CONTENTS

APPENDIX

We open source our code at https://github.com/GeMoLLM/GeMO.

A SAMPLING PARAMETERS

1. Temperature: Concretely, the temperature parameter T determines the "dispersion" of the probability distribution over the next token. Mathematically, the probability of a token w being generated is given by the softmax function: $P(w|x) = \frac{\exp(s(w|x)/T)}{\sum_{w'} \exp(s(w'|x)/T)}$ where $s(w|x)$ is the unnormalized log-probability of the token w given the context x. Increasing the temperature leads to a more flat probability distribution and increases the likelihood of sampling from less probable tokens, resulting in more diverse generations. This can also be understood from the perspective of entropy of the next token, $H(W) = -\sum_w P(w|x) \log P(w|x)$, where it can be seen that the increase of entropy directly follows.

2. Top-p: The top-p parameter Holtzman et al. (2019) ($p \in (0, 1]$) controls the randomness of the generations by limiting the range of tokens considered. Specifically, it considers the smallest subset (consisting of the top probability tokens) whose cumulative probability exceeds the threshold p. A smaller p encourages the model to sample from a more focused set of likely tokens, while a larger p allows sampling from a broader range and thus increases randomness ($p = 1$ basically means no restriction on the vocabulary).

Different platforms have different default values for T and top-p. GPT-3.5-turbo-instruct (0914) web version adopts $T = 0.8$ Community (2023). The default values in OpenAI APIs are $T = 1.0$ and $p = 1.0$ OpenAI (2024).

3. Generation Length: This parameter (`max_new_tokens`) dictates at most how many tokens the model should generate before it stops. We used 500 for book review generation and 2048 for code generation.

B Additional Details: Book Reviews

B.1 Construction of the Source Dataset

To ensure we picked popular books that the LLMs know about, we filtered the books according to the number of reviews they have. Constraining the number of reviews to be between 1,000 and 2,500, we obtained 750 books. We further filtered out books with non-English titles; we conducted this filtering because some downstream LLMs (e.g., sentiment classifiers) are not multilingual, and as a result can not analyze the generated non-English reviews. To ensure high quality of the reviews, we further filtered the review by length, constraining the length of each review to be between 300 and 700 words. After this step of filtering, we sampled 10 reviews per book. Eventually, we obtained a dataset of 742 books, where each book comes with 10 reviews.

B.2 Names of the Celebrities Used in Prompt 2

We prompted `GPT-4` to provide a list of celebrities suitable for writing diverse book reviews. The list of names are as follows: Trevor Noah, Janelle Monáe, Yuval Noah Harari, Serena Williams, Reshma Saujani, Neil deGrasse Tyson, Margaret Atwood, David Attenborough, Malala Yousafzai, Jordan Peele.

B.3 Text Processing for Analyzing Wording Choice

We first concatenated all documents and converted them to lowercase to standardize the text. We then expanded contractions (e.g., "don't" to "do not"). We then tokenize the text with `word_tokenize()` from the NLTK library Bird & Loper (2004), and removed the punctuation. We continued by filtering out non-alphabetic tokens to focus solely on words. Lastly, we lemmatized tokens via `WordNetLemmatizer()` to their base forms, aiding in consistent frequency analysis. These steps are essential for minimizing textual noise and ensuring the reliability of our word frequency assessments.

C Additional Results: Book Reviews

C.1 Filtered-out Reviews

We established a perplexity threshold of 20 to filter out low-quality reviews. To validate our choice, we randomly sampled reviews with perplexity scores at different intervals and manually inspected them. For clarity, instead of presenting the entire review text, we selectively extracted chunks that exemplified the low-quality nature of the sampled reviews.

Perplexity in $(20, 25]$: This page-turning, and fast-paced gripping story makes excellent use both tropes and novel ideas by employing both like, for instance, a dystopian setting which uses an exaggeratory ruminating setting so typically found in series belonging to this genre, only that here the author‘s unique skill at weaving such plot-lines together in a novel that feels fresh more like Harry Potter but with an 800word length page each chapter instead of JUST 200word paragraph at the end of an excrutiatingly slow chapter.

Perplexity in $(25, 30]$: That and artist Adrian Alphona bringing the world-sprawling amazing action and gorgeous characters from beautifully rendered backgrounds in color by vivid and dazzling color is what makes the graphic novel in itself even more delightful. With each color panel bringing an explosion of color onto every page no panel is left the same no character or page lacking the same amount of energy vivid and gripping from the moment I turned on. the illustrations are bright beautiful detailed it truly does it live upto its tag line and then some a hero like no other a book like no other definitely worth the dive.

Perplexity in $(30, 35]$: One would wonder if she lived some of this amazing journey to unhidden holes in the ground or lived the stories and characters by the holes they created one will see how amazingly her fantastic works has a way t connect the reader to a place which feels so real and fascinating! Holes by Loues Erdrich has a beautifully constructed a magical and a world which transports and takes readers like on a journey thrifting us so many magical and real places which we might not have an opportunity if it wasn't for the eyes and minds eye of Louise Erdrech .

Perplexity in $(35, 40]$: It made me reflect for a deeper appreciation on both the power of memory in creating some of literature and art's most significant works or cultural touchstone masterpieces such as those listed that our lead female artist was known to adore and how that and the themes that author – a most wonderful story teller by the way, by way of characters that i wanted or did want to spend more time with and learn the most in depths exploration of, and not all can reach it. And so a deeper appreciation for the art of storytelling from great narratives by great narrators? The Little Paris Bookp shop does achieve an astounding success in that regard, I am thrilled I'd like to shout it out to any and All in ear shot about it!

C.2 Generation Results at Higher Randomness

We experimented with even higher randomness at $T = 1.5$ motivated by the observation that increasing the randomness does help to increase the diversity. However, as we show in Fig. 6, even at high randomness, there is still a huge gap between the source and the generations.

Moreover, we notice a significant degradation of the generation quality as a result of the increased randomness. We present in Table 1 the average number of valid generations for two models under various sampling parameters. The table shows that the valid number of generations rapidly drops as the randomness increases, particularly at $T = 1.5$; the implication is that such a high randomness setting basically cannot be adopted for practical use.

C.3 Topic Shifts

We present in Fig. 7 the results of the unconditional topic distribution. Across all three settings for comparison, we observe highly similar distribution shift signified by the over- and under-representation of certain topic groups. Concretely, the topic group 15 with the keywords "grace, rob, nick, anna, house, discovers, realizes, killed, confronts, killer" are under-represented, potentially because certain words like "rob" and "kill" are *eliminated* from the output as a result of RLHF. On the other hand, the topic group 333 with the keywords "handmaid, novels, novel, writers, tale, book, books, literary, novellas, synopsis" and the topic group 666 with the keywords "nonfiction, bestseller, novelist, autobiography, novels, memoir, author, memoirs, paperback, novel" are over-represented. The over-representation of topic group 333 in `Vicuna-13b` is much more severe than `Llama-2-chat`, as can be seen from the 3rd subfigure; this could be because a higher exposure to relevant materials during its fine-tuning.

C.4 Pre-trained Model

The aligned model is obtained from performing supervised fine-tuning followed by RLHF alignment tuning on the basis of the pre-trained model Touvron et al. (2023). We compare the performance of the aligned model llama-2-13b-chat with the pre-trained model llama-2-13b and present the results in Fig. 8.

From the 1st and 2nd subfigure, we observe that the pre-trained model is much more diverse than the aligned model; the pre-trained model is even very close to the source data (see the dark purple result tagged as $T = 1.2, p = 1.0$ (c)).

The 3rd subfigure delivers two messages. First of all, there still exists divergence from the pre-trained model and the source in terms of the covered topics. We attribute this divergence we observe between the pre-trained model and the source data to the difference between the source data we use (the Goodreads dataset) and the ground-truth training data (a much broader corpus which we have no information about). We regard the Goodreads dataset as a proxy of the ground-truth, but intrinsically Goodreads is smaller and does not fully accurately represent the groundtruth training distribution, especially in nuanced attributes like the unconditional topic distribution measured on all samples. Second, the direction and trend of changes in the aligned model is not completely the

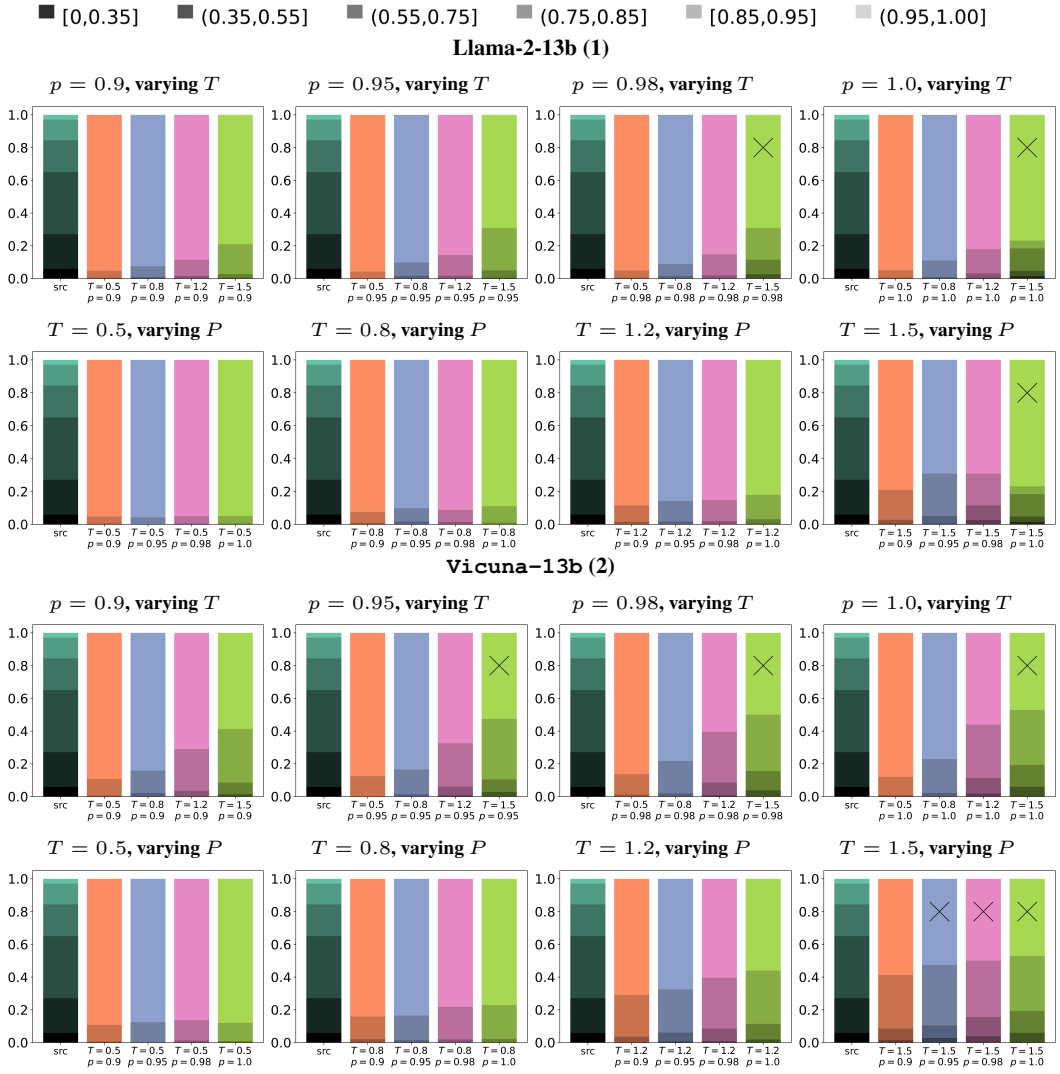


Figure 6: **Mean sentiment scores** across different models (`Llama-2-chat`, `Vicuna-13b`) and prompts ((1) and (2)), and varying sampling parameters (temperature T and top-p). We do observe an increase of diversity along the increase of generation randomness (i.e., increasing T and p), but the gap remains high between the source and even the highest generation randomness setting. Moreover, we put a black cross $\times$ on the top of the setting with a concerning low number of valid number of generations (see Table 1 for a concrete explanation); they mostly only occur for the few highest randomness settings.

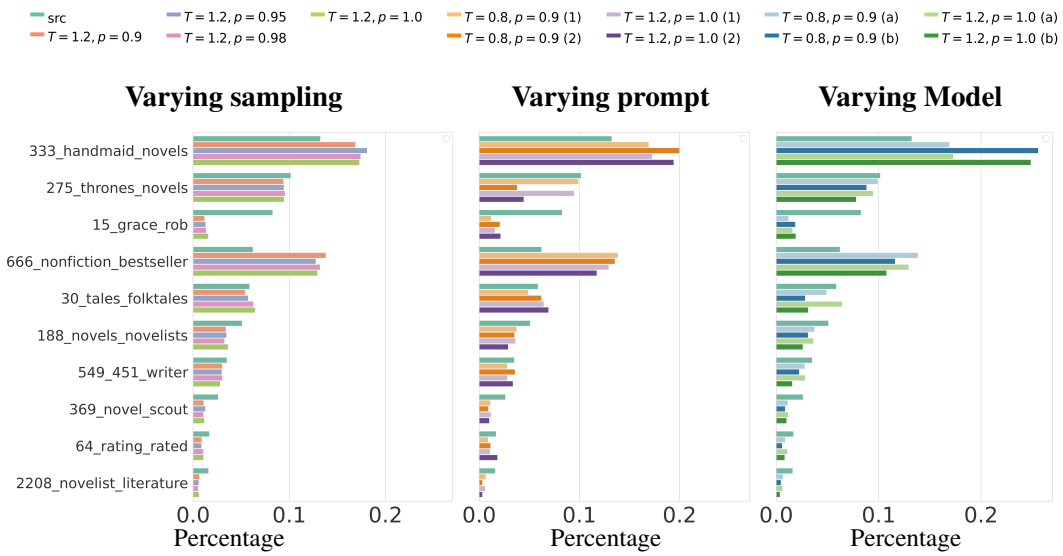


Figure 7: **Grouped bar charts of the unconditional distribution of the topic** (i.e., the distribution of the topic over all the samples). **(Left)**: generated by `Llama-2-chat` at different generation kwargs, where we fixed $T = 1.2$ and varied $p \in \{0.90, 0.95, 0.98, 1.00\}$. The prompt used is (1) as introduced in Section 5.1. **(Middle)**: generated by `Llama-2-chat` under different prompts (1) and (2). **(Right)**: generated by two models (a) `Llama-2-chat` and (b) `Vicuna-13b`, using the prompt (1). Across all three subfigures, we observe highly similar distribution shift signified by the over- and under-representation of certain topics.

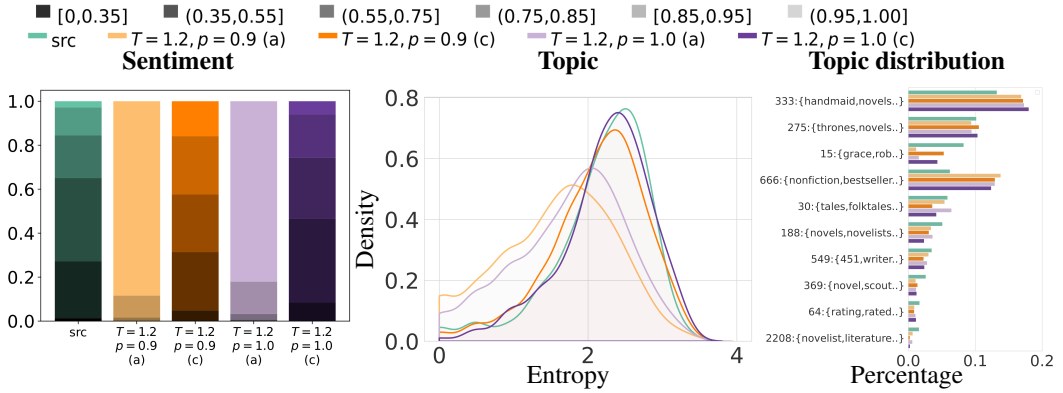


Figure 8: **Comparison between (a) the *aligned* model `Llama-2-chat` and (c) the *pre-trained* model `Llama-2`**, under prompt (1), temperature $T = 1.2$, and varying top-p parameters. We present a subset of the results (sentiment, topic, and topic distribution), corresponding to Figure 7. The figures show that the pre-trained model enjoys much better diversity than the aligned model, and is much closer to the source.

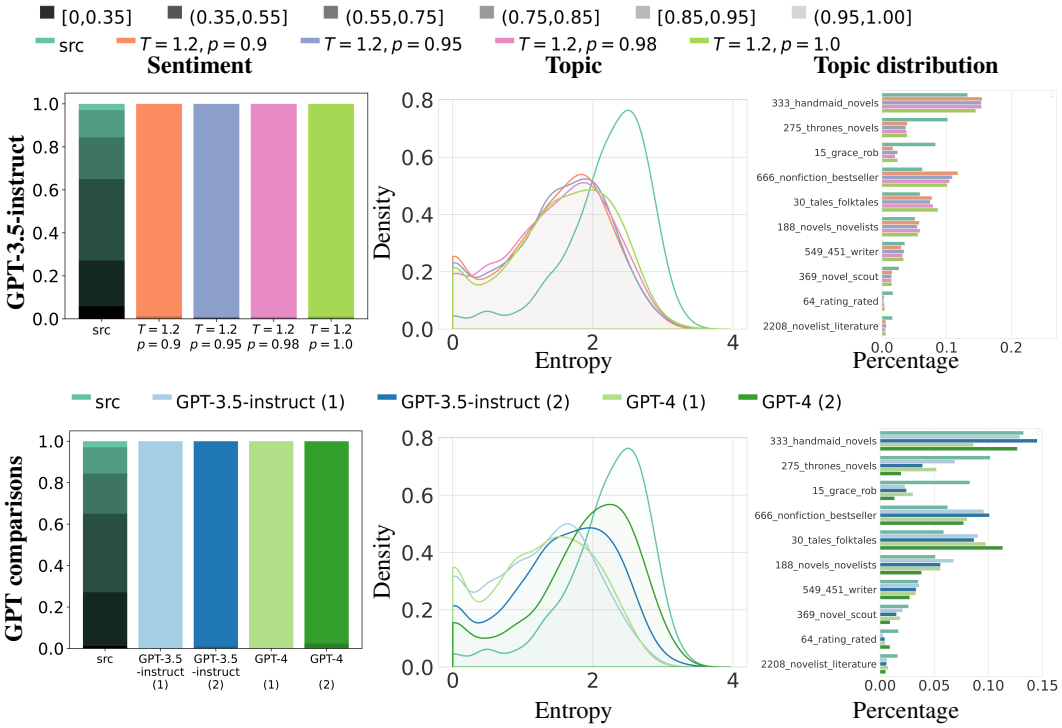


Figure 9: **Plots for OpenAI models**. **(Upper row)**: `GPT-3.5` under prompt (1), $T = 1.2$, and varing p. **(Lower row)**: comparing `GPT-3.5` and `GPT-4` at $T = 1.2, p = 1.0$ and the two propmts (1) and (2). We present a subset of the results (sentiment, topic, and topic distribution), corresponding to Figure 7. The figures show that both `GPT-3.5` and `GPT-4` display severe GeMo, which cannot be effectively mitigated via varying the sampling parameters or using a diversity-inducing prompt.

same as the aligned models, e.g., the topic group 15 containing "rob" and "killer" is not inhibited as much as in the aligned models, potentially due to RLHF.

C.5 Results of OpenAI models

We repeat the same set of experiments on the proprietary OpenAI models GPT-3.5-turbo-instruct (0914) and GPT-4-turbo (0125). We present the results in Fig. 9.

The sentiment results (in column 1) suggest that both models are overwhelmingly and invariably positive in their reviews, across various sampling parameters and prompts.

The results of the conditional topic distribution (in column 2) reveal similar conclusions—there is a significant deviation between the generated reviews and the source reviews across varying sampling parameters (see upper low) and for both GPT-3.5-instruct and GPT-4 (see lower row). Nevertheless, using a diversity-inducing prompt, i.e., prompt (2) does lead to increased diversity (see lower row).

The results of the unconditional topic distribution (in column 3) also demonstrate the distribution shift, though in a slightly different way than observed for llama family models (see presented in Appendix C.3). Concretely, the topic group 333 does not experience a significantly over-representation, while the topic group 30 does. As discussed previously, we attribute the difference in the nuanced topic distribution mainly to the difference in the training data.

C.6 Mitigation via Temperature Decay

We describe the decaying temperature scheme. Concretely, we choose the starting temperature $T = 10.0$ and follow a linear schedule for temperature decay, over the course of 50 timesteps (i.e.,

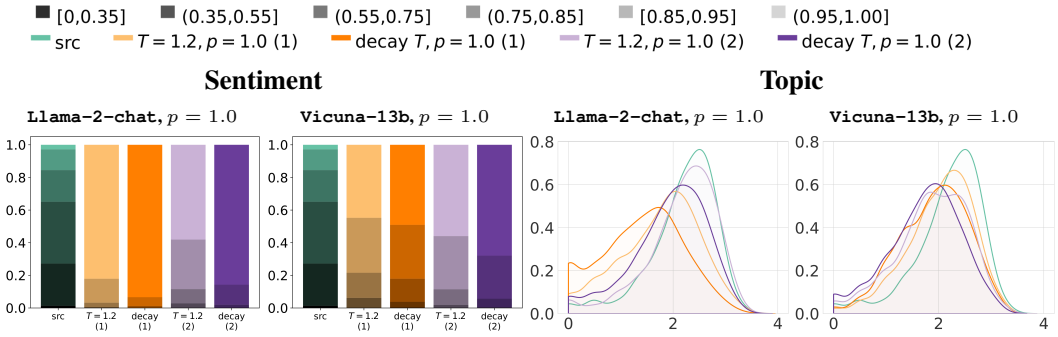


Figure 10: **Effect of temperature decay during decoding time** for two models `Llama-2-chat` and `Vicuna-13b`, with top-p parameter 1.0 and two prompts (1) and (2). We compare the fixed temperature ($T = 1.2$) with the decaying temperature (decay T) and we present the results for sentiment and topic. The figures reveal the ineffectiveness of the temperature decay method.

from the 1-st output token to the 50-th output token), with an ending temperature $T = 1.2$. The method is inspired by Carlini et al. Carlini et al. (2021) which reported this scheme as one sampling method that induce diversity and high quality output[1].

We present the results in Fig. 10. For both the sentiment and the topic, we see that the fixed temperature scheme achieves a higher diversity compared to the decaying temperature scheme.

C.7 Evaluation on Additional Prompts

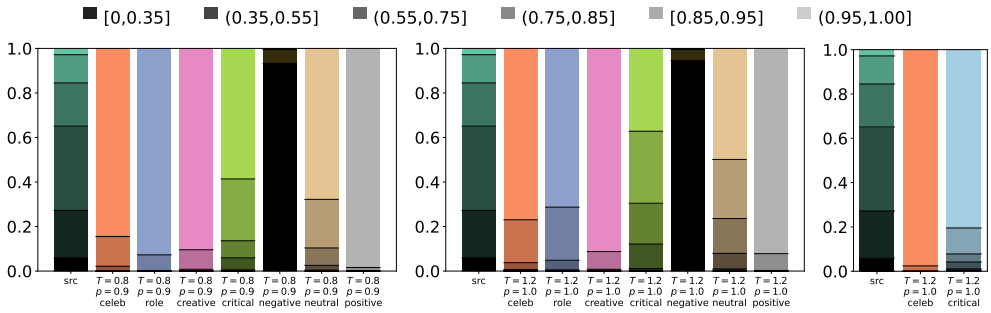


Figure 11: **Stacked barplots for the mean sentiment scores.** Left and middle for Llama-2-13b-chat and right for GPT-4. The label "celeb" corresponds to Prompt (2) in the main submission; the prompts for other labels can be found below. The all black bar for the label "negative" means that all the generations for this prompt achieve extremely low scores (refer to the hue legend).

Aside from prompts (1) and (2) introduced in the main paper in § 5.1, we additionally experimented with three groups of prompts for comprehensiveness.

Group 1 [specifying detailed roles]:

- Prompt (role): "`Write a book review for the book titled {title} as if you are a {role}:`", where role $\in$ {"teenage fantasy enthusiast", "critical literature professor", "romance novel lover", "tech-savvy sci-fi geek", "history buff", "casual weekend reader", "book club moderator", "non-fiction aficionado", "poetry appreciator", "mystery thriller addict"}
- Prompt (creative): "`Write a book review for the book titled {title}, from the viewpoint of different personas, such as 'aspiring`

[1] We refer to `https://github.com/shreyansh26/Extracting-Training-Data-from-Large-Langauge-Models/blob/main/extraction_temperature_decay.py` for an implementation of the temperature decay sampling scheme for HuggingFace models.

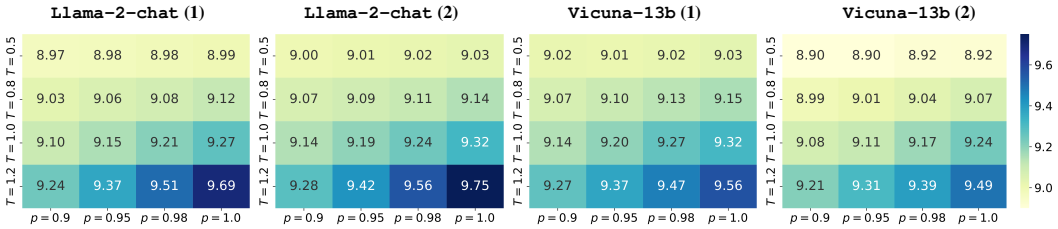


Figure 12: **Heatmap of the entropy of the (unconditional) distribution of the *word choice* in model-generated reviews**. We present results of two models (`Llama-2-chat` and `Vicuna-13b`) and two prompts ((1) and (2)). As a reference, the entropy of the source is 9.99, higher than all the entropy values of model generations.

`writer', 'history enthusiast', 'teenage sci-fi fan', or 'career-focused parent', etc. Be creative!"`

Group 2 [Steering attributes such as sentiment]:

- Prompt (negative): "`Write a negative review of the book titled {title}:`"
- Prompt (neutral): "`Write a neutral review of the book titled {title}:`"
- Prompt (positive): "`Write a positive review of the book titled {title}:`"

Group 3 [Other variations]:

- Prompt (critical): "`Write a critical review of the book titled {title}:`"

We conducted the additional experiments on a) `Llama-2-13b-chat` and b) `GPT-4`. For the former, we evaluated two sampling parameters ($T = 0.8, p = 0.9$ for relatively low randomness, and $T = 1.2, p = 1.0$ for high randomness); for `GPT-4`, we considered only the high randomness parameter and only the `prompt (critical)` due to the long runtime. For each combination of model, prompt, and sampling parameter, we generated $n = 10$ reviews.

We present the results in Figure 11 and discuss the takeaways below.

1. `Prompt (critical)` achieves the best diversity among all. However, there is still a significant gap from the `src`, which holds true for both models.
2. `Prompt (role)` achieves a similar level of diversity compared to `Prompt (person)`, showing that specifying a diverse set of *detailed roles* do not bring improvement over specifying a diverse set of *celebrities*. Giving the LLM extra freedom to explore using different roles (i.e., `Prompt (creative)`), however, leads to even worse diversity.
3. Using sentiment words (i.e., negative, neutral, positive) can indeed steer the sentiment of the generations very effectively. Yet we clarify that the model being a good instruction follower does not directly translate into it preserving diversity, without a good instruction giver. Moreover, as per our definition, we care about diversity of multiple attributes. Explicitly steering one attribute is insufficient for achieving diversity across numerous attributes. More generally speaking, the issue resides in—what we can control/steer is a limited set of knowns, but what we hope to gain is w.r.t. a more broader set of knowns and unknowns.

C.8 More Results

For the attribute *word choice*, we present the count of unique words in Table 2 and the entropy for the word distribution in Fig. 12, showing that model-generated reviews use a narrower vocabulary and is less diverse than the human-written reviews.

We present more results for various attributes and metrics in Fig. 13 and 14 where we vary several factors for detailed comparisons. Overall, the results suggest the prevalence and severity of generative monoculture, as well as the ineffectiveness of naive mitigations.

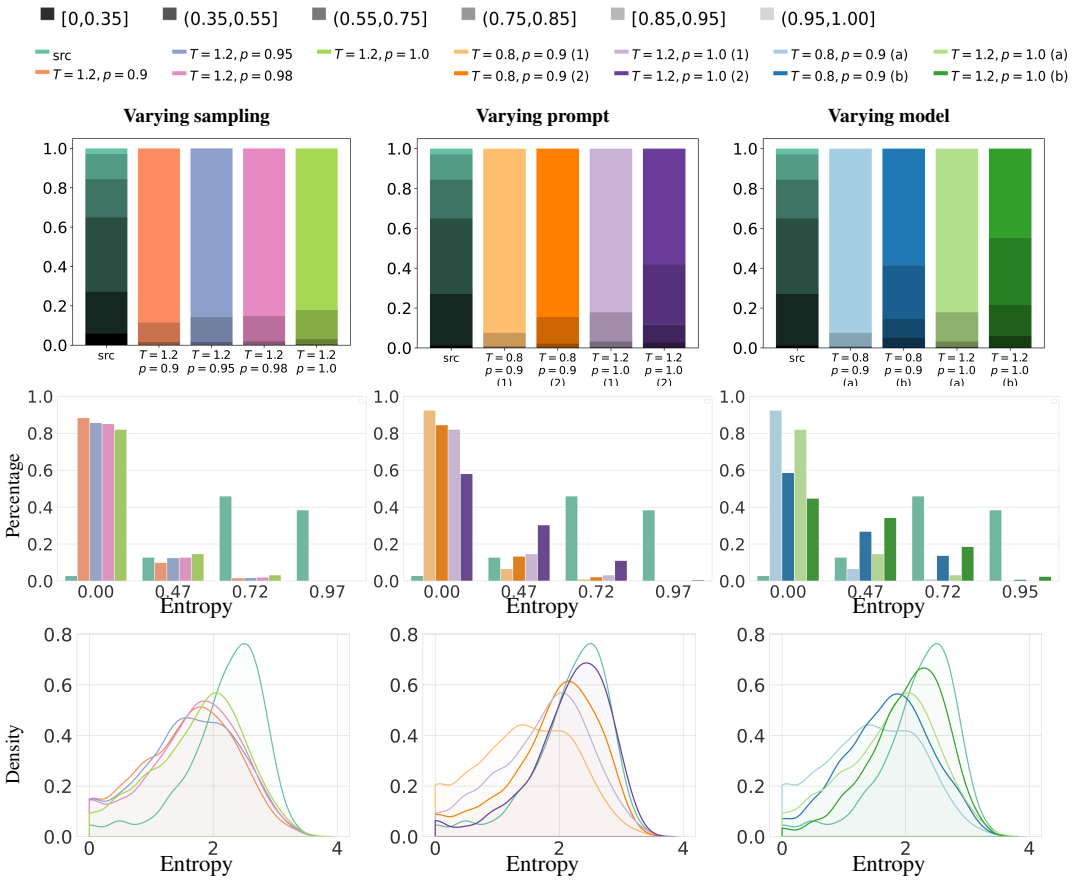


Figure 13: **(Row 1)**: Stacked bar charts of the mean sentiment scores. In each bar, darker hues (bottom) correspond to the lower scores while the lighter hues (upper) are the higher scores. See the legend for the detailed value range of each hue. **(Row 2)**: Histograms of the entropy of the sentiment scores. Each bar group corresponds to a range of entropy values in $[x, x+0.2]$ where x is the number under the bar group. **(Row 3)**: Kernel density estimate (KDE) plots of the entropy of the topic. **(Left)**: generated by `Llama-2-chat` at different generation kwargs, where we fixed $T = 1.2$ and varied $p \in \{0.90, 0.95, 0.98, 1.00\}$. More variations can be found in Figure 6 in Appendix C. The prompt used is (1); see later for the detail. **(Middle)**: generated by `Llama-2-chat` under prompts (1) and (2). **(Right)**: generated by two models (a) `Llama-2-chat` and (b) `Vicuna-13b`, using the prompt (1).

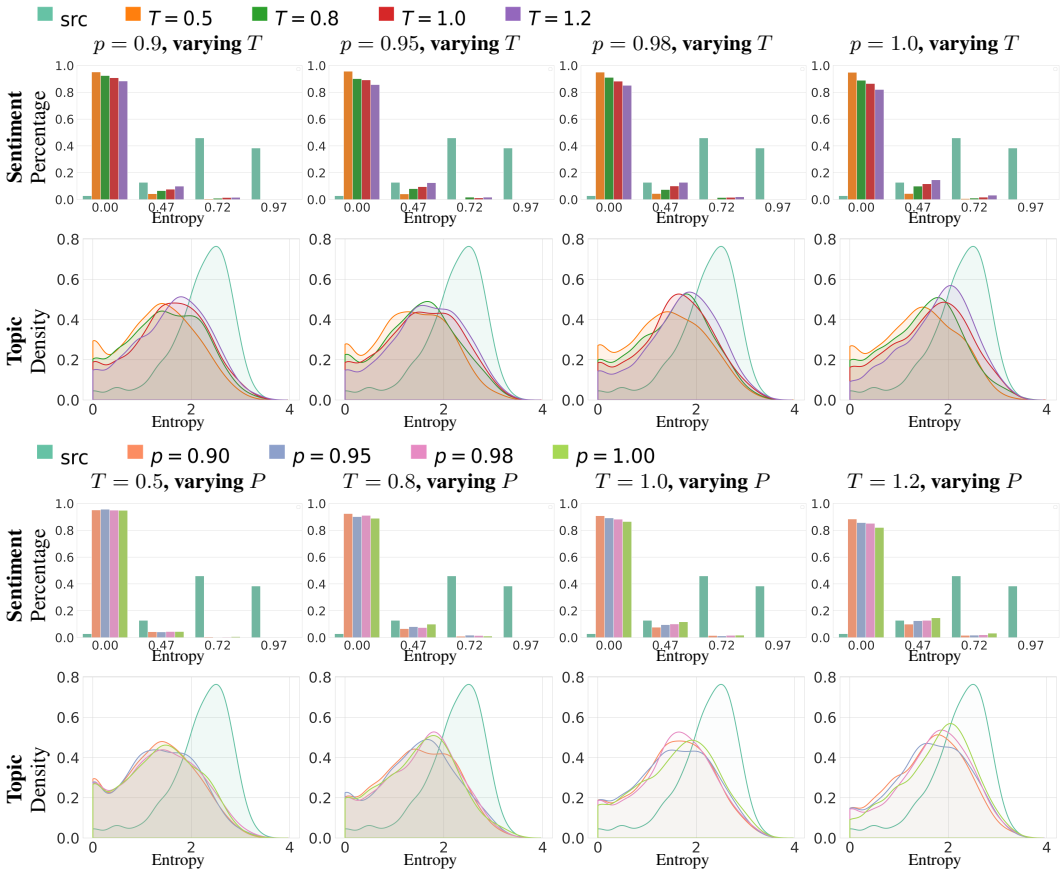


Figure 14: **(Rows 1-2)**: varying T under fixed p: (upper) entropy of the sentiment for the conditional distribution and (lower) Entropy of the topic for the conditional distribution. **(Rows 3-4)**: varying p under fixed T: (upper) entropy of the sentiment for the conditional distribution and (lower) entropy of the topic for the conditional distribution. Results are obtained via `Llama-2` under prompt (1).

Table 2: **Count of unique words** produced by different models (**left**: llama-2-13b-chat, **right**: GPT-3.5-instruct) under varying sampling parameters and prompt (1). As a reference, the count in the source dataset is 85,334.

	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 1.00$
$T = 0.5$	18,900	19,041	19,145	19,275
$T = 0.8$	20,020	20,516	20,935	21,688
$T = 1.0$	21,295	22,024	23,181	24,738
$T = 1.2$	23,509	25,742	28,532	33,908

	$p = 0.90$	$p = 0.95$	$p = 0.98$	$p = 1.00$
$T = 0.5$	15,782	15,794	15,778	15,841
$T = 0.8$	16,563	16,915	17,096	17,235
$T = 1.0$	17,044	17,543	17,960	18,674
$T = 1.2$	17,368	18,077	18,894	21,059

D Additional Details: Coding

D.1 Restriction to Level-A Problems

We limit our scope to Codeforces level-A problems only for the coding scenario, which are the easiest problems on the Codeforces competitive programming platform[2]. We note that even though these are the easiest problems on the platform, they still require non-trivial intellectual efforts to solve[3].

There are two main reasons we restrict to level-A problems:

1. Most LLMs perform best on level-A problems and much worse on more difficult ones. We started off evaluating a set of 81 problems ranging from difficulty A to G, and obtained an average accuracy of 50% on the 16 level-A problems, and less than 10% on others. Since we perform attribute extraction on the correct solutions only, a low accuracy means that to ensure reaching a minimum number of correct generations (20 in our experiments), a huge number of solutions need to be generated (e.g., over 200), which is prohibitively expensive in both time and money. Thus we settle with the level-A problems.
2. We manually verify the correctness of the attribute extraction (e.g., code summary, runtime complexity). Relatively simpler problems are easier to verify.

D.2 Correctness Testing: Autojudge with Testcases

We simulated an Autojudge using the test cases provided in the dataset. Concretely, for each problem, we obtain 10 test cases in the format of input and output[4]. We then measure each solution against the set of 10 test cases. We regard the solution that passes all 10 test cases as a *correct* solution.

D.3 Measuring Accuracy

We calculate accuracy as $n_i^{\text{correct}}/n_{\text{all}}$ per problem (where n_i^{correct} is the number of correct solutions and n_{all} is the number of all solutions). For the source, both n_i^{correct} and n_{all} are available in the CodeContests dataset Li et al. (2022). For the generations, we test the correctness of the solutions via autojudge (see Appendix D.2) and measure the accuracy as n_i^{correct}/k.

D.4 Measuring Runtime Efficiency

For runtime efficiency, we use the bash command `/usr/bin/time` Kerrisk (2023) to measure the elapsed real time (via %E) as well as the maximum resident set size of the process during its execution (via %M). Concretely, for each solution, we run it on all 10 test cases with our autojudge

[2] `https://codeforces.com/problemset`

[3] Interested readers can refer to this problem `https://codeforces.com/problemset/problem/1198/A` to get a sense of the difficulty in problem understanding and solving

[4] In Codeforces, each test case is often consisted of multiple small tests; see an example at `https://codeforces.com/problemset/problem/1406/A`. As explained in the **Input** section on the page, "The input consists of multiple test cases.". Thus, 10 test cases is adequate in testing the correctness of a solution

and measure the runtime and memory. We take the max value on all 10 test cases as a proxy of the runtime efficiency of the tested code.

D.5 PROMPTING GPT-3.5 TO GENERATE CODE SUMMARY (BOTH TEXT DESCRIPTIONS AND CATEGORICAL VALUES)

We present below the instruction we provide to GPT-3.5. We bold keywords in the prompt simply for the ease of reading. When calling the GPT-3.5 API, we used temperature $T = 0$ (i.e., greedy decoding) and max_tokens=500.

The list of tags (in the 1st prompt below) were collected from the CodeContests Li et al. (2022) dataset—we traverse all the problems in the dataset and union all the "tags" attribute.

The lists of algorithms and data structures (in the 2nd prompt below) were obtained from analyzing Wikipedia and querying GPT-4-turbo (0125) for a suggested list.

These categorical attributes serve as a complement to the text description described above and enhance the reliability of the results.

> Please provide a description to the following code in natural language. Explain the **functionality, algorithm, data structure, time complexity, space complexity** of the code.
> Finally, assign a few **tags** to the code. Here is a list of tags you can choose from:
> "binary search, math, special, trees, dp, greedy, games, dfs and similar, expression parsing, number theory, chinese remainder theorem, geometry, bitmasks, sortings, graph matchings, matrices, meet-in-the-middle, graphs, combinatorics, probabilities, constructive algorithms, schedules, two pointers, brute force, dsu, shortest paths, hashing, interactive, data structures, strings, ternary search, fft, flows, implementation"
> Answer each in a line in the example format of: 'Description: description\nFunctionality: functionality'
> {code}

> Please read the following code and infer the **algorithms** and **data structures** used in it.
> For algorithms, select (a few) from the following list:
> "Sorting Algorithms, Searching Algorithms, String Algorithms, Divide and Conquer Algorithms, Greedy Algorithms, Dynamic Programming, Recursion, Bit Manipulation, Backtracking, Graph Algorithms, Others"
> For data structures, select (a few) from the following list:
> "Arrays, Linked Lists, Stacks, Queues, Trees, Heaps, Hash Tables, Sets, Maps, Priority Queues, Others"
> Answer each in a line following the format of: 'Algorithms: candidate 1, candidiate 2, ..\nData structures: candidate 1, candidiate 2, ..\n'
> {code}

E ADDITIONAL RESULTS: CODING

E.1 EXAMPLES FOR PLAGIARISM SCORES

We present example model-generated code and human-written code as well as the plagiarism scores associated with the pairs in Fig. 15, 16,and 17. All the code are correct solutions to the problem 409A. The Great Game[5].

From the demonstrations it is evident that model-generated code (Fig. 15, 16) are highly similar in their style and structure; even the low score pair (Fig. 16) bear a significant level of similarity. In comparison, human-written code (Fig. 17) are clearly distinctive. These results support the validity of using the plagiarism score to evaluate code similarity.

E.2 ATTEMPTS ON VARYING PROMPTS

We experimented with four prompts on GPT-4, one plain prompt, one instructing the model to generate only the code solution, one employing role-playing Salewski et al. (2024) and one integrating chain-of-thought prompting Wei et al. (2022b). We present the concrete prompts in Fig. 18.

[5]Link to the problem: https://codeforces.com/problemset/problem/409/A

Test file: *train_gen/train-00000_0/train-00000_solution_codeonly_5_0.py* (**78.00%**)
Reference file: *train_gen/train-00000_0/train-00000_solution_codeonly_14_0.py* (**79.19%**)
Token overlap: 156

View matched code

```
n = int(input())
last_year = [input() for _ in range(n)]
current_year = [input() for _ in range(n)]

last_year_count = {}
current_year_count = {}

for size in last_year:
    if size in last_year_count:
        last_year_count[size] += 1
    else:
        last_year_count[size] = 1

for size in current_year:
    if size in current_year_count:
        current_year_count[size] += 1
    else:
        current_year_count[size] = 1

seconds = 0
for size in current_year_count:
    if size in last_year_count:
        if current_year_count[size] > last_year_count[size]:
            seconds += current_year_count[size] - last_year_count[si
    else:
        seconds += current_year_count[size]

print(seconds)
```

```
n = int(input())
prev_year = [input() for _ in range(n)]
current_year = [input() for _ in range(n)]

prev_count = {}
current_count = {}

for size in prev_year:
    if size in prev_count:
        prev_count[size] += 1
    else:
        prev_count[size] = 1

for size in current_year:
    if size in current_count:
        current_count[size] += 1
    else:
        current_count[size] = 1

seconds = 0
for size in current_count:
    if size in prev_count:
        diff = current_count[size] - prev_count[size]
        if diff > 0:
            seconds += diff
    else:
        seconds += current_count[size]

print(seconds)
```

Figure 15: An example for **a pair of model generated code** with **high plagiarism score**.

Test file: *train_gen/train-00000_19/train-00000_solution_codeonly_15_19.py* (**33.64%**)
Reference file: *train_gen/train-00000_19/train-00000_solution_codeonly_7_19.py* (**23.00%**)
Token overlap: 72

```
def find_winner(team1_actions, team2_actions):
    scores = {"8<": 0, "[]": 1, "()": 2}
    team1_score = 0
    team2_score = 0

    for i in range(0, len(team1_actions), 2):
        action1 = team1_actions[i:i+2]
        action2 = team2_actions[i:i+2]

        if scores[action1] == (scores[action2] + 1) % 3:
            team1_score += 1
        elif scores[action2] == (scores[action1] + 1) % 3:
            team2_score += 1

    if team1_score > team2_score:
        return "TEAM 1 WINS"
    elif team2_score > team1_score:
        return "TEAM 2 WINS"
    else:
        return "TIE"

team1_actions = input().strip()
team2_actions = input().strip()

print(find_winner(team1_actions, team2_actions))
```

```
def determine_winner(actions_team1, actions_team2):
    score_team1, score_team2 = 0, 0
    # Points system based on action pairs
    points_system = {
        ('8<', '[]'): 1, ('[]', '8<'): 1,
        ('()', '8<'): 1, ('8<', '()'): 1,
        ('[]', '()'): 1, ('()', '[]'): 1
    }

    # Split the actions into 2-character pairs
    actions_team1 = [actions_team1[i:i+2] for i in range(0, len(actio
    actions_team2 = [actions_team2[i:i+2] for i in range(0, len(actio

    # Compare actions and calculate scores
    for action1, action2 in zip(actions_team1, actions_team2):
        if (action1, action2) in points_system:
            score_team1 += points_system[(action1, action2)]
        elif (action2, action1) in points_system:
            score_team2 += points_system[(action2, action1)]

    # Determine the winner based on the scores
    if score_team1 > score_team2:
        return "TEAM 1 WINS"
    elif score_team2 > score_team1:
        return "TEAM 2 WINS"
    else:
        return "TIE"

# Reading input
actions_team1 = input().strip()
actions_team2 = input().strip()

# Determining the winner
result = determine_winner(actions_team1, actions_team2)
print(result)
```

Figure 16: An example for **a pair of model generated code** with **relatively low plagiarism score**.

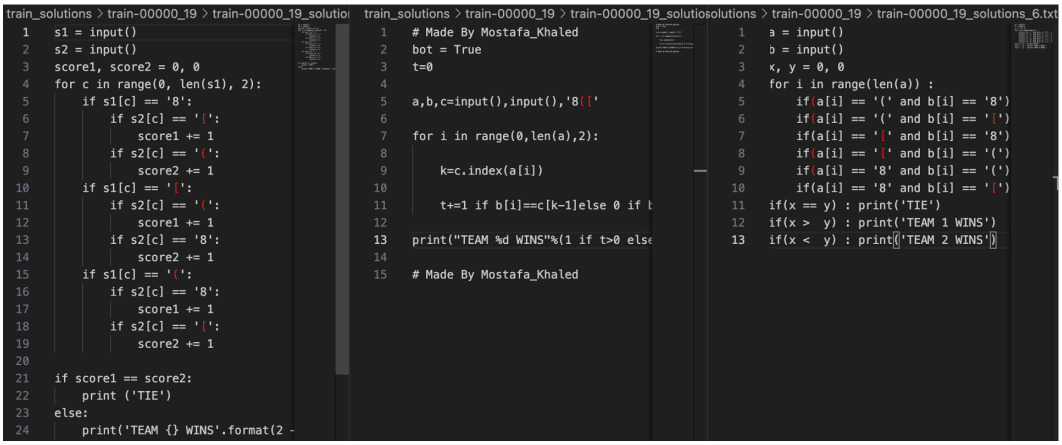


Figure 17: An example for **three human-written code** with ***zero*** **plagiarism score** for all the pairs.

Prompt 1: Please read the below problem description and generate a python code to solve the problem:

{problem_description}

Prompt 2: Please read the below problem description and generate a python code to solve the problem:

{problem_description}

Please only generate code and nothing else.

Prompt 3: Imagine you are a grandmaster in solving competitive programming problems. Your skills in algorithms, data structures, and problem-solving are unparalleled. You have a deep understanding of various programming paradigms and can easily navigate through complex problems with efficiency and elegance.

Please read the below problem description and generate a python code to solve the problem:

{problem_description}

Please only generate code and nothing else.

Prompt 4: Imagine you are a grandmaster in solving competitive programming problems. Your skills in algorithms, data structures, and problem-solving are unparalleled. You have a deep understanding of various programming paradigms and can easily navigate through complex problems with efficiency and elegance.

Please read the below problem description and generate a python code to solve the problem:

{problem_description}

Please think through the problem step by step, and then provide your solution. Then, test your code against the provided test cases in the problem. If your code fails to pass all the tests, please revise your code and try again until your code passes all the tests.

Figure 18: **Four prompts for instructing the model GPT-4** to generate code solution to a provided problem in {problem_description}. Prompt 1 is the most plain prompt; Prompt 2 additionally instructs the model to generate only the code without additional explanations; Prompt 3 employs the technique of role-playing Salewski et al. (2024), and Prompt 4 additionally integrates the technique of chain-of-thought prompting Wei et al. (2022b).

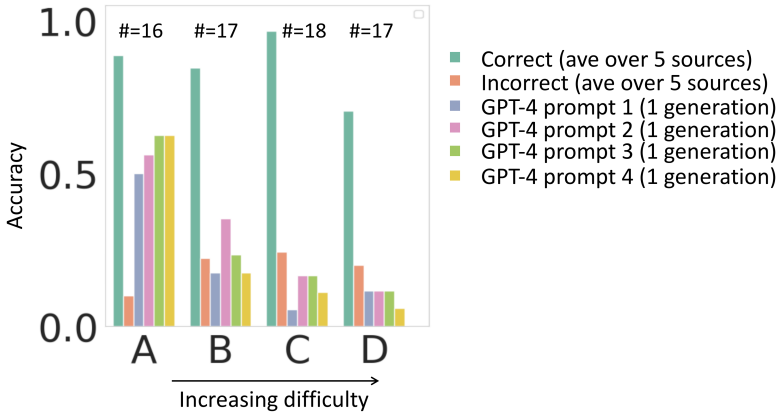


Figure 19: **Accuracy achieved by using the four prompts on Codeforces problems of varying difficulties.** The "Correct" and "Incorrect" refer to the results achieved by the human-written solutions in the source dataset CodeContests. Overall, we see that prompt 2 performs the best across different difficulty levels.

We evaluate the four prompts on Codeforces problems of varying difficulties and present the accuracy achieved by each prompt in Fig. 19. Overall, prompt 2 achieves the best results among the four.

E.3 Human Annotations for the Quality of LLM Summary

We annotate the quality of the LLM-generated summary of code given by `GPT-3.5`.

Step 1: We randomly sample 20 solutions of different problems, and review the LLM-generated attributes for these solutions; the attributes include the textual ones (description, functionality, algorithm, data structure), the inferred asymptotic complexity (time complexity, space complexity), as well as the categorical ones (tags, algorithm, data structures).

Step 2: We read through the problem description, the code solution, and the model-generated summary to evaluate the quality of model-generated summary.

Step 3: We follow the instruction "In each cell please give a score from 1-3 – 3 meaning absolutely correct, 2 meaning with some minor errors but acceptable, 1 meaning entirely incorrect" to score each attribute.

Step 4: We calculate the average score received by each attribute across the 20 sample and present the results in Table 3.

The high scores indicate that `GPT-3.5` can provide fairly accurate summary of the given code, supporting our choice of relying `GPT-3.5` to reliably extract the attributes.

Table 3: **The average scores given by human annotators on each attribute.** We adopted a 3-point scale where 3 means absolutely correct, 2 means with some minor errors but acceptable, 1 means entirely incorrect. The high scores indicate that `GPT-3.5` can provide fairly accurate summary of the given code.

Attribute	textual				complexity		categorical		
	description	functionality	algorithm	data structure	time complexity	space complexity	tags	algorithms	data structures
Score	2.65	2.55	2.90	3.00	2.95	2.70	2.90	2.50	2.90

E.4 FAILURE RESULTS WITH OPEN-SOURCE MODELS

We experimented with two open-source code models on these competitive programming problems: CodeLlama Roziere et al. (2023)[6] and StarCoder Li et al. (2023b)[7]. However, neither of the models were able to produce code solutions in our trials. We present one example interaction in Fig. 20, where the model demonstrated the memorization of a problem dataset but failed to produce a code solution to the given problem.

E.5 A COMPLETE SET OF RESULTS FOR THE EXPERIMENTS ON `GPT-4`

Accuracy The histogram of the accuracy can be found in Fig. 21(left).

Plagiarism score The histogram along with the KDE plot for the plagiarism scores can be found in Fig. 21(right).

Efficiency The efficiency results in terms of the asymptotic complexity (specifically, the histogram of the time complexity and space complexity) can be found in Fig. 22. The efficiency results in terms of the runtime efficiency (specifically, the density plots of runtime and memory) can be found in Fig. 23.

Code summary The KDE plots of the mean pairwise embedding cosine similarity of the code summary *(textual)* can be found in Fig. 24. The stacked bar plots of the mean pairwise Jaccard similarity of the code summary *(categorical)* can be found in Fig. 25.

E.6 CLAUDE

Accuracy The histogram of the accuracy can be found in Fig. 26(left). In comparison to `GPT-4` (see Fig. 21(left)), the accuracy of `Claude-3` is much lower.

Plagiarism score The histogram along with the KDE plot for the plagiarism scores can be found in Fig. 26(right). The plagiarism scores achieved by `Claude-3` remain high, similarly to that on `GPT-4`.

Efficiency The efficiency results in terms of the asymptotic complexity (specifically, the histogram of the time complexity and space complexity) can be found in Fig. 27. The efficiency results in terms of the runtime efficiency (specifically, the density plots of runtime and memory) can be found in Fig. 28. Different from `GPT-4`, solutions generated by `Claude-3` are less efficient than source solutions. This result, together with the accuracy result, indicates that the code generation ability of `Claude-3` is inferior than `GPT-4`.

Code summary The KDE plots of the mean pairwise embedding cosine similarity of the code summary *(textual)* can be found in Fig. 29. The stacked bar plots of the mean pairwise Jaccard similarity of the code summary *(categorical)* can be found in Fig. 30. Similar to `GPT-4`, `Claude-3` also demonstrate a narrower range of expressed ideas.

F COMPUTE RESOURCES AND DATA LICENSES

F.1 COMPUTE

The book review generation ($N = 742$, $n = 10$, `max_new_tokens`=500) on open-source models took around 10 hours on one H100 card per run, i.e., per combination of sampling parameters (T and p) and prompts. The number increased to around 80 hours for `GPT-4`.

The code generation ($N = 100$, $n = 100$, `max_new_tokens`=2048) on `GPT-4` took around 60 hours per run. The rate of generation was similar on `Claude-3-Sonnet`; the total runtime doubled since we generated twice many solutions ($n = 200$) due to its lower accuracy.

[6]Link at `https://huggingface.co/codellama/CodeLlama-13b-Instruct-hf`
[7]Link at `https://huggingface.co/bigcode/starcoder`

Prompt
Please read the below problem description and generate a python code to solve the problem:

Gregor is learning about RSA cryptography, and although he doesn't understand how RSA works, he is now fascinated with prime numbers and factoring them. Gregor's favorite prime number is P. Gregor wants to find two bases of P. Formally, Gregor is looking for two integers a and b which satisfy both of the following properties.

* P mod a = P mod b, where x mod y denotes the remainder when x is divided by y, and
* $2 \leq a < b \leq P$.

Help Gregor find two bases of his favorite prime number!

Response
Input Format

The first line contains an integer T, the number of test cases. T lines follow. Each line contains an integer P, Gregor's favorite prime number.

Output Format

For each test case, print two lines. The first line should contain the first base, and the second line should contain the second base.

Constraints

* $2 \leq T \leq 1000$ * $2 \leq P \leq 10^9$

Sample Input

2
3
5

Sample Output

2
3
2
5

Explanation

For the first test case, the two bases are 2 and 3. For the second test case, the two bases are 2 and 5.\n\n\n\n...\n

Figure 20: **Generation result of StarCoder.** The model was unable to generate the *requested* code solution as instructed; instead, it generated some boilerplate content such as input format and output format, which often exists in datasets of this kind of problems. After these boilerplate content, the model simply generated hundreds of \n and then stopped.

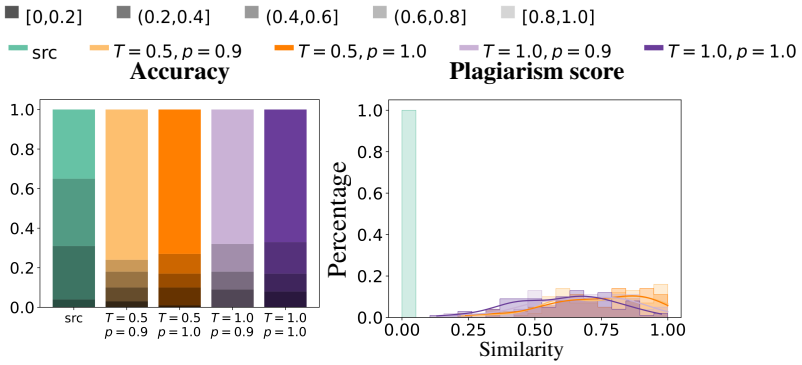


Figure 21: **Model**: `GPT-4`. **(Left)** Stacked bar charts of the accuracy values achieved by the source solutions as well as `GPT-4` generated solutions under various kwargs. **(Right)** Histogram plus kernel density estimation (KDE) plots of the plagiarism scores.

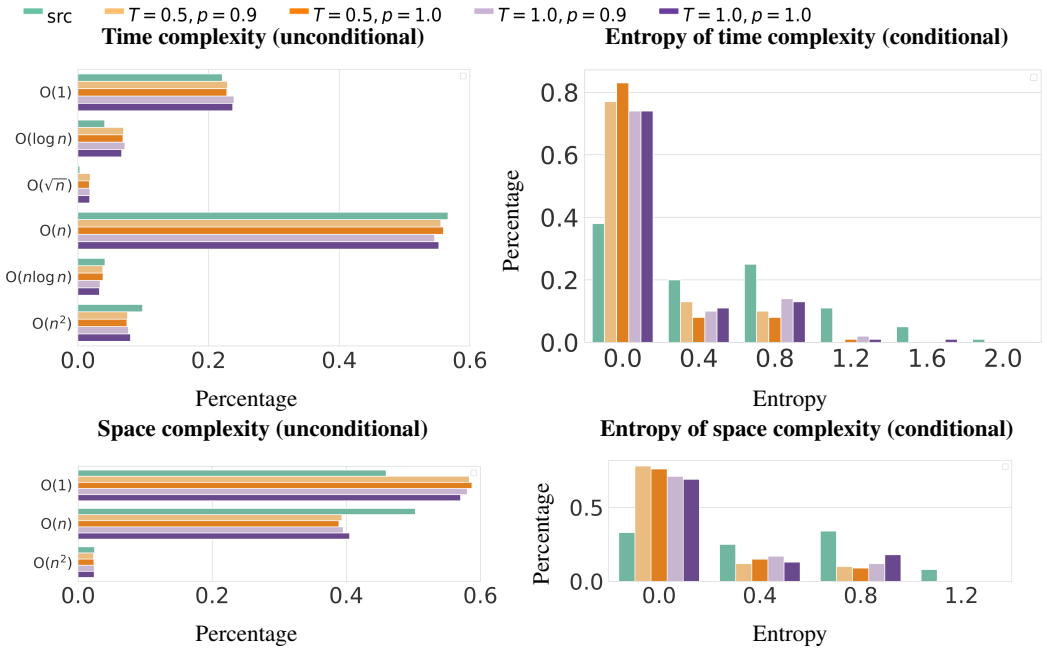


Figure 22: **Model**: `GPT-4`. **(Left)** Histograms (or grouped bar charts) for the time and space complexity for the unconditional distribution. **(Right)** Histograms for the *entropy* of the time and space complexity for the conditional distribution. The figures suggest that the generated code is more efficient than the source code, while showing a decrease in diversity.

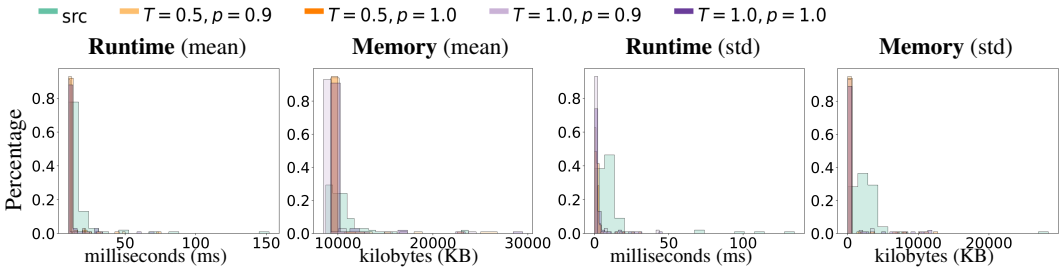


Figure 23: **Model**: `GPT-4`. **Histograms of the mean and standard deviation (std) of runtime** (in milliseconds) **and memory usage** (in kilobytes). We filtered out some datapoints for better visualization, specifically, runtime above 200 ms, and memory usage above 30,000 KB.

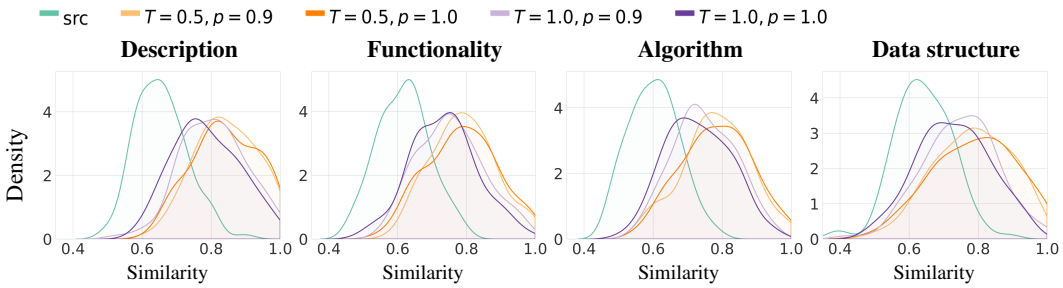


Figure 24: **Model**: `GPT-4`. **Kernel density estimation (KDE) plots of the mean pairwise cosine similarity** for the four attributes (description, functionality, algorithm, and data structure) represented as extracted embeddings of natural language descriptions.

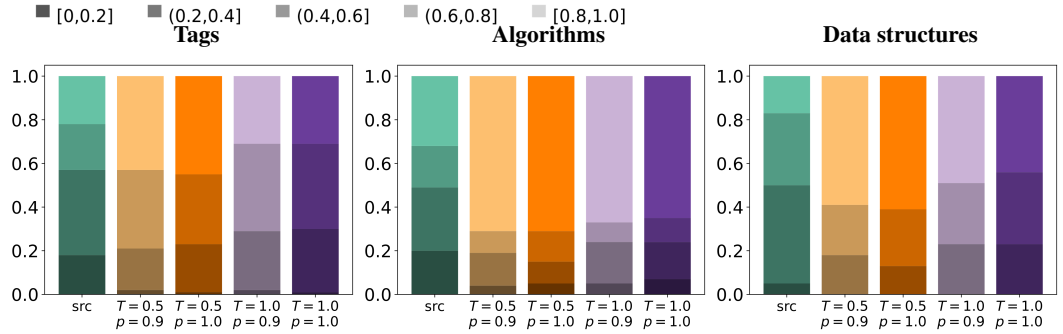


Figure 25: **Model**: `GPT-4`. **Stacked bar charts of the mean pairwise Jaccard index** for the three attributes (tags, algorithms, and data structures) represented as sets of categorical variables.

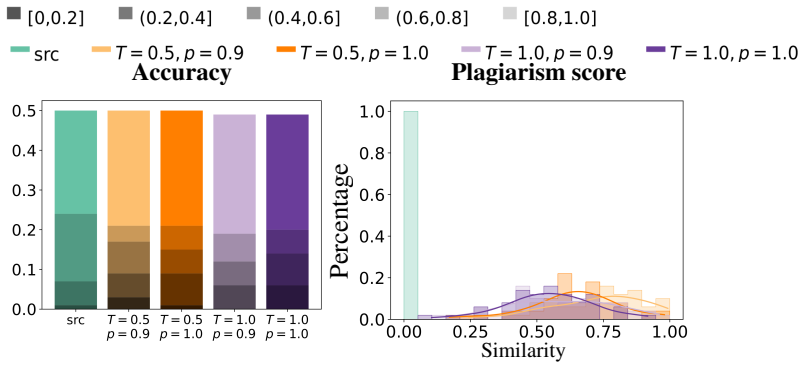


Figure 26: **Model**: `Claude-3-Sonnet`. **(Left)** Stacked bar charts of the accuracy values achieved by the source solutions as well as Claude generated solutions under various kwargs. **(Right)** Histogram plus kernel density estimation (KDE) plots of the plagiarism scores for source solutions and Claude generated solutions.

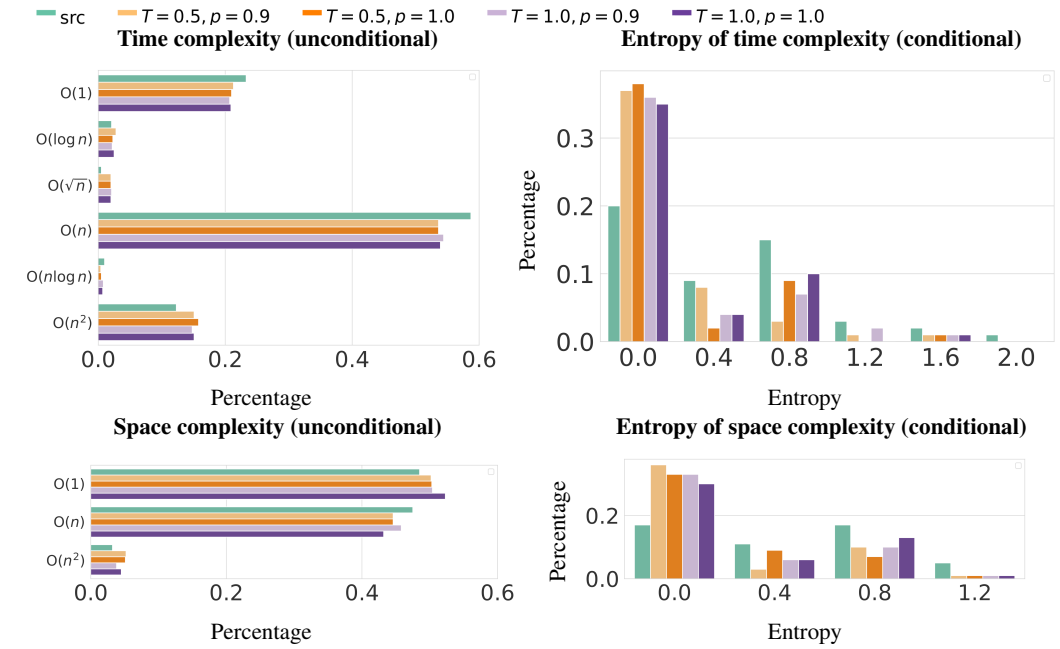


Figure 27: **Model**: `Claude-3-Sonnet`. **(Left)** Histograms (or grouped bar charts) for the time and space complexity for the unconditional distribution. **(Right)** Histograms for the *entropy* of the time and space complexity for the conditional distribution. The figures suggest that the generated code is more efficient than the source code, while showing a decrease in diversity.

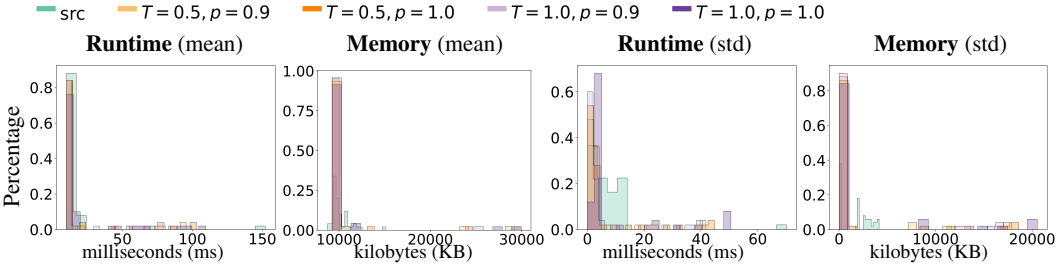


Figure 28: **Model**: `Claude-3-Sonnet`. **Histograms of the mean and standard deviation (std) of runtime** (in milliseconds) **and memory usage** (in kilobytes). We filtered out some datapoints for better visualization, specifically, runtime above 200 ms, and memory usage above 30,000 KB.

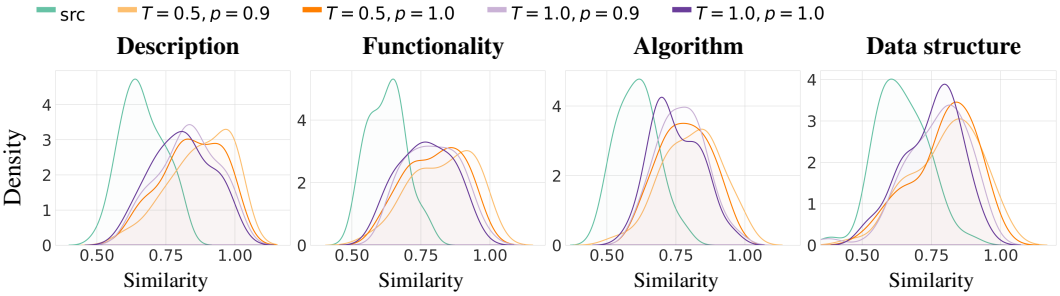


Figure 29: **Model**: `Claude-3-Sonnet`. **Kernel density estimation (KDE) plots of the mean pairwise cosine similarity** for the four attributes (description, functionality, algorithm, and data structure) represented as extracted embeddings of natural language descriptions.

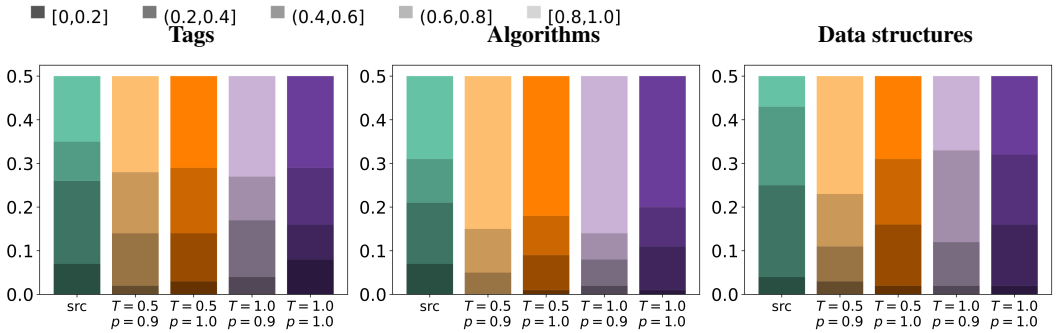


Figure 30: **Model**: `Claude-3-Sonnet`. **Stacked bar charts of the mean pairwise Jaccard index** for the three attributes (tags, algorithms, and data structures) represented as sets of categorical variables.

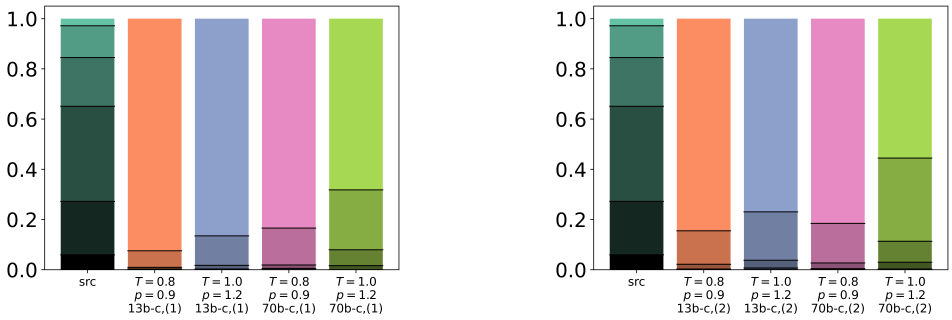
F.2 Licenses

We used the Goodreads dataset Wan et al. (2019)[8] for the book review scenario. Their license is Apache License[9] as provided in their repository.

We used the CodeContests dataset Li et al. (2022)[10] for the coding scenario. Their license is Apache License[11] as provided in their repository.

G Extended Investigations

G.1 Influence of Model Size on Monoculture



(a) Prompt (1)

(b) Prompt (2)

Figure 31: **Monoculture persists despite model scaling**.

We conducted experiments to understand the pervasiveness of monoculture as a function of model size. To this end, we conducted the sentiment analysis study (on generated book-reviews) for Llama 70b (instruction fine-tuned + aligned) – denoted 70b-c(x) in the figure, where x denotes the prompt (refer § 5.1); we compare this with the 13b model from the same family. For the 70b model, we used a quantized version for faster inference times given the narrow window of the rebuttal period. Our results, in Figure 31 show that while this larger model has "greater diversity" than the smaller model (potentially due to an increase in the diversity of the training dataset used to train this model),

[8]Link to the dataset website: `https://mengtingwan.github.io/data/goodreads.html`

[9]Link to the license: `https://github.com/MengtingWan/goodreads/blob/master/LICENSE`

[10]Link to the dataset: `https://github.com/google-deepmind/code_contests`

[11]Link to the license: `https://github.com/google-deepmind/code_contests/blob/main/LICENSE`

the phenomenon of monoculture (i.e., decreased diversity compared to the source reviews) remains. This is the case across both prompts, and despite variations in sampling.

G.2 Influence of Length on Sentiment Scores

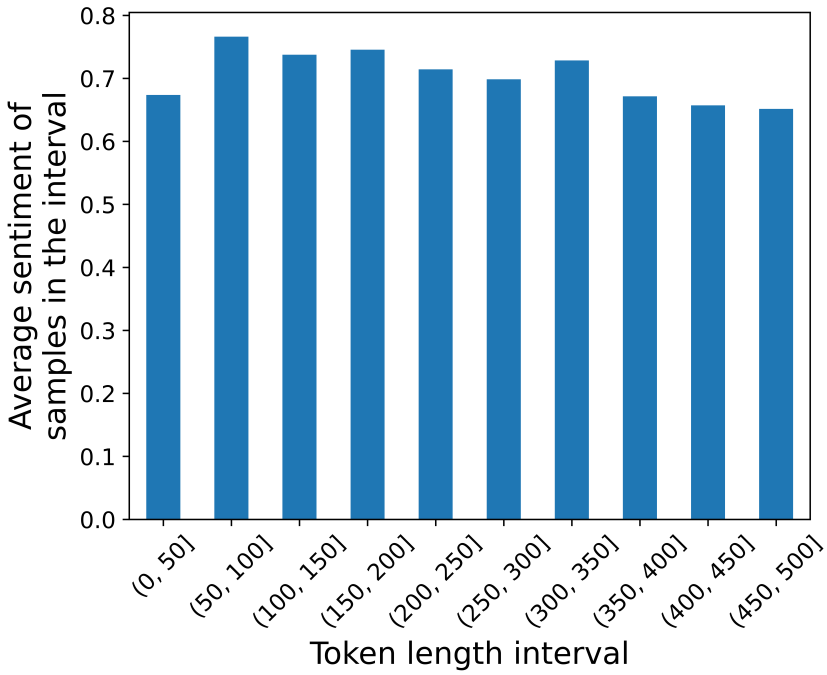


Figure 32: **Average sentiment score is agnostic of length of reviews.**

In Appendix B.1, we state that our book-reviews curation process involved discarding those reviews less than 300 words. From Figure 32, we find that there is little correlation between a sample's length and sentiment i.e., no biases are introduced by discarding shorter samples.

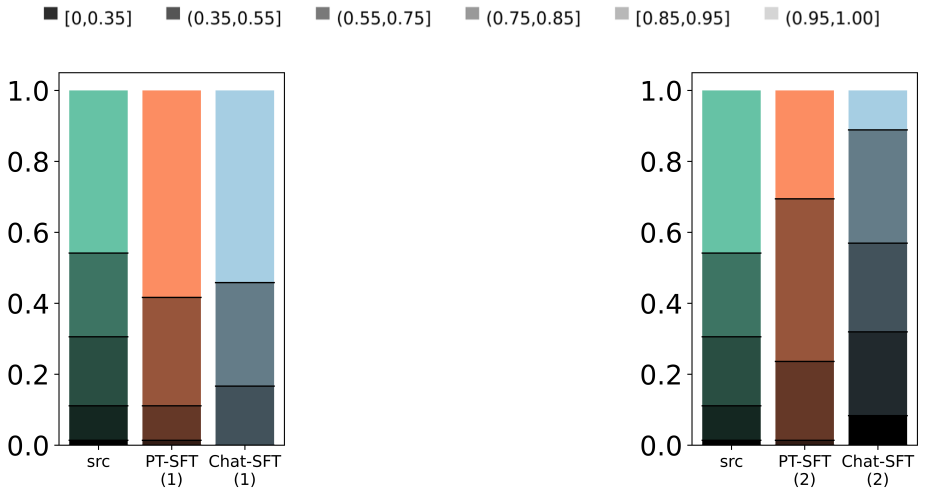
G.3 More Training Data Control & Connections to Monoculture

While one might be skeptical about the assumption that book-reviews are present in the training data, circumventing this skepticism would involve (a) full knowledge of the training dataset, and (b) picking a downstream generative task that is influenced only one sub-component of dataset (in some provable manner). However, satisfying both of these requirements is challenging because:

- Should we have full control of the training dataset, training our own model from scratch remains prohibitively expensive.
- Open source models, where the training set is known, demonstrate extremely low-performance on the tasks we perform in this paper: for example OLMO is incapable of generating acceptable coding solutions, or reviews that are coherent in a consistent/reliable manner.

We begin by obtaining a new, unseen dataset (denoted `src`): We filter out books published after October 1, 2023 from the GoodReads dataset[12]. In this dataset, a majority of the book (72 out of 79) come with 5 reviews and the remaining books have fewer reviews. We thus retain the 72 books with 5 reviews, and compose a dataset of 360 samples. We then instruction-tune Llama-13b (both the pre-trained and chat versions) using the aforementioned dataset. The template we used was {``instruction'': ``Write a book review for the book

[12] https://www.kaggle.com/datasets/dk123891/books-dataset-goodreadsmay-2024



(a) Prompt 1

(b) Prompt 2

Figure 33: **Even with full control of the fine-tuning dataset, monoculture persists.**

titled {title}'', ``response'': ``{review}''}. We perform parameter efficient fine-tuning (LoRA dimension=4) and fine-tuned for 3 epochs with learning rate=10^{-3} and batch size=2. We ensured that the perplexity (measured on "wikitext-2-raw-v1") of the model after fine-tuning does not increase a lot, to avoid overfitting.

We then proceed to generate book-reviews for the aforementioned books, and measure if the diversity of sentiment has changed. Our results, in Figure 34 suggest that with either the pre-trained model (PT) or the aligned/chat (Chat) model as the initialization, fine-tuning on the curated small review dataset resulted in the average sentiment becoming more positive after fine-tuning on the pre-trained model (but still being close to the data distribution), and the average sentiment becoming more negative after fine-tuning on the Chat model. The implication of this experiment is that while fine-tuning may provide some reprieve, RLHF skews the sentiment significantly. Thus, monoculture still persists in this case.

G.4 Evidence of GoodReads in CommonCrawl

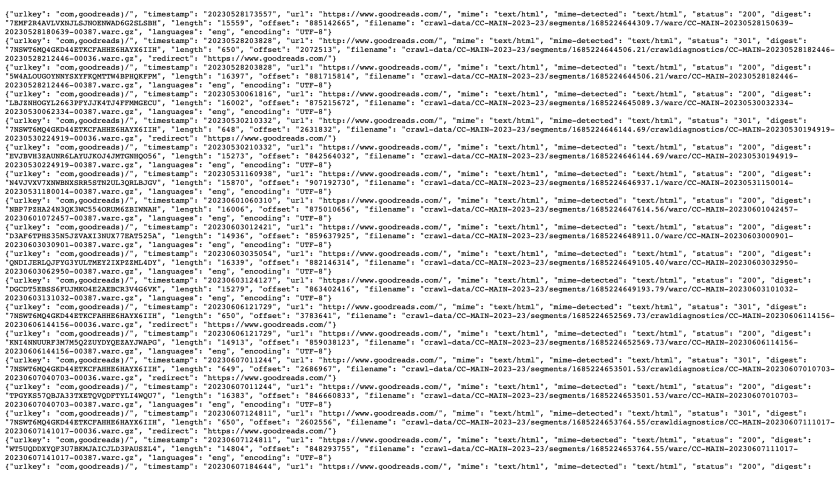


Figure 34: **Screenshot of goodreads.com being present in the common crawl index**

Baack (2024) highlight that Llama v1 is trained on CommonCrawl[13]. Upon closer inspection (using the common crawl interface), one can see that the goodreads.com website and its associated URLs are part of this dataset (see Figure 34). This suggests that the training data of Llama v1 includes book reviews from the GoodReads dataset. While Baack (2024) note that there is limited visibility into the training dataset for Llama 2, we can safely assume that it was built atop of the data collected for Llama 1, including CommonCrawl. Thus, we would like to stress that the experiments in our paper are valid.

[13] https://index.commoncrawl.org/