RoCoIns: Enhancing Robustness of Large Language Models through Code-Style Instructions

Yuansen Zhang[1*] **Xiao Wang**[1*†] **Zhiheng Xi**[1] **Han Xia**[1]
Tao Gui[2†] **Qi Zhang**[1†] **Xuanjing Huang**[31]
[1] School of Computer Science, Fudan University, Shanghai, China
[2] Institute of Modern Languages and Linguistics, Fudan University, Shanghai, China
[3] International Human Phenome Institutes, Shanghai, China
zhangys22@m.fudan.edu.cn, {xiao_wang20,tgui,qz}@fudan.edu.cn

Abstract

Large Language Models (LLMs) have showcased remarkable capabilities in following human instructions. However, recent studies have raised concerns about the robustness of LLMs when prompted with instructions combining textual adversarial samples. In this paper, drawing inspiration from recent works that LLMs are sensitive to the design of the instructions, we utilize instructions in code style, which are more structural and less ambiguous, to replace typically natural language instructions. Through this conversion, we provide LLMs with more precise instructions and strengthen the robustness of LLMs. Moreover, under few-shot scenarios, we propose a novel method to compose in-context demonstrations using both clean and adversarial samples (*adversarial context method*) to further boost the robustness of the LLMs. Experiments on eight robustness datasets show that our method consistently outperforms prompting LLMs with natural language instructions. For example, with gpt-3.5-turbo, our method achieves an improvement of 5.68% in test set accuracy and a reduction of 5.66 points in Attack Success Rate (ASR).

Keywords: Large Language Models, Robustness, Code-style Instructions

1. Introduction

Large language models (LLMs) have shown increasing power in following human instructions and solving various NLP tasks. (Sanh et al., 2022; Chung et al., 2022; Ouyang et al., 2022a; Wang et al., 2023c,b; Xi et al., 2023).

However, recent evaluations in terms of LLMs have revealed their insufficient robustness when prompted with instructions containing textual adversarial samples, raising concerns about their real-world applications (Liu et al., 2023; Wang et al., 2023a; Ye et al., 2023; Chen et al., 2023). By inserting slight perturbations into clean samples at the character, word, or sentence level (Gao et al., 2018; Ren et al., 2019a; Li et al., 2019), the outputs of LLMs occasionally deviate from the expected results. For example, in Aspect-based sentiment analysis tasks, when inverting the sentiment polarity of the target aspects, the performance of gpt-3.5-turbo falls by nearly 35% under zero-shot scenarios (Ye et al., 2023).

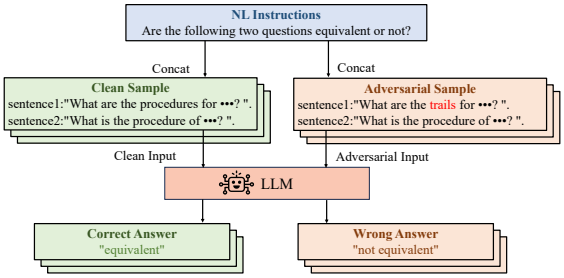
In response to textual adversarial attacks, various adversarial defense methods have been proposed, such as adversarial training (Jiang et al., 2020), interval bound propagation (Dvijotham et al., 2018) and randomized smoothing (Cohen et al., 2019). However, all these methods require parameters update of models, which can be infeasible when it comes to powerful modern LLMs such as GPT-3 (Brown et al., 2020) with only APIs provided. Consequently, limited research has been conducted on enhancing the robustness of such *closed source black-box* LLMs.

To alleviate this problem, we explore enhancing the robustness of LLMs through instructions design. Typically, instructions are formulated using natural language. However, the inherent ambiguity of natural language can make LLMs extremely sensitive to instructions, as even slight modifications to the instructions can result in a significant drop in performance (Zhao et al., 2021; Holtzman et al., 2022). Besides, we believe that introducing adversarial samples into the instructions aggravates this phenomenon and leads to low robustness. Therefore, it is important to design an instruction format that overcomes these shortcomings.

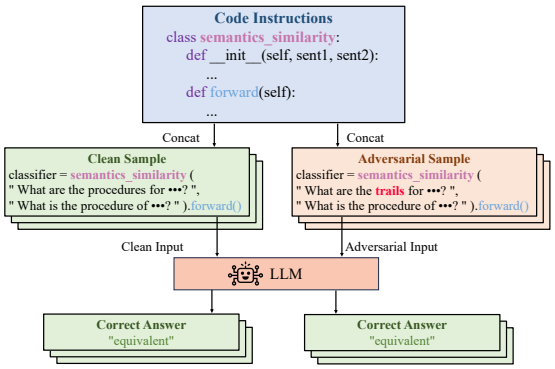
In this paper, we introduce a novel approach **RoCoIns**: Enhancing **Ro**bustness of LLMs through **Co**de-Style **Ins**tructions. The overall framework is shown in Figure 1. We convert the instruction formats from natural language to code style. The advantages of code, such as being more structural and less ambiguous, provide LLMs with clearer and more concise instructions (Mishra et al., 2023; Wang et al., 2022b; Li et al., 2023a), which lead to robustness improvement. Additionally, we propose the *adversarial context method* to further boost the robustness of LLMs. Inspired by (Dai et al., 2023; von Oswald et al., 2022) that in-context learning (ICL) can be considered as implicit finetuning, we hypothesize that by incorporating both clean

[*] Equal contribution.
[†] Corresponding Author

LREC-COLING 2024, pages 14186–14203



(a) Prompt LLMs with natural language instructions



(b) Prompt LLMs with code-style instructions

Figure 1: An illustration of prompting LLMs with natural language instructions and code-style instructions for the *semantics consistent judgment* tasks. The input sample contains a sentence pair. We show a clean sample and an adversarial sample, respectively. This code-style instruction can be applied to arbitrary tasks with task-specific design.

and adversarial samples to compose the in-context demonstrations can be viewed as a type of implicit adversarial training. We verify the effectiveness of the method on eight datasets and decrease the average Attack Success Rate (ASR) by 5.66 points with gpt-3.5-turbo. We conduct further analysis to demonstrate the advantages of using code-style instructions.

To sum up, our contributions are as follows:

- We introduce **RoCoIns**, a novel approach to enhance the robustness of LLMs against textual adversarial attacks by utilizing code-style instructions.
- Moreover, we propose the *adversarial context method* to further boost the robustness of LLMs.
- We conduct experiments on eight robustness datasets and verify the effectiveness of our method, which outperforms prompting LLMs with natural language instructions.

2. Background

2.1. Textual Adversarial Attack

Textual adversarial attacks commonly generate explicit adversarial samples by substituting components of sentences with their equivalents while preserving a high degree of semantic similarity (Ren et al., 2019a; Wang et al., 2021a). Given a clean sentence $x = (t_1, t_2, \ldots, t_n)$, where $t_i, 1 \leq i \leq n$ denotes each token in the sentence. l represents its ground truth label. Textual adversarial attacks replace some original tokens with their counterparts to fool the objective model. For example, substituting t_i with $\hat{t_i}$ creates an adversary: $\hat{x} = (t_1, t_2, \ldots, \hat{t_i}, \ldots, t_n)$. For an adversary, the objective model F generates its label as follows:

$$\hat{l} = \text{argmax}\, F\left(\cdot|\hat{x}\right) \tag{1}$$

where $\hat{l} \neq l$ means a successful attack.

In this paper, we **mainly focus on attacking samples rather than instructions**. The detailed difference with other attack formats, such as prompt attacks, can be found in Section 6.

2.2. In-context learning with LLMs

Due to the remarkable ICL abilities of LLMs, by providing LLMs with a few demonstration input-output pairs, they can predict the label for an unseen input without parameter updates. Formally, we randomly select k sample pairs $\{(x_i, y_i)\}_{i=1}^{k}$ from the training set and concatenate them as a string to compose the in-context demonstrations $D = x_1 \oplus y_1 \cdot x_2 \oplus y_2 \cdot \ldots \cdot x_k \oplus y_k$, where $\oplus$ means concatenation between the input and output within a sample and $\cdot$ means concatenation between different samples. During inference, a new test input x_{test} is appended to the demonstrations, and $D \cdot x_{test}$ is fed into the model for completion and thereby generates an answer $y_{test}^{'}$.

3. Method

In this section, we first describe how we recast the instructions from natural language to code style (Section 3.1). Then we introduce the *adversarial context method* (Section 3.2).

3.1. Formulating Instructions into Code Style

Considering an example in a task with the form $(\mathcal{T}, \mathcal{S}, \mathcal{L})$, where $\mathcal{T}$ denotes the task instruction, $\mathcal{S}$ refers to input sample and $\mathcal{L}$ represents the corresponding label to be generated. Typically, both $\mathcal{T}$ and $\mathcal{L}$ are expressed in the natural language format. However, due to the inherent ambiguity, LLMs

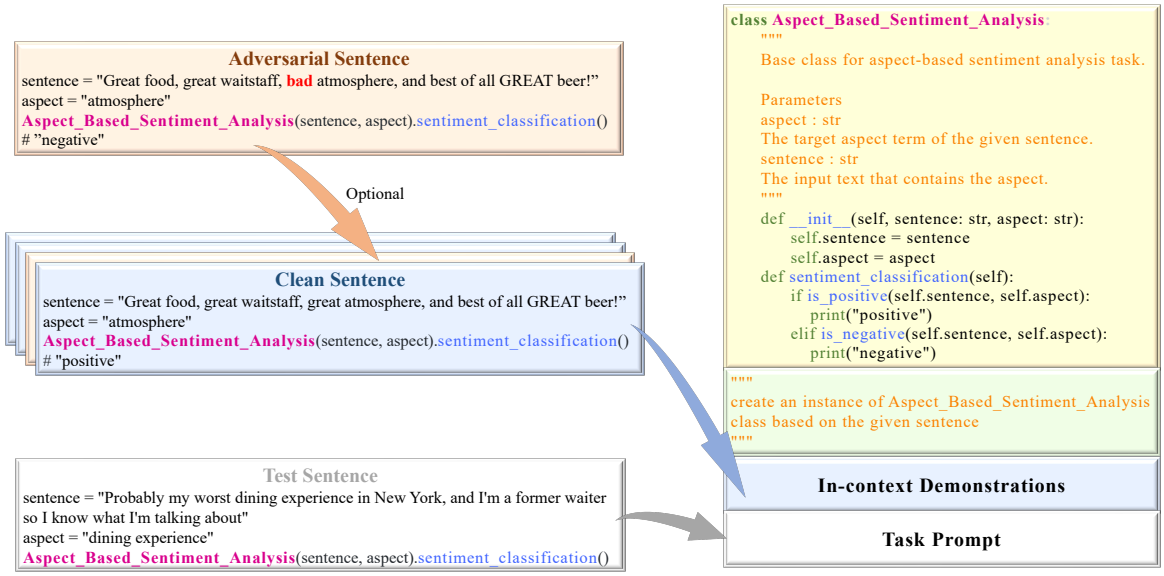


Figure 2: Components of code-style instructions. (**1**) Class definition mainly contains the class name, annotation, initial function and implementation function. (**2**) In-context demonstrations consist of k (adversarial) samples in the corresponding code style. (**3**) Task prompt follows the same format as demonstrations without a ground truth label.

have shown extreme sensitivity to these natural language instructions, as even slight modifications to the instructions can lead to a substantial decrease in performance (Zhao et al., 2021; Holtzman et al., 2022). In contrast, code-style instructions, which are less ambiguous and more structural, can serve as an alternative to natural language instructions and provide LLMs with more concise instructions. The primary idea of our method is to convert $\mathcal{T}$ from its original natural language format to a semantically equivalent non-executable pseudo-code format. In this work, we mainly define a **Python class** to achieve this conversion. To illustrate our method, we utilize the aspect-based sentiment analysis (ABSA) task as a running example (Figure 2). ABSA aims to determine the sentiment polarity ("*positive*", "*neutral*" or "*negative*") of an aspect presented in a sentence. Our code-style instruction mainly consists of the following components:

Class Name First, we convert the explanation of the task into the class name. The class name can be viewed as a summary of the task.

Annotation The annotations provide task descriptions that are typically rephrased versions of natural language instructions. Besides, the annotations also contain descriptions of the parameters, including their types and explanations.

Initial Functions The initial function defines the input components of this task. For example, in the ABSA task (Figure 2), we define two class instance variables *sentence* and *aspect*, which will be utilized in the subsequent implementation functions.

Implementation Functions The implementation functions detail the solution process for the task. This part is typically constructed based on the annotations and serves as a pseudo-code alternative version of the annotations. Following (Mishra et al., 2023), the implementation function may include sub-task functions, which are usually not explicitly defined and convey their functionalities through descriptive names and parameters. For example, in the ABSA task (Figure 2), $is\_positive$ and $is\_negative$ are two sub-task functions to determine the sentiment polarity of an aspect word.

Task Prompt Once the class is defined, we can utilize it by creating an instance object. These objects, accompanied by their properties definitions, compose the task prompt. Figure 2 provides an example of the task prompt. Typically, we compose an in-context demonstration by concatenating the task prompt with its ground truth label. Finally, we concatenate the class definition with several in-context demonstrations and a task prompt containing a test sample to construct the model input and expect the model to generate final outputs.

3.2. Adversarial Context Method

In this work, we propose the *adversarial context* method to further boost the robustness of LLMs.

Recent studies have shown that ICL can be regarded as a form of implicit fine-tuning (Dai et al., 2023; von Oswald et al., 2022). (Dai et al., 2023) theoretically prove that Transformer attention has a dual form of gradient descent and demonstrate that ICL behaves similarly to explicit finetuning from multiple perspectives. **Thus, we hypothesize that utilizing both clean and adversarial samples to compose in-context demonstrations can be regarded as a type of implicit adversarial training.** Formally, following the definitions in Section 2.1 and 2.2, we first transform the clean sample pair (x_i, y_i) and adversarial sample pair $(\hat{x_i}, \hat{y_i})$ into their corresponding code-style format (x_i^c, y_i^c) and $(\hat{x_i^c}, \hat{y_i^c})$. Then we compose the demonstrations $D = x_1^c \oplus y_1^c \cdot \hat{x_1^c} \oplus \hat{y_1^c} \cdot ... \cdot x_m^c \oplus y_m^c \cdot \hat{x_m^c} \oplus \hat{y_m^c}$ by concatenating both clean and adversarial samples. We keep the total number of sample pairs unchangeable.

4. Experiments

4.1. Experimental setup

Model We conduct experiments mainly using the GPT-3.5 Series models with `text-davinci-003` and `gpt-3.5-turbo` from OpenAI [1]. We choose these two models because GPT-3.5 Series models have shown remarkable code understanding abilities, making them better suited to our proposed method (White et al., 2023). These two models support an input length of up to 8k and 4k tokens, respectively.

Hyperparameters We acquire the predictions of the models through OpenAI API [2]. We prompt LLMs with greedy decode by setting the sampling temperature $t = 0$. Besides, we set the max number of generated tokens to 128 tokens.

Datasets In this paper, we mainly conduct experiments on two adversarial datasets: **AdvGLUE** (Wang et al., 2022) and **Restaurant** (Xing et al., 2020). AdvGLUE is an adversarial version of the GLUE (Wang et al., 2018) dataset, consisting of SST-2, QQP, MNLI, QNLI and RTE. We use the test set of AdvGLUE. Restaurant is an aspect-based sentiment analysis robustness dataset generated from SemEval 2014 Restaurant dataset (Pontiki et al., 2014) by infusing three types of transformations (RevNon, RevTgt and AddDiff) into it (Xing et al., 2020). We randomly select 300 samples from the test set of Restaurant for each transformation to compose our test set **Restaurant-T**.

[1] https://platform.openai.com/docs/models/gpt-3-5

[2] https://openai.com/api

Few-shot Setting For each task, we randomly select k samples from the dataset. The choice of k is varied between different tasks according to its number of classes and we explain the reason in Section 5.2.2. The detailed value of k for each task can be found in Table 1.

Instructions Design The natural language instructions for AdvGLUE are the same with (Wang et al., 2023a) and for Restaurant-T, we choose the same prompts following (Chen et al., 2023). For code-style instructions, we follow Figure 2 to construct instructions for different tasks.

Evaluation Following (Wang et al., 2023a), we use Attack Success Rate (ASR) as the evaluation metric for robustness. ASR is formally defined as :

$$ASR = \sum_{(x,y)\in T} \frac{\mathbb{1}[f(A(x)) \neq y]}{\mathbb{1}[f(x) = y]} \quad (2)$$

where dataset $T = \{(x_i, y_i)\}_{i=1}^{N}$ consists of N samples and A refers to an adversarial attack method, which generates adversarial samples. In general, the model's robustness against adversarial attacks is inversely proportioned to ASR. All experiments in this paper are conducted 3 times with different demonstrations and we report the mean results.

Baselines

1) **Zero-shot NL Prompting** To evaluate the impact of few-shot prompting on enhancing the robustness of language models (LLMs), we consider zero-shot natural language prompting as a baseline for comparison. The zero-shot results of AdvGLUE are from (Wang et al., 2023a).
2) **Few-shot NL Prompting** Under few-shot settings, we compare our approach with natural language prompting. By using the same natural language instructions with zero-shot prompting, we additionally provide LLMs with a few [Problem, Answer] samples to help LLMs better understand the tasks and standardize output formats.
3) **Few-shot CoT Prompting** Since Chain-of-Thought (CoT) (Wei et al., 2023b) has verified its effectiveness in improving performance on various tasks, we also incorporate CoT as a baseline to explore its effectiveness in robustness improvement. Specifically, we provide LLMs with a set of [Problem, Rational, Answer] samples to encourage LLMs to think step-by-step and generate final answers.

Model	Method	Dataset(ASR)								Avg(ASR)	Avg(Acc)
		AdvGLUE					Restaurant-T				
		SST-2	QQP	MNLI	QNLI	RTE	RevTgt	RevNon	AddDiff		
Random		50.0	50.0	66.7	50.0	50.0	66.7	66.7	66.7	58.35	41.67
Zero-Shot											
davinci-003	NL	44.6	55.1	44.6	38.5	34.6	44.11	20.00	13.19	36.84	–
gpt-3.5-turbo	NL	39.9	18.0	32.2	34.5	24.74	49.42	36.09	42.67	34.68	–
Few-Shot											
Shot Number		4	6	6	4	4	6	6	6		
davinci-003	NL	25.39	23.94	25	24.03	15.18	25.09	11.39	10.16	20.02	72.07
	CoT	23.07	26.56	23.8	23.62	14.28	24.04	9.35	7.65	19.05	73.43
	Code	23.97	25.35	22.54	21.73	**12.65**	23.52	**5.79**	5.08	17.58	75.82
	Code+adv	**20.93**	**22.53**	**22.33**	**20.21**	**12.65**	**21.97**	6.52	**4.66**	**16.47**	**77.20**
gpt-3.5-turbo	NL	19.23	23.07	21.73	18.75	22.05	29.19	17.03	16.4	20.93	70.45
	CoT	21.08	20.63	14.85	18.34	**21.42**	34.67	14.02	14.28	19.91	71.14
	Code	17.83	18.46	18.55	14.28	21.43	23.35	14.4	**10.08**	17.29	74.73
	Code+adv	**16.43**	**9.23**	**14.4**	**10.71**	22.85	**22.43**	**13.4**	12.71	**15.27**	**76.13**

Table 1: Experiments performances on AdvGLUE and Restaurant-T datasets. We report the $ASR(\downarrow)$ for each method. We also report the average accuracy($Avg(Acc)\uparrow$) in the last column. In this table, Our methods and the best results are highlighted in **bold**. NL and **Code** refer to prompting with natural language and code-style instructions, respectively. **Code+adv** refers to our proposed *adversarial context method*.

Model	Method	Clean	Adversarial
davinci-003	NL	86.78	72.07
	Code	87.25(+0.47)	75.82(+3.75)
	Code+adv	-	77.20(+5.13)
gpt-3.5-turbo	NL	85.61	70.45
	Code	86.13(+0.52)	74.73(+4.28)
	Code+adv	-	76.13(+5.68)

Table 2: Average Accuracy on the 8 clean and adversarial datasets for NL, **Code** and **Code+Adv** methods.

4.2. Results

NL instructions vs. Code-style instructions As shown in Table 1, prompting LLMs with code-style instructions consistently outperforms prompting with natural language instructions. Specifically, code-style instructions result in a 2.44 and 3.64 point reduction in ASR on `text-davinci-003` and `gpt-3.5-turbo`, respectively. We also provide the average accuracy in Table 2. We observe a slight improvement by using code-style instructions when prompting LLMs with clean samples(0.47 and 0.52), but a relatively huge improvement with adversarial samples(3.75 and 4.28), **which indicates the advantages of using code-style prompts when faced with adversarial samples**. A more detailed analysis of the advantages of code-style instructions is provided in Section 5.

Adversarial context further enhances the robustness We further demonstrate the effectiveness of our proposed *adversarial context method*. Compared to natural language prompting, the adversarial context method leads to a decrease of 3.55 points in ASR for `text-davinci-003` and 5.66 points for `gpt-3.5-turbo`. Besides, from Tabel 2, incorporating our adversarial context method results in a significant improvement in accuracy. Specifically, there is an improvement of 5.13 points with `text-davinci-003` and 5.68 points with `gpt-3.5-turbo`. We hypothesize that the improvement brought by adversarial samples could be attributed to the implicit adversarial training through in-context learning. Additionally, introducing adversarial samples prompts the LLMs to recognize specific adversarial attacks, such as spelling errors and word substitutions. The findings also suggest that more advanced models, like `gpt-3.5-turbo`, potentially benefit more from code-style instructions and the adversarial context method than `text-davinci-003`.

Zero-Shot vs. Few-shot As shown in Table 1, zero-shot prompting exhibits low robustness on both `text-davinci-003` and `gpt-3.5-turbo`. In particular, for some tasks, the LLMs perform only slightly better or even worse than random guessing (for example, QQP on `text-davinci-003`). However, when prompting LLMs with additional in-context demonstrations, the robustness of LLMs improves by a large margin. By few-shot prompting, the average ASR of `text-davinci-003` and `gpt-3.5-turbo` decrease by 16.82 and 13.75 points, respectively. This indicates the strong few-shot learning abilities of LLMs. **By leveraging only a few examples, LLMs can better understand the task and yield stronger robustness**.

Chain-of-thought helps with robustness We then explore whether CoT can help improve the robustness of LLMs. **By promoting LLMs to**

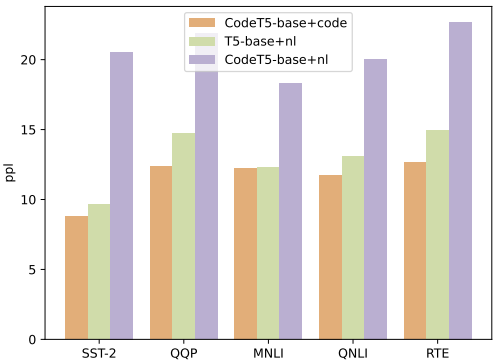


Figure 3: Perplexity for AdvGLUE dataset on T5-base with natural language instructions and CodeT5-base with both natural language and code-style instructions. We report the logarithm of their initial values.

think step-by-step to generate final answers, we find that for most tasks, LLMs showcase better robustness than directly prompting LLMs to generate the final answer. On average, we present a decline in ASR by 0.97 and 1.02 points with `text-davinci-003` and `gpt-3.5-turbo`, respectively. However, we also observe a decrease in robustness on specific datasets, such as RevTgt on `gpt-3.5-turbo`. By analyzing the reasoning steps, we find that the over-complicated and neutral-oriented reasoning process contributes to the failure of CoT.

5. Analysis

5.1. Perplexity: Code vs NL

To take a closer look at the advantages of using code-style instructions, we hypothesize that utilizing code-style instructions can provide LLMs pre-trained on code data with more precise instructions, consequently resulting in performance improvement. To verify our hypothesis, we compare the perplexity of a pre-trained language model on the natural language instructions and a pre-trained code model on both the natural language and code-style instructions. Specifically, we calculate the mean perplexity *ppl* of a dataset T consisting of N samples using the following formula:

$$ppl_M(T) = \frac{1}{n} \sum_{(x,y)\in T} \prod_{i=1}^{m} P_M(y_i|y_1 \dots y_{i-1}, x)^{-\frac{1}{m}} \tag{3}$$

where m refers to the length of the generated tokens. For each sample (x, y) in T, we convert it to both natural language format (x_{nl}, y_{nl}) and code-style format (x_c, y_c) and then calculate the perplexity with two models M_{nl} and M_c. A lower perplexity suggests the models are less confused by the instructions and output format.

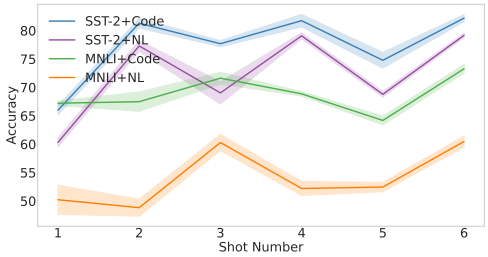


Figure 4: Accuracy with the different number of in-context demonstrations on SST-2 and MNLI adversarial dataset. The experiment is conducted on gpt-3.5-turbo.

Due to the *black-box* features of LLMs, obtaining logits directly from LLMs is challenging. Therefore, following (Li et al., 2023a), we use T5 (Raffel et al., 2019a) and CodeT5 (Wang et al., 2021b), which are further pre-trained on code data for T5, to compute perplexities. We use the AdvGLUE dataset to calculate the perplexity of T5-base with natural language instructions and CodeT5-base with both natural language and code-style instructions. As shown in Figure 3, utilizing the pre-trained code model with code-style instructions consistently results in the lowest perplexity, surpassing the performance of using natural language instructions in both the pre-trained language and code models. **This observation suggests that converting instructions into code style better align with the pretraining data distribution for pre-trained code models**.

Prompt	SST-2	MNLI	QNLI
NL	19.23	21.73	18.75
NL(complicated)	19.45	21.64	18.97
Class Exec	**17.83**	18.55	14.28
Class Init	20.93	**16.12**	16.52
Func Exec	18.6	17.52	**14.03**

Table 3: ASR for different code-style instructions design. "class exec" is the code format used in our main experiments and is highlighted in light grey. The experiment is conducted on gpt-3.5-turbo.

5.2. Ablation Studies

5.2.1. Different Code-style Instructions

To explore whether using code-style instructions can generally obtain better robustness, following (Wang et al., 2022b) and (Li et al., 2023a), we design two different code-style instructions *class init*

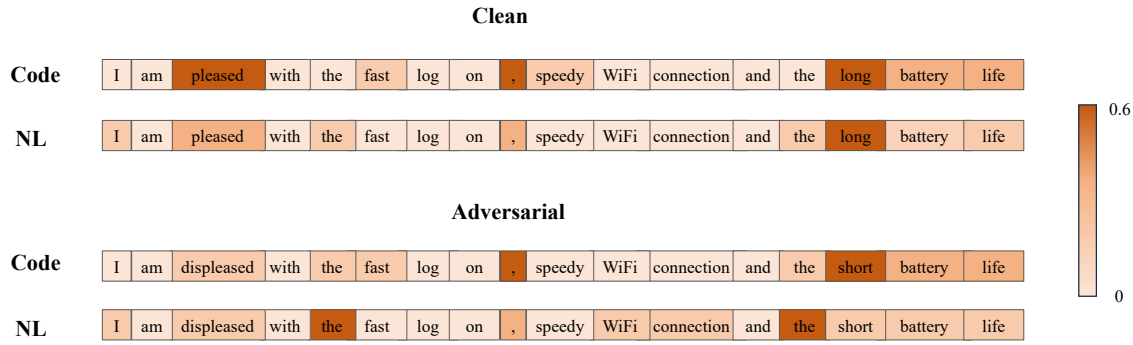


Figure 5: Visualization of a sample's gradient on each word when fine-tuning CodeT5 with code-style instruction and T5 with natural language instruction, respectively. The sample is selected from the Restaurant-T dataset with both its clean and adversarial versions. The sample aims to determine the sentiment polarity of the aspect "*battery life*" in the sentence with "*positive*" or "*negative*".

and *func exec*. The *class init* provides LLMs with incomplete code with partial parameter input of the class as triggers to prompt LLMs to complete the code. The *func exec* converts the class definition into function definition. We also design a more complicated NL prompt for comparison. We report the results in Table 3, from which we find that code-style instructions almost always outperform natural language instructions with lower ASR, showcasing the overall superiority of code-style instructions. Besides, using more complicated natural language instructions does not help with the robustness of LLMs.

5.2.2. Number of In-context Demonstrations

To investigate the influence of different numbers of demonstrations, we perform experiments on SST-2 and MNLI, ranging from 1 to 6 shots. As shown in Figure 4, we find that the completeness and balance of labels are significant for the task performance. For example, for SST-2 with two labels *positive* and *negative*, when prompting with an even number of demonstrations (*positive* and *negative* are both included and the number of each label is equal), we consistently get better results than those with incomplete labels (1-shot) or with imbalanced labels (3-shot, 5-shot). Therefore in Section 4.2, we choose shot number k according to the number of labels for different tasks.

5.2.3. Different Part of Code Instructions

To assess the influence of different parts of our code-style prompts, we conduct three transformations on the instructions. As shown in Table 4, randomly replacing the *Class Name* and *Sub-task Name* has minimal impact on performance while removing the annotation leads to a slight decline in performance. This shows the toughness of our code-style instructions against different disturbance.

Component	SST-2	RTE
Code Instructions	83.1	74.07
- Class Name	82.43(−0.67)	72.83(−1.24)
- Sub-task Name	81.08(−2.02)	75.31(+1.24)
- Annotation	78.38(−4.72)	69.14(−4.93)

Table 4: Results for different parts of code-style instructions. We report the Acc with gpt-3.5-torbo. We conduct three separate experiments: randomly replacing the *Class Name*, randomly replacing the *Sub-task Name* and removing the *Annotation*.

5.3. Visualization Analysis

To further investigate which part the model focuses on, we select a sample from the Restaurant-T dataset with both its original and adversarial forms. We then utilize natural language instructions and code-style instructions to wrap the sample and then fine-tuning them using T5-base and CodeT5-base, respectively. We extract the gradients of each token from the model embedding layer and average them across dimensions. Moreover, we normalize the gradients within the sentence and obtain final gradients. We visualize the gradients in Figure 5. Since the gradients can reflect how much the model focuses on the token (Li et al., 2016; Madsen et al., 2022), for clean sentence, both natural language instruction and code-style instruction focus on the right word "long". While for adversarial sentence, code-style instruction with CodeT5-base can still lead the model to pay attention to the phrase "*short battery life*". However, using natural language with T5-base, the model focuses on irrelevant phrases such as "*the*". Therefore, code-style prompts may help the model focus

more on the important part of a sentence.

5.4. Discussion for User-friendliness

Although showing impressive performance with code-style instructions, it may be difficult for non-professional users to transform the prompts into code. Actually, the code-style prompts we design are straightforward and can be easily adapted to arbitrary tasks with similar structures. Users can follow our structure either manually or through LLM-based methods (e.g., utilizing LLMs) to construct task-specific code-style prompts. To verify the simplicity of our method, we select several tasks (SST-2, MNLI, RTE) and concatenate them to prompt GPT-4 to generate code-style prompts for new tasks. The results are shown in Table 5. We observe that LLM-based prompts almost match the performance of manually designed prompts.

New Task	LLM-based	Manual
QNLI	$72.07_{\pm 2.1}$	73.64
RevTgt	$70.22_{\pm 2.7}$	71.33

Table 5: Results for LLM(GPT-4)-based prompts with gpt-3.5-turbo. We report the Acc of new tasks QNLI and RevTgt. "LLM-based" refers to GPT-4 generated prompts. "Manual" refers to our manually designed prompts.

6. Related Work

Textual Adversarial Attacks/Defenses Textual attacks typically generate explicit adversarial examples by adding small perturbations into clean examples while maintaining lexical correctness, grammatical correctness and semantic similarity (Ren et al., 2019b; Wang et al., 2021a). These adversarial methods can be divided into character-level (Gao et al., 2018), word-level (Ren et al., 2019a) and sentence-level (Li et al., 2019). In response to adversarial attacks, various defense methods have been proposed(Jiang et al., 2020; Wang et al., 2022a). Adversarial training (Zhu et al., 2019a) is a widely adopted approach that iteratively solves a two-layered min-max optimization problem. Interval bound propagation (Dvijotham et al., 2018) is proposed to find worst-case adversaries. Besides, randomized smoothing (Cohen et al., 2019) and adversarial detection (Alshemali and Kalita, 2019; Mozes et al., 2021) are also popular methods in defending adversarial attacks. However, all these methods require parameter updates and can be unattainable when faced with *closed source black-box* LLMs. Therefore, in this work, we propose a novel approach to enhance the robustness of LLMs through instructions design without the need for parameter updates.

Robustness Concerns for LLMs While the progress of LLMs has shown remarkable abilities in following human instructions and generating safe content, recent works pose concerns about the robustness of LLMs (Liu et al., 2023; Shi et al., 2024)(Wang et al., 2022). Attacks based on prompts have showcased the possibility of attacking LLMs with adversarial prompts (Zhu et al., 2023; Ni et al., 2023). For example, "jailbreak" attempts to modify clean prompts to elicit undesirable responses from LLMs (Wei et al., 2023a). (Zou et al., 2023) find adversarial attacks on aligned language models and prove their universal and transferable attack ability. **In this paper, our attack aims at destroying the samples while keeping the prompts clean, which is different from prompts attacks that focus on destroying prompts**.

In-context Learning With the model scale growing, directly fine-tuning the model can be extremely expensive due to storage and time complexities (Rae et al., 2022; Chowdhery et al., 2022; Smith et al., 2022). Alternatively, in-context learning (ICL) has been verified to be an effective way for LLMs to learn a new task by conditioning on a few training examples (Brown et al., 2020). There are already lots of works demonstrating the perfect ICL abilities of LLMs in solving complex tasks, such as solving mathematical reasoning problems (Wei et al., 2023b). On the other hand, plenty of studies have investigated the mechanism behind ICL. (Xie et al., 2022) explained ICL from the perspective of implicit Bayesian inference. (Dai et al., 2023; von Oswald et al., 2022) viewed ICL as implicit fine-tuning and theoretically demonstrated that Transformer attention has a dual form of gradient descent. Therefore, in this work, we hypothesize that incorporating adversarial samples as demonstrations can be viewed as a form of adversarial training and propose the *adversarial context method*.

Code-style Instructions for Different Tasks Due to the advantages of code format, plenty of works have used code-style instructions to tackle complex tasks. (Gao et al., 2023) used programs to split the decomposition and computation of a mathematical problem. (Wang et al., 2022b) leverage LLMs text-to-structure translation capability to solve structured prediction tasks. (Madaan et al., 2022) frame structured commonsense reasoning tasks as code generation tasks. (Li et al., 2023a) recast the structured output of IE tasks in the form of code instead of natural language. Besides, similar to our work, (Mishra et al., 2023) utilizes pseudo-code instructions to prompt pre-trained models such as CodeGen (Nijkamp et al., 2023) to improve the performance of pre-trained language models.

7. Conclusion

In this paper, we propose **RoCoIns** to utilize code-style instructions instead of natural language instructions to enhance the robustness of *closed source black-box* models against textual adversarial attacks. Instructions in code style, which are more structural and less ambiguous than natural language instructions, provide LLMs with more precise instructions. Besides, we propose *adversarial context method* to further boost the robustness. Experiments show that our method consistently outperforms prompting LLMs with natural language instructions under the few-shot setting. We conduct further analysis to verify the advantages of using code-style instructions.

8. Limitations

Due to the limitation of *closed source black-box* models, we cannot dig into the LLMs to explore the reason for the effectiveness of using code-style instructions. Furthermore, while we have investigated various designs for code-style instructions, there is still a need for further exploration of better prompt design. Besides, querying the GPT-series models can lead to economic expenses and cause environmental pollution.

9. Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by National Natural Science Foundation of China (No. 62206057, 61976056, 62076069, 61906176), Shanghai Rising-Star Program (23QA1400200), Natural Science Foundation of Shanghai (23ZR1403500), Program of Shanghai Academic Research Leader under grant 22XD1401100, CCF-Baidu Open Fund, and CCF-Baichuan Fund. The computations in this research were performed using the CFFF platform of Fudan University.

10. Bibliographical References

Basemah Alshemali and Jugal Kalita. 2019. Toward mitigating adversarial texts. *International Journal of Computer Applications*, 178(50):1–7.

Simran Arora, Avanika Narayan, Mayee F. Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2022. Ask me anything: A simple strategy for prompting language models.

Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. 2021. Deep learning for ai. *Communications of the ACM*, page 58–65.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

BSI. 1973a. *Natural Fibre Twines*, 3rd edition. British Standards Institution, London. BS 2570.

BSI. 1973b. Natural fibre twines. BS 2570, British Standards Institution, London. 3rd. edn.

A. Castor and L. E. Pollux. 1992. The use of user modelling to guide inference and learning. *Applied Intelligence*, 2(1):37–53.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Xuanting Chen, Junjie Ye, Can Zu, Nuo Xu, Rui Zheng, Minlong Peng, Jie Zhou, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. How robust is gpt-3.5 to predecessors? a comprehensive study on language understanding tasks.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages.

J.L. Chercheur. 1994. *Case-Based Reasoning*, 2nd edition. Morgan Kaufman Publishers, San Mateo, CA.

Jonathan H Choi, Kristin E Hickman, Amy Monahan, and Daniel Schwarcz. 2023. Chatgpt goes to law school. *Available at SSRN*.

N. Chomsky. 1973. Conditions on transformations. In *A festschrift for Morris Halle*, New York. Holt, Rinehart & Winston.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, and Sebastian Gehrmann. 2022. Palm: Scaling language modeling with pathways.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Jeremy Cohen, Elan Rosenfeld, and J.Zico Kolter. 2019. Certified adversarial robustness via randomized smoothing.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers.

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. 2018. Training verified learners with learned verifiers.

Umberto Eco. 1990. *The Limits of Interpretation*. Indian University Press.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3816–3830, Online. Association for Computational Linguistics.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. Badnets: Identifying vulnerabilities in the machine learning model supply chain.

Dominik Hintersdorf, Lukas Struppek, and Kristian Kersting. 2023. Balancing transparency and risk: The security and privacy risks of open-source machine learning models.

Paul Gerhard Hoel. 1971a. *Elementary Statistics*, 3rd edition. Wiley series in probability and mathematical statistics. Wiley, New York, Chichester. ISBN 0 471 40300.

Paul Gerhard Hoel. 1971b. *Elementary Statistics*, 3rd edition, Wiley series in probability and mathematical statistics, pages 19–33. Wiley, New York, Chichester. ISBN 0 471 40300.

Ari Holtzman, Peter West, Vered Shwartz, Yejin Choi, and Luke Zettlemoyer. 2022. Surface form competition: Why the highest probability answer isn't always right.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. 2019. Adversarial examples are not bugs, they are features.

Otto Jespersen. 1922. *Language: Its Nature, Development, and Origin*. Allen and Unwin.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. Textbugger: Generating adversarial text against real-world applications. In *Proceedings 2019 Network and Distributed System Security Symposium*.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and understanding neural models in nlp.

Peng Li, Tianxiang Sun, Qiong Tang, Hang Yan, Yuanbin Wu, Xuanjing Huang, and Xipeng Qiu. 2023a. Codeie: Large code generation models are better few-shot information extractors.

Yingcong Li, M. Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. 2023b. Transformers as algorithms: Generalization and stability in in-context learning.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods.

Yugeng Liu, Tianshuo Cong, Zhengyu Zhao, Michael Backes, Yun Shen, and Yang Zhang. 2023. Robustness over time: Understanding adversarial examples' effectiveness on longitudinal versions of large language models.

Li Lucy and David Bamman. 2021. Gender and representation bias in GPT-3 generated stories. In *Proceedings of the Third Workshop on Narrative Understanding*, pages 48–55, Virtual. Association for Computational Linguistics.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards deep learning models resistant to adversarial attacks.

Andreas Madsen, Siva Reddy, and Sarath Chandar. 2022. Post-hoc interpretability for neural NLP: A survey. *ACM Computing Surveys*, 55(8):1–42.

Sewon Min, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Noisy channel language model prompting for few-shot text classification.

Mayank Mishra, Prince Kumar, Riyaz Bhat, Rudra Murthy V au2, Danish Contractor, and Srikanth Tamilselvam. 2023. Prompting with pseudo-code instructions.

Maximilian Mozes, Pontus Stenetorp, Bennett Kleinberg, and Lewis D. Griffin. 2021. Frequency-guided word substitutions for detecting textual adversarial examples.

Yuansheng Ni, Sichao Jiang, Xinyu wu, Hui Shen, and Yuli Zhou. 2023. Evaluating the robustness to instructions of large language models.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis.

OpenAI. 2022. Introducing chatgpt. `https://openai.com/blog/chatgpt`.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022a. Training language models to follow instructions with human feedback.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022b. Training language models to follow instructions with human feedback.

XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, and Susannah Young. 2022. Scaling language models: Methods, analysis & insights from training gopher.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and PeterJ. Liu. 2019a. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv: Learning*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and PeterJ. Liu. 2019b. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv: Learning,arXiv: Learning*.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019a. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy. Association for Computational Linguistics.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019b. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Joshua Robinson, Christopher Michael Rytting, and David Wingate. 2023. Leveraging large

language models for multiple choice question answering.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Tali Bers, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, and Matthias Gallé. 2023. Bloom: A 176b-parameter open-access multilingual language model.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2023a. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models.

Yiqiu Shen, Laura Heacock, Jonathan Elias, Keith D Hentel, Beatriu Reig, George Shih, and Linda Moy. 2023b. Chatgpt and other large language models are double-edged swords.

Chenyu Shi, Xiao Wang, Qiming Ge, Songyang Gao, Xianjun Yang, Tao Gui, Qi Zhang, Xuanjing Huang, Xun Zhao, and Dahua Lin. 2024. Navigating the overkill in large language models. *arXiv preprint arXiv:2401.17633*.

Charles Joseph Singer, E. J. Holmyard, and A. R. Hall, editors. 1954–58. *A history of technology*. Oxford University Press, London. 5 vol.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model.

Jannik Strötgen and Michael Gertz. 2012. Temporal tagging on different domains: Challenges, strategies, and gold standards. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, pages 3746–3753, Istanbul, Turkey. European Language Resource Association (ELRA).

S. Superman, B. Batman, C. Catwoman, and S. Spiderman. 2000. *Superheroes experiences with books*, 20th edition. The Phantom Editors Associates, Gotham City.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2022. Transformers learn in-context by gradient descent.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Jindong Wang, Xixu Hu, Wenxin Hou, Hao Chen, Runkai Zheng, Yidong Wang, Linyi Yang, Haojun Huang, Wei Ye, Xiubo Geng, Binxin Jiao, Yue Zhang, and Xing Xie. 2023a. On the robustness of chatgpt: An adversarial and out-of-distribution perspective.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. 2023b. Orthogonal subspace learning for language model continual learning. *arXiv preprint arXiv:2310.14152*.

Xiao Wang, Shihan Dou, Limao Xiong, Yicheng Zou, Qi Zhang, Tao Gui, Liang Qiao, Zhanzhan Cheng, and Xuanjing Huang. 2022a. Miner: Improving out-of-vocabulary named entity recognition from an information theoretic perspective. *arXiv preprint arXiv:2204.04391*.

Xiao Wang, Qin Liu, Tao Gui, Qi Zhang, Yicheng Zou, Xin Zhou, Jiacheng Ye, Yongxin Zhang, Rui Zheng, Zexiong Pang, et al. 2021a. Textflint: Unified multilingual robustness evaluation toolkit for natural language processing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 347–355.

Xiao Wang, Weikang Zhou, Can Zu, Han Xia, Tianze Chen, Yuansen Zhang, Rui Zheng, Junjie Ye, Qi Zhang, Tao Gui, et al. 2023c. Instructuie: Multi-task instruction tuning for unified information extraction. *arXiv preprint arXiv:2304.08085*.

Xingyao Wang, Sha Li, and Heng Ji. 2022b. Code4struct: Code generation for few-shot structured prediction from natural language.

Yue Wang, Weishi Wang, Shafiq Joty, and StevenC.H. Hoi. 2021b. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *Cornell University - arXiv*.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2023a. Jailbroken: How does llm safety training fail?

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023b. Chain-of-thought prompting elicits reasoning in large language models.

Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2022. An explanation of in-context learning as implicit bayesian inference.

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Li Zhang, Liam Dugan, Hainiu Xu, and Chris Callison-Burch. 2023. Exploring the curious case of code prompts.

Shuai Zhao, Zhuoqian Liang, Jinming Wen, and Jie Chen. 2022. Sparsing and smoothing for the seq2seq models. *IEEE Transactions on Artificial Intelligence*, pages 1–10.

Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2019a. Freelb: Enhanced adversarial training for natural language understanding. *Cornell University - arXiv*.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2019b. Freelb: Enhanced adversarial training for natural language understanding.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Neil Zhenqiang Gong, Yue Zhang, and Xing Xie. 2023. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models.

11. Language Resource References

Pontiki, Maria and Galanis, Dimitris and Pavlopoulos, John and Papageorgiou, Harris and Androutsopoulos, Ion and Manandhar, Suresh. 2014. *SemEval-2014 Task 4: Aspect Based Sentiment Analysis*. Association for Computational Linguistics.

Rajpurkar, Pranav and Zhang, Jian and Lopyrev, Konstantin and Liang, Percy. 2016. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. Association for Computational Linguistics.

Socher, Richard and Perelygin, Alex and Wu, Jean and Chuang, Jason and Manning, Christopher D. and Ng, Andrew and Potts, Christopher. 2013. *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*. Association for Computational Linguistics.

Boxin Wang and Chejian Xu and Shuohang Wang and Zhe Gan and Yu Cheng and Jianfeng Gao and Ahmed Hassan Awadallah and Bo Li. 2022. *Adversarial GLUE: A Multi-Task Benchmark for Robustness Evaluation of Language Models*. Conference on Neural Information Processing Systems.

Williams, Adina and Nangia, Nikita and Bowman, Samuel. 2018. *A Broad-Coverage Challenge*

Corpus for Sentence Understanding through Inference. Association for Computational Linguistics.

Xing, Xiaoyu and Jin, Zhijing and Jin, Di and Wang, Bingning and Zhang, Qi and Huang, Xuanjing. 2020. *Tasty Burgers, Soggy Fries: Probing Aspect Robustness in Aspect-Based Sentiment Analysis*. Association for Computational Linguistics.

Appendices

A. Datasets

The statistics of the datasets in our paper have been presented in Table 6.

A.1. AdvGLUE

AdvGLUE (Wang et al., 2022) is a multi-task benchmark to evaluate modern language models. The benchmark contains SST-2, QQP, MNLI, QNLI and RTE. It incorporates diverse forms of attacks at the word level, sentence level and also contains human-written samples. SST-2 (Socher et al., 2013) consists of movie reviews with human-annotated sentiments. The task is to predict the sentiment of given sentences. QQP is a collection of question pairs from the community question-answering website Quora. The task is to determine whether the given two questions are semantics equivalent. MNLI (Williams et al., 2018) consists of a set of sentence pairs accompanied by annotations indicating textual entailment. The objective of this task is to determine whether a given premise sentence implies the hypothesis (*entailment*), contradicts it (*contradiction*), or has no clear relationship with it (*neutral*). QNLI (Rajpurkar et al., 2016) is a question-answering dataset consisting of question-paragraph pairs. The goal is to determine whether the paragraph contains the answer to the question. RTE datasets come from a series of annual textual entailment challenges. The goal is to judge the relationships between two sentences, which include *entailment* and *not entailment*

A.2. Restaurant

The Restaurant dataset is an Aspect-based sentiment analysis sourced from SemEvall 2014 dataset (Pontiki et al., 2014) and in this work, we use its adversarial version from (Xing et al., 2020). The adversarial transformation contains three parts: RevTgt, RevNon and AddDiff. RevTgt is to generate sentences that reverse the original sentiment of the target aspect. RevNon aims to perturb the sentiments of the non-target aspects. Specifically, for all the non-target aspects with the same sentiment as the target aspects, we reverse their sentiments. AddDiff further investigates if adding more non-target aspects can confuse the model. We add extra aspects that possess sentiments opposite to the target aspect. In this work, we random select 300 samples from each transformation to conduct our experiments.

B. Prompt Design

B.1. Natural Language Prompts

The natural language prompts of AdvGLUE are the same as (Wang et al., 2023a). We present all the natural language prompts in Table 7.

B.2. Code-style Prompts

All prompts for the tasks in our paper will be presented in this section.

Prompts for SST-2:

```
class Sentiment_Classification:
    """
    Base class for judging whether the sentiment
        of the given sentence is "positive" or
        "negative".

    Parameters
    ----------
    input_text : str
        The input sentence.

    """
    def __init__(self, input_text):
        self.input_text = input_text

    def sentiment_classification(self):
        polarity =
            self.input_text.sentiment.polarity

        if polarity > 0:
            print('positive')
        elif polarity < 0:
            print('negative')
```

Prompts for QQP:

```
class Semantics_Consistent_Judgement:
    """
    Base class for judging whether the semantics
        of the two sentences are consistent.

    Parameters
    ----------
    input_text1 : str
        The first input sentence.
    input_text2 : str
        The second input sentence.
    """
    def __init__(self, input_text1, input_text2):
```

Dataset	Task	Sample	Class
SST2	sentiment classification	148	2
QQP	quora question pairs	78	2
MNLI	multi-genre natural language inference	121	3
QNLI	question-answering NLI	148	2
RTE	textual entailment recognition	81	2
Restaurant-T	aspect-based sentiment analysis	900	3

Table 6: Statistics of test sets in this paper. For the Restaurant-T dataset, we randomly select 300 samples from each transformation (RevNon, RevTgt and AddDiff) and lead to a total of 900 samples.

Dataset	Prompt
SST-2	Please classify the following sentence into either positive or negative. Answer me with "positive" or "negative", just one word.
QQP	Are the following two questions equivalent or not? Answer me with "equivalent" or "not_equivalent".
MNLI	Are the following two sentences entailment, neutral or contradiction? Answer me with "entailment", "neutral" or "contradiction".
QNLI	Are the following question and sentence entailment or not_entailment? Answer me with "entailment" or "not_entailment".
RTE	Are the following two sentences entailment or not_entailment? Answer me with "entailment" or "not_entailment".
Restaurant-T	What is the sentiment towards 'sentence' in terms of 'aspect word'? Are they viewed positively, negatively, or neutrally?

Table 7: natural language prompts

```
        self.input_text1 = input_text1
        self.input_text2 = input_text2

    def semantics_similarity(self):

        similarity =
            cosine_similarity(self.input_text1,
            self.input_text2)

        if similarity > 0:
            print("equivalent")
        elif similarity < 0:
            print("not_equivalent")
```

Prompts for MNLI:

```
class Entailment_Judgement:
    """
    Base class for judging whether the premise
        and the hypothesis are "entailment",
        "neutral" or "contradiction".

    Parameters
    ----------
    premise : str
        The input premise.
    hypothesis : str
        The input hypothesis.
    """
    def __init__(self, premise: str, hypothesis:
        str):
        self.premise = premise
        self.hypothesis = hypothesis

    def determine_relationship(self):
        if is_entailment(self.premise,
            self.hypothesis):
            print("entailment")
        elif is_contradiction(self.premise,
            self.hypothesis):
            print("contradiction")
        else:
            print("neutral")
```

Prompts for QNLI:

```
class Answer_Verification:
    """
    Given a question, determines whether the
        provided text contains the correct
        answer to the question.
    The relationship consists of "entailment"
        and "not entailment".

    Parameters
    ----------
    question : str
        The input question.
    text : str
        The input text.
    """
    def __init__(self, question, text):
        self.question = question
        self.text = text
```

```
    def determine_relationship(self):
        if is_entailment(self.question,
             self.text):
            print("entailment")
        else:
            print("not_entailment")
```

Prompts for RTE:

```
class Entailment_Judgement:
    """
    Base class for judging whether the two
         sentences are "entailment" or
         "not_entailment".

    Parameters
    ----------
    sentence1 : str
        The first input sentence.
    sentence2 : str
        The second input sentence.
    """
    def __init__(self, premise: str, hypothesis:
         str, relationship: str):
        self.sentence1 = sentence1
        self.sentence2 = sentence2

    def determine_relationship(self):
        if is_entailment(self.sentence1,
             self.sentence2):
            print("entailment")
        else:
            print("not_entailment")
```

Prompts for Restaurant-T:

```
class Aspect_Based_Sentiment_Analysis:
    """
    Base class for aspect-based sentiment
         analysis task.

    Parameters
    ----------
    aspect : str
        The target aspect term of the given
             sentence.
    sentence : str
        The input text that contains the aspect.
    """
    def __init__(self, sentence: str, aspect:
         str):
        self.sentence = sentence
        self.aspect = aspect
    def sentiment_classification(self):
    if is_positive(self.sentence, self.aspect):
        print("positive")
    elif is_negative(self.sentence, self.aspect):
        print("negative")
```

B.3. Other Code-style prompts Design

In Section 5.2.2, we design two other different code-style prompts: *class init* and *func exec*. Specifically, the "class init" prompt provides LLMs with incomplete code with partial parameter input of the class as triggers to prompt LLMs to complete the code. The "func exec" converts the class definition into a function definition. The detailed results of these two designs can be found in Table 2. Besides, following your advice, we will also add other ablation experiments with regard to the prompt design in the subsequent version of our paper, such as the influence of certain parts of the prompt (Class name, annotation etc.) We use the QNLI task as an example to present the different prompts.

Prompts for *class init*:

```
class Answer_Verification:
    """
    Given a question, determines whether the
         provided text contains the correct
         answer to the question.
    The relationship consists of "entailment"
         and "not entailment".

    Parameters
    ----------
    question : str
        The input question.
    text : str
        The input text.
    """
    def __init__(self, question, text,
         relationship):
        self.question = question
        self.text = text
        self.relationship = relationship
```

Prompts for *func exec*:

```
def Answer_Verification(question: str, text:
     str):
    """
    Given a question, determines whether the
         provided text contains the correct
         answer to the question.
    The relationship consists of "entailment"
         and "not entailment".

    Args:
        question (str): The input question.
        text (str): The input text.

    Returns:
        str: "entailment", or "not entailment".
    """

    if is_entailment(question, text):
        print("entailment")
    else:
        print("not_entailment")
```

C. Detailed Experiments Results

The detailed experiments of accuracy can be found in this section. Table 8 and Table 10 present the accuracy of datasets before (Original) and after adversarial transformations (Adversarial) with `text-davinci-003`. Moreover, Table 10 and Table 11 present the results of gpt-3.5-turbo. All the results are under few-shot settings. From Tabel 8 and Table 9, we can conclude that using code-style prompts only acquire little improvement on the original datasets. Specifically, using code-style instructions outperforms using natural language instructions by 5.13 and 5.68 points in accuracy with `text-davinci-003` and `gpt-3.5-turbo`, respectively. However, when employing adversarial samples in the instructions, using code-style instructions acquires a relatively larger improvement. Specifically, we get 3.02 and 5.68 points with `text-davinci-003` and `gpt-3.5-turbo`, respectively, showcasing the advantages of code-style instructions in resisting adversarial attacks.

Method	Dataset								AVG
	AdvGLUE					Restaurant-T			
	SST-2	QQP	MNLI	QNLI	RTE	RevTgt	RevNon	AddDiff	
NL	96.18	82.05	79.33	79.72	92.59	91.66	92.67	80	86.78
Code	98.47	83.33	84.2	77.7	92.59	91	92	78.67	87.25

Table 8: Original datasets results with text-davinci-003

Method	Dataset								AVG
	AdvGLUE					Restaurant-T			
	SST-2	QQP	MNLI	QNLI	RTE	RevTgt	RevNon	AddDiff	
NL	70.94	67.95	61.15	63.51	82.72	67	**86**	76	72.07
CoT	74.32	71.79	70.25	**67.57**	81.48	70.66	82	69.33	73.43
Code	75	73.07	68.59	66.21	**87.65**	70.67	**88**	77.33	75.82
Code+adv	**78.38**	**75.64**	**72.72**	64.86	**87.65**	**72.66**	87.33	**78.33**	**77.20**

Table 9: Adversarial datasets results of text-davinci-003

Method	Dataset								AVG
	AdvGLUE					Restaurant-T			
	SST-2	QQP	MNLI	QNLI	RTE	RevTgt	RevNon	AddDiff	
NL	99.23	83.33	76.03	75.67	83.95	91.33	92	83.33	85.61
Code	98.47	83.33	80.16	75.67	86.42	91.33	92	81.67	86.13

Table 10: Original datasets results of gpt-3.5-turbo

Method	Dataset								AVG
	AdvGLUE					Restaurant-T			
	SST-2	QQP	MNLI	QNLI	RTE	RevTgt	RevNon	AddDiff	
NL	79.05	76.92	60.33	62.83	67.9	67.3	77.3	72	70.45
CoT	78.37	73.07	**74.38**	64.18	69.13	62	79	69	71.14
Code	83.1	74.35	69.42	73.64	**74.07**	71.33	79.6	**72.33**	74.73
Code+adv	**83.78**	**82.05**	72.73	**74.32**	72.83	**72.33**	**80**	71	**76.13**

Table 11: Adversarial datasets results of gpt-3.5-turbo