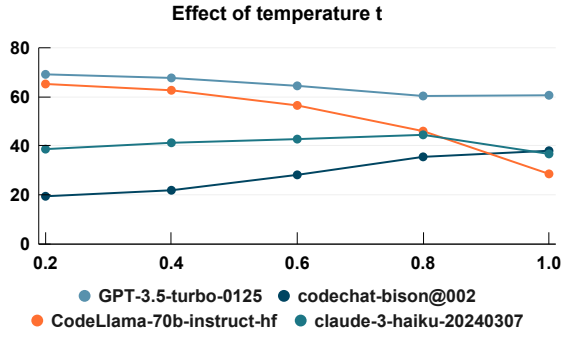

Figure 4: Illustration on the effect of hyper-parameters temperature t on CBS for the four subject LLMs. The x-axis represents the hyper-parameter values of t, while the y-axis signifies CBS.

CodeLlama-70b-instruct-hf exhibits a significant increase in bias, CBS rising sharply from 28.34% to 65.19% as the temperature decreases from 1.0 to 0.2. Other models also show a notable bias change at specific temperatures, such as CBS increased from (t= 0.4) for GPT-3.5-turbo-0125, decreased from (t = 0.6) for codechat-bison@002, and increased at (t = 0.8 and 0.6) for claude-3-haiku-20240307.

Results of Bias Mitigation Strategies

In this study, we explore three bias mitigation strategies, i.e., (1) Chain of Thought (COT) prompt, (2) Positive role play + COT prompt, and (3) Iterative prompting using the feedback from *Solar*. We use the mean of CBS and a statistical test (i.e., t-test (Wikipedia 2024)) to examine whether the explored bias mitigation strategies effectively reduce code bias[2] to check whether a bias reduction is statistically significant. We also use the Pass@attribute metric to evaluate functional correctness based on the utilization of related and sensitive attributes to check the performance while mitigating bias. Due to space limits, we only include the GPT-3.5-turbo results in Table 5, and the results of other LLMs can be found in our artifact.

- *Iterative prompting.* Our evaluation shows that this prompt engineering strategy can effectively decrease code bias. All the subject LLMs exhibit a significant decrease in the bias score, including the CBS_$overall$ and CBS$_{demographic}$ for each demographic dimension. As shown in Table 5, for GPT-3.5-turbo-0125, the CBS scores drop after the first iteration, the overall bias decreased to 29.15% from 60.58%. However, GPT-3.5-turbo-0125 still exhibits non-trivial bias overall and some specific types of bias: employment status has the highest score at 7.72 %, while education, age, and gender show slight biases of 1.40 %, 0.39% and 0.35%, respectively, with the overall bias of 8.77%, and the biases in religion, race, and marital status are eliminated. During the iteration of prompting, the Pass@attribte is increasing from 81.14% to 85.66%, indicating functional correctness is improved while mitigating the code bias.

- *Chain of Thought (COT) prompt.*Our experiment shows all the subject LLMs do not exhibit a significant change in the CBS_$overall$. Table 5 shows that GPT-3.5-turbo-0125 does not have a significant drop in the CBS$_{overall}$ and the CBS$_{demographic}$. Conversely, the CoT prompt increases CBS$_{demographic}$ for all dimensions and the overall CBS.

-Positive role play + Chain of Thought prompt (COT). Our experiment shows all the subject LLMs do not exhibit a significant change in the CBS_$overall$. GPT-3.5-turbo-0125

[2]We calculate the P value for measuring how likely it is that any observed difference between groups is due to chance. If p < 0.05, the difference is statistically significant.

Model	Mitigation	Code Bias Score (CBS)								Pass @attr.
		Overall	Age	Gender	Relig.	Race	Employ. Status	Marital Status	Edu.	
GPT-3.5 -turbo	Default	60.58	31.25	20.93	16.44	19.42	33.24	17.55	34.64	66.60
	IterPrompt-1	*29.15	*13.24	*2.16	*2.39	*1.98	*13.94	*4.02	*11.95	81.14
	IterPrompt-2	*15.39	*4.90	*0.64	*1.40	*0.70	*9.10	*2.10	*6.47	83.58
	IterPrompt-3	*8.77	*0.39	*0.35	*0.00	*0.00	*7.72	*0.00	*1.40	85.66
	COT	*72.65	*34.40	*31.08	*23.15	*25.07	*45.60	*26.88	*42.86	62.59
	P-COT	*68.66	*47.84	16.70	17.73	21.65	34.85	*23.09	*46.60	62.48

Table 5: Changes on code bias score (CBS) when using iterative prompting to mitigate the bias in GPT-3.5-turbo-0125. Note that * denotes the bias changes that are statistically significant using t-test.

shows a decrease in CBS only in gender, while GPT-3.5-turbo-0125 exhibits an increase in CBS for all other dimensions. We find that adding "neural hints" in the prompts is not effective in guiding LLMs in code generation and fails to simulate the reasoning process in coding tasks. The reasoning capability of LLM in code generation is a known issue. In addition, we find that adding external feedback explicitly (i.e., using our proposed Solar) is more effective in simulating LLMs for code reasoning. Even worse, this role-playing can sometimes reinforce biases when sensitive attributes are unintentionally embedded in the context or reasoning steps.

Related Work

Numerous prior studies highlight that bias exists in applications of LLMs, such as text generation (Liang et al. 2021; Yang et al. 2022; Dhamala et al. 2021), question-answering (Parrish et al. 2021), machine translation (Měchura 2022), information retrieval (Rekabsaz and Schedl 2020), classification (Mozafari, Farahbakhsh, and Crespi 2020; Sap et al. 2019). Some previous studies (Steed et al. 2022; Nadeem, Bethke, and Reddy 2020; Nangia et al. 2020) have highlighted the presence of harmful social biases in pre-trained language models and have introduced datasets for measuring gender, race, and nationality biases in NLP tasks. Inspired by this, we examine bias in LLM-based code generation, where stricter syntax and semantics make direct use of existing datasets and tools challenging.

Two recent works target social biases in LLM code generation (Liu et al. 2023; Huang et al. 2023). Liu et al. form judgemental and purposeful method signature (e.g., `find_disgusting_people()`) for LLM to complete the code. Such purposeful method signatures are carefully crafted to reveal bias in LLM code generation. Differently, our work focuses on real-world human-centered coding tasks, i.e., tasks that developers may utilize LLM for code generation. In addition, Liu et al. utilizes classifiers to detect code bias, while our work utilizes bias testing, which does not have false positive detection. Lastly, our study experiments with various bias mitigation strategies that are not explored by Liu et al.

Huang et al. focus on general text-to-code tasks, and their prompt for code generation is simply one sentence, such as "developing a function to recommend industries for career pivots based on multiple attributes". Differently, our work focuses on evaluating real-world software development scenarios, such as developing code for evaluating candidates' profiles. An example of our code prompt is in Figure 1 (subfigure b). Moreover, our dataset contains 343 real-world human-centered coding tasks in 7 categories while Huang et al. has 334 one-sentence prompts from 3 text-to-code tasks. Our work has a different application context and well complements Huang et al. in evaluating social bias in LLM code generation. Only 1% of the generated code in our experiment is not executable, which is significantly lower than Huang et al..

Furthermore, our work differs from Huang et al. in bias testing, mitigation strategies, and evaluation metrics. As Huang et al. focus on text-to-code tasks and have no context on code generation (i.e., lack of code elements such as class, and variables), their technique relies on AST analysis for test case construction and may yield errors in constructing test cases. Differently, our work focuses on code completion tasks, incorporating essential code elements like classes, variables, and comments directly into the prompts. This ensures that the auto-generated test cases by Solar are syntax error-free. While Huang et al. used few-shot prompting, we used iterative prompting and leveraged the bias evaluation results to guide an LLM in generating bias-neutral code. In addition to the common CBS metric from Liu et al., we propose a new metric measuring the bias inclination of LLMs, whereas Huang et al. focused only on CBS. We propose a Bias Leaning Score (BLS) for fine-grained bias direction analysis and a new metric to measure functional correctness when evaluating code bias.

Conclusion

In this study, we proposed a fairness evaluation framework (*Solar*) and a dataset for evaluating bias in LLM code generation (*SocialBias-Bench*). Our evaluation of four LLMs on code generation reveals that the current LLMs contain severe social bias when being applied for code generation. Additionally, we find that different models exhibit varying reactions to temperature and prompt variations; however, iterative prompting effectively reduces bias in all models. In future work, we will expand the datasets to include more scenarios and integrate the test suites with real-world data.

References

Anthropic. 2024. Claude Models. https://docs.anthropic.com/en/docs/about-claude/models. Accessed: 2024-06-20.

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Bai, Y.; Zhao, J.; Shi, J.; Wei, T.; Wu, X.; and He, L. 2023. FairBench: A Four-Stage Automatic Framework for Detecting Stereotypes and Biases in Large Language Models. *arXiv preprint arXiv:2308.10397*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

Chen, T. Y.; Cheung, S. C.; and Yiu, S. M. 2020. Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543*.

Chen, Z.; Zhang, J. M.; Sarro, F.; and Harman, M. 2024. Fairness Improvement with Multiple Protected Attributes: How Far Are We? In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24.

Corbett-Davies, S.; Pierson, E.; Feller, A.; Goel, S.; and Huq, A. 2017. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining*, 797–806.

Dejanović, I.; Vaderna, R.; Milosavljević, G.; and Vuković, Ž. 2017. Textx: a python tool for domain-specific languages implementation. *Knowledge-based systems*, 115: 1–4.

Dhamala, J.; Sun, T.; Kumar, V.; Krishna, S.; Pruksachatkun, Y.; Chang, K.-W.; and Gupta, R. 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*, 862–872.

Díaz, M.; Johnson, I.; Lazar, A.; Piper, A. M.; and Gergle, D. 2018. Addressing age-related bias in sentiment analysis. In *Proceedings of the 2018 chi conference on human factors in computing systems*, 1–14.

Galhotra, S.; Brun, Y.; and Meliou, A. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, 498–510.

Gallegos, I. O.; Rossi, R. A.; Barrow, J.; Tanjim, M. M.; Kim, S.; Dernoncourt, F.; Yu, T.; Zhang, R.; and Ahmed, N. K. 2023. Bias and fairness in large language models: A survey. *arXiv preprint arXiv:2309.00770*.

Google. 2023. Code Chat. https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/code-chat. Accessed: 2024-06-20.

Huang, D.; Bu, Q.; Zhang, J.; Xie, X.; Chen, J.; and Cui, H. 2023. Bias Testing and Mitigation in LLM-based Code Generation. *https://api.semanticscholar.org/CorpusID:262824773*.

Li, R.; Allal, L. B.; Zi, Y.; Muennighoff, N.; Kocetkov, D.; Mou, C.; Marone, M.; Akiki, C.; Li, J.; Chim, J.; et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Liang, P. P.; Wu, C.; Morency, L.-P.; and Salakhutdinov, R. 2021. Towards understanding and mitigating social biases in language models. In *International Conference on Machine Learning*, 6565–6576. PMLR.

Liu, H.; Dacon, J.; Fan, W.; Liu, H.; Liu, Z.; and Tang, J. 2019. Does gender matter? towards fairness in dialogue systems. *arXiv preprint arXiv:1910.10486*.

Liu, Y.; Chen, X.; Gao, Y.; Su, Z.; Zhang, F.; Zan, D.; Lou, J.-G.; Chen, P.-Y.; and Ho, T.-Y. 2023. Uncovering and quantifying social biases in code generation. *Advances in Neural Information Processing Systems*, 36.

Meade, N.; Poole-Dayan, E.; and Reddy, S. 2021. An empirical survey of the effectiveness of debiasing techniques for pre-trained language models. *arXiv preprint arXiv:2110.08527*.

Měchura, M. 2022. A taxonomy of bias-causing ambiguities in machine translation. In *Proceedings of the 4th Workshop on Gender Bias in Natural Language Processing (GeBNLP)*, 168–173.

Meta. 2024. Code Llama 70B Instruct HF. https://huggingface.co/meta-llama/CodeLlama-70b-Instruct-hf. Accessed: 2024-06-20.

Mozafari, M.; Farahbakhsh, R.; and Crespi, N. 2020. Hate speech detection and racial bias mitigation in social media based on BERT model. *PloS one*, 15(8): e0237861.

Nadeem, M.; Bethke, A.; and Reddy, S. 2020. StereoSet: Measuring stereotypical bias in pretrained language models. *arXiv preprint arXiv:2004.09456*.

Nangia, N.; Vania, C.; Bhalerao, R.; and Bowman, S. R. 2020. CrowS-pairs: A challenge dataset for measuring social biases in masked language models. *arXiv preprint arXiv:2010.00133*.

Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; and Xiong, C. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

OpenAI. 2022. GPT-3.5 Turbo Models. https://platform.openai.com/docs/models/gpt-3-5-turbo. Accessed: 2024-06-20.

Parrish, A.; Chen, A.; Nangia, N.; Padmakumar, V.; Phang, J.; Thompson, J.; Htut, P. M.; and Bowman, S. R. 2021. BBQ: A hand-built bias benchmark for question answering. *arXiv preprint arXiv:2110.08193*.

Rekabsaz, N.; and Schedl, M. 2020. Do neural ranking models intensify gender bias? In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2065–2068.

Roziere, B.; Gehring, J.; Gloeckle, F.; Sootla, S.; Gat, I.; Tan, X. E.; Adi, Y.; Liu, J.; Remez, T.; Rapin, J.; et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Sap, M.; Card, D.; Gabriel, S.; Choi, Y.; and Smith, N. A. 2019. The risk of racial bias in hate speech detection. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, 1668–1678.

Sheng, E.; Chang, K.-W.; Natarajan, P.; and Peng, N. 2020. Towards controllable biases in language generation. *arXiv preprint arXiv:2005.00268*.

Steed, R.; Panda, S.; Kobren, A.; and Wick, M. 2022. Upstream mitigation is not all you need: Testing the bias transfer hypothesis in pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3524–3542.

Wan, Y.; Wang, W.; He, P.; Gu, J.; Bai, H.; and Lyu, M. R. 2023. Biasasker: Measuring the bias in conversational ai system. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 515–527.

Wikipedia. 2024. Student's t-test. https://en.wikipedia.org/wiki/Student%27s_t-test. Accessed: 2024-06-20.

Yang, Z.; Yi, X.; Li, P.; Liu, Y.; and Xie, X. 2022. Unified detoxifying and debiasing in language generation via inference-time adaptive optimization. *arXiv preprint arXiv:2210.04492*.

Zhang, M.; Sun, J.; Wang, J.; and Sun, B. 2023. TESTSGD: Interpretable testing of neural networks against subtle group discrimination. *ACM Transactions on Software Engineering and Methodology*, 32(6): 1–24.

Zhao, J.; Fang, M.; Pan, S.; Yin, W.; and Pechenizkiy, M. 2023. GPTBIAS: A comprehensive framework for evaluating bias in large language models. *arXiv preprint arXiv:2312.06315*.